# LinuxCNC V2.7.13, 2018-05-09

# Contents

I Contents 1

II Using LinuxCNC 3

1 General Info 4

1.1 User Foreword . . . 4
1.2 LinuxCNC User Introduction . . . 5
1.2.1 How LinuxCNC Works . . . 5
1.2.2 Graphical User Interfaces . . . 6
1.2.2.1 Additional Features . . . 12
1.2.3 Virtual Control Panels . . . 12
1.2.4 Languages . . . 14
1.2.5 Thinking Like a Machine Operator . . . 14
1.2.6 Modes of Operation . . . 14
1.3 Important User Concepts . . . 15
1.3.1 Trajectory Control . . . 15
1.3.1.1 Trajectory Planning . . . 15
1.3.1.2 Path Following . . . 15
1.3.1.3 Programming the Planner . . . 15
1.3.1.4 Planning Moves . . . 16
1.3.2 G Code . . . 17
1.3.2.1 Defaults . . . 17
1.3.2.2 Feed Rate . . . 17
1.3.2.3 Tool Radius Offset . . . 17
1.3.3 Homing . . . 17
1.3.4 Tool Changes . . . 17
1.3.5 Coordinate Systems . . . 18
1.3.5.1 G53 Machine Coordinate . . . 18
1.3.5.2 G54-59.3 User Coordinates . . . 18

1.3.5.3 When You Are Lost . . . 18
1.3.6 Machine Configurations . . . 18
1.4 CNC Machine Overview . . . 20
1.4.1 Mechanical Components . . . 20
1.4.1.1 Axes . . . 20
1.4.1.2 Spindle . . . 21
1.4.1.3 Coolant . . . 21
1.4.1.4 Feed and Speed Override . . . 21
1.4.1.5 Block Delete Switch . . . 21
1.4.1.6 Optional Program Stop Switch . . . 21
1.4.2 Control and Data Components . . . 21
1.4.2.1 Linear Axes . . . 21
1.4.2.2 Rotational Axes . . . 21
1.4.2.3 Controlled Point . . . 22
1.4.2.4 Coordinated Linear Motion . . . 22
1.4.2.5 Feed Rate . . . 22
1.4.2.6 Coolant . . . 22
1.4.2.7 Dwell . . . 22
1.4.2.8 Units . . . 22
1.4.2.9 Current Position . . . 23
1.4.2.10 Selected Plane . . . 23
1.4.2.11 Tool Carousel . . . 23
1.4.2.12 Tool Change . . . 23
1.4.2.13 Pallet Shuttle . . . 23
1.4.2.14 Path Control Mode . . . 23
1.4.3 Interpreter Interaction with Switches . . . 23
1.4.3.1 Feed and Speed Override Switches . . . 23
1.4.3.2 Block Delete Switch . . . 23
1.4.3.3 Optional Program Stop Switch . . . 24
1.4.4 Tool Table . . . 24
1.4.5 Parameters . . . 24
1.5 Running LinuxCNC . . . 25
1.5.1 Invoking LinuxCNC . . . 25
1.5.2 Configuration Launcher . . . 25
1.5.3 Next steps in configuration . . . 27
1.5.4 Simulator Configurations . . . 27
1.5.5 Configuration Resources . . . 28
1.6 Stepper Configuration Wizard . . . 28
1.6.1 Introduction . . . 28

1.6.2 Start Page . . . 29
1.6.3 Basic Information . . . 30
1.6.4 Latency Test . . . 31
1.6.5 Parallel Port Setup . . . 33
1.6.6 Parallel Port 2 Setup . . . 34
1.6.7 Axis Configuration . . . 35
1.6.7.1 Test This Axis . . . 36
1.6.8 Spindle Configuration . . . 38
1.6.8.1 Spindle Speed Control . . . 38
1.6.8.2 Spindle-synchronized motion . . . 39
1.6.8.3 Determining Spindle Calibration . . . 39
1.6.9 Options . . . 40
1.6.10 Machine Configuration Complete . . . 40
1.6.11 Axis Travel and Home . . . 41
1.6.11.1 Operating without Limit Switches . . . 41
1.6.11.2 Operating without Home Switches . . . 41
1.6.11.3 Home and Limit Switch wiring options . . . 41
1.7 Mesa Configuration Wizard . . . 42
1.7.1 Step by Step Instructions . . . 44
1.7.2 Create or Edit . . . 44
1.7.3 Basic Machine Information . . . 45
1.7.4 External Configuration . . . 47
1.7.5 GUI Configuration . . . 49
1.7.6 Mesa Configuration . . . 52
1.7.7 Mesa I/O Setup . . . 53
1.7.8 Parport configuration . . . 57
1.7.9 Axis Configuration . . . 58
1.7.10 Spindle Configuration . . . 65
1.7.11 Advanced Options . . . 67
1.7.12 HAL Components . . . 68
1.7.13 Advanced Usage Of PNCconf . . . 69
1.8 Linux FAQ . . . 70
1.8.1 Automatic Login . . . 70
1.8.2 Automatic Startup . . . 70
1.8.3 Terminal . . . 70
1.8.4 Man Pages . . . 70
1.8.5 List Modules . . . 71
1.8.6 Editing a Root File . . . 71
1.8.6.1 The Command Line Way . . . 71

1.8.6.2 The GUI Way . . . 71
1.8.6.3 Root Access . . . 71
1.8.7 Terminal Commands . . . 71
1.8.7.1 Working Directory . . . 71
1.8.7.2 Changing Directories . . . 72
1.8.7.3 Listing files in a directory . . . 72
1.8.7.4 Finding a File . . . 72
1.8.7.5 Searching for Text . . . 72
1.8.7.6 Diagnostic Messages . . . 73
1.8.8 Convenience Items . . . 73
1.8.8.1 Terminal Launcher . . . 73
1.8.9 Hardware Problems . . . 73
1.8.9.1 Hardware Info . . . 73
1.8.9.2 Monitor Resolution . . . 73
1.8.10 Paths . . . 73
1.9 Lathe User Information . . . 74
1.9.1 Lathe Mode . . . 74
1.9.2 Lathe Tool Table . . . 74
1.9.3 Lathe Tool Orientation . . . 74
1.9.4 Tool Touch Off . . . 76
1.9.4.1 X Touch Off . . . 77
1.9.4.2 Z Touch Off . . . 77
1.9.4.3 The Z Machine Offset . . . 78
1.9.5 Spindle Synchronized Motion . . . 78
1.9.6 Arcs . . . 78
1.9.6.1 Arcs and Lathe Design . . . 78
1.9.6.2 Radius & Diameter Mode . . . 78
1.9.7 Tool Path . . . 79
1.9.7.1 Control Point . . . 79
1.9.7.2 Cutting Angles without Cutter Comp . . . 79
1.9.7.3 Cutting a Radius . . . 81
1.9.7.4 Using Cutter Comp . . . 83

**2 User Interfaces** **84**
2.1 AXIS GUI . . . 84
2.1.1 Introduction . . . 84
2.1.2 Getting Started . . . 85
2.1.2.1 A Typical Session . . . 85
2.1.3 AXIS Display . . . 86

2.1.3.1 Menu Items . . . 86
2.1.3.2 Toolbar buttons . . . 89
2.1.3.3 Graphical Display Area . . . 90
2.1.3.4 Text Display Area . . . 92
2.1.3.5 Manual Control . . . 92
2.1.3.6 MDI . . . 94
2.1.3.7 Feed Override . . . 94
2.1.3.8 Spindle Speed Override . . . 94
2.1.3.9 Jog Speed . . . 95
2.1.3.10 Max Velocity . . . 95
2.1.4 Keyboard Controls . . . 95
2.1.5 Show LinuxCNC Status (linuxcnctop) . . . 96
2.1.6 MDI interface . . . 96
2.1.7 axis-remote . . . 97
2.1.8 Manual Tool Change . . . 97
2.1.9 Python modules . . . 97
2.1.10 Using AXIS in Lathe Mode . . . 98
2.1.11 Advanced Configuration . . . 98
2.1.11.1 Program Filters . . . 99
2.1.11.2 The X Resource Database . . . 100
2.1.11.3 ~/.axisrc . . . 100
2.1.11.4 USER_COMMAND_FILE . . . 100
2.1.11.5 user_live_update() . . . 100
2.1.11.6 External Editor . . . 100
2.1.11.7 Virtual Control Panel . . . 101
2.1.11.8 Preview Control . . . 101
2.1.11.9 Axisui Pins . . . 101
2.2 GMOCCAPY . . . 102
2.2.1 Introduction . . . 102
2.2.2 Requirements . . . 103
2.2.3 How To Get gmoccapy . . . 103
2.2.4 Basic Configuration . . . 103
2.2.4.1 The DISPLAY Section . . . 104
2.2.4.2 The RS274NGC Section . . . 112
2.2.4.3 The MACRO Section . . . 112
2.2.4.4 The TRAJ Section . . . 115
2.2.5 HAL Pins . . . 115
2.2.5.1 Right And Bottom Button Lists . . . 115
2.2.5.2 Velocities And Overrides . . . 117

2.2.5.3 Jog Hal Pins . . . 118
2.2.5.4 Jog Velocities And Turtle-Jog Hal Pin . . . 118
2.2.5.5 Jog Increment Hal Pins . . . 119
2.2.5.6 Hardware Unlock Pin . . . 119
2.2.5.7 Error Pins . . . 119
2.2.5.8 User Created Message HAL Pins . . . 119
2.2.5.9 Spindle Feedback Pins . . . 120
2.2.5.10 Pins To Indicate Program Progress Information . . . 120
2.2.5.11 Tool related pin . . . 121
2.2.6 Auto Tool Measurement . . . 122
2.2.6.1 Tool Measurement Pins . . . 123
2.2.6.2 Tool Measurement INI File Modifications . . . 123
2.2.6.3 Needed Files . . . 124
2.2.6.4 Needed Hal Connections . . . 124
2.2.7 The Settings Page . . . 124
2.2.7.1 Appearance . . . 125
2.2.7.2 Hardware . . . 130
2.2.7.3 Advanced Settings . . . 132
2.2.8 Lathe Specific Section . . . 134
2.2.9 Plasma Specific Section . . . 136
2.2.10 Video On You Tube . . . 136
2.2.10.1 Basic Usage . . . 136
2.2.10.2 Simulated Jog Wheels . . . 137
2.2.10.3 Settings Page . . . 137
2.2.10.4 Simulated Hardware Button . . . 137
2.2.10.5 User Tabs . . . 137
2.2.10.6 Tool Measurement Video . . . 137
2.2.11 Known problems . . . 137
2.2.11.1 Strange numbers in the info area . . . 137
2.2.11.2 Not ending macro . . . 138
2.3 NGCGUI . . . 139
2.3.1 Overview . . . 139
2.3.2 Demonstration Configurations . . . 140
2.3.3 Library Locations . . . 142
2.3.4 Standalone Usage . . . 143
2.3.4.1 Standalone NGCGUI . . . 143
2.3.4.2 Standalone PYNGCGUI . . . 143
2.3.5 Embedding NGCGUI . . . 144
2.3.5.1 Embedding NGCGUI in Axis . . . 144

2.3.5.2 Embedding PYNGCGUI as a gladevcp tab page in a gui . . . 144
2.3.5.3 Additional INI File items required for ngcgui or pyngcgui . . . 145
2.3.5.4 Truetype Tracer . . . 146
2.3.5.5 INI File Path Specifications . . . 146
2.3.5.6 Summary of INI File item details for NGCGUI usage . . . 147
2.3.6 File Requirements for NGCGUI Compatibility . . . 149
2.3.6.1 Single-File Gcode (.ngc) Subroutine Requirements . . . 149
2.3.6.2 Gcode-meta-compiler (.gcmc) file requirements . . . 151
2.3.7 DB25 Example . . . 153
2.4 Touchy GUI . . . 155
2.4.1 Panel Configuration . . . 156
2.4.1.1 HAL connections . . . 156
2.4.1.2 Recommended for any setup . . . 157
2.4.2 Setup . . . 157
2.4.2.1 Enabling Touchy . . . 157
2.4.2.2 Preferences . . . 157
2.4.2.3 Macros . . . 157
2.5 TkLinuxCNC GUI . . . 158
2.5.1 Introduction . . . 158
2.5.2 Getting Started . . . 159
2.5.2.1 A typical session with TkLinuxCNC . . . 159
2.5.3 Elements of the TkLinuxCNC window . . . 159
2.5.3.1 Main buttons . . . 160
2.5.3.2 Offset display status bar . . . 160
2.5.3.3 Coordinate Display Area . . . 160
2.5.3.4 Automatic control . . . 160
2.5.3.5 Manual Control . . . 161
2.5.3.6 Code Entry . . . 162
2.5.3.7 Jog Speed . . . 162
2.5.3.8 Feed Override . . . 162
2.5.3.9 Spindle speed Override . . . 162
2.5.4 Keyboard Controls . . . 162
2.6 MINI GUI . . . 164
2.6.1 Introduction . . . 164
2.6.2 Screen layout . . . 165
2.6.3 Menu Bar . . . 166
2.6.4 Control Button Bar . . . 167
2.6.4.1 MANUAL . . . 167
2.6.4.2 AUTO . . . 168

2.6.4.3 MDI . . . 169
2.6.4.4 [FEEDHOLD] — [CONTINUE] . . . 169
2.6.4.5 [ABORT] . . . 169
2.6.4.6 [ESTOP] . . . 170
2.6.5 Left Column . . . 170
2.6.5.1 Axis Position Displays . . . 170
2.6.5.2 Feed rate Override . . . 170
2.6.5.3 Messages . . . 171
2.6.6 Right Column . . . 171
2.6.6.1 Program Editor . . . 171
2.6.6.2 Backplot Display . . . 172
2.6.6.3 Tool Page . . . 173
2.6.6.4 Offset Page . . . 173
2.6.7 Keyboard Bindings . . . 174
2.6.7.1 Common Keys . . . 174
2.6.7.2 Manual Mode . . . 175
2.6.7.3 Auto Mode . . . 175
2.6.8 Misc . . . 176
2.7 KEYSTICK GUI . . . 176
2.7.1 Introduction . . . 176
2.7.2 Installing . . . 176
2.7.3 Using . . . 177

**3 Programming** **178**
3.1 Coordinate Systems . . . 178
3.1.1 Introduction . . . 178
3.1.2 Machine Coordinate System . . . 178
3.1.3 Coordinate Systems . . . 179
3.1.3.1 Default Coordinate System . . . 180
3.1.3.2 Setting Coordinate System Offsets . . . 180
3.1.4 Global Offsets . . . 181
3.1.4.1 The G92 Commands . . . 181
3.1.4.2 Setting G92 Values . . . 181
3.1.5 Sample Program Using Offsets . . . 182
3.2 G Code Overview . . . 183
3.2.1 Overview . . . 183
3.2.2 Format of a line . . . 183
3.2.2.1 Block Delete . . . 184
3.2.2.2 Line Number . . . 184

3.2.2.3 Word . . . 184
3.2.2.4 Number . . . 185
3.2.3 Parameters . . . 185
3.2.3.1 Numbered Parameters . . . 186
3.2.3.2 Subroutine Parameters . . . 188
3.2.3.3 Named Parameters . . . 188
3.2.3.4 Predefined Named Parameters . . . 188
3.2.3.5 System Parameters . . . 190
3.2.4 Expressions . . . 191
3.2.5 Binary Operators . . . 191
3.2.6 Equality and floating-point values . . . 192
3.2.7 Functions . . . 192
3.2.8 Repeated Items . . . 193
3.2.9 Item order . . . 193
3.2.10 Commands and Machine Modes . . . 193
3.2.11 Polar Coordinates . . . 194
3.2.12 Modal Groups . . . 195
3.2.13 Comments . . . 196
3.2.14 Messages . . . 197
3.2.15 Probe Logging . . . 197
3.2.16 Logging . . . 197
3.2.17 Debug Messages . . . 197
3.2.18 Print Messages . . . 197
3.2.19 Comment Parameters . . . 198
3.2.20 File Requirements . . . 198
3.2.21 File Size . . . 198
3.2.22 G Code Order of Execution . . . 198
3.2.23 G Code Best Practices . . . 199
3.2.24 Linear and Rotary Axis . . . 200
3.2.25 Common Error Messages . . . 200
3.3 G Codes . . . 200
3.3.1 Conventions . . . 200
3.3.2 G Code Quick Reference Table . . . 200
3.3.3 G0 Rapid Move . . . 201
3.3.3.1 Rapid Velocity Rate . . . 202
3.3.4 G1 Linear Move . . . 202
3.3.5 G2, G3 Arc Move . . . 203
3.3.5.1 Center Format Arcs . . . 203
3.3.5.2 Center Format Examples . . . 205

3.3.5.3 Radius Format Arcs . . . 207
3.3.6 G4 Dwell . . . 207
3.3.7 G5 Cubic Spline . . . 208
3.3.8 G5.1 Quadratic Spline . . . 208
3.3.9 G5.2 G5.3 NURBS Block . . . 209
3.3.10 G7 Lathe Diameter Mode . . . 210
3.3.11 G8 Lathe Radius Mode . . . 210
3.3.12 G10 L1 Set Tool Table . . . 211
3.3.13 G10 L2 Set Coordinate System . . . 211
3.3.14 G10 L10 Set Tool Table . . . 213
3.3.15 G10 L11 Set Tool Table . . . 213
3.3.16 G10 L20 Set Coordinate System . . . 214
3.3.17 G17 - G19.1 Plane Select . . . 214
3.3.18 G20, G21 Units . . . 214
3.3.19 G28, G28.1 Go/Set Predefined Position . . . 215
3.3.20 G30, G30.1 Go/Set Predefined Position . . . 215
3.3.21 G33 Spindle Synchronized Motion . . . 216
3.3.22 G33.1 Rigid Tapping . . . 216
3.3.23 G38.n Straight Probe . . . 217
3.3.24 G40 Compensation Off . . . 218
3.3.25 G41, G42 Cutter Compensation . . . 219
3.3.26 G41.1, G42.1 Dynamic Cutter Compensation . . . 219
3.3.27 G43 Tool Length Offset . . . 220
3.3.28 G43.1: Dynamic Tool Length Offset . . . 220
3.3.29 G43.2: Apply additional Tool Length Offset . . . 221
3.3.30 G49: Cancel Tool Length Compensation . . . 221
3.3.31 G53 Move in Machine Coordinates . . . 221
3.3.32 G54-G59.3 Select Coordinate System . . . 222
3.3.33 G61, G61.1 Exact Path Mode . . . 222
3.3.34 G64 Path Blending . . . 222
3.3.35 G73 Drilling Cycle with Chip Breaking . . . 223
3.3.36 G76 Threading Cycle . . . 224
3.3.37 Canned Cycles . . . 226
3.3.37.1 Common Words . . . 226
3.3.37.2 Sticky Words . . . 226
3.3.37.3 Repeat Cycle . . . 227
3.3.37.4 Retract Mode . . . 227
3.3.37.5 Canned Cycle Errors . . . 227
3.3.37.6 Preliminary and In-Between Motion . . . 227

3.3.37.7 Why use a canned cycle? . . . 228
3.3.38 G80 Cancel Canned Cycle . . . 229
3.3.39 G81 Drilling Cycle . . . 230
3.3.40 G82 Drilling Cycle, Dwell . . . 233
3.3.41 G83 Peck Drilling Cycle . . . 234
3.3.42 G84 Right-Hand Tapping Cycle . . . 234
3.3.43 G85 Boring Cycle, Feed Out . . . 234
3.3.44 G86 Boring Cycle, Spindle Stop, Rapid Move Out . . . 235
3.3.45 G87 Back Boring Cycle . . . 235
3.3.46 G88 Boring Cycle, Spindle Stop, Manual Out . . . 235
3.3.47 G89 Boring Cycle, Dwell, Feed Out . . . 235
3.3.48 G90, G91 Distance Mode . . . 235
3.3.49 G90.1, G91.1 Arc Distance Mode . . . 236
3.3.50 G92 Coordinate System Offset . . . 236
3.3.51 G92.1, G92.2 Reset G92 Offsets . . . 236
3.3.52 G92.3 Restore G92 Offsets . . . 237
3.3.53 G93, G94, G95: Feed Rate Mode . . . 237
3.3.54 G96, G97 Spindle Control Mode . . . 237
3.3.55 G98, G99 Canned Cycle Return Level . . . 238
3.4 M Codes . . . 238
3.4.1 M Code Quick Reference Table . . . 238
3.4.2 M0, M1 Program Pause . . . 238
3.4.3 M2, M30 Program End . . . 239
3.4.4 M60 Pallet Change Pause . . . 239
3.4.5 M3, M4, M5 Spindle Control . . . 240
3.4.6 M6 Tool Change . . . 240
3.4.6.1 Manual Tool Change . . . 240
3.4.6.2 Tool Changer . . . 240
3.4.7 M7, M8, M9 Coolant Control . . . 240
3.4.8 M19 Orient Spindle . . . 241
3.4.9 M48, M49 Speed and Feed Override Control . . . 241
3.4.10 M50 Feed Override Control . . . 241
3.4.11 M51 Spindle Speed Override Control . . . 242
3.4.12 M52 Adaptive Feed Control . . . 242
3.4.13 M53 Feed Stop Control . . . 242
3.4.14 M61 Set Current Tool . . . 242
3.4.15 M62 - M65 Digital Output Control . . . 242
3.4.16 M66 Wait on Input . . . 243
3.4.17 M67 Analog Output,Synchronized . . . 243

3.4.18 M68 Analog Output, Immediate . . . 244
3.4.19 M70 Save Modal State . . . 244
3.4.20 M71 Invalidate Stored Modal State . . . 245
3.4.21 M72 Restore Modal State . . . 245
3.4.22 M73 Save and Autorestore Modal State . . . 246
3.4.22.1 Selectively Restoring Modal State . . . 247
3.4.23 M100 - M199 User Defined Commands . . . 247
3.5 O Codes . . . 248
3.5.1 Subroutines . . . 249
3.5.2 Looping . . . 250
3.5.3 Conditional . . . 251
3.5.4 Repeat . . . 251
3.5.5 Indirection . . . 252
3.5.6 Calling Files . . . 252
3.5.7 Subroutine return values . . . 252
3.6 Other Codes . . . 253
3.6.1 F: Set Feed Rate . . . 253
3.6.2 S: Set Spindle Speed . . . 253
3.6.3 T: Select Tool . . . 253
3.7 G Code Examples . . . 254
3.7.1 Mill Examples . . . 254
3.7.1.1 Helical Hole Milling . . . 254
3.7.1.2 Slotting . . . 254
3.7.1.3 Grid Probe . . . 254
3.7.1.4 Smart Probe . . . 254
3.7.1.5 Tool Length Probe . . . 254
3.7.1.6 Hole Probe . . . 254
3.7.1.7 Cutter Compensation . . . 254
3.7.2 Lathe Examples . . . 255
3.7.2.1 Threading . . . 255
3.8 RS274/NGC Differences . . . 255
3.8.1 Changes from RS274/NGC . . . 255
3.8.2 Additions to RS274/NGC . . . 255
3.9 Image to G Code . . . 256
3.9.1 What is a depth map? . . . 257
3.9.2 Integrating image-to-gcode with the AXIS user interface . . . 257
3.9.3 Using image-to-gcode . . . 257
3.9.4 Option Reference . . . 257
3.9.4.1 Units . . . 257

3.9.4.2 Invert Image . . . 257
3.9.4.3 Normalize Image . . . 257
3.9.4.4 Expand Image Border . . . 257
3.9.4.5 Tolerance (units) . . . 257
3.9.4.6 Pixel Size (units) . . . 258
3.9.4.7 Plunge Feed Rate (units per minute) . . . 258
3.9.4.8 Feed Rate (units per minute) . . . 258
3.9.4.9 Spindle Speed (RPM) . . . 258
3.9.4.10 Scan Pattern . . . 258
3.9.4.11 Scan Direction . . . 258
3.9.4.12 Depth (units) . . . 258
3.9.4.13 Step Over (pixels) . . . 258
3.9.4.14 Tool Diameter . . . 259
3.9.4.15 Safety Height . . . 259
3.9.4.16 Tool Type . . . 259
3.9.4.17 Lace bounding . . . 259
3.9.4.18 Contact angle . . . 259
3.9.4.19 Roughing offset and depth per pass . . . 259

**4 Tool Compensation** **261**
4.1 Tool Compensation . . . 261
4.1.1 Tool Length Offsets . . . 261
4.1.1.1 Touch Off . . . 261
4.1.1.2 Using G10 L1/L10/L11 . . . 262
4.1.2 Tool Table . . . 262
4.1.2.1 Tool Table Format . . . 262
4.1.2.2 Tool Changers . . . 263
4.1.3 Cutter Compensation . . . 264
4.1.3.1 Overview . . . 265
4.1.3.2 Examples . . . 266
4.2 Tool Edit GUI . . . 267
4.2.1 Overview . . . 267
4.2.2 Column Sorting . . . 268
4.2.3 Column Selection . . . 268
4.2.4 Stand Alone Use . . . 269

**III Configuration** 270

**5 General Info** 271
5.1 Integrator Concepts . . . 271
5.1.1 File Locations . . . 271
5.1.1.1 Installed . . . 271
5.1.1.2 Command Line . . . 271
5.1.2 Files . . . 272
5.1.3 Stepper Systems . . . 272
5.1.3.1 Base Period . . . 272
5.1.3.2 Step Timing . . . 273
5.1.4 Servo Systems . . . 273
5.1.4.1 Basic Operation . . . 273
5.1.4.2 Proportional term . . . 274
5.1.4.3 Integral term . . . 274
5.1.4.4 Derivative term . . . 274
5.1.4.5 Loop tuning . . . 275
5.1.4.6 Manual tuning . . . 275
5.1.5 RTAI . . . 275
5.1.5.1 ACPI . . . 275
5.2 Latency Test . . . 275
5.3 Starting LinuxCNC . . . 278
5.3.1 Running LinuxCNC . . . 278
5.3.1.1 Configuration Selector . . . 279
5.4 Stepper Tuning . . . 279
5.4.1 Getting the most out of Software Stepping . . . 279
5.4.1.1 Run a Latency Test . . . 279
5.4.1.2 Figure out what your drives expect . . . 279
5.4.1.3 Choose your BASE_PERIOD . . . 280
5.4.1.4 Use steplen, stepspace, dirsetup, and/or dirhold . . . 281
5.4.1.5 No Guessing! . . . 281
5.5 Stepper Diagnostics . . . 282
5.5.1 Common Problems . . . 282
5.5.1.1 Stepper Moves One Step . . . 282
5.5.1.2 No Steppers Move . . . 282
5.5.1.3 Distance Not Correct . . . 282
5.5.2 Error Messages . . . 282
5.5.2.1 Following Error . . . 282
5.5.2.2 RTAPI Error . . . 283
5.5.3 Testing . . . 283
5.5.3.1 Step Timing . . . 283

**6 Configuration** **284**
6.1 Stepper Quickstart . . . 284
6.1.1 Latency Test . . . 284
6.1.2 Sherline . . . 284
6.1.3 Xylotex . . . 284
6.1.4 Machine Information . . . 284
6.1.5 Pinout Information . . . 285
6.1.6 Mechanical Information . . . 285
6.2 INI Configuration . . . 286
6.2.1 The INI File Components . . . 286
6.2.1.1 Comments . . . 286
6.2.1.2 Sections . . . 287
6.2.1.3 Variables . . . 287
6.2.1.4 Custom Sections and Variables . . . 288
6.2.1.5 Include Files . . . 288
6.2.2 INI File Sections . . . 289
6.2.2.1 [EMC] Section . . . 289
6.2.2.2 [DISPLAY] Section . . . 289
6.2.2.3 [FILTER] Section . . . 291
6.2.2.4 [RS274NGC] Section . . . 292
6.2.2.5 [EMCMOT] Section . . . 293
6.2.2.6 [TASK] Section . . . 293
6.2.2.7 [HAL] section . . . 293
6.2.2.8 [HALUI] section . . . 294
6.2.2.9 [APPLICATIONS] Section . . . 295
6.2.2.10 [TRAJ] Section . . . 295
6.2.2.11 [AXIS_<num>] Section . . . 297
6.2.2.12 [EMCIO] Section . . . 302
6.3 Homing Configuration . . . 303
6.3.1 Overview . . . 303
6.3.2 Homing Sequence . . . 303
6.3.3 Configuration . . . 305
6.3.3.1 HOME_SEARCH_VEL . . . 305
6.3.3.2 HOME_LATCH_VEL . . . 305
6.3.3.3 HOME_FINAL_VEL . . . 305
6.3.3.4 HOME_IGNORE_LIMITS . . . 305
6.3.3.5 HOME_USE_INDEX . . . 306
6.3.3.6 HOME_OFFSET . . . 306
6.3.3.7 HOME . . . 306

6.3.3.8 HOME_IS_SHARED . . . 306
6.3.3.9 HOME_SEQUENCE . . . 306
6.3.3.10 VOLATILE_HOME . . . 306
6.3.3.11 LOCKING_INDEXER . . . 307
6.3.3.12 Immediate Homing . . . 307
6.4 Lathe Configuration . . . 307
6.4.1 Default Plane . . . 307
6.4.2 INI Settings . . . 307
6.5 HALTCL Files . . . 307
6.5.1 Compatibility . . . 308
6.5.2 Haltcl Commands . . . 308
6.5.3 Haltcl Inifile variables . . . 308
6.5.4 Converting .hal files to .tcl files . . . 309
6.5.5 Haltcl Notes . . . 310
6.5.6 Haltcl Examples . . . 310
6.5.7 Haltcl Interactive . . . 310
6.5.8 Haltcl Distribution Examples (sim) . . . 311
6.6 Remap Extending G code . . . 311
6.6.1 Introduction: Extending the RS274NGC Interpreter by Remapping Codes . . . 311
6.6.1.1 A Definition: Remapping Codes . . . 311
6.6.1.2 Why would you want to extend the RS274NGC Interpreter? . . . 311
6.6.2 Getting started . . . 312
6.6.2.1 Picking a code . . . 312
6.6.2.2 Parameter handling . . . 313
6.6.2.3 Handling results . . . 313
6.6.2.4 Execution sequencing . . . 313
6.6.2.5 An minimal example remapped code . . . 313
6.6.3 Configuring Remapping . . . 314
6.6.3.1 The REMAP statement . . . 314
6.6.3.2 Useful REMAP option combinations . . . 314
6.6.3.3 The argspec parameter . . . 315
6.6.4 Upgrading an existing configuration for remapping . . . 318
6.6.5 Remapping tool change-related codes: T, M6, M61 . . . 319
6.6.5.1 Overview . . . 319
6.6.5.2 Understanding the role of iocontrol with remapped tool change codes . . . 319
6.6.5.3 Specifying the M6 replacement . . . 320
6.6.5.4 Configuring iocontrol with a remapped M6 . . . 322
6.6.5.5 Writing the change and prepare O-word procedures . . . 322
6.6.5.6 Making minimal changes to the built in codes, including `M6` . . . 322

6.6.5.7 Specifying the T (prepare) replacement . . . 323
6.6.5.8 Error handling: dealing with abort . . . 324
6.6.5.9 Error handling: failing a remapped code NGC procedure . . . 325
6.6.6 Remapping other existing codes: S, M0, M1, M60 . . . 325
6.6.6.1 Automatic gear selection be remapping S (set spindle speed) . . . 325
6.6.6.2 Adjusting the behavior of M0, M1, M60 . . . 325
6.6.7 Creating new G-code cycles . . . 326
6.6.8 Configuring Embedded Python . . . 326
6.6.8.1 Python plugin : ini file configuration . . . 326
6.6.8.2 Executing Python statements from the interpreter . . . 327
6.6.9 Programming Embedded Python in the RS274NGC Interpreter . . . 327
6.6.9.1 The Python plugin namespace . . . 327
6.6.9.2 The Interpreter as seen from Python . . . 327
6.6.9.3 The Interpreter `__init__` and `__delete__` functions . . . 328
6.6.9.4 Calling conventions: NGC to Python . . . 328
6.6.9.5 Calling conventions: Python to NGC . . . 331
6.6.9.6 Built in modules . . . 332
6.6.10 Adding Predefined Named Parameters . . . 333
6.6.11 Standard Glue routines . . . 333
6.6.11.1 T: `prepare_prolog` and `prepare_epilog` . . . 333
6.6.11.2 M6: `change_prolog` and `change_epilog` . . . 334
6.6.11.3 G code Cycles: `cycle_prolog` and `cycle_epilog` . . . 335
6.6.11.4 S (Set Speed) : `setspeed_prolog` and `setspeed_epilog` . . . 335
6.6.11.5 F (Set Feed) : `setfeed_prolog` and `setfeed_epilog` . . . 336
6.6.11.6 M61 Set tool number : `settool_prolog` and `settool_epilog` . . . 336
6.6.12 Remapped code execution . . . 336
6.6.12.1 NGC procedure call environment during remaps . . . 336
6.6.12.2 Nested remapped codes . . . 336
6.6.12.3 Sequence number during remaps . . . 336
6.6.12.4 Debugging flags . . . 336
6.6.12.5 Debugging Embedded Python code . . . 336
6.6.13 Axis Preview and Remapped code execution . . . 338
6.6.14 Remappable Codes . . . 338
6.6.14.1 Existing codes which can be remapped . . . 338
6.6.14.2 Currently unallocated G-codes: . . . 338
6.6.14.3 Currently unallocated M-codes: . . . 339
6.6.14.4 readahead time and execution time . . . 340
6.6.14.5 plugin/pickle hack . . . 340
6.6.14.6 Module, methods, classes, etc reference . . . 340

6.6.15 Introduction: Extending Task Execution . . . 340
6.6.15.1 Why would you want to change Task Execution? . . . 340
6.6.15.2 A diagram: task, interp, iocontrol, UI (??) . . . 340
6.6.16 Models of Task execution . . . 340
6.6.16.1 Traditional iocontrol/iocontrolv2 execution . . . 340
6.6.16.2 Redefining IO procedures . . . 340
6.6.16.3 Execution-time Python procedures . . . 340
6.6.17 A short survey of LinuxCNC program execution . . . 340
6.6.17.1 Interpreter state . . . 341
6.6.17.2 Task and Interpreter interaction, Queuing and Read-Ahead . . . 341
6.6.17.3 Predicting the machine position . . . 341
6.6.17.4 Queue-busters break position prediction . . . 341
6.6.17.5 How queue-busters are dealt with . . . 342
6.6.17.6 Word order and execution order . . . 342
6.6.17.7 Parsing . . . 342
6.6.17.8 Execution . . . 343
6.6.17.9 Procedure execution . . . 343
6.6.17.10 How tool change currently works . . . 343
6.6.17.11 How Tx (Prepare Tool) works . . . 344
6.6.17.12 How M6 (Change tool) works . . . 344
6.6.17.13 How M61 (Change tool number) works . . . 345
6.6.18 Optional Interpreter features: ini file configuration . . . 345
6.6.19 Named parameters and inifile variables . . . 346
6.6.20 Named parameters and HAL items . . . 346
6.6.21 Status . . . 347
6.6.22 Changes . . . 347
6.6.23 Debugging . . . 347
6.7 Moveoff Component . . . 347
6.7.1 Modifying an existing configuration . . . 349
6.8 Stepper Configuration . . . 351
6.8.1 Introduction . . . 351
6.8.2 Maximum step rate . . . 352
6.8.3 Pinout . . . 352
6.8.3.1 Standard Pinout HAL . . . 352
6.8.3.2 Overview . . . 354
6.8.3.3 Changing the standard_pinout.hal . . . 354
6.8.3.4 Changing polarity of a signal . . . 354
6.8.3.5 Adding PWM Spindle Speed Control . . . 354
6.8.3.6 Adding an enable signal . . . 355
6.8.3.7 External ESTOP button . . . 355

**7 Control Panels** **356**
7.1 Python Virtual Control Panel . . . 356
7.1.1 Introduction . . . 356
7.1.2 Panel Construction . . . 357
7.1.3 Security . . . 358
7.1.4 AXIS . . . 358
7.1.5 Stand Alone . . . 359
7.1.6 Widgets . . . 360
7.1.6.1 Syntax . . . 360
7.1.6.2 General Notes . . . 360
7.1.6.3 Label . . . 361
7.1.6.4 Multi_Label . . . 362
7.1.6.5 LEDs . . . 362
7.1.6.6 Buttons . . . 363
7.1.6.7 Number Displays . . . 365
7.1.6.8 Number Inputs . . . 367
7.1.6.9 Images . . . 370
7.1.6.10 Containers . . . 371
7.2 PyVCP Examples . . . 375
7.2.1 AXIS . . . 375
7.2.2 Floating . . . 375
7.2.3 Jog Buttons . . . 376
7.2.3.1 Create the Widgets . . . 377
7.2.3.2 Make Connections . . . 379
7.2.4 Port Tester . . . 379
7.2.5 GS2 RPM Meter . . . 382
7.2.5.1 The Panel . . . 382
7.2.5.2 The Connections . . . 384
7.3 Glade Virtual Control Panel . . . 385
7.3.1 What is GladeVCP? . . . 385
7.3.1.1 PyVCP versus GladeVCP at a glance . . . 385
7.3.2 A Quick Tour with the Example Panel . . . 386
7.3.2.1 Exploring the example panel . . . 388
7.3.2.2 Exploring the User Interface description . . . 388
7.3.2.3 Exploring the Python callback . . . 389
7.3.3 Creating and Integrating a Glade user interface . . . 389
7.3.3.1 Prerequisite: Glade installation . . . 389
7.3.3.2 Running Glade to create a new user interface . . . 389
7.3.3.3 Testing a panel . . . 390

7.3.3.4 Preparing the HAL command file . . . 390
7.3.3.5 Integrating into Axis like PyVCP . . . 391
7.3.3.6 Embeding as a Tab . . . 391
7.3.3.7 Integrating into Touchy . . . 392
7.3.4 GladeVCP command line options . . . 392
7.3.5 Understanding the gladeVCP startup process . . . 393
7.3.6 HAL Widget reference . . . 394
7.3.6.1 Widget and HAL pin naming . . . 394
7.3.6.2 Python attributes and methods of HAL Widgets . . . 395
7.3.6.3 Setting pin and widget values . . . 395
7.3.6.4 The hal-pin-changed signal . . . 395
7.3.6.5 Buttons . . . 396
7.3.6.6 Scales . . . 397
7.3.6.7 SpinButton . . . 397
7.3.6.8 Hal_Dial . . . 397
7.3.6.9 Jog Wheel . . . 399
7.3.6.10 Speed Control . . . 401
7.3.6.11 Label . . . 402
7.3.6.12 Containers . . . 402
7.3.6.13 LED . . . 403
7.3.6.14 ProgressBar . . . 404
7.3.6.15 ComboBox . . . 404
7.3.6.16 Bars . . . 405
7.3.6.17 Meter . . . 406
7.3.6.18 HAL_Graph . . . 407
7.3.6.19 Gremlin tool path preview for .ngc files . . . 407
7.3.6.20 HAL_Offset . . . 409
7.3.6.21 DRO widget . . . 409
7.3.6.22 Combi_DRO widget . . . 410
7.3.6.23 IconView (File Select) . . . 413
7.3.6.24 Calculator widget . . . 416
7.3.6.25 Tooleditor widget . . . 417
7.3.6.26 Offsetpage . . . 417
7.3.6.27 HAL_sourceview widget . . . 419
7.3.6.28 MDI history . . . 420
7.3.6.29 Animated function diagrams: HAL widgets in a bitmap . . . 420
7.3.7 Action Widgets reference . . . 421
7.3.7.1 VCP Action widgets . . . 422
7.3.7.2 VCP ToggleAction widgets . . . 422

7.3.7.3 The Action_MDI Toggle and Action_MDI widgets . . . 422
7.3.7.4 A simple example: Execute MDI command on button press . . . 422
7.3.7.5 Parameter passing with Action_MDI and ToggleAction_MDI widgets . . . 423
7.3.7.6 An advanced example: Feeding parameters to an O-word subroutine . . . 423
7.3.7.7 Preparing for an MDI Action, and cleaning up afterwards . . . 424
7.3.7.8 Using the LinuxCNC Stat object to deal with status changes . . . 424
7.3.8 GladeVCP Programming . . . 425
7.3.8.1 User Defined Actions . . . 425
7.3.8.2 An example: adding custom user callbacks in Python . . . 426
7.3.8.3 HAL value change events . . . 426
7.3.8.4 Programming model . . . 426
7.3.8.5 Initialization sequence . . . 427
7.3.8.6 Multiple callbacks with the same name . . . 428
7.3.8.7 The GladeVCP `-U <useropts>` flag . . . 428
7.3.8.8 Persistent variables in GladeVCP . . . 428
7.3.8.9 Using persistent variables . . . 429
7.3.8.10 Saving the state on Gladvcp shutdown . . . 430
7.3.8.11 Saving state when Ctrl-C is pressed . . . 430
7.3.8.12 Hand-editing .ini files . . . 430
7.3.8.13 Adding HAL pins . . . 431
7.3.8.14 Adding timers . . . 431
7.3.8.15 Setting HAL widget properties programmatically . . . 431
7.3.8.16 Examples, and rolling your own GladeVCP application . . . 432
7.3.9 FAQ . . . 432
7.3.10 Troubleshooting . . . 433
7.3.11 Implementation note: Key handling in Axis . . . 433
7.3.12 Adding Custom Widgets . . . 433
7.3.13 Auxiliary Gladevcp Applications . . . 433
7.4 Gscreen . . . 434
7.4.1 Intro . . . 434
7.4.1.1 Glade File . . . 436
7.4.1.2 PyGTK . . . 436
7.4.2 GladeVCP . . . 436
7.4.2.1 Overview . . . 437
7.4.2.2 Build a GladeVCP Panel . . . 437
7.4.3 Building a simple clean-sheet custom screen . . . 438
7.4.4 Handler file example . . . 441
7.4.4.1 Adding Features . . . 441
7.4.5 Gscreen Start Up . . . 442
7.4.6 INI Settings . . . 443

**8 User Interfaces** **444**
8.1 HAL User Interface . . . 444
8.1.1 Introduction . . . 444
8.1.2 Halui pin reference . . . 444
8.2 Halui Examples . . . 449
8.2.1 Remote Start . . . 449
8.2.2 Pause & Resume . . . 450
8.3 Python Interface . . . 450
8.3.1 The linuxcnc Python module . . . 450
8.3.2 Usage Patterns for the LinuxCNC NML interface . . . 451
8.3.3 Reading LinuxCNC status . . . 451
8.3.3.1 linuxcnc.stat attributes . . . 451
8.3.3.2 The axis dictionary . . . 456
8.3.4 Preparing to send commands . . . 457
8.3.5 Sending commands through `linuxcnc.command` . . . 458
8.3.5.1 `linuxcnc.command` attributes . . . 458
8.3.5.2 `linuxcnc.command` methods: . . . 458
8.3.6 Reading the error channel . . . 461
8.3.7 Reading ini file values . . . 461
8.3.8 The `linuxcnc.positionlogger` type . . . 461
8.3.8.1 members . . . 462
8.3.8.2 methods . . . 462

**9 Drivers** **463**
9.1 Parallel Port Driver . . . 463
9.1.1 Loading . . . 463
9.1.2 PCI Port Address . . . 464
9.1.3 Pins . . . 465
9.1.4 Parameters . . . 465
9.1.5 Functions . . . 465
9.1.6 Common problems . . . 466
9.1.7 Using DoubleStep . . . 466
9.2 AX5214H Driver . . . 466
9.2.1 Installing . . . 466
9.2.2 Pins . . . 467
9.2.3 Parameters . . . 467
9.2.4 Functions . . . 467
9.3 GS2 VFD Driver . . . 467
9.3.1 Command Line Options . . . 467

9.3.2 Pins . . . 468
9.3.3 Parameters . . . 469
9.4 Mesa HostMot2 Driver . . . 469
9.4.1 Introduction . . . 469
9.4.2 Firmware Binaries . . . 469
9.4.3 Installing Firmware . . . 470
9.4.4 Loading HostMot2 . . . 470
9.4.5 Watchdog . . . 470
9.4.5.1 Pins: . . . 470
9.4.5.2 Parameters: . . . 470
9.4.6 HostMot2 Functions . . . 470
9.4.7 Pinouts . . . 471
9.4.8 PIN Files . . . 472
9.4.9 Firmware . . . 472
9.4.10 HAL Pins . . . 472
9.4.11 Configurations . . . 473
9.4.12 GPIO . . . 475
9.4.12.1 Pins . . . 475
9.4.12.2 Parameters . . . 475
9.4.13 StepGen . . . 476
9.4.13.1 Pins . . . 476
9.4.13.2 Parameters . . . 476
9.4.13.3 Output Parameters . . . 477
9.4.14 PWMGen . . . 477
9.4.14.1 Pins . . . 477
9.4.14.2 Parameters . . . 477
9.4.14.3 Output Parameters . . . 478
9.4.15 Encoder . . . 478
9.4.15.1 Pins . . . 478
9.4.15.2 Parameters . . . 479
9.4.16 5i25 Configuration . . . 479
9.4.16.1 Firmware . . . 479
9.4.16.2 Configuration . . . 479
9.4.16.3 SSERIAL Configuration . . . 479
9.4.16.4 7i77 Limits . . . 480
9.4.17 Example Configurations . . . 480
9.5 Motenc Driver . . . 480
9.5.1 Pins . . . 480
9.5.2 Parameters . . . 481

9.5.3 Functions . . . 481
9.6 MB2HAL . . . 482
9.6.1 Example config file. . . . 482
9.7 Opto22 Driver . . . 486
9.7.1 The Adapter Card . . . 486
9.7.2 The Driver . . . 487
9.7.3 Pins . . . 487
9.7.4 Parameters . . . 487
9.7.5 FUNCTIONS . . . 487
9.7.6 Configuring I/O Ports . . . 488
9.7.7 Pin Numbering . . . 488
9.8 Pico Drivers . . . 488
9.8.1 Command Line Options . . . 489
9.8.2 Pins . . . 489
9.8.3 Parameters . . . 490
9.8.4 Functions . . . 491
9.9 Pluto P Driver . . . 491
9.9.1 General Info . . . 491
9.9.1.1 Requirements . . . 491
9.9.1.2 Connectors . . . 492
9.9.1.3 Physical Pins . . . 492
9.9.1.4 LED . . . 492
9.9.1.5 Power . . . 492
9.9.1.6 PC interface . . . 492
9.9.1.7 Rebuilding the FPGA firmware . . . 492
9.9.1.8 For more information . . . 493
9.9.2 Pluto Servo . . . 493
9.9.2.1 Pinout . . . 493
9.9.2.2 Input latching and output updating . . . 495
9.9.2.3 HAL Functions, Pins and Parameters . . . 495
9.9.2.4 Compatible driver hardware . . . 495
9.9.3 Pluto Step . . . 495
9.9.3.1 Pinout . . . 496
9.9.3.2 Input latching and output updating . . . 496
9.9.3.3 Step Waveform Timings . . . 496
9.9.3.4 HAL Functions, Pins and Parameters . . . 497
9.10 Servo To Go Driver . . . 497
9.10.1 Installing . . . 497
9.10.2 Pins . . . 498

9.10.3 Parameters . . . 498
9.10.4 Functions . . . 499
9.11 ShuttleXpress . . . 499
9.11.1 Description . . . 499
9.11.2 Setup . . . 499
9.11.3 Pins . . . 499
9.12 General Mechatronics Driver . . . 500
9.12.1 I/O connectors . . . 501
9.12.1.1 Pins . . . 502
9.12.1.2 Parameters . . . 502
9.12.2 Axis connectors . . . 503
9.12.2.1 Axis interface modules . . . 503
9.12.2.2 Encoder . . . 504
9.12.2.3 Stepgen module . . . 506
9.12.2.4 Enable and Fault signals . . . 509
9.12.2.5 Axis DAC . . . 509
9.12.3 CAN-bus servo amplifiers . . . 510
9.12.3.1 Pins . . . 511
9.12.3.2 Parameters . . . 511
9.12.4 Watchdog timer . . . 511
9.12.4.1 Pins . . . 511
9.12.4.2 Parameters . . . 511
9.12.5 End-, homing- and E-stop switches . . . 512
9.12.5.1 Pins . . . 513
9.12.5.2 Parameters . . . 513
9.12.6 Status LEDs . . . 513
9.12.6.1 CAN . . . 513
9.12.6.2 RS485 . . . 513
9.12.6.3 EMC . . . 514
9.12.6.4 Boot . . . 514
9.12.6.5 Error . . . 514
9.12.7 RS485 I/O expander modules . . . 514
9.12.7.1 Relay output module . . . 515
9.12.7.2 Digital input module . . . 516
9.12.7.3 DAC & ADC module . . . 516
9.12.7.4 Teach Pendant module . . . 517
9.12.8 Errata . . . 518
9.12.8.1 GM6-PCI card Errata . . . 518
9.13 VFS11 VFD Driver . . . 518

9.13.1 Command Line Options . . . 519
9.13.2 Pins . . . 519
9.13.3 Parameters . . . 520
9.13.4 INI file configuration . . . 521
9.13.5 HAL example . . . 522
9.13.6 Panel operation . . . 522
9.13.7 Error Recovery . . . 522
9.13.8 Configuring the VFS11 VFD for Modbus usage . . . 523
9.13.8.1 Connecting the Serial Port . . . 523
9.13.8.2 Modbus setup . . . 523
9.13.9 Programming Note . . . 523

**10 Driver Examples** **524**
10.1 PCI Parallel Port . . . 524
10.2 Spindle Control . . . 525
10.2.1 0-10v Spindle Speed . . . 525
10.2.2 PWM Spindle Speed . . . 525
10.2.3 Spindle Enable . . . 525
10.2.4 Spindle Direction . . . 525
10.2.5 Spindle Soft Start . . . 526
10.2.6 Spindle Feedback . . . 526
10.2.6.1 Spindle Synchronized Motion . . . 526
10.2.6.2 Spindle At Speed . . . 527
10.3 MPG Pendant . . . 528
10.4 GS2 Spindle . . . 530

**11 PLC** **531**
11.1 Classicladder Introduction . . . 531
11.1.1 History . . . 531
11.1.2 Introduction . . . 531
11.1.3 Example . . . 532
11.1.4 Basic Latching On-Off Circuit . . . 532
11.2 Classicladder Programming . . . 533
11.2.1 Ladder Concepts . . . 533
11.2.2 Languages . . . 534
11.2.3 Components . . . 534
11.2.3.1 Files . . . 534
11.2.3.2 Realtime Module . . . 534
11.2.3.3 Variables . . . 534

11.2.4 Loading the Classic Ladder user module . . . 535
11.2.5 Classic Ladder GUI . . . 536
11.2.5.1 Sections Manager . . . 536
11.2.5.2 Section Display . . . 536
11.2.5.3 The Variable Windows . . . 537
11.2.5.4 Symbol Window . . . 540
11.2.5.5 The Editor window . . . 541
11.2.5.6 Config Window . . . 542
11.2.6 Ladder objects . . . 543
11.2.6.1 CONTACTS . . . 543
11.2.6.2 IEC TIMERS . . . 544
11.2.6.3 TIMERS . . . 544
11.2.6.4 MONOSTABLES . . . 545
11.2.6.5 COUNTERS . . . 545
11.2.6.6 COMPARE . . . 546
11.2.6.7 VARIABLE ASSIGNMENT . . . 546
11.2.6.8 COILS . . . 548
11.2.7 Classic Ladder Variables . . . 549
11.2.8 GRAFCET Programming . . . 550
11.2.9 Modbus . . . 551
11.2.9.1 MODBUS Settings . . . 554
11.2.9.2 MODBUS Info . . . 555
11.2.9.3 Communication Errors . . . 555
11.2.9.4 MODBUS Bugs . . . 555
11.2.10 Setting up Classic Ladder . . . 556
11.2.10.1 Add the Modules . . . 556
11.2.10.2 Adding Ladder Logic . . . 556
11.3 Classicladder Examples . . . 562
11.3.1 Wrapping Counter . . . 562
11.3.2 Reject Extra Pulses . . . 563
11.3.3 External E-Stop . . . 564
11.3.4 Timer/Operate Example . . . 567

**12 HAL** **569**
12.1 HAL Introduction . . . 569
12.1.1 HAL is based on traditional system design techniques . . . 569
12.1.1.1 Part Selection . . . 569
12.1.1.2 Interconnection Design . . . 569
12.1.1.3 Implementation . . . 570

12.1.1.4 Testing . . . 570
12.1.1.5 Summary . . . 570
12.1.2 HAL Concepts . . . 571
12.1.3 HAL components . . . 572
12.1.3.1 External Programs with HAL hooks . . . 572
12.1.3.2 Internal Components . . . 572
12.1.3.3 Hardware Drivers . . . 573
12.1.3.4 Tools and Utilities . . . 573
12.1.4 Timing Issues In HAL . . . 573
12.2 Basic HAL Reference . . . 574
12.2.1 HAL Commands . . . 574
12.2.1.1 loadrt . . . 575
12.2.1.2 addf . . . 575
12.2.1.3 loadusr . . . 576
12.2.1.4 net . . . 577
12.2.1.5 setp . . . 578
12.2.1.6 sets . . . 579
12.2.1.7 unlinkp . . . 579
12.2.1.8 Obsolete Commands . . . 579
12.2.2 HAL Data . . . 580
12.2.2.1 Bit . . . 580
12.2.2.2 Float . . . 580
12.2.2.3 s32 . . . 580
12.2.2.4 u32 . . . 580
12.2.3 HAL Files . . . 580
12.2.4 HAL Components . . . 580
12.2.5 Logic Components . . . 581
12.2.5.1 and2 . . . 581
12.2.5.2 not . . . 581
12.2.5.3 or2 . . . 582
12.2.5.4 xor2 . . . 582
12.2.5.5 Logic Examples . . . 582
12.2.6 Conversion Components . . . 583
12.2.6.1 weighted_sum . . . 583
12.3 HAL Tutorial . . . 583
12.3.1 Introduction . . . 583
12.3.1.1 Notation . . . 584
12.3.1.2 Tab-completion . . . 584
12.3.1.3 The RTAPI environment . . . 584

12.3.2 A Simple Example . . . 584
12.3.2.1 Loading a component . . . 584
12.3.2.2 Examining the HAL . . . 585
12.3.2.3 Making realtime code run . . . 586
12.3.2.4 Changing Parameters . . . 587
12.3.2.5 Saving the HAL configuration . . . 588
12.3.2.6 Exiting halrun . . . 588
12.3.2.7 Restoring the HAL configuration . . . 588
12.3.2.8 Removing HAL from memory . . . 589
12.3.3 Halmeter . . . 589
12.3.4 Stepgen Example . . . 591
12.3.4.1 Installing the components . . . 591
12.3.4.2 Connecting pins with signals . . . 592
12.3.4.3 Setting up realtime execution - threads and functions . . . 593
12.3.4.4 Setting parameters . . . 594
12.3.4.5 Run it! . . . 594
12.3.5 Halscope . . . 595
12.3.5.1 Hooking up the scope probes . . . 597
12.3.5.2 Capturing our first waveforms . . . 600
12.3.5.3 Vertical Adjustments . . . 601
12.3.5.4 Triggering . . . 602
12.3.5.5 Horizontal Adjustments . . . 604
12.3.5.6 More Channels . . . 605
12.3.5.7 More samples . . . 606
12.4 General Reference . . . 606
12.4.1 General Naming Conventions . . . 606
12.4.2 Hardware Driver Naming Conventions . . . 607
12.4.2.1 Pin/Parameter names . . . 607
12.4.2.2 Function Names . . . 608
12.5 Core Components . . . 608
12.5.1 Motion . . . 609
12.5.1.1 Options . . . 609
12.5.1.2 Pins . . . 609
12.5.1.3 Parameters . . . 612
12.5.1.4 Functions . . . 612
12.5.2 Axis (Joints) . . . 612
12.5.2.1 Pins . . . 612
12.5.2.2 Parameters . . . 613
12.5.3 iocontrol . . . 614

12.5.3.1 Pins . . . 614
12.5.4 ini settings . . . 614
12.5.4.1 Pins . . . 614
12.6 Canonical Device Interfaces . . . 615
12.6.1 Introduction . . . 615
12.6.2 Digital Input . . . 615
12.6.2.1 Pins . . . 615
12.6.2.2 Parameters . . . 615
12.6.2.3 Functions . . . 615
12.6.3 Digital Output . . . 615
12.6.3.1 Pins . . . 616
12.6.3.2 Parameters . . . 616
12.6.3.3 Functions . . . 616
12.6.4 Analog Input . . . 616
12.6.4.1 Pins . . . 616
12.6.4.2 Parameters . . . 616
12.6.4.3 Functions . . . 616
12.6.5 Analog Output . . . 616
12.6.5.1 Pins . . . 616
12.6.5.2 Parameters . . . 617
12.6.5.3 Functions . . . 617
12.7 HAL Tools . . . 617
12.7.1 Halcmd . . . 617
12.7.2 Halmeter . . . 617
12.7.3 Halshow . . . 619
12.7.4 Halscope . . . 619
12.7.5 Sim Pin . . . 619
12.7.6 Simulate Probe . . . 620
12.7.7 Hal Histogram . . . 621
12.8 Halshow . . . 622
12.8.1 Starting Halshow . . . 622
12.8.2 HAL Tree Area . . . 622
12.8.3 HAL Show Area . . . 624
12.8.4 Watch Tab . . . 627
12.9 HAL Components . . . 628
12.9.1 Commands and Userspace Components . . . 628
12.9.2 Realtime Components List . . . 629
12.9.2.1 Core LinuxCNC components . . . 630
12.9.2.2 Logic and Bitwise components . . . 630

12.9.2.3 Arithmetic and float-components . . . 631
12.9.2.4 Type conversion . . . 632
12.9.2.5 Hardware Drivers . . . 633
12.9.2.6 Kinematics . . . 633
12.9.2.7 Motor control . . . 634
12.9.2.8 BLDC and 3-phase motor control . . . 634
12.9.2.9 Other . . . 634
12.9.3 HAL API calls . . . 636
12.9.4 RTAPI calls . . . 636
12.10 HAL Component Descriptions . . . 637
12.10.1 Stepgen . . . 637
12.10.2 PWMgen . . . 644
12.10.3 Encoder . . . 645
12.10.4 PID . . . 648
12.10.5 Simulated Encoder . . . 650
12.10.6 Debounce . . . 651
12.10.7 Siggen . . . 651
12.10.8 lut5 . . . 652
12.11 HAL Examples . . . 654
12.11.1 Connecting Two Outputs . . . 654
12.11.2 Manual Toolchange . . . 654
12.11.3 Compute Velocity . . . 655
12.11.4 Soft Start . . . 656
12.11.5 Stand Alone HAL . . . 658
12.12 The HAL Component Generator . . . 659
12.12.1 Introduction . . . 659
12.12.2 Installing . . . 660
12.12.3 Definitions . . . 660
12.12.4 Instance creation . . . 660
12.12.5 Implicit Parameters . . . 660
12.12.6 Syntax . . . 660
12.12.6.1 HAL functions . . . 662
12.12.6.2 Options . . . 662
12.12.6.3 License and Authorship . . . 663
12.12.6.4 Per-instance data storage . . . 663
12.12.6.5 Comments . . . 664
12.12.7 Restrictions . . . 664
12.12.8 Convenience Macros . . . 664
12.12.9 Components with one function . . . 665

12.12.10 Component Personality . . . 665
12.12.11 Compiling . . . 665
12.12.12 Compiling realtime components outside the source tree . . . 665
12.12.13 Compiling userspace components outside the source tree . . . 665
12.12.14 Examples . . . 666
12.12.14.1 constant . . . 666
12.12.14.2 sincos . . . 666
12.12.14.3 out8 . . . 666
12.12.14.4 hal_loop . . . 667
12.12.14.5 arraydemo . . . 667
12.12.14.6 rand . . . 667
12.12.14.7 logic . . . 668
12.13 Creating Userspace Python Components . . . 669
12.13.1 Basic usage . . . 669
12.13.2 Userspace components and delays . . . 669
12.13.3 Create pins and parameters . . . 669
12.13.3.1 Changing the prefix . . . 670
12.13.4 Reading and writing pins and parameters . . . 670
12.13.4.1 Driving output (HAL_OUT) pins . . . 670
12.13.4.2 Driving bidirectional (HAL_IO) pins . . . 670
12.13.5 Exiting . . . 671
12.13.6 Helpful Functions . . . 671
12.13.6.1 component_exists . . . 671
12.13.6.2 component_is_ready . . . 671
12.13.6.3 connect . . . 671
12.13.6.4 new_signal . . . 671
12.13.6.5 pin_has_writer . . . 671
12.13.6.6 set_p . . . 671
12.13.7 Constants . . . 672
12.13.8 System Information . . . 672
12.13.9 Project ideas . . . 672

**13 Advanced Topics** **673**
13.1 Kinematics . . . 673
13.1.1 Introduction . . . 673
13.1.1.1 Joints vs. Axes . . . 673
13.1.2 Trivial Kinematics . . . 673
13.1.3 Non-trivial kinematics . . . 674
13.1.3.1 Forward transformation . . . 675

13.1.3.2 Inverse transformation . . . 675
13.1.4 Implementation details . . . 676
13.2 PID Tuning . . . 676
13.2.1 PID Controller . . . 676
13.2.1.1 Control loop basics . . . 677
13.2.1.2 Theory . . . 677
13.2.1.3 Loop Tuning . . . 678

**IV Glossary** **679**

**V Legal Section** **685**

**14 Copyright Terms** **686**

**15 GNU Free Documentation License** **687**

**16 Index** **691**

# Part I

# Contents

The LinuxCNC Team

This handbook is a work in progress. If you are able to help with writing, editing, or graphic preparation please contact any member of the writing team or join and send an email to emc-users@lists.sourceforge.net.

Copyright © 2000-2015 LinuxCNC.org

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

LINUX® is the registered trademark of Linus Torvalds in the U.S. and other countries. The registered trademark Linux® is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

The LinuxCNC project is not affiliated with Debian®. *Debian* is a registered trademark owned by Software in the Public Interest, Inc.

The LinuxCNC project is not affiliated with UBUNTU®. *UBUNTU* is a registered trademark owned by Canonical Limited.

# Part II

# Using LinuxCNC

# Chapter 1

# General Info

## 1.1 User Foreword

LinuxCNC is modular and flexible. These attributes lead many to see it as a confusing jumble of little things and wonder why it is the way it is. This page attempts to answer that question before you get into the thick of things.

LinuxCNC started at the National Institute of Standards and Technology in the USA. It grew up using Unix as its operating system. Unix made it different. Among early Unix developers there grew a set of code writing ideas that some call the Unix way. These early LinuxCNC authors followed those ways.

Eric S. Raymond, in his book The Art of Unix Programming, summarizes the Unix philosophy as the widely-used engineering philosophy, "Keep it Simple, Stupid" (KISS Principle). He then describes how he believes this overall philosophy is applied as a cultural Unix norm, although unsurprisingly it is not difficult to find severe violations of most of the following in actual Unix practice:

- Rule of Modularity: Write simple parts connected by clean interfaces.
- Rule of Clarity: Clarity is better than cleverness.
- Rule of Composition: Design programs to be connected to other programs.
- Rule of Separation: Separate policy from mechanism; separate interfaces from engines.[1]

Mr. Raymond offered several more rules but these four describe essential characteristics of the LinuxCNC motion control system.

The **Modularity** rule is critical. Throughout these handbooks you will find talk of the interpreter or task planner or motion or HAL. Each of these is a module or collection of modules. It's modularity that allows you to connect together just the parts you need to run your machine.

The **Clarity** rule is essential. LinuxCNC is a work in progress — it is not finished nor will it ever be. It is complete enough to run most of the machines we want it to run. Much of that progress is achieved because many users and code developers are able to look at the work of others and build on what they have done.

The **Composition** rule allows us to build a predictable control system from the many modules available by making them connectable. We achieve connectability by setting up standard interfaces to sets of modules and following those standards.

The **Separation** rule requires that we make distinct parts that do little things. By separating functions debugging is much easier and replacement modules can be dropped into the system and comparisons easily made.

What does the Unix way mean for you as a user of LinuxCNC. It means that you are able to make choices about how you will use the system. Many of these choices are a part of machine integration, but many also affect the way you will use your machine. As you read you will find many places where you will need to make comparisons. Eventually you will make choices, "I'll use this interface rather than that" or, "I'll write part offsets this way rather than that way." Throughout these handbooks we describe the range of abilities currently available.

[1] Found at http://en.wikipedia.org/wiki/Unix_philosophy, 07/06/2008

As you begin your journey into using LinuxCNC we offer two cautionary notes:[2]

- Paraphrasing the words of Doug Gwyn on UNIX: "LinuxCNC was not designed to stop its users from doing stupid things, as that would also stop them from doing clever things."
- Likewise the words of Steven King: "LinuxCNC is user-friendly. It just isn't promiscuous about which users it's friendly with."

## 1.2 LinuxCNC User Introduction

### 1.2.1 How LinuxCNC Works

When you start LinuxCNC from the CNC menu or by typing *linuxcnc* in the terminal the LinuxCNC script starts the Configuration Selector. When you pick a configuration LinuxCNC reads the INI file then loads the HAL files in the order that they are listed in the INI file.

When you create a configuration with the Stepper Configuration Wizard or Mesa Hardware Wizard or by picking a sample configuration from the Configuration Selector LinuxCNC creates some directories and files in your user directory. In the following example the name of the configuration is *My Lathe*.

- linuxcnc
  - configs
    - My_Lathe
      - My_Lathe.ini (this is read by the linuxcnc script)
      - My_Lathe.hal (this hal file is loaded before the gui)
      - post_gui.hal (this hal file is loaded after the gui)
      - linuxcnc.var (stores the parameters)
      - linuxcnc.var.bak (a backup parameter file)
      - tool.tbl (the tool table file)
  - nc_files
    - *.ngc (G code files)

There can be other directorys and files in a configuration but the above is usually the minimum files used.

LinuxCNC can control machine tools, robots, or other automated devices. It can control servo motors, stepper motors, relays, and other devices related to motion control. LinuxCNC can control up to 9 axes in coordinated motion.

There are five main components to the LinuxCNC software:

- a motion controller (EMCMOT)
- a discrete I/O controller (EMCIO)
- a task executor which coordinates them (EMCTASK)
- a graphical user interface.
- a hardware abstraction layer (HAL)

[2] Found at http://en.wikipedia.org/wiki/Unix_philosophy, 07/06/2008

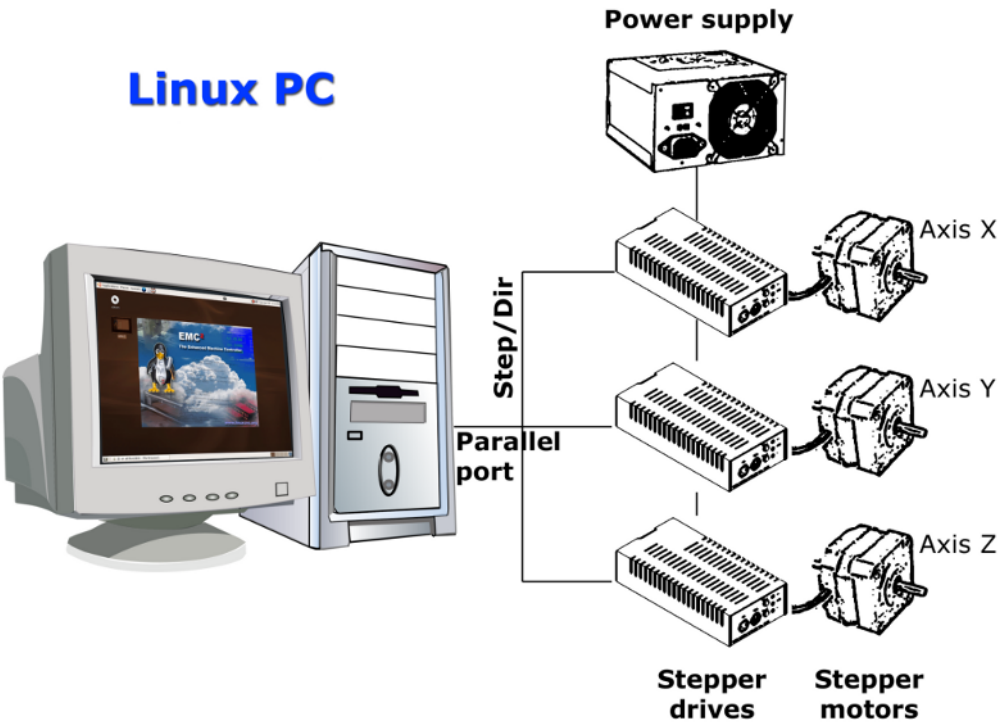


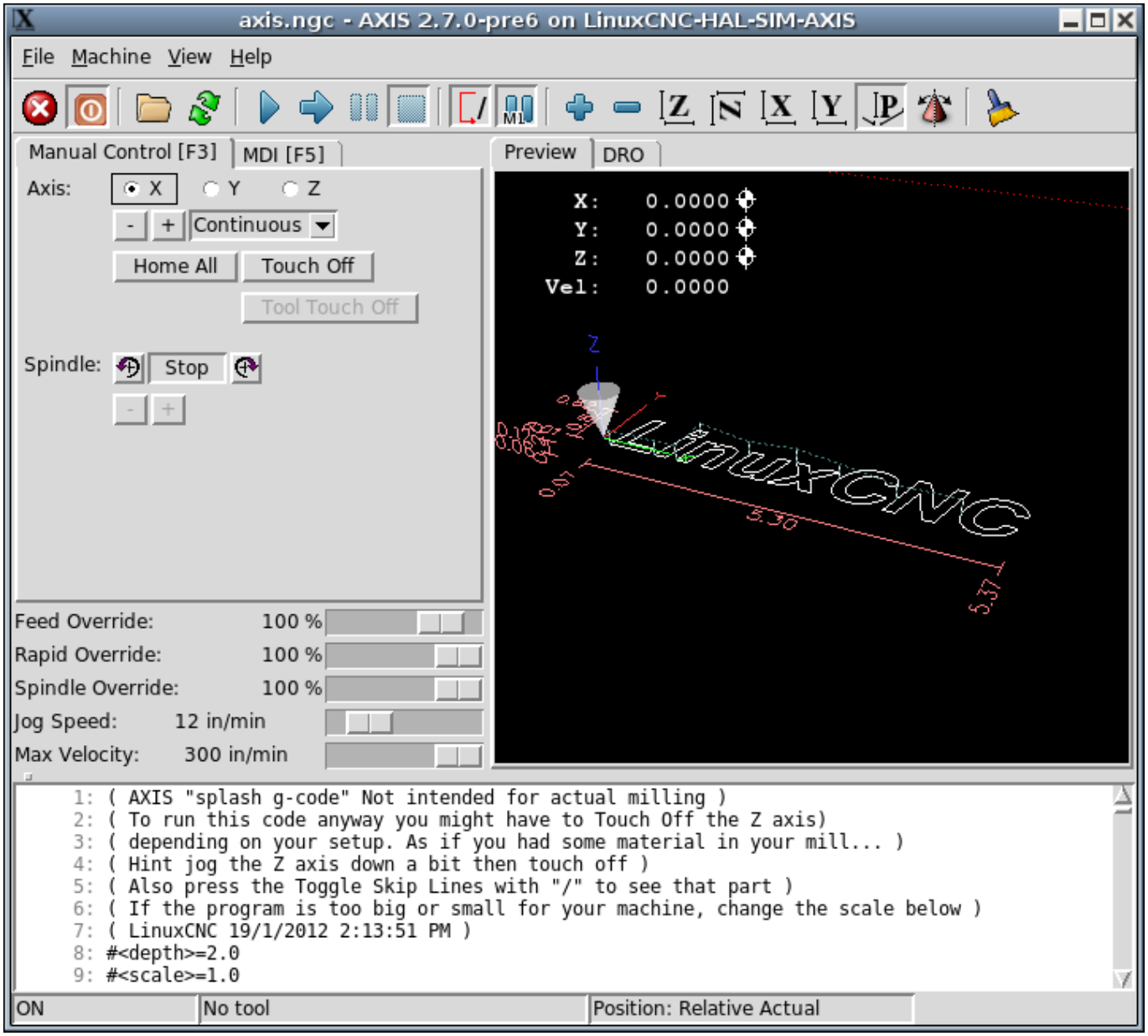

Figure 1.1: Simple LinuxCNC Controlled Machine

The above figure shows a simple block diagram showing what a typical 3-axis LinuxCNC system might look like. This diagram shows a stepper motor system. The PC, running Linux as its operating system, is actually controlling the stepper motor drives by sending signals through the printer port. These signals (pulses) make the stepper drives move the stepper motors. The LinuxCNC system can also run servo motors via servo interface cards or by using an extended parallel port to connect with external control boards. As we examine each of the components that make up an LinuxCNC system we will remind the reader of this typical machine.

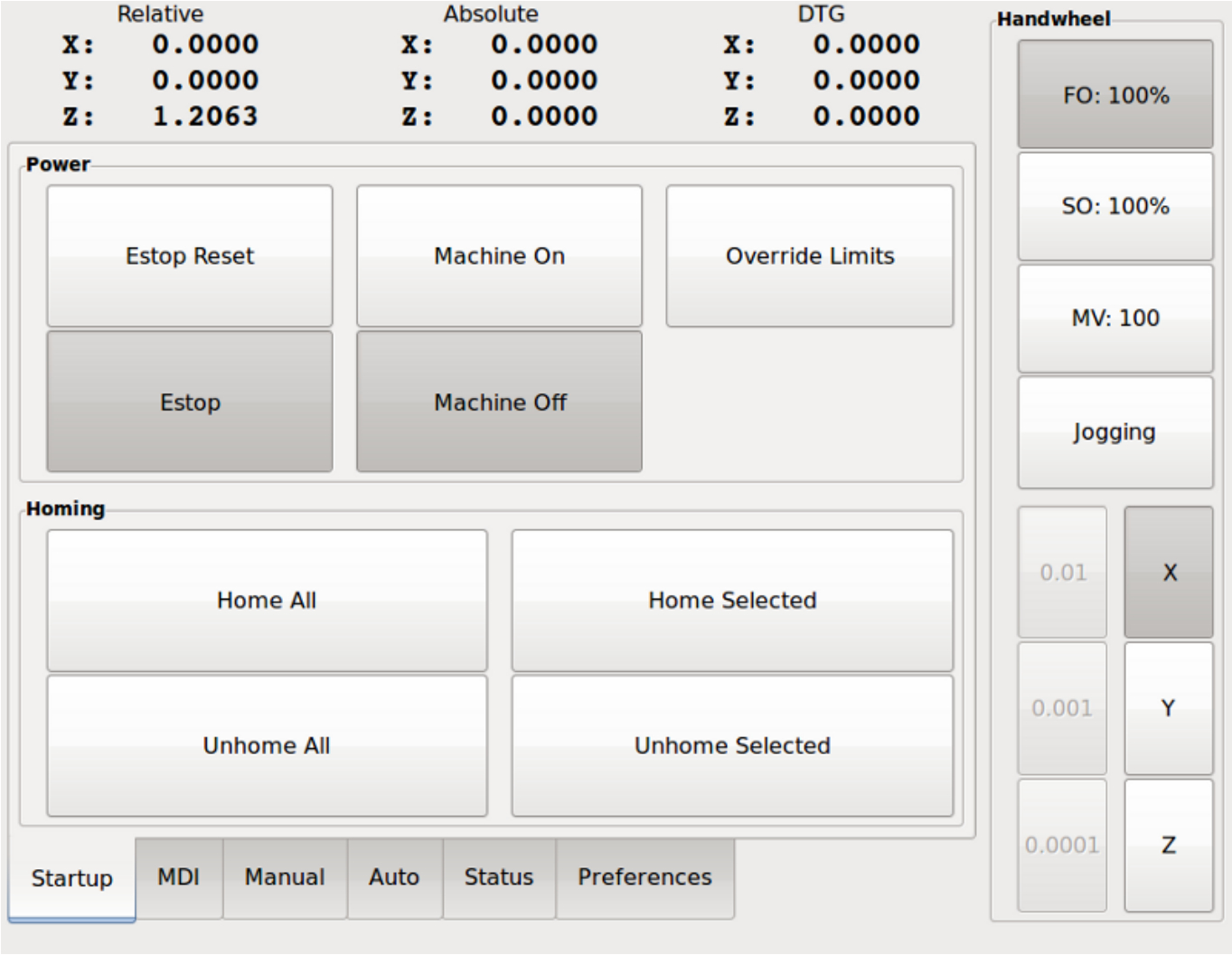
### 1.2.2 Graphical User Interfaces

A user interface is the part of the LinuxCNC that the machine tool operator interacts with. The LinuxCNC comes with several types of user interfaces:

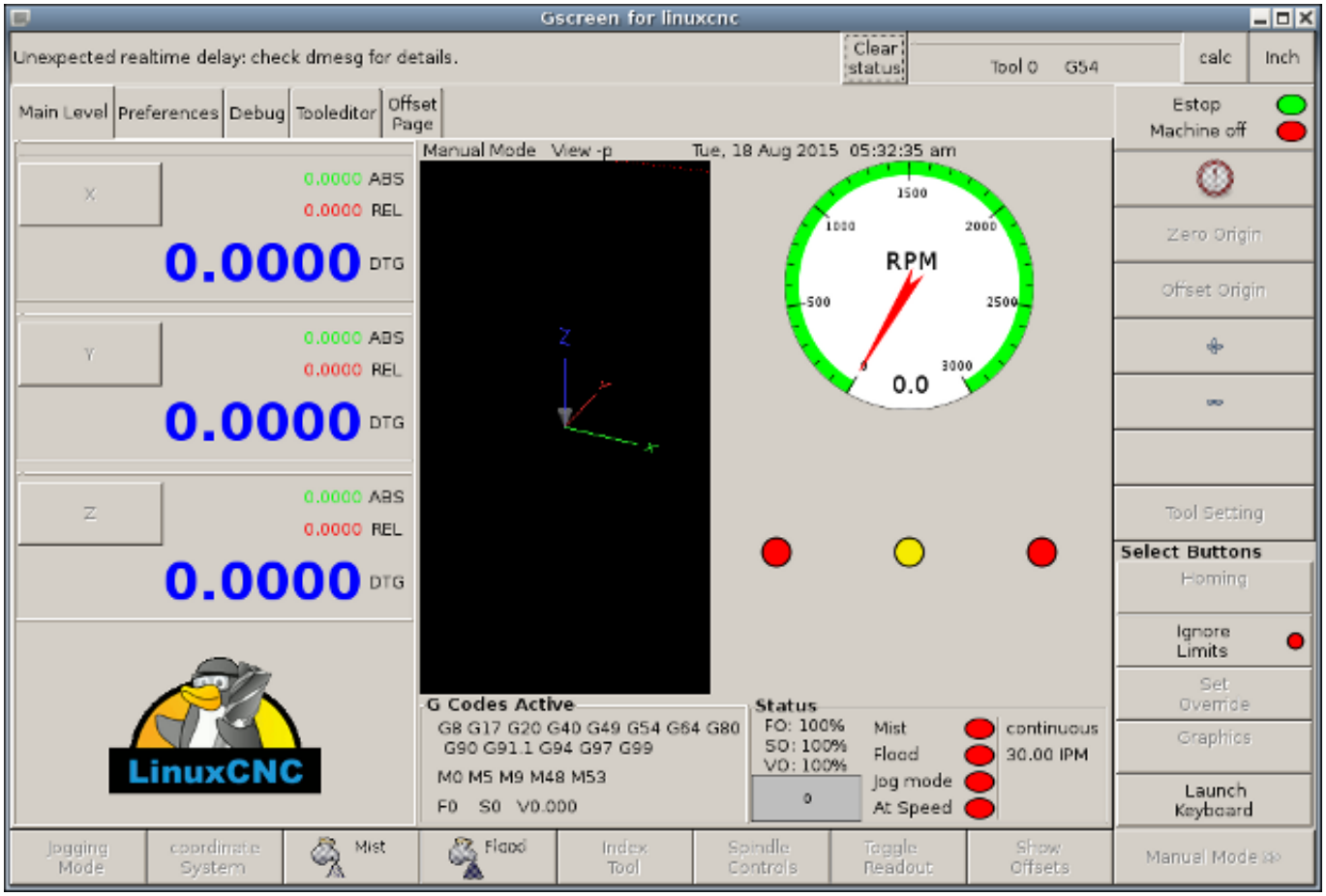
Axis, the standard keyboard GUI interface.



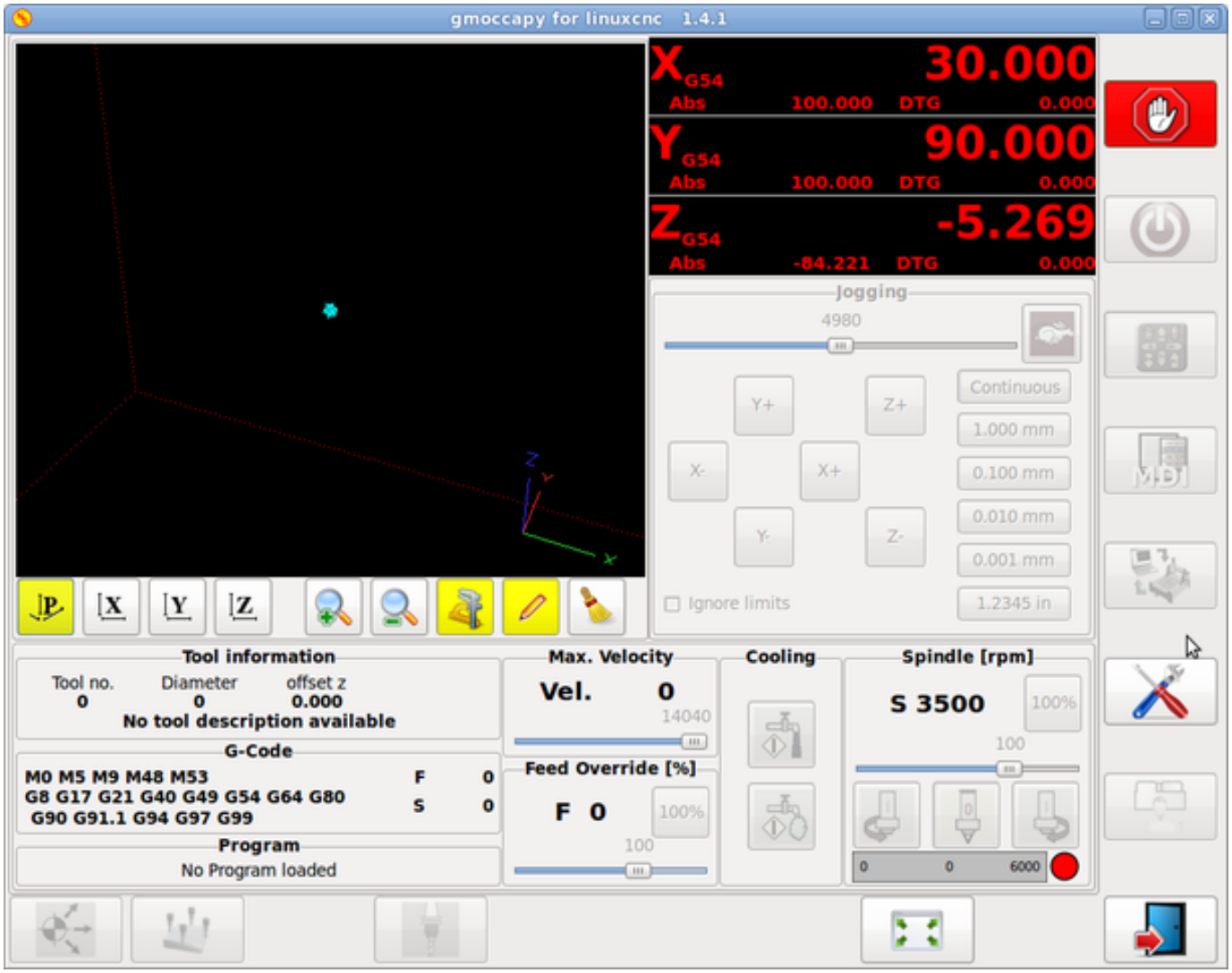

Touchy, a touch screen GUI.



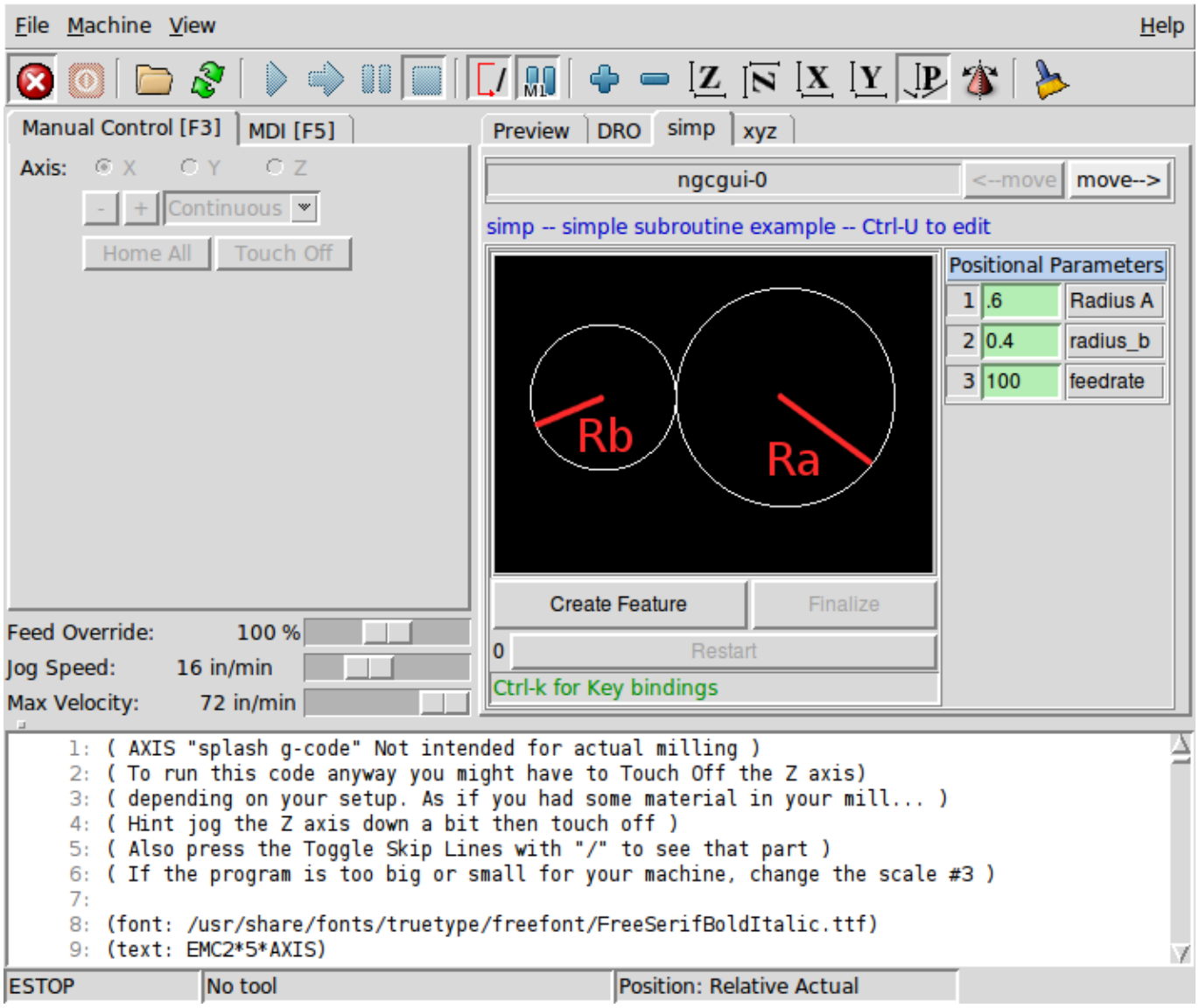

Gscreen, a configurable base touch screen GUI.



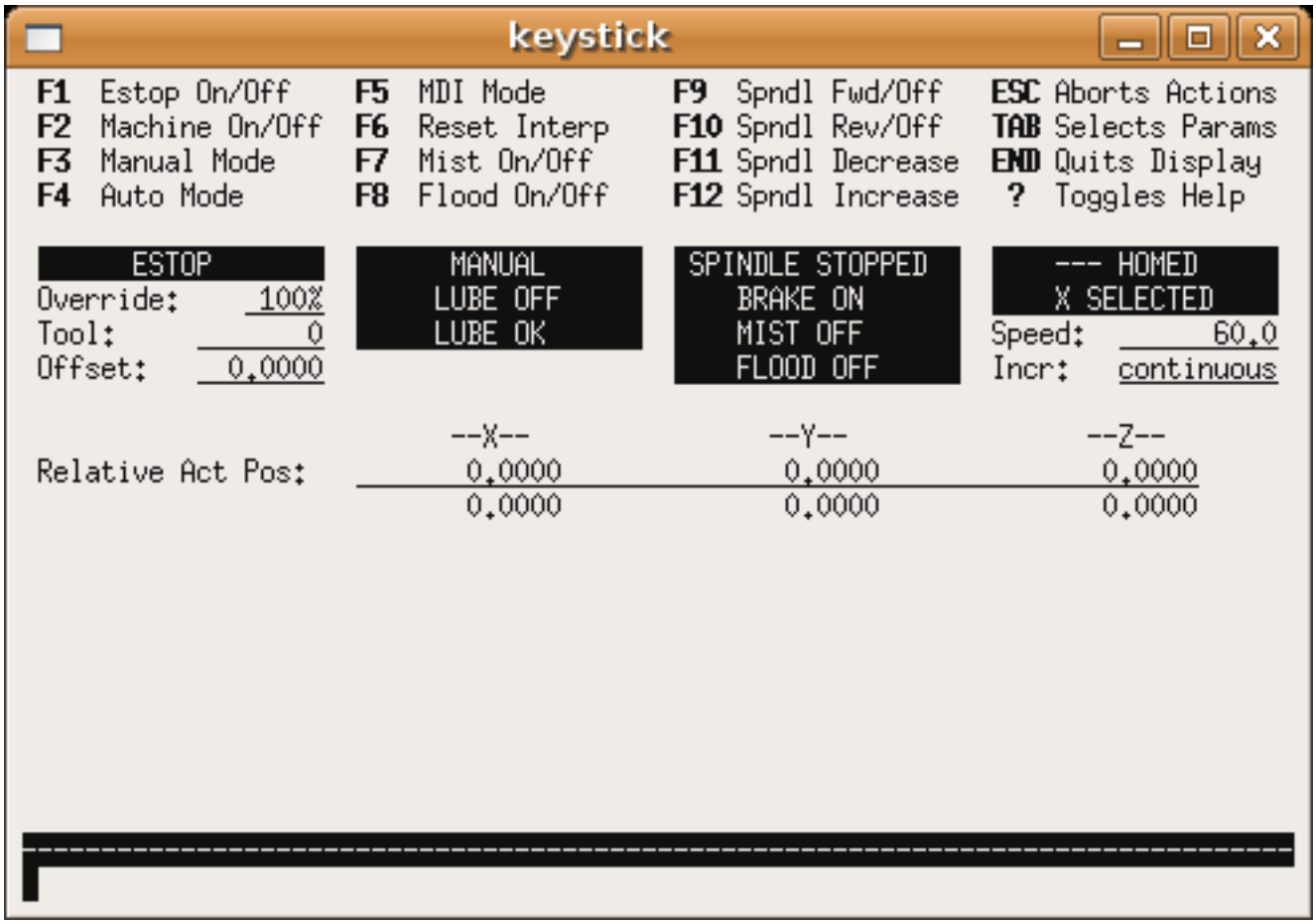

gmoccapy, a touch screen GUI based on Gscreen.




NGCGUI, a subroutine GUI that provides *fill in the blanks* programming of G code. It also supports concatenation of subroutine files to enable you to build a complete G code file without programming. The following screen shot shows NGCGUI imbedded into Axis.




Keystick, a character-based screen graphics program suitable for minimal installations (without the X server running).




#### 1.2.2.1 Additional Features

- *Xemc*, an X-Windows program. A simulator configuration of Xemc can be ran from the configuration picker.
- *halui* - a HAL based user interface which allows to control LinuxCNC using knobs and switches. See the HALUI chapter for more information.
- *linuxcncrsh* - a telnet based user interface which allows commands to be sent to LinuxCNC from remote computers.

### 1.2.3 Virtual Control Panels

- *PyVCP* a python based virtual control panel that can be added to the Axis GUI or be stand alone.

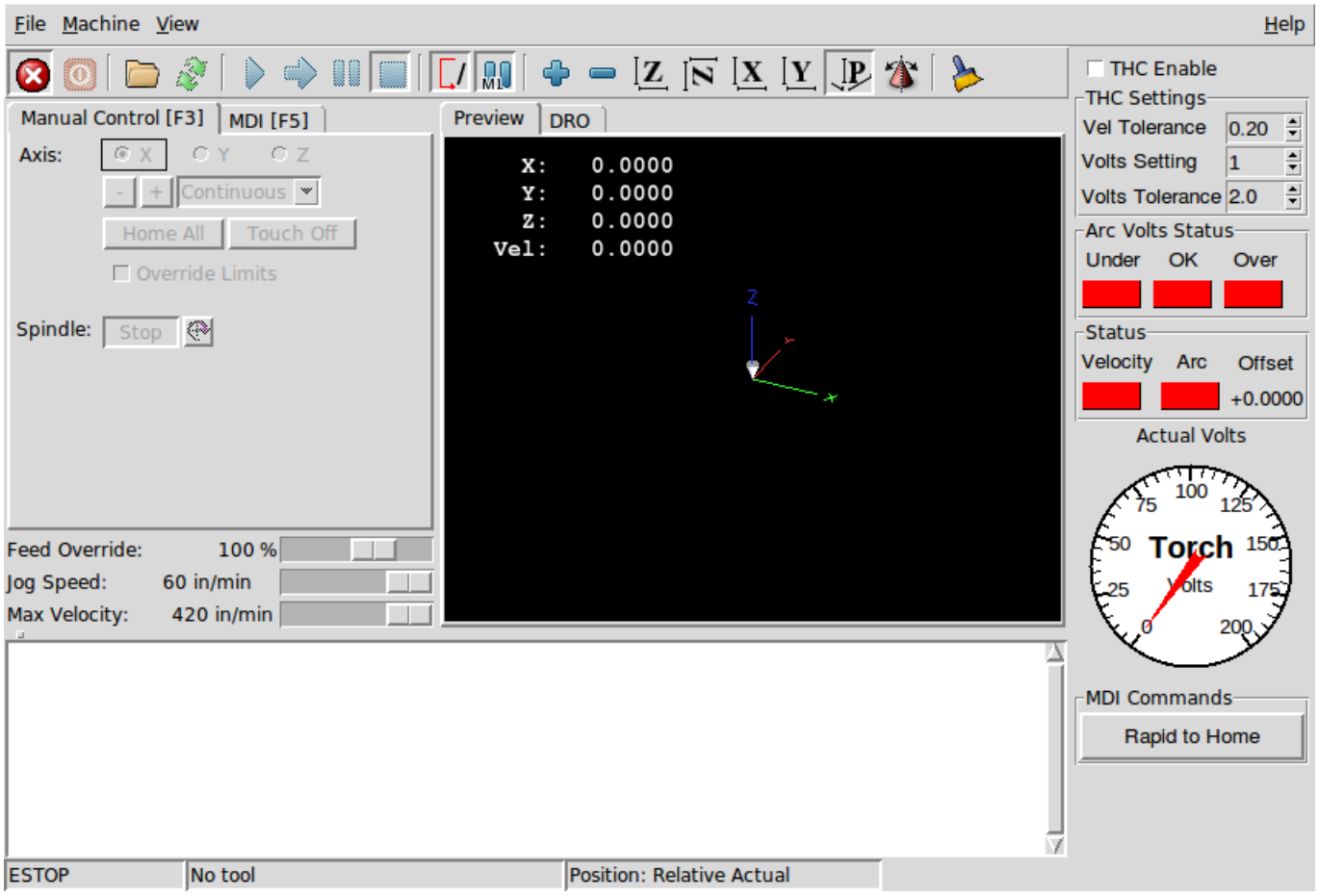



Figure 1.2: PyVCP with Axis

- *GladeVCP* - a glade based virtual control panel that can be added to the Axis GUI or be stand alone.

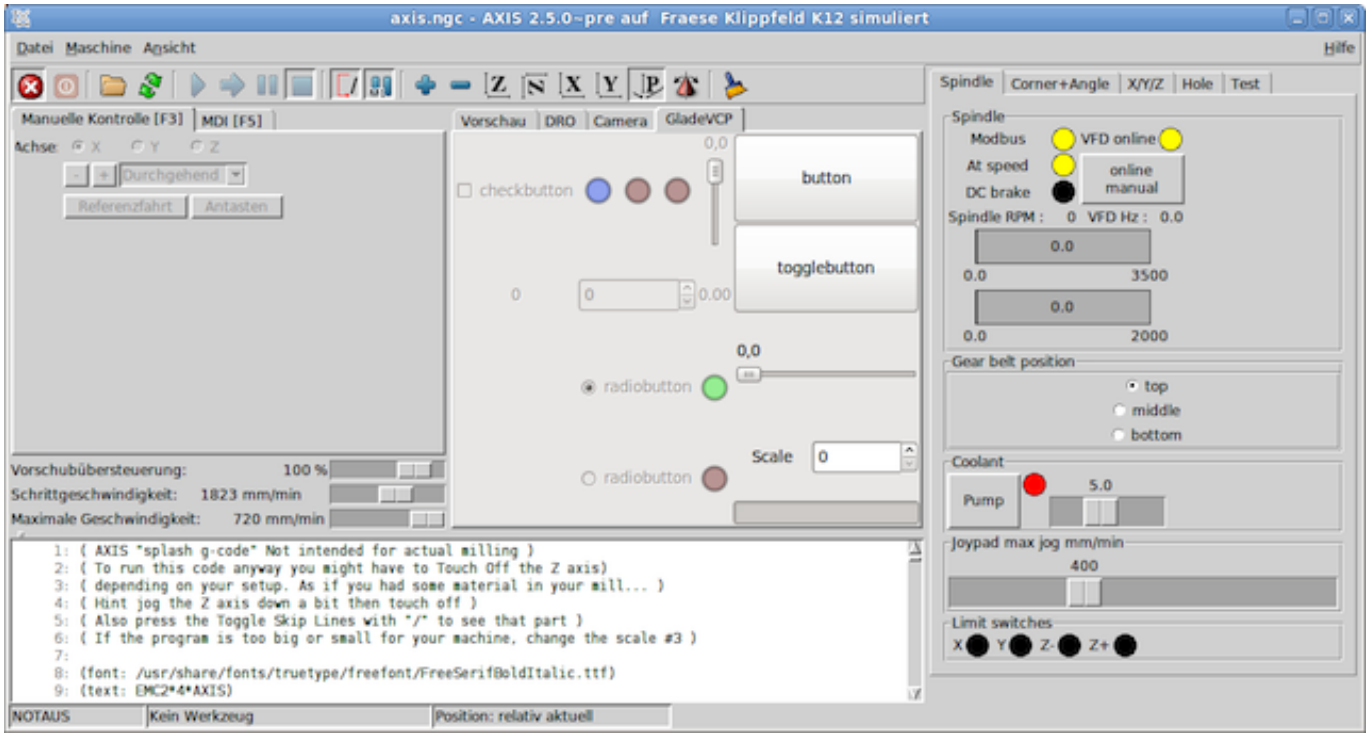



Figure 1.3: GladeVCP with Axis

See the PyVCP chapter and theGladeVCP chapter for more information on Virtual Control Panels.

### 1.2.4 Languages

LinuxCNC uses translation files to translate LinuxCNC User Interfaces into many languages. You just need to log in with the language you intend to use and when you start up LinuxCNC it comes up in that language. If your language has not been translated contact a developer on the IRC or the mailing list if you can assist in the translation.

### 1.2.5 Thinking Like a Machine Operator

This book will not even pretend that it can teach you to run a mill or a lathe. Becoming a machinist takes time and hard work. An author once said, "We learn from experience, if at all." Broken tools, gouged vices, and scars are the evidence of lessons taught. Good part finish, close tolerances, and careful work are the evidence of lessons learned. No machine, no computer program, can take the place of human experience.

As you begin to work with the LinuxCNC program, you will need to place yourself in the position of operator. You need to think of yourself in the role of the one in charge of a machine. It is a machine that is either waiting for your command or executing the command that you have just given it. Throughout these pages we will give information that will help you become a good operator of the LinuxCNC system. You will need some information right up front here so that the following pages will make sense to you.

### 1.2.6 Modes of Operation

When LinuxCNC is running, there are three different major modes used for inputting commands. These are *Manual*, *Auto*, and *MDI*. Changing from one mode to another makes a big difference in the way that the LinuxCNC control behaves. There are specific things that can be done in one mode that cannot be done in another. An operator can home an axis in manual mode but not in auto or MDI modes. An operator can cause the machine to execute a whole file full of G-codes in the auto mode but not in manual or MDI.

In manual mode, each command is entered separately. In human terms a manual command might be *turn on coolant* or *jog X at 25 inches per minute*. These are roughly equivalent to flipping a switch or turning the hand wheel for an axis. These commands are normally handled on one of the graphical interfaces by pressing a button with the mouse or holding down a key on the keyboard. In auto mode, a similar button or key press might be used to load or start the running of a whole program of G-code that is stored in a file. In the MDI mode the operator might type in a block of code and tell the machine to execute it by pressing the <return> or <enter> key on the keyboard.

Some motion control commands are available and will cause the same changes in motion in all modes. These include *abort*, *estop*, and *feed rate override*. Commands like these should be self explanatory.

The AXIS user interface hides some of the distinctions between Auto and the other modes by making Auto-commands available at most times. It also blurs the distinction between Manual and MDI because some Manual commands like Touch Off are actually implemented by sending MDI commands. It does this by automatically changing to the mode that is needed for the action the user has requested.

## 1.3 Important User Concepts

This chapter covers important user concepts that should be understood before attempting to run a CNC machine with g code.

### 1.3.1 Trajectory Control

#### 1.3.1.1 Trajectory Planning

Trajectory planning, in general, is the means by which LinuxCNC follows the path specified by your G Code program, while still operating within the limits of your machinery.

A G Code program can never be fully obeyed. For example, imagine you specify as a single-line program the following move:

```
G1 X1 F10 (G1 is linear move, X1 is the destination, F10 is the speed)
```

In reality, the whole move can't be made at F10, since the machine must accelerate from a stop, move toward X=1, and then decelerate to stop again. Sometimes part of the move is done at F10, but for many moves, especially short ones, the specified feed rate is never reached at all. Having short moves in your G Code can cause your machine to slow down and speed up for the longer moves if the *naive cam detector* is not employed with G64 Pn.

The basic acceleration and deceleration described above is not complex and there is no compromise to be made. In the INI file the specified machine constraints such as maximum axis velocity and axis acceleration must be obeyed by the trajectory planner.

For more information on the Trajectory Panner ini options see the Trajectory Section in the INI chapter.

#### 1.3.1.2 Path Following

A less straightforward problem is that of path following. When you program a corner in G Code, the trajectory planner can do several things, all of which are right in some cases: it can decelerate to a stop exactly at the coordinates of the corner, and then accelerate in the new direction. It can also do what is called blending, which is to keep the feed rate up while going through the corner, making it necessary to round the corner off in order to obey machine constraints. You can see that there is a trade off here: you can slow down to get better path following, or keep the speed up and have worse path following. Depending on the particular cut, the material, the tooling, etc., the programmer may want to compromise differently.

Rapid moves also obey the current trajectory control. With moves long enough to reach maximum velocity on a machine with low acceleration and no path tolerance specified, you can get a fairly round corner.

#### 1.3.1.3 Programming the Planner

The trajectory control commands are as follows:

- *G61* - (Exact Path Mode) visits the programmed point exactly, even though that means it might temporarily come to a complete stop in order to change direction to the next programmed point.
- *G61.1* - (Exact Stop Mode) tells the planner to come to an exact stop at every segment's end.
- *G64* - (Blend Without Tolerance Mode) G64 is the default setting when you start LinuxCNC. G64 is just blending and the naive cam detector is not enabled. G64 and G64 P0 tell the planner to sacrifice path following accuracy in order to keep the feed rate up. This is necessary for some types of material or tooling where exact stops are harmful, and can work great as long as the programmer is careful to keep in mind that the tool's path will be somewhat more curvy than the program specifies. When using G0 (rapid) moves with G64 use caution on clearance moves and allow enough distance to clear obstacles based on the acceleration capabilities of your machine.
- *G64 P- Q-* - (Blend With Tolerance Mode) This enables the *naive cam detector* and enables blending with a tolerance. If you program G64 P0.05, you tell the planner that you want continuous feed, but at programmed corners you want it to slow down enough so that the tool path can stay within 0.05 user units of the programmed path. The exact amount of slowdown depends on the geometry of the programmed corner and the machine constraints, but the only thing the programmer needs to worry about is the tolerance. This gives the programmer complete control over the path following compromise. The blend tolerance can be changed throughout the program as necessary. Beware that a specification of G64 P0 has the same effect as G64 alone (above), which is necessary for backward compatibility for old G Code programs. See the G64 section of the G code chapter.
- *Blending without tolerance* - The controlled point will touch each specified movement at at least one point. The machine will never move at such a speed that it cannot come to an exact stop at the end of the current movement (or next movement, if you pause when blending has already started). The distance from the end point of the move is as large as it needs to be to keep up the best contouring feed.
- *Naive Cam Detector* - Successive G1 moves that involve only the XYZ axes that deviate less than Q- from a straight line are merged into a single straight line. This merged movement replaces the individual G1 movements for the purposes of blending with tolerance. Between successive movements, the controlled point will pass no more than P- from the actual endpoints of the movements. The controlled point will touch at least one point on each movement. The machine will never move at such a speed that it cannot come to an exact stop at the end of the current movement (or next movement, if you pause when blending has already started) On G2/3 moves in the G17 (XY) plane when the maximum deviation of an arc from a straight line is less than the G64 Q- tolerance the arc is broken into two lines (from start of arc to midpoint, and from midpoint to end). those lines are then subject to the naive cam algorithm for lines. Thus, line-arc, arc-arc, and arc-line cases as well as line-line benefit from the *naive cam detector*. This improves contouring performance by simplifying the path.

In the following figure the blue line represents the actual machine velocity. The red lines are the acceleration capability of the machine. The horizontal lines below each plot is the planned move. The upper plot shows how the trajectory planner will slow the machine down when short moves are encountered to stay within the limits of the machines acceleration setting to be able to come to an exact stop at the end of the next move. The bottom plot shows the effect of the Naive Cam Detector to combine the moves and do a better job of keeping the velocity as planned.

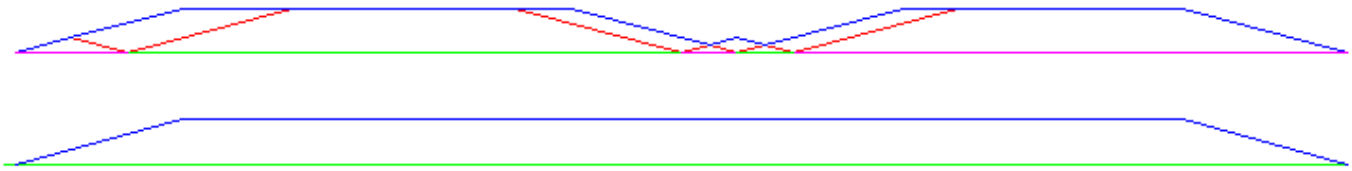


Figure 1.4: Naive Cam Detector

#### 1.3.1.4 Planning Moves

Make sure moves are *long enough* to suit your machine/material. Principally because of the rule that the machine will never move at such a speed that it cannot come to a complete stop at the end of the current movement, there is a minimum movement length that will allow the machine to keep up a requested feed rate with a given acceleration setting.

The acceleration and deceleration phase each use half the ini file MAX_ACCELERATION. In a blend that is an exact reversal, this causes the total axis acceleration to equal the ini file MAX_ACCELERATION. In other cases, the actual machine acceleration is somewhat less than the ini file acceleration

//// This is a duplicate paragraph to the one below without latexmath.////

To keep up the feed rate, the move must be longer than the distance it takes to accelerate from 0 to the desired feed rate and then stop again. Using A as **1/2** the ini file MAX_ACCELERATION and F as the feed rate **in units per second**, the acceleration time is $\mathbf{t_a = F/A}$ and the acceleration distance is $\mathbf{d_a = F*t_a/2}$. The deceleration time and distance are the same, making the critical distance $\mathbf{d = d_a + d_d = 2 * d_a = F^2/A}$.

For example, for a feed rate of 1 inch per second and an acceleration of **10 inches/sec$^2$**, the critical distance is $\mathbf{1^2/10 = 1/10 = 0.1}$ **inches**.

For a feed rate of 0.5 inch per second, the critical distance is $\mathbf{5^2/100 = 25/10 = 0.025}$ inches.

## 1.3.2 G Code

### 1.3.2.1 Defaults

When LinuxCNC first starts up many G and M codes are loaded by default. The current active G and M codes can be viewed on the MDI tab in the *Active G-Codes:* window in the AXIS interface. These G and M codes define the behavior of LinuxCNC and it is important that you understand what each one does before running LinuxCNC. The defaults can be changed when running a G-Code file and left in a different state than when you started your LinuxCNC session. The best practice is to set the defaults needed for the job in the preamble of your G-Code file and not assume that the defaults have not changed. Printing out the G-Code Quick Reference page can help you remember what each one is.

### 1.3.2.2 Feed Rate

How the feed rate is applied depends on if an axis involved with the move is a rotary axis. Read and understand the Feed Rate section if you have a rotary axis or a lathe.

### 1.3.2.3 Tool Radius Offset

Tool Radius Offset (G41/42) requires that the tool be able to touch somewhere along each programmed move without gouging the two adjacent moves. If that is not possible with the current tool diameter you will get an error. A smaller diameter tool may run without an error on the same path. This means you can program a cutter to pass down a path that is narrower than the cutter without any errors. See the Cutter Compensation Section for more information.

## 1.3.3 Homing

After starting LinuxCNC each axis must be homed prior to running a program or running a MDI command.

If your machine does not have home switches a match mark on each axis can aid in homing the machine coordinates to the same place each time.

Once homed your soft limits that are set in the ini file will be used.

If you want to deviate from the default behavior, or want to use the Mini interface you will need to set the option NO_FORCE_HOMING = 1 in the [TRAJ] section of your ini file. More information on homing can be found in the Integrator Manual.

## 1.3.4 Tool Changes

There are several options when doing manual tool changes. See the [EMCIO] section for information on configuration of these options. Also see the G28 and G30 section of the G code chapter.

### 1.3.5 Coordinate Systems

The Coordinate Systems can be confusing at first. Before running a CNC machine you must understand the basics of the coordinate systems used by LinuxCNC. In depth information on the LinuxCNC Coordinate Systems is in the Coordinate System Section of this manual.

#### 1.3.5.1 G53 Machine Coordinate

When you home LinuxCNC you set the G53 Machine Coordinate System to 0 for each axis homed.

- No other coordinate systems or tool offsets are changed by homing.

The only time you move in the G53 machine coordinate system is when you program a G53 on the same line as a move. Normally you are in the G54 coordinate system.

#### 1.3.5.2 G54-59.3 User Coordinates

Normally you use the G54 Coordinate System. When an offset is applied to a current user coordinate system a small blue ball with lines will be at the machine origin when your DRO is displaying *Position: Relative Actual* in Axis. If your offsets are temporary use the Zero Coordinate System from the Machine menu or program *G10 L2 P1 X0 Y0 Z0* at the end of your G Code file. Change the *P* number to suit the coordinate system you wish to clear the offset in.

- Offsets stored in a user coordinate system are retained when LinuxCNC is shut down.
- Using the *Touch Off* button in Axis sets an offset for the chosen User Coordinate System.

#### 1.3.5.3 When You Are Lost

If you're having trouble getting 0,0,0 on the DRO when you think you should, you may have some offsets programmed in and need to remove them.

- Move to the Machine origin with G53 G0 X0 Y0 Z0
- Clear any G92 offset with G92.1
- Use the G54 coordinate system with G54
- Set the G54 coordinate system to be the same as the machine coordinate system with G10 L2 P1 X0 Y0 Z0 R0
- Turn off tool offsets with G49
- Turn on the Relative Coordinate Display from the menu

Now you should be at the machine origin X0 Y0 Z0 and the relative coordinate system should be the same as the machine coordinate system.

### 1.3.6 Machine Configurations

The following diagram shows a typical mill showing direction of travel of the tool and the mill table and limit switches. Notice how the mill table moves in the opposite direction of the Cartesian coordinate system arrows shown by the *Tool Direction* image. This makes the *tool* move in the correct direction in relation to the material.

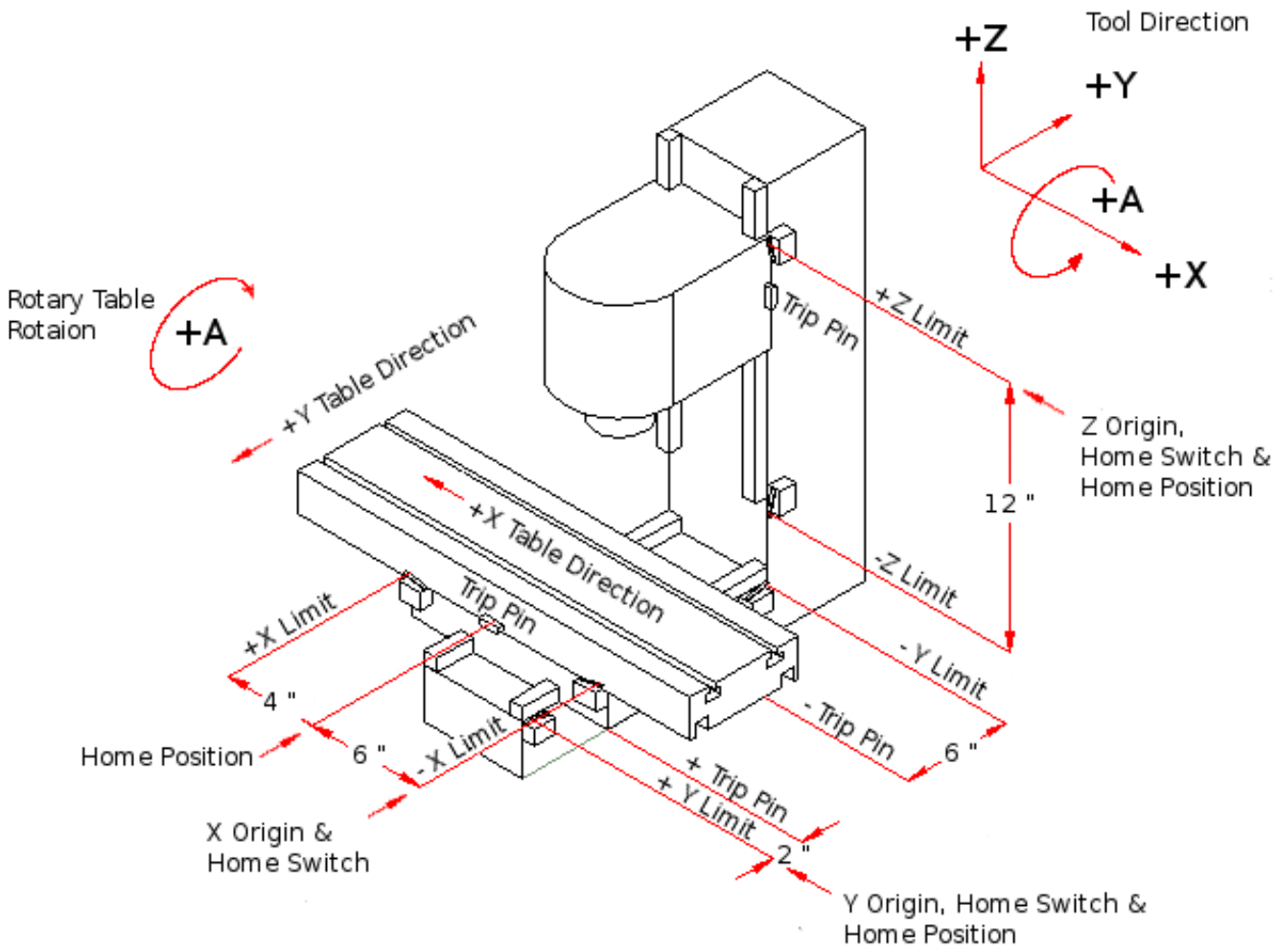



Figure 1.5: Mill Configuration

The following diagram shows a typical lathe showing direction of travel of the tool and limit switches.

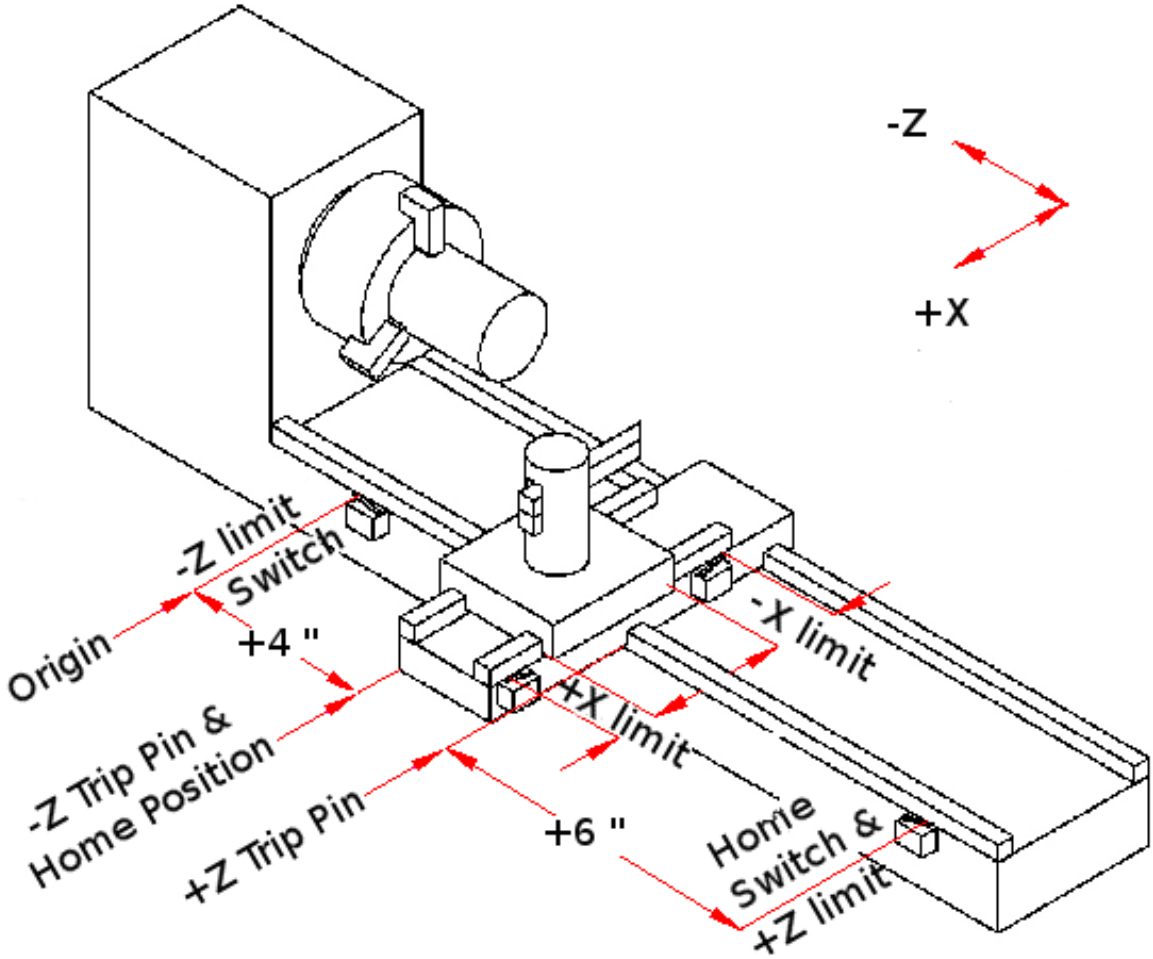



Figure 1.6: Lathe Configuration

## 1.4 CNC Machine Overview

This section gives a brief description of how a CNC machine is viewed from the input and output ends of the Interpreter.

### 1.4.1 Mechanical Components

A CNC machine has many mechanical components that may be controlled or may affect the way in which control is exercised. This section describes the subset of those components that interact with the Interpreter. Mechanical components that do not interact directly with the Interpreter, such as the jog buttons, are not described here, even if they affect control.

#### 1.4.1.1 Axes

Any CNC machine has one or more Axes. Different types of CNC machines have different combinations. For instance, a *4-axis milling machine* may have XYZA or XYZB axes. A lathe typically has XZ axes. A foam-cutting machine may have XYUV axes. In LinuxCNC, the case of a XYYZ *gantry* machine with two motors for one axis is better handled by kinematics rather than by a second linear axis. [3]

[3] If the motion of mechanical components is not independent, as with hexapod machines, the RS274/NGC language and the canonical machining functions will still be usable, as long as the lower levels of control know how to control the actual mechanisms to produce the same relative motion of tool and workpiece

**Primary Linear Axes axesprimary linear primary linear** The X, Y, and Z axes produce linear motion in three mutually orthogonal directions.

**Secondary Linear Axes axessecondary linear secondary linear** The U, V, and W axes produce linear motion in three mutually orthogonal directions. Typically, X and U are parallel, Y and V are parallel, and Z and W are parallel.

**Rotational Axes axesrotational rotational** The A, B and C axes produce angular motion (rotation). Typically, A rotates around a line parallel to X, B rotates around a line parallel to Y, and C rotates around a line parallel to Z.

#### 1.4.1.2 Spindle

A CNC machine typically has a spindle which holds one cutting tool, probe, or the material in the case of a lathe. The spindle may or may not be controlled by the CNC software.

#### 1.4.1.3 Coolant

If a CNC machine has components to provide mist coolant and/or flood coolant they can be controlled by G codes.

#### 1.4.1.4 Feed and Speed Override

A CNC machine can have separate feed and speed override controls, which let the operator specify that the actual feed rate or spindle speed used in machining at some percentage of the programmed rate.

#### 1.4.1.5 Block Delete Switch

A CNC machine can have a block delete switch. See the Block Delete Section.

#### 1.4.1.6 Optional Program Stop Switch

A CNC machine can have an optional program stop switch. See the Optional Program Stop Section.

### 1.4.2 Control and Data Components

#### 1.4.2.1 Linear Axes

The X, Y, and Z axes form a standard right-handed coordinate system of orthogonal linear axes. Positions of the three linear motion mechanisms are expressed using coordinates on these axes.

The U, V and W axes also form a standard right-handed coordinate system. X and U are parallel, Y and V are parallel, and Z and W are parallel (when A, B, and C are rotated to zero).

#### 1.4.2.2 Rotational Axes

The rotational axes are measured in degrees as wrapped linear axes in which the direction of positive rotation is counterclockwise when viewed from the positive end of the corresponding X, Y, or Z-axis. By *wrapped linear axis*, we mean one on which the angular position increases without limit (goes towards plus infinity) as the axis turns counterclockwise and deceases without limit (goes towards minus infinity) as the axis turns clockwise. Wrapped linear axes are used regardless of whether or not there is a mechanical limit on rotation.

Clockwise or counterclockwise is from the point of view of the workpiece. If the workpiece is fastened to a turntable which turns on a rotational axis, a counterclockwise turn from the point of view of the workpiece is accomplished by turning the turntable in a direction that (for most common machine configurations) looks clockwise from the point of view of someone standing next to the machine. [4]

as would be produced by independent axes. This is called *kinematics*.

[4] If the parallelism requirement is violated, the system builder will have to say how to distinguish clockwise from counterclockwise.

#### 1.4.2.3 Controlled Point

The controlled point is the point whose position and rate of motion are controlled. When the tool length offset is zero (the default value), this is a point on the spindle axis (often called the gauge point) that is some fixed distance beyond the end of the spindle, usually near the end of a tool holder that fits into the spindle. The location of the controlled point can be moved out along the spindle axis by specifying some positive amount for the tool length offset. This amount is normally the length of the cutting tool in use, so that the controlled point is at the end of the cutting tool. On a lathe, tool length offsets can be specified for X and Z axes, and the controlled point is either at the tool tip or slightly outside it (where the perpendicular, axis-aligned lines touched by the *front* and *side* of the tool intersect).

#### 1.4.2.4 Coordinated Linear Motion

To drive a tool along a specified path, a machining center must often coordinate the motion of several axes. We use the term *coordinated linear motion* to describe the situation in which, nominally, each axis moves at constant speed and all axes move from their starting positions to their end positions at the same time. If only the X, Y, and Z axes (or any one or two of them) move, this produces motion in a straight line, hence the word *linear* in the term. In actual motions, it is often not possible to maintain constant speed because acceleration or deceleration is required at the beginning and/or end of the motion. It is feasible, however, to control the axes so that, at all times, each axis has completed the same fraction of its required motion as the other axes. This moves the tool along same path, and we also call this kind of motion coordinated linear motion.

Coordinated linear motion can be performed either at the prevailing feed rate, or at traverse rate, or it may be synchronized to the spindle rotation. If physical limits on axis speed make the desired rate unobtainable, all axes are slowed to maintain the desired path.

#### 1.4.2.5 Feed Rate

The rate at which the controlled point moves is nominally a steady rate which may be set by the user. In the Interpreter, the feed rate is interpreted as follows (unless *inverse time feed* or *feed per revolution* modes are being used, in which case see Section G93-G94-G95-Mode).

1. If any of XYZ are moving, F is in units per minute in the XYZ cartesian system, and all other axes (ABCUVW) move so as to start and stop in coordinated fashion.
2. Otherwise, if any of UVW are moving, F is in units per minute in the UVW cartesian system, and all other axes (ABC) move so as to start and stop in coordinated fashion.
3. Otherwise, the move is pure rotary motion and the F word is in rotary units in the ABC *pseudo-cartesian* system.

#### 1.4.2.6 Coolant

Flood coolant and mist coolant may each be turned on independently. The RS274/NGC language turns them off together see Section M7 M8 M9.

#### 1.4.2.7 Dwell

A machining center may be commanded to dwell (i.e., keep all axes unmoving) for a specific amount of time. The most common use of dwell is to break and clear chips, so the spindle is usually turning during a dwell. Regardless of the Path Control Mode (see Section Path Control) the machine will stop exactly at the end of the previous programmed move, as though it was in exact path mode.

#### 1.4.2.8 Units

Units used for distances along the X, Y, and Z axes may be measured in millimeters or inches. Units for all other quantities involved in machine control cannot be changed. Different quantities use different specific units. Spindle speed is measured in revolutions per minute. The positions of rotational axes are measured in degrees. Feed rates are expressed in current length units per minute, or degrees per minute, or length units per spindle revolution, as described in Section G93 G94 G95.

#### 1.4.2.9 Current Position

The controlled point is always at some location called the *current position*, and the controller always knows where that is. The numbers representing the current position must be adjusted in the absence of any axis motion if any of several events take place:

1. Length units are changed.
2. Tool length offset is changed.
3. Coordinate system offsets are changed.

#### 1.4.2.10 Selected Plane

There is always a *selected plane*, which must be the XY-plane, the YZ-plane, or the XZ-plane of the machining center. The Z-axis is, of course, perpendicular to the XY-plane, the X-axis to the YZ-plane, and the Y-axis to the XZ-plane.

#### 1.4.2.11 Tool Carousel

Zero or one tool is assigned to each slot in the tool carousel.

#### 1.4.2.12 Tool Change

A machining center may be commanded to change tools.

#### 1.4.2.13 Pallet Shuttle

The two pallets may be exchanged by command.

#### 1.4.2.14 Path Control Mode

The machining center may be put into any one of three path control modes: (1) exact stop mode, (2) exact path mode, or (3) continuous mode with optional tolerance. In exact stop mode, the machine stops briefly at the end of each programmed move. In exact path mode, the machine follows the programmed path as exactly as possible, slowing or stopping if necessary at sharp corners of the path. In continuous mode, sharp corners of the path may be rounded slightly so that the feed rate may be kept up (but by no more than the tolerance, if specified). See Sections G61/G61.1 and G64.

### 1.4.3 Interpreter Interaction with Switches

The Interpreter interacts with several switches. This section describes the interactions in more detail. In no case does the Interpreter know what the setting of any of these switches is.

#### 1.4.3.1 Feed and Speed Override Switches

The Interpreter will interpret RS274/NGC commands which enable *M48* or disable *M49* the feed and speed override switches. For certain moves, such as the traverse out of the end of a thread during a threading cycle, the switches are disabled automatically.

LinuxCNC reacts to the speed and feed override settings when these switches are enabled.

See the M48 M49 Override section for more information.

#### 1.4.3.2 Block Delete Switch

If the block delete switch is on, lines of G code which start with a slash (the block delete character) are not interpreted. If the switch is off, such lines are interpreted. Normally the block delete switch should be set before starting the NGC program.

#### 1.4.3.3 Optional Program Stop Switch

If this switch is on and an M1 code is encountered, program execution is paused.

### 1.4.4 Tool Table

A tool table is required to use the Interpreter. The file tells which tools are in which tool changer slots and what the size and type of each tool is. The name of the tool table is defined in the ini file:

```
[EMCIO]

# tool table file
TOOL_TABLE = tooltable.tbl
```

The default filename probably looks something like the above, but you may prefer to give your machine its own tool table, using the same name as your ini file, but with a tbl extension:

```
TOOL_TABLE = acme_300.tbl
```

or

```
TOOL_TABLE = EMC-AXIS-SIM.tbl
```

For more information on the specifics of the tool table format, see the Tool Table Format Section.

### 1.4.5 Parameters

In the RS274/NGC language view, a machining center maintains an array of numerical parameters defined by a system definition (RS274NGC_MAX_PARAMETERS). Many of them have specific uses especially in defining coordinate systems. The number of numerical parameters can increase as development adds support for new parameters. The parameter array persists over time, even if the machining center is powered down. LinuxCNC uses a parameter file to ensure persistence and gives the Interpreter the responsibility for maintaining the file. The Interpreter reads the file when it starts up, and writes the file when it exits.

All parameters are available for use in G code programs.

The format of a parameter file is shown in the following table. The file consists of any number of header lines, followed by one blank line, followed by any number of lines of data. The Interpreter skips over the header lines. It is important that there be exactly one blank line (with no spaces or tabs, even) before the data. The header line shown in the following table describes the data columns, so it is suggested (but not required) that that line always be included in the header.

The Interpreter reads only the first two columns of the table. The third column, *Comment*, is not read by the Interpreter.

Each line of the file contains the index number of a parameter in the first column and the value to which that parameter should be set in the second column. The value is represented as a double-precision floating point number inside the Interpreter, but a decimal point is not required in the file. All of the parameters shown in the following table are required parameters and must be included in any parameter file, except that any parameter representing a rotational axis value for an unused axis may be omitted. An error will be signaled if any required parameter is missing. A parameter file may include any other parameter, as long as its number is in the range 1 to 5400. The parameter numbers must be arranged in ascending order. An error will be signaled if not. Any parameter included in the file read by the Interpreter will be included in the file it writes as it exits. The original file is saved as a backup file when the new file is written. Comments are not preserved when the file is written.

Table 1.1: Parameter File Format

| Parameter Number | Parameter Value | Comment |
|---|---|---|
| 5161 | 0.0 | G28 Home X |
| 5162 | 0.0 | G28 Home Y |

See the Parameters section for more information.

## 1.5 Running LinuxCNC

### 1.5.1 Invoking LinuxCNC

After installation, LinuxCNC starts just like any other Linux program: run it from the terminal by issuing the command *linuxcnc*, or select it in the Applications - CNC menu.

### 1.5.2 Configuration Launcher

When starting LinuxCNC from the CNC menu or from the command line without specifiying an ini file the Configuration Selector dialog starts.

The Configuration Selector dialog allows the user to pick one of their existing configurations (My Configurations) or select a new one (from the Sample Configurations) to be copied to their home directory. Copied configurations will appear under My Configurations on the next invocation of the Configuration Selector.

The Configuration Selector offers a selection of configurations organized:

- *My Configurations* - User configurations located in ~/linuxcnc/configs
- *Sample Configurations* - Sample configurations, when selected are copied to ~/linuxcnc/configs. Once you copy a sample configuration if you use the launcher pick it from *My Configurations*
  - *sim* - Configurations that include simulated hardware. These can be used for testing or learning how LinuxCNC works.
  - *by_interface* - Configurations organized by GUI.
  - *by_machine* - Configurations organized by machine.
  - *apps* - Applications that do not require starting linuxcnc but may be useful for testing or trying applications like PyVCP or GladeVCP.
  - *attic* - Obsolete or historical configurations.

The sim configurations are often the most useful starting point for new users and are organized around supported guis:

- axis - Keyboard and Mouse Gui
- gmoccapy - Touch Screen Gui
- gscreen - Touch Screen Gui
- low_graphics - Keyboard Gui
- tklinuxcnc - Keyboard and Mouse Gui(no longer maintained)
- touchy - Touch Screen Gui

A gui configuration directory may contain subdirectories with configurations that illustrate special situations or the embedding of other applications.

The by_interface configurations are organized around common, supported interfaces like:

- general mechatronics
- mesa

- parport
- pico
- pluto
- servotogo
- vigilant
- vitalsystems

Related hardware may be required to use these configurations as starting points for a system.

The by_machine configurations are organized around complete, known systems like:

- boss
- cooltool
- sherline
- smithy
- tormach

A complete system may be required to use these configurations.

The apps items are typically 1) utilities that don't require starting linuxcnc or 2) demonstrations of applications that can be used with linuxcnc:

- info - creates a file with system information that may be useful for problem diagnosis.
- gladevcp - Example gladevcp applications.
- halrun - Starts halrun in an terminal.
- latency - Applications to investigate latency
  - latency-test - standard test
  - latency-plot - stripchart
  - latency-histogram - histogram
- parport - Applications to test parport.
- pyvcp - Example pyvcp applications.
- xhc-hb04 - Applications to test an xhc-hb04 USB wireless MPG

**Note**
Under the Apps directory, only applications that are usefully modified by the user are offered for copying to the user's directory.

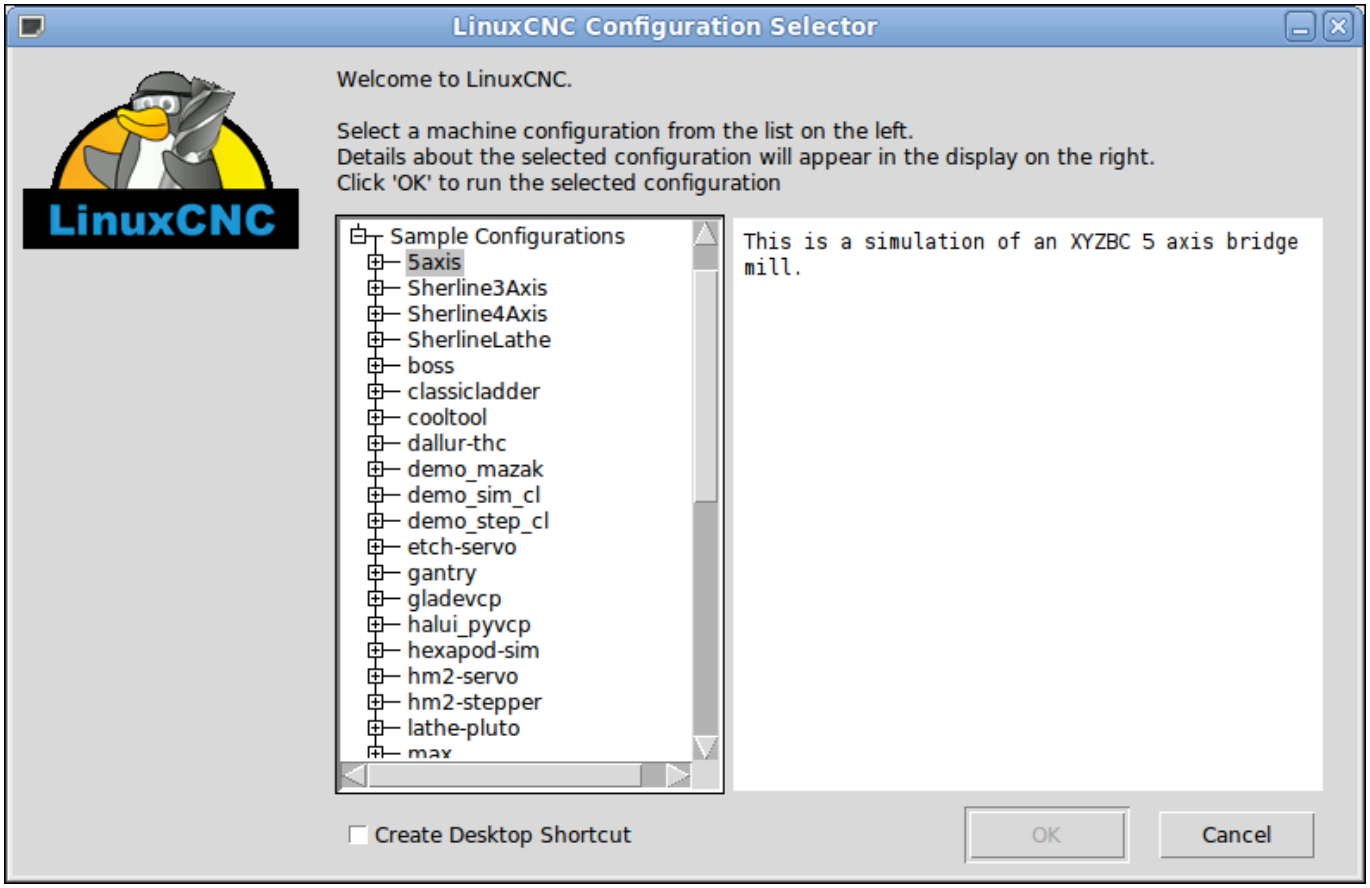



Figure 1.7: LinuxCNC Configuration Selector

Click any of the listed configurations to display specific information about it. Double-click a configuration or click OK to start the configuration. Select *Create Desktop Shortcut* and then click OK to add an icon on the Ubuntu desktop to directly launch this configuration without showing the Configuration Selector screen.

When you select a configuration from the Sample Configurations section, it will automaticly place a copy of that config in the linuxcnc/configs directory.

### 1.5.3 Next steps in configuration

After finding the sample configuration that uses the same interface hardware as your machine (or a simulator configuration), and saving a copy to your home directory, you can customize it according to the details of your machine. Refer to the Integrator Manual for topics on configuration.

### 1.5.4 Simulator Configurations

All configurations listed under Sample Configurations/sim are intended to run on any computer. No specific hardware is required and real-time support is not needed.

These configurations are useful for studying indivdual capabilities or options. The sim configurations are organized according to the graphical user interface used in the demonstration. The directory for axis contains the most choices and subdirectories because it is the most tested GUI. The capabilities demonstrated with any specific GUI may be available in other GUIs as well.

### 1.5.5 Configuration Resources

The Configuration Selector copies all files needed for a configuration to a new subdirectory of ~/linuxcnc/configs (equivalently: /home/username/linuxcnc/configs). Each created directory will include at least one ini file (iniflename.ini) that is used to describe a specific configuration.

File resources within the copied directroy will typically include one or more ini file (filename.ini) for related configurations and a tool table file (toolfilename.tbl). Additionally, resources may include halfiles (filename.hal, filename.tcl), a README file for describing the directory, and configuration specific information in a text file named after a specific configuration (inifilename.txt). That latter two files are displayed when using the Configuration Selector.

The supplied sample configurations may specifiy HALFILEs in the configuration ini file that are not present in the copied directory because they are found in the system Halfile library. These files can be copied to the user configuration directory and altered as required by the user for modification or test. Since the user configuration directory is searched first when finding Halfiles, local modifications will then prevail.

The Configuration selector makes a symbolic link in the user configuration directory (named hallib) that points to the system Halfile library. This link simplifies copying a library file. For example, to copy the library core_sim.hal file in order to make local modifications:

```
cd ~/linuxcnc/configs/name_of_configuration
cp hallib/core_sim.hal core_sim.hal
```

## 1.6 Stepper Configuration Wizard

### 1.6.1 Introduction

LinuxCNC is capable of controlling a wide range of machinery using many different hardware interfaces.

Stepconf is a program that generates configuration files for LinuxCNC for a specific class of CNC machine: those that are controlled via a *standard parallel port*, and controlled by signals of type *step & direction*.

Stepconf is installed when you install LinuxCNC and is in the CNC menu.

Stepconf places a file in the linuxcnc/config directory to store the choices for each configuration you create. When you change something, you need to pick the file that matches your configuration name. The file extension is .stepconf.

The Stepconf Wizard works best with at least 800 x 600 screen resolution.

### 1.6.2 Start Page

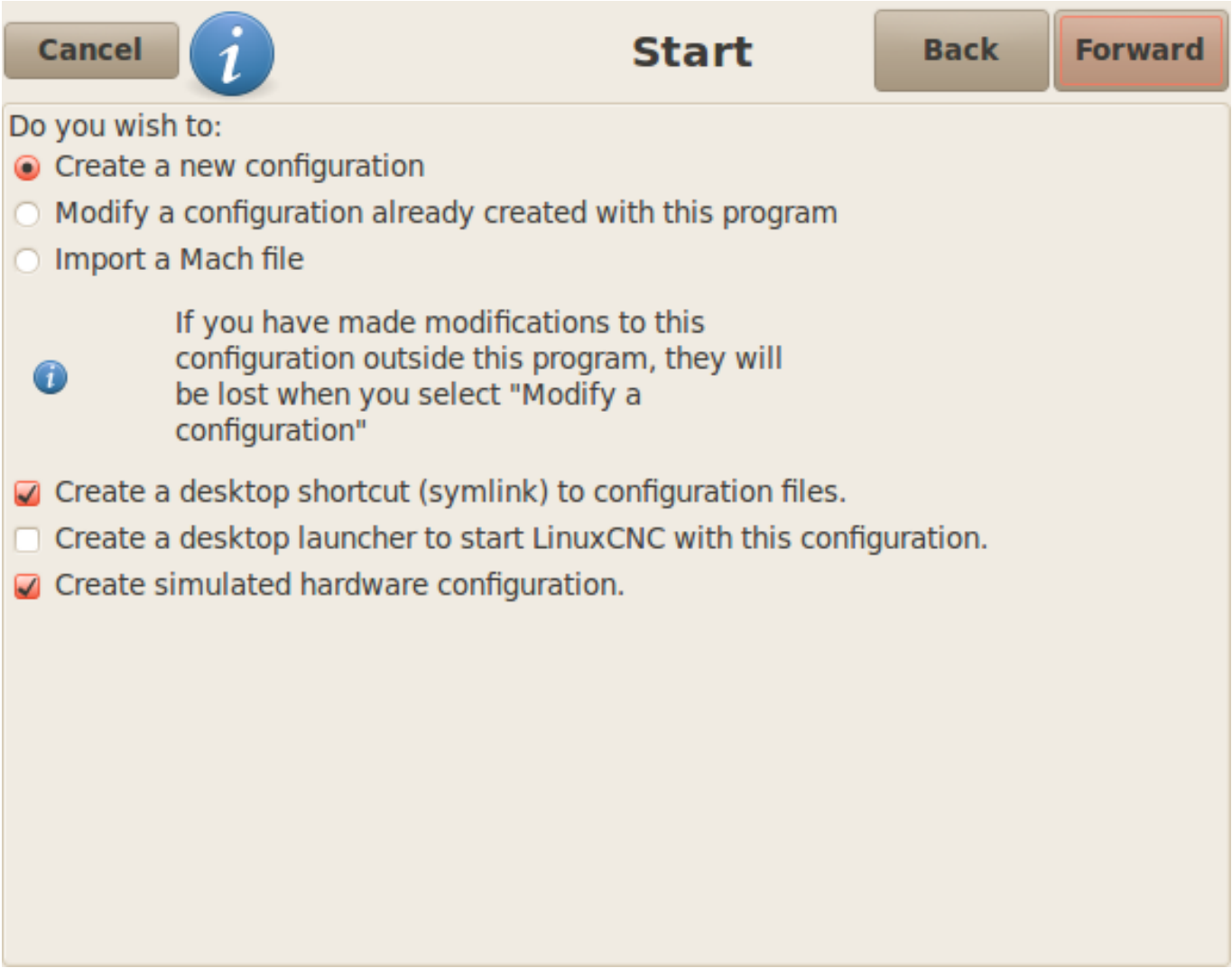



Figure 1.8: Entry Page

- *Create New* - Creates a fresh configuration.
- *Modify* - Modify an existing configuration. After selecting this a file picker pops up so you can select the .stepconf file for modification. If you made any modifications to the main .hal or the .ini file these will be lost. Modifications to custom.hal and custom_postgui.hal will not be changed by the Stepconf Wizard. Stepconf will highlight the lastconf that was built.
- *Import* - Import a Mach configuration file and attempt to convert it to a linuxcnc config file. After the import, you will go though the pages of stepconf to confirm/modify the entries. The original mach xml file will not be changed.

These next options wiil be recorded in a preference file for the next run of Stepconf.

- *Create Desktop Shortcut* - This will place a link on your desktop to the files.
- *Create Desktop Launcher* - This will place a launcher on your desktop to start your application.
- *Create Simulated Hardware* - This allows you to build a config for testing, even if you don't have the actual hardware.

### 1.6.3 Basic Information

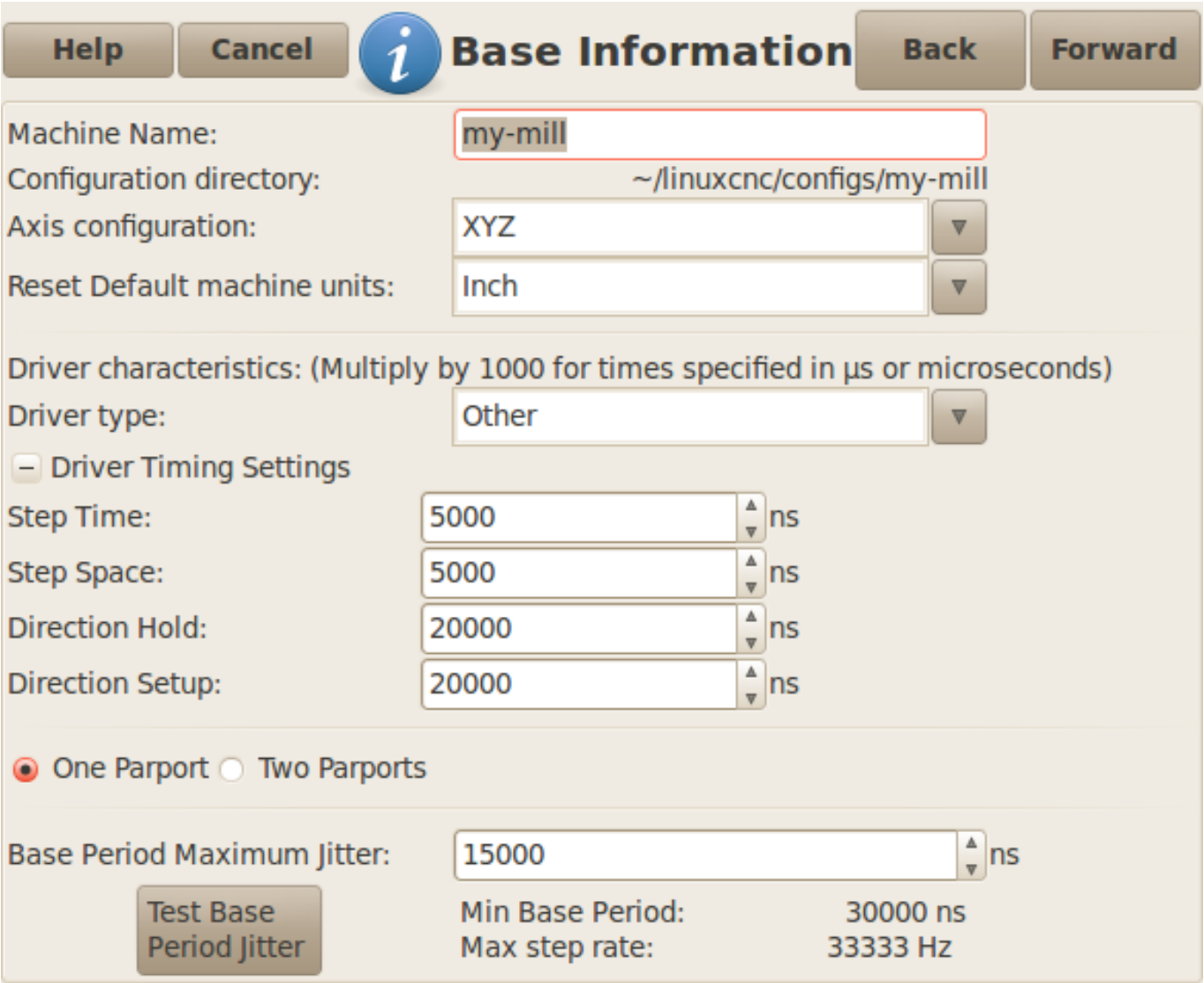



Figure 1.9: Basic Information Page

- *Machine Name* - Choose a name for your machine. Use only uppercase letters, lowercase letters, digits, - and _.
- *Axis Configuration* - Choose XYZ (Mill), XYZA (4-axis mill) or XZ (Lathe).
- *Machine Units* - Choose Inch or mm. All subsequent entries will be in the chosen units. Changing this also changes the default values in the Axes section. If you change this after selecting values in any of the axes sections, they will be over-written by the default values of the selected units.
- *Driver Type* - If you have one of the stepper drivers listed in the pull down box, choose it. Otherwise, select *Other* and find the timing values in your driver's data sheet and enter them as *nano seconds* in the *Driver Timing Settings*. If the data sheet gives a value in microseconds, multiply by 1000. For example, enter 4.5us as 4500ns.

A list of some popular drives, along with their timing values, is on the LinuxCNC.org Wiki under Stepper Drive Timing.

Additional signal conditioning or isolation such as optocouplers and RC filters on break out boards can impose timing constraints of their own, in addition to those of the driver. You may find it necessary to add some time to the drive requirements to allow for this.

The LinuxCNC Configuration Selector has configs for Sherline already configured.

- *Step Time* - How long the step pulse is *on* in nano seconds. If your not sure about this setting a value of 20,000 will work with most drives.
- *Step Space* - Minimum time between step pulses in nano seconds. If your not sure about this setting a value of 20,000 will work with most drives.
- *Direction Hold* - How long the direction pin is held after a change of direction in nanoseconds. If your not sure about this setting a value of 20,000 will work with most drives.
- *Direction Setup* - How long before a direction change after the last step pulse in nanoseconds. If your not sure about this setting a value of 20,000 will work with most drives.
- *One / Two Parport* - Select how many parallel port are to be configured.
- *Base Period Maximum Jitter* - Enter the result of the Latency Test here. To run a latency test press the *Test Base Period Jitter* button. See the Latency Test section for more details.
- *Max Step Rate* -Stepconf automatically calculates the Max Step Rate based on the driver characteristics entered and the latency test result.
- *Min Base Period* - Stepconf automatically determines the Min Base Period based on the driver characteristics entered and latency test result.

### 1.6.4 Latency Test

While the test is running, you should *abuse* the computer. Move windows around on the screen. Surf the web. Copy some large files around on the disk. Play some music. Run an OpenGL program such as glxgears. The idea is to put the PC through its paces while the latency test checks to see what the worst case numbers are. Run the test at least a few minutes. The longer you run the test the better it will be at catching events that might occour at less frequent intervals. This is a test for your computer only, so no hardware needs to be connected to run the test.

**Warning**
Do not attempt run LinuxCNC while the latency test is running.

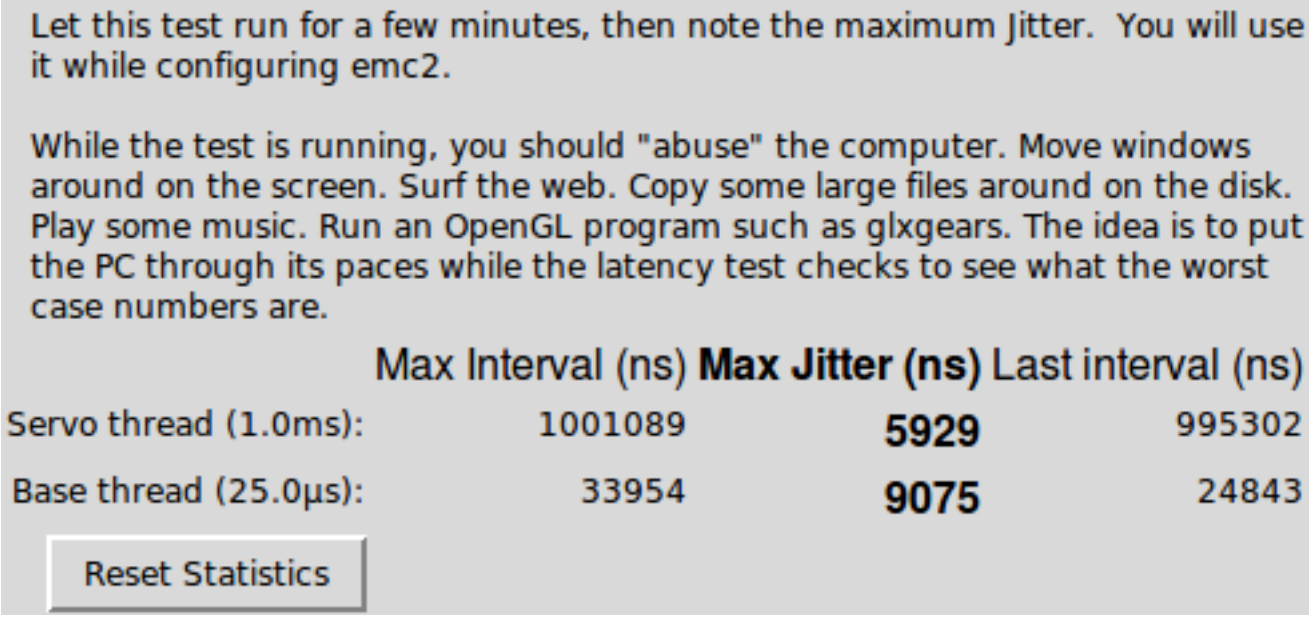



Figure 1.10: Latency Test

Latency is how long it takes the PC to stop what it is doing and respond to an external request. In our case, the request is the periodic *heartbeat* that serves as a timing reference for the step pulses. The lower the latency, the faster you can run the heartbeat, and the faster and smoother the step pulses will be.

Latency is far more important than CPU speed. The CPU isn't the only factor in determining latency. Motherboards, video cards, USB ports, SMI issues, and a number of other things can hurt the latency.

| |
|---|
| **Troubleshooting SMI Issues (LinuxCNC.org Wiki)** |
| Fixing Realtime problems caused by SMI on Ubuntu |
| http://wiki.linuxcnc.org/cgi-bin/wiki.pl?FixingSMIIssues |

The important numbers are the *max jitter*. In the example above 9075 nanoseconds, or 9.075 microseconds, is the highest jitter. Record this number, and enter it in the Base Period Maximum Jitter box.

If your Max Jitter number is less than about 15-20 microseconds (15000-20000 nanoseconds), the computer should give very nice results with software stepping. If the max latency is more like 30-50 microseconds, you can still get good results, but your maximum step rate might be a little disappointing, especially if you use microstepping or have very fine pitch leadscrews. If the numbers are 100 us or more (100,000 nanoseconds), then the PC is not a good candidate for software stepping. Numbers over 1 millisecond (1,000,000 nanoseconds) mean the PC is not a good candidate for LinuxCNC, regardless of whether you use software stepping or not.

### 1.6.5 Parallel Port Setup

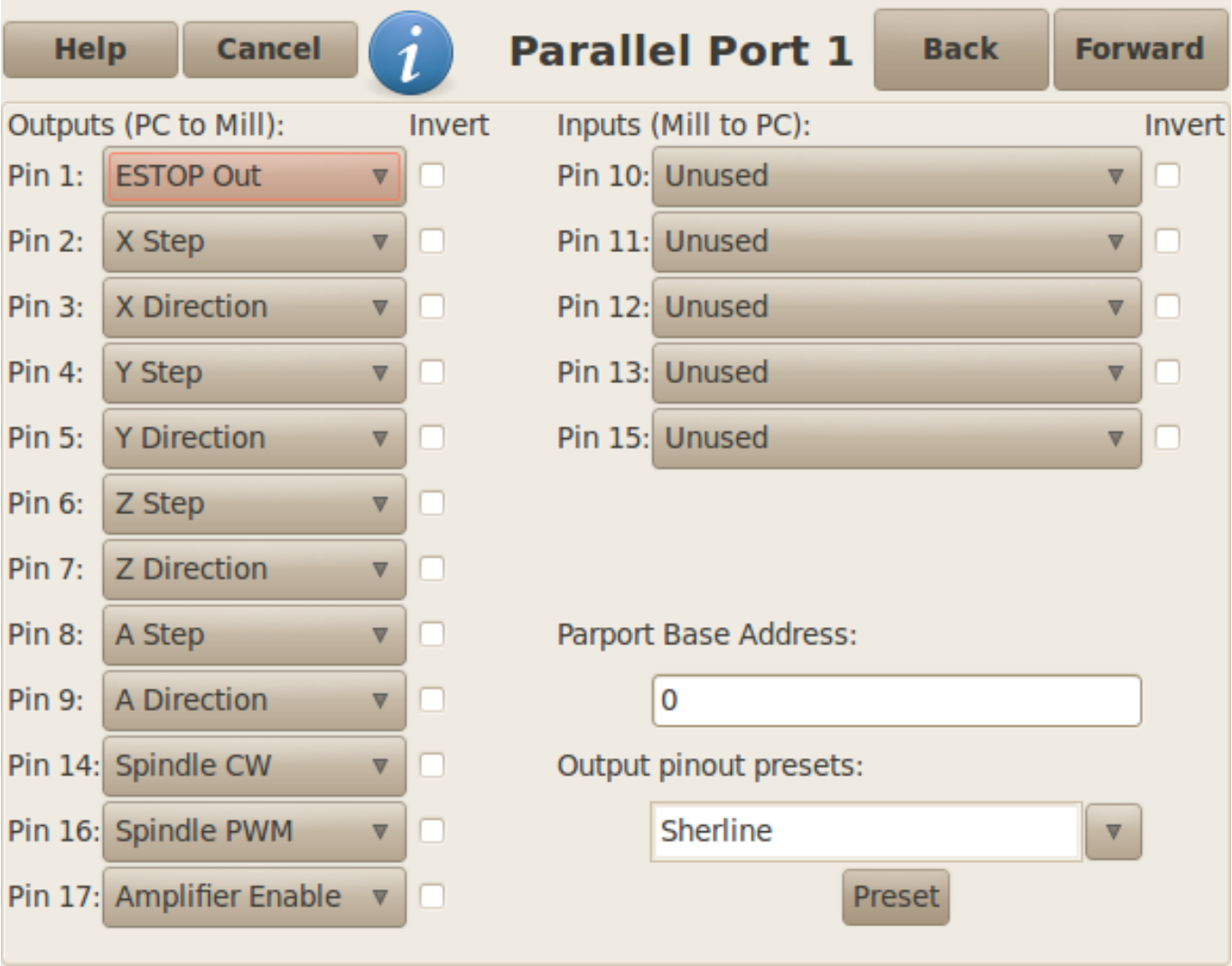



Figure 1.11: Parallel Port Setup Page

You may specify the address as a hexidecimal (often 0x378) or as linux's default port number (probably 0)

For each pin, choose the signal which matches your parallel port pinout. Turn on the *invert* check box if the signal is inverted (0V for true/active, 5V for false/inactive).

- *Output pinout presets* - Automatically set pins 2 through 9 according to the Sherline standard (Direction on pins 2, 4, 6, 8) or the Xylotex standard (Direction on pins 3, 5, 7, 9).
- *Inputs and Outputs* - If the input or output is not used set the option to *Unused*.
- *External E Stop* - This can be selected from an input pin drop down box. A typical E Stop chain uses all normally closed contacts.
- *Homing & Limit Switches* - These can be selected from an input pin drop down box for most configurations.
- *Charge Pump* - If your driver board requires a charge pump signal select Charge Pump from the drop down list for the output pin you wish to connect to your charge pump input. The charge pump output is connected to the base thread by Stepconf. The charge pump output will be about 1/2 of the maximum step rate shown on the Basic Machine Configuration page.

### 1.6.6 Parallel Port 2 Setup

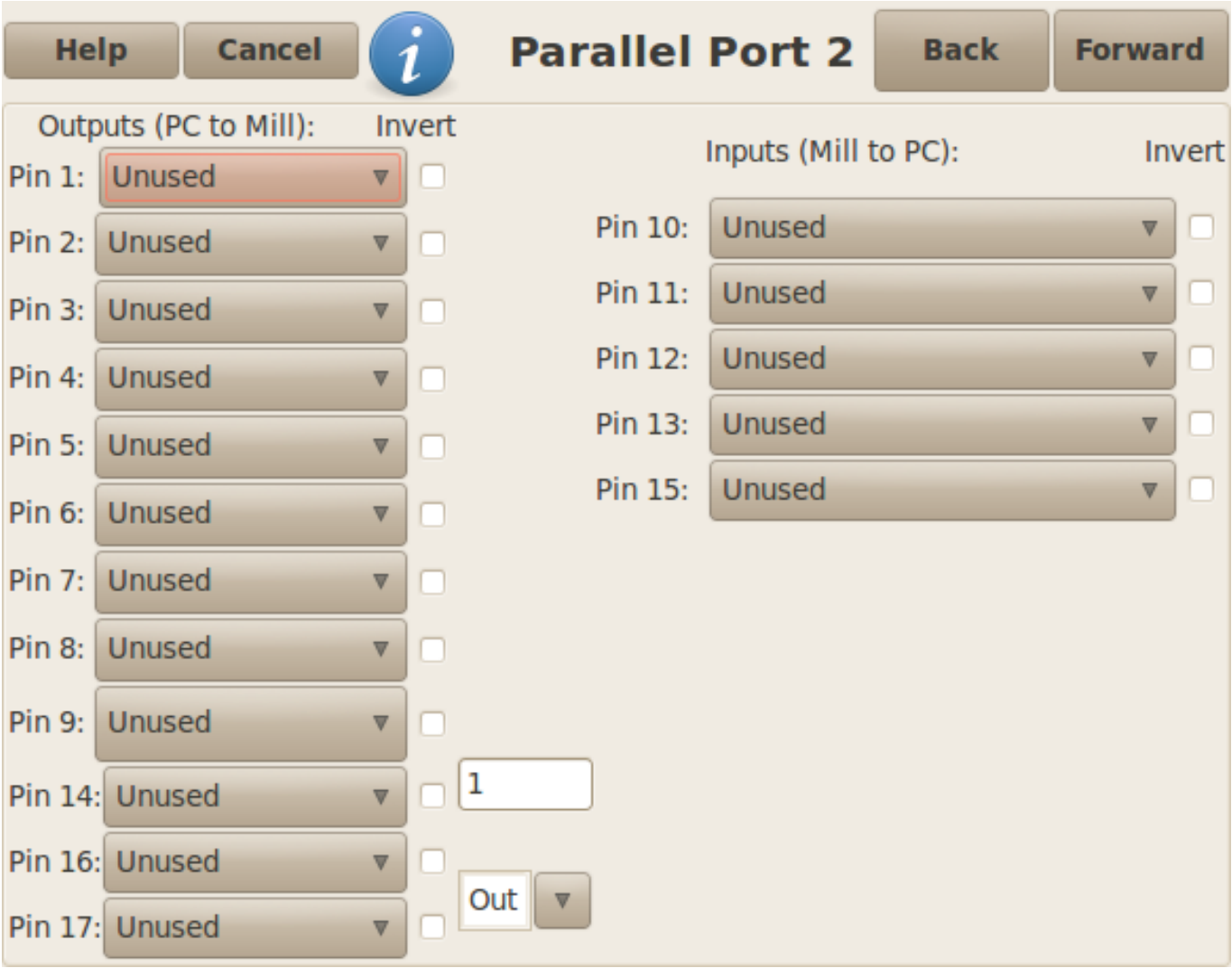



Figure 1.12: Parallel Port 2 Setup Page

The second Parallel port (if selected) can be configured and It's pins assigned on this page.
No step and direction signals can be selected.
You may select in or out to maximizes the number of input/output pins that are available.
You may specify the address as a hexidecimal (often 0x378) or as linux's default port number (probably 1).

### 1.6.7 Axis Configuration

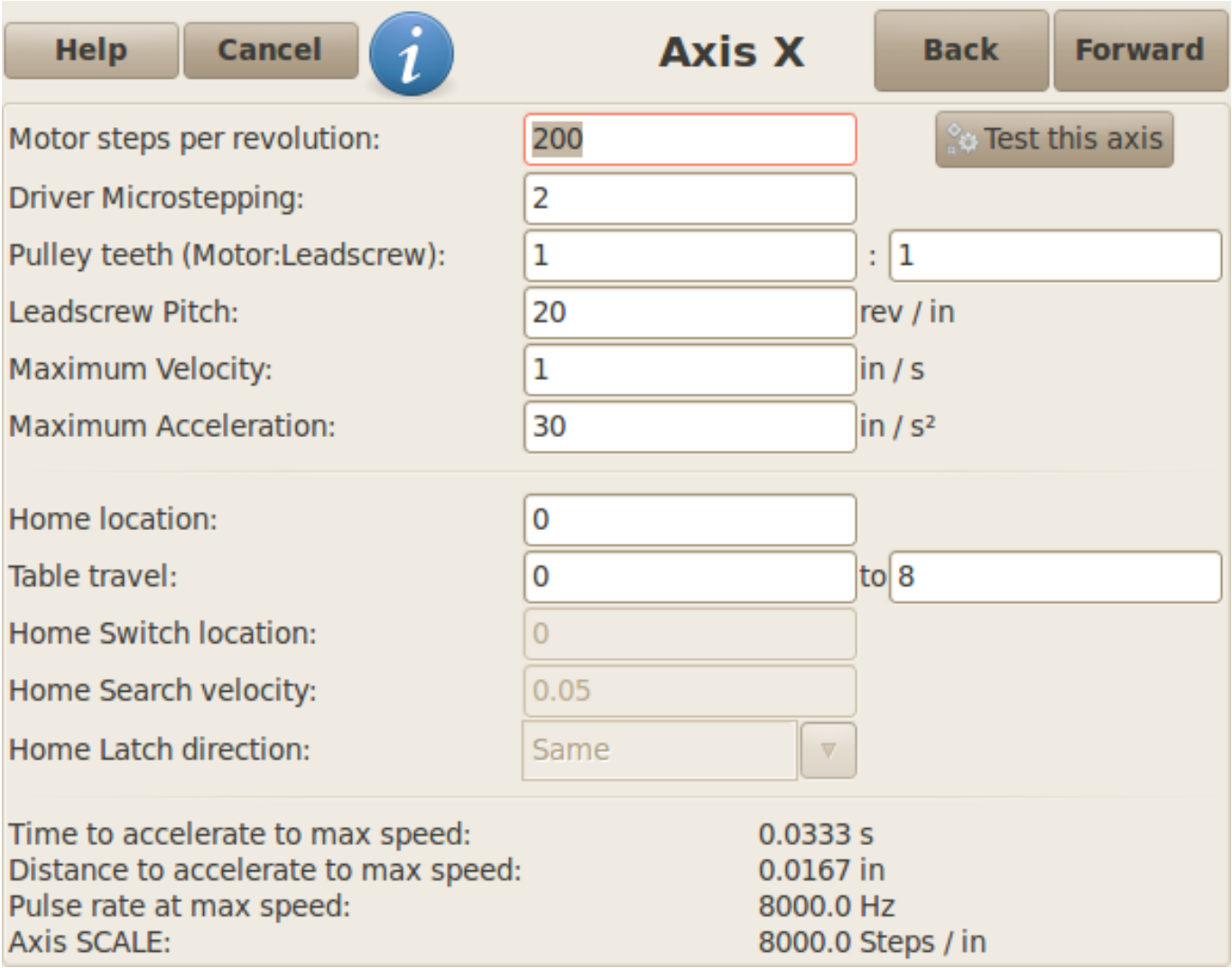



Figure 1.13: Axis Configuration Page

- *Motor Steps Per Revolution* - The number of full steps per motor revolution. If you know how many degrees per step the motor is (e.g., 1.8 degree), then divide 360 by the degrees per step to find the number of steps per motor revolution.
- *Driver Microstepping* - The amount of microstepping performed by the driver. Enter *2* for half-stepping.
- *Pulley Ratio* - If your machine has pulleys between the motor and leadscrew, enter the ratio here. If not, enter *1:1*.
- *Leadscrew Pitch* - Enter the pitch of the leadscrew here. If you chose *Inch* units, enter the number of threads per inch If you chose *mm* units, enter the number of millimeters per revolution (e.g., enter 2 for 2mm/rev). If the machine travels in the wrong direction, enter a negative number here instead of a positive number, or invert the direction pin for the axis.
- *Maximum Velocity* -Enter the maximum velocity for the axis in units per second.
- *Maximum Acceleration* - The correct values for these items can only be determined through experimentation. See Finding Maximum Velocity to set the speed and Finding Maximum Acceleration to set the acceleration.
- *Home Location* - The position the machine moves to after completing the homing procedure for this axis. For machines without home switches, this is the location the operator manually moves the machine to before pressing the Home button. If

you combine the home and limit switches you must move off of the switch to the home position or you will get a joint limit error.

- *Table Travel* - The range of travel for that axis based on the machine origin. The home location must be inside the *Table Travel* and not equal to one of the Table Travel values.
- *Home Switch Location* - The location at which the home switch trips or releases reletive to the machine origin. This item and the two below only appear when Home Switches were chosen in the Parallel Port Pinout. If you combine home and limit switches the home switch location can not be the same as the home position or you will get a joint limit error.
- *Home Search Velocity* - The velocity to use when searching for the home switch. If the switch is near the end of travel, this velocity must be chosen so that the axis can decelerate to a stop before hitting the end of travel. If the switch is only closed for a short range of travel (instead of being closed from its trip point to one end of travel), this velocity must be chosen so that the axis can decelerate to a stop before the switch opens again, and homing must always be started from the same side of the switch. If the machine moves the wrong direction at the beginning of the homing procedure, negate the value of *Home Search Velocity*.
- *Home Latch Direction* - Choose *Same* to have the axis back off the switch, then approach it again at a very low speed. The second time the switch closes, the home position is set. Choose *Opposite* to have the axis back off the switch and when the switch opens, the home position is set.
- *Time to accelerate to max speed* - Time to reach maximum speed calculated from *Max Acceleration* and *Max Velocity*.
- *Distance to accelerate to max speed* - Distance to reach maximum speed from a standstill.
- *Pulse rate at max speed* - Information computed based on the values entered above. The greatest *Pulse rate at max speed* determines the *BASE_PERIOD*. Values above 20000Hz may lead to slow response time or even lockups (the fastest usable pulse rate varies from computer to computer)
- *Axis SCALE* - The number that will be used in the ini file [SCALE] setting. This is how many steps per user unit.
- *Test this axis* - This will open a window to allow testing for each axis. This can be used after filling out all the information for this axis.

#### 1.6.7.1 Test This Axis

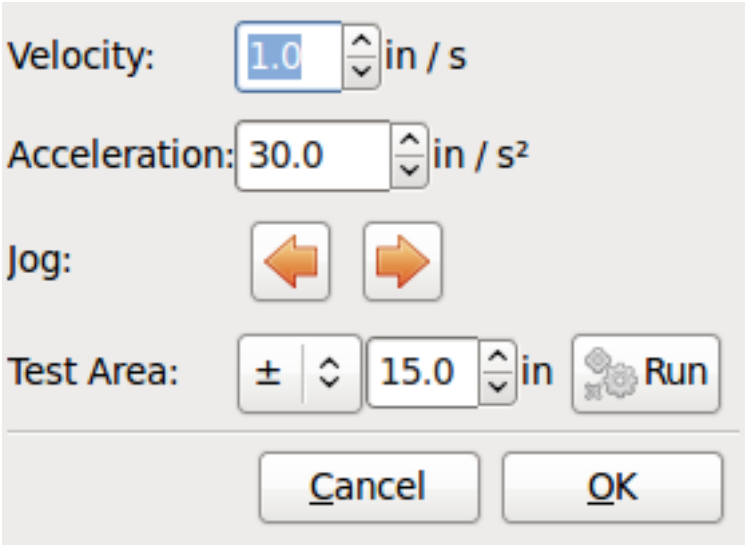



Figure 1.14: Test This Axis

Test this axis is a basic tester that only outputs step and direction signals to try different values for acceleration and velocity.

**Important**
In order to use test this axis you have to manually enable the axis if this is required. If your driver has a charge pump you will have to bypass it. Test this axis does not react to limit switch inputs. Use with caution.

**Finding Maximum Velocity**

Begin with a low Acceleration (for example, **`2 inches/s`**$^2$ or **`50 mm/s`**$^2$) and the velocity you hope to attain. Using the buttons provided, jog the axis to near the center of travel. Take care because with a low acceleration value, it can take a surprising distance for the axis to decelerate to a stop.

After gaging the amount of travel available, enter a safe distance in Test Area, keeping in mind that after a stall the motor may next start to move in an unexpected direction. Then click Run. The machine will begin to move back and forth along this axis. In this test, it is important that the combination of Acceleration and Test Area allow the machine to reach the selected Velocity and *cruise* for at least a short distance — the more distance, the better this test is. The formula **`d = 0.5 * v * v/a`** gives the minimum distance required to reach the specified velocity with the given acceleration. If it is convenient and safe to do so, push the table against the direction of motion to simulate cutting forces. If the machine stalls, reduce the speed and start the test again.

If the machine did not obviously stall, click the *Run* button off. The axis now returns to the position where it started. If the position is incorrect, then the axis stalled or lost steps during the test. Reduce Velocity and start the test again.

If the machine doesn't move, stalls, or loses steps, no matter how low you turn Velocity, verify the following:

- Correct step waveform timings
- Correct pinout, including *Invert* on step pins
- Correct, well-shielded cabling
- Physical problems with the motor, motor coupling, leadscrew, etc.

Once you have found a speed at which the axis does not stall or lose steps during this testing procedure, reduce it by 10% and use that as the axis *Maximum Velocity*.

**Finding Maximum Acceleration** With the Maximum Velocity you found in the previous step, enter the acceleration value to test. Using the same procedure as above, adjust the Acceleration value up or down as necessary. In this test, it is important that the combination of Acceleration and Test Area allow the machine to reach the selected Velocity. Once you have found a value at which the axis does not stall or lose steps during this testing procedure, reduce it by 10% and use that as the axis Maximum Acceleration.

### 1.6.8 Spindle Configuration

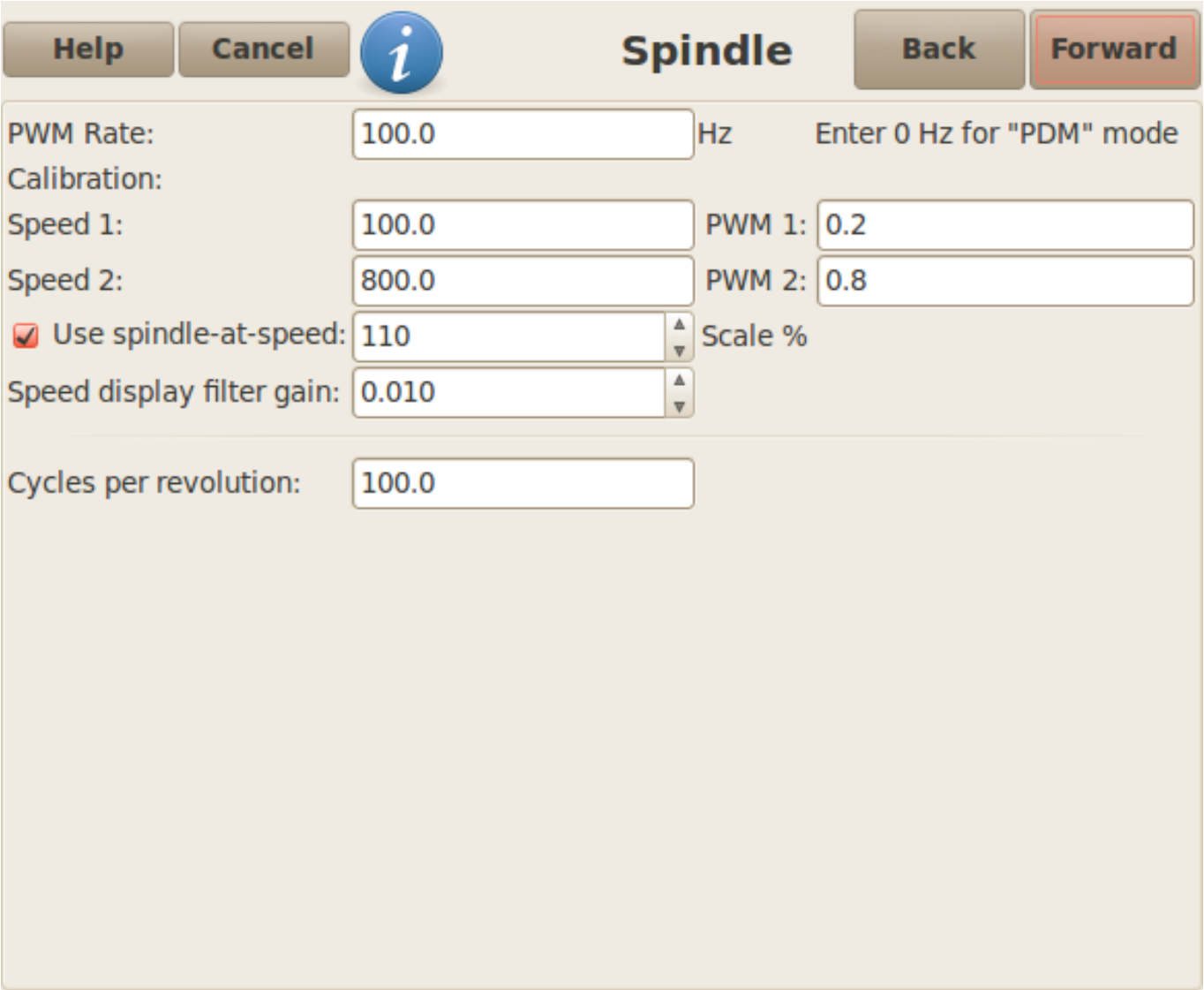



Figure 1.15: Spindle Configuration Page

This page only appears when *Spindle PWM* is chosen in the *Parallel Port Pinout* page for one of the outputs.

#### 1.6.8.1 Spindle Speed Control

If *Spindle PWM* appears on the pinout, the following information should be entered:

- *PWM Rate* - The *carrier frequency* of the PWM signal to the spindle. Enter *0* for PDM mode, which is useful for generating an analog control voltage. Refer to the documentation for your spindle controller for the appropriate value.
- *Speed 1 and 2, PWM 1 and 2* - The generated configuration file uses a simple linear relationship to determine the PWM value for a given RPM value. If the values are not known, they can be determined. For more information see Determining Spindle Calibration.

#### 1.6.8.2 Spindle-synchronized motion

When the appropriate signals from a spindle encoder are connected to LinuxCNC via HAL, LinuxCNC supports lathe threading. These signals are:

- *Spindle Index* - Is a pulse that occurs once per revolution of the spindle.
- *Spindle Phase A* - This is a pulse that occurs in multiple equally-spaced locations as the spindle turns.
- *Spindle Phase B (optional)* - This is a second pulse that occurs, but with an offset from Spindle Phase A. The advantages to using both A and B are direction sensing, increased noise immunity, and increased resolution.

If *Spindle Phase A* and *Spindle Index* appear on the pinout, the following information should be entered:

- *Use Spindle-At-Speed* - With encoder feedback one can choose to have linuxcnc wait for the spindle to reach the commanded speed before feed moves. Select this option and set the *close enough* scale.
- *Speed Display Filter Gain* - Setting for adjusting the stability of the visual spindle speed display.
- *Cycles per revolution* - The number of cycles of the *Spindle A* signal during one revolution of the spindle. This option is only enabled when an input has been set to *Spindle Phase A*
- *Maximum speed in thread* - The maximum spindle speed used in threading. For a high spindle RPM or a spindle encoder with high resolution, a low value of *BASE_PERIOD* is required.

#### 1.6.8.3 Determining Spindle Calibration

Enter the following values in the Spindle Configuration page:

| | | | |
|---|---|---|---|
| Speed 1: | 0 | PWM 1: | 0 |
| Speed 2: | 1000 | PWM 2: | 1 |

Finish the remaining steps of the configuration process, then launch LinuxCNC with your configuration. Turn the machine on and select the MDI tab. Start the spindle turning by entering: *M3 S100*. Change the spindle speed by entering a different S-number: *S800*. Valid numbers (at this point) range from 1 to 1000.

For two different S-numbers, measure the actual spindle speed in RPM. Record the S-numbers and actual spindle speeds. Run Stepconf again. For *Speed* enter the measured speed, and for *PWM* enter the S-number divided by 1000.

Because most spindle drivers are somewhat nonlinear in their response curves, it is best to:

- Make sure the two calibration speeds are not too close together in RPM
- Make sure the two calibration speeds are in the range of speeds you will typically use while milling

For instance, if your spindle will go from 0 RPM to 8000 RPM, but you generally use speeds from 400 RPM (10%) to 4000 RPM (100%), then find the PWM values that give 1600 RPM (40%) and 2800 RPM (70%).

### 1.6.9 Options

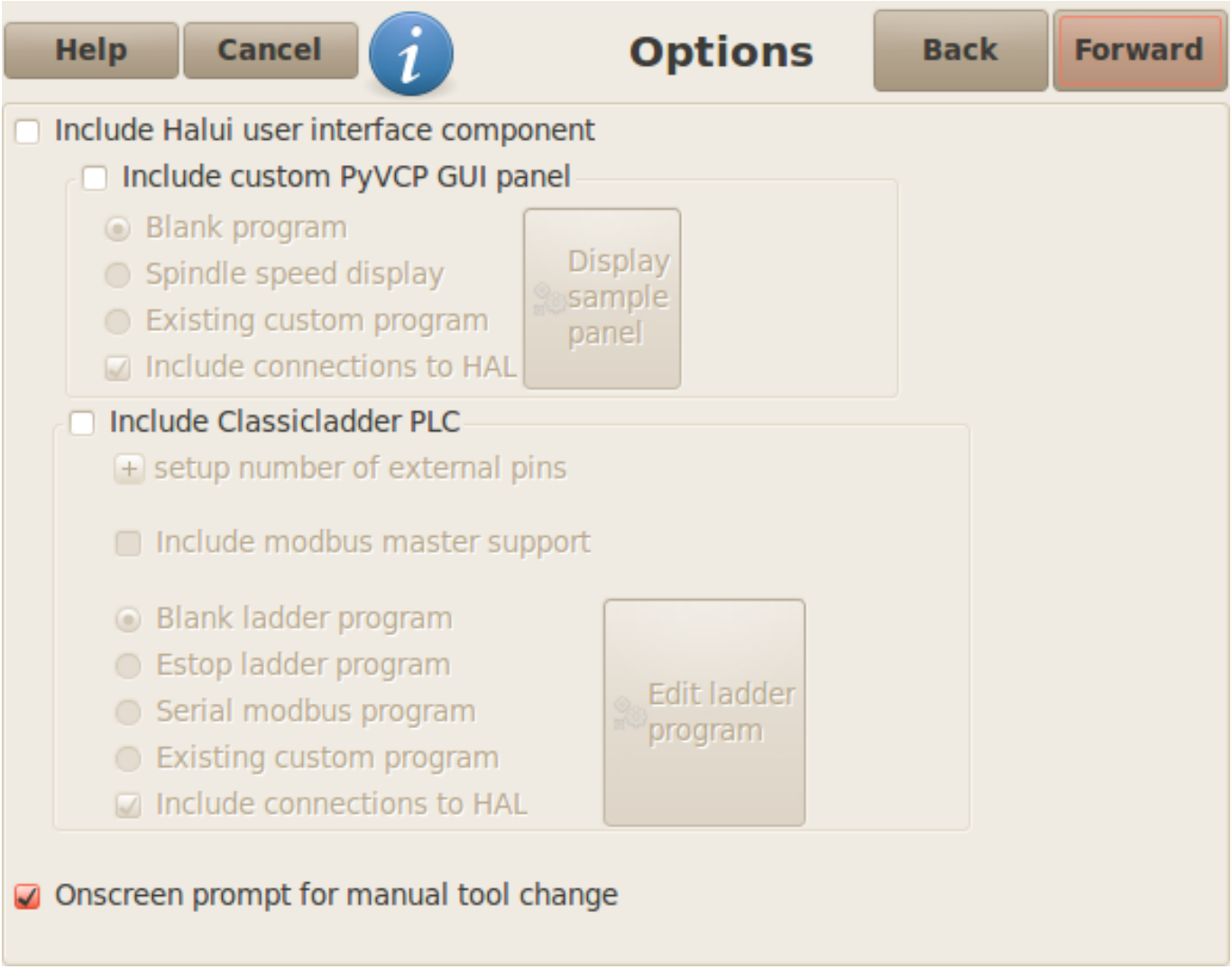



Figure 1.16: Advanced Configuration

- *Include Halui* - This will add the Halui user interface component. See the HALUI Chapter for more information on.
- *Include pyVCP* - This option adds the pyVCP panel base file or a sample file to work on. See the PyVCP Chapter for more information.
- *Include ClassicLadder PLC* - This option will add the ClassicLadder PLC (Programmable Logic Controller). See the Classicladder Chapter for more information.
- *Onscreen Prompt For Tool Change* - If this box is checked, LinuxCNC will pause and prompt you to change the tool when *M6* is encountered. This feature is usually only useful if you have presettable tools.

### 1.6.10 Machine Configuration Complete

Click *Apply* to write the configuration files. Later, you can re-run this program and tweak the settings you entered before.

### 1.6.11 Axis Travel and Home

For each axis, there is a limited range of travel. The physical end of travel is called the *hard stop*.

Before the *hard stop* there is a *limit switch*. If the limit switch is encountered during normal operation, LinuxCNC shuts down the motor amplifier. The distance between the *hard stop* and *limit switch* must be long enough to allow an unpowered motor to coast to a stop.

Before the *limit switch* there is a *soft limit*. This is a limit enforced in software after homing. If a MDI command or g code program would pass the soft limit, it is not executed. If a jog would pass the soft limit, it is terminated at the soft limit.

The *home switch* can be placed anywhere within the travel (between hard stops). As long as external hardware does not deactivate the motor amplifiers when the limit switch is reached, one of the limit switches can be used as a home switch.

The *zero position* is the location on the axis that is 0 in the machine coordinate system. Usually the *zero position* will be within the *soft limits*. On lathes, constant surface speed mode requires that machine *X=0* correspond to the center of spindle rotation when no tool offset is in effect.

The *home position* is the location within travel that the axis will be moved to at the end of the homing sequence. This value must be within the *soft limits*. In particular, the *home position* should never be exactly equal to a *soft limit*.

#### 1.6.11.1 Operating without Limit Switches

A machine can be operated without limit switches. In this case, only the soft limits stop the machine from reaching the hard stop. Soft limits only operate after the machine has been homed.

#### 1.6.11.2 Operating without Home Switches

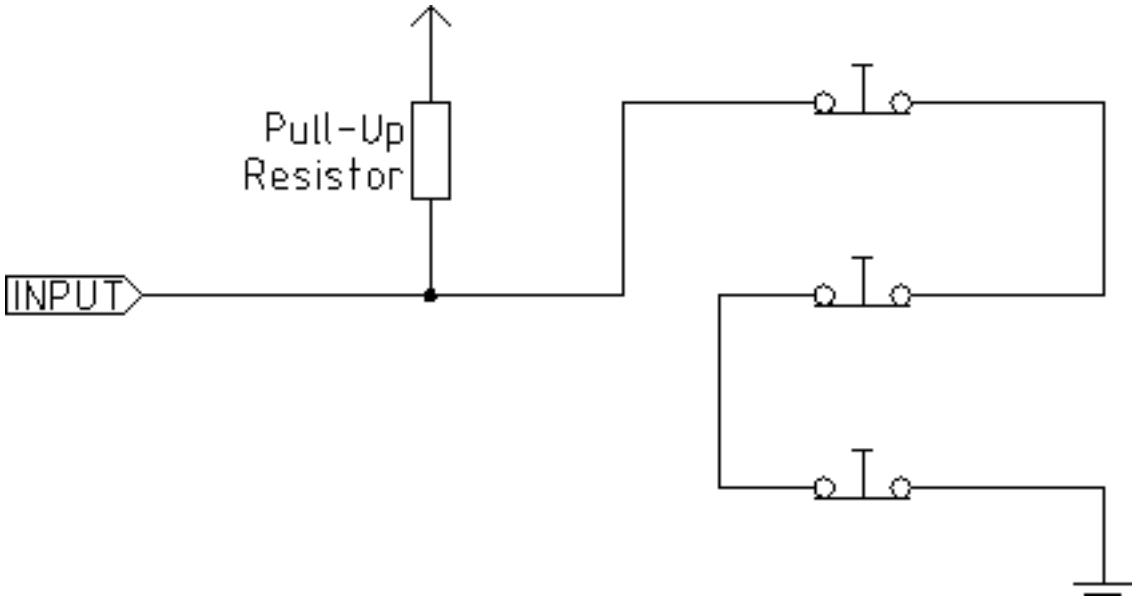
A machine can be operated without home switches. If the machine has limit switches, but no home switches, it is best to use a limit switch as the home switch (e.g., choose *Minimum Limit + Home X* in the pinout). If the machine has no switches at all, or the limit switches cannot be used as home switches for another reason, then the machine must be homed *by eye* or by using match marks. Homing by eye is not as repeatable as homing to switches, but it still allows the soft limits to be useful.

#### 1.6.11.3 Home and Limit Switch wiring options

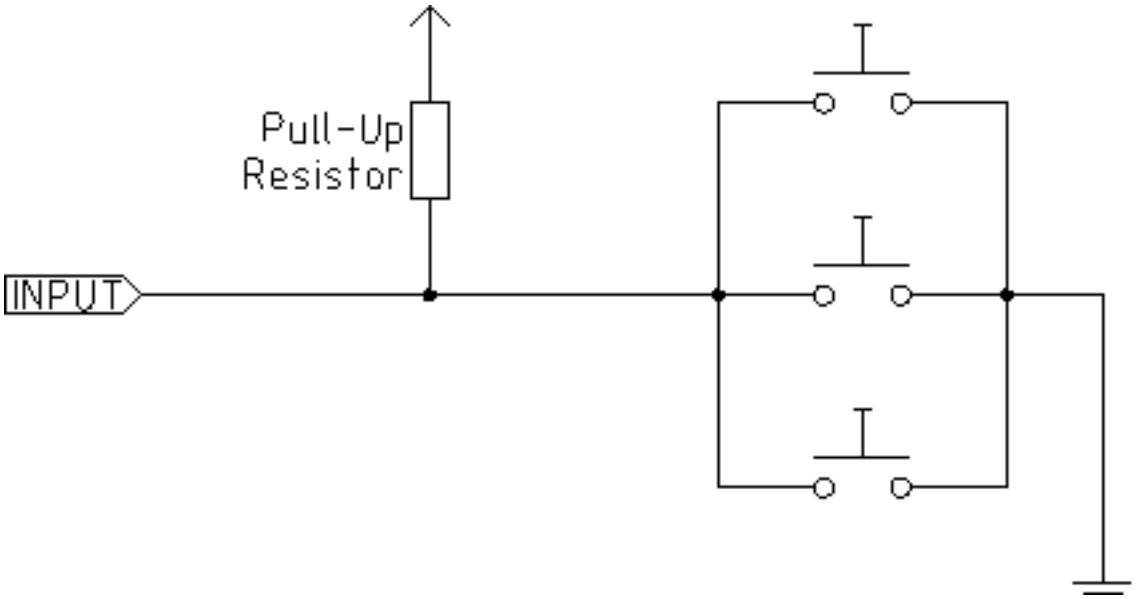
The ideal wiring for external switches would be one input per switch. However, the PC parallel port only offers a total of 5 inputs, while there are as many as 9 switches on a 3-axis machine. Instead, multiple switches are wired together in various ways so that a smaller number of inputs are required.

The figures below show the general idea of wiring multiple switches to a single input pin. In each case, when one switch is actuated, the value seen on INPUT goes from logic HIGH to LOW. However, LinuxCNC expects a TRUE value when a switch is closed, so the corresponding *Invert* box must be checked on the pinout configuration page. The pull up resistor show in the diagrams pulls the input high until the connection to ground is made and then the input goes low. Otherwise the input might float between on and off when the circuit is open. Typically for a parallel port you might use 47k.

**Normally Closed Switches** Wiring N/C switches in series (simplified diagram)




**Normally Open Switches** Wiring N/O switches in parallel (simplified diagram)




The following combinations of switches are permitted in Stepconf:

- Combine home switches for all axes
- Combine limit switches for all axes
- Combine both limit switches for one axis
- Combine both limit switches and the home switch for one axis
- Combine one limit switch and the home switch for one axis

## 1.7 Mesa Configuration Wizard

PNCconf is made to help build configurations that utilize specific Mesa *Anything I/O* products.

It can configure closed loop servo systems or hardware stepper systems. It uses a similar *wizard* approach as Stepconf (used for software stepping, parallel port driven systems).

PNCconf is still in a development stage (Beta) so there are some bugs and lacking features. Please report bugs and suggestions to the LinuxCNC forum page or mail-list.

There are two trains of thought when using PNCconf:

One is to use PNCconf to always configure your system - if you decide to change options, reload PNCconf and allow it to configure the new options. This will work well if your machine is fairly standard and you can use custom files to add non standard features. PNCconf tries to work with you in this regard.

The other is to use PNCconf to build a config that is close to what you want and then hand edit everything to tailor it to your needs. This would be the choice if you need extensive modifications beyond PNCconf's scope or just want to tinker with / learn about LinuxCNC

You navigate the wizard pages with the forward, back, and cancel buttons there is also a help button that gives some help information about the pages, diagrams and an output page.

**Tip**
PNCconf's help page should have the most up to date info and has additional details.

### 1.7.1 Step by Step Instructions

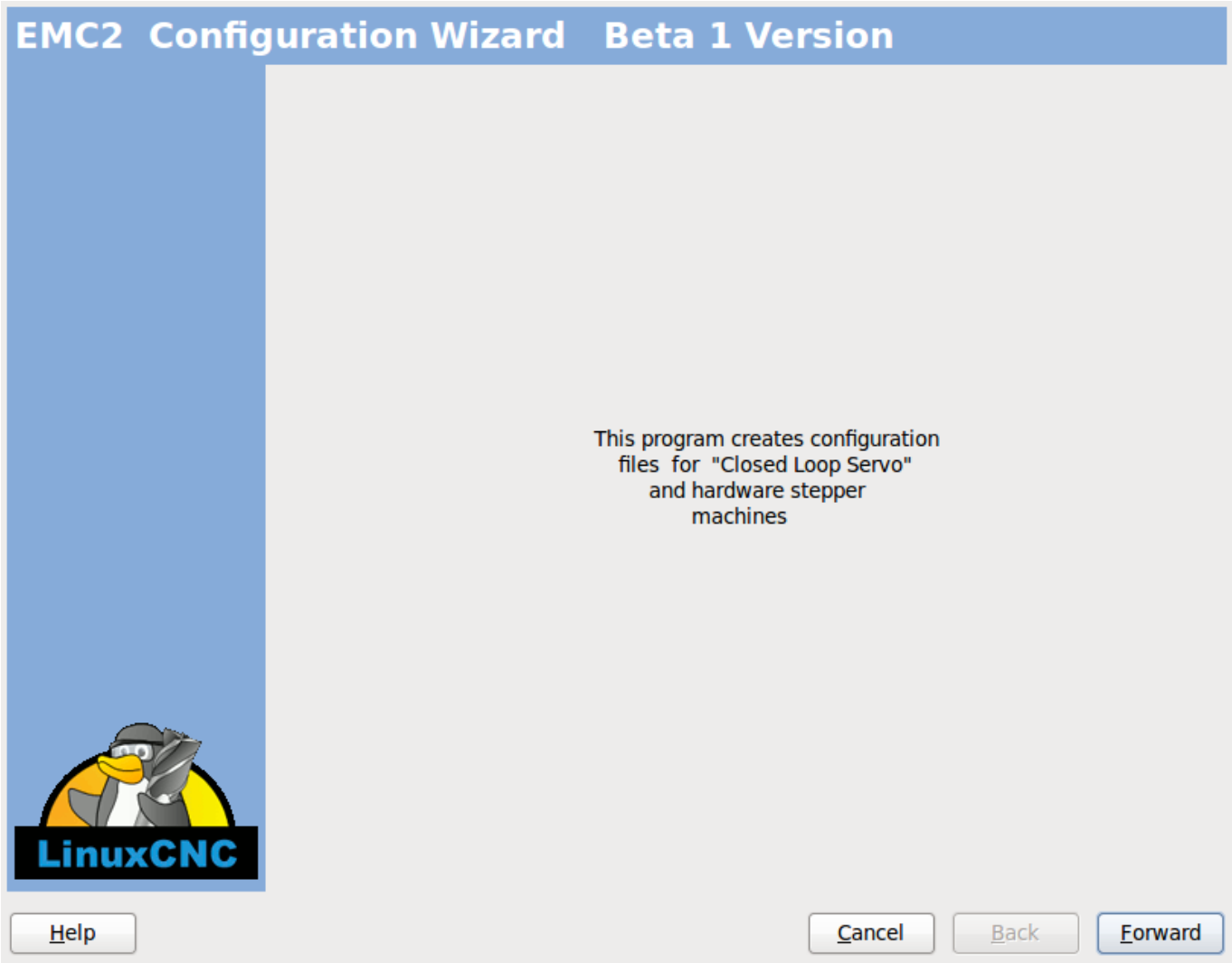



Figure 1.17: PnCConf Splash

### 1.7.2 Create or Edit

This allows you to select a previously saved configuration or create a new one. If you pick *Modify a configuration* and then press next a file selection box will show. Pncconf preselects your last saved file. Choose the the config you wish to edit. If you made any changes to the main hal or ini files **Pncconf will over write** those files and those changes will be lost. Some files will not be over written and Pncconf places a note in those files. It also allows you to select desktop shortcut / launcher options. A desktop shortcut will place a folder icon on the desktop that points to your new configuration files. Otherwise you would have to look in your home folder under linuxcnc/configs.

A Desktop launcher will add an icon to the desktop for starting your config directly. You can also launch it from the main menu by using the Configuration Selector *LinuxCNC* found in CNC menu and selecting your config name.

### 1.7.3 Basic Machine Information

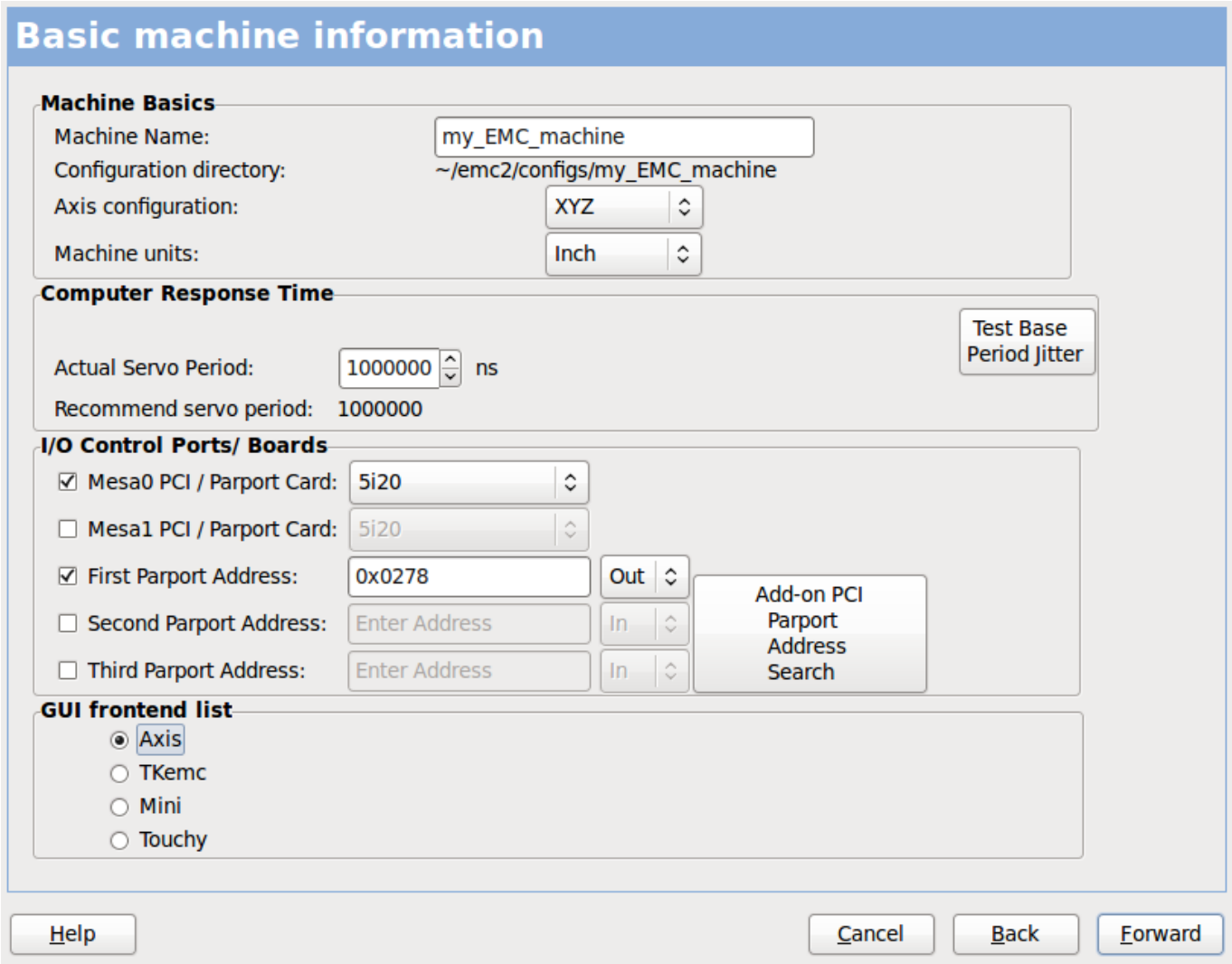



Figure 1.18: PnCConf Basic

**Machine Basics**

If you use a name with spaces PNCconf will replace the spaces with underscore (as a loose rule Linux doesn't like spaces in names) Pick an axis configuration - this selects what type of machine you are building and what axes are available. The Machine units selector allows data entry of metric or imperial units in the following pages.

---

**Tip**

Defaults are not converted when using metric so make sure they are sane values!

---

**Computer Response Time**

The servo period sets the heart beat of the system. Latency refers to the amount of time the computer can be longer then that period. Just like a railroad, LinuxCNC requires everything on a very tight and consistent time line or bad things happen. LinuxCNC requires and uses a *real time* operating system, which just means it has a low latency ( lateness ) response time when LinuxCNC requires its calculations and when doing LinuxCNCs calculations it cannot be interrupted by lower priority requests (such as user input to screen buttons or drawing etc).

Testing the latency is very important and a key thing to check early. Luckily by using the Mesa card to do the work that requires the fastest response time (encoder counting and PWM generation) we can endure a lot more latency then if we used the parallel port for these things. The standard test in LinuxCNC is checking the BASE period latency (even though we are not using a base period). If you press the *test base period jitter* button, this launches the latency test window ( you can also load this directly from the applications/cnc panel ). The test mentions to run it for a few minutes but the longer the better. consider 15 minutes a bare minimum and overnight even better. At this time use the computer to load things, use the net, use USB etc we want to know the worst case latency and to find out if any particular activity hurts our latency. We need to look at base period jitter. Anything under 20000 is excellent - you could even do fast software stepping with the machine 20000 - 50000 is still good for software stepping and fine for us. 50000 - 100000 is really not that great but could still be used with hardware cards doing the fast response stuff. So anything under 100000 is usable to us. If the latency is disappointing or you get a bad hiccup periodically you may still be able to improve it.

**Tip**
There is a user compiled list of equipment and the latency obtained on the LinuxCNC wiki : http://wiki.linuxcnc.org/cgi-bin/-wiki.pl?Latency-Test Please consider adding your info to the list. Also on that page are links to info about fixing some latency problems.

Now we are happy with the latency and must pick a servo period. In most cases a servo period of 1000000 ns is fine ( that gives a 1 kHz servo calculation rate - 1000 calculations a second) if you are building a closed loop servo system that controls torque (current) rather then velocity (voltage) a faster rate would be better - something like 200000 (5 kHz calculation rate). The problem with lowering the servo rate is that it leaves less time available for the computer to do other things besides LinuxCNC's calculations. Typically the display (GUI) becomes less responsive. You must decide on a balance. Keep in mind that if you tune your closed loop servo system then change the servo period you probably will need to tune them again.

**I/O Control Ports/Boards**
PNCconf is capable of configuring machines that have up to two Mesa boards and three parallel ports. Parallel ports can only be used for simple low speed (servo rate) I/O.

**Mesa**
You must choose at least one Mesa board as PNCconf will not configure the parallel ports to count encoders or output step or PWM signals. The mesa cards available in the selection box are based on what PNCconf finds for firmware on the systems. There are options to add custom firmware and/or *blacklist* (ignore) some firmware or boards using a preference file. If no firmware is found PNCconf will show a warning and use internal sample firmware - no testing will be possible. One point to note is that if you choose two PCI Mesa cards there currently is no way to predict which card is 0 and which is 1 - you must test - moving the cards could change their order. If you configure with two cards both cards must be installed for tests to function.

**Parallel Port**
Up to 3 parallel ports (referred to as parports) can be used as simple I/O. You must set the address of the parport. You can either enter the Linux parallel port numbering system (0,1,or 2) or enter the actual address. The address for an on board parport is often 0x0278 or 0x0378 (written in hexadecimal) but can be found in the BIOS page. The BIOS page is found when you first start your computer you must press a key to enter it (such as F2). On the BIOS page you can find the parallel port address and set the mode such as SPP, EPP, etc on some computers this info is displayed for a few seconds during start up. For PCI parallel port cards the address can be found by pressing the *parport address search* button. This pops up the help output page with a list of all the PCI devices that can be found. In there should be a reference to a parallel port device with a list of addresses. One of those addresses should work. Not all PCI parallel ports work properly. Either type can be selected as *in* (maximum amount of input pins) or *out* (maximum amount of output pins)

**GUI Front-end list**
This specifies the graphical display screens LinuxCNC will use. Each one has different option.

AXIS

- fully supports lathes.
- is the most developed and used front-end

- is designed to be used with mouse and keyboard
- is tkinter based so integrates PYVCP (python based virtual control panels) naturally.
- has a 3D graphical window.
- allows VCP integrated on the side or in center tab

TOUCHY

- Touchy was designed to be used with a touchscreen, some minimal physical switches and a MPG wheel.
- requires cycle-start, abort, and single-step signals and buttons
- It also requires shared axis MPG jogging to be selected.
- is GTK based so integrates GLADE VCP (virtual control panels) naturally.
- allows VCP panels integrated in the center Tab
- has no graphical window
- look can be changed with custom themes

MINI

- standard on OEM Sherline machines
- does not use Estop
- no VCP integration

TkLinuxCNC

- hi contrast bright blue screen
- separate graphics window
- no VCP integration

### 1.7.4 External Configuration

This page allows you to select external controls such as for jogging or overrides.

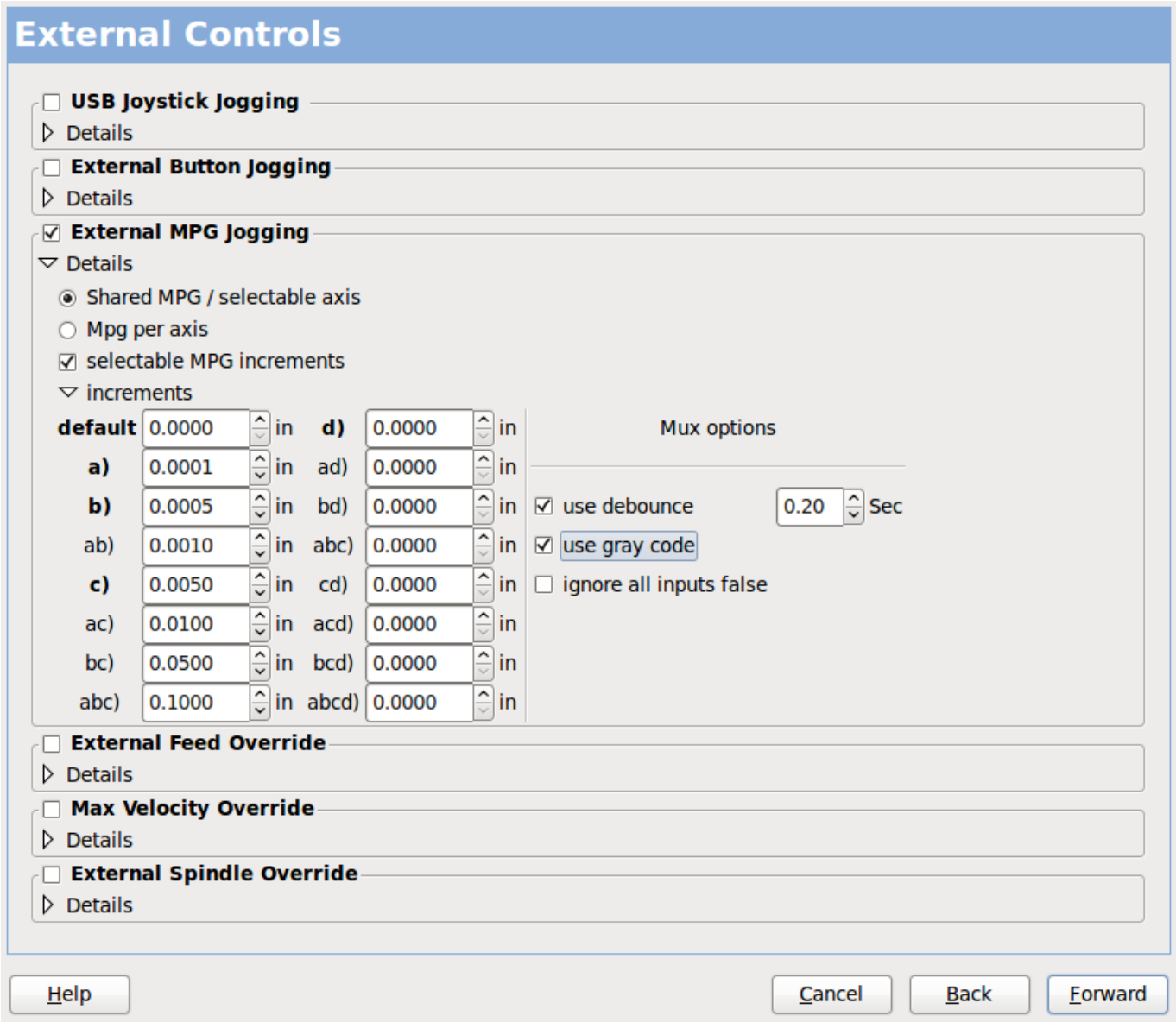



Figure 1.19: GUI External

If you select a Joystick for jogging, You will need it always connected for LinuxCNC to load. To use the analog sticks for useful jogging you probably need to add some custom HAL code. MPG jogging requires a pulse generator connected to a MESA encoder counter. Override controls can either use a pulse generator (MPG) or switches (such as a rotary dial). External buttons might be used with a switch based OEM joystick.

**Joystick jogging**

Requires a custom *device rule* to be installed in the system. This is a file that LinuxCNC uses to connect to LINUX's device list. PNCconf will help to make this file.

*Search for device rule* will search the system for rules, you can use this to find the name of devices you have already built with PNCconf.

*Add a device rule* will allow you to configure a new device by following the prompts. You will need your device available.

*test device* allows you to load a device, see its pin names and check its functions with halmeter.

joystick jogging uses HALUI and hal_input components.

**External buttons**
allows jogging the axis with simple buttons at a specified jog rate. Probably best for rapid jogging.

**MPG Jogging**
Allows you to use a Manual Pulse Generator to jog the machine's axis.

MPG's are often found on commercial grade machines. They output quadrature pulses that can be counted with a MESA encoder counter. PNCconf allows for an MPG per axis or one MPG shared with all axis. It allows for selection of jog speeds using switches or a single speed.

The selectable increments option uses the mux16 component. This component has options such as debounce and gray code to help filter the raw switch input.

**Overrides**
PNCconf allows overrides of feed rates and/or spindle speed using a pulse generator (MPG) or switches (eg. rotary).

### 1.7.5 GUI Configuration

Here you can set defaults for the display screens, add virtual control panels (VCP), and set some LinuxCNC options..

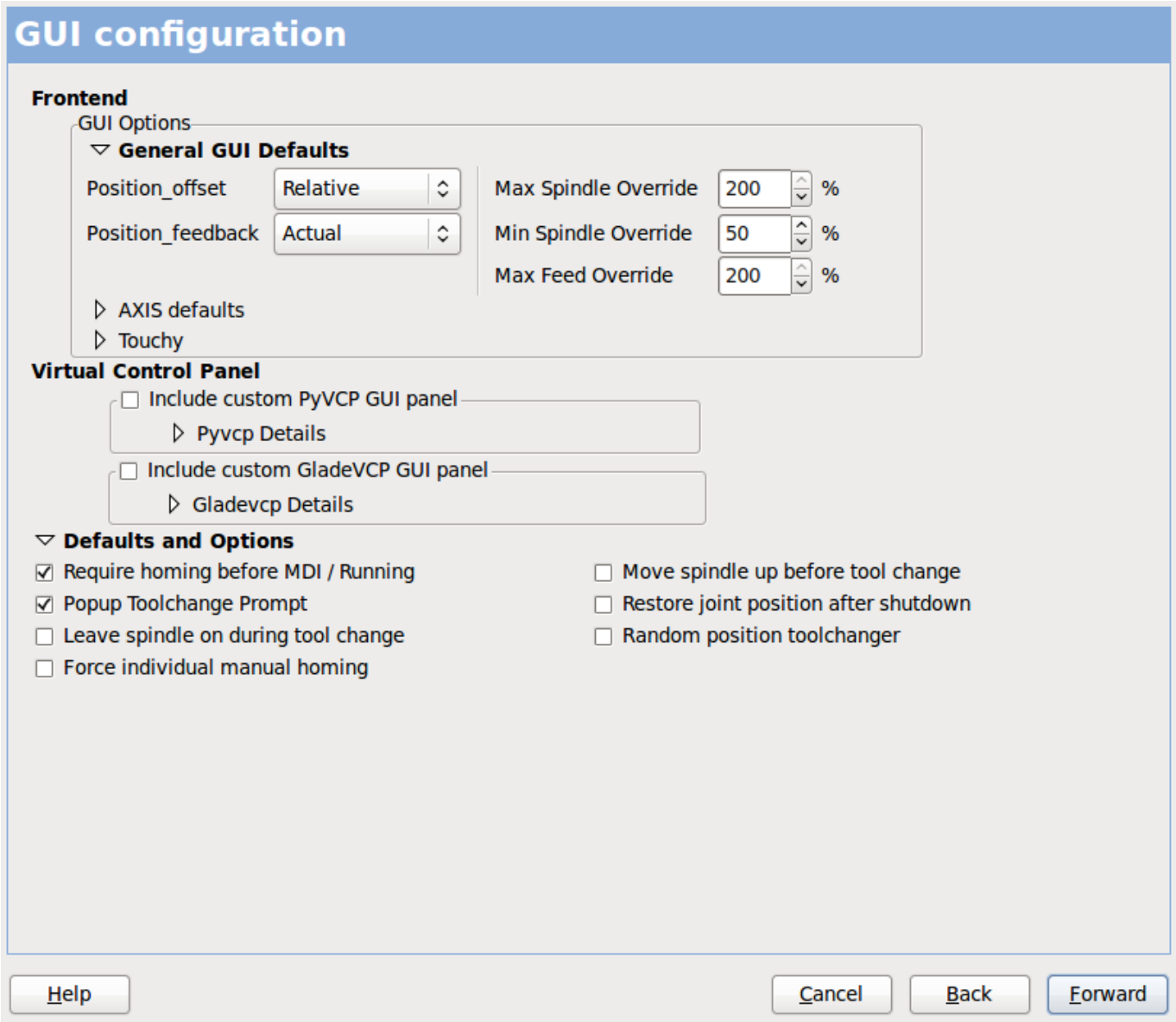



Figure 1.20: GUI Configuration

**Front-end GUI Options**

The default options allows general defaults to be chosen for any display screen.

AXIS defaults are options specific to AXIS. If you choose size , position or force maximize options then PNCconf will ask if it's alright to overwrite a preference file (.axisrc). Unless you have manually added commands to this file it is fine to allow it. Position and force max can be used to move AXIS to a second monitor if the system is capable.

Touchy defaults are options specific to Touchy. Most of Touchy's options can be changed while Touchy is running using the preference page. Touchy uses GTK to draw its screen, and GTK supports themes. Themes controls the basic look and feel of a program. You can download themes from the net or edit them yourself. There are a list of the current themes on the computer that you can pick from. To help some of the text to stand out PNCconf allows you to override the Themes's defaults. The position and force max options can be used to move Touchy to a second monitor if the system is capable.

**VCP options**

Virtual Control Panels allow one to add custom controls and displays to the screen. AXIS and Touchy can integrate these controls inside the screen in designated positions. There are two kinds of VCPs - pyVCP which uses *Tkinter* to draw the screen and GLADE VCP that uses *GTK* to draw the screen.

**PyVCP**

PyVCPs screen XML file can only be hand built. PyVCPs fit naturally in with AXIS as they both use TKinter.

HAL pins are created for the user to connect to inside their custom HAL file. There is a sample spindle display panel for the user to use as-is or build on. You may select a blank file that you can later add your controls *widgets* to or select a spindle display sample that will display spindle speed and indicate if the spindle is at requested speed.

PNCconf will connect the proper spindle display HAL pins for you. If you are using AXIS then the panel will be integrated on the right side. If not using AXIS then the panel will be separate *stand-alone* from the front-end screen.

You can use the geometry options to size and move the panel, for instance to move it to a second screen if the system is capable. If you press the *Display sample panel* button the size and placement options will be honored.

**GLADE VCP**

GLADE VCPs fit naturally inside of TOUCHY screen as they both use GTK to draw them, but by changing GLADE VCP's theme it can be made to blend pretty well in AXIS. (try Redmond)

It uses a graphical editor to build its XML files. HAL pins are created for the user to connect to, inside of their custom HAL file.

GLADE VCP also allows much more sophisticated (and complicated) programming interaction, which PNCconf currently doesn't leverage. (see GLADE VCP in the manual)

PNCconf has sample panels for the user to use as-is or build on. With GLADE VCP PNCconf will allow you to select different options on your sample display.

Under *sample options* select which ones you would like. The zero buttons use HALUI commands which you could edit later in the HALUI section.

Auto Z touch-off also requires the classic ladder touch-off program and a probe input selected. It requires a conductive touch-off plate and a grounded conductive tool. For an idea on how it works see:

http://wiki.linuxcnc.org/cgi-bin/wiki.pl?ClassicLadderExamples#Single_button_probe_touchoff

Under *Display Options*, size, position, and force max can be used on a *stand-alone* panel for such things as placing the screen on a second monitor if the system is capable.

You can select a GTK theme which sets the basic look and feel of the panel. You Usually want this to match the front-end screen. These options will be used if you press the *Display sample button*. With GLADE VCP depending on the front-end screen, you can select where the panel will display.

You can force it to be stand-alone or with AXIS it can be in the center or on the right side, with Touchy it can be in the center.

**Defaults and Options**

- Require homing before MDI / Running
  - If you want to be able to move the machine before homing uncheck this checkbox.
- Popup Tool Prompt
  - Choose between an on screen prompt for tool changes or export standard signal names for a User supplied custom tool changer Hal file
- Leave spindle on during tool change:
  - Used for lathes
- Force individual manual homing
- Move spindle up before tool change
- Restore joint position after shutdown
  - Used for non-trivial kinematics machines
- Random position tool changers
  - Used for tool changers that do not return the tool to the same pocket. You will need to add custom HAL code to support tool changers.

### 1.7.6 Mesa Configuration

The Mesa configuration pages allow one to utilize different firmwares. On the basic page you selected a Mesa card here you pick the available firmware and select what and how many components are available.

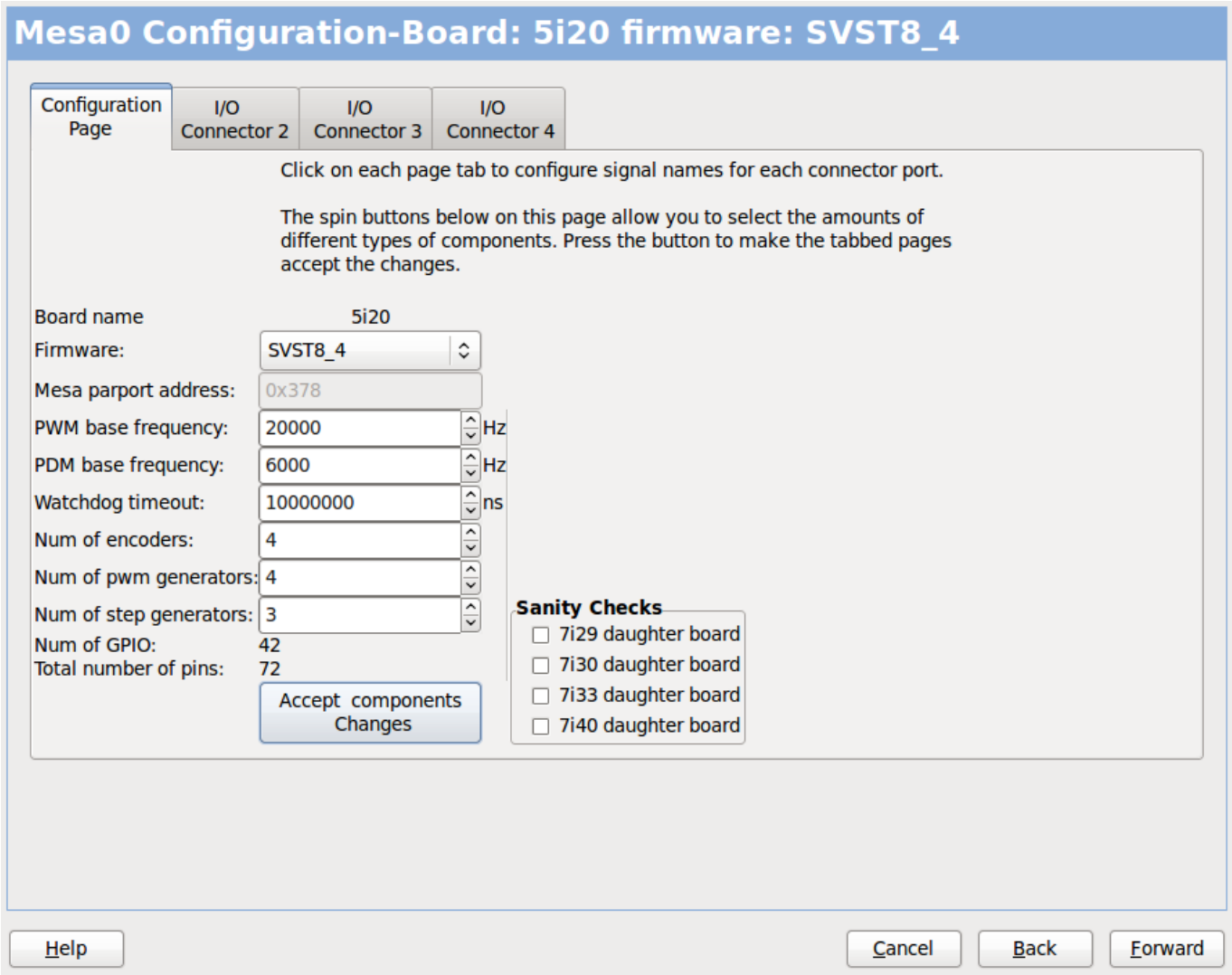



Figure 1.21: Mesa Configuration

Parport address is used only with Mesa parport card, the 7i43. An on board parallel port usually uses 0x278 or 0x378 though you should be able to find the address from the BIOS page. The 7i43 requires the parallel port to use the EPP mode, again set in the BIOS page. If using a PCI parallel port the address can be searched for by using the search button on the basic page.

**Note**
Many PCI cards do not support the EPP protocol properly.

PDM PWM and 3PWM base frequency sets the balance between ripple and linearity. If using Mesa daughter boards the docs for the board should give recommendations

**Important**
It's important to follow these to avoid damage and get the best performance.

```
The 7i33 requires PDM and a PDM base frequency of 6 mHz
The 7i29 requires PWM and a PWM base frequency of 20 Khz
The 7i30 requires PWM and a PWM base frequency of 20 Khz
The 7i40 requires PWM and a PWM base frequency of 50 Khz
The 7i48 requires UDM and a PWM base frequency of 24 Khz
```

Watchdog time out is used to set how long the MESA board will wait before killing outputs if communication is interrupted from the computer. Please remember Mesa uses *active low* outputs meaning that when the output pin is on, it is low (approx 0 volts) and if it's off the output in high (approx 5 volts) make sure your equipment is safe when in the off (watchdog bitten) state.

You may choose the number of available components by deselecting unused ones. Not all component types are available with all firmware.

Choosing less then the maximum number of components allows one to gain more GPIO pins. If using daughter boards keep in mind you must not deselect pins that the card uses. For instance some firmware supports two 7i33 cards, If you only have one you may deselect enough components to utilize the connector that supported the second 7i33. Components are deselected numerically by the highest number first then down with out skipping a number. If by doing this the components are not where you want them then you must use a different firmware. The firmware dictates where, what and the max amounts of the components. Custom firmware is possible, ask nicely when contacting the LinuxCNC developers and Mesa. Using custom firmware in PNCconf requires special procedures and is not always possible - Though I try to make PNCconf as flexible as possible.

After choosing all these options press the *Accept Component Changes* button and PNCconf will update the I/O setup pages. Only I/O tabs will be shown for available connectors, depending on the Mesa board.

### 1.7.7 Mesa I/O Setup

The tabs are used to configure the input and output pins of the Mesa boards. PNCconf allows one to create custom signal names for use in custom HAL files.

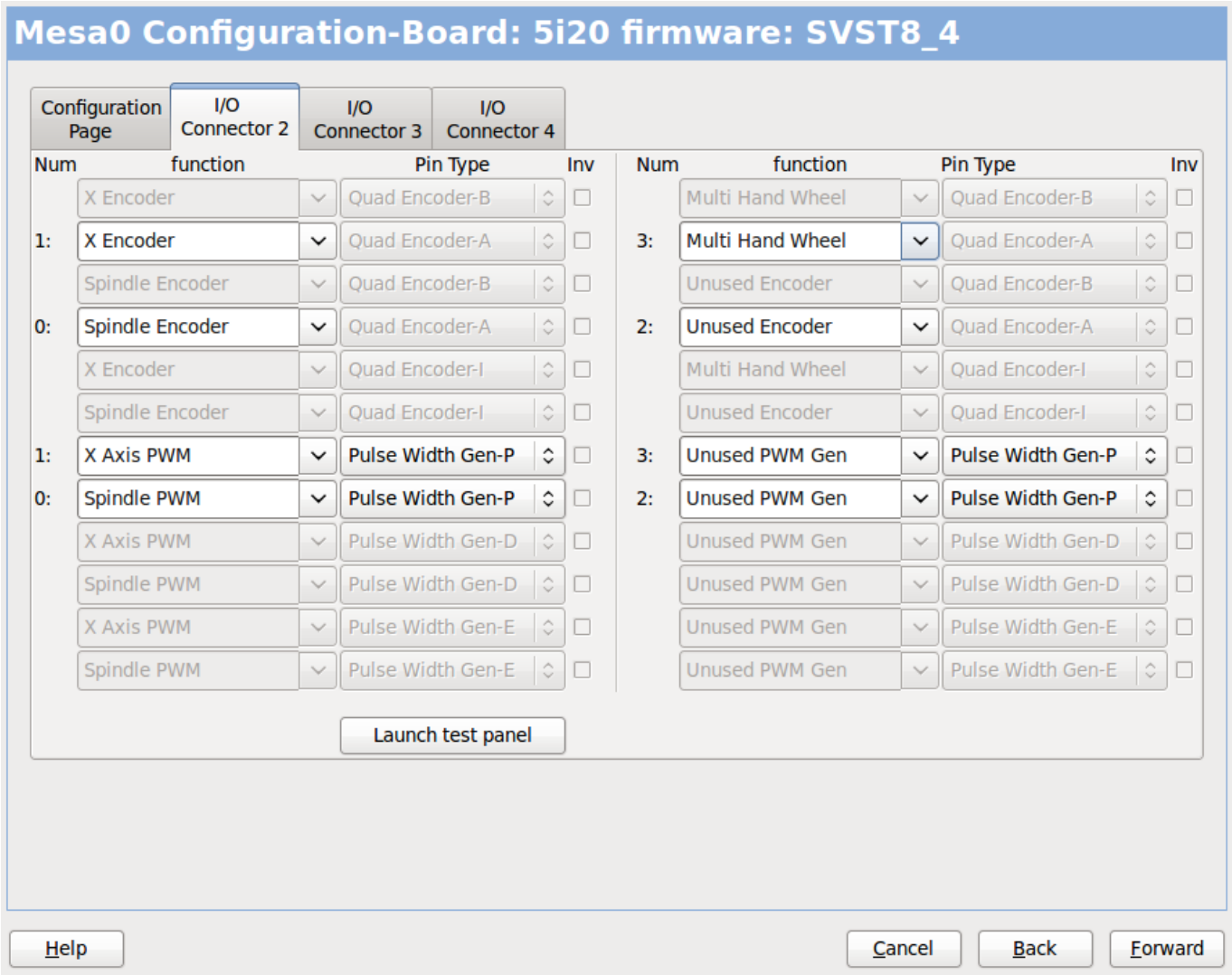



Figure 1.22: Mesa I/O C2

On this tab with this firmware the components are setup for a 7i33 daughter board, usually used with closed loop servos. Note the component numbers of the encoder counters and PWM drivers are not in numerical order. This follows the daughter board requirements.

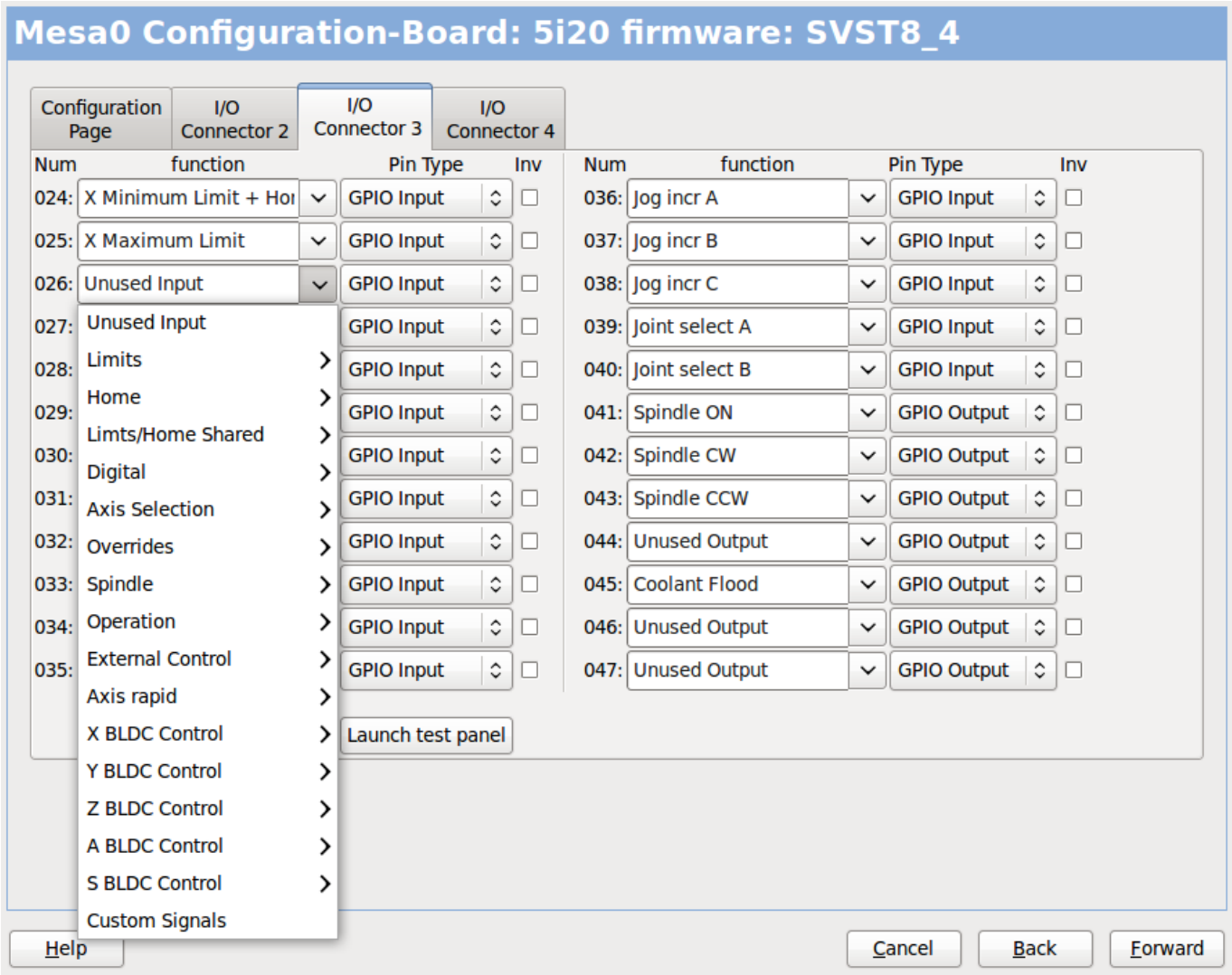



Figure 1.23: Mesa I/O C3

On this tab all the pins are GPIO. Note the 3 digit numbers - they will match the HAL pin number. GPIO pins can be selected as input or output and can be inverted.

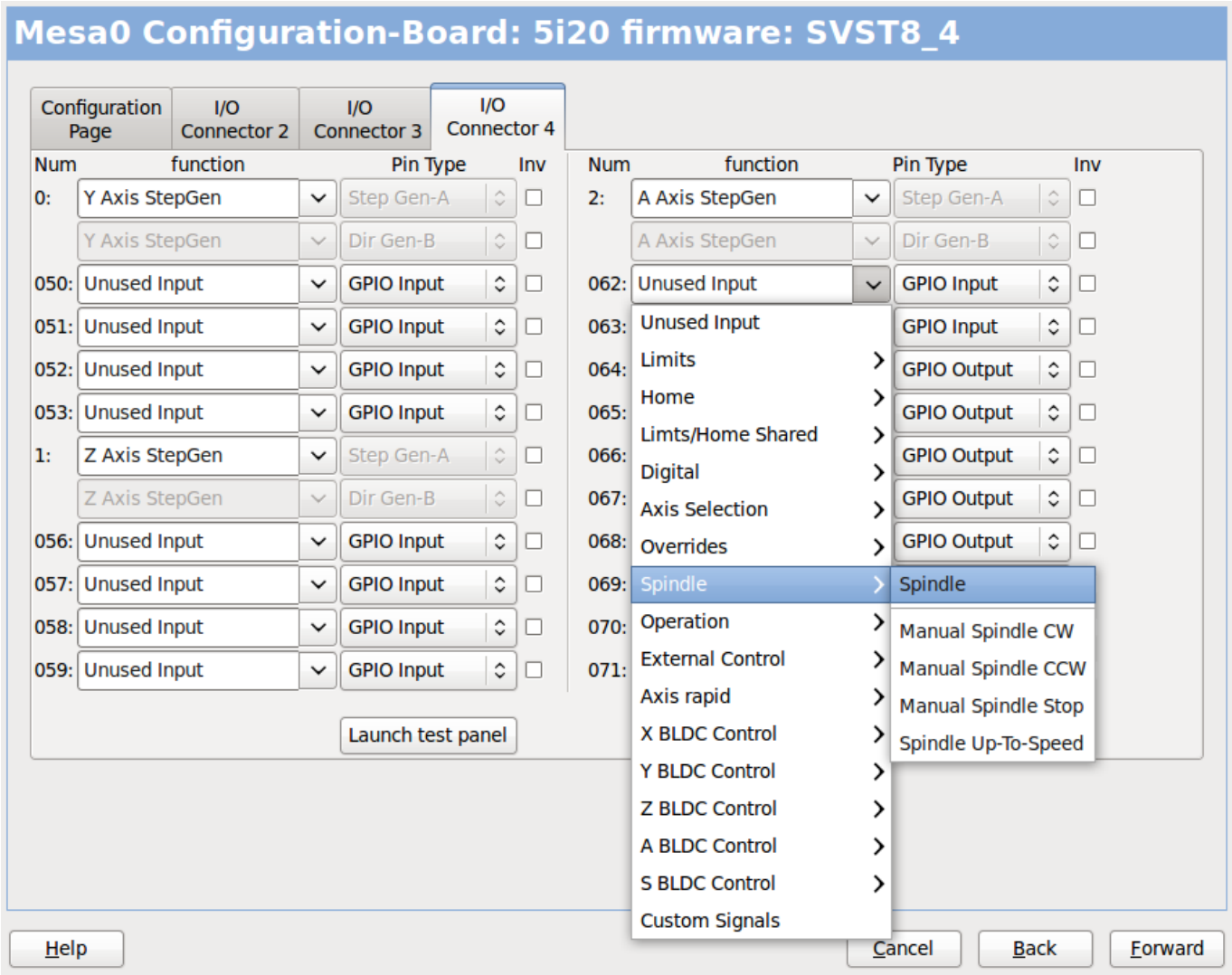
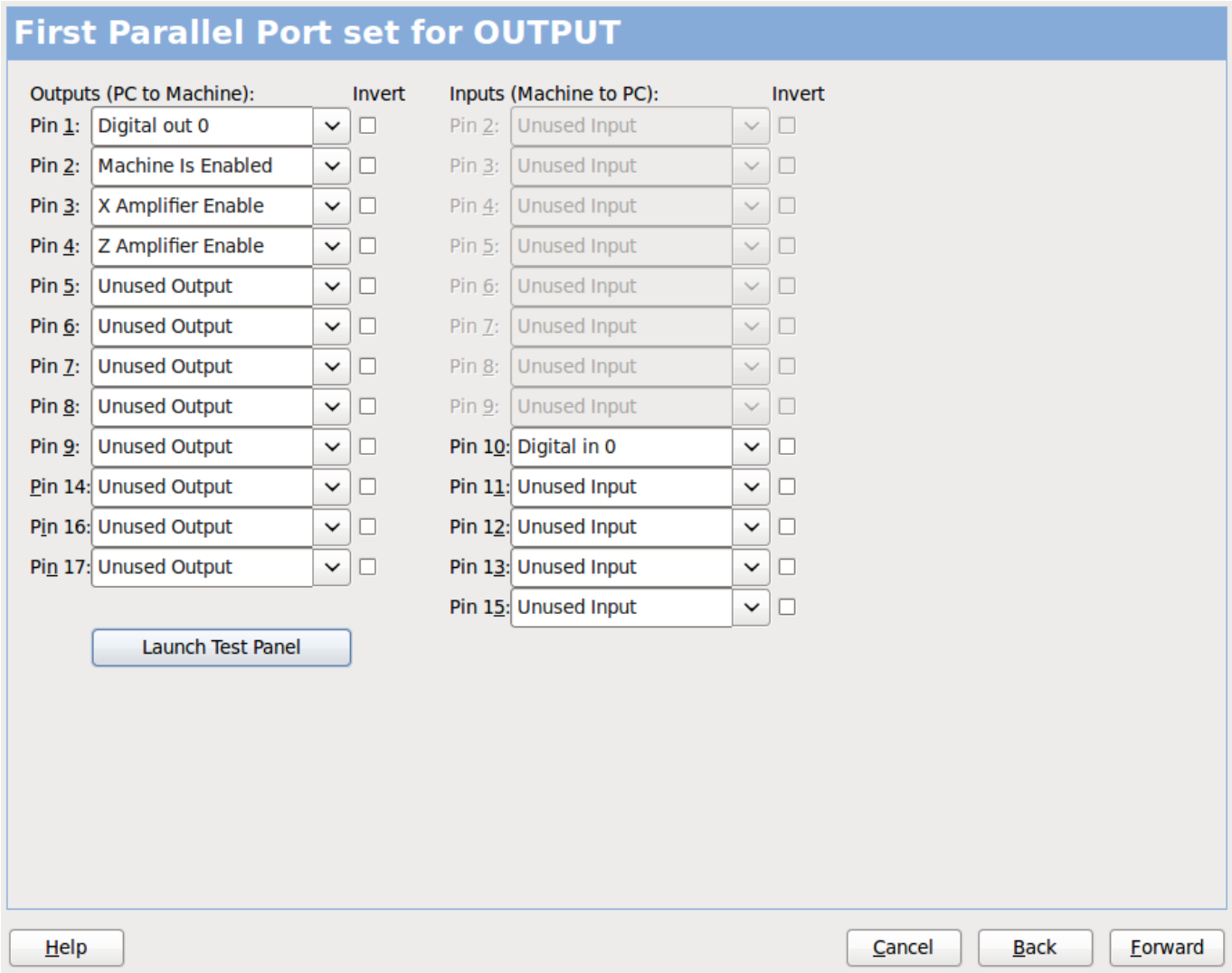



Figure 1.24: Mesa I/O C4

On this tab there are a mix of step generators and GPIO. Step generators output and direction pins can be inverted. Note that inverting a Step Gen-A pin (the step output pin) changes the step timing. It should match what your controller expects.

### 1.7.8 Parport configuration




The parallel port can be used for simple I/O similar to Mesa's GPIO pins.

### 1.7.9 Axis Configuration

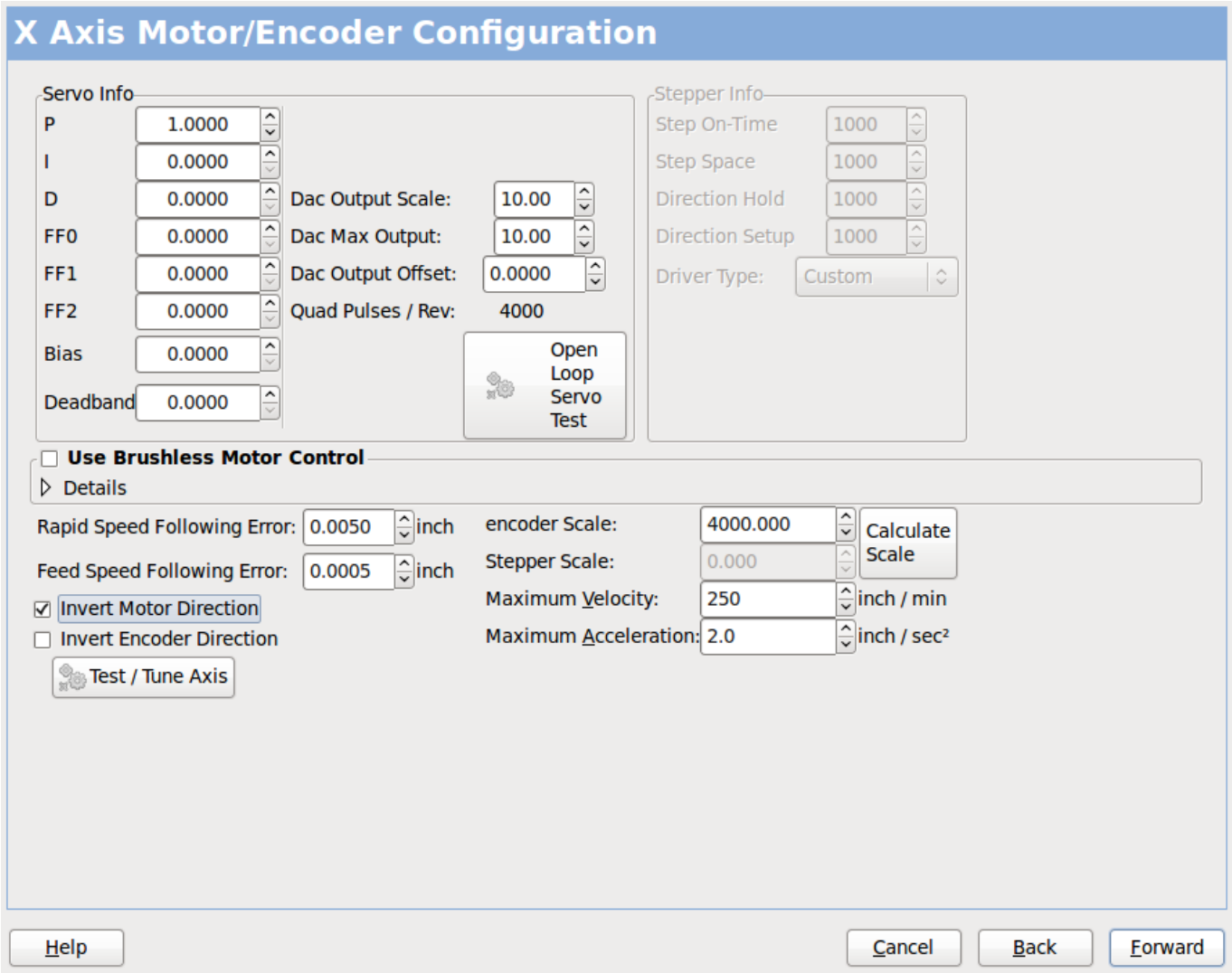



Figure 1.25: Axis Drive Configuration

This page allows configuring and testing of the motor and/or encoder combination . If using a servo motor an open loop test is available, if using a stepper a tuning test is available.

**Open Loop Test**

An open loop test is important as it confirms the direction of the motor and encoder. The motor should move the axis in the positive direction when the positive button is pushed and also the encoder should count in the positive direction. The axis movement should follow the Machinery's Handbook [5] standards or AXIS graphical display will not make much sense. Hopefully the help page and diagrams can help figure this out. Note that axis directions are based on TOOL movement not table movement. There is no acceleration ramping with the open loop test so start with lower DAC numbers. By moving the axis a known distance one can confirm the encoder scaling. The encoder should count even without the amp enabled depending on how power is supplied to the encoder.

[5] "axis nomenclature" in the chapter "Numerical Control" in the "Machinery's Handbook" published by Industrial Press.

**Warning**
If the motor and encoder do not agree on counting direction then the servo will run away when using PID control.

Since at the moment PID settings can not be tested in PNCconf the settings are really for when you re-edit a config - enter your tested PID settings.

DAC scaling, max output and offset are used to tailor the DAC output.

**Compute DAC**

These two values are the scale and offset factors for the axis output to the motor amplifiers. The second value (offset) is subtracted from the computed output (in volts), and divided by the first value (scale factor), before being written to the D/A converters. The units on the scale value are in true volts per DAC output volts. The units on the offset value are in volts. These can be used to linearize a DAC.

Specifically, when writing outputs, the LinuxCNC first converts the desired output in quasi-SI units to raw actuator values, e.g., volts for an amplifier DAC. This scaling looks like: The value for scale can be obtained analytically by doing a unit analysis, i.e., units are [output SI units]/[actuator units]. For example, on a machine with a velocity mode amplifier such that 1 volt results in 250 mm/sec velocity, Note that the units of the offset are in machine units, e.g., mm/sec, and they are pre-subtracted from the sensor readings. The value for this offset is obtained by finding the value of your output which yields 0.0 for the actuator output. If the DAC is linearized, this offset is normally 0.0.

The scale and offset can be used to linearize the DAC as well, resulting in values that reflect the combined effects of amplifier gain, DAC non-linearity, DAC units, etc. To do this, follow this procedure:

- Build a calibration table for the output, driving the DAC with a desired voltage and measuring the result:

Table 1.2: Output Voltage Measurements

| Raw | Measured |
|---|---|
| -10 | **-9.93** |
| -9 | **-8.83** |
| 0 | **-0.96** |
| 1 | **-0.03** |
| 9 | **9.87** |
| 10 | **10.07** |

- Do a least-squares linear fit to get coefficients a, b such that meas=a*raw+b
- Note that we want raw output such that our measured result is identical to the commanded output. This means
  - cmd=a*raw+b
  - raw=(cmd-b)/a
- As a result, the a and b coefficients from the linear fit can be used as the scale and offset for the controller directly.

MAX OUTPUT: The maximum value for the output of the PID compensation that is written to the motor amplifier, in volts. The computed output value is clamped to this limit. The limit is applied before scaling to raw output units. The value is applied symmetrically to both the plus and the minus side.

**Tuning Test** The tuning test unfortunately only works with stepper based systems. Again confirm the directions on the axis is correct. Then test the system by running the axis back and forth, If the acceleration or max speed is too high you will lose steps. While jogging, Keep in mind it can take a while for an axis with low acceleration to stop. Limit switches are not functional during this test. You can set a pause time so each end of the test movement. This would allow you to set up and read a dial indicator to see if you are loosing steps.

**Stepper Timing** Stepper timing needs to be tailored to the step controller's requirements. Pncconf supplies some default controller timing or allows custom timing settings . See http://wiki.linuxcnc.org/cgi-bin/wiki.pl?Stepper_Drive_Timing for some more known timing numbers (feel free to add ones you have figured out). If in doubt use large numbers such as 5000 this will only limit max speed.

**Brushless Motor Control** These options are used to allow low level control of brushless motors using special firmware and daughter boards. It also allows conversion of HALL sensors from one manufacturer to another. It is only partially supported and will require one to finish the HAL connections. Contact the mail-list or forum for more help.

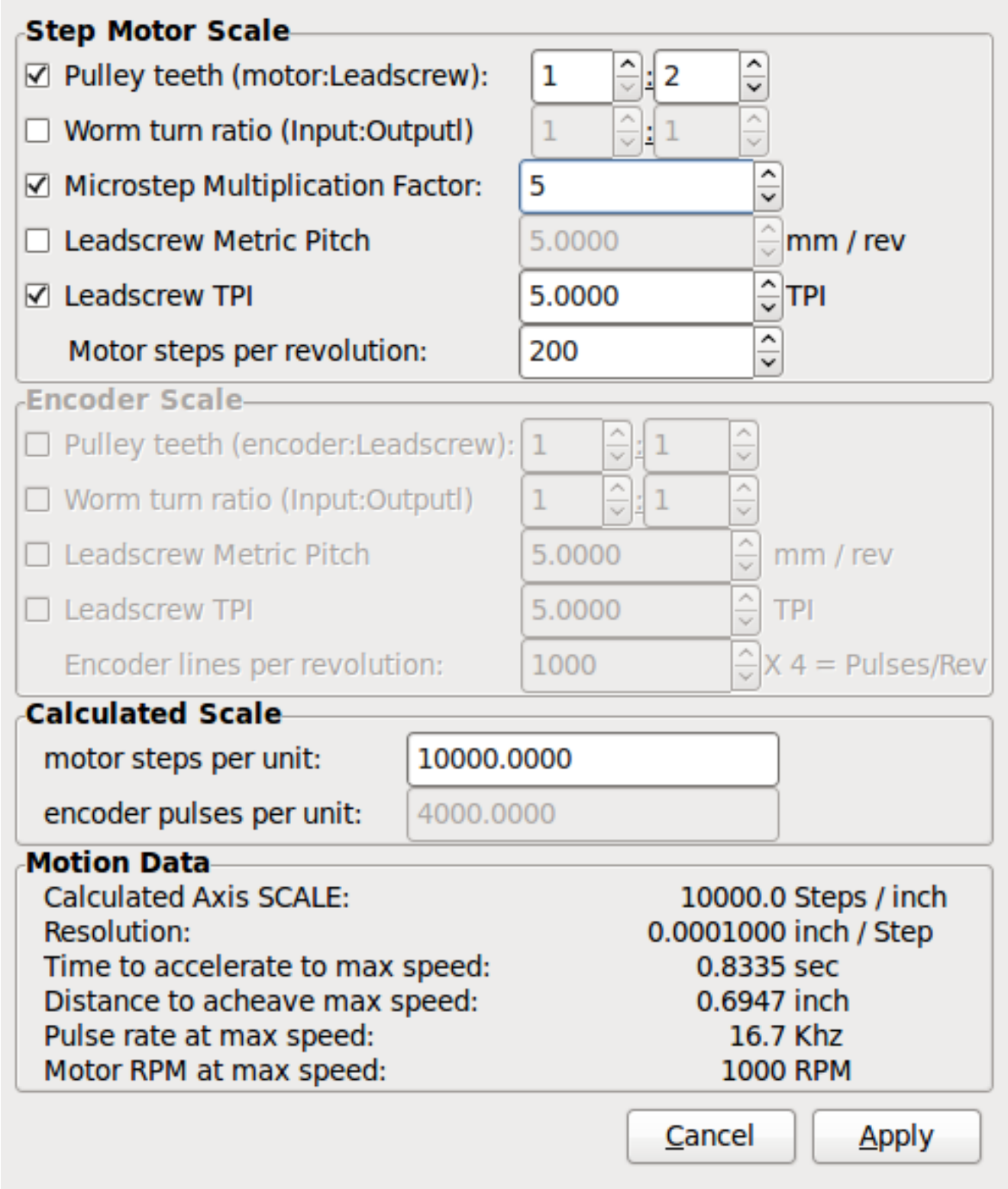



Figure 1.26: Axis Scale Calculation

The scale settings can be directly entered or one can use the *calculate scale* button to assist. Use the check boxes to select appropriate calculations. Note that *pulley teeth* requires the number of teeth not the gear ratio. Worm turn ratio is just the opposite it requires the gear ratio. If your happy with the scale press apply otherwise push cancel and enter the scale directly.

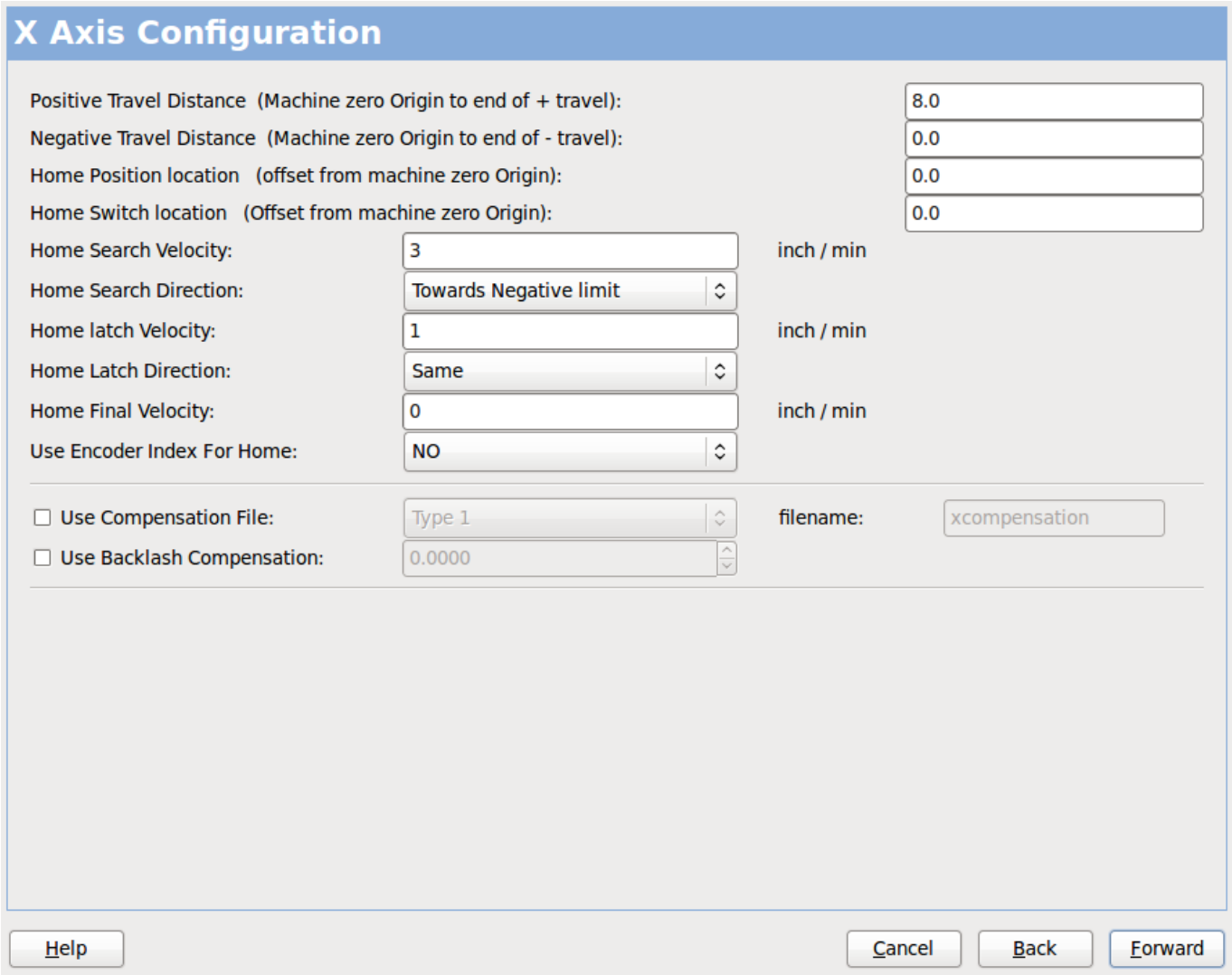



Figure 1.27: Axis Configuration

Also refer to the diagram tab for two examples of home and limit switches. These are two examples of many different ways to set homing and limits.

**Important**
It is very important to start with the axis moving in the right direction or else getting homing right is very difficult!

Remember positive and negative directions refer to the TOOL not the table as per the Machinists handbook.

ON A TYPICAL KNEE OR BED MILL

- when the TABLE moves out that is the positive Y direction
- when the TABLE moves left that is the positive X direction
- when the TABLE moves down that is the positive Z direction
- when the HEAD moves up that is the positive Z direction

ON A TYPICAL LATHE

- when the TOOL moves right, away from the chuck
- that is the positive Z direction
- when the TOOL moves toward the operator
- that is the positive X direction. Some lathes have X
- opposite (eg tool on back side), that will work fine but
- AXIS graphical display can not be made to reflect this.

When using homing and / or limit switches LinuxCNC expects the HAL signals to be true when the switch is being pressed / tripped. If the signal is wrong for a limit switch then LinuxCNC will think the machine is on end of limit all the time. If the home switch search logic is wrong LinuxCNC will seem to home in the wrong direction. What it actually is doing is trying to BACK off the home switch.

Decide on limit switch location.

Limit switches are the back up for software limits in case something electrical goes wrong eg. servo runaway. Limit switches should be placed so that the machine does not hit the physical end of the axis movement. Remember the axis will coast past the contact point if moving fast. Limit switches should be *active low* on the machine. eg. power runs through the switches all the time - a loss of power (open switch) trips. While one could wire them the other way, this is fail safe. This may need to be inverted so that the HAL signal in LinuxCNC in *active high* - a TRUE means the switch was tripped. When starting LinuxCNC if you get an on-limit warning, and axis is NOT tripping the switch, inverting the signal is probably the solution. (use HALMETER to check the corresponding HAL signal eg. axis.0.pos-lim-sw-in X axis positive limit switch)

Decide on the home switch location.

If you are using limit switches You may as well use one as a home switch. A separate home switch is useful if you have a long axis that in use is usually a long way from the limit switches or moving the axis to the ends presents problems of interference with material. eg a long shaft in a lathe makes it hard to home to limits with out the tool hitting the shaft, so a separate home switch closer to the middle may be better. If you have an encoder with index then the home switch acts as a course home and the index will be the actual home location.

Decide on the MACHINE ORIGIN position.

MACHINE ORIGIN is what LinuxCNC uses to reference all user coordinate systems from. I can think of little reason it would need to be in any particular spot. There are only a few G codes that can access the MACHINE COORDINATE system.( G53, G30 and G28 ) If using tool-change-at-G30 option having the Origin at the tool change position may be convenient. By convention, it may be easiest to have the ORIGIN at the home switch.

Decide on the (final) HOME POSITION.

this just places the carriage at a consistent and convenient position after LinuxCNC figures out where the ORIGIN is.

Measure / calculate the positive / negative axis travel distances.

Move the axis to the origin. Mark a reference on the movable slide and the non-movable support (so they are in line) move the machine to the end of limits. Measure between the marks that is one of the travel distances. Move the table to the other end of travel. Measure the marks again. That is the other travel distance. If the ORIGIN is at one of the limits then that travel distance will be zero.

**(machine) ORIGIN**
: The Origin is the MACHINE zero point. (not the zero point you set your cutter / material at). LinuxCNC uses this point to reference everything else from. It should be inside the software limits. LinuxCNC uses the home switch location to calculate the origin position (when using home switches or must be manually set if not using home switches.

**Travel distance**
: This is the maximum distance the axis can travel in each direction. This may or may not be able to be measured directly from origin to limit switch. The positive and negative travel distances should add up to the total travel distance.

**POSITIVE TRAVEL DISTANCE**
This is the distance the Axis travels from the Origin to the positive travel distance or the total travel minus the negative travel distance. You would set this to zero if the origin is positioned at the positive limit. The will always be zero or a positive number.

**NEGATIVE TRAVEL DISTANCE**
This is the distance the Axis travels from the Origin to the negative travel distance. or the total travel minus the positive travel distance. You would set this to zero if the origin is positioned at the negative limit. This will always be zero or a negative number. If you forget to make this negative PNCconf will do it internally.

**(Final) HOME POSITION**
This is the position the home sequence will finish at. It is referenced from the Origin so can be negative or positive depending on what side of the Origin it is located. When at the (final) home position if you must move in the Positive direction to get to the Origin, then the number will be negative.

**HOME SWITCH LOCATION**
This is the distance from the home switch to the Origin. It could be negative or positive depending on what side of the Origin it is located. When at the home switch location if you must move in the Positive direction to get to the Origin, then the number will be negative. If you set this to zero then the Origin will be at the location of the limit switch (plus distance to find index if used)

**Home Search Velocity**
Course home search velocity in units per minute.

**Home Search Direction**
Sets the home switch search direction either negative (ie. towards negative limit switch) or positive (ie. towards positive limit switch)

**Home Latch Velocity**
Fine Home search velocity in units per minute

**Home Final Velocity**
Velocity used from latch position to (final) home position in units per minute. Set to 0 for max rapid speed

**Home latch Direction**
Allows setting of the latch direction to the same or opposite of the search direction.

**Use Encoder Index For Home**
LinuxCNC will search for an encoder index pulse while in the latch stage of homing.

**Use Compensation File**
Allows specifying a Comp filename and type. Allows sophisticated compensation. See AXIS Section of the INI Chapter.

**Use Backlash Compensation**
Allows setting of simple backlash compensation. Can not be used with Compensation File. See AXIS Section of the INI Chapter.

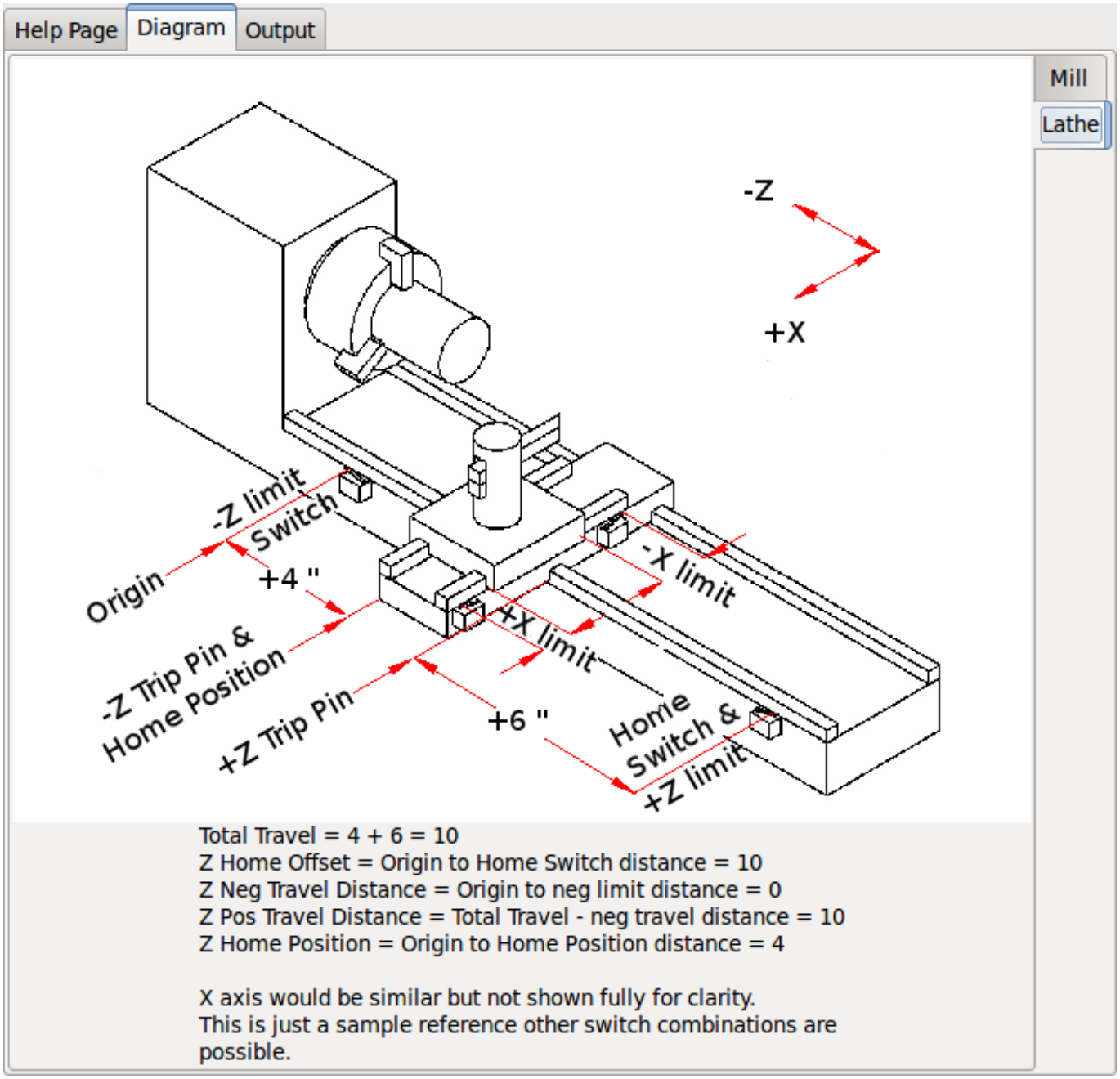



Figure 1.28: AXIS Help Diagram

The diagrams should help to demonstrate an example of limit switches and standard axis movement directions. In this example the Z axis was two limit switches, the positive switch is shared as a home switch. The MACHINE ORIGIN (zero point) is located at the negative limit. The left edge of the carriage is the negative trip pin and the right the positive trip pin. We wish the FINAL HOME POSITION to be 4 inches away from the ORIGIN on the positive side. If the carriage was moved to the positive limit we would measure 10 inches between the negative limit and the negative trip pin.

### 1.7.10 Spindle Configuration

If you select spindle signals then this page is available to configure spindle control.

**Tip**
Many of the option on this page will not show unless the proper option was selected on previous pages!

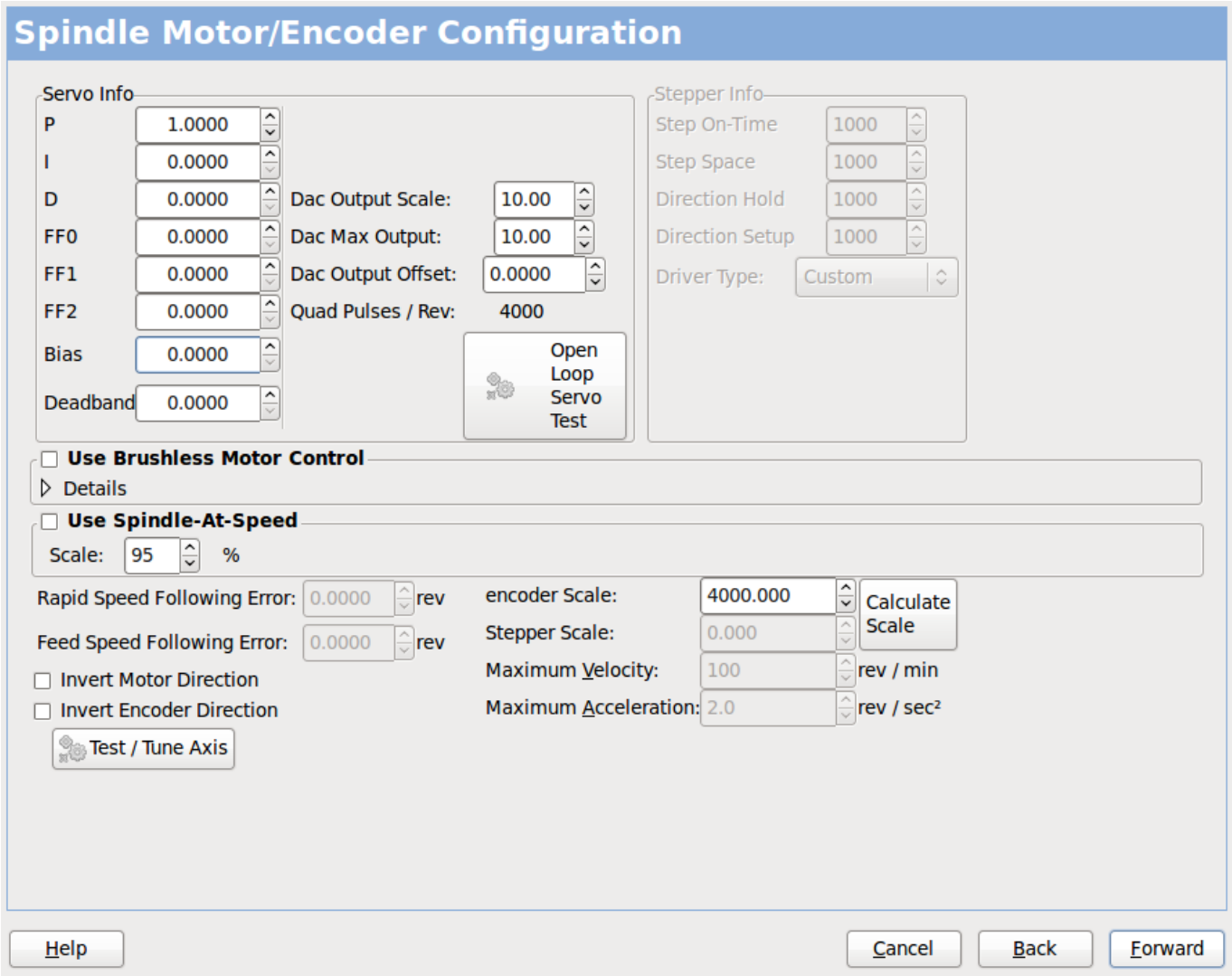



Figure 1.29: Spindle Configuration

This page is similar to the axis motor configuration page.

There are some differences:

- Unless one has chosen a stepper driven spindle there is no acceleration or velocity limiting.
- There is no support for gear changes or ranges.
- If you picked a VCP spindle display option then spindle-at-speed scale and filter settings may be shown.
- Spindle-at-speed allows LinuxCNC to wait till the spindle is at the requested speed before moving the axis. This is particularly handy on lathes with constant surface feed and large speed diameter changes. It requires either encoder feedback or a digital spindle-at-speed signal typically connected to a VFD drive.
- If using encoder feedback, you may select a spindle-at-speed scale setting that specifies how close the actual speed must be to the requested speed to be considered at-speed.

- If using encoder feedback, the VCP speed display can be erratic - the filter setting can be used to smooth out the display. The encoder scale must be set for the encoder count / gearing used.
- If you are using a single input for a spindle encoder you must add the line: setp hm2_7i43.0.encoder.00.counter-mode 1 (changing the board name and encoder number to your requirements) into a custom HAL file. See the Encoders Section in Hostmot2 for more info about counter mode.

### 1.7.11 Advanced Options

This allows setting of HALUI commands and loading of classicladder and sample ladder programs. If you selected GLADE VCP options such as for zeroing axis, there will be commands showing. See the HALUI Chapterfor more info on using custom halcmds. There are several ladder program options. The Estop program allows an external ESTOP switch or the GUI frontend to throw an Estop. It also has a timed lube pump signal. The Z auto touch-off is with a touch-off plate, the GLADE VCP touch-off button and special HALUI commands to set the current user origin to zero and rapid clear. The serial modbus program is basically a blank template program that sets up classicladder for serial modbus. See the Classicladder Chapter in the manual.

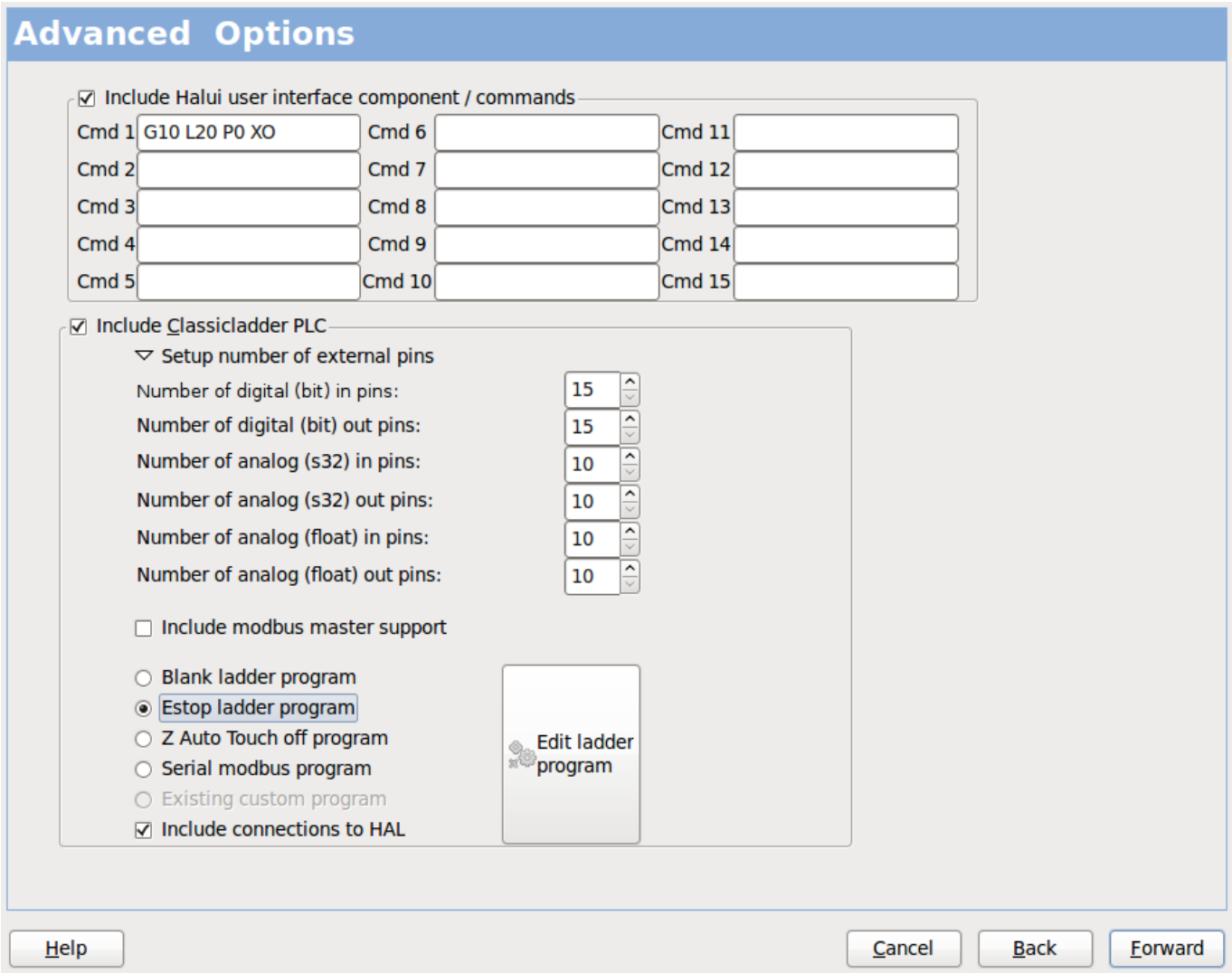



Figure 1.30: Advanced Options

### 1.7.12 HAL Components

On this page you can add additional HAL components you might need for custom HAL files. In this way one should not have to hand edit the main HAL file, while still allowing user needed components.

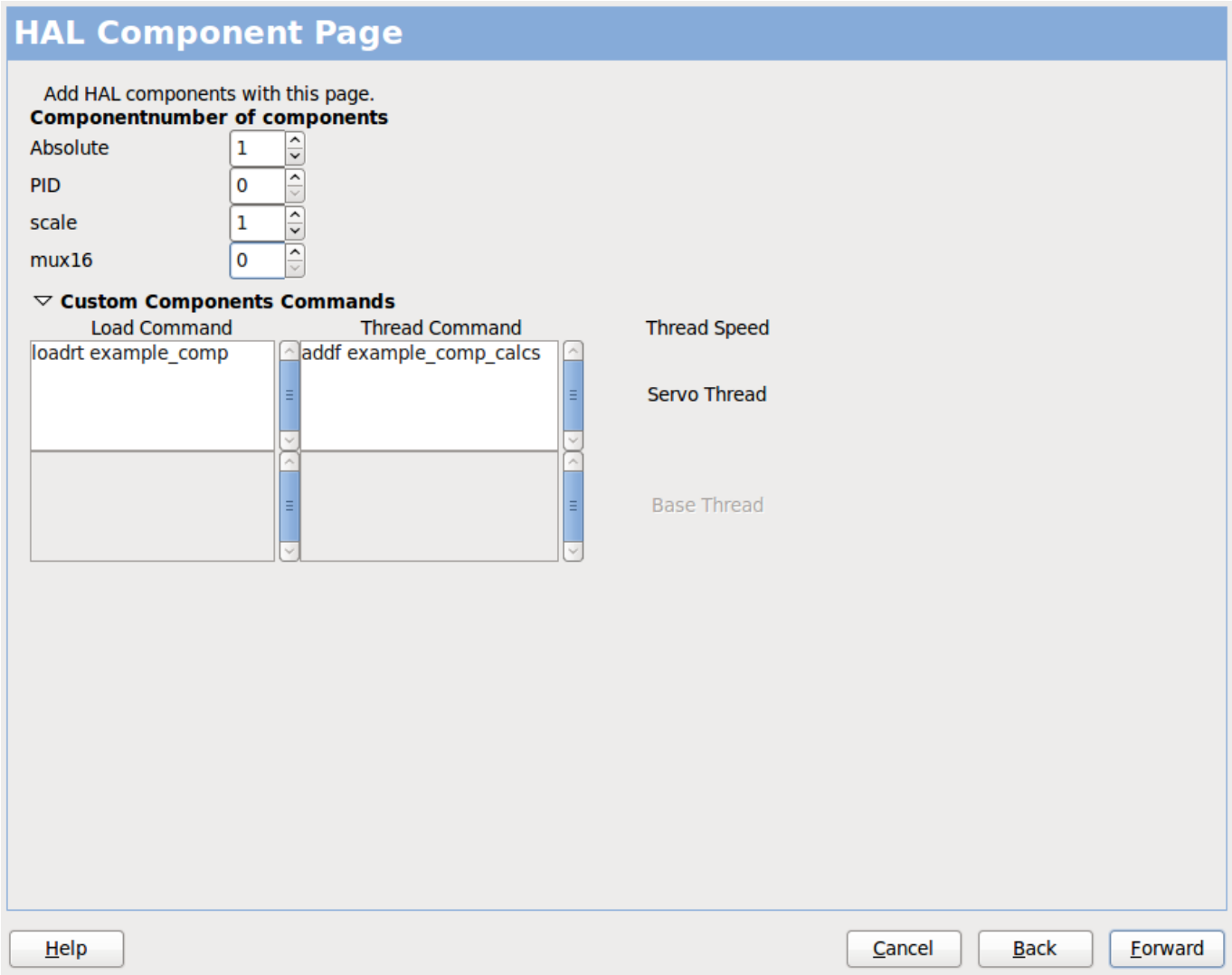



Figure 1.31: HAL Components

The first selection is components that pncconf uses internally. You may configure pncconf to load extra instances of the components for your custom HAL file.

Select the number of instances your custom file will need, pncconf will add what it needs after them.

Meaning if you need 2 and pncconf needs 1 pncconf will load 3 instances and use the last one.

**Custom Component Commands**

This selection will allow you to load HAL components that pncconf does not use. Add the loadrt or loadusr command, under the heading *loading command* Add the addf command under the heading *Thread command*. The components will be added to the thread between reading of inputs and writing of outputs, in the order you write them in the *thread command*.

### 1.7.13 Advanced Usage Of PNCconf

PNCconf does its best to allow flexible customization by the user. PNCconf has support for custom signal names, custom loading of components, custom HAL files and custom firmware.

There are also signal names that PNCconf always provides regardless of options selected, for user's custom HAL files With some thought most customizations should work regardless if you later select different options in PNCconf.

Eventually if the customizations are beyond the scope of PNCconf's frame work you can use PNCconf to build a base config or use one of LinuxCNC's sample configurations and just hand edit it to what ever you want.

**Custom Signal Names**
: If you wish to connect a component to something in a custom HAL file write a unique signal name in the combo entry box. Certain components will add endings to your custom signal name:

Encoders will add < customname > +:

- position
- count
- velocity
- index-enable
- reset

Steppers add:

- enable
- counts
- position-cmd
- position-fb
- velocity-fb

PWM add:

- enable
- value

GPIO pins will just have the entered signal name connected to it

In this way one can connect to these signals in the custom HAL files and still have the option to move them around later.

**Custom Signal Names**
: The Hal Components page can be used to load components needed by a user for customization.

**Loading Custom Firmware**
: PNCconf searches for firmware on the system and then looks for the XML file that it can convert to what it understands. These XML files are only supplied for officially released firmware from the LinuxCNC team. To utilize custom firmware one must convert it to an array that PNCconf understands and add its file path to PNCconf's preference file. By default this path searches the desktop for a folder named custom_firmware and a file named firmware.py.

The hidden preference file is in the user's home file, is named .pncconf-preferences and require one to select *show hidden files* to see and edit it. The contents of this file can be seen when you first load PNCconf - press the help button and look at the output page.

Ask on the LinuxCNC mail-list or forum for info about converting custom firmware. Not all firmware can be utilized with PNCconf.

**Custom HAL Files**

There are four custom files that you can use to add HAL commands to:

- custom.hal is for HAL commands that don't have to be run after the GUI frontend loads. It is run after the configuration-named HAL file.
- custom_postgui.hal is for commands that must be run after AXIS loads or a standalone PYVCP display loads.
- custom_gvcp.hal is for commands that must be run after glade VCP is loaded.
- shutdown.hal is for commands to run when LinuxCNC shuts down in a controlled manner.

## 1.8 Linux FAQ

These are some basic Linux commands and techniques for new to Linux users. More complete information can be found on the web or by using the man pages.

### 1.8.1 Automatic Login

When you install LinuxCNC with the Ubuntu LiveCD the default is to have to log in each time you turn the computer on. To enable automatic login go to *System* > *Administration* > *Login Window*. If it is a fresh install the Login Window might take a second or three to pop up. You will have to have your password that you used for the install to gain access to the Login Window Preferences window. In the Security tab check off Enable Automatic Login and pick a user name from the list (that would be you).

### 1.8.2 Automatic Startup

To have LinuxCNC start automatically with your config after turning on the computer go to *System* > *Preferences* > *Sessions* > *Startup Applications*, click Add. Browse to your config and select the .ini file. When the file picker dialog closes, add linuxcnc and a space in front of the path to your .ini file.

Example:

```
linuxcnc /home/mill/linuxcnc/config/mill/mill.ini
```

### 1.8.3 Terminal

Many things need to be done from the terminal like checking the kernel message buffer with *dmesg*. Ubuntu and Linux Mint have a keyboard shortcut Ctrl + Alt + t. Most modern file managers support the right key to open a terminal just make sure your right clicking on a blank area or a directory not a file name. Most OS's have the terminal as a menu item, usually in Accessories.

### 1.8.4 Man Pages

A man page (short for manual page) is a form of software documentation usually found on a Unix or Unix-like operating system like Linux.

To view a man page open up a terminal to find out something about the find command in the terminal window type:

```
man find
```

Use the Page Up and Page Down keys to view the man page and the Q key to quit viewing.

**Note**

Viewing the man page from the terminal may not get the expeced man page. For example if you type in man abs you will get the C abs not the LinuxCNC abs. It is best to view the LinuxCNC man pages in the HTML documents.

### 1.8.5 List Modules

Sometimes when troubleshooting you need to get a list of modules that are loaded. In a terminal window type:

```
lsmod
```

If you want to send the output from lsmod to a text file in a terminal window type:

```
lsmod > mymod.txt
```

The resulting text file will be located in the home directory if you did not change directories when you opened up the terminal window and it will be named mymod.txt or what ever you named it.

### 1.8.6 Editing a Root File

When you open the file browser and you see the Owner of the file is root you must do extra steps to edit that file. Editing some root files can have bad results. Be careful when editing root files. Generally, you can open and view most root files, but they will open in *read only* mode.

#### 1.8.6.1 The Command Line Way

Open a terminal and type

```
sudo gedit
```

Open the file with File > Open > Edit

#### 1.8.6.2 The GUI Way

1. Right click on the desktop and select Create Launcher
2. Type a name in like sudo edit
3. Type *gksudo "gnome-open %u"* as the command and save the launcher to your desktop
4. Drag a file onto your launcher to open and edit

#### 1.8.6.3 Root Access

In Ubuntu you can become root by typing in "sudo -i" in a terminal window then typing in your password. Be careful, because you can really foul things up as root if you don’t know what you’re doing.

### 1.8.7 Terminal Commands

#### 1.8.7.1 Working Directory

To find out the path to the present working directory in the terminal window type:

```
pwd
```

#### 1.8.7.2 Changing Directories

To move up one level in the terminal window type:

```
cd ..
```

To move up two levels in the terminal window type:

```
cd ../..
```

To move down to the linuxcnc/configs subdirectory in the terminal window type:

```
cd linuxcnc/configs
```

#### 1.8.7.3 Listing files in a directory

To view a list of all the files and subdirectories in the terminal window type:

```
dir
```

or

```
ls
```

#### 1.8.7.4 Finding a File

The find command can be a bit confusing to a new Linux user. The basic syntax is:

```
find starting-directory parameters actions
```

For example to find all the .ini files in your linuxcnc directory you first need to use the pwd command to find out the directory. Open a new terminal window and type:

```
pwd
```

And pwd might return the following result:

```
/home/joe
```

With this information put the command together like this:

```
find /home/joe/linuxcnc -name \*.ini -print
```

The -name is the name of the file your looking for and the -print tells it to print out the result to the terminal window. The *.ini tells find to return all files that have the .ini extension. The backslash is needed to escape the shell meta-characters. See the find man page for more information on find.

#### 1.8.7.5 Searching for Text

```
grep -irl 'text to search for' *
```

This will find all the files that contain the *text to search for* in the current directory and all the subdirectories below it, while ignoring the case. The -i is for ignore case and the -r is for recursive (include all subdirectories in the search). The -l option will return a list of the file names, if you leave the -l off you will also get the text where each occourance of the "text to search for" is found. The * is a wild card for search all files. See the grep man page for more information.

#### 1.8.7.6 Diagnostic Messages

To view the diagnostic messages use "dmesg" from the command window. To save the diagnostic messages to a file use the redirection operator >, like this:

```
dmesg > bootmsg.txt
```

The contents of this file can be copied and pasted on line to share with people trying to help you diagnose your problem.

To clear the message buffer type this:

```
sudo dmesg -c
```

This can be helpful to do just before launching LinuxCNC, so that there will only be a record of information related to the current launch of LinuxCNC.

To find the built in parallel port address use grep to filter the information out of dmesg.

After boot up open a terminal and type:

```
dmesg|grep parport
```

### 1.8.8 Convenience Items

#### 1.8.8.1 Terminal Launcher

If you want to add a terminal launcher to the panel bar on top of the screen you typically can right click on the panel at the top of the screen and select "Add to Panel". Select Custom Application Launcher and Add. Give it a name and put gnome-terminal in the command box.

### 1.8.9 Hardware Problems

#### 1.8.9.1 Hardware Info

To find out what hardware is connected to your motherboard in a terminal window type:

```
lspci -v
```

#### 1.8.9.2 Monitor Resolution

During installation Ubuntu attempts to detect the monitor settings. If this fails you are left with a generic monitor with a maximum resolution of 800x600.

Instructions for fixing this are located here:

https://help.ubuntu.com/community/FixVideoResolutionHowto

### 1.8.10 Paths

**Relative Paths** Relative paths are based on the startup directory which is the directory containing the ini file. Using relative paths can facilitate relocation of configurations but requires a good understanding of linux path specifiers.

```
./f0        is the same as f0, e.g., a file named f0 in the startup directory
../f1       refers to a file f1 in the parent directory
../../f2    refers to a file f2 in the parent of the parent directory
../../../f3 etc.
```

## 1.9 Lathe User Information

### 1.9.1 Lathe Mode

If your CNC machine is a lathe, there are some specific changes you will probably want to make to your ini file in order to get the best results from LinuxCNC.

If you are using the AXIS display, have Axis display your lathe tools properly. See the INI Configuration section for more details.

To set up AXIS for Lathe Mode.

```
[DISPLAY]

# Tell the Axis Display our machine is a lathe.
LATHE = TRUE
```

Lathe Mode in Axis does not set your default plane to G18 (XZ). You must program that in the preamble of each gcode file or (better) add it to your ini file, like this:

```
[RS274NGC]

# g-code modal codes (modes) that the interpreter is initialized with on startup
RS274NGC_STARTUP_CODE = G18 G20 G90
```

If your using Gmoccapy then see the the Gmoccapy Lathe section.

### 1.9.2 Lathe Tool Table

The "Tool Table" is a text file that contains information about each tool. The file is located in the same directory as your configuration and is called "tool.tbl" by default. The tools might be in a tool changer or just changed manually. The file can be edited with a text editor or be updated using G10 L1,L10,L11. There is also a built-in tool table editor in the Axis display. The maximum number of entries in the tool table is 56. The maximum tool and pocket number is 99999.

Earlier versions of LinuxCNC had two different tool table formats for mills and lathes, but since the 2.4.x release, one tool table format is used for all machines. Just ignore the parts of the tool table that don't pertain to your machine, or which you don't need to use. For more information on the specifics of the tool table format, see the Tool Table Section.

### 1.9.3 Lathe Tool Orientation

The following figure shows the lathe tool orientations with the center line angle of each orientation and info on FRONTANGLE and BACKANGLE. The FRONTANGLE and BACKANGLE are clockwise starting at a line parallel to Z+.

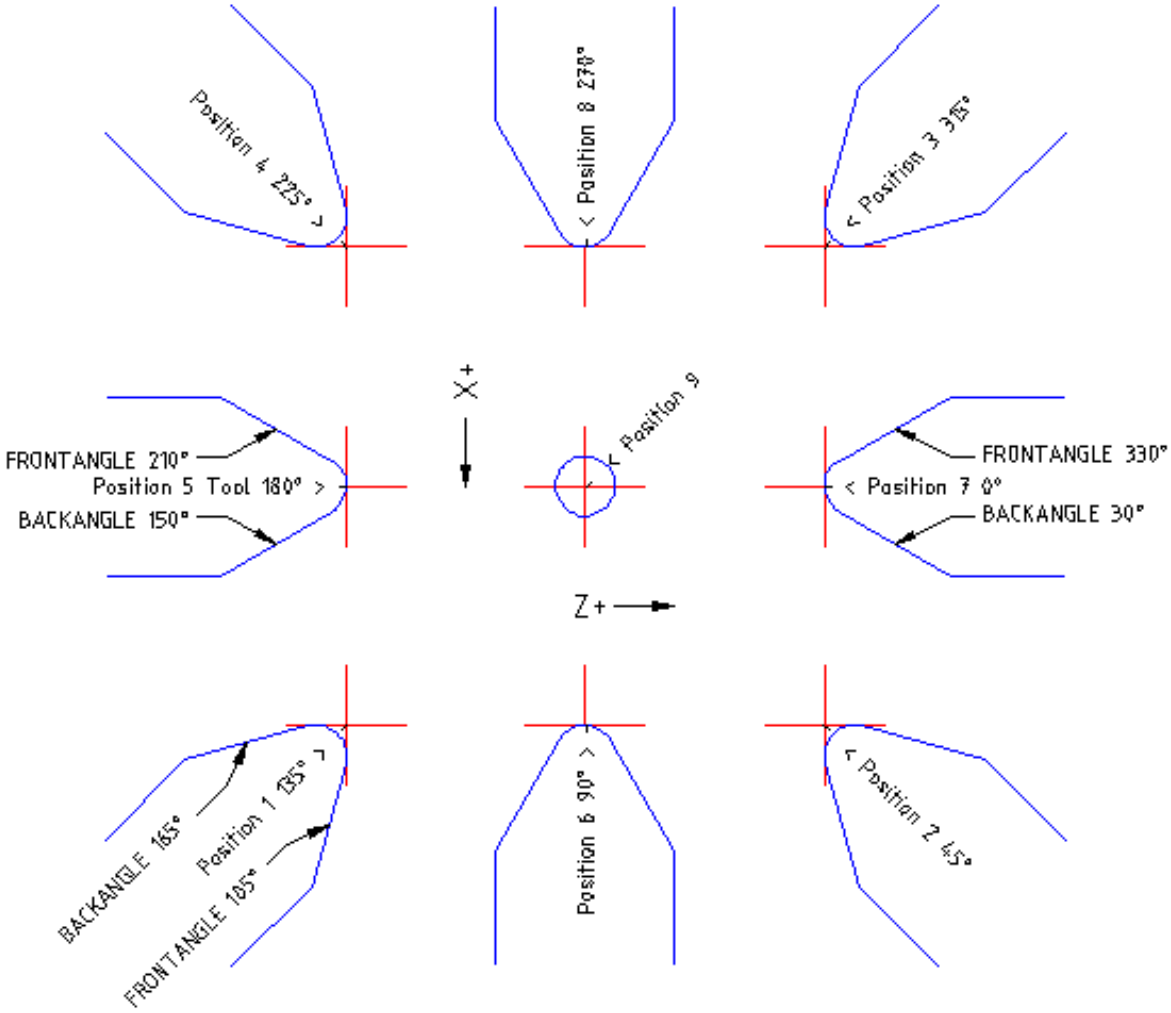
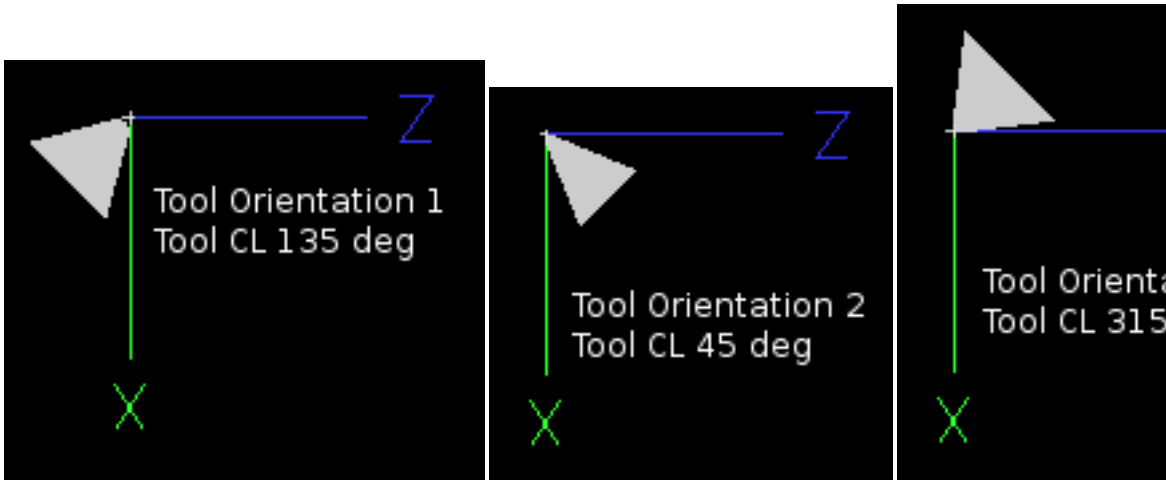
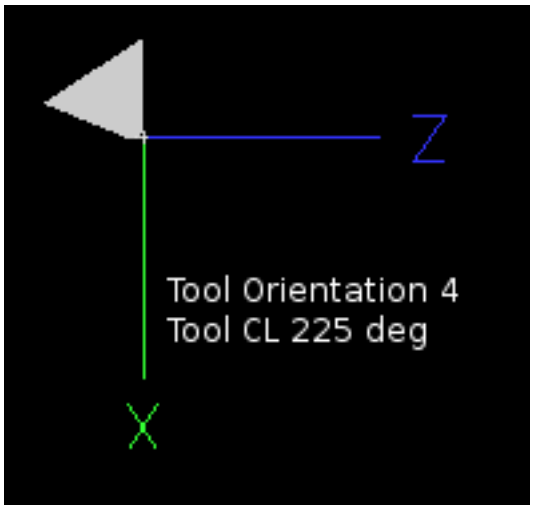


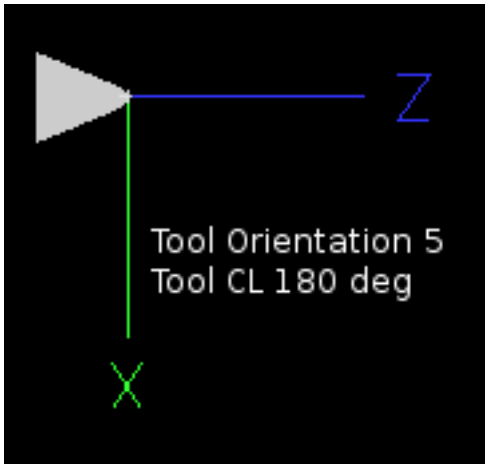

Figure 1.32: Lathe Tool Orientations

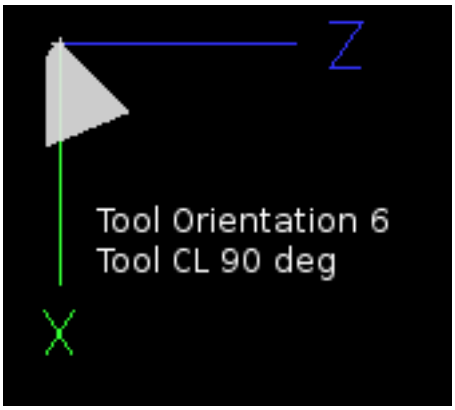
In AXIS the following figures show what the Tool Positions look like, as entered in the tool table.

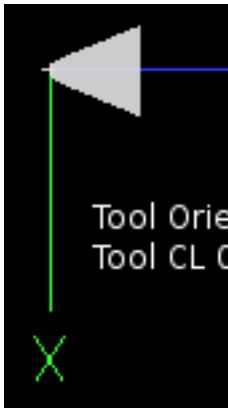
**Tool Positions 1, 2, 3 & 4**



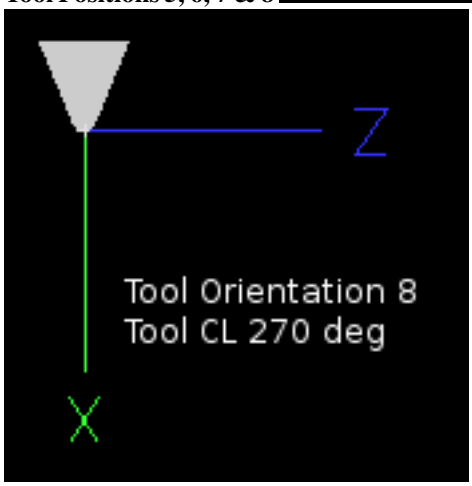



**Tool Positions 5, 6, 7 & 8**




### 1.9.4 Tool Touch Off

When running in lathe mode in AXIS you can set the X and Z in the tool table using the Touch Off window. If you have a tool turret you normally have *Touch off to fixture* selected when setting up your turret. When setting the material Z zero you have *Touch off to material* selected. For more information on the G codes used for tools see M6, Tn, and G43. For more information on tool touch off options in Axis see Tool Touch Off.

#### 1.9.4.1 X Touch Off

The X axis offset for each tool is normally an offset from the center line of the spindle.

One method is to take your normal turning tool and turn down some stock to a known diameter. Using the Tool Touch Off window enter the measured diameter (or radius if in radius mode) for that tool. Then using some layout fluid or a marker to coat the part bring each tool up till it just touches the dye and set it's X offset to the diameter of the part used using the tool touch off. Make sure any tools in the corner quadrants have the nose radius set properly in the tool table so the control point is correct. Tool touch off automatically adds a G43 so the current tool is the current offset.

A typical session might be:

1. Home each axis if not homed.
2. Set the current tool with *Tn M6 G43* where *n* is the tool number.
3. Select the X axis in the Manual Control window.
4. Move the X to a known position or take a test cut and measure the diameter.
5. Select Touch Off and pick Tool Table then enter the position or the diameter.
6. Follow the same sequence to correct the Z axis.

Note: if you are in Radius Mode you must enter the radius, not the diameter.

#### 1.9.4.2 Z Touch Off

The Z axis offsets can be a bit confusing at first because there are two elements to the Z offset. There is the tool table offset, and the machine coordinate offset. First we will look at the tool table offsets. One method is to use a fixed point on your lathe and set the Z offset for all tools from this point. Some use the spindle nose or chuck face. This gives you the ability to change to a new tool and set its Z offset without having to reset all the tools.

A typical session might be:

1. Home each axis if not homed.
2. Make sure no offsets are in effect for the current coordinate system.
3. Set the current tool with *Tn M6 G43* where *n* is the tool number.
4. Select the Z axis in the Manual Control window.
5. Bring the tool close to the control surface. Using a cylinder move the Z away from the control surface until the cylinder just passes between the tool and the control surface.
6. Select Touch Off and pick Tool Table and set the position to 0.0.
7. Repeat for each tool using the same cylinder.

Now all the tools are offset the same distance from a standard position. If you change a tool like a drill bit you repeat the above and it is now in sync with the rest of the tools for Z offset. Some tools might require a bit of cyphering to determine the control point from the touch off point. For example, if you have a 0.125" wide parting tool and you touch the left side off but want the right to be Z0, then enter 0.125" in the touch off window.

#### 1.9.4.3 The Z Machine Offset

Once all the tools have the Z offset entered into the tool table, you can use any tool to set the machine offset using the machine coordinate system.

A typical session might be:

1. Home each axis if not homed.
2. Set the current tool with "Tn M6" where "n" is the tool number.
3. Issue a G43 so the current tool offset is in effect.
4. Bring the tool to the work piece and set the machine Z offset.

If you forget to set the G43 for the current tool when you set the machine coordinate system offset, you will not get what you expect, as the tool offset will be added to the current offset when the tool is used in your program.

### 1.9.5 Spindle Synchronized Motion

Spindle synchronized motion requires a quadrature encoder connected to the spindle with one index pulse per revolution. See the motion man page and the Spindle Control Example for more information.

**Threading** The G76 threading cycle is used for both internal and external threads. For more information see the G76 Section.

**Constant Surface Speed** CSS or Constant Surface Speed uses the machine X origin modified by the tool X offset to compute the spindle speed in RPM. CSS will track changes in tool offsets. The X machine origin should be when the reference tool (the one with zero offset) is at the center of rotation. For more information see the G96 Section.

**Feed per Revolution** Feed per revolution will move the Z axis by the F amount per revolution. This is not for threading, use G76 for threading. For more information see the G95 Section.

### 1.9.6 Arcs

Calculating arcs can be mind challenging enough without considering radius and diameter mode on lathes as well as machine coordinate system orientation. The following applies to center format arcs. On a lathe you should include G18 in your preamble as the default is G17 even if you're in lathe mode, in the user interface Axis. Arcs in G18 XZ plane use I (X axis) and K (Z axis) offsets.

#### 1.9.6.1 Arcs and Lathe Design

The typical lathe has the spindle on the left of the operator and the tools on the operator side of the spindle center line. This is typically set up with the imaginary Y axis (+) pointing at the floor.

The following will be true on this type of setup:

- The Z axis (+) points to the right, away from the spindle.
- The X axis (+) points toward the operator, and when on the operator side of the spindle the X values are positive.

Some lathes with tools on the back side have the imaginary Y axis (+) pointing up.

G2/G3 Arc directions are based on the axis they rotate around. In the case of lathes, it is the imaginary Y axis. If the Y axis (+) points toward the floor, you have to look up for the arc to appear to go in the correct direction. So looking from above you reverse the G2/G3 for the arc to appear to go in the correct direction.

#### 1.9.6.2 Radius & Diameter Mode

When calculating arcs in radius mode you only have to remember the direction of rotation as it applies to your lathe.

When calculating arcs in diameter mode X is diameter and the X offset (I) is radius even if you're in G7 diameter mode.

## 1.9.7 Tool Path

### 1.9.7.1 Control Point

The control point for the tool follows the programmed path. The control point is the intersection of a line parallel to the X and Z axis and tangent to the tool tip diameter, as defined when you touch off the X and Z axes for that tool. When turning or facing straight sided parts the cutting path and the tool edge follow the same path. When turning radius and angles the edge of the tool tip will not follow the programmed path unless cutter comp is in effect. In the following figures you can see how the control point does not follow the tool edge as you might assume.

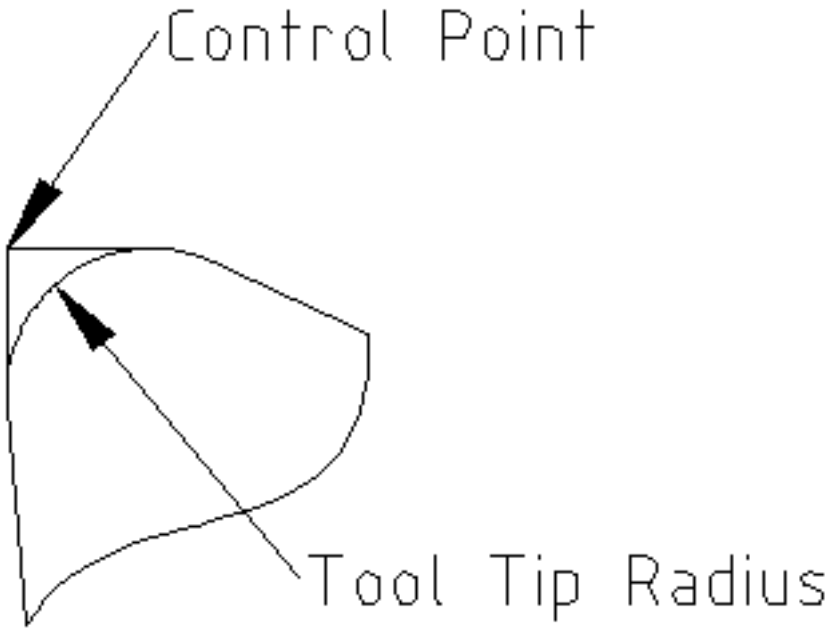



Figure 1.33: Control Point

### 1.9.7.2 Cutting Angles without Cutter Comp

Now imagine we program a ramp without cutter comp. The programmed path is shown in the following figure. As you can see in the figure the programmed path and the desired cut path are one and the same as long as we are moving in an X or Z direction only.

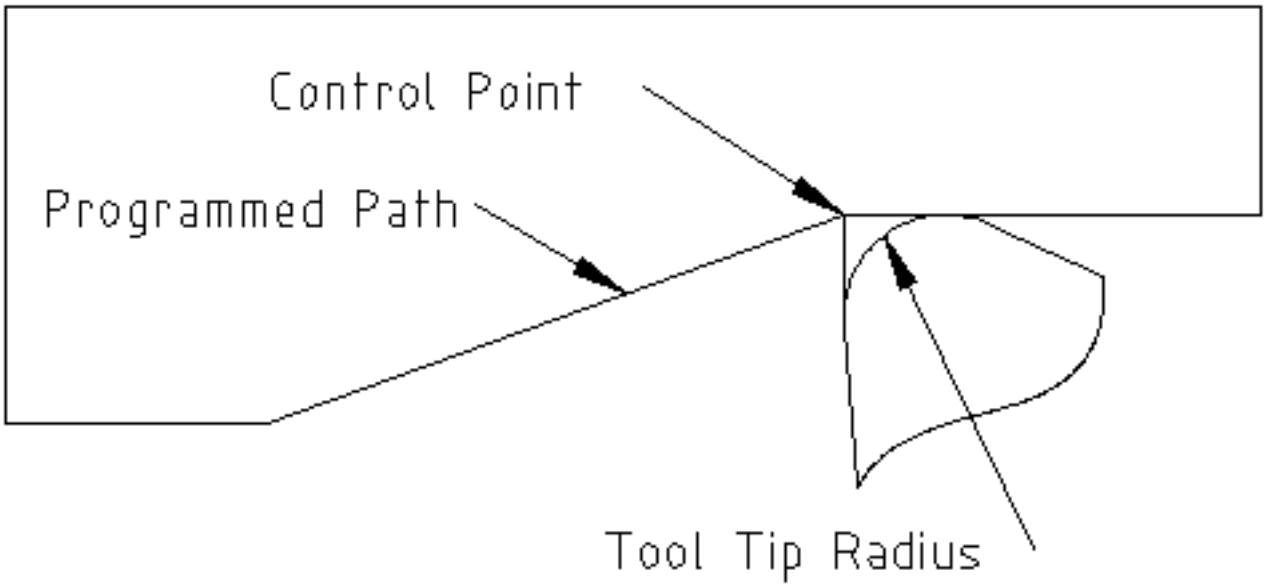



Figure 1.34: Ramp Entry

Now as the control point progresses along the programmed path the actual cutter edge does not follow the programmed path as shown in the following figure. There are two ways to solve this, cutter comp and adjusting your programmed path to compensate for tip radius.

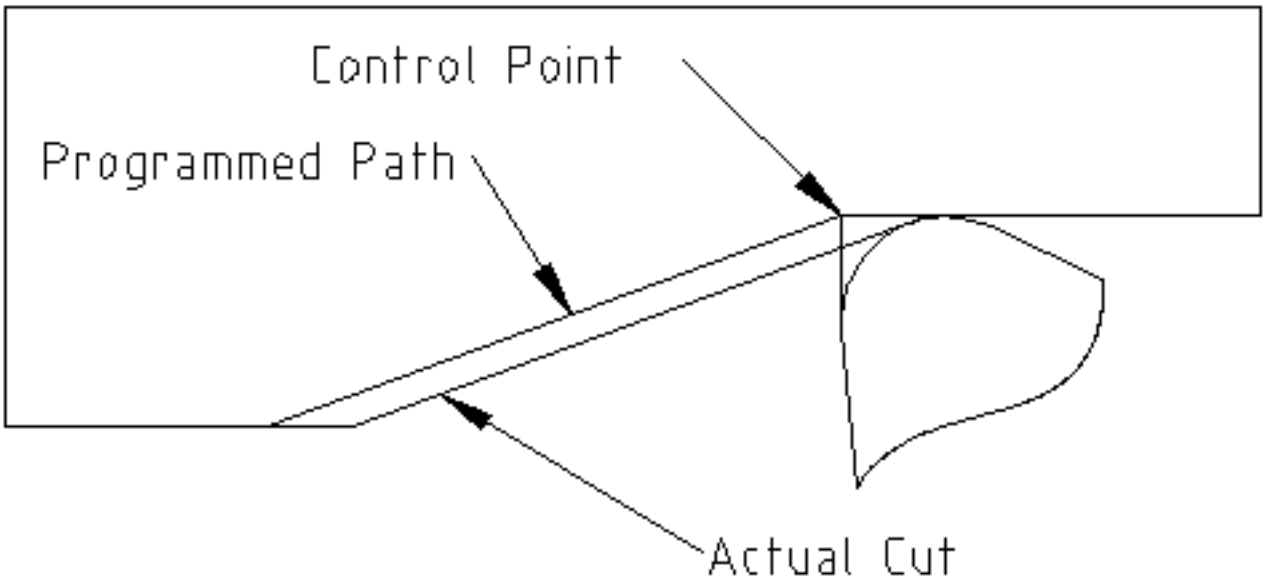



Figure 1.35: Ramp Path

In the above example it is a simple exercise to adjust the programmed path to give the desired actual path by moving the programmed path for the ramp to the left the radius of the tool tip.

#### 1.9.7.3 Cutting a Radius

In this example we will examine what happens during a radius cut without cutter comp. In the next figure you see the tool turning the OD of the part. The control point of the tool is following the programmed path and the tool is touching the OD of the part.

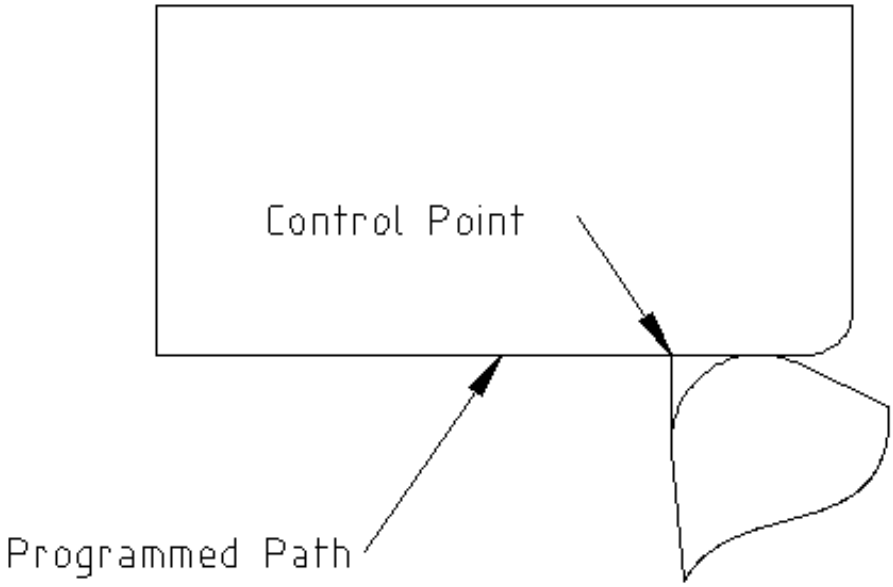



Figure 1.36: Turning Cut

In this next figure you can see as the tool approaches the end of the part the control point still follows the path but the tool tip has left the part and is cutting air. You can also see that even though a radius has been programmed the part will actually end up with a square corner.

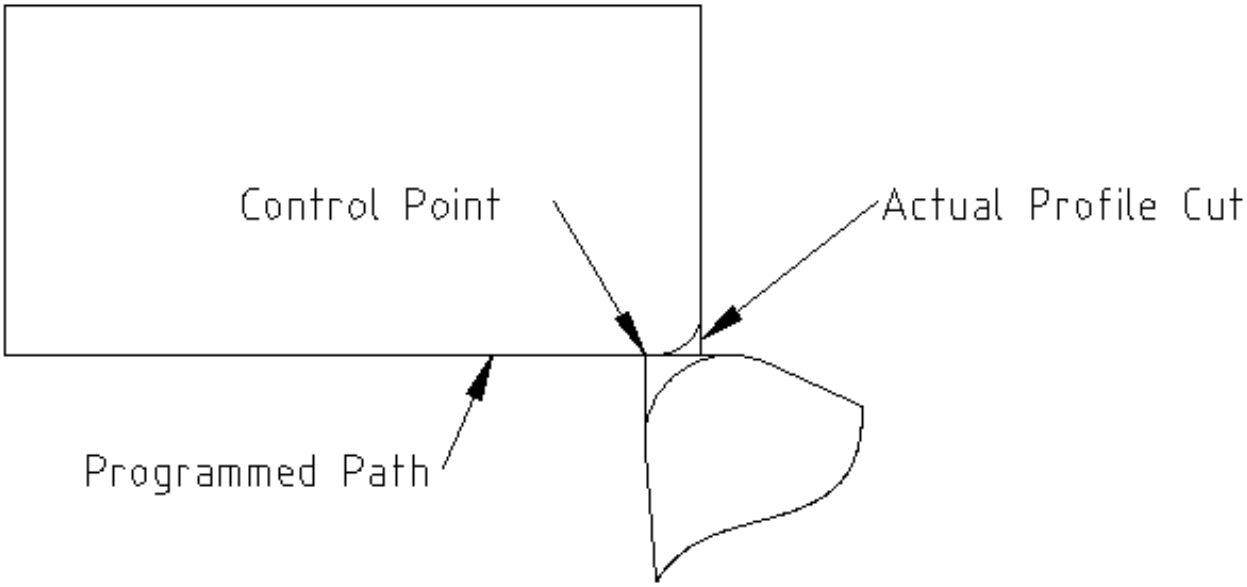



Figure 1.37: Radius Cut

Now you can see as the control point follows the radius programmed the tool tip has left the part and is now cutting air.

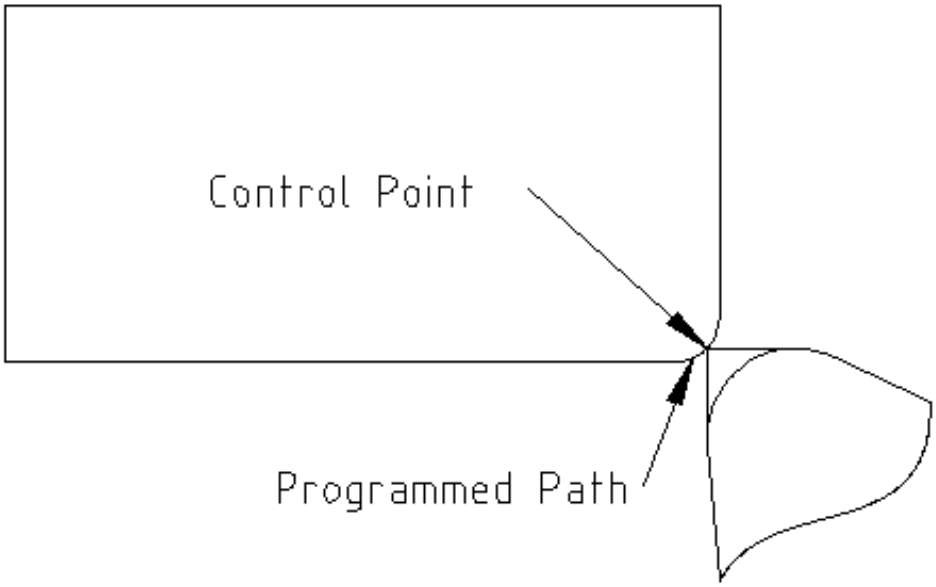



Figure 1.38: Radius Cut

In the final figure we can see the tool tip will finish cutting the face but leave a square corner instead of a nice radius. Notice also that if you program the cut to end at the center of the part a small amount of material will be left from the radius of the tool. To finish a face cut to the center of a part you have to program the tool to go past center at least the nose radius of the tool.

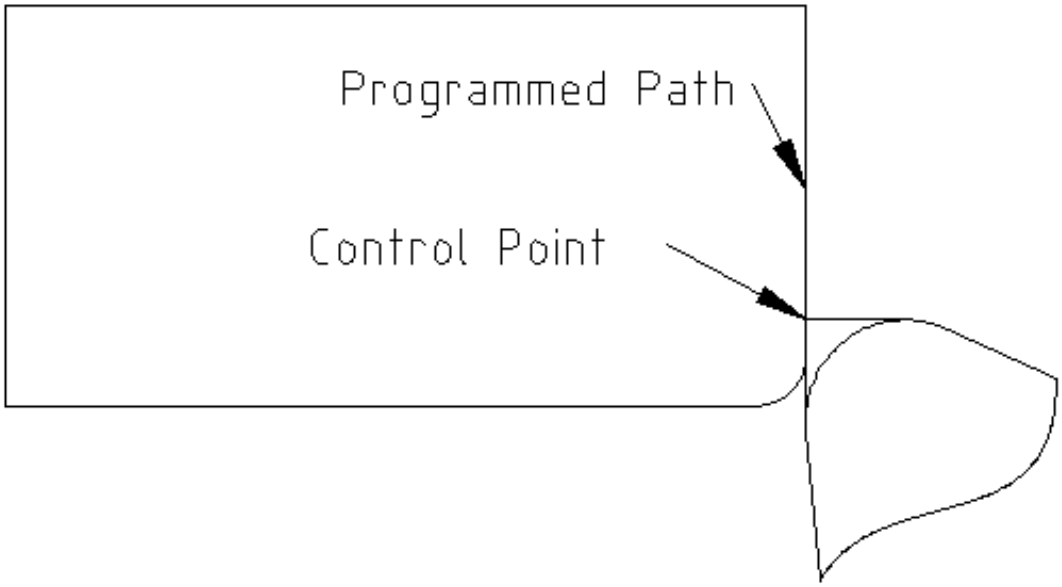



Figure 1.39: Face Cut

#### 1.9.7.4 Using Cutter Comp

When using cutter comp on a lathe think of the tool tip radius as the radius of a round cutter. When using cutter comp the path must be large enough for a round tool that will not gouge into the next line. When cutting straight lines on the lathe you might not want to use cutter comp. For example boring a hole with a tight fitting boring bar you may not have enough room to do the exit move. The entry move into a cutter comp arc is important to get the correct results.

# Chapter 2

# User Interfaces

## 2.1 AXIS GUI

### 2.1.1 Introduction

AXIS is a graphical front-end for LinuxCNC which features a live preview and backplot. It is written in Python and uses Tk and OpenGL to display its user interface.

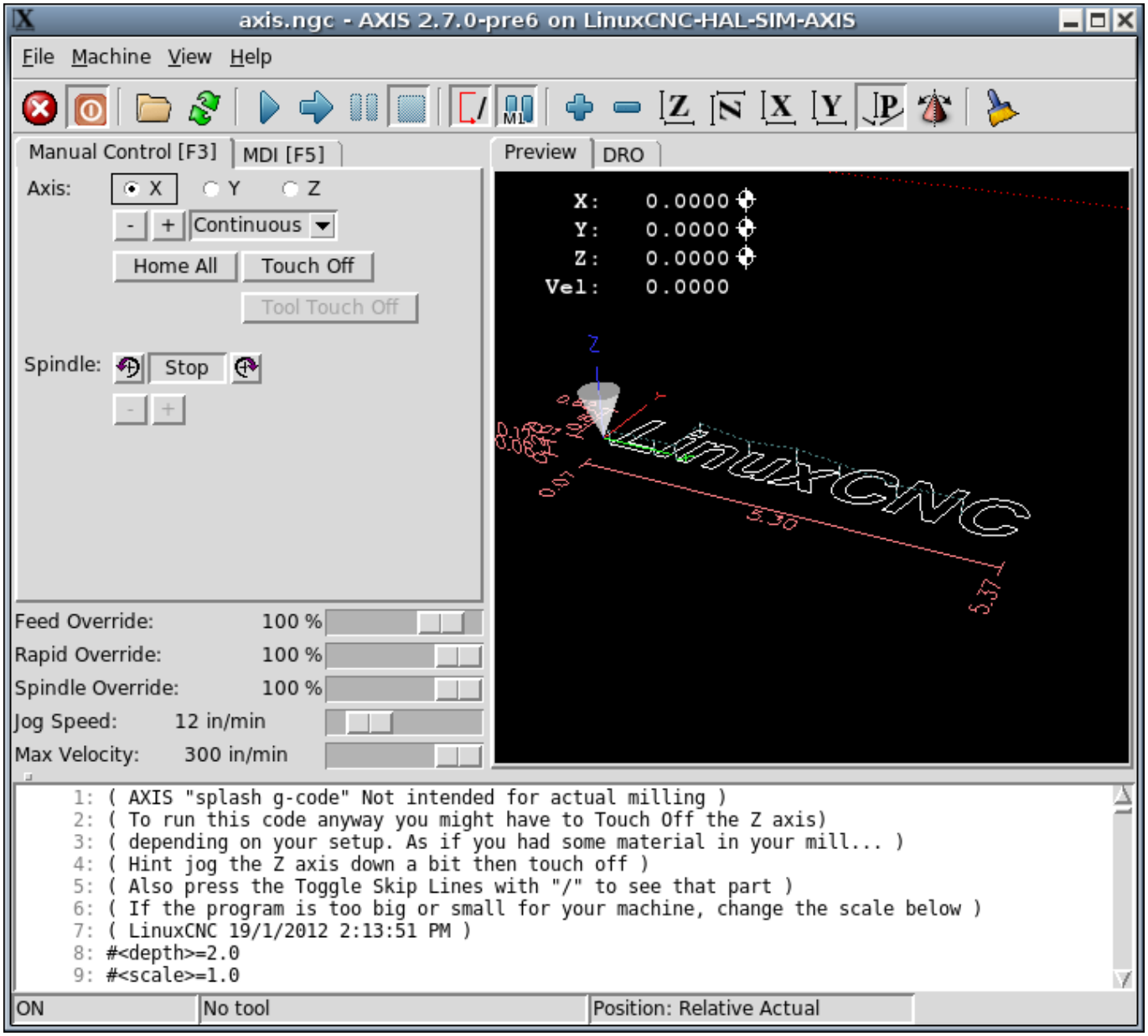



Figure 2.1: AXIS Window

### 2.1.2 Getting Started

If your configuration is not currently set up to use AXIS, you can change it by editing the .ini file. In the section *[DISPLAY]* change the *DISPLAY* line to read *DISPLAY = axis*.

The sample configuration *sim/axis.ini* is already configured to use AXIS as its front-end.

#### 2.1.2.1 A Typical Session

1. Start LinuxCNC.
2. Reset E-STOP (F1) and turn the Machine Power (F2) on.
3. Home all axes.

4. Load the g-code file.
5. Use the preview plot to verify that the program is correct.
6. Load the material.
7. Set the proper offset for each axis by jogging and using the Touch Off button as needed.
8. Run the program.

**Note**
To run the same program again depends on your setup and requirements. You might need to load more material and set offsets or move over and set an offset then run the program again. If your material is fixtured then you might need to only run the program again. See the Machine Menu for more information on the run command.

### 2.1.3 AXIS Display

The AXIS window contains the following elements:

- A display area that shows one of the following:
  - a preview of the loaded file (in this case, *axis.ngc*), as well as the current location of the CNC machine's *controlled point*. Later, this area will display the path the CNC machine has moved through, called the *backplot*
  - a large readout showing the current position and all offsets.
- A menu bar and toolbar that allow you to perform various actions
- *Manual Control Tab* - which allows you to make the machine move, turn the spindle on or off, and turn the coolant on or off if included in the ini file.
- *MDI Tab* - where G-code programs can be entered manually, one line at a time. This also shows the *Active G Codes* which shows which modal G Codes are in effect.
- *Feed Override* - which allows you to scale the speed of programmed motions. The default maximum is 120% and can be set to a different value in the ini file. See the Display Section of the INI file for more information.
- *Spindle Override* - which allows you to scale the spindle speed up or down.
- *Jog Speed* - which allows you to set the jog speed within the limits set in the ini file. See the Display Section of the INI file for more information.
- *Max Velocity* - which allows you to restrict the maximum velocity of all programmed motions (except spindle synchronized motion).
- A text display area that shows the loaded G-Code.
- A status bar which shows the state of the machine. In this screen shot, the machine is turned on, does not have a tool inserted, and the displayed position is *Relative* (showing all offsets), and *Actual* (showing feedback position).

#### 2.1.3.1 Menu Items

Some menu items might be grayed out depending on how you have your .ini file configured. For more information on configuration see the INI Chapter.

FILE MENU

- *Open...* - Opens a standard dialog box to open a g code file to load in AXIS. If you have configured LinuxCNC to use a filter program you can also open it up. See the FILTER Section of the INI configuration for more information.

- *Recent Files* - Displays a list of recently opened files.
- *Edit...* - Open the current G code file for editing if you have an editor configured in your ini file. See the DISPLAY Section for more information on specifying an editor to use.
- *Reload* - Reload the current g code file. If you edited it you must reload it for the changes to take affect. If you stop a file and want to start from the beginning then reload the file. The toolbar reload is the same as the menu.
- *Save gcode as...* - Save the current file with a new name.
- *Properties* - The sum of the rapid and feed moves. Does not factor in acceleration, blending or path mode so time reported will never be less than the actual run time.
- *Edit tool table...* - Same as Edit if you have defined an editor you can open the tool table and edit it.
- *Reload tool table* - After editing the tool table you must reload it.
- *Ladder editor* - If you have loaded Classic Ladder you can edit it from here. See the Classicladder Chapter for more information.
- *Quit* - Terminates the current LinuxCNC session.

### Machine Menu

- *Toggle Emergency Stop F1* - Change the state of the Emergency Stop.
- *Toggle Machine Power F2* - Change the state of the Machine Power if the Emergency Stop is not on.
- *Run Program* - Run the currently loaded program from the beginning.
- *Run From Selected Line* - Select the line you want to start from first. Use with caution as this will move the tool to the expected position before the line first then it will execute the rest of the code.

**Warning**
Do not use *Run From Selected Line* if your g code program contains subroutines.

- *Step* - Single step through a program.
- *Pause* - Pause a program.
- *Resume* - Resume running from a pause.
- *Stop* - Stop a running program. When run is selected after a stop the program will start from the beginning.
- *Stop at M1* - If an M1 is reached, and this is checked, program execution will stop on the M1 line. Press Resume to continue.
- *Skip lines with "/"* - If a line begins with / and this is checked, the line will be skipped.
- *Clear MDI history* - Clears the MDI history window.
- *Copy from MDI history* - Copies the MDI history to the clipboard
- *Paste to MDI history* - Paste from the clipboard to the MDI history window
- *Calibration* - Starts the Servo Axis Calibration assistant. Calibration reads the HAL file and for every *setp* that uses a variable from the ini file that is in an [AXIS_n] section it creates an entry that can be edited and tested.
- *Show HAL Configuration* - Opens the HAL Configuration window where you can monitor HAL Components, Pins, Parameters, Signals, Functions, and Threads.
- *HAL Meter* - Opens a window where you can monitor a single HAL Pin, Signal, or Parameter.

- *HAL Scope* - Opens a virtual oscilloscope that allows plotting HAL values vs. time.
- *Show LinuxCNC Status* - Opens a window showing LinuxCNC's status.
- *Set Debug Level* - Opens a window where debug levels can be viewed and some can be set.
- *Homing* - Home one or all axes.
- *Unhoming* - Unhome one or all axes.
- *Zero Coordinate System* - Set all offsets to zero in the coordinate system chosen.

- *Tool touch off to workpiece* - When performing Touch Off, the value entered is relative to the current workpiece (*G5x*) coordinate system, as modified by the axis offset (*G92*). When the Touch Off is complete, the Relative coordinate for the chosen axis will become the value entered. See G10 L10 in the G code chapter.
- *Tool touch off to fixture* - When performing Touch Off, the value entered is relative to the ninth (*G59.3*) coordinate system, with the axis offset (*G92*) ignored. This is useful when there is a tool touch-off fixture at a fixed location on the machine, with the ninth (*G59.3*) coordinate system set such that the tip of a zero-length tool is at the fixture's origin when the Relative coordinates are 0. See G10 L11 in the G code chapter.

### View Menu

- *Top View* - The Top View (or Z view) displays the G code looking along the Z axis from positive to negative. This view is best for looking at X & Y.
- *Rotated Top View* - The Rotated Top View (or rotated Z view) also displays the G code looking along the Z axis from positive to negative. But sometimes it's convenient to display the X & Y axes rotated 90 degrees to fit the display better. This view is also best for looking at X & Y.
- *Side View* - The Side View (or X view) displays the G code looking along the X axis from positive to negative. This view is best for looking at Y & Z.
- *Front View* - The Front View (or Y view) displays the G code looking along the Y axis from negative to positive. This view is best for looking at X & Z.
- *Perspective View* - The Perspective View (or P view) displays the G code looking at the part from an adjustable point of view, defaulting to X+, Y-, Z+. The position is adjustable using the mouse and the drag/rotate selector. This view is a compromise view, and while it does do a good job of trying to show three (to nine!) axes on a two-dimensional display, there will often be some feature that is hard to see, requiring a change in viewpoint. This view is best when you would like to see all three (to nine) axes at once.

**Point of View**

The AXIS display pick menu *View* refers to *Top*, *Front*, and *Side* views. These terms are correct if the CNC machine has its Z axis vertical, with positive Z up. This is true for vertical mills, which is probably the most popular application, and also true for almost all EDM machines, and even vertical turret lathes, where the part is turning below the tool.

The terms *Top*, *Front*, and *Side* might be confusing however, in other CNC machines, such as a standard lathe, where the Z axis is horizontal, or a horizontal mill, again where the Z axis is horizontal, or even an inverted vertical turret lathe, where the part is turning above the tool, and the Z axis positive direction is down!

Just remember that positive Z axis is (almost) always away from the part. So be familiar with your machine's design and interpret the display as needed.

- *Display Inches* - Set the AXIS display scaling for inches.
- *Display MM* - Set the AXIS display scaling for millimeters.
- *Show Program* - The preview display of the loaded G code program can be entirely disabled if desired.

- *Show Program Rapids* - The preview display of the loaded G code program will always show the feedrate moves (G1,G2,G3) in white. But the display of rapid moves (G0) in cyan can be disabled if desired.
- *Alpha-blend Program* - This option makes the preview of complex programs easier to see, but may cause the preview to display more slowly.
- *Show Live Plot* - The highlighting of the feedrate paths (G1,G2,G3) as the tool moves can be disabled if desired.
- *Show Tool* - The display of the tool cone/cylinder can be disabled if desired.
- *Show Extents* - The display of the extents (maximum travel in each axis direction) of the loaded G code program can be disabled if desired.
- *Show Offsets* - The selected fixture offset (G54-G59.3) origin location can be shown as a set of three orthogonal lines, one each of red, blue, and green. This offset origin (or fixture zero) display can be disabled if desired.
- *Show Machine Limits* - The machine's maximum travel limits for each axis, as set in the ini file, are shown as a rectangular box drawn in red dashed lines. This is useful when loading a new G code program, or when checking for how much fixture offset would be needed to bring the G code program within the travel limits of your machine. It can be shut off if not needed.
- *Show Velocity* - A display of velocity is sometimes useful to see how close your machine is running to its design velocities. It can be disabled if desired.
- *Show Distance to Go* - Distance to go is a very handy item to know when running an unknown G code program for the first time. In combination with the rapid override and feedrate override controls, unwanted tool and machine damage can be avoided. Once the G code program has been debugged and is running smoothly, the Distance to Go display can be disabled if desired.
- *Clear Live Plot* - As the tool travels in the Axis display, the G code path is highlighted. To repeat the program, or to better see an area of interest, the previously highlighted paths can be cleared.
- *Show Commanded Position* - This is the position that LinuxCNC will try to go to. Once motion has stopped, this is the position LinuxCNC will try to hold.
- *Show Actual Position* - Actual Position is the measured position as read back from the system's encoders or simulated by step generators. This may differ slightly from the Commanded Position for many reasons including PID tuning, physical constraints, or position quantization.
- *Show Machine Position* - This is the position in unoffset coordinates, as established by Homing.
- *Show Relative Position* - This is the Machine Position modified by *G5x*, *G92*, and *G43* offsets.

HELP MENU

- *About Axis* - We all know what this is.
- *Quick Reference* - Shows the keyboard shortcut keys.

#### 2.1.3.2 Toolbar buttons

From left to right in the Axis display, the toolbar buttons (keyboard shortcuts shown [in brackets]) are:

- Toggle Emergency Stop [F1] (also called E-Stop)
- Toggle Machine Power [F2]
- Open G Code file [O]

- Reload current file [Ctrl-R]
- Begin executing the current file [R]
- Execute next line [T]
- Pause Execution [P] Resume Execution[S]
- Stop Program Execution [ESC]
- Toggle Skip lines with "/" [Alt-M-/]
- Toggle Optional Pause [Alt-M-1]
- Zoom In
- Zoom Out
- Top view
- Rotated Top view
- Side view
- Front view
- Perspective view
- Toggle between Drag and Rotate Mode [D]
- Clear live backplot [Ctrl-K]

#### 2.1.3.3 Graphical Display Area

**Coordinate Display** In the upper-left corner of the program display is the coordinate position display for each axis. To the right of the number an origin symbol is shown if the axis has been homed.

A limit symbol is shown on the right side of the coordinate position number if the axis is on one of its limit switches.

To properly interpret the coordinate position numbers, refer to the *Position:* indicator in the status bar. If the position is *Machine Actual*, then the displayed number is in the machine coordinate system. If it is *Relative Actual*, then the displayed number is in the offset coordinate system. When the coordinates displayed are relative and an offset has been set, the display will include a cyan machine origin marker.

If the position is *Commanded*, then the exact coordinate given in a G code command is displayed. If it is *Actual*, then it is the position the machine has actually moved to. These values can be different from commanded postition due to following error, dead band, encoder resolution, or step size. For instance, if you command a movement to X 0.0033 on your mill, but one step of your stepper motor or one encoder count is 0.00125, then the *Commanded* position might be 0.0033, but the *Actual* position will be 0.0025 (2 steps) or 0.00375 (3 steps).

**Preview Plot** When a file is loaded, a preview of it is shown in the display area. Fast moves (such as those produced by the *G0* command) are shown as cyan lines. Moves at a feed rate (such as those produced by the *G1* command) are shown as solid white lines. Dwells (such as those produced by the *G4* command) are shown as small pink *X* marks.

G0 (Rapid) moves prior to a feed move will not show on the preview plot. Rapid moves after a T<n> (Tool Change) will not show on the preview until after the first feed move. To turn either of these features off program a G1 without any moves prior to the G0 moves.

**Program Extents** The *extents* of the program in each axis are shown. At the ends, the least and greatest coordinate values are indicated. In the middle, the difference between the coordinates is shown.

When some coordinates exceed the *soft limits* in the .ini file, the relevant dimension is shown in a different color and enclosed by a box. In figure below the maximum soft limit is exceeded on the X axis as indicated by the box surrounding the coordinate value. The minimum X travel of the program is -1.95, the maximum X travel is 1.88, and the program requires 3.83 inches of X travel. To move the program so it's within the machine's travel in this case, jog to the left and Touch Off X again.

**Tool Cone** When no tool is loaded, the location of the tip of the tool is indicated by the *tool cone*. The *tool cone* does not provide guidance on the form, length, or radius of the tool.

When a tool is loaded (for instance, with the MDI command *T1 M6* ), the cone changes to a cylinder which shows the diameter of the tool given in the tool table file.

**Backplot** When the machine moves, it leaves a trail called the backplot. The color of the line indicates the type of motion: Yellow for jogs, faint green for rapid movements, red for straight moves at a feed rate, and magenta for circular moves at a feed rate.

**Grid** Axis can optionally display a grid when in orthogonal views. Enable or disable the grid using the *Grid* menu under *View*. When enabled, the grid is shown in the top and rotated top views; when coordinate system is not rotated, the grid is shown in the front and side views as well. The presets in the *Grid* menu are controlled by the inifile item `[DISPLAY]GRIDS`; if unspecified, the default is `10mm 20mm 50mm 100mm 1in 2in 5in 10in`.

Specifying a very small grid may decrease performance.

**Interacting** By left-clicking on a portion of the preview plot, the line will be highlighted in both the graphical and text displays. By left-clicking on an empty area, the highlighting will be removed.

By dragging with the left mouse button pressed, the preview plot will be shifted (panned).

By dragging with shift and the left mouse button pressed, or by dragging with the mouse wheel pressed, the preview plot will be rotated. When a line is highlighted, the center of rotation is the center of the line. Otherwise, the center of rotation is the center of the entire program.

By rotating the mouse wheel, or by dragging with the right mouse button pressed, or by dragging with control and the left mouse button pressed, the preview plot will be zoomed in or out.

By clicking one of the *Preset View* icons, or by pressing *V*, several preset views may be selected.

#### 2.1.3.4 Text Display Area

By left-clicking a line of the program, the line will be highlighted in both the graphical and text displays.

When the program is running, the line currently being executed is highlighted in red. If no line has been selected by the user, the text display will automatically scroll to show the current line.

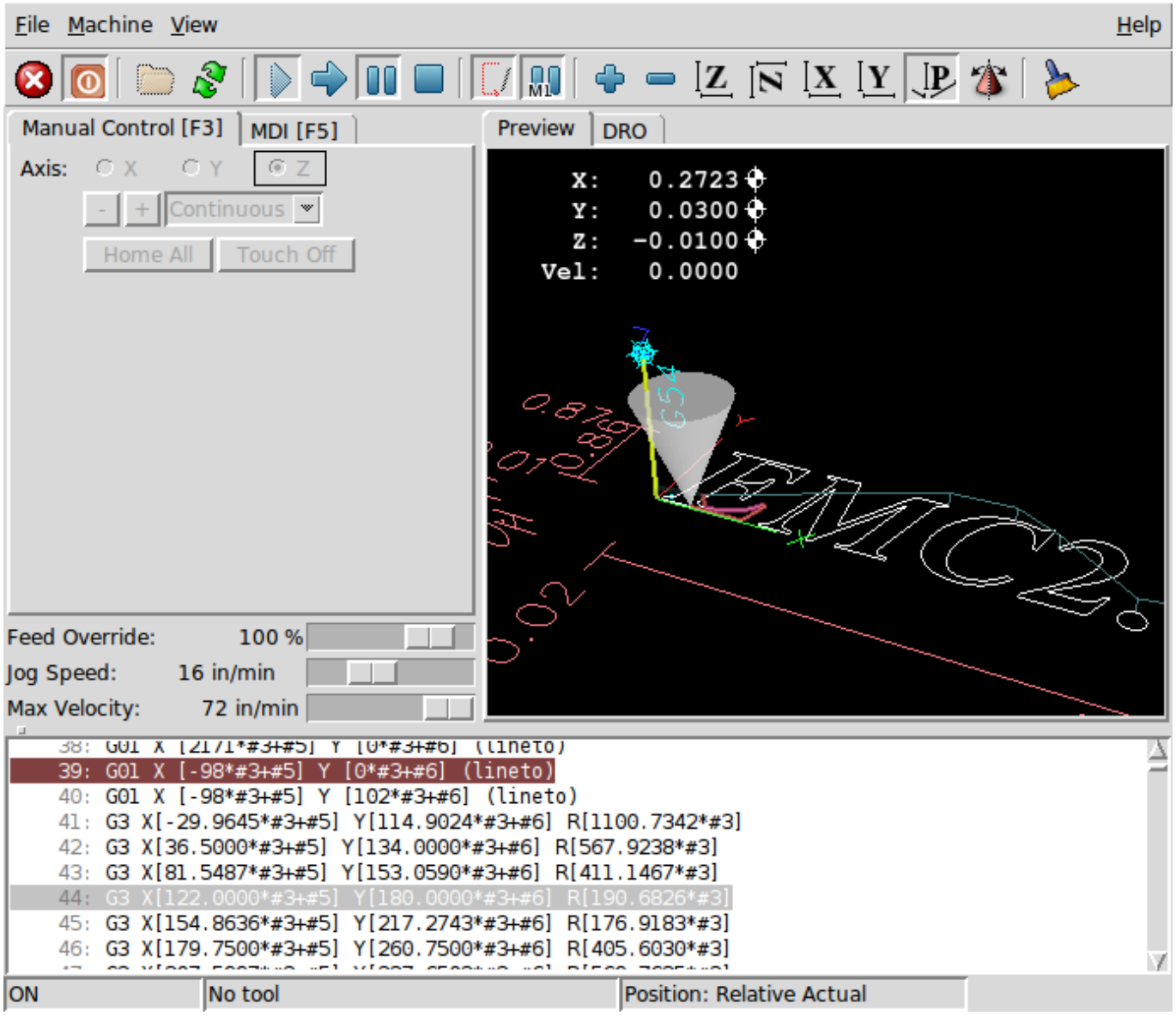



Figure 2.2: Current and Selected Lines

#### 2.1.3.5 Manual Control

While the machine is turned on but not running a program, the items in the *Manual Control* tab can be used to move the machine or control its spindle and coolant.

When the machine is not turned on, or when a program is running, the manual controls are unavailable.

Many of the items described below are not useful on all machines. When AXIS detects that a particular pin is not connected in HAL, the corresponding item in the Manual Control tab is removed. For instance, if the HAL pin *motion.spindle-brake* is not

connected, then the *Brake* button will not appear on the screen. If the environment variable *AXIS_NO_AUTOCONFIGURE* is set, this behavior is disabled and all the items will appear.

**The Axis group** *Axis* allows you to manually move the machine. This action is known as *jogging*. First, select the axis to be moved by clicking it. Then, click and hold the + or - button depending on the desired direction of motion. The first four axes can also be moved by the arrow keys (X and Y), PAGE UP and PAGE DOWN keys (Z), and the [ and ] keys (A).

If *Continuous* is selected, the motion will continue as long as the button or key is pressed. If another value is selected, the machine will move exactly the displayed distance each time the button is clicked or the key is pressed. By default, the available values are *0.1000, 0.0100, 0.0010, 0.0001*

See the DISPLAY Section for more information on setting the increments.

**Homing** If your machine has home switches and a homing sequence defined for all axes the button will read *Home All*. The *Home All* button or the Ctrl-HOME key will home all axes using the homing sequence. Pressing the HOME key will home the current axis, even if a homing sequence is defined.

If your machine has home switches and no homing sequence is defined or not all axes have a homing sequence the button will read *Home* and will home the selected axis only. Each axis must be selected and homed separately.

If your machine does not have home switches defined in the configuration the *Home* button will set the current selected axis current position to be the absolute position 0 for that axis and will set the *is-homed* bit for that axis.

See the Homing Configuration Chapter for more information.

**Touch Off** By pressing *Touch Off* or the END key, the *G54 offset* for the current axis is changed so that the current axis value will be the specified value. Expressions may be entered using the rules for rs274ngc programs, except that variables may not be referred to. The resulting value is shown as a number.

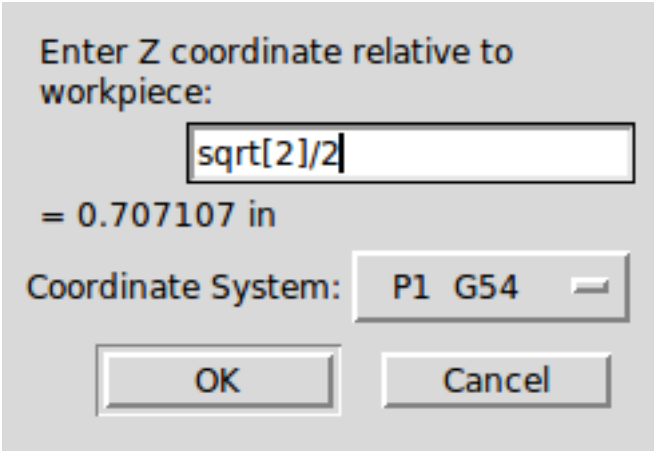



Figure 2.3: Touch Off

See also the *Tool touch off to workpiece* and *Tool touch off to fixture* options in the Machine menu.

**Override Limits** By pressing Override Limits, the machine will temporarily be allowed to jog off of a physical limit switch. This check box is only available when a limit switch is tripped. The override is reset after one jog. If the axis is configured with separate positive and negative limit switches, LinuxCNC will allow the jog only in the correct direction. *Override Limits will not allow a jog past a soft limit. The only way to disable a soft limit on an axis is to Unhome it.*

**The Spindle group** The buttons on the first row select the direction for the spindle to rotate: Counterclockwise, Stopped, Clockwise. Counterclockwise will only show up if the pin *motion.spindle-reverse* is in the HAL file (it can be *net trick-axis motion.spindle-reverse* ). The buttons on the next row increase or decrease the rotation speed. The checkbox on the third row allows the spindle brake to be engaged or released. Depending on your machine configuration, not all the items in this group may appear. Pressing the spindle start button sets the *S* speed to 1.

**The Coolant group** The two buttons allow the *Mist* and *Flood* coolants to be turned on and off. Depending on your machine configuration, not all the items in this group may appear.

#### 2.1.3.6 MDI

MDI allows G-code commands to be entered manually. When the machine is not turned on, or when a program is running, the MDI controls are unavailable.

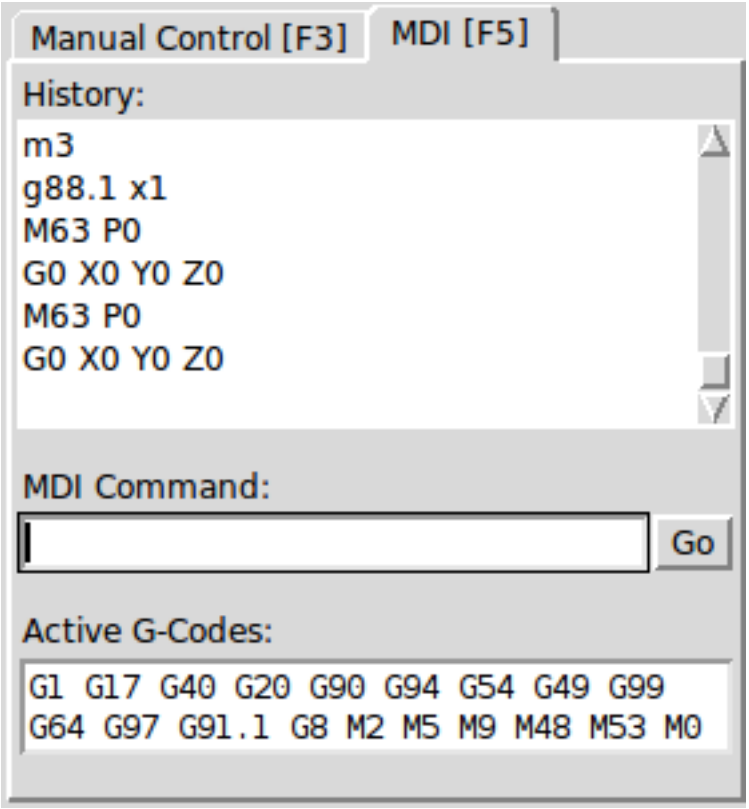



Figure 2.4: The MDI tab

- *History* - This shows MDI commands that have been typed earlier in this session.
- *MDI Command* - This allows you to enter a g-code command to be executed. Execute the command by pressing Enter or by clicking *Go*.
- *Active G-Codes* - This shows the *modal codes* that are active in the interpreter. For instance, *G54* indicates that the *G54 offset* is applied to all coordinates that are entered. When in Auto the Active G-Codes represent the codes after any read ahead by the interpreter.

#### 2.1.3.7 Feed Override

By moving this slider, the programmed feed rate can be modified. For instance, if a program requests *F60* and the slider is set to 120%, then the resulting feed rate will be 72.

#### 2.1.3.8 Spindle Speed Override

By moving this slider, the programmed spindle speed can be modified. For instance, if a program requests S8000 and the slider is set to 80%, then the resulting spindle speed will be 6400. This item only appears when the HAL pin *motion.spindle-speed-out* is connected.

#### 2.1.3.9 Jog Speed

By moving this slider, the speed of jogs can be modified. For instance, if the slider is set to 1 in/min, then a .01 inch jog will complete in about .6 seconds, or 1/100 of a minute. Near the left side (slow jogs) the values are spaced closely together, while near the right side (fast jogs) they are spaced much further apart, allowing a wide range of jog speeds with fine control when it is most important.

On machines with a rotary axis, a second jog speed slider is shown. This slider sets the jog rate for the rotary axes (A, B and C).

#### 2.1.3.10 Max Velocity

By moving this slider, the maximum velocity can be set. This caps the maximum velocity for all programmed moves except spindle-synchronized moves.

### 2.1.4 Keyboard Controls

Almost all actions in AXIS can be accomplished with the keyboard. A full list of keyboard shortcuts can be found in the AXIS Quick Reference, which can be displayed by choosing Help > Quick Reference. Many of the shortcuts are unavailable when in MDI mode.

**Feed Override Keys** The Feed Override keys behave differently when in Manual Mode. The keys '12345678 will select an axis if it is programed. If you have 3 axis then ' will select axis 0, 1 will select axis 1, and 2 will select axis 2. The remainder of the number keys will still set the Feed Override. When running a program '1234567890 will set the Feed Override to 0% - 100%.

The most frequently used keyboard shortcuts are shown in the following Table

Table 2.1: Most Common Keyboard Shortcuts

| Keystroke | Action Taken | Mode |
|---|---|---|
| F1 | Toggle Emergency Stop | Any |
| F2 | Turn machine on/off | Any |
| `, 1 .. 9, 0 | Set feed override from 0% to 100% | Varies |
| X, ` | Activate first axis | Manual |
| Y, 1 | Activate second axis | Manual |
| Z, 2 | Activate third axis | Manual |
| A, 3 | Activate fourth axis | Manual |
| I | Select jog increment | Manual |
| C | Continuous jog | Manual |
| Control-Home | Perform homing sequence | Manual |
| End | Touch off: Set G54 offset for active axis | Manual |
| Left, Right | Jog first axis | Manual |
| Up, Down | Jog second axis | Manual |
| Pg Up, Pg Dn | Jog third axis | Manual |
| [, ] | Jog fourth axis | Manual |
| O | Open File | Manual |
| Control-R | Reload File | Manual |
| R | Run file | Manual |
| P | Pause execution | Auto |
| S | Resume Execution | Auto |
| ESC | Stop execution | Auto |
| Control-K | Clear backplot | Auto/Manual |
| V | Cycle among preset views | Auto/Manual |
| Shift-Left,Right | Rapid X Axis | Manual |
| Shift-Up,Down | Rapid Y Axis | Manual |

Table 2.1: (continued)

| Keystroke | Action Taken | Mode |
|---|---|---|
| Shift-PgUp, PgDn | Rapid Z Axis | Manual |
| @ | toggle Actual/Commanded | Any |
| # | toggle Relative/Machine | Any |

### 2.1.5 Show LinuxCNC Status (linuxcnctop)

AXIS includes a program called *linuxcnctop* which shows some of the details of LinuxCNC's state. You can run this program by invoking Machine > Show LinuxCNC Status

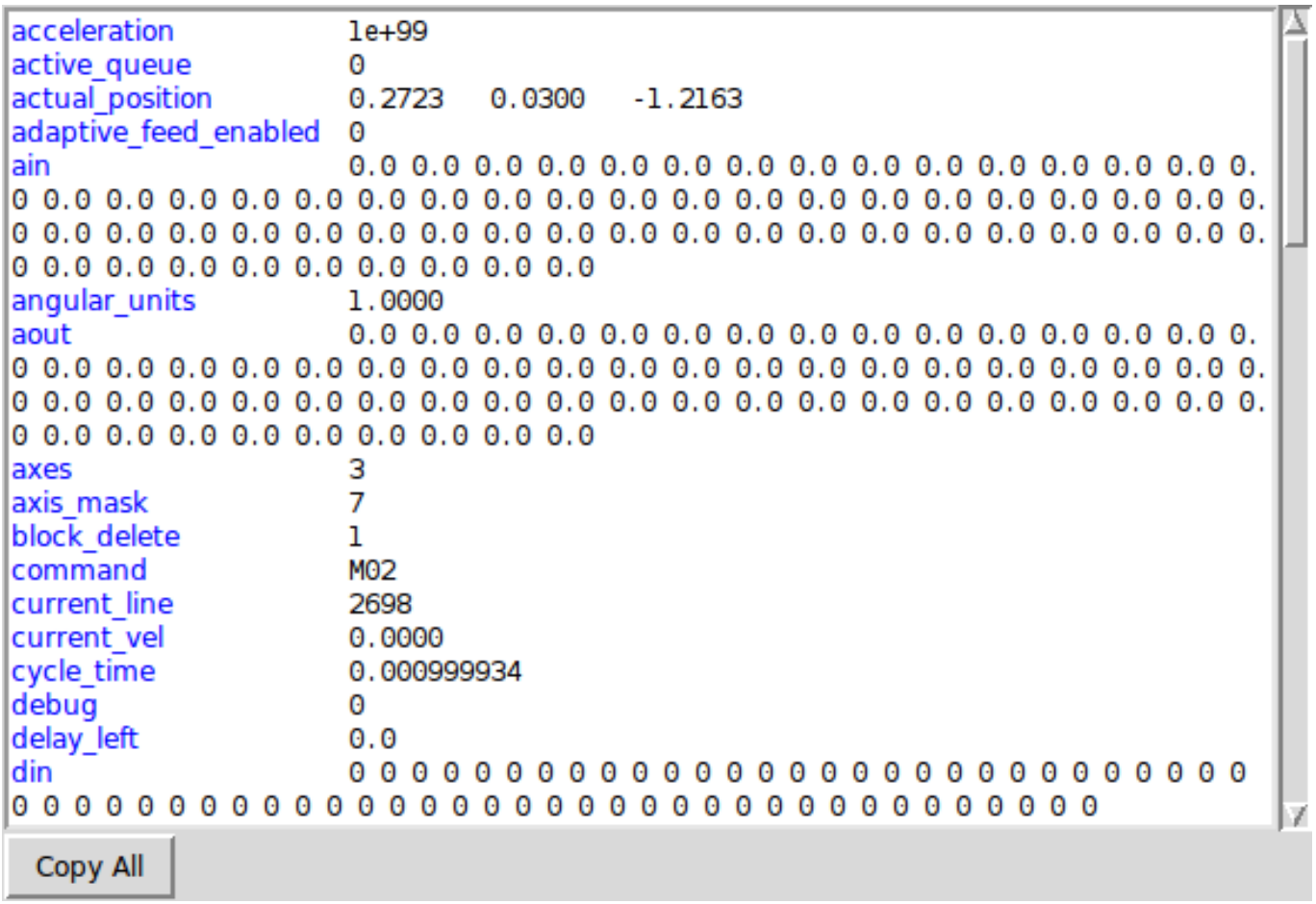



Figure 2.5: LinuxCNC Status Window

The name of each item is shown in the left column. The current value is shown in the right column. If the value has recently changed, it is shown on a red background.

### 2.1.6 MDI interface

AXIS includes a program called `mdi` which allows text-mode entry of MDI commands to a running LinuxCNC session. You can run this program by opening a terminal and typing

```
mdi
```

Once it is running, it displays the prompt *MDI>*. When a blank line is entered, the machine's current position is shown. When a command is entered, it is sent to LinuxCNC to be executed.

This is a sample session of mdi.

```
$ mdi
MDI>
(0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
MDI> G1 F5 X1
MDI>
(0.5928500000000374, 0.0, 0.0, 0.0, 0.0, 0.0)
MDI>
(1.0000000000000639, 0.0, 0.0, 0.0, 0.0, 0.0)
```

### 2.1.7 axis-remote

AXIS includes a program called *axis-remote* which can send certain commands to a running AXIS. The available commands are shown by running *axis-remote --help* and include checking whether AXIS is running (*--ping*), loading a file by name, reloading the currently loaded file (*--reload*), and making AXIS exit (*--quit*).

### 2.1.8 Manual Tool Change

LinuxCNC includes a userspace HAL component called *hal_manualtoolchange*, which shows a window prompt telling you what tool is expected when a *M6* command is issued. After the OK button is pressed, execution of the program will continue.

The hal_manualtoolchange component includes a hal pin for a button that can be connected to a physical button to complete the tool change and remove the window prompt (hal_manualtoolchange.change_button).

The HAL configuration file *configs/sim/axis_manualtoolchange.hal* shows the HAL commands necessary to use this component.

hal_manualtoolchange can be used even when AXIS is not used as the GUI. This component is most useful if you have presettable tools and you use the tool table.

**Note**
Important Note: Rapids will not show on the preview after a T<n> is issued until the next feed move after the M6. This can be very confusing to most users. To turn this feature off for the current tool change program a G1 with no move after the T<n>.

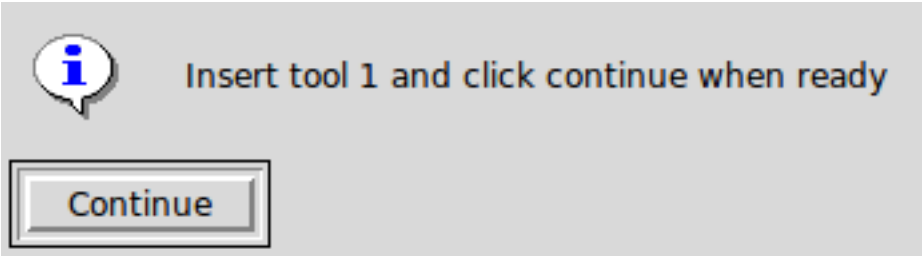



Figure 2.6: The Manual Toolchange Window

### 2.1.9 Python modules

AXIS includes several Python modules which may be useful to others. For more information on one of these modules, use *pydoc <module name>* or read the source code. These modules include:

- *emc* provides access to the LinuxCNC command, status, and error channels
- *gcode* provides access to the rs274ngc interpreter
- *rs274* provides additional tools for working with rs274ngc files
- *hal* allows the creation of userspace HAL components written in Python
- *_togl* provides an OpenGL widget that can be used in Tkinter applications
- *minigl* provides access to the subset of OpenGL used by AXIS

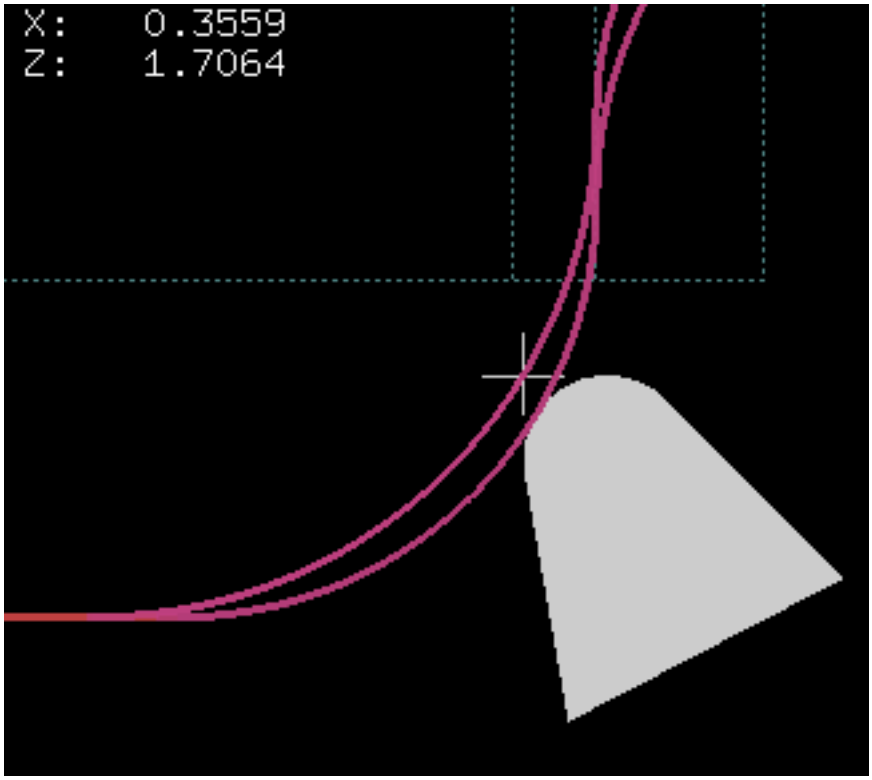
To use these modules in your own scripts, you must ensure that the directory where they reside is on Python's module path. When running an installed version of LinuxCNC, this should happen automatically. When running *in-place*, this can be done by using *scripts/rip-environment*.

### 2.1.10 Using AXIS in Lathe Mode

By including the line *LATHE = 1* in the [DISPLAY] section of the ini file, AXIS selects lathe mode. The *Y* axis is not shown in coordinate readouts, the view is changed to show the Z axis extending to the right and the X axis extending towards the bottom of the screen, and several controls (such as those for preset views) are removed. The coordinate readouts for X are replaced with diameter and radius.

Pressing *V* zooms out to show the entire file, if one is loaded.

When in lathe mode, the shape of the loaded tool (if any) is shown.




**Lathe Tool Shape**

### 2.1.11 Advanced Configuration

When AXIS is started it creates the HAL pins for the GUI then it executes the HAL file named in *[HAL]POSTGUI_HALFILE* in the ini file. Only one post GUI file may be used. Place all HAL commands that connect to the GUI HAL pins in the post gui HAL file.

For more information on ini file settings that can change how AXIS works see the Display Section of the INI Configuration Chapter.

#### 2.1.11.1 Program Filters

AXIS has the ability to send loaded files through a *filter program*. This filter can do any desired task: Something as simple as making sure the file ends with *M2*, or something as complicated as generating G-Code from an image.

The *[FILTER]* section of the ini file controls how filters work. First, for each type of file, write a *PROGRAM_EXTENSION* line. Then, specify the program to execute for each type of file. This program is given the name of the input file as its first argument, and must write rs274ngc code to standard output. This output is what will be displayed in the text area, previewed in the display area, and executed by LinuxCNC when *Run*. The following lines add support for the *image-to-gcode* converter included with LinuxCNC:

```
[FILTER]
PROGRAM_EXTENSION = .png,.gif Greyscale Depth Image
png = image-to-gcode
gif = image-to-gcode
```

It is also possible to specify an interpreter:

```
PROGRAM_EXTENSION = .py Python Script
py = python
```

In this way, any Python script can be opened, and its output is treated as g-code. One such example script is available at *nc_files/holecircle.py*. This script creates g-code for drilling a series of holes along the circumference of a circle.

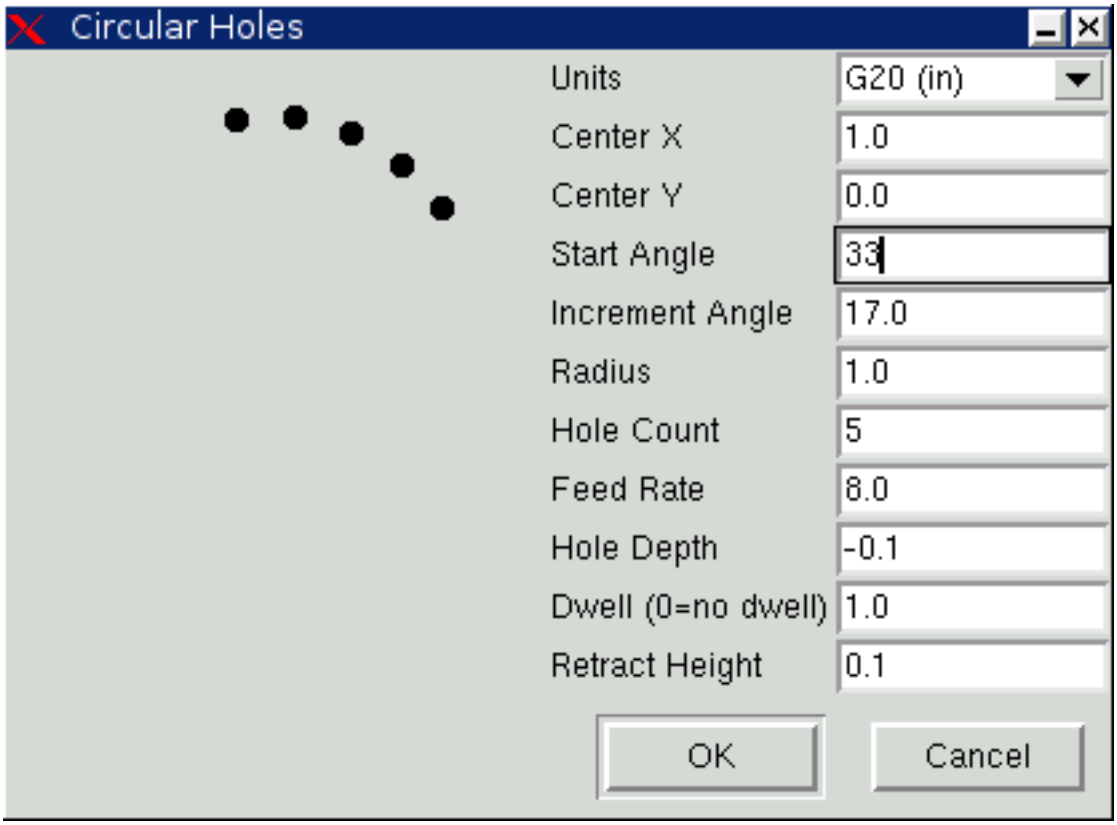



Figure 2.7: Circular Holes

If the environment variable AXIS_PROGRESS_BAR is set, then lines written to stderr of the form

```
FILTER_PROGRESS=%d
```

will set the AXIS progress bar to the given percentage. This feature should be used by any filter that runs for a long time.

#### 2.1.11.2 The X Resource Database

The colors of most elements of the AXIS user interface can be customized through the X Resource Database. The sample file *axis_light_background* changes the colors of the backplot window to a *dark lines on white background* scheme, and also serves as a reference for the configurable items in the display area. The sample file *axis_big_dro* changes the position readout to a larger size font. To use these files:

```
xrdb -merge /usr/share/doc/emc2/axis_light_background

xrdb -merge /usr/share/doc/emc2/axis_big_dro
```

For information about the other items which can be configured in Tk applications, see the Tk man pages.

Because modern desktop environments automatically make some settings in the X Resource Database that adversely affect AXIS, by default these settings are ignored. To make the X Resource Database items override AXIS defaults, include the following line in your X Resources:

```
    *Axis*optionLevel: widgetDefault
```

this causes the built-in options to be created at the option level *widgetDefault*, so that X Resources (which are level *userDefault*) can override them.

#### 2.1.11.3 ~/.axisrc

If it exists, the contents of `~/.axisrc` are executed as Python source code just before the AXIS GUI is displayed. The details of what may be written in the `~/.axisrc` are subject to change during the development cycle.

The following adds Control-Q as a keyboard shortcut for Quit.

```
root_window.bind("<Control-q>", "destroy .")
help2.append(("Control-Q", "Quit"))
```

The following stops the "Do you really want to quit" dialog.

```
root_window.tk.call("wm","protocol",".","WM_DELETE_WINDOW","destroy .")
```

#### 2.1.11.4 USER_COMMAND_FILE

A configuration-specific python file may be specified with an ini file setting *[DISLAY]USER_COMMAND_FILE=filename.py*. Like a `~/.axisrc` file, this file is sourced just before the AXIS GUI is displayed. This file is specific to an ini file configuration not the user's home directory. When this file is specified, an existing `~/.axisrc` file is ignored.

#### 2.1.11.5 user_live_update()

The axis gui includes a no-op (placeholder) function named *user_live_update()* that is executed at the conclusion of the update() function of its LivePlotter class. This function may be implemented within a `~/.axisrc` python script or a *[DISPLAY]USER_COMMAND_FILE* python script to make custom, periodic actions. The details of what may be accomplished in this function are dependent on the axis gui implementation and subject to change during the development cycle.

#### 2.1.11.6 External Editor

The menu options File > Edit... and File > Edit Tool Table... become available after defining the editor in the ini section [DISPLAY]. Useful values include EDITOR=gedit and EDITOR=gnome-terminal -e vim. For more information, see the Display Section of the INI Configuration Chapter.

#### 2.1.11.7 Virtual Control Panel

AXIS can display a custom virtual control panel in the right-hand pane. You can program buttons, indicators, data displays and more. For more information, see the PyVCP Chapter and the GladeVCP Chapter.

#### 2.1.11.8 Preview Control

Special comments can be inserted into the G Code file to control how the preview of AXIS behaves. In the case where you want to limit the drawing of the preview use these special comments. Anything between the (AXIS,hide) and (AXIS,show) will not be drawn during the preview. The (AXIS,hide) and (AXIS,show) must be used in pairs with the (AXIS,hide) being first. Anything after a (AXIS,stop) will not be drawn during the preview.

These comments are useful to unclutter the preview display (for instance while debugging a larger g-code file, one can disable the preview on certain parts that are already working OK).

- (AXIS,hide) Stops the preview (must be first)
- (AXIS,show) Resumes the preview (must follow a hide)
- (AXIS,stop) Stops the preview from here to the end of the file.
- (AXIS,notify,the_text) Displays the_text as an info display This display can be useful in the Axis preview when (debug,message) comments are not displayed.

#### 2.1.11.9 Axisui Pins

To improve the interaction of AXIS with physical jog wheels, the axis currently selected in the GUI is also reported on a pin with a name like *axisui.jog.x*. One of these pins is *TRUE* at one time, and the rest are *FALSE*. These are meant to control motion's jog-enable pins.

**Axisui Pins** Axis has Hal pins to indicate which jog radio button is selected in the *Manual Control* tab.

```
Type  Dir    Name
bit   OUT    axisui.jog.x
bit   OUT    axisui.jog.y
bit   OUT    axisui.jog.z
bit   OUT    axisui.jog.a
bit   OUT    axisui.jog.b
bit   OUT    axisui.jog.c
bit   OUT    axisui.jog.u
bit   OUT    axisui.jog.v
bit   OUT    axisui.jog.w
```

Axis has a Hal pin to indicate the jog increment selected on the *Manual Tab*.

```
Type  Dir    Name
float OUT               axisui.jog.increment
```

Axis has Hal input pins to clear the pop up notifications for errors and information.

```
Type  Dir    Name
bit   IN     axisui.notifications-clear
bit   IN     axisui.notifications-clear-error
bit   IN     axisui.notifications-clear-info
```

Axis has a Hal input pin that disables/enables the *Pause/Resume* funtion.

```
Type  Dir    Name
bit   IN     axisui.resume-inhibit
```

## 2.2 GMOCCAPY

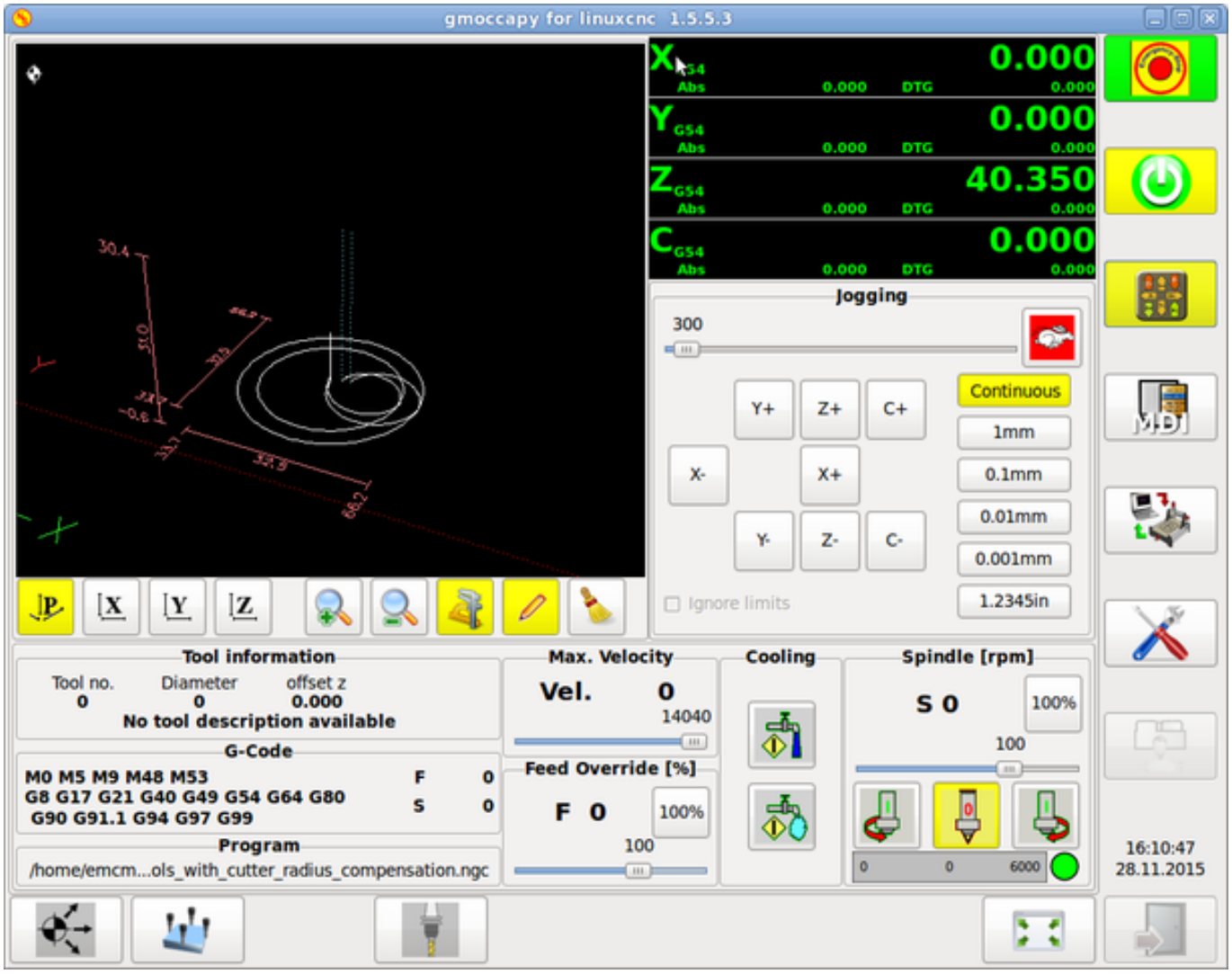
### 2.2.1 Introduction

*GMOCCAPY* is a GUI for LinuxCNC, designed to be used with a touch screen, but can also be used on normal screens with a mouse or hardware buttons and MPG wheels, as it presents HAL Pins for the most common needs. Please find more information in the following.

It offers the possibility to display up to 4 axis, support a lathe mode for normal and back tool lathe and can be adapted to nearly every need, because gmoccapy supports embedded tabs and side panels. As a good example for that see gmoccapy_plasma

It has support for integrated virtual keyboard (onboard or matchbox-keyboard), so there is no need for a hardware keyboard or mouse, but it can also be used with that hardware. Gmoccapy offers a separate settings page to configure most settings of the GUI without editing files.

*GMOCCAPY* can be localized very easy, because the corresponding files are separated from the linuxcnc.po files, so there is no need to translate unneeded stuff. The files are placed in **/src/po/gmoccapy**. Just copy the gmoccapy.pot file to something like fr.po and translate that file with gtranslator or poedit. After a make you got the GUI in your preference language. Please publish your translation, so it can be included in the official packages and be published to other users. At the Moment it is available in English, German, Spanish, Polish, Serbian and Hungarian. Feel free to help me to introduce more languages, **nieson@web.de**. If you need help, don't hesitate to ask.



### 2.2.2 Requirements

Gmoccapy has been tested on Ubuntu 10.04 and 12.04 and Debian Wheezy, with LinuxCNC 2.6 ,2.7, and master, if you use other versions, please inform about problems or solutions on the LinuxCNC forum or the German CNC Ecke Forum or LinuxCNC users mailing list

The minimum screen resolution for gmoccapy, using it without side panels is **979 x 750 Pixel**, so it should fit to every standard screen.

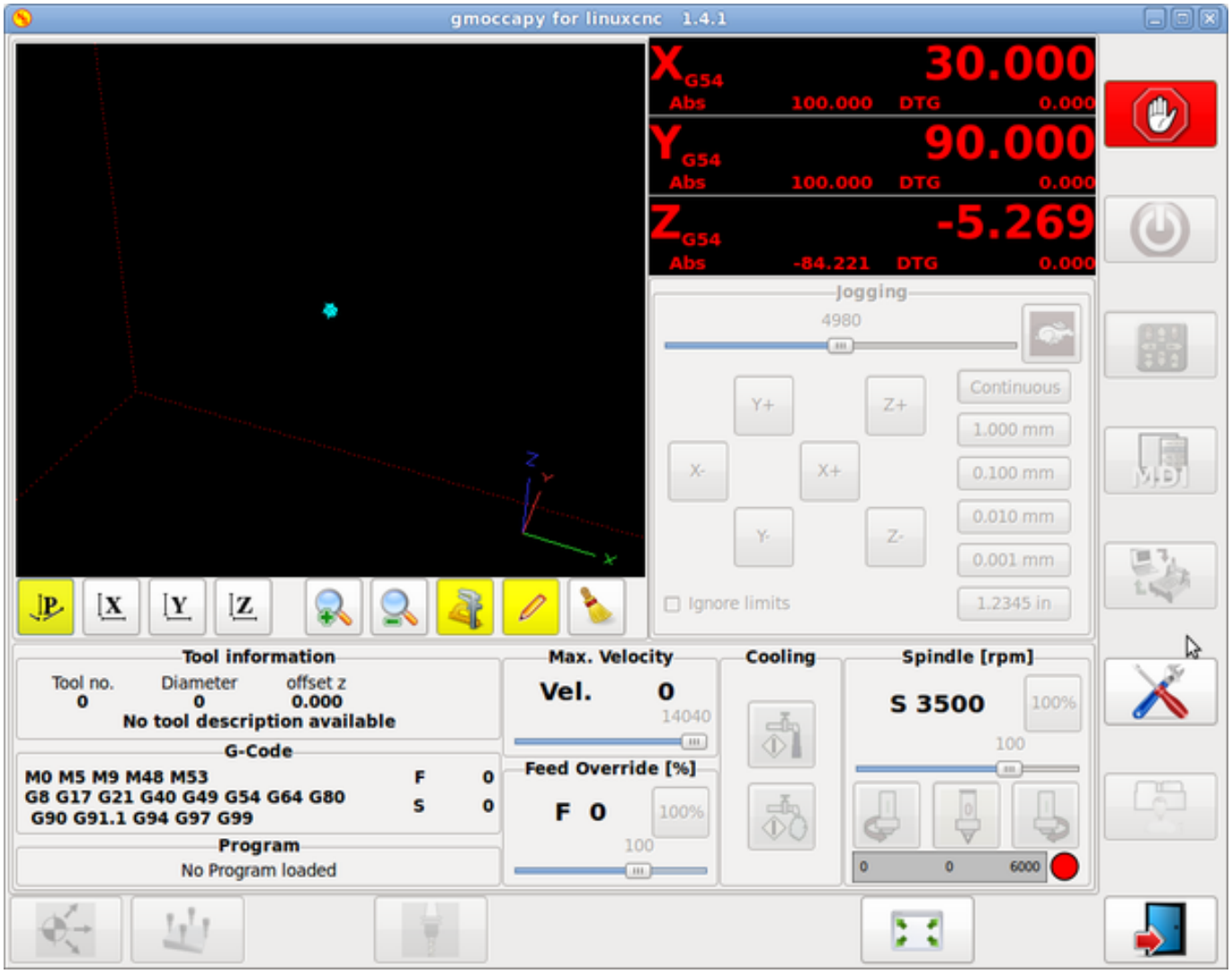
### 2.2.3 How To Get gmoccapy

Beginning with LinuxCNC 2.6 gmoccapy is included in the standard installation. So the easiest way to get gmoccapy on you controlling PC, is just to get the ISO and install from the CD / DVD /USB-Stick.

If you do have already installed an earlier LinuxCNC version, check how to update here.

You will receive updates with the regular deb packages.

You will get a similar screen to the following: The design may variate depending on your config.




### 2.2.4 Basic Configuration

There is really not to much to configure just to run gmoccapy, but there are some points you should take care off if you want to use all the features of the GUI.

You will find the following INI files included, just to show the basics:

* gmoccapy.ini
* gmoccapy_4_axis.ini
* gmoccapy_lathe.ini
* gmoccapy_lathe_imperial.ini
* gmoccapy_left_panel.ini
* gmoccapy_right_panel.ini
* gmoccapy_messages.ini
* gmoccapy_pendant.ini
* gmoccapy_sim_hardware_button.ini
* gmoccapy_tool_sensor.ini
* gmoccapy_with_user_tabs.ini

The names should explain the main intention of the different INI Files.

If you use an existing configuration of your machine, just edit your INI according to this document.

**Important**
If you want to use MACROS, don't forget to set the path to your macros or subroutines folder as described below.

So let us take a closer look to the the INI file and what you need to include to use gmoccapy on your machine:

#### 2.2.4.1 The DISPLAY Section

```
[DISPLAY]
DISPLAY = gmoccapy
PREFERENCE_FILE_PATH = gmoccapy_preferences
DEFAULT_LINEAR_VELOCITY = 166.666
MAX_LINEAR_VELOCITY = 166.666
MAX_FEED_OVERRIDE = 1.5
MAX_SPINDLE_OVERRIDE = 1.2
MIN_SPINDLE_OVERRIDE = 0.5
LATHE = 1
BACK_TOOL_LATHE = 1
PROGRAM_PREFIX = ../../nc_files/
```

The most important part is to tell LinuxCNC to use gmoccapy, editing the [DISPLAY] section.

```
[DISPLAY]
DISPLAY = gmoccapy

PREFERENCE_FILE_PATH = gmoccapy_preferences
```

The line PREFERENCE_FILE_PATH gives the location and name of the preferences file to be used. In most cases this line will not be needed, it is used by gmoccapy to store your settings of the GUI, like themes, DRO units, colors, and keyboard settings, etc., see settings page for more details.

**Note**
If no path or file is given, gmoccapy will use as default <your_machinename>.pref, if no machine name is given in your INI File it will use gmoccapy.pref The file will be stored in your config directory, so the settings will not be mixed if you use several configs. If you only want to use one file for several machines, you need to include PREFERENCE_FILE_PATH in your INI.

```
DEFAULT_LINEAR_VELOCITY = 166.666
```

Sets the default linear velocity in machine units per second.

**Note**
If no value is given, a value of 15 will be applied. If you don't set max linear velocity, the default linear velocity will be reduced to the default value max linear velocity (60).

If you don't set max velocity in TRAJ, it may be reduced as well see TRAJ section

```
MAX_LINEAR_VELOCITY = 166.666
```

Sets the value of the max velocity for jogging in machine units per second.

[NOTE] If no value is given, a value of 60 will be applied.

```
MAX_FEED_OVERRIDE = 1.5
```

Sets the maximum feed override, in the example given, you will be allowed to override the feed by 150%.

```
MAX_SPINDLE_OVERRIDE = 1.2
MIN_SPINDLE_OVERRIDE = 0.5
```

Will allow you to change the spindle override within a limit from 50% to 120%.

```
LATHE = 1
BACK_TOOL_LATHE = 1
```

The first line set the screen layout to control a lathe.

The second line is optional and will switch the X axis in a way you need for a back tool lathe. Also the keyboard shortcuts will react in a different way.

**Tip**
See also the Lathe Specific Section

- PROGRAM_PREFIX = ../../nc_files/

Is the entry to tell linuxcnc/gmoccapy where to look for the ngc files.

**Note**
If not specified Gmoccapy will look in the following order for ngc files: linuxcnc/nc_files and then the users home directory.

**Configuration of tabs and side panels** You can add embedded programs to gmoccapy like you can do in axis, touchy and gscreen. All is done by gmoccapy automatically if you include a few lines in your INI file in the DISPLAY section.

If you never used a glade panel, I recommend to read the excellent documentation. Glade VCP

**Example**

```
EMBED_TAB_NAME = DRO
EMBED_TAB_LOCATION = ntb_user_tabs
EMBED_TAB_COMMAND = gladevcp -x {XID} dro.glade

EMBED_TAB_NAME = Second user tab
EMBED_TAB_LOCATION = ntb_preview
EMBED_TAB_COMMAND = gladevcp -x {XID} vcp_box.glade
```

All you have to take care off, is that you include for every tab or side panel the mentioned three lines,

- EMBED_TAB_NAME = Represents the name of the tab or side panel, it is up to you what name you use, but it must be present!
- EMBED_TAB_LOCATION = Is the place where your program will be placed in the GUI.

VALID VALUES ARE:

- ntb_user_tabs (as main tab, covering the complete screen)'
- ntb_preview (as tab on the preview side)'
- box_left (on the left, complete high of the screen)
- box_right (on the right, in between the normal screen and the button list)
- box_coolant_and_spindle (will hide the coolant and spindle frames and introduce your glade file here)
- box_cooling (will hide the cooling frame and introduce your glade file)
- box_spindle (will hide the spindle frame and introduce your glade file)
- box_vel_info (will hide the velocity frames and introduce your glade file)
- box_custom_1 (will introduce your glade file left of vel_frame)
- box_custom_2 (will introduce your glade file left of cooling_frame)
- box_custom_3 (will introduce your glade file left of spindle_frame)
- box_custom_4 (will introduce your glade file right of spindle_frame)

See the different INI files included to see the differences

- EMBED_TAB_COMMAND = the command to execute, i.e.

```
gladevcp -x {XID} dro.glade
```

Includes a custom glade file called dro.glade in the mentioned location The file must be placed in the config folder of your machine.

```
gladevcp h_buttonlist.glade
```

Will just open a new user window called h_buttonlist.glade note the difference, this one is stand alone, and can be moved around independent from gmoccapy window.

```
camview-emc -w {XID}
```

Will add a live image from a web cam to the location you specified. Take care that camview-emc is installed, as it is not by default. Detailed information for camview and linuxcnc at: cam view

```
gladevcp -c gladevcp -u hitcounter.py -H manual-example.hal manual-example.ui
```

Will add a the panel manual-example.ui, include a custom python handler, hitcounter.py and make all connections after realizing the panel according to manual-example.hal.

Here are some examples:

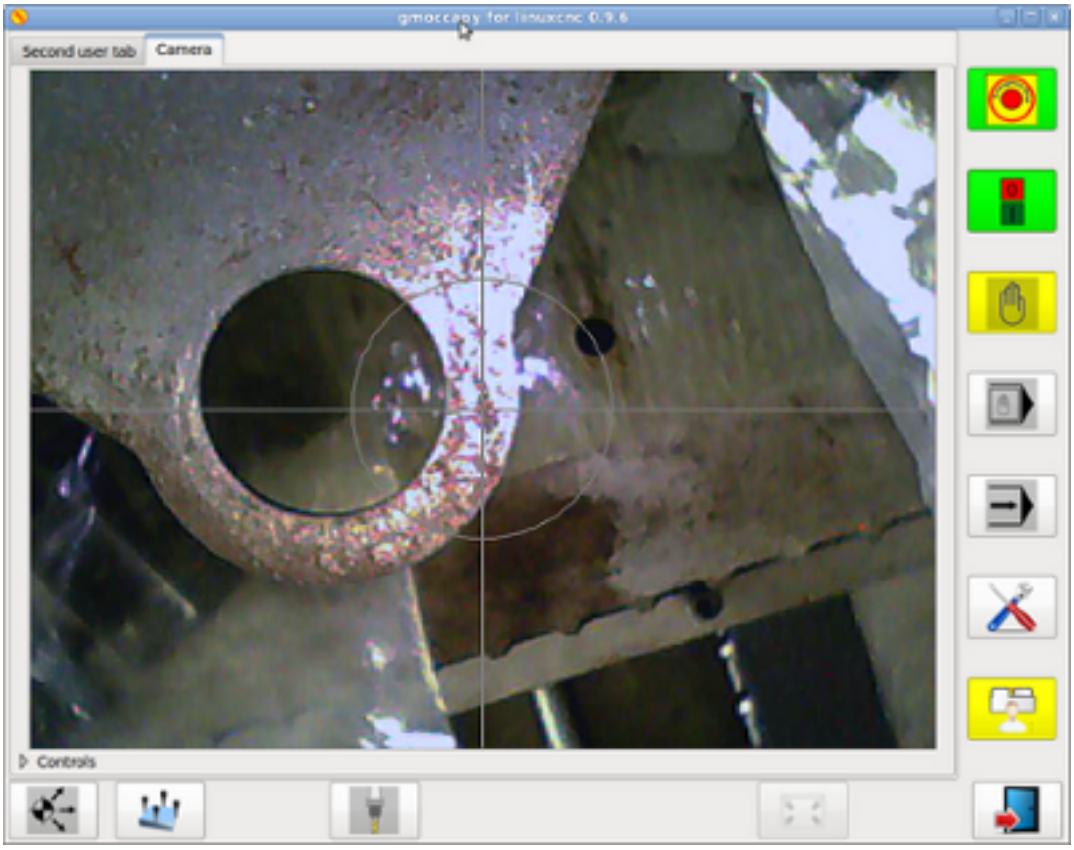



Figure 2.8: ntb_user_tabs - with integrated camview program

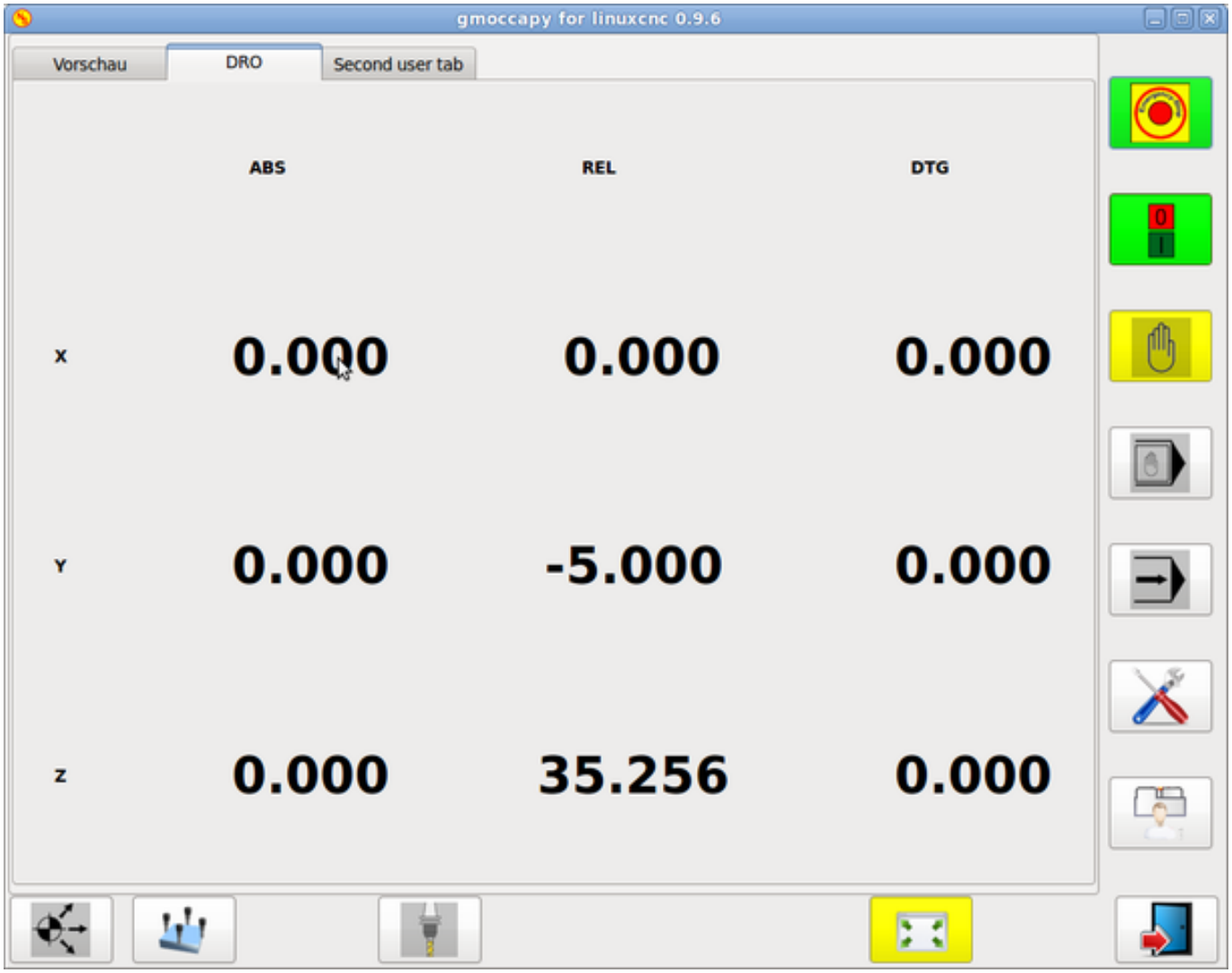



Figure 2.9: ntb_preview - as maximized version

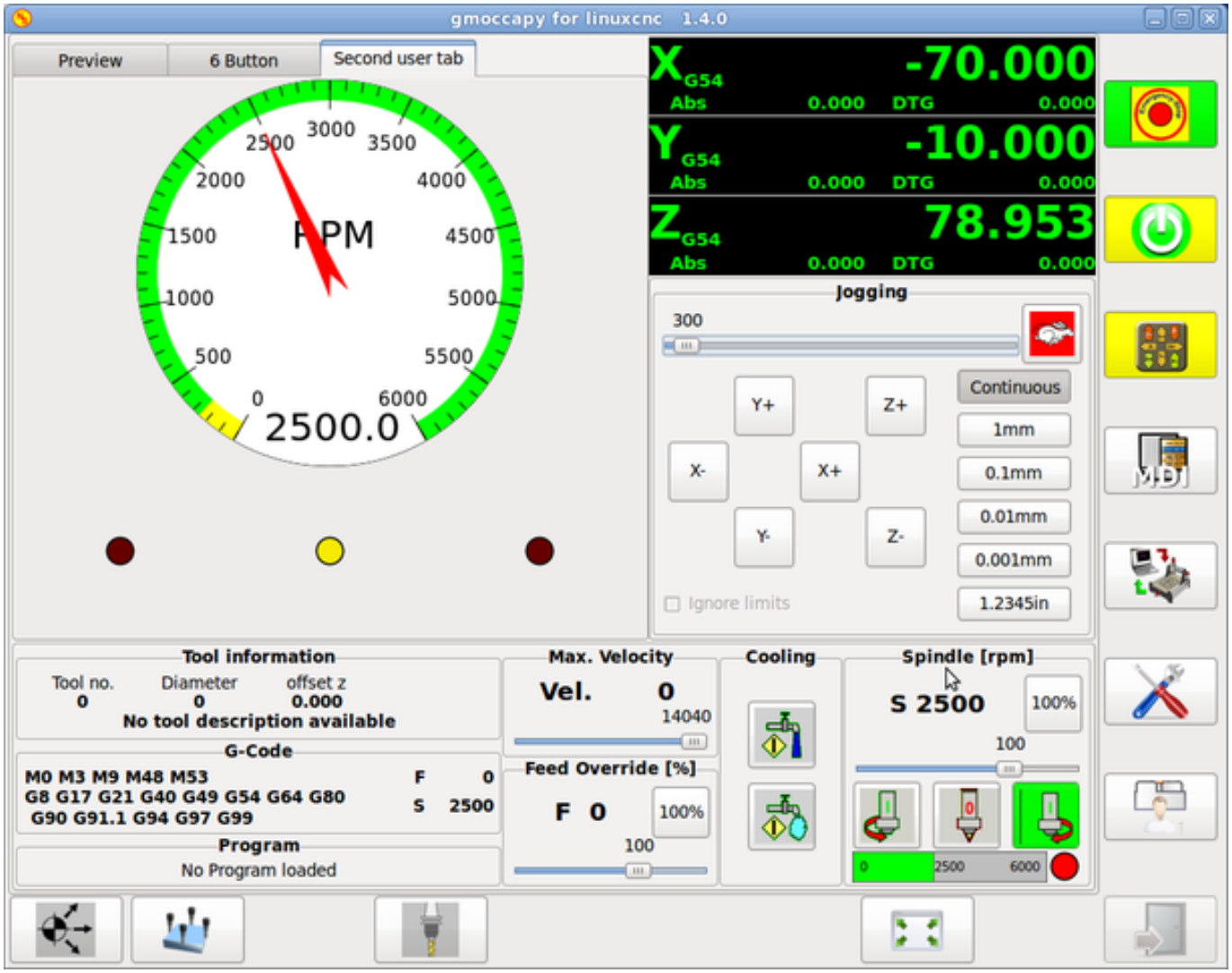



Figure 2.10: ntb_preview

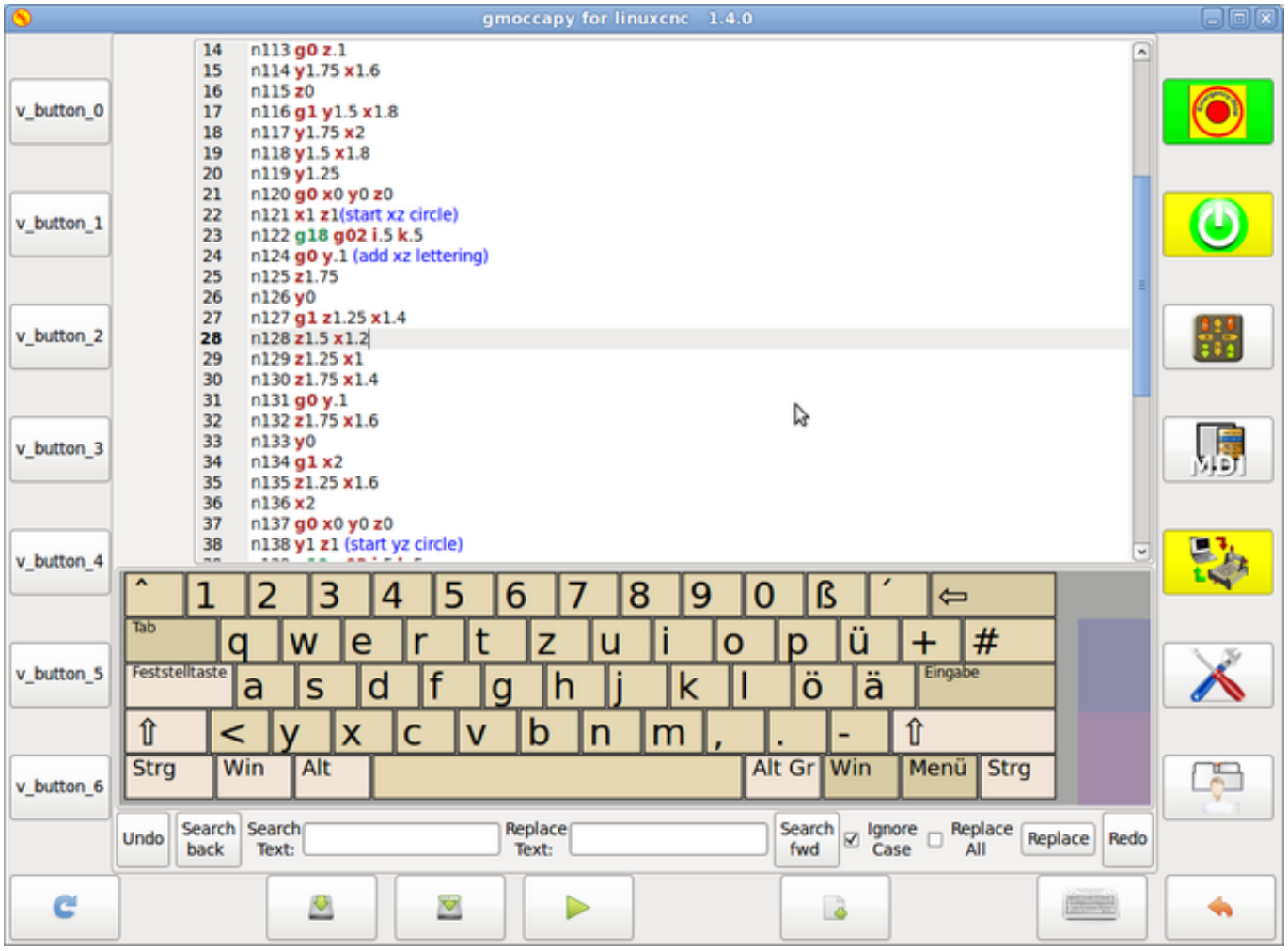



Figure 2.11: box_left - showing gmoccapy in edit mode

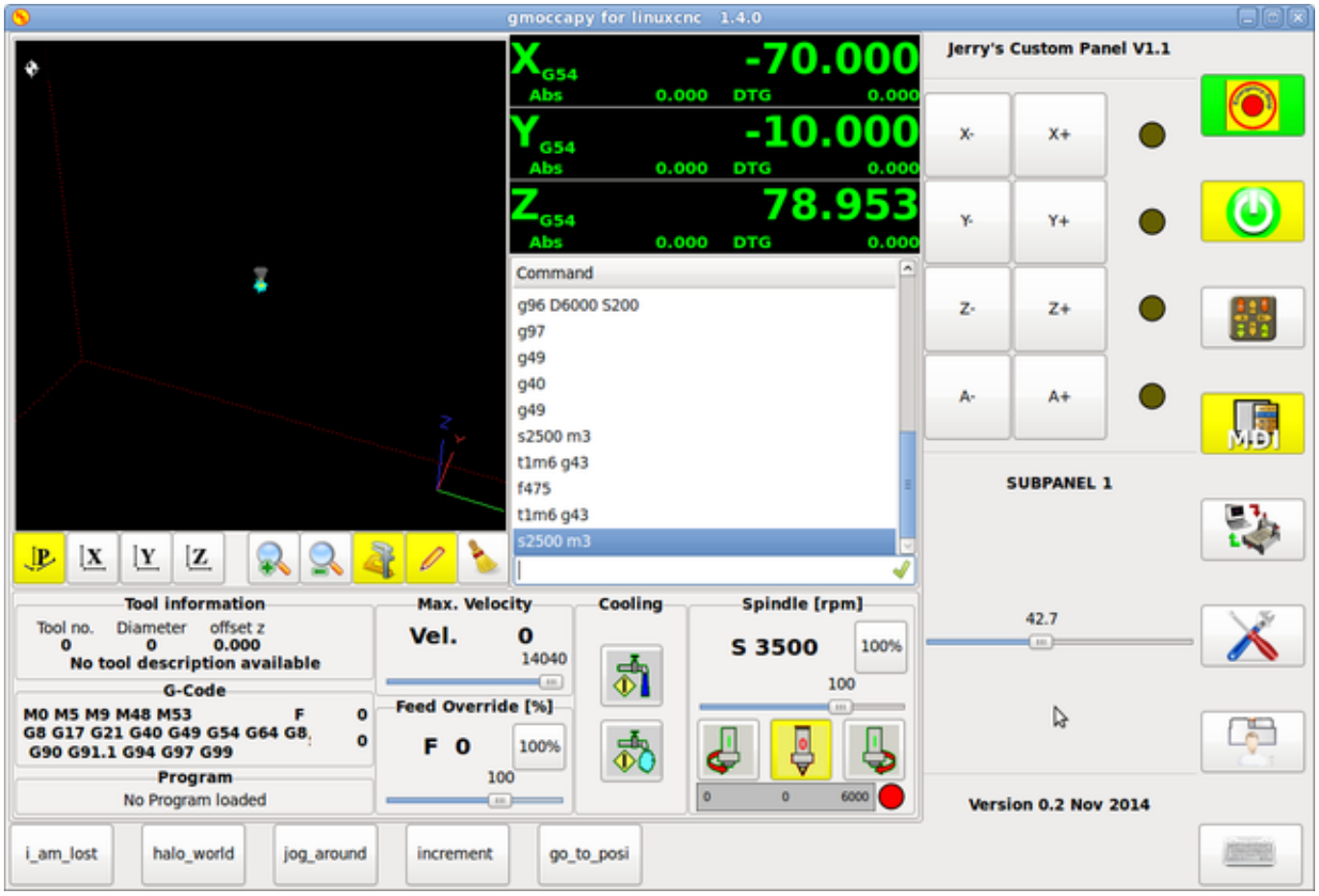



Figure 2.12: box_right - and gmoccapy in MDI mode

**Configuration of User Created Messages** Gmoccapy has the ability to create hal driven user messages. To use them you need to introduce some lines in the [DISPLAY] section of the INI file.

Here is how to set up 3 user pop up message dialogs the messages support pango markup language. Detailed information about the markup language can be found at Pango Markup

```
MESSAGE_TEXT    = The text to be displayed, may be pango markup formatted
MESSAGE_TYPE    = "status" , "okdialog" , "yesnodialog"
MESSAGE_PINNAME = is the name of the hal pin group to be created
```

- *status* : Will just display a message as pop up window, using the messaging system of gmoccapy
- *okdialog* : Will hold focus on the message dialog and will activate a "-waiting" Hal_Pin OUT. Closing the message will reset the waiting pin
- *yesnodialog* : Will hold focus on the message dialog and will activate a "-waiting" Hal_Pin bit OUT it will also give access to an "-response" Hal_Pin Bit Out, this pin will hold 1 if the user clicks OK, and in all other states it will be 0 Closing the message will reset the waiting pin The response Hal Pin will remain 1 until the dialog is called again

**Example**

```
MESSAGE_TEXT = This is a <span background="#ff0000" foreground="#ffffff">
info-message</span> test
MESSAGE_TYPE = status
MESSAGE_PINNAME = statustest
```

```
MESSAGE_TEXT = This is a yes no dialog test
MESSAGE_TYPE = yesnodialog
MESSAGE_PINNAME = yesnodialog

MESSAGE_TEXT = Text can be <small>small</small>, <big>big</big>, <b>bold</b <i>italic</i>,  ↩
    and even be <span color="red">colored</span>.
MESSAGE_TYPE = okdialog
MESSAGE_PINNAME = okdialog
```

The specific hal pin conventions for these can be found under the User Messages hal pin section.

#### 2.2.4.2 The RS274NGC Section

```
[RS274NGC]
SUBROUTINE_PATH = macros
```

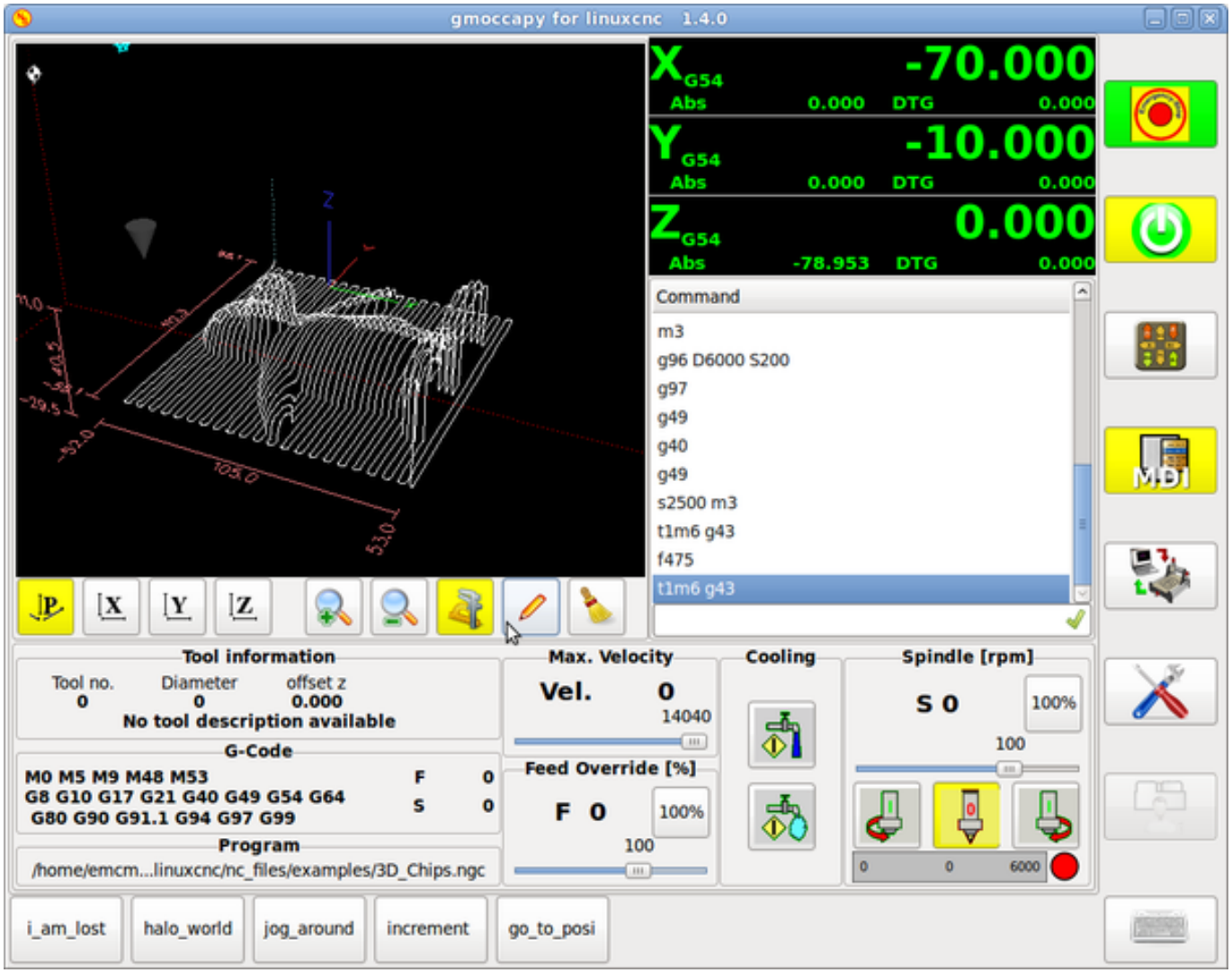
Sets the path to search for macros and other subroutines.

#### 2.2.4.3 The MACRO Section

You can add macros to gmoccapy, similar to touchy's way. A macro is nothing else than a ngc-file. You are able to execute complete CNC programs in MDI mode, by just pushing one button. To do so, you have to add a section like so:

```
[MACROS]
MACRO = i_am_lost
MACRO = hello_world
MACRO = jog_around
MACRO = increment xinc yinc
MACRO = go_to_position X-pos Y-pos Z-pos
```

This will add 5 macros to the MDI button list. Please note, that maximal 9 macros will appear in the GUI, due to place reasons. But it is no error placing more in your INI file.




The name of the file must be **exactly the same** as the name given in the MACRO line. So the macro ***i_am_lost*** will call the file ***i_am_lost.ngc***.

THE MACRO FILES MUST FOLLOW SOME RULES:

- the name of the file need to be the same as the name mentioned in the macro line, just with the ngc extension
- The file must contain a subroutine like so: ***O<i_am_lost> sub***, the name of the sub must match exactly (**case sensitive**) the name of the macro
- the file must end with an endsub ***O<i_am_lost> endsub*** followed by an ***M2*** command
- the files need to be placed in a folder specified in your INI file in the RS274NGC section (see RS274NGC)

The code in between sub and endsub will be executed by pushing the corresponding macro button.

**Note**
You will find the sample macros in macros folder placed in the gmoccapy sim folder.

Gmoccapy will also accept macros asking for parameters like:

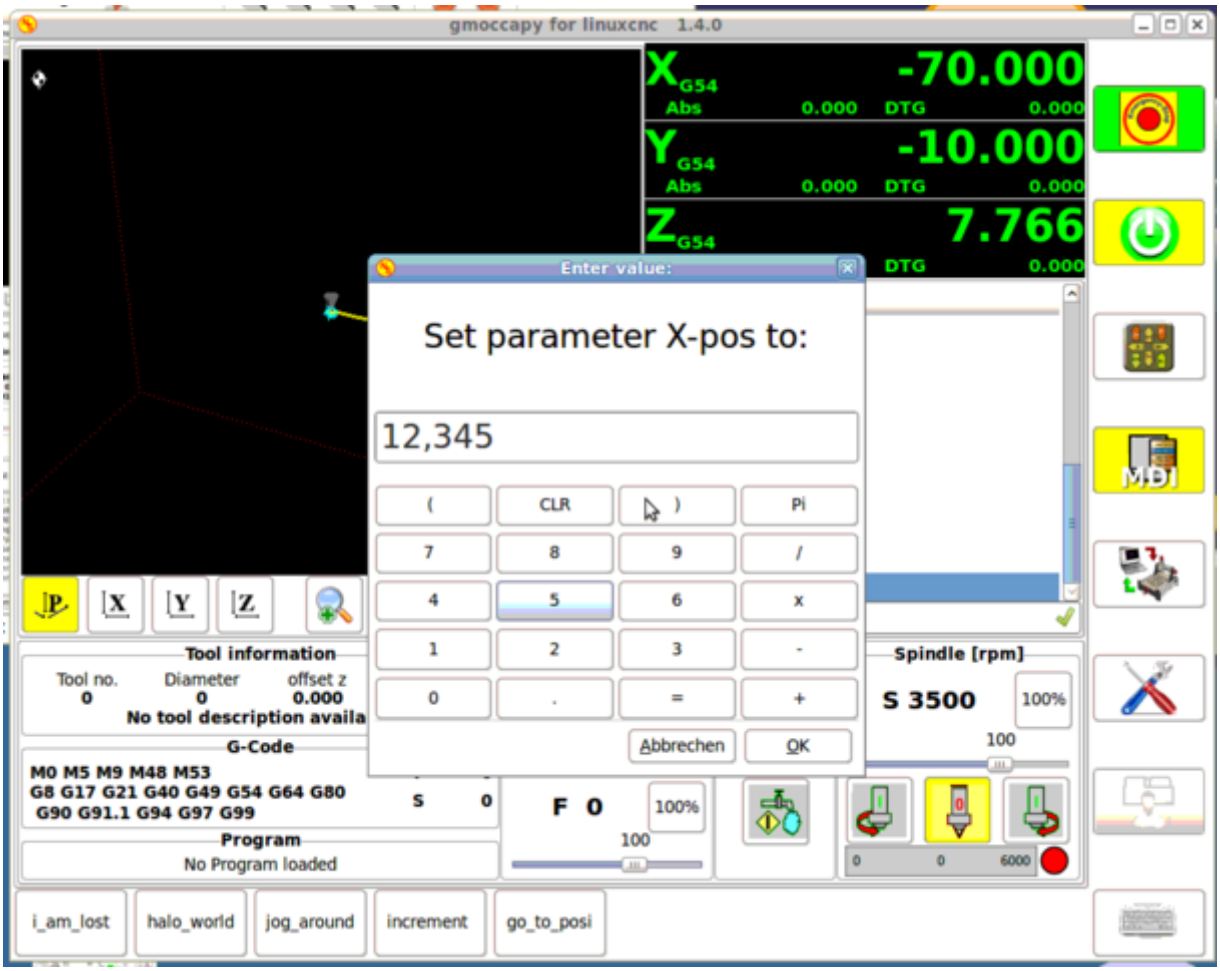
```
go_to_position X-pos Y-pos Z-pos
```

The parameters must be separated by spaces. This calls a file *go_to_position.ngc* with the following content:

```
; Test file go to position
; will jog the machine to a given position

O<go_to_position> sub

G17
G21
G54
G61
G40
G49
G80
G90

;#1 = <X-Pos>
;#2 = <Y-Pos>
;#3 = <Z-Pos>

(DBG, Will now move machine to X = #1 , Y = #2 , Z = #3)
G0 X #1 Y #2 Z #3

O<go_to_position> endsub
M2
```

After pushing the ***execute macro button***, you will be asked to enter the values for ***X-pos Y-pos Z-pos*** and the macro will only run if all values have been given.

**Note**
If you would like to use a macro without any movement, see also the notes in known problems




#### 2.2.4.4 The TRAJ Section

```
MAX_VELOCITY = 230.000
```

Sets the maximal velocity of the machine, this value will also take influence to default velocity.

### 2.2.5 HAL Pins

gmoccapy exports several hal pin to be able to react to hardware devices. The goal is to get a GUI that may be operated in a tool shop, completely/mostly without mouse or keyboard.

**Note**
You will have to do all connections to gmoccapy pins in your postgui.hal file, because they are not available before loading the GUI completely.

#### 2.2.5.1 Right And Bottom Button Lists

The screen has two main button lists, one on the right side an one on the bottom. The right handed buttons will not change during operation, but the bottom button list will change very often. The buttons are count from up to down and from left to right beginning with "0".

In hal_show you will see the right (vertical) buttons are:

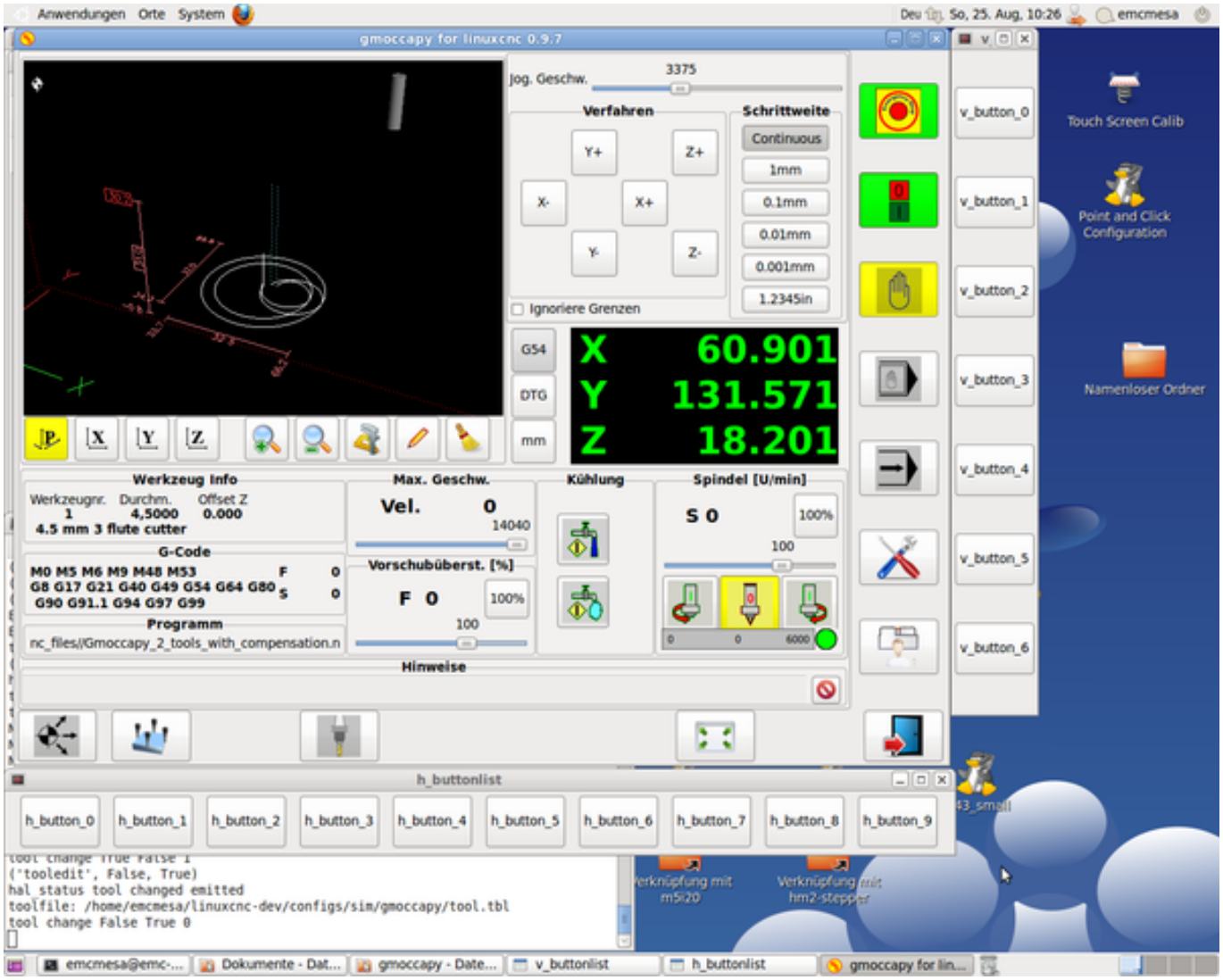
- gmoccapy.v-button-0
- gmoccapy.v-button-1
- gmoccapy.v-button-2
- gmoccapy.v-button-3
- gmoccapy.v-button-4
- gmoccapy.v-button-5
- gmoccapy.v-button-6

and the bottom (horizontal) buttons are:

- gmoccapy.h-button-0
- gmoccapy.h-button-1
- gmoccapy.h-button-2
- gmoccapy.h-button-3
- gmoccapy.h-button-4
- gmoccapy.h-button-5
- gmoccapy.h-button-6
- gmoccapy.h-button-7
- gmoccapy.h-button-8
- gmoccapy.h-button-9

As the buttons in the bottom list will change according the mode and other influences, the hardware buttons will activate different functions, and you don't have to take care about switching functions around in hal, because that is done completely by gmoccapy!

These pins are made available to be able to use the screen without an touch panel, or protect it from excessive use by placing hardware buttons around the panel.




### 2.2.5.2 Velocities And Overrides

All sliders from gmoccapy can be connected to hardware encoder or hardware potentiometers.

To connect encoders the following pin are exported:

- gmoccapy.max-vel-counts = HAL_S32 (Maximal Velocity of the machine)
- gmoccapy.jog-speed-counts = HAL_S32 (Jog velocity)
- gmoccapy.spindle-override-counts = HAL_S32 (spindle override)
- gmoccapy.feed-override-counts = HAL_S32 (feed override)
- gmoccapy.reset-feed-override = HAL_BIT (reset the feed override to 100 %)
- gmoccapy.reset-spindle-override = HAL_BIT (reset the spindle override to 100 %)

To connect potentiometers, use the following hal pin:

- gmoccapy.analog-enable = HAL_BIT Must be True, to allow analog inputs
- gmoccapy.jog-vel-value = HAL_FLOAT To adjust the jog velocity slider

- gmoccapy.max-vel-value = HAL_FLOAT To adjust the max velocity slider
- gmoccapy.feed-override-value = HAL_FLOAT To adjust the feed override slider
- gmoccapy.spindle-override-value = HAL_FLOAT To adjust the spindle override slider

The float pin do accept values from 0.0 to 1.0, being the percentage value you want to set the slider value.

[WARNING] If you use both connection types, do not connect the same slider to both pin, as the influences between the two has not been tested! Different sliders may be connected to the one or other hal connection type.

[IMPORTANT] Please be aware, that for the jog velocity depends on the turtle button state, it will lead to different slider scales depending on the mode (turtle or rabbit). Please take also a look to jog velocities and turtle-jog hal pin for more details.

**Example**

```
Spindle Override Min Value =  20 %
Spindle Override Max Value = 120 %
gmoccapy.analog-enable = 1
gmoccapy.spindle-override-value = 0.25

value to set = Min Value + (Max Value - Min Value) * gmoccapy.spindle-override-value
value to set = 20 + (120 - 20) * 0.25
value to set = 45 %
```

#### 2.2.5.3 Jog Hal Pins

All axis given in the INI File have a jog-plus and a jog-minus pin, so hardware momentary switches can be used to jog the axis.

For the standard config following hal Pin will be available:

- gmoccapy.jog-x-plus
- gmoccapy.jog-x-minus
- gmoccapy.jog-y-plus
- gmoccapy.jog-y-minus
- gmoccapy.jog-z-plus
- gmoccapy.jog-z-minus

If you use a 4 axis INI file, there will be two additional pins

- gmoccapy.jog-<your fourth axis letter >-plus
- gmoccapy.jog-<your fourth axis letter >-minus

For a "C" axis you will see:

- gmoccapy.jog-c-plus
- gmoccapy.jog-c-minus

#### 2.2.5.4 Jog Velocities And Turtle-Jog Hal Pin

The jog velocity can be selected with the corresponding slider. The scale of the slider will be modified if the turtle button (the one showing a rabbit or a turtle) has been toggled. If the button is not visible, it might have been disabled on the settings page. If the button shows the rabbit-icon, the scale is from min to max machine velocity. If it shows the turtle, the scale will reach only 1/20 of max velocity by default. The used divider can be set on the settings page.

So using a touch screen it is much easier to select smaller velocities.

#### 2.2.5.5 Jog Increment Hal Pins

The jog increments are selectable through hal pins, so a select hardware switch can be used to select the increment to use. There will be a maximum of 10 hal pin for the increments given in the INI File, if you give more increments in your INI File, they will be not reachable from the GUI as they will not be displayed.

If you have 6 increments in your hal you will get **7** pins:

- gmoccapy.jog-inc-0
- gmoccapy.jog-inc-1
- gmoccapy.jog-inc-2
- gmoccapy.jog-inc-3
- gmoccapy.jog-inc-4
- gmoccapy.jog-inc-5
- gmoccapy.jog-inc-6

jog-inc-0 is unchangeable and will represent continuous jogging.

#### 2.2.5.6 Hardware Unlock Pin

To be able to use a key switch to unlock the settings page the following pin is exported.

- gmoccapy.unlock-settings

The settings page is unlocked if the pin is high. To use this pin, you need to activate it on the settings page.

#### 2.2.5.7 Error Pins

- gmoccapy.error
- gmoccapy.delete-message

gmoccapy.error is an bit out pin, to indicate an error, so a light can lit or even the machine may be stopped. It will be reseted with the pin gmoccapy.delete-message. gmoccapy.delete-message will delete the first error and reset the gmoccapy.error pin to False after the last error has been cleared.

**Note**
Messages or user infos will not affect the gmoccapy.error pin, but the gmoccapy.delete-message pin will delete the last message if no error is shown!

#### 2.2.5.8 User Created Message HAL Pins

gmoccapy may react to external errors, using 3 different user messages: All are HAL_BIT pin.

*Status*

- gmoccapy.messages.statustest

*Yesnodialog*

- gmoccapy.messages.yesnodialog
- gmoccapy.messages.yesnodialog-waiting
- gmoccapy.messages.yesnodialog-responce

*Okdialog*

- gmoccapy.messages.okdialog
- gmoccapy.messages.okdialog-waiting

To add user created message you need to add the message to the INI file in the [DISPLAY] section. Here are a couple of examples.

```
MESSAGE_BOLDTEXT = LUBE SYSTEM FAULT
MESSAGE_TEXT = LUBE FAULT
MESSAGE_TYPE = okdialog
MESSAGE_PINNAME = lube-fault

MESSAGE_BOLDTEXT = NONE
MESSAGE_TEXT = X SHEAR PIN BROKEN
MESSAGE_TYPE = status
MESSAGE_PINNAME = xpin
```

To *connect* new pins to and input you need to do this in the postgui HAL file. Here are some example connections that have the signal connected to an input some place else in the HAL file.

```
net gmoccapy-lube-fault gmoccapy.messages.lube-fault
net gmoccapy-lube-fault-waiting gmoccapy.messages.lube-fault-waiting
net gmoccapy-xpin gmoccapy.messages.xpin
```

For more information on HAL files and the net command see the Basic HAL Reference.

#### 2.2.5.9 Spindle Feedback Pins

There are two pins for spindle feedback

- gmoccapy.spindle_feedback_bar
- gmoccapy.spindle_at_speed_led

*gmoccapy.spindle_feedback_bar* will accept an float input to show the spindle speed. *gmoccapy.spindle_at_speed_led* is an bit-pin to lit the GUI led if spindle is at speed.

#### 2.2.5.10 Pins To Indicate Program Progress Information

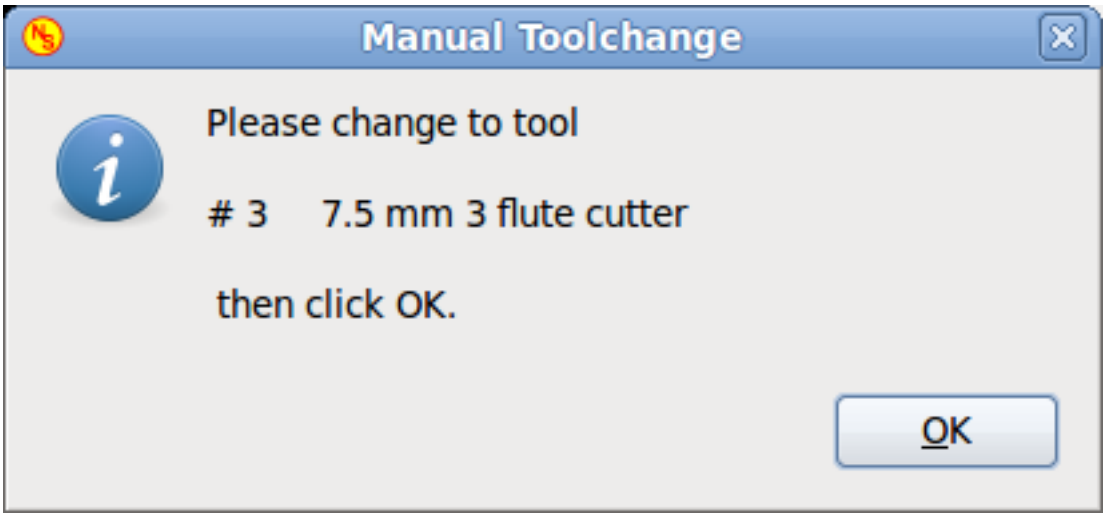
There are three pins giving information over the program progress

- gmoccapy.program.length HAL_S32 showing the total number of lines of the program
- gmoccapy.program.current-line HAL_S32 indicating the current working line of the program
- gmoccapy.program.progress HAL_FLOAT giving the program progress in percentage

The values may not be very accurate, if you are working with subroutines or large remap procedures, also loops will cause different values.

#### 2.2.5.11 Tool related pin

**Tool Change Pin** This pin are provided to use gmoccapy's internal tool change dialog, similar to the one known from axis, but with several modifications, so you will not only get the message to change to *tool number 3*, but also the description of that tool like *7.5 mm 3 flute cutter*. The information is taken from the tool table, so it is up to you what to display.

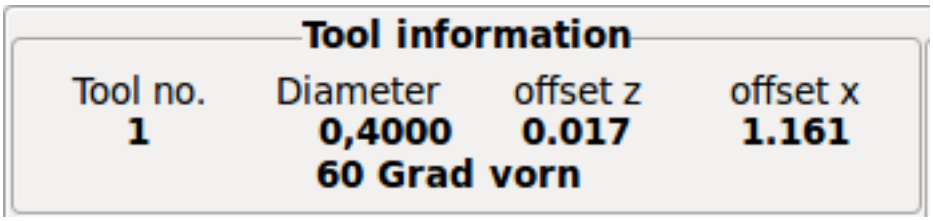



- gmoccapy.toolchange-number HAL_S32 The number of the tool to be changed
- gmoccapy.toolchange-change HAL_BIT Indicate that a tool has to be changed
- gmoccapy.toolchange-changed HAL_BIT Indicate toll has been changed

Usually they are connected like this for a manual tool change:

```
net tool-change gmoccapy.toolchange-change <= iocontrol.0.tool-change
net tool-changed gmoccapy.toolchange-changed <= iocontrol.0.tool-changed
net tool-prep-number gmoccapy.toolchange-number <= iocontrol.0.tool-prep-number
net tool-prep-loop iocontrol.0.tool-prepare <= iocontrol.0.tool-prepared
```

**Tool Offset Pins** This pins allows you to show the active tool offset values for X and Z in the tool information frame. You should know that they are only active after G43 has been sent.




- gmoccapy.tooloffset-x
- gmoccapy.tooloffset-z

Connect them in your postgui hal.

```
net tooloffset-x gmoccapy.tooloffset-x <= motion.tooloffset.x
net tooloffset-z gmoccapy.tooloffset-z <= motion.tooloffset.z
```

Please note, that gmoccapy takes care of its own to update the offsets, sending an G43 after any tool change, **but not in auto mode!**

**Important**
So writing a program makes you responsible to include an G43 after each tool change!

### 2.2.6 Auto Tool Measurement

Gmoccapy offers an integrated auto tool measurement. To use this feature, you will need to do some additional settings and you may want to use the offered hal pin to get values in your own ngc remap procedure.

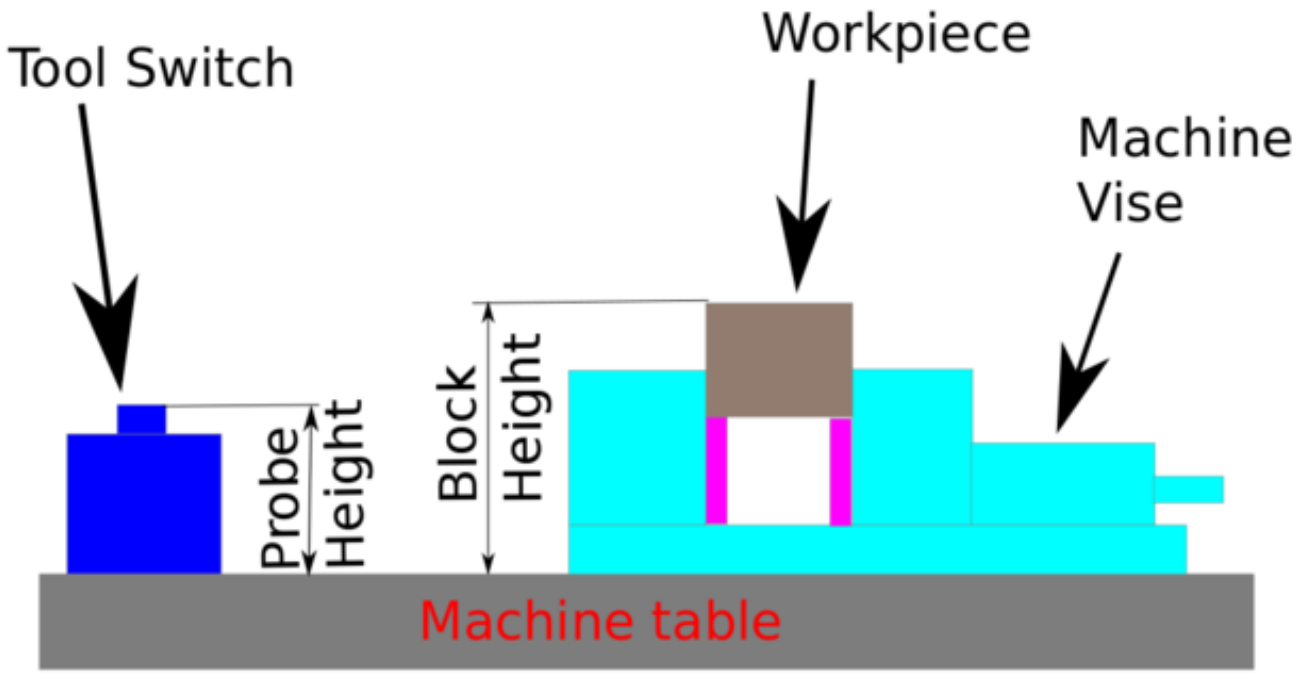
[IMPORTANT] Before starting the first test, do not forget to enter the probe height and probe velocities on the settings page! See Settings Page Tool Measurement

It might be also a good idea to take a look at the tool measurement video: see tool measurement related videos

Tool Measurement in gmoccapy is done a little bit different to many other GUI. You should follow these steps:

- touch of you workpiece in X and Y
- measure the hight of your block from the base where your tool switch is located, to the upper face of the block (including chuck etc.)
- Push the button block height and enter the measured value
- Go to auto mode and start your program

here is a small sketch:



With the first given tool change the tool will be measured and the offset will be set automatically to fit the block height. The advantage of the gmoccapy way is, that you do not need a reference tool.

**Note**
Your program must contain a tool change at the beginning! The tool will be measured, even it has been used before, so there is no danger, if the block height has changed. There are several videos showing the way to do that on you tube.

#### 2.2.6.1 Tool Measurement Pins

Gmoccapy offers 5 pins for tool measurement purpose. The pins are mostly used to be read from a gcode subroutine, so the code can react to different values.

- gmoccapy.toolmeasurement HAL_BIT enable or not tool measurement
- gmoccapy.blockheight HAL_FLOAT the measured value of the top face of the workpiece
- gmoccapy.probeheight HAL_FLOAT the probe switch height
- gmoccapy.searchvel HAL_FLOAT the velocity to search for the tool probe switch
- gmoccapy.probevel HAL_FLOAT the velocity to probe tool length

#### 2.2.6.2 Tool Measurement INI File Modifications

Modify your INI File to include the following:

**The RS274NGC section**

```
[RS274NGC]
# Enables the reading of INI and HAL values from gcode
FEATURES=12

# is the sub, with is called when a error during tool change happens
ON_ABORT_COMMAND=O <on_abort> call

# The remap code
REMAP=M6  modalgroup=6 prolog=change_prolog ngc=change epilog=change_epilog
```

**The Tool Sensor Section** The position of the tool sensor and the start position of the probing movement, all values are absolute coordinates, except MAXPROBE, what must be given in relative movement.

```
[TOOLSENSOR]
X = 10
Y = 10
Z = -20
MAXPROBE =  -20
```

**The Change Position Section** This is not named TOOL_CHANGE_POSITION on purpose - canon uses that name and will interfere otherwise. The position to move the machine before giving the change tool command. All values are in absolute coordinates.

```
[CHANGE_POSITION]
X = 10
Y = 10
Z = -2
```

**The Python Section** The Python plug ins serves interpreter and task.

```
[PYTHON]
# The path to start a search for user modules
PATH_PREPEND = python
# The start point for all.
TOPLEVEL = python/toplevel.py
```

#### 2.2.6.3 Needed Files

You must copy the following files to your config directory

First make a directory *python* in your config folder from *your_linuxcnc-dev_directory/configs/sim/gmoccapy/python* copy *toplevel.py* to your *config_dir/python* folder. Copy *remap.py* to your *config_dir/python* folder Copy *stdglue.py* to your *config_dir/python* folder.

From *your_linuxcnc-dev_directory/configs/sim/gmoccapy/macros* copy *on_abort.ngc* to the directory specified in the SUBROUTINE_PATH see RS274NGC Section. From *your_linuxcnc-dev_directory/configs/sim/gmoccapy/macros* copy *change.ngc* to the directory specified as SUBROUTINE_PATH see RS274NGC Section. Open *change.ngc* with a editor and uncomment the following lines (49 and 50):

```
F #<_hal[gmoccapy.probevel]>
G38.2 Z-4
```

You may want to modify this file to fit more your needs.

#### 2.2.6.4 Needed Hal Connections

Connect the tool probe in your hal file like so:

```
net probe  motion.probe-input <= <your_input_pin>
```

The line might look like this:

```
 net probe  motion.probe-input <= parport.0.pin-15-in
```

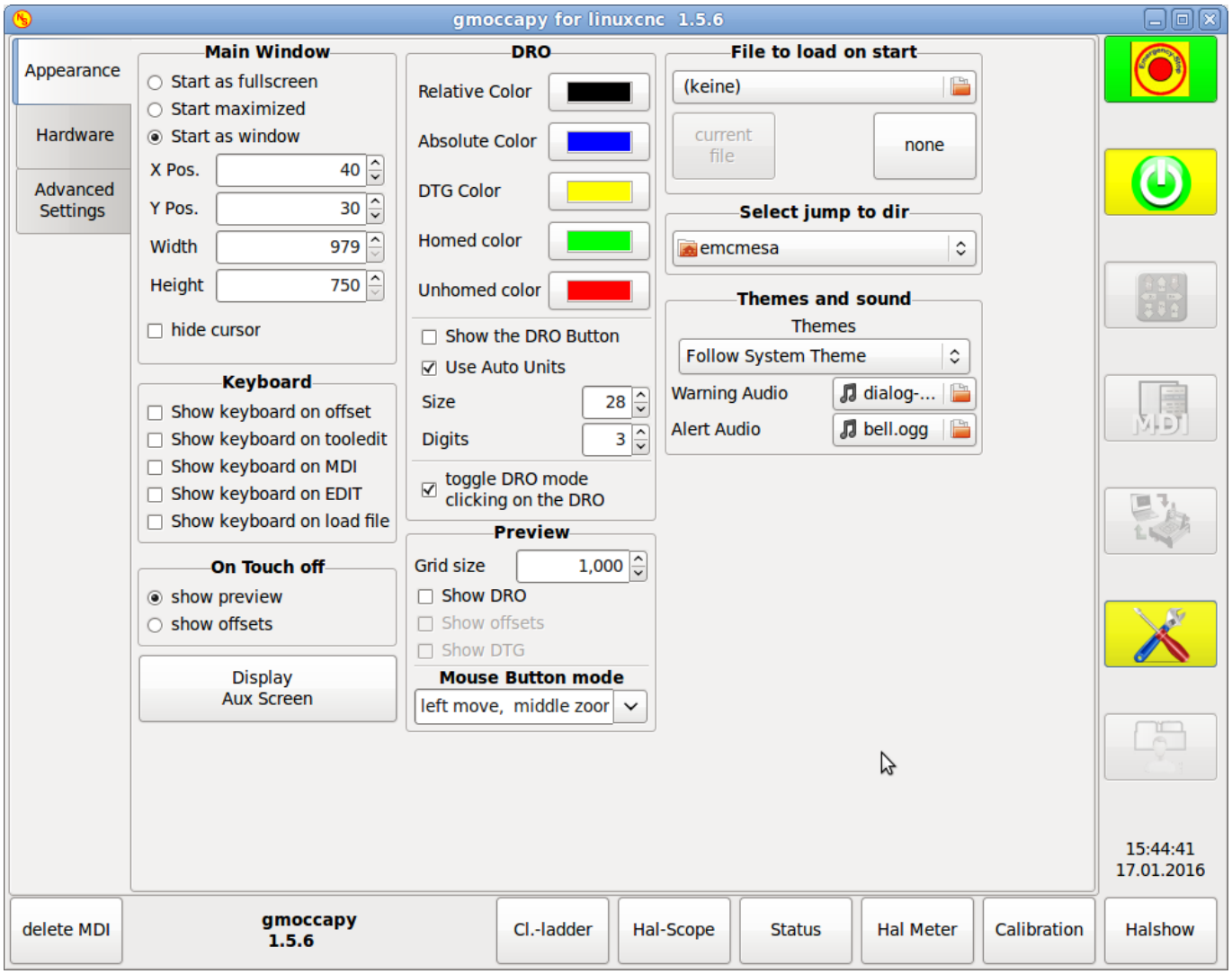
In your postgui.hal file add:

```
# The next lines are only needed if the pins had been connected before
unlinkp iocontrol.0.tool-change
unlinkp iocontrol.0.tool-changed
unlinkp iocontrol.0.tool-prep-number
unlinkp iocontrol.0.tool-prepared

# link to gmoccapy toolchange, so you get the advantage of tool description on change  ↩
    dialog
net tool-change gmoccapy.toolchange-change <= iocontrol.0.tool-change
net tool-changed gmoccapy.toolchange-changed <= iocontrol.0.tool-changed
net tool-prep-number gmoccapy.toolchange-number <= iocontrol.0.tool-prep-number
net tool-prep-loop iocontrol.0.tool-prepare <= iocontrol.0.tool-prepared
```

### 2.2.7 The Settings Page

To enter the page you will have to click on and give an unlock code, witch is **123** as default. If you want to change it at this time you will have to edit the hidden preference file, see the display section for details.

The page looks at the moment like so:




The page is separated in three main tabs:

#### 2.2.7.1 Appearance

On this tab you will find the following options:

**Main Window**

Here you can select how you wish the GUI to start. The main reason for this was the wish to get an easy way for the user to set the starting options without the need to touch code.

You have three options:

- start as full screen
- start maximized
- start as window
  If you select start as window the spinboxes to set the position and size will get active.
  One time set, the GUI will start every time on the place and with the size selected.
  Nevertheless the user can change the size and position using the mouse, but that will not have any influence on the settings.

***hide the cursor*** does allow to hide the cursor, what is very useful if you use a touch screen.

**Keyboard**

The check-boxes allows the user to select if he want the on board keyboard to be shown immediately, when entering the MDI Mode, when entering the offset page, the tooledit widget or when open a program in the EDIT mode. The keyboard button on the bottom button list will not been affected by this settings, so you be able to show or hide the keyboard by pressing the button. The default behavior will be set by the check-boxes.

Default are :

- show keyboard on offset = True
- show keyboard on tooledit = False
- show keyboard on MDI = True
- show keyboard on EDIT = True
- show keyboard on load file = False

If the keyboard layout is not correct, i.e. clicking X gives Z, than the layout has not been set properly, related to your locale settings. For onboard it can be solved with a small batch file with the following content:

**Note**
If this section is not sensitive, you have not installed a virtual keyboard, + supported are *onboard* and *matchbox-keyboard*.

```
#!/bin/bash
setxkbmap -model pc105 -layout de -variant basic
```

The letters "de" are for German, you will have to set them according to your locale settings. Just execute this file before starting LinuxCNC, it can be done also adding a starter to your local folder.

```
./config/autostart
```

So that the layout is set automatically on starting.

For matchbox-keyboard you will have to make your own layout, for a German layout ask in the forum.

**Note**
If this section is not sensitive, you have not installed a virtual keyboard, supported are *onboard* and *matchbox-keyboard*.

**On Touch Off**

give the option to show the preview tab or the offset page tab if you enter the touch off mode by clicking the corresponding bottom button.

- show preview
- show offsets

As the notebook tabs are shown, you are able to switch between both views in any case.

**Show Aux Display**

By clicking this button a additional window will be opened. This button is only sensitive, if a file named *gmoccapy2.glade* is located in your config folder. You can build the Aux Screen using Glade.

**Warning**
*The main window of the aux screen must be named window2*

**DRO Options**

You have the option to select the background colors of the different DRO states. So users suffering from protanopia (red/green weakness) are able to select proper colors

By default the backgrounds are:

- Relative mode = black
- Absolute mode = blue
- Distance to go = yellow

The foreground color of the DRO can be selected with:

- homed color = green
- unhomed color = red

*show dro in preview*
the DRO will be shown in the preview window +

*show offsets*+ the Offsets will be shown in the preview window +

*show DTG*
the distance to go will be shown in the preview window +

*show DRO Button*
will allow you to display additional buttons on the left side of the DRO.

It will display:
* one button to switch from relative to absolute coordinates,
* one button to toggle between distance to go and the other states
* and one button to toggle the units from metric to imperial and vice versa.

[WARNING] It is not recommended to use this option, because the user will loose the auto unit option, which will toggle the units according to the active gcode G20 / G21.

[NOTE] **You can change through the DRO modes (absolute, relative, distance to go) by clicking on the DRO!**

*Use Auto Units*
allows to disable the auto units option of the display, so you can run a program in inches and watch the DRO in mm. +

*size*
allows to set the size of the DRO font, default is 28, if you use a bigger screen you may want to increase the size up to 56. If you do use 4 axis, the DRO font size will be 3/4 of the value, because of space reason. +

*digits*
sets the number of digits of the DRO from 1 to 5.

**Note**
Imperial will show one digit more that metric. So if you are in imperial machine units and set the digit value to 1, you will get no digit at all in metric.

*toggle DRO mode*
if not active, a mouse click on the DRO will not take any action.
By default this checkbox is active, so every click on any DRO will toogle the DRO readout from actual to relative to DTG (distance to go). +

**Preview**

*Grid Size* Sets the grid size of the preview window. Unfortunately the size **has to be set in inches**, even if your machine units are metric. We do hope to fix that in a future release.

[NOTE] The grid will not be shown in perspective view.

*Show DRO*
Will show the a DRO also in the preview window, it will be shown automatically in fullsize preview

*Show DTG* will show also the DTG (direct distance to end point) in the preview, only if Show DRO is active and not full size preview.

*Show Offsets* will show the offsets in the preview window.

[NOTE] If you only check this option and leave the others unchecked, you will get in full size preview a offset page

*Mouse Button Mode* this combobox you can select the button behavior of the mouse to rotate, move or zoom within the preview.

- left rotate, middle move, right zoom
- left zoom, middle move, right rotate
- left move, middle rotate, right zoom
- left zoom, middle rotate, right move
- left move, middle zoom, right rotate
- left rotate, middle zoom, right move

Default is left move, middle zoom, right rotate.

The mouse wheel will still zoom the preview in every mode.

**Tip**
If you select an element in the preview, the selected element will be taken as rotation center point.

**File to load on start up**

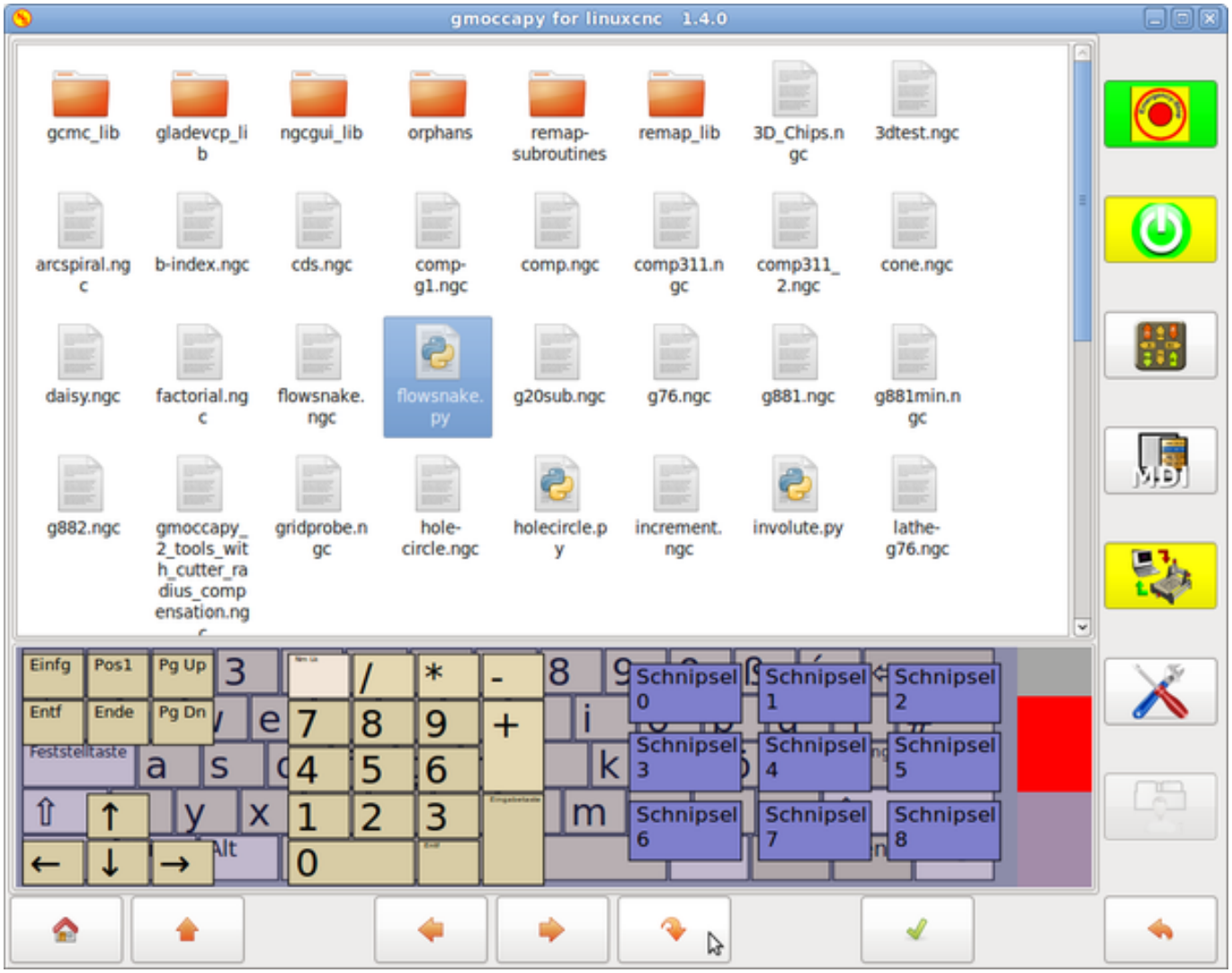
Select the file you want to be loaded on start up. In other GUI changing this was very cumbersome, because the users where forced to edit the INI File.

Select the file you want to be loaded on start up. If a file is loaded, it can be set by pressing the current button to avoid that any program is loaded at start up, just press the None button.

The file selection screen will use the filters you have set in the INI File, if there aren't any filters given, you will only see **ngc** files. The path will be set according to the INI settings in [DISPLAY] PROGRAM_PREFIX

**Jump to dir**

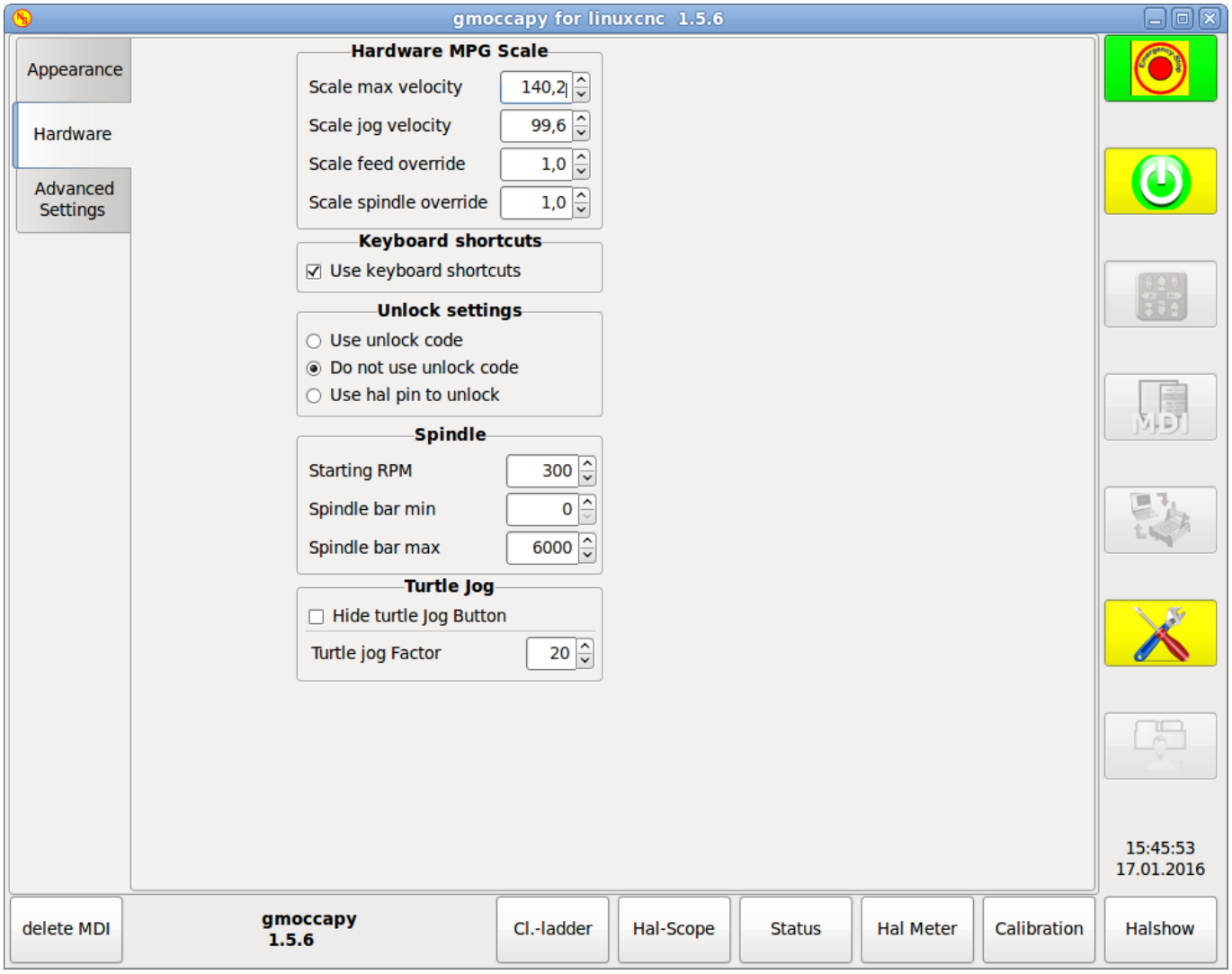
you can set here the directory to jump to if you press the corresponding button in the file selection dialog.




**Themes and Sounds**

This lets the user select what desktop theme to apply and what error and messages sounds should be played. By default "Follow System Theme" is set.

#### 2.2.7.2 Hardware




**Hardware MPG Scales**

For the different Hal Pin to connect MPG Wheels to, you may select individual scales to be applied. The main reason for this was my own test to solve this through hal connections, resulting in a very complex hal file. Imagine a user having an MPG Wheel with 100 ipr and he wants to slow down the max vel from 14000 to 2000 mm/min, that needs 12000 impulses, resulting in 120 turns of the wheel! Or an other user having a MPG Wheel with 500 ipr and he wants to set the spindle override witch has limits from 50 to 120 % so he goes from min to max within 70 impulses, meaning not even 1/4 turn.

By default all scales are set using the calculation:

```
(MAX - MIN)/100
```

**Keyboard shortcuts**

Some users want to jog there machine using the keyboard buttons and there are others that will never allow this. So everybody can select whether to use them or not.

Default is to use keyboard shortcuts.

Please take care if you use a lathe, than the shortcuts will be different. See the Lathe section

- Arrow Left or NumPad_Left = X minus

- Arrow Right or NumPad_Right = X plus
- Arrow up NumPad_Up = Y plus
- Arrow Down NumPad_Down = Y minus
- Page Up NumPad_Page_Up = Z plus
- Page Down NumPad_Page_Down = Z minus
- F1 = Estop (will work even if keyboard shortcuts are disabled)
- F2 = Machine on
- ESC = Abort

There are additional keys for message handling, see Message behavior and appearance

- WINDOWS = Delete last message
- <STRG><SPACE> = Delete all messages

**Unlock options**

There are three options to unlock the settings page:

- use unlock code (the user must give a code to get in)
- Do not use unlock code (There will be no security check)
- Use hal pin to unlock (hardware pin must be high to unlock the settings, see hardware unlock pin

Default is use unlock code (default = **123**)

**Spindle**

The start RPM sets the rpm to be used if the spindle is started and no S value has been set.

The start RPM sets the rpm to be used if the spindle is started and no S value has been set.

With the MIN and MAX settings you set the limits of the spindle bar shown in the INFO frame on the main screen. It is no error giving wrong values. If you give a maximum of 2000 and your spindle makes 4000 rpm, only the bar level will be wrong on higher speeds than 2000 rpm.

```
default values are
MIN = 0
MAX = 6000
```

**Turtle Jog**

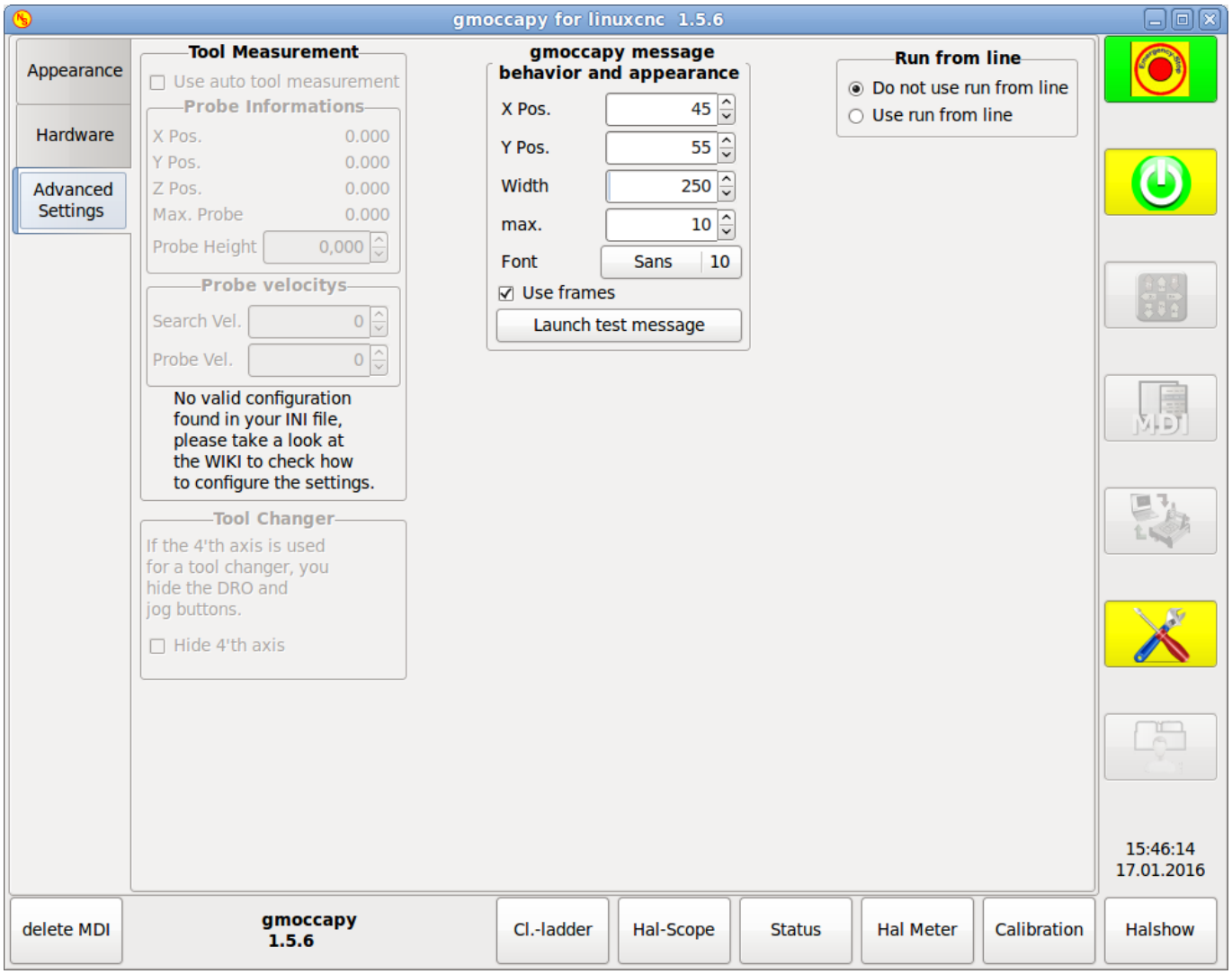
This settings will have influence on the jog velocities.

- *hide turtle jog button* will hide the button right of the jog velocity slider, if you hide this button, please take care that it shows the rabbit icon, otherwise you will not be able to jog faster than the turtle jog velocity, which is calculated using the turtle jog factor.
- *Turtle jog factor* sets the scale to apply for turtle jog mode. If you set a factor of 20, the max jog velocity will be 1/20 of max velocity of the machine if in turtle mode (button pressed, showing the turtle)

**Note**

This button can be activated using the turtle-jog hal pin.

### 2.2.7.3 Advanced Settings




**Tool Measurement**

If this part is not sensitive, you do not have a valid INI file configuration to use tool measurement.

Please check Auto Tool Measurement

- Use auto tool measurement : If checked, after each tool change, a tool measurement will be done, the result will be stored in the tool table and an G43 will be executed after the change.

**Probe Information** The following informations are taken from your INI file and must be given in absolute coordinates

- X Pos. = The X position of the tool switch
- Y Pos. = The Y position of the tool switch
- Z Pos. = The Z position of the tool switch, we will go as rapid move to this coordinate
- Max. Probe = is the distance to search for contact, an error will be launched, if no contact is given. The distance has to be given in relative coordinates, beginning the move from Z Pos., so you have to give a negative value to go down!
- Probe Height = is the height of your probe switch, you can measure it. Just touch off the base where the probe switch is located and set that to zero. Then make a tool change and watch the tool_offset_z value, that is the hight you must enter here.

PROBE VELOCITIES

- Search Vel. = The velocity to search for the tool switch, after contact the tool will go up again and then goes toward the probe again with probe vel, so you will get better results.
- Probe Vel. = Is the velocity for the second movement to the switch, it should be slower to get better touch results.(In sim mode, this is commented out in macros/change.ngc, otherwise the user would have to click twice on the probe button)

**Tool Changer** If your 4'th axis is used in a tool changer, you may want to hide the DRO and all the other buttons related to that axis.

You can do that by checking the checkbox, that will hide:

- 4'th axis DRO
- 4'th axis Jog button
- 4'th axis home button
- column of 4'th axis in the offsetpage
- column of 4'th axis in the tolleditor

**Message Behavior And Appearance**

This will display small pop up windows displaying the message or error text, the behavior is very similar to the one axis uses. You can delete a specific message, by clicking on it's close button, if you want to delete the last one, just hit the WINDOWS key on your keyboard, or delete all messages at ones with <STRG><SPACE>.

You are able to set some options:

- X Pos = The position of the top left corner of the message in X counted in pixel from the top left corner of the screen.
- Y Pos = The position of the top left corner of the message in Y counted in pixel from the top left corner of the screen.
- Width = The width of the message box
- max = The maximum messages you want to see at ones, if you set this to 10, the 11th message will delete the first one, so you will only see the last 10 ones.
- Font = The font and size you want to use to display the messages
- use frames = If you activate the checkbox, each message will be displayed in a frame, so it is much easier to distinguish the messages. But you will need a little bit more space.
- The button launch test message will just do what it is supposed to, it will show a message, so you can see the changes of your settings without the need to generate an error.

**Run From Line Option** You can allow or disallow the run from line. This will set the corresponding button insensitive (grayed out), so the user will not be able to use this option. The default is disable run from line.

**Warning**
It is not recommend to use run from line, as LinuxCNC will not take care of any previous lines in the code before the starting line. So errors or crashes are very probable.

**Log Actions** If this button is active, nearly every button press or relevant action of LinuxCNC will be logged in the ALARM history. This is very useful for debugging.

### 2.2.8 Lathe Specific Section

If in the INI File LATHE = 1 is given, the GUI will change its appearance to the special needs for a lathe. Mainly the Y axis will be hidden and the jog buttons will be arranged in a different order.

Figure 2.13: Normal Lathe

Figure 2.14: Back Tool Lathe

As you see the R DRO has a black background and the D DRO is gray. This will change according to the active G-Code G7 or G8. The active mode is visible by the black background, meaning in the shown images G8 is active.

The next difference to the standard screen is the location of the Jog Button. X and Z have changed places and Y is gone. You will note that the X+ and X- buttons changes there places according to normal or back tool lathe.

Also the keyboard behavior will change:

Normal Lathe:

- Arrow Left NumPad_Left = Z minus
- Arrow Right NumPad_Right = Z plus
- Arrow up NumPad_Up = X minus
- Arrow Down NumPad_Down = X plus

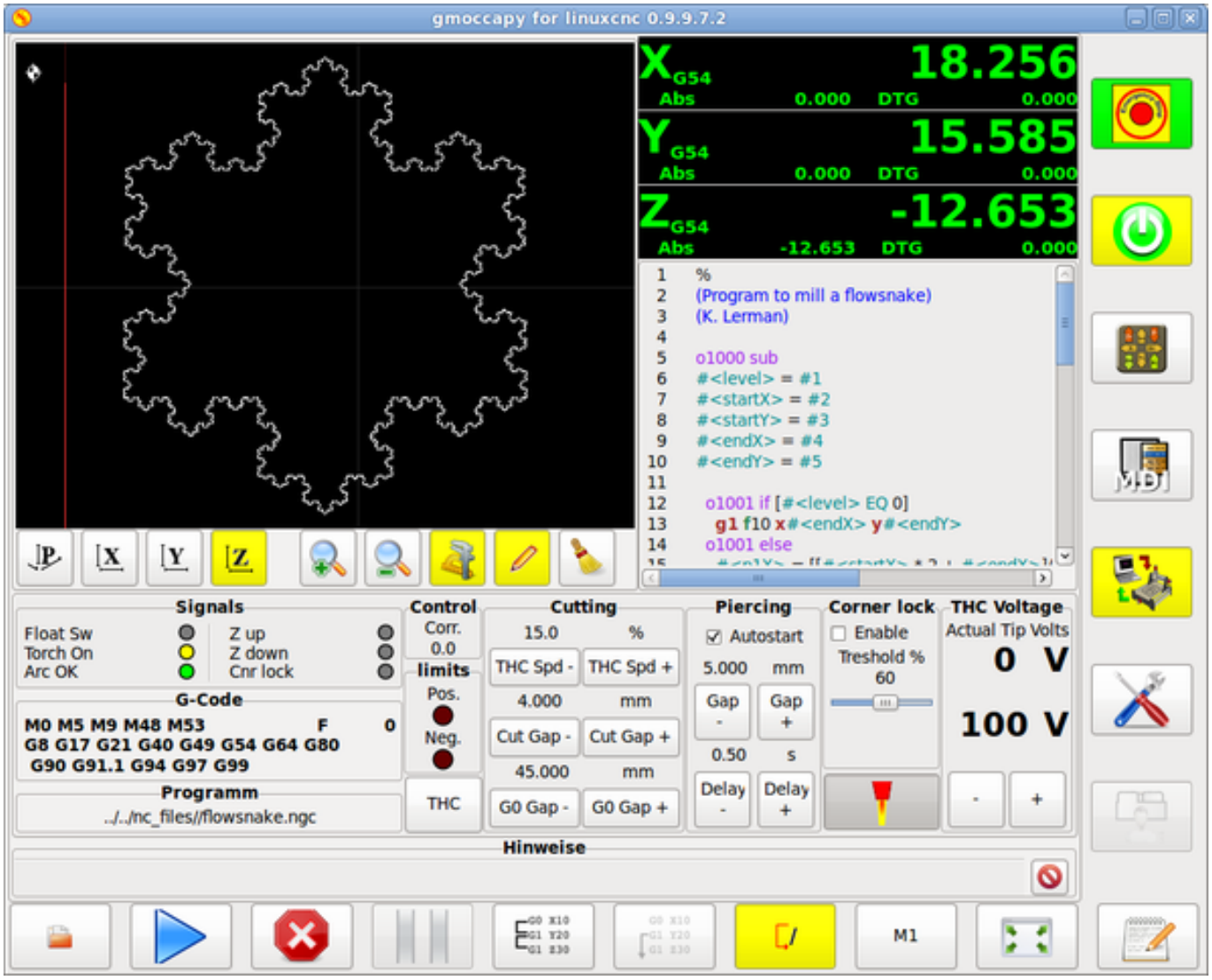
Back Tool Lathe:

- Arrow Left NumPad_Left = Z minus

- Arrow Right NumPad_Right = Z plus
- Arrow up NumPad_Up = X plus
- Arrow Down NumPad_Down = X minus

The tool information frame will show not only the Z offset, but also the X offset and the tool table is showing all lathe relevant information.

### 2.2.9 Plasma Specific Section




There is a very good WIKI, which is actually growing, maintained by Marius see Plasma wiki page

### 2.2.10 Video On You Tube

This are videos showing gmoccapy in action, unfortunately the videos don't show the latest version of gmoccapy, but the way to use it will not change much in the future. I will try to actualize the videos as soon as possible.

#### 2.2.10.1 Basic Usage

https://www.youtube.com/watch?v=O5B-s3uiI6g

#### 2.2.10.2 Simulated Jog Wheels

http://youtu.be/ag34SGxt97o

#### 2.2.10.3 Settings Page

https://www.youtube.com/watch?v=AuwhSHRJoiI

#### 2.2.10.4 Simulated Hardware Button

Deutsch = http://www.youtube.com/watch?v=DTqhY-MfzDE

English = http://www.youtube.com/watch?v=ItVWJBK9WFA

#### 2.2.10.5 User Tabs

http://www.youtube.com/watch?v=rG1zmeqXyZI

#### 2.2.10.6 Tool Measurement Video

Auto Tool Measurement Simulation = http://youtu.be/rrkMw6rUFdk

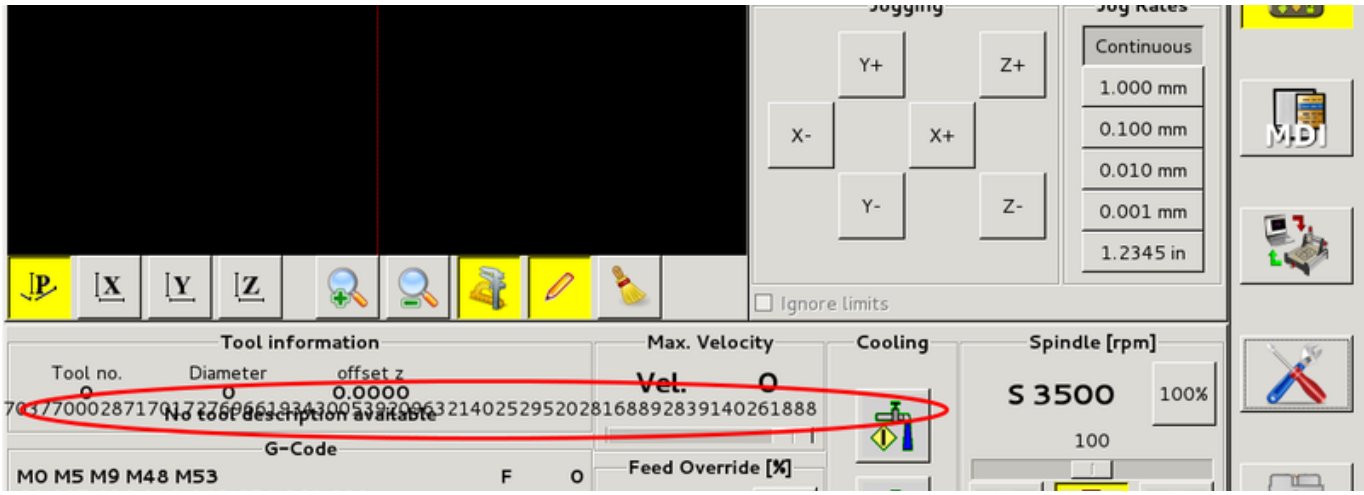
Auto Tool Measurement Screen = http://youtu.be/Z2ULDj9dzvk

Auto Tool Measurement Machine = http://youtu.be/1arucCaDdX4

### 2.2.11 Known problems

#### 2.2.11.1 Strange numbers in the info area

If you get strange numbers in the info area of gmoccapy like:




You have made your config file using an older version of StepConfWizard. It has made a wrong entry in the INI file under the [TRAJ] named MAX_LINEAR_VELOCITY = xxx. Change that entry to MAX_VELOCITY = xxx

#### 2.2.11.2 Not ending macro

If you use a macro without movement, like this one:

```
 o<zeroxy> sub

G92.1
G92.2
G40

G10 L20 P0 X0 Y0

o<zeroxy> endsub
m2
```

gmoccapy will not see the end of the macro, because the interpreter needs to change its state to IDLE, but the macro does not even set the interpreter to a new state. To avoid that just add a G4 P0.1 line to get the needed signal. The correct macro would be:

```
 o<zeroxy> sub

G92.1
G92.2
G40

G10 L20 P0 X0 Y0

G4 P0.1

o<zeroxy> endsub
m2
```

## 2.3 NGCGUI

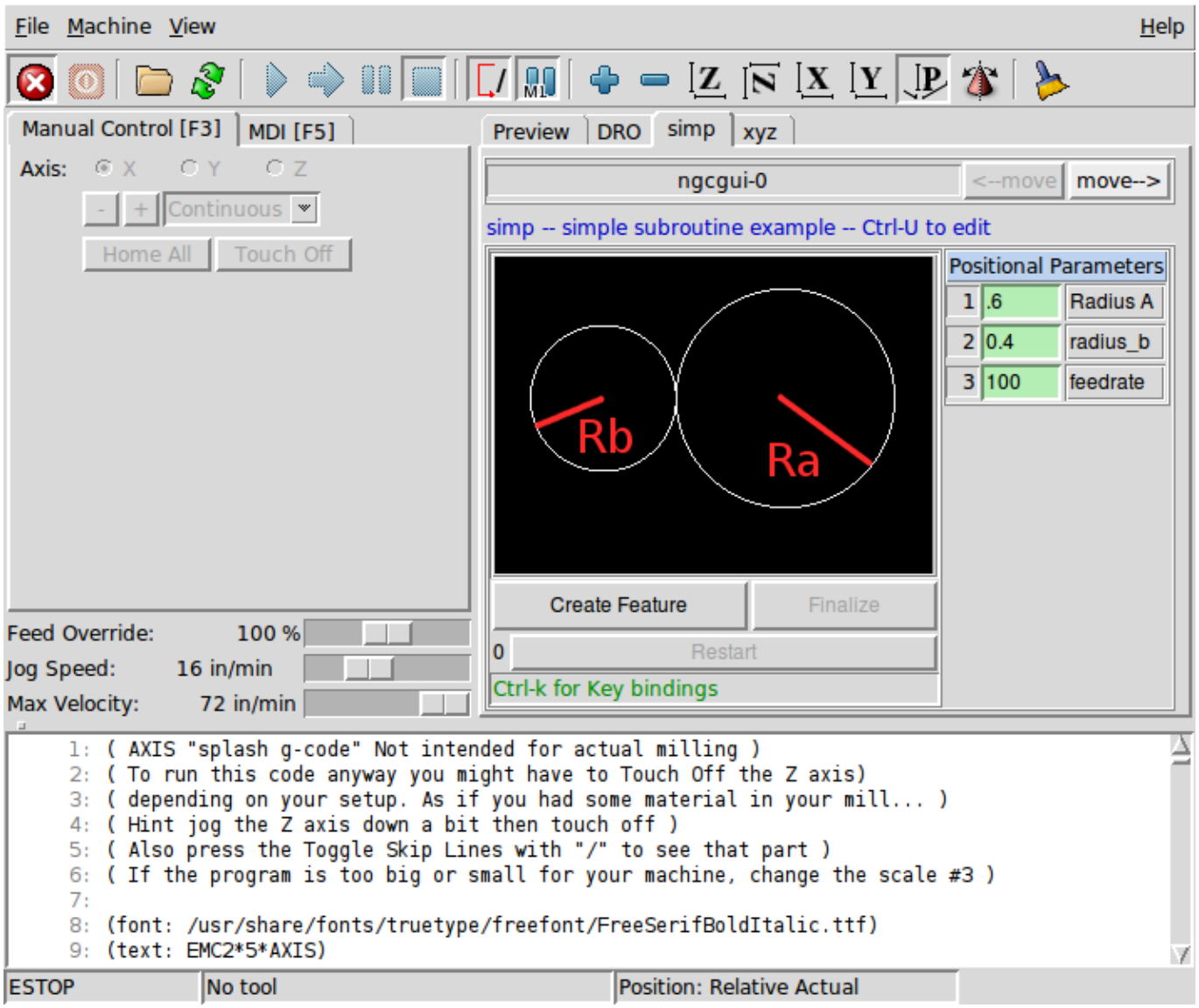



Figure 2.15: NGCGUI imbedded into Axis

### 2.3.1 Overview

- *NGCGUI* is a Tcl application to work with subroutines. It allows you to have a conversational interface with LinuxCNC. You can organize the subroutines in the order you need them to run and concatenate the subroutines into one file for a complete part program.
- *NGCGUI* can run as a standalone application or can be embedded in multiple tab pages in the axis GUI
- *PYNGCGUI* is an alternate, python implementation of ngcgui.
- *PYNGCGUI* can run as a standalone application or can be embedded as a tab page (with its own set of multiple subroutine tabs) in any GUI that supports embedding of gladevcp applications axis, touchy, gscreen and gmoccapy.

Using NGCGUI or PYNGCGUI:

- Tab pages are provided for each subroutine specified in the INI file
- New subroutines tab pages can be added on the fly using the custom tab
- Each subroutine tab page provides entry boxes for all subroutine parameters
- The entry boxes can have a default value and an label that are identified by special comments in the subroutine file
- Subroutine invocations can be concatenated together to form a multiple step program
- Any single-file G code subroutine that conforms to ngcgui conventions can be used
- Any gcmc (G code-meta-compiler) program that conforms to ngcgui conventions for tagging variables can be used. (The gcmc executable must be installed separately, see: http://www.vagrearg.org/content/gcmc)

**Note**
NGCGUI and PYNGCGUI implement the same functions and both process .ngc and .gcmc files that conform to a few ngcgui-specific conventions. In this document, the term *NGCGUI* generally refers to either application.

### 2.3.2 Demonstration Configurations

A number of demonstration configurations are located in the sim directory of the Sample Configurations offered by the LinuxCNC configuration picker. The configuration picker is on the system's main menu:

```
CNC > LinuxCNC
```

Examples are included for the axis, touchy, gscreen, and gmoccapy. These examples demonstrate both 3-axis (XYZ) cartesian configurations (like mills) and lathe (XZ) setups. Some examples show the use of a pop up keyboard for touch screen systems and other examples demonstrate the use of files created for the gcmc (G code Meta Compiler) application. The touchy examples also demonstrate incorporation of a gladevcp back plot viewer (gremlin_view).

The simplest application is found as:

```
Sample Configurations/sim/axis/ngcgui /ngcgui_simple
```

A comprehensive example showing gcmc compatibility is at:

```
Sample Configurations/sim/axis/ngcgui/ngcgui_gcmc
```

A comprehensive example embedded as a gladevcp app and using gcmc is at:

```
Sample Configurations/sim/gscreen/ngcgui/pyngcgui_gcmc
```

The example sim configurations make use of library files that provide example G code subroutine (.ngc) files and G code-meta-compiler (.gcmc) files:

- *nc_files/ngcgui_lib*
  - *arc1.ngc* - basic arc using cutter radius compensation
  - *arc2.ngc* - arc speced by center, offset, width, angle (calls arc1)
  - *backlash.ngc* - routine to measure an axis backlash with dial indicator
  - *db25.ngc* - creates a DB25 plug cutout
  - *gosper.ngc* - a recursion demo (flowsnake)
  - *helix.ngc* - helix or D-hole cutting

  - *helix_rtheta.ngc* - helix or D-hole positioned by radius and angle
  - *hole_circle.ngc* - equally spaced holes on a circle
  - *ihex.ngc* - internal hexagon
  - *iquad.ngc* - internal quadrilateral
  - *ohex.ngc* - outside hexagon
  - *oquad.ngc* - outside quadrilateral
  - *qpex_mm.ngc* - demo of qpockets (mm based)
  - *qpex.ngc* - demo of qpockets (inch based)
  - *qpocket.ngc* - quadrilateral pocket
  - *rectangle_probe.ngc* - probe a rectangular area
  - *simp.ngc* - a simple subroutine example that creates two circles
  - *slot.ngc* - slot from connecting two endpoints
  - *xyz.ngc* - machine exerciser constrained to a box shape
- *nc_files/ngcgui_lib/lathe*
  - *g76base.ngc* - gui for g76 threading
  - *g76diam.ngc* - threading speced by major, minor diameters
  - *id.ngc* - bores the inside diameter
  - *od.ngc* - turns the outside diameter
  - *taper-od.ngc* - turns a taper on the outside diameter
- *nc_files/gcmc_lib*
  - *drill.gcmc* - drill holes in rectangle pattern
  - *square.gcmc* - simple demo of variable tags for gcmc files
  - *star.gcmc* - gcmc demo illustrating functions and arrays
  - *wheels.gcmc* - gcmc demo of complex patterns

To try a demonstration, select a sim configuration and start the linuxCNC program.

If using the axis gui, press the *E-Stop* then *Machine Power* then *Home All*. Pick a ngcgui tab, fill in any empty blanks with sensible values and press *Create Feature* then *Finalize*. Finally press the *Run* button to watch it run. Experiment by creating multiple features and features from different tab pages.

To create several subroutines concatinated into a single file, go to each tab fill in the blanks, press *Create Feature* then using the arrow keys move any tabs needed to put them in order. Now press *Finalize* and answer the prompt to create

Other guis will have similar functionality but the buttons and names may be different.

**Notes**

The demonstration configs create tab pages for just a few of the provided examples. Any gui with a custom tab can open any of the library example subroutines or any user file if it is in the linuxCNC subroutine path.

To see special key bindings, click inside an ngcgui tab page to get focus and then presss Control-k.

The demonstration subroutines should run on the simulated machine configurations included in the distribution. A user should always understand the behavior and purpose of a program before running on a real machine.

### 2.3.3 Library Locations

In linuxCNC installations installed from deb packages, the simulation configs for ngcgui use symbolic links to non-user-writable LinuxCNC libraries for:

- *nc_files/ngcgui_lib* ngcgui-compatible subfiles
- *nc_files/ngcgui_lib/lathe* ngcgui-compatible lathe subfiles
- *nc_files/gcmc_lib* ngcgui-gcmc-compatible programs
- *nc_files/ngcgui_lib/utilitysubs* Helper subroutines
- *nc_files/ngcgui_lib/mfiles* User M files

These libraries are located by ini file items that specify the search paths used by linuxCNC (and ngcgui):

```
[RS274NGC]
SUBROUTINE_PATH = ../../nc_files/ngcgui_lib:../../nc_files/gcmc_lib:../../nc_files/ ↩
    ngcgui_lib/utilitysubs
USER_M_PATH     = ../../nc_files/ngcgui_lib/mfiles
```

**Note**
These are long lines (not continued on multiple lines) that specify the directories used in a search patch. The directory names are separated by colons (:). No spaces should occur between directory names.

A user can create new directories for their own subroutines and M-files and add them to the search path(s).

For example, a user could create directories from the terminal with the commands:

```
mkdir /home/myusername/mysubs
mkdir /home/myusername/mymfiles
```

And then create or copy system-provided files to these user-writable directories. For instance, a user might create a ngcgui-compatible subfile named:

```
/home/myusername/mysubs/example.ngc
```

To use files in new directories, the ini file must be edited to include the new subfiles and to augment the search path(s). For this example:

```
[RS274NGC]
...
SUBROUTINE_PATH = /home/myusername/mysubs:../../nc_files/ngcgui_lib:../../nc_files/gcmc_lib ↩
    :../../nc_files/ngcgui_lib/utilitysubs
USER_M_PATH     = /home/myusername/mymfiles:../../nc_files/ngcgui_lib/mfiles

[DISPLAY]
...
NGCGUI_SUBFILE = example.ngc
...
```

LinuxCNC (and ngcgui) use the first file found when searching directories in the search path. With this behavior, you can supersede an ngcgui_lib subfile by placing a subfile with an identical name in a directory that is found earlier in the path search. More information can be found in the INI chapter of the Integrators Manual.

### 2.3.4 Standalone Usage

#### 2.3.4.1 Standalone NGCGUI

For usage, type in a terminal:

```
ngcgui --help
Usage:
  ngcgui --help | -?
  ngcgui [Options] -D nc_files_directory_name
  ngcgui [Options] -i LinuxCNC_inifile_name
  ngcgui [Options]

  Options:
         [-S subroutine_file]
         [-p preamble_file]
         [-P postamble_file]
         [-o output_file]
         [-a autosend_file]            (autosend to axis default:auto.ngc)
         [--noauto]                    (no autosend to axis)
         [-N | --nom2]                 (no m2 terminator (use %))
         [--font [big|small|fontspec]] (default: "Helvetica -10 normal")
         [--horiz|--vert]              (default: --horiz)
         [--cwidth comment_width]      (width of comment field)
         [--vwidth varname_width]      (width of varname field)
         [--quiet]                     (fewer comments in outfile)
         [--noiframe]                  (default: frame displays image)
```

**Note**
As a standalone application, ngcgui handles a single subroutine file which can be invoked multiple times. Multiple standalone ngcgui applications can be started independently.

#### 2.3.4.2 Standalone PYNGCGUI

For usage, type in a terminal:

```
pyngcgui --help
Usage:
pyngcgui [Options] [sub_filename]
Options requiring values:
    [-d | --demo] [0|1|2] (0: DEMO standalone toplevel)
                          (1: DEMO embed new notebook)
                          (2: DEMO embed within existing notebook)
    [-S | --subfile       sub_filename]
    [-p | --preamble      preamble_filename]
    [-P | --postamble     postamble_filename]
    [-i | --ini           inifile_name]
    [-a | --autofile      auto_filename]
    [-t | --test          testno]
    [-K | --keyboardfile  glade_file] (use custom popupkeyboard glade file)
Solo Options:
    [-v | --verbose]
    [-D | --debug]
    [-N | --nom2]         (no m2 terminator (use %))
    [-n | --noauto]       (save but do not automatically send result)
    [-k | --keyboard]     (use default popupkeybaord)
    [-s | --sendtoaxis]   (send generated ngc file to axis gui)
Notes:
```

```
      A set of files is comprised of a preamble, subfile, postamble.
      The preamble and postamble are optional.
      One set of files can be specified from cmdline.
      Multiple sets of files can be specified from an inifile.
      If --ini is NOT specified:
         search for a running linuxCNC and use its inifile
```

**Note**
As a standalone application, pyngcgui can read an ini file (or a running linuxCNC application) to create tab pages for multiple subfiles.

## 2.3.5 Embedding NGCGUI

### 2.3.5.1 Embedding NGCGUI in Axis

The following INI file items go in the [DISPLAY] section. (See additional sections below for additional items needed)

- *TKPKG = Ngcgui 1.0* - the NGCGUI package
- *TKPKG = Ngcguittt 1.0* - the True Type Tracer package for generating text for engraving (optional, must follow TKPKG = Ngcgui).
- *TTT = truetype-tracer* - name of the truetype tracer program (it must be in user PATH)
- *TTT_PREAMBLE = in_std.ngc* - Optional, specifies filename for preamble used for ttt created subfiles. (alternate: mm_std.ngc)

**Note**
The optional truetype tracer items are used to specify an ngcgui-compatible tab page that uses the application truetype-tracer. The truetype-tracer application must be installed independently and located in the user PATH.

### 2.3.5.2 Embedding PYNGCGUI as a gladevcp tab page in a gui

The following INI file items go in the [DISPLAY] section for use with the axis, gscreen, or touchy guis. (See additional sections below for additional items needed)

**EMBED_ Items**

```
EMBED_TAB_NAME = Pyngcgui - name to appear on embedded tab
EMBED_TAB_COMMAND = gladevcp -x {XID} pyngcgui_axis.ui - invokes gladevcp
EMBED_TAB_LOCATION = name_of_location - where the embeded page is located
```

**Note**
The EMBED_TAB_LOCATION specifier is not used for the axis gui. While pyngcgui can be embedded in axis, integration is more complete when using ngcgui (using TKPKG = Ngcgui 1.0). To specify the EMBED_TAB_LOCATION for other guis, see the DISPLAY Section of the INI Configuration Chapter.

**Note**
The truetype tracer gui front-end is not currently available for gladevcp applications.

#### 2.3.5.3 Additional INI File items required for ngcgui or pyngcgui

The following INI file items go in the [DISPLAY] section for any gui that embeds either ngcgui or pyngcgui.

- *NGCGUI_FONT = Helvetica -12 normal* - specifices the font name,size, normal|bold
- *NGCGUI_PREAMBLE = in_std.ngc* - the preamble file to be added in front of the subroutines. When concatenating several common subroutine invocations, this preamble is only added once. For mm-based machines, use mm_std.ngc
- *NGCGUI_SUBFILE = filename1.ngc* - creates a tab from the filename1 subroutine
- *NGCGUI_SUBFILE = filename2.ngc* - creates a tab from the filename2 subroutine
- *... etc*
- *NGCGUI_SUBFILE = gcmcname1.gcmc* - creates a tab from the gcmcname1 file
- *NGCGUI_SUBFILE = gcmcname2.gcmc* - creates a tab from the gcmcname2 file
- *... etc*
- *NGCGUI_SUBFILE = ""* - creates a custom tab that can open any subroutine in the search path
- *NGCGUI_OPTIONS = opt1 opt2 ...* - NGCGUI options
  - *nonew* - disallow making a new custom tab
  - *noremove* - disallow removing any tab page
  - *noauto* - no autosend (use makeFile, then save or manually send)
  - *noiframe* - no internal image, display images on separate top level widget
  - *nom2* - do not terminate with m2, use % terminator. This option eliminates all the side effects of m2 termination
- *GCMC_INCLUDE_PATH = dirname1:dirname2* - search directories for gcmc include files

This is an example of embedding NGCGUI into Axis. The subroutines need to be in a directory specified by the [RS274NGC]SUBROUT
Some example subroutines use other subroutines so check to be sure you have the dependences, if any, in a SUBROUTINE_PATH directory. Some subroutines may use custom Mfiles which must be in a directory specified by the [RS274NGC]USER_M_PATH.

The Gcode-meta-compiler (gcmc) can include statements like: include("filename.inc.gcmc"); By default, gcmc includes the current directory which, for linuxCNC, will be the directory containing the linuxCNC ini file. Additional directories can be prepended to the gcmc search order with the GCMC_INCLUDE_PATH item.

**Sample axis-gui-based INI**

```
[RS274NGC]
...
SUBROUTINE_PATH   = ../../nc_files/ngcgui_lib:../../ngcgui_lib/utilitysubs
USER_M_PATH       = ../../nc_files/ngcgui_lib/mfiles

[DISPLAY]
TKPKG             = Ngcgui    1.0
TKPKG             = Ngcguittt 1.0
# Ngcgui must precede Ngcguittt

NGCGUI_FONT       = Helvetica -12 normal
# specify filenames only, files must be in [RS274NGC]SUBROUTINE_PATH
NGCGUI_PREAMBLE   = in_std.ngc
NGCGUI_SUBFILE    = simp.ngc
NGCGUI_SUBFILE    = xyz.ngc
NGCGUI_SUBFILE    = iquad.ngc
NGCGUI_SUBFILE    = db25.ngc
NGCGUI_SUBFILE    = ihex.ngc
NGCGUI_SUBFILE    = gosper.ngc
```

```
# specify "" for a custom tab page
NGCGUI_SUBFILE    = ""
#NGCGUI_SUBFILE   = "" use when image frame is specified if
#                      opening other files is required
#                      images will be put in a top level window
NGCGUI_OPTIONS    =
#NGCGUI_OPTIONS   = opt1 opt2 ...
# opt items:
#   nonew      -- disallow making a new custom tab
#   noremove   -- disallow removing any tab page
#   noauto     -- no auto send (makeFile, then manually send)
#   noiframe   -- no internal image, image on separate top level
GCMC_INCLUDE_PATH = /home/myname/gcmc_includes

TTT               = truetype-tracer
TTT_PREAMBLE      = in_std.ngc

PROGRAM_PREFIX    = ../../nc_files
```

**Note**
The above is not a complete axis gui INI — the items show are those used by ngcgui. Many additional items are required by LinuxCNC to have a complete INI file.

#### 2.3.5.4 Truetype Tracer

Ngcgui_ttt provides support for truetype-tracer (v4). It creates an axis tab page which allows a user to create a new ngcgui tab page after entering text and selecting a font and other parameters. (Truetype-tracer must be installed independently).

To embed ngcgui_ttt in axis, specify the following items in addition to ngcgui items:

```
Item:    [DISPLAY]TKPKG = Ngcgui_ttt version_number
Example: [DISPLAY]TKPKG = Ngcgui_ttt 1.0
Note:    Mandatory, specifies loading of ngcgui_ttt in an axis tab page named ttt.
         Must follow the TKPKG = Ngcgui item.

Item:    [DISPLAY]TTT = path_to_truetype-tracer
Example: [DISPLAY]TTT = truetype-tracer
Note:    Optional, if not specified, attempt to use /usr/local/bin/truetype-tracer.
         Specify with absolute pathname or as a simple executable name
         in which case the user PATH environment will used to find the program.

Item:    [DISPLAY]TTT_PREAMBLE = preamble_filename
Example: [DISPLAY]TTT_PREAMBLE = in_std.ngc
Note:    Optional, specifies filename for preamble used for ttt created subfiles.
```

#### 2.3.5.5 INI File Path Specifications

Ngcgui uses the linuxCNC search path to find files.

The search path begins with the standard directory specified by:

```
[DISPLAY]PROGRAM_PREFIX = directory_name
```

followed by multiple directories specified by:

```
[RS274NGC]SUBROUTINE_PATH = directory1_name:directory1_name:directory3_name ...
```

Directories may be specified as absolute paths or relative paths.

```
Example: [DISPLAY]PROGRAM_PREFIX = /home/myname/linuxcnc/nc_files
Example: [DISPLAY]PROGRAM_PREFIX = ~/linuxcnc/nc_files
Example: [DISPLAY]PROGRAM_PREFIX = ../../nc_files
```

An absolute path beginning with a "/" specifies a complete filesystem location. A path beginning with a "~/" specifies a path starting from the user's home directory. A path beginning with "~username/" specifies a path starting in username's home directory.

**Relative Paths** Relative paths are based on the startup directory which is the directory containing the INI file. Using relative paths can facilitate relocation of configurations but requires a good understanding of linux path specifiers.

```
./d0        is the same as d0, e.g., a directory named d0 in the startup directory
../d1       refers to a directory d1 in the parent directory
../../d2    refers to a directory d2 in the parent of the parent directory
../../../d3 etc.
```

Multiple directories can be specified with [RS274NGC]SUBROUTINE_PATH by separating them with colons. The following example illustrates the format for multiple directories and shows the use of relative and absolute paths.

**Multiple Directories Example:**

```
[RS274NGC]SUBROUTINE_PATH = ../../nc_files/ngcgui_lib:../../nc_files/ngcgui_lib/utilitysubs ↩
    :/tmp/tmpngc`
```

This is one long line, do not continue on multiple lines. When linuxCNC and/or ngcgui searches for files, the first file found in the search is used.

LinuxCNC (and ngcgui) must be able to find all subroutines including helper routines that are called from within ngcgui subfiles. It is convenient to place utility subs in a separate directory as indicated in the example above.

The distribution includes the ngcgui_lib directory and demo files for preambles, subfiles, postambles and helper files. To modify the behavior of the files, you can copy any file and place it in an earlier part of the search path. The first directory searched is [DISPLAY]PROGRAM_PREFIX. You can use this directory but it is better practice to create dedicated directory(ies) and put them at the beginning of the [RS274NGC]SUBROUTINE_PATH.

In the following example, files in /home/myname/linuxcnc/mysubs will be found before files in ../../nc_files/ngcgui_lib.

**Adding User Directory Example:**

```
[RS274NGC]SUBROUTINE_PATH = /home/myname/linuxcnc/mysubs:../../nc_files/ngcgui_lib:../../ ↩
    nc_files/ngcgui_lib/utilitysubs`
```

New users may inadvertently try to use files that are not structured to be compatible with ngcgui requirements. Ngcgui will likely report numerous errors if the files are not coded per its conventions. Good practice suggests that ngcgui-compatible subfiles should be placed in a directory dedicated to that purpose and that preamble, postamble, and helper files should be in separate directory(ies) to discourage attempts to use them as subfiles. Files not intended for use as subfiles can include a special comment: "(not_a_subfile)" so that ngcgui will reject them automatically with a relevant message.

#### 2.3.5.6 Summary of INI File item details for NGCGUI usage

```
Item:    [RS274NGC]SUBROUTINE_PATH = dirname1:dirname2:dirname3 ...
Example: [RS274NGC]SUBROUTINE_PATH = ../../nc_files/ngcgui_lib:../../nc_files/ngcgui_lib/u
Note:    Optional, but very useful to organize subfiles and utility files

Item:    [RS274NGC]USER_M_PATH = dirname1:dirname2:dirname3 ...
```

```
Example: [RS274NGC]USER_M_PATH = ../../nc_files/ngcgui_lib/mfiles
Note:    Optional, needed to locate custom user mfiles


Item:    [DISPLAY]EMBED_TAB_NAME = name to display on embedded tab page
Example: [DISPLAY]EMBED_TAB_NAME = Pyngcgui
Note:    The entries: EMBED_TAB_NAME,EMBED_TAB_COMMAND,EMBED_TAB_LOCATION
         define an embedded application for several linuxCNC guis

Item:    [DISPLAY]EMBED_TAB_COMMAND = programname followed by arguments
Example: [DISPLAY]EMBED_TAB_COMMAND = gladevcp -x {XID} pyngcgui_axis.ui
Note:    For gladevcp applications, see the <<cha:glade-vcp,GladeVCP Chapter>>

Item:    [DISPLAY]EMBED_TAB_LOCATION = name_of_location
Example: [DISPLAY]EMBED_TAB_LOCATION = notebook_main
Note:    See example INI files for possible locations
         Not required for the axis gui


Item:    [DISPLAY]PROGRAM_PREFIX = dirname
Example: [DISPLAY]PROGRAM_PREFIX = ../../nc_files
Note:    Mandatory and needed for numerous linuxCNC functions
         It is the first directory used in the search for files


item:    [DISPLAY]TKPKG = Ngcgui version_number
Example: [DISPLAY]TKPKG = Ngcgui 1.0
Note:    Required only for axis gui embedding, specifies loading of ngcgui axis tab pages


Item:    [DISPLAY]NGCGUI_FONT = font_descriptor
Example: [DISPLAY]NGCGUI_FONT = Helvetica -12 normal
Note:    Optional, font_descriptor is a tcl-compatible font specifier
         with items for fonttype -fontsize fontweight
         Default is: Helvetica -10 normal
         Smaller font sizes may be useful for small screens
         Larger font sizes may be helpful for touch screen applications

Item:    [DISPLAY]NGCGUI_SUBFILE = subfile_filename
Example: [DISPLAY]NGCGUI_SUBFILE = simp.ngc
Example: [DISPLAY]NGCGUI_SUBFILE = square.gcmc
Example: [DISPLAY]NGCGUI_SUBFILE = ""
Note:    Use one or more items to specify ngcgui-compatible
         subfiles or gcmc programs that require a tab page on startup.
         A "Custom" tab will be created when the filename is "".
         A user can use a "Custom" tab to browse the file system
         and identify preamble, subfile, and postamble files.

Item:    [DISPLAY]NGCGUI_PREAMBLE = preamble_filename
Example: [DISPLAY]NGCGUI_PREAMBLE = in_std.ngc
Note:    Optional, when specified, the file is prepended to a subfile.
         Files created with "Custom" tab pages use the preamble specified
         with the page.

Item:    [DISPLAY]NGCGUI_POSTAMBLE = postamble_filename
Example: [DISPLAY]NGCGUI_POSTAMBLE = bye.ngc
Note:    Optional, when specified, the file is appended to a subfiles.
```

```
         Files created with "Custom" tab pages use the postamble specified
         with the page.

Item:    [DISPLAY]NGCGUI_OPTIONS = opt1 opt2 ...
Example: [DISPLAY]NGCGUI_OPTIONS = nonew noremove
Note:    Multiple options are separated by blanks.
         By default, ngcgui configures tab pages so that:
            1) a user can make new tabs
            2) a user can remove tabs (except for the last remaining one)
            3) finalized files are automatically sent to linuxCNC
            4) an image frame (iframe) is made available to display
               an image for the subfile (if an image is provided)
            5) the ngcgui result file sent to linuxCNC is terminated with
               an m2 (and incurs m2 side-effects)

         The options nonew, noremove, noauto, noiframe, nom2 respectively
         disable these default behaviors.

         By default, if an image (.png,.gif,jpg,pgm) file
         is found in the same directory as the subfile, the
         image is displayed in the iframe.  Specifying
         the noiframe option makes available additional buttons
         for selecting a preamble, subfile, and postamble and
         additional checkboxes.  Selections of the checkboxes
         are always available with special keys:
           Ctrl-R Toggle "Retain values on Subfile read"
           Ctrl-E Toggle "Expand subroutine"
           Ctrl-a Toggle "Autosend"
          (Ctrl-k lists all keys and functions)

         If noiframe is specified and an image file is found,
         the image is displayed in a separate window and
         all functions are available on the tab page.

         The NGCGUI_OPTIONS apply to all ngcgui tabs except that the
         nonew, noremove, and noiframe options are not applicable
         for "Custom" tabs.  Do not use "Custom" tabs if you want
         to limit the user's ability to select subfiles or create
         additional tab pages.

Item:    [DISPLAY]GCMC_INCLUDE_PATH = dirname1:dirname2:...
Example: [DISPLAY]GCMC_INCLUDE_PATH = /home/myname/gcmc_includes:/home/myname/gcmc_include
Note:    Optional, each directory will be included when gcmc is invoked
         using the option: --include dirname
```

### 2.3.6 File Requirements for NGCGUI Compatibility

#### 2.3.6.1 Single-File Gcode (.ngc) Subroutine Requirements

An NGCGUI-compatible subfile contains a single subroutine definition. The name of the subroutine must be the same as the filename (not including the .ngc suffix). LinuxCNC supports named or numbered subroutines, but only named subroutines are compatible with NGCGUI. For more information see the O-Codes Chapter.

The first non-comment line should be a sub statement. The last non-comment line should be a endsub statement.

**examp.ngc:**

```
(info: info_text_to_appear_at_top_of_tab_page)
; comment line beginning with semicolon
( comment line using parentheses)
o<examp> sub
  BODY_OF_SUBROUTINE
o<examp> endsub
; comment line beginning with semicolon
( comment line using parentheses)
```

The body of the subroutine should begin with a set of statements that define local named parameters for each positional parameter expected for the subroutine call. These definitions must be consecutive beginning with #1 and ending with the last used parameter number. Definitions must be provided for each of these parameters (no omissions).

**Parameter Numbering**

```
#<xparm> = #1
#<yparm> = #2
#<zparm> = #3
```

LinuxCNC considers all numbered parameters in the range #1 thru #30 to be calling parameters so ngcgui provides entry boxes for any occurence of parameters in this range. It is good practice to avoid use of numbered parameters #1 through #30 anywhere else in the subroutine. Using local, named parameters is recommended for all internal variables.

Each defining statement may optionally include a special comment and a default value for the parameter.

**Statement Prototype**

```
#<vname> = #n (=default_value)
or
#<vname> = #n (comment_text)
or
#<vname> = #n (=default_value comment_text)
```

**Parameter Examples**

```
#<xparm> = #1 (=0.0)
#<yparm> = #2 (Ystart)
#<zparm> = #3 (=0.0 Z start setting)
```

If a default_value is provided, it will be entered in the entry box for the parameter on startup.

If comment_text is included, it will be used to identify the input instead of the parameter name.

**Global Named Parameters** Notes on global named parameters and ngcgui:

(global named parameters have a leading underscore in the name, like #<_someglobalname>)

As in many programming languages, use of globals is powerful but can often lead to unexpected consequences. In LinuxCNC, existing global named parameters will be valid at subroutine execution and subroutines can modify or create global named parameters.

Passing information to subroutines using global named parameters is discouraged since such usage requires the establishment and maintenance of a well-defined global context that is difficult to maintain. Using numbered parameters #1 thru #30 as subroutine inputs should be sufficient to satisfy a wide range of design requirements.

While input global named parameters are discouraged, linuxCNC subroutines must use global named parameters for returning results. Since ngcgui-compatible subfiles are aimed at gui usage, return values are not a common requirement. However, ngcgui is useful as a testing tool for subroutines which do return global named parameters and it is common for ngcgui-compatible subfiles to call utility subroutine files that return results with global named parameters.

To support these usages, ngcgui ignores global named parameters that include a colon (:) character in their name. Use of the colon (:) in the name prevents ngcgui from making entryboxes for these parameters.

**Global Named Parameters**

```
o<examp> sub
...
#<_examp:result> = #5410       (return the current tool diameter)
...
o<helper> call [#<x1>] [#<x2>] (call a subroutine)
#<xresult> = #<_helper:answer> (immediately localize the helper global result)
#<_helper:answer> = 0.0        (nullify global named parameter used by subroutine)
...
o<examp> endsub
```

In the above example, the utility subroutine will be found in a separate file named helper.ngc. The helper routine returns a result in a global named parameter named #<_helper:answer.

For good practice, the calling subfile immediately localizes the result for use elsewhere in the subfile and the global named parameter used for returning the result is nullified in an attempt to mitigate its inadvertent use elsewhere in the global context. (A nullification value of 0.0 may not always be a good choice).

Ngcgui supports the creation and concatenation of multiple features for a subfile and for multiple subfiles. It is sometimes useful for subfiles to determine their order at runtime so ngcgui inserts a special global parameter that can be tested within subroutines. The parameter is named #<_feature:>. Its value begins with a value of 0 and is incremented for each added feature.

**Additional Features** A special *info* comment can be included anywhere in an ngcgui-compatible subfile. The format is:

```
(info: info_text)
```

The info_text is displayed near the top of the ngcgui tab page in axis.

Files not intended for use as subfiles can include a special comment so that ngcgui will reject them automatically with a relevant message.

```
(not_a_subfile)
```

An optional image file (.png,.gif,.jpg,.pgm) can accompany a subfile. The image file can help clarify the parameters used by the subfile. The image file should be in the same directory as the subfile and have the same name with an appropriate image suffix, e.g. the subfile example.ngc could be accompanied by an image file examp.png. Ngcgui attempts to resize large images by subsampling to a size with maximum width of 320 and maximum height of 240 pixels.

None of the conventions required for making an ngcgui-compatible subfile preclude its use as general purpose subroutine file for LinuxCNC.

The LinuxCNC distribution includes a library (ngcgui_lib directory) that includes both example ngcgui-compatible subfiles and utility files to illustrate the features of LinuxCNC subroutines and ngcgui usage. Another libary (gcmc_lib) provides examples for subroutine files for the Gcode meta compiler (gcmc)

Additional user sumitted subroutines can be found on the Forum in the Subroutines Section.

#### 2.3.6.2 Gcode-meta-compiler (.gcmc) file requirements

Files for the Gcode-meta-compiler (gcmc) are read by ngcgui and it creates entry boxes for variables tagged in the file. When a feature for the file is finalized, ngcgui passes the file as input to the gcmc compiler and, if the compile is successful, the resulting gcode file is sent to linuxCNC for execution. The resulting file is formatted as single-file subroutine; .gcmc files and .ngc files can be intermixed by ngcgui.

The variables identified for inclusion in ngcgui are tagged with lines that will appear as comments to the gcmc compiler.

**Example variable tags formats**

```
//ngcgui: varname1 =
//ngcgui: varname2 = value2
//ngcgui: varname3 = value3, label3;
```

**Examples:**

```
//ngcgui: zsafe =
//ngcgui: feedrate = 10
//ngcgui: xl = 0, x limit
```

For these examples, the entry box for varname1 will have no default, the entry box for varname2 will have a default of value2, and the entry box for varname 3 will have a default of value 3 and a label label3 (instead of varname3). The default values must be numbers.

To make it easier to modify valid lines in a gcmc file, alternate tag line formats accepted. The alternate formats ignore trailing semicolons (;) and trailing comment markers (//) With this provision, it is often makes it possible to just add the //ngcgui: tag to existing lines in a .gcmc file.

**Alternate variable tag formats**

```
//ngcgui: varname2 = value2;
//ngcgui: varname3 = value3; //, label3;
```

**Examples:**

```
//ngcgui: feedrate = 10;
//ngcgui: xl = 0; //, x limit
```

An info line that will appear at the top of a tab page may be optionally included with a line tagged as:

**Info tag**

```
//ngcgui: info: text_to_appear_at_top_of_tab_page
```

When required, options can be passed to the gcmc compiler with a line tagged:

**Option line tag format**

```
//ngcgui: -option_name [ [=] option_value]
```

**Examples:**

```
//ngcgui: -I
//ngcgui: --imperial
//ngcgui: --precision 5
//ngcgui: --precision=6
```

Options for gcmc are available with the terminal command:

```
gcmc --help
```

A gcmc program by default uses metric mode. The mode can be set to inches with the option setting:

```
//ngcgui: --imperial
```

A preamble file, if used, can set a mode (g20 or g21) that conflicts with the mode used by a gcmc file. To ensure that the gcmc program mode is in effect, include the following statement in the .gcmc file:

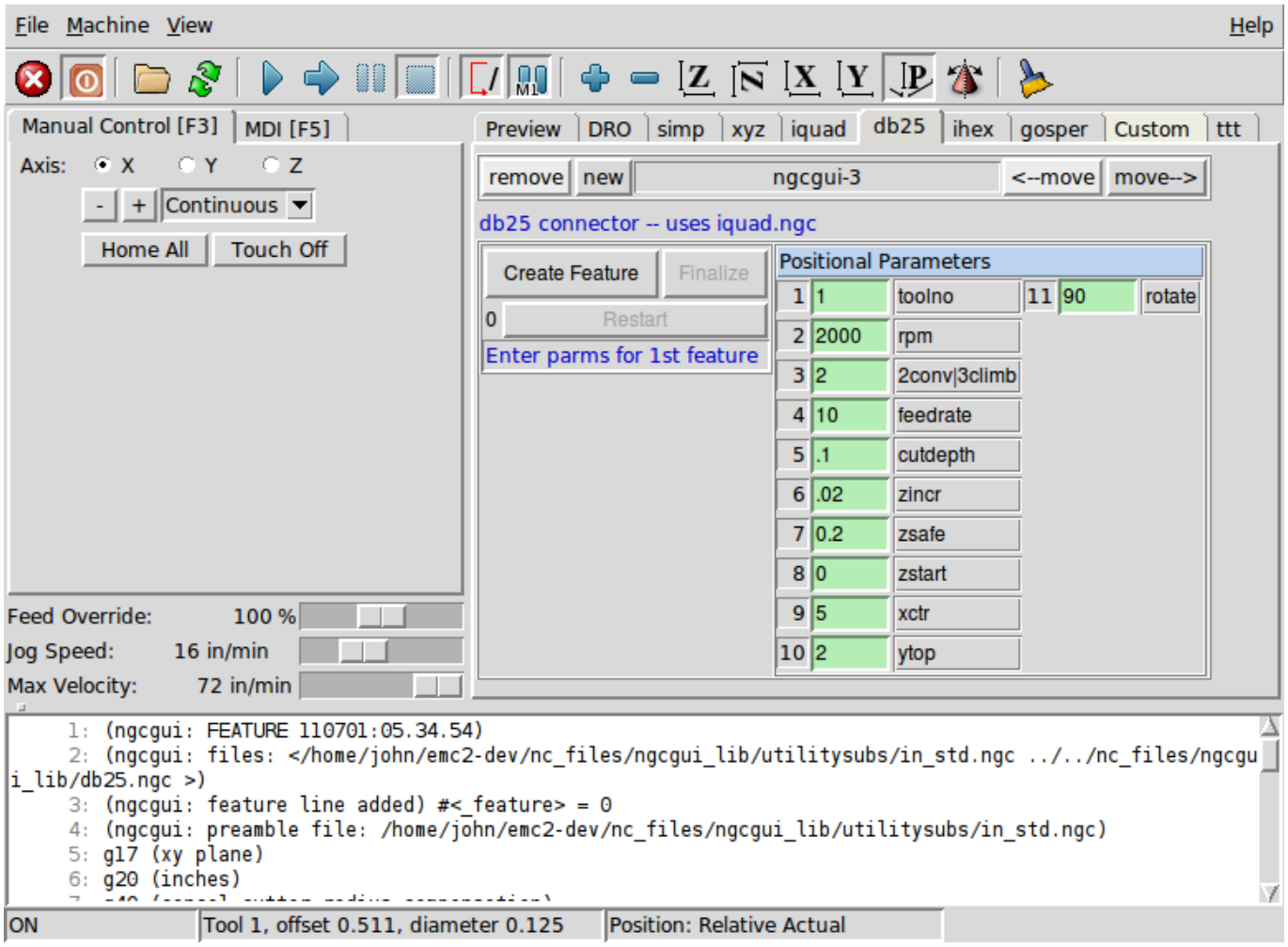
```
include("ensure_mode.gcmc")
```

and provide a proper path for gcmc include_files in the ini file, for example:

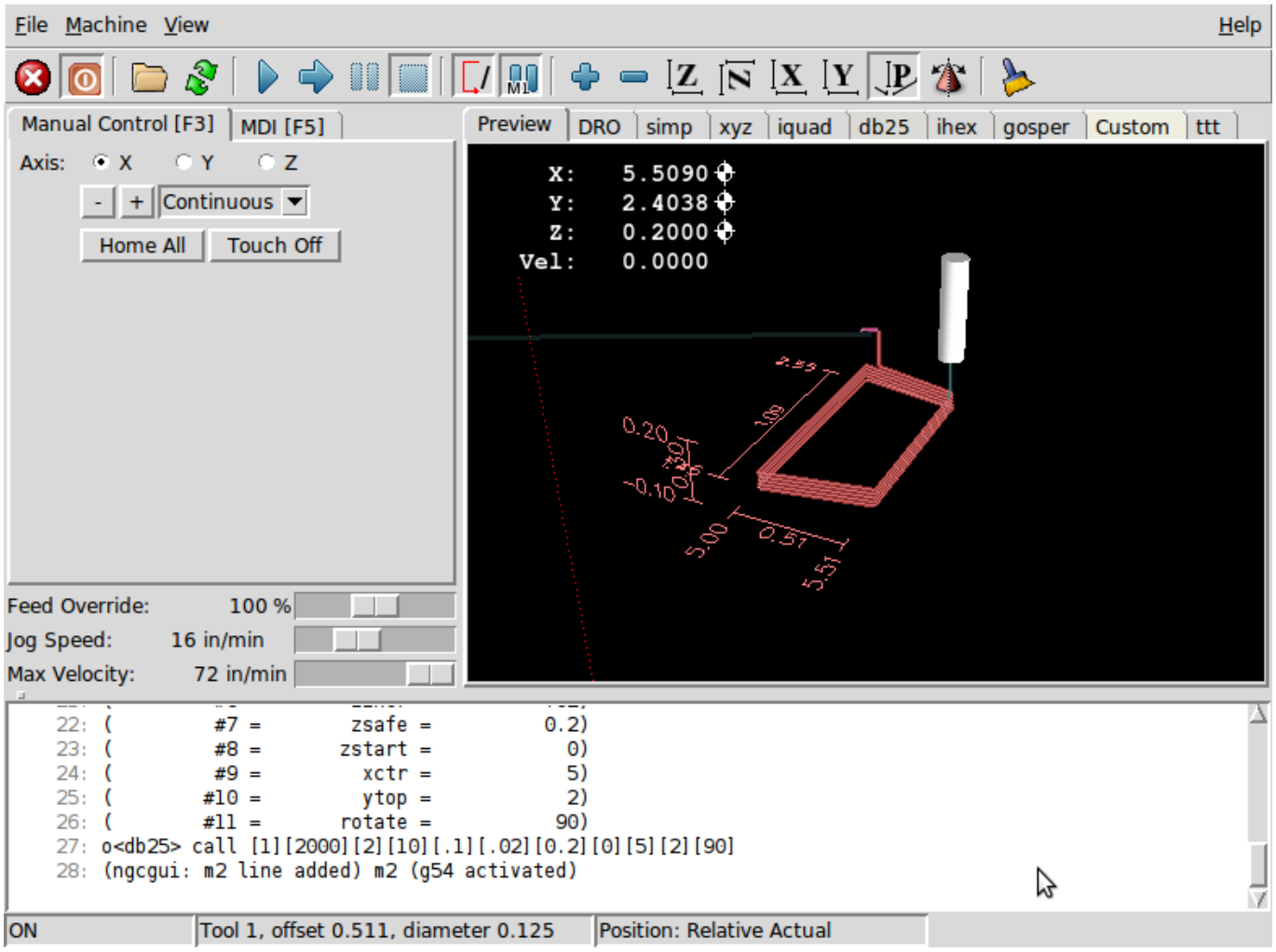
```
[DISPLAY]
GCMC_INCLUDE_PATH = ../../nc_files/gcmc_lib
```

### 2.3.7 DB25 Example

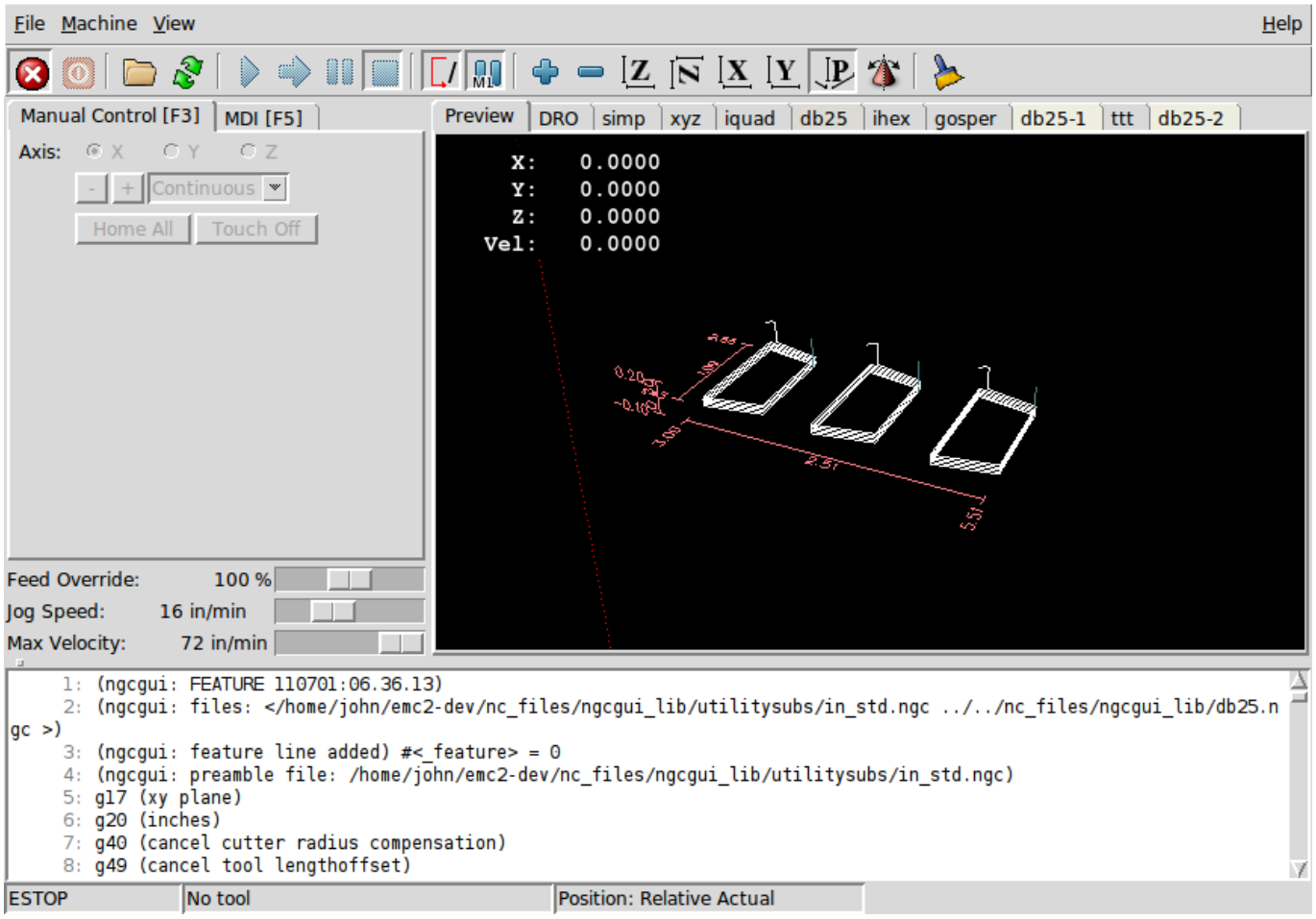
The following shows the DB25 subroutine. In the first photo you see where you fill in the blanks for each variable.




This photo shows the backplot of the DB25 subroutine.




This photo shows the use of the new button and the custom tab to create three DB25 cutouts in one program.




## 2.4 Touchy GUI

Touchy is a user interface for LinuxCNC meant for use on machine control panels, and therefore does not require keyboard or mouse.

It is meant to be used with a touch screen, and works in combination with a wheel/MPG and a few buttons and switches.

The *Handwheel* tab has radio buttons to select between *Feed Override*, *Spindle Override*, *Maximum Velocity* and *Jogging* functions for the wheel/MPG input. Radio buttons for axis selection and increment for jogging are also provided.

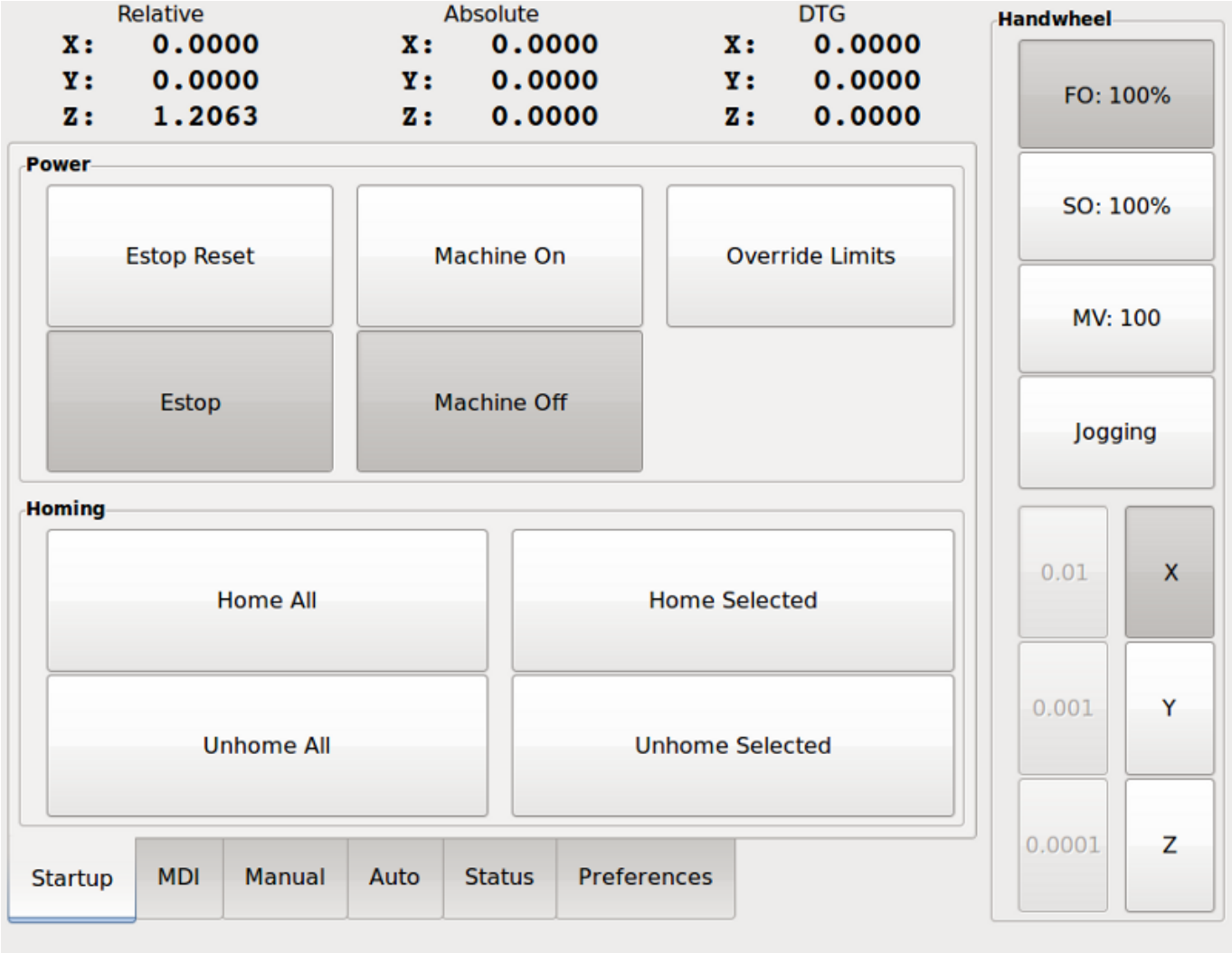



Figure 2.16: Touchy

## 2.4.1 Panel Configuration

### 2.4.1.1 HAL connections

Touchy requires that you create a file named *touchy.hal* in your configuration directory (the directory your ini file is in) to connect its controls. Touchy executes the HAL commands in this file after it has made its own pins available for connection.

For more information on HAL files and the net command see the Basic HAL Reference.

Touchy has several output pins that are meant to be connected to the motion controller to control wheel jogging:

- *touchy.jog.wheel.increment*, which is to be connected to the *axis.N.jog-scale* pin of each axis N.
- *touchy.jog.wheel.N*, which is to be connected to *axis.N.jog-enable* for each axis N.

*N* represents the axis number 0-8.

- In addition to being connected to *touchy.wheel-counts*, the wheel counts should also be connected to *axis.N.jog-counts* for each axis N. If you use HAL component *ilowpass* to smooth wheel jogging, be sure to smooth only *axis.N.jog-counts* and not *touchy.wheel-counts*.

REQUIRED CONTROLS

- Abort button (momentary contact) connected to the HAL pin *touchy.abort*
- Cycle start button (momentary contact) connected to *touchy.cycle-start*
- Wheel/MPG, connected to *touchy.wheel-counts* and motion pins as described above
- Single block (toggle switch) connected to *touchy.single-block*

OPTIONAL CONTROLS

- For continuous jog, one center-off bidirectional momentary toggle (or two momentary buttons) for each axis, hooked to *touchy.jog.continuous.x.negative*, *touchy.jog.continuous.x.positive*, etc.
- If a quill up button is wanted (to jog Z to the top of travel at top speed), a momentary button connected to *touchy.quill-up*.

OPTIONAL PANEL LAMPS

- *touchy.jog.active* shows when the panel jogging controls are live
- *touchy.status-indicator* is on when the machine is executing G-code, and flashes when the machine is executing but is in pause/feedhold.

#### 2.4.1.2 Recommended for any setup

- Estop button hardwired in the estop chain

### 2.4.2 Setup

#### 2.4.2.1 Enabling Touchy

To use Touchy, in the *[DISPLAY]* section of your ini file change the display selector line to *DISPLAY = touchy*

#### 2.4.2.2 Preferences

When you start Touchy the first time, check the Preferences tab. If using a touchscreen, choose the option to hide the pointer for best results.

The Status Window is a fixed height, set by the size of a fixed font. This can be affected by the Gnome DPI, configured in System / Preferences / Appearance / Fonts / Details. If the bottom of the screen is cut off, reduce the DPI setting.

All other font sizes can be changed on the Preferences tab.

#### 2.4.2.3 Macros

Touchy can invoke O-word macros using the MDI interface. To configure this, in the *[TOUCHY]* section of the ini file, add one or more *MACRO* lines. Each should be of the format

*MACRO=increment xinc yinc*

In this example, increment is the name of the macro, and it accepts two parameters, named xinc and yinc.

Now, place the macro in a file named *increment.ngc*, in the *PROGRAM_PREFIX* directory or any directory in the *SUBROUTINE_PATH*.

It should look like:

```
O<increment> sub
G91 G0 X#1 Y#2
G90
O<increment> endsub
```

Notice the name of the sub matches the file name and macro name exactly, including case.

When you invoke the macro by pressing the Macro button on the MDI tab in Touchy, you can enter values for xinc and yinc. These are passed to the macro as *#1* and *#2* respectively. Parameters you leave empty are passed as value 0.

If there are several different macros, press the Macro button repeatedly to cycle through them.

In this simple example, if you enter -1 for xinc and press cycle start, a rapid *G0* move will be invoked, moving one unit to the left.

This macro capability is useful for edge/hole probing and other setup tasks, as well as perhaps hole milling or other simple operations that can be done from the panel without requiring specially-written gcode programs.

## 2.5 TkLinuxCNC GUI

### 2.5.1 Introduction

TkLinuxCNC is one of the first graphical front-ends for LinuxCNC. It is written in Tcl and uses the Tk toolkit for the display. Being written in Tcl makes it very portable (it runs on a multitude of platforms). A separate backplot window can be displayed as shown.

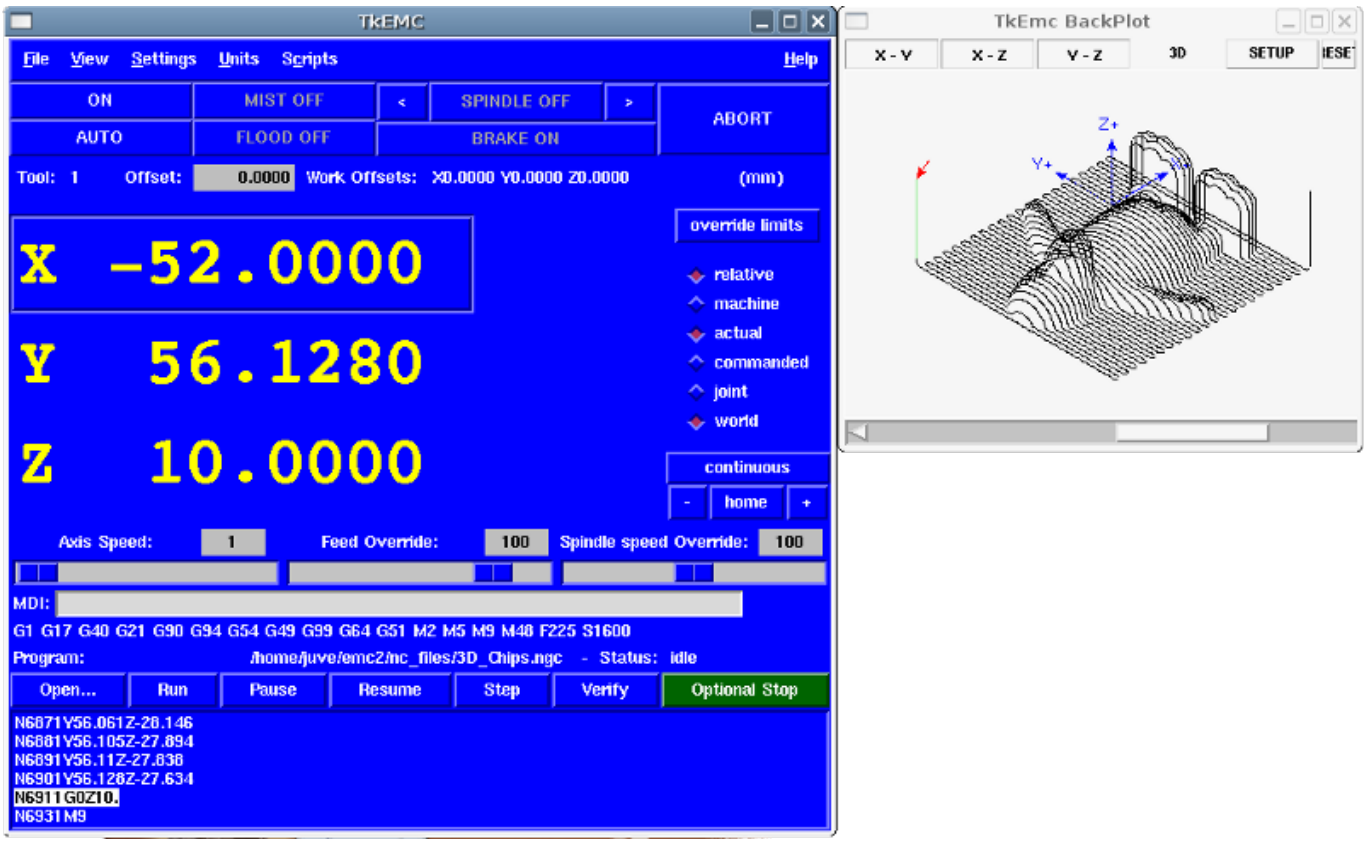



Figure 2.17: TkLinuxCNC Window

### 2.5.2 Getting Started

To select TkLinuxCNC as the front-end for LinuxCNC, edit the .ini file. In the section *[DISPLAY]* change the *DISPLAY* line to read

```
DISPLAY = tklinuxcnc
```

Then, start LinuxCNC and select that ini file. The sample configuration *sim/tklinuxcnc/tklinuxcnc.ini* is already configured to use TkLinuxCNC as its front-end.

#### 2.5.2.1 A typical session with TkLinuxCNC

1. Start LinuxCNC and select a configuration file.
2. Clear the *E-STOP* condition and turn the machine on (by pressing F1 then F2).
3. *Home* each axis.
4. Load the file to be milled.
5. Put the stock to be milled on the table.
6. Set the proper offsets for each axis by jogging and either homing again or right-clicking an axis name and entering an offset value. [1]
7. Run the program.
8. To mill the same file again, return to step 6. To mill a different file, return to step 4. When you're done, exit LinuxCNC.

### 2.5.3 Elements of the TkLinuxCNC window

The TkLinuxCNC window contains the following elements:

- A menubar that allows you to perform various actions
- A set of buttons that allow you to change the current working mode, start/stop spindle and other relevant I/O
- Status bar for various offset related displays
- Coordinate display area
- A set of sliders which control *Jogging speed*, *Feed Override* , and *Spindle speed Override* which allow you to increase or decrease those settings
- Manual data input text box *MDI*
- Status bar display with active G-codes, M-codes, F- and S-words
- Interpreter related buttons
- A text display area that shows the G-code source of the loaded file

[1] For some of these actions it might be necessary to change the mode LinuxCNC is currently running in.

#### 2.5.3.1 Main buttons

From left to right, the buttons are:

- Machine enable: *ESTOP* > *ESTOP RESET* > *ON*
- Toggle mist coolant
- Decrease spindle speed
- Set spindle direction *SPINDLE OFF* > *SPINDLE FORWARD* . *SPINDLE REVERSE*
- Increase spindle speed
- Abort

then on the second line:

- Operation mode: *MANUAL* > *MDI* > *AUTO*
- Toggle flood coolant
- Toggle spindle brake control

#### 2.5.3.2 Offset display status bar

The Offset display status bar displays the currently selected tool (selected with Txx M6), the tool length offset (if active), and the work offsets (set by right-clicking the coordinates).

#### 2.5.3.3 Coordinate Display Area

The main part of the display shows the current position of the tool. The color of the position readout depends on the state of the axis. If the axis is unhomed the axis will be displayed in yellow letters. Once homed it will be displayed in green letters. If there is an error with the current axis TkLinuxCNC will use red letter to show that. (for example if an hardware limit switch is tripped).

To properly interpret these numbers, refer to the radio boxes on the right. If the position is *Machine*, then the displayed number is in the machine coordinate system. If it is *Relative*, then the displayed number is in the offset coordinate system. Further down the choices can be *actual* or *commanded*. Actual refers to the feedback coming from encoders (if you have a servo machine), and the *commanded* refers to the position command send out to the motors. These values can differ for several reasons: Following error, deadband, encoder resolution, or step size. For instance, if you command a movement to X 0.0033 on your mill, but one step of your stepper motor is 0.00125, then the *Commanded* position will be 0.0033 but the *Actual* position will be 0.0025 (2 steps) or 0.00375 (3 steps).

Another set of radio buttons allows you to choose between *joint* and *world* view. These make little sense on a normal type of machine (e.g. trivial kinematics), but help on machines with non-trivial kinematics like robots or stewart platforms. (you can read more about kinematics in the Integrator Manual).

**Backplot** When the machine moves, it leaves a trail called the backplot. You can start the backplot window by selecting View→Backplot.

#### 2.5.3.4 Automatic control

**Buttons for control** The buttons in the lower part of TkLinuxCNC are used to control the execution of a program: *Open* to load a program, *Verify* to check it for errors, *Run* to start the actual cutting, *Pause* to stop it while running, *Resume* to resume an already paused program, *Step* to advance one line in the program and *Optional Stop* to toggle the optional stop switch (if the button is green the program execution will be stopped on any M1 encountered).

Figure 2.18: TkLinuxCNC Interpreter / program control

**Text Program Display Area** When the program is running, the line currently being executed is highlighted in white. The text display will automatically scroll to show the current line.

#### 2.5.3.5 Manual Control

**Implicit keys** TkLinuxCNC allows you to manually move the machine. This action is known as *jogging*. First, select the axis to be moved by clicking it. Then, click and hold the + or - button depending on the desired direction of motion. The first four axes can also be moved by the keyboard arrow keys (X and Y), the PAGE UP and PAGE DOWN keys (Z) and the *[* and *]* keys (A/4th).

If *Continuous* is selected, the motion will continue as long as the button or key is pressed. If another value is selected, the machine will move exactly the displayed distance each time the button is clicked or the key is pressed. The available values are: *1.0000, 0.1000, 0.0100, 0.0010, 0.0001*

By pressing *Home* or the HOME key, the selected axis will be homed. Depending on your configuration, this may just set the axis value to be the absolute position 0.0, or it may make the machine move to a specific home location through use of *home switches*. See the Homing Chapter for more information.

By pressing *Override Limits*, the machine will temporarily be permitted to jog outside the limits defined in the .ini file. (Note: if *Override Limits* is active the button will be displayed using a red color).

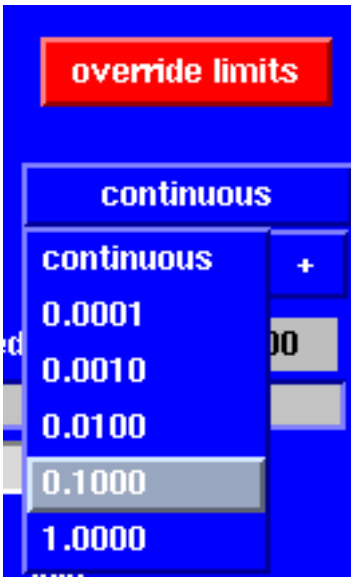



Figure 2.19: TkLinuxCNC Override Limits & Jogging increments example

**The Spindle group** The button on the first row selects the direction for the spindle to rotate: Counterclockwise, Stopped, Clockwise. The buttons next to it allow the user to increase or decrease the rotation speed. The button on the second row allows

the spindle brake to be engaged or released. Depending on your machine configuration, not all the items in this group may have an effect.

**The Coolant group** The two buttons allow the *Mist* and *Flood* coolants to be turned on and off. Depending on your machine configuration, not all the items in this group may appear.

#### 2.5.3.6 Code Entry

Manual Data Input (also called MDI), allows G-code programs to be entered manually, one line at a time. When the machine is not turned on, and not set to MDI mode, the code entry controls are unavailable.

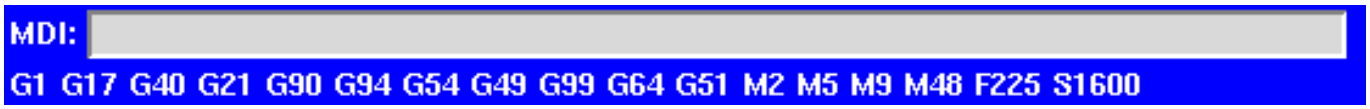



Figure 2.20: The Code Entry tab

**MDI:** This allows you to enter a g-code command to be executed. Execute the command by pressing Enter.

**Active G-Codes** This shows the *modal codes* that are active in the interpreter. For instance, *G54* indicates that the *G54 offset* is applied to all coordinates that are entered.

#### 2.5.3.7 Jog Speed

By moving this slider, the speed of jogs can be modified. The numbers above refer to axis units / second. The text box with the number is clickable. Once clicked a popup window will appear, allowing for a number to be entered.

#### 2.5.3.8 Feed Override

By moving this slider, the programmed feed rate can be modified. For instance, if a program requests *F60* and the slider is set to 120%, then the resulting feed rate will be 72. The text box with the number is clickable. Once clicked a popup window will appear, allowing for a number to be entered.

#### 2.5.3.9 Spindle speed Override

The spindle speed override slider works exactly like the feed override slider, but it controls to the spindle speed. If a program requested S500 (spindle speed 500 RPM), and the slider is set to 80%, then the resulting spindle speed will be 400 RPM. This slider has a minimum and maximum value defined in the ini file. If those are missing the slider is stuck at 100%. The text box with the number is clickable. Once clicked a popup window will appear, allowing for a number to be entered.

### 2.5.4 Keyboard Controls

Almost all actions in TkLinuxCNC can be accomplished with the keyboard. Many of the shortcuts are unavailable when in MDI mode.

The most frequently used keyboard shortcuts are shown in the following table.

Table 2.2: Most Common Keyboard Shortcuts

| Keystroke | Action Taken |
|---|---|
| F1 | Toggle Emergency Stop |
| F2 | Turn machine on/off |
| `, 1 .. 9, 0 | Set feed override from 0% to 100% |

Table 2.2: (continued)

| Keystroke | Action Taken |
|---|---|
| X, ` | Activate first axis |
| Y, 1 | Activate second axis |
| Z, 2 | Activate third axis |
| A, 3 | Activate fourth axis |
| Home | Send active axis Home |
| Left, Right | Jog first axis |
| Up, Down | Jog second axis |
| Pg Up, Pg Dn | Jog third axis |
| [, ] | Jog fourth axis |
| ESC | Stop execution |

## 2.6 MINI GUI

### 2.6.1 Introduction

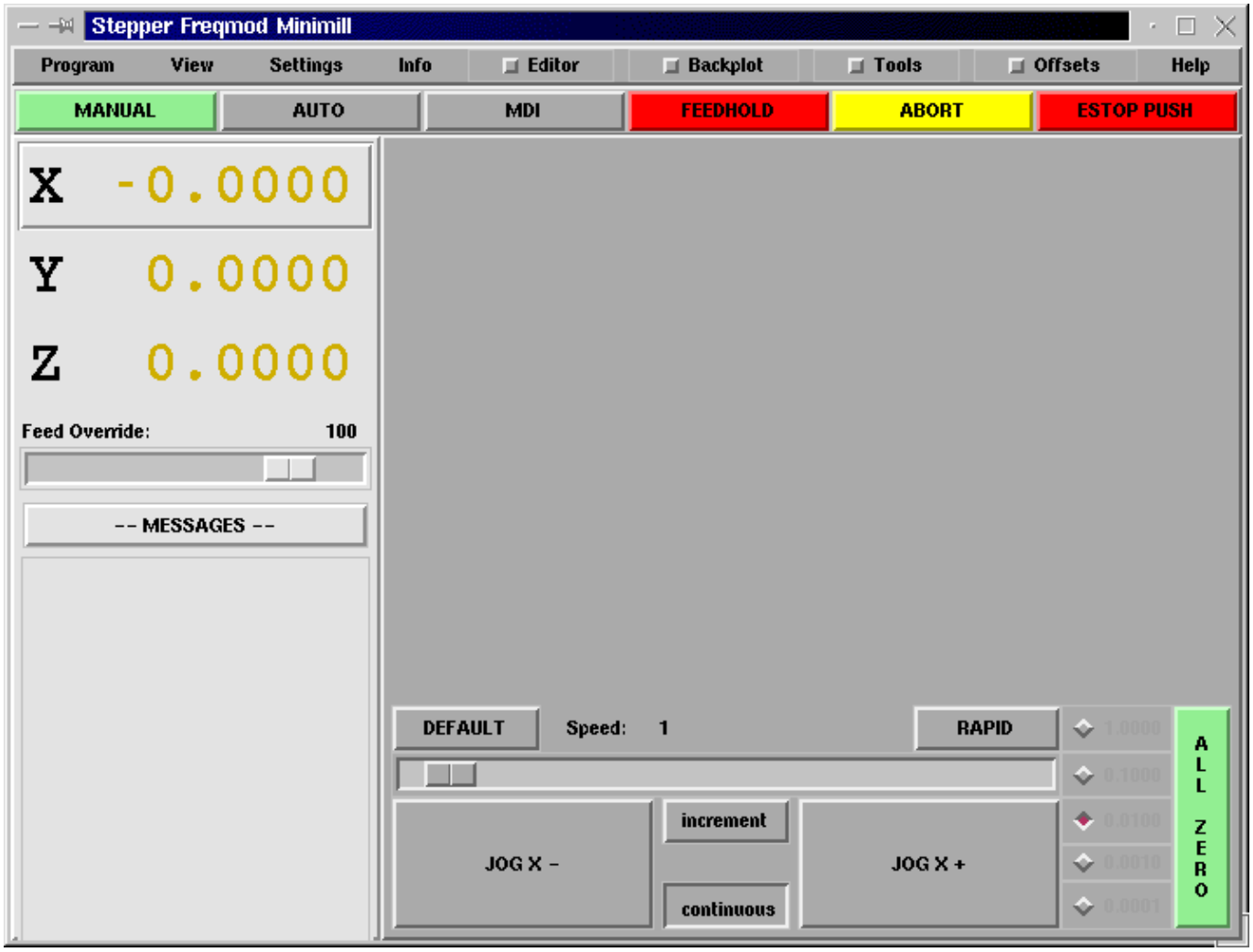



Figure 2.21: The Mini Graphical Interface

Mini was designed to be a full screen graphical interface. It was first written for the Sherline CNC but is available for anyone to use, copy, and distribute under the terms of the GPL copyright.

Rather than popup new windows for each thing that an operator might want to do, Mini allows you to display these within the regular screen. Parts of this chapter are copied from the instructions that were written for that mill by Joe Martin and Ray Henry. [2]

[2] Much of this chapter quotes from a chapter of the Sherline CNC Operators Manual.

### 2.6.2 Screen layout

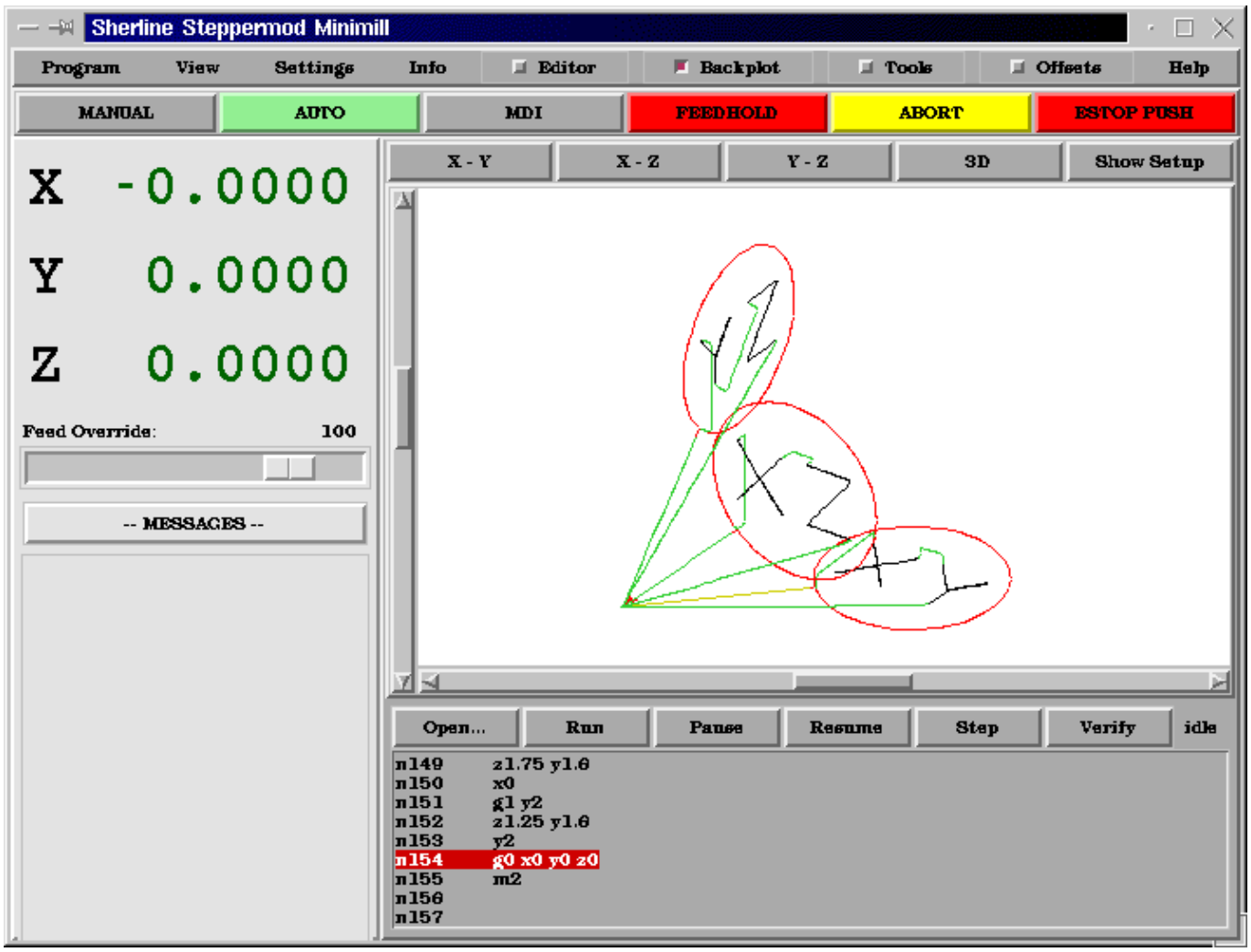



Figure 2.22: Mini Display for a Running LinuxCNC

The Mini screen is laid out in several sections. These include a menu across the top, a set of main control buttons just below the menu, and two rather large columns of information that show the state of your machine and allow you to enter commands or programs.

When you compare starting screen with run screen you will see many differences. In the second figure

- each axis has been homed — the display numbers are dark green
- the LinuxCNC mode is auto — the auto button has a light green background
- the backplotter has been turned on — backplot is visible in the pop-in window
- the tool path from the program is showing in the display.

Once you start working with Mini you will quickly discover how easily it shows the conditions of the LinuxCNC and allows you to make changes to it.

### 2.6.3 Menu Bar

The first row is the menu bar across the top. Here you can configure the screen to display additional information. Some of the items in this menu are very different from what you may be accustomed to with other programs. You should take a few minutes and look under each menu item in order to familiarize yourself with the features that are there.

The menu includes each of the following sections and subsections.

- *Program* - This menu includes both reset and exit functions. Reset will return the LinuxCNC to the condition that it was in when it started. Some startup configuration items like the normal program units can be specified in the ini file.
- *View* - This menu includes several screen elements that can be added so that you can see additional information during a run. These include
  - *Position_Type* - This menu item adds a line above the main position displays that shows whether the displays are in inches or metric and whether they are Machine or Relative location and if they are Actual positions or Commanded positions. These can be changed using the Settings menu described below.
  - *Tool_Info* - This adds a line immediately below the main position displays that shows which tool has been selected and the length of offset applied.
  - *Offset_Info* - adds a line immediately below the tool info that shows what offsets have been applied. This is a total distance for each axis from machine zero.
  - *Show_Restart* - adds a block of buttons to the right of the program display in auto mode. These allow the operator to restart a program after an abort or estop. These will pop in whenever estop or abort is pressed but can be shows by the operator anytime auto mode is active by selecting this menu item.
  - *Hide_Restart* - removes the block of buttons that control the restart of a program that has been aborted or estopped.
  - *Show_Split_Right* - changes the nature of the right hand column so that it shows both mode and pop-in information.
  - *Show_Mode_Full* - changes the right hand column so that the mode buttons or displays fill the entire right side of the screen. In manual mode, running with mode full you will see spindle and lube control buttons as well as the motion buttons.
  - *Show_Popin_Full* - changes the right hand column so that the popin fills the entire right side of the screen.
- *Settings* - These menu items allow the operator to control certain parameters during a run.
  - *Actual_Position* - sets the main position displays to actual(machine based) values.
  - *Commanded_Position* - sets the main position displays to the values that they were commanded to.
  - *Machine_Position* - sets the main position displays to the absolute distance from where the machine was homed.
  - *Relative_Position* - sets the main position displays to show the current position including any offsets like part zeros that are active. For more information on offsets see the Coordinate System Chapter.
- *Info* - lets you see a number of active things by writing their values into the MESSAGE pad.
  - *Program_File* - will write the currently active program file name.
  - *Editor_File* - will write the currently active file if the editor pop in is active and a file has been selected for editing.
  - *Parameter_File* - will write the name of the file being used for program parameters. You can find more on this in the chapters on offsets and using variables for programming.
  - *Tool_File* - will write the name of the tool file that is being used during this run.
  - *Active_G-Codes* - will write a list of all of the modal program codes that are active whenever this item is selected. For more information about modal codes see the Modal Groups section.
- *Help* - opens a text window pop in that displays the contents of the help file.

You will notice between the info menu and the help menu there are a set of four buttons. These are called check buttons because they have a small box that shows red if they have been selected. These four buttons, Editor, Backplot, Tools, and Offsets pop in each of these screens. If more than one pop-in is active (button shown as red) you can toggle between these pop-ins by right clicking your mouse.

### 2.6.4 Control Button Bar

Below the menu line is a horizontal line of control buttons. These are the primary control buttons for the interface. Using these buttons you can change mode from [MANUAL] to [AUTO] to [MDI] (Manual Data Input). These buttons show a light green background whenever that mode is active.

You can also use the [FEEDHOLD], [ABORT], and [ESTOP] buttons to control a programmed move.

#### 2.6.4.1 MANUAL

This button or pressing <F3> sets the LinuxCNC to Manual mode and displays an abbreviated set of buttons the operator can use to issue manual motion commands. The labels of the jog buttons change to match the active axis. Whenever Show_Mode_Full is active in in manual mode, you will see spindle and lube control buttons as well as the motion buttons. A keyboard <i> or <I> will switch from continuous jog to incremental jog. Pressing that key again will toggle the increment size through the available sizes.

Figure 2.23: Manual Mode Buttons

**From the Sherline CNC Operators Manual:**

A button has been added to designate the present position as the home position. We felt that a machine of this type (Sherline 5400) would be simpler to operate if it didn't use a machine home position. This button will zero out any offsets and will home all axes right where they are.

Axis focus is important here. Notice in startup figure that in manual mode you see a line or *groove* around the X axis to highlight its position display. This groove says that X is the active axis. It will be the target for jog moves made with the *plus* and *minus* jog buttons. You can change axis focus by clicking on any other axis display. You can also change axis focus in manual mode if you press its name key on your keyboard. Case is not important here. [Y] or [y] will shift the focus to the Y axis. [A] or [a] will shift the focus to the A axis. To help you remember which axis will jog when you press the jog buttons, the active axis name is displayed on them.

LinuxCNC can jog (move a particular axis) as long as you hold the button down when it is set for *continuous*, or it can jog for a preset distance when it is set for *incremental*. You can also jog the active axis by pressing the plus [+] or minus [-] keys on the keyboard. Again, case is not important for keyboard jogs. The two small buttons between the large jog buttons let you set which kind of jog you want. When you are in incremental mode, the distance buttons come alive. You can set a distance by pressing it with the mouse. You can toggle between distances by pressing [i] or [I] on the keyboard. Incremental jog has an interesting and often unexpected effect. If you press the jog button while a jog is in progress, it will add the distance to the position it was at when the second jog command was issued. Two one-inch jog presses in close succession will not get you two inches of movement. You have to wait until the first one is complete before jogging again.

Jog speed is displayed above the slider. It can be set using the slider by clicking in the slider's open slot on the side you want it to move toward, or by clicking on the [Default] or [Rapid] buttons. This setting only affects the jog move while in manual mode. Once a jog move is initiated, jog speed has no effect on the jog. As an example of this, say you set jog mode to *incremental* and the increment to 1 inch. Once you press the [Jog] button it will travel that inch at the rate at which it started.

#### 2.6.4.2 AUTO

When the Auto button is pressed, or <F4> on the keyboard, and LinuxCNC is set to that mode, a set of the traditional auto operation buttons is displayed, and a small text window opens to show a part program. During run the active line will be displayed as white lettering on a red background.

In the auto mode, many of the keyboard keys are bound to controls. For example, the numbers above the qwerty keys are bound to feed rate override. The 0 sets 100%, 9 sets 90% and such. Other keys work much the same as they do with the tkLinuxCNC graphical interface.

Figure 2.24: Auto Mode

Auto mode does not normally display the active or modal codes. If the operator wishes to check these, use menu Info→Active_G-Codes. This will write all modal codes onto the message scratch pad.

If abort or estop is pressed during a run, a set of buttons will display to the right of the text that allow the operator to shift the restart line forward or backward. If the restart line is not the last active line, it will be highlighted as white letters on a blue background. Caution, a very slow feed rate, and a finger poised over the pause button is advised during any program restart.

**From the Sherline CNC Operators Manual:**

The real heart of CNC machine tool work is the auto mode. Sherline's auto mode displays the typical functions that people have come to expect from LinuxCNC. Along the top are a set of buttons which control what is happening in auto mode. Below them is the window that shows the part of the program currently being executed. As the program runs, the active line shows in white letters on a red background. The first three buttons, [Open], [Run], and [Pause] do about what you'd expect. [Pause] will stop the run right where it is. The next button, [Resume], will restart motion. They are like feedhold if used this way. Once [Pause] is pressed and motion has stopped, [Step] will resume motion and continue it to the end of the current block. Press [Step] again to get the motion of the next block. Press [Resume] and the interpreter goes back to reading ahead and running the program. The combination of [Pause] and [Step] work a lot like single block mode on many controllers. The difference is that [Pause] does not let motion continue to the end of the current block. Feed rate Override ... can be very handy as you approach a first cut. Move in quickly at 100 percent, throttle back to 10% and toggle between [Feedhold] and 10% using the pause button. When you are satisfied that you've got it right, hit the zero to the right of nine (feedrate=100%) and go.

The [Verify] button runs the interpreter through the code without initiating any motion. If Verify finds a problem it will stop the read near the problem block and put up some sort of message. Most of the time you will be able to figure out the problem with your program by reading the message and looking in the program window at the highlighted line. Some of the messages are not very helpful. Sometimes you will need to read a line or two ahead of the highlight to see the problem. Occasionally the message will refer to something well ahead of the highlight line. This often happens if you forget to end your program with an acceptable code like %, M2, M30, or M60.

#### 2.6.4.3 MDI

The MDI button or <F5> sets the Manual Data Input mode. This mode displays a single line of text for block entry and shows the currently active modal codes for the interpreter.

**From the Sherline CNC Operators Manual:**

MDI mode allows you to enter single blocks and have the interpreter execute them as if they were part of a program (kind of like a one-line program). You can execute circles, arcs, lines and such. You can even test sets of program lines by entering one block, waiting for that motion to end, and then enter the next block. Below the entry window, there is a listing of all of the current modal codes. This listing can be very handy. I often forget to enter a g00 before I command a motion. If nothing happens I look down there to see if G80 is in effect. G80 stops any motion. If it's there I remember to issue a block like G00 X0 Y0 Z0. In MDI you are entering text from the keyboard so none of the main keys work for commands to the running machine. [F1] will Estop the control.

Since many of the keyboard keys are needed for entry, most of the bindings that were available in auto mode are not available here.

#### 2.6.4.4 [FEEDHOLD] — [CONTINUE]

Feedhold is a toggle. When the LinuxCNC is ready to handle or is handling a motion command this button shows the feedhold label on a red background. If feedhold has been pressed then it will show the continue label. Using it to pause motion has the advantage of being able to restart the program from where you stopped it. Feedhold will toggle between zero speed and whatever feed rate override was active before it was pressed. This button and the function that it activates is also bound to the pause button on most keyboards.

#### 2.6.4.5 [ABORT]

The abort button stops any motion when it is pressed. It also removes the motion command from the LinuxCNC. No further motions are cued up after this button is pressed. If you are in auto mode, this button removes the rest of the program from the

motion cue. It also records the number of the line that was executing when it was pressed. You can use this line number to restart the program after you have cleared up the reasons for pressing it.

#### 2.6.4.6 [ESTOP]

The estop button is also a toggle but it works in three possible settings.

- When Mini starts up it will show a raised button with red background with black letters that say *ESTOP PUSH*. This is the correct state of the machine when you want to run a program or jog an axis. Estop is ready to work for you when it looks like this.
- If you push the estop button while a motion is being executed, you will see a recessed gray button that says *ESTOPPED*. You will not be able to move an axis or do any work from the Mini gui when the estop button displays this way. Pressing it with your mouse will return Mini to normal ready condition.
- A third view is possible here. A recessed green button means that estop has been take off but the machine has not been turned on. Normally this only happens when <F1> estop has been pressed but <F2> has not been pressed.

Joe Martin says, "When all else fails press a software [ESTOP]." This does everything that abort does but adds in a reset so that the LinuxCNC returns to the standard settings that it wakes up on. If you have an external estop circuit that watches the relevant parallel port or DIO pin, a software estop can turn off power to the motors.

**From the Sherline CNC Operators Manual:**

Most of the time, when we abort or E-Stop it's because something went wrong. Perhaps we broke a tool and want to change it. We switch to manual mode and raise the spindle, change tools, and assuming that we got the length the same, get ready to go on. If we return the tool to the same place where the abort was issued, LinuxCNC will work perfectly. It is possible to move the restart line back or ahead of where the abort happened. If you press the [Back] or [Ahead] buttons you will see a blue highlight that shows the relationship between the abort line and the one on which LinuxCNC will start up again. By thinking through what is happening at the time of the restart you can place the tool tip where it will resume work in an acceptable manner. You will need to think through things like tool offsets, barriers to motion along a diagonal line, and such, before you press the [Restart] button.

### 2.6.5 Left Column

There are two columns below the control line. The left side of the screen displays information of interest to the operator. There are very few buttons to press here.

#### 2.6.5.1 Axis Position Displays

The axis position displays work exactly like they do with tkLinuxCNC. The color of the letters is important.

- Red indicates that the machine is sitting on a limit switch or the polarity of a min or max limit is set wrong in the ini file.
- Yellow indicates that the machine is ready to be homed.
- Green indicates that the machine has been homed.

The position can be changed to display any one of several values by using the menu settings. The startup or default settings can be changed in the ini file so these displays wake up just the way that you want them.

#### 2.6.5.2 Feed rate Override

Immediately below the axis position displays is the feed rate override slider. You can operate feed rate override and feedhold in any mode of operation. Override will change the speed of jogs or feed rate in manual or MDI modes. You can adjust feed rate override by grabbing the slider with your mouse and dragging it along the groove. You can also change feed rate a percent at a time by clicking in the slider's groove. In auto mode you can also set feed override in 10% increments by pressing the top row of numbers. This slider is a handy visual reference to how much override is being applied to programmed feed rate.

#### 2.6.5.3 Messages

The message display located under the axis positions is a sort of scratch pad for LinuxCNC. If there are problems it will report them there. If you try to home or move an axis when the [ESTOP] button is pressed, you'll get a message that says something about commanding motion when LinuxCNC is not ready. If an axis faults out for something like falling behind, the message pad will show what happened. If you want to remind an operator to change a tool, for example, you can add a line of code to your program that will display in the message box. An example might be (msg, change to tool #3 and press resume). This line of code, included in a program, will display *change to tool #3 and press resume* in the message box. The word msg, (with comma included) is the command to make this happen; without *msg,* the message wouldn't be displayed. It will still show in the auto modes' display of the program file.

To erase messages simply click the message button at the top of the pad or, on the keyboard, hold down the [Alt] key and press the [m] key.

### 2.6.6 Right Column

The right column is a general purpose place to display and work. Here you can see the modal buttons and text entry or displays. Here you can view a plot of the tool path that will be commanded by your program. You can also write programs and control tools and offsets here. The modal screens have been described above. Each of the popin displays are described in detail below.

#### 2.6.6.1 Program Editor

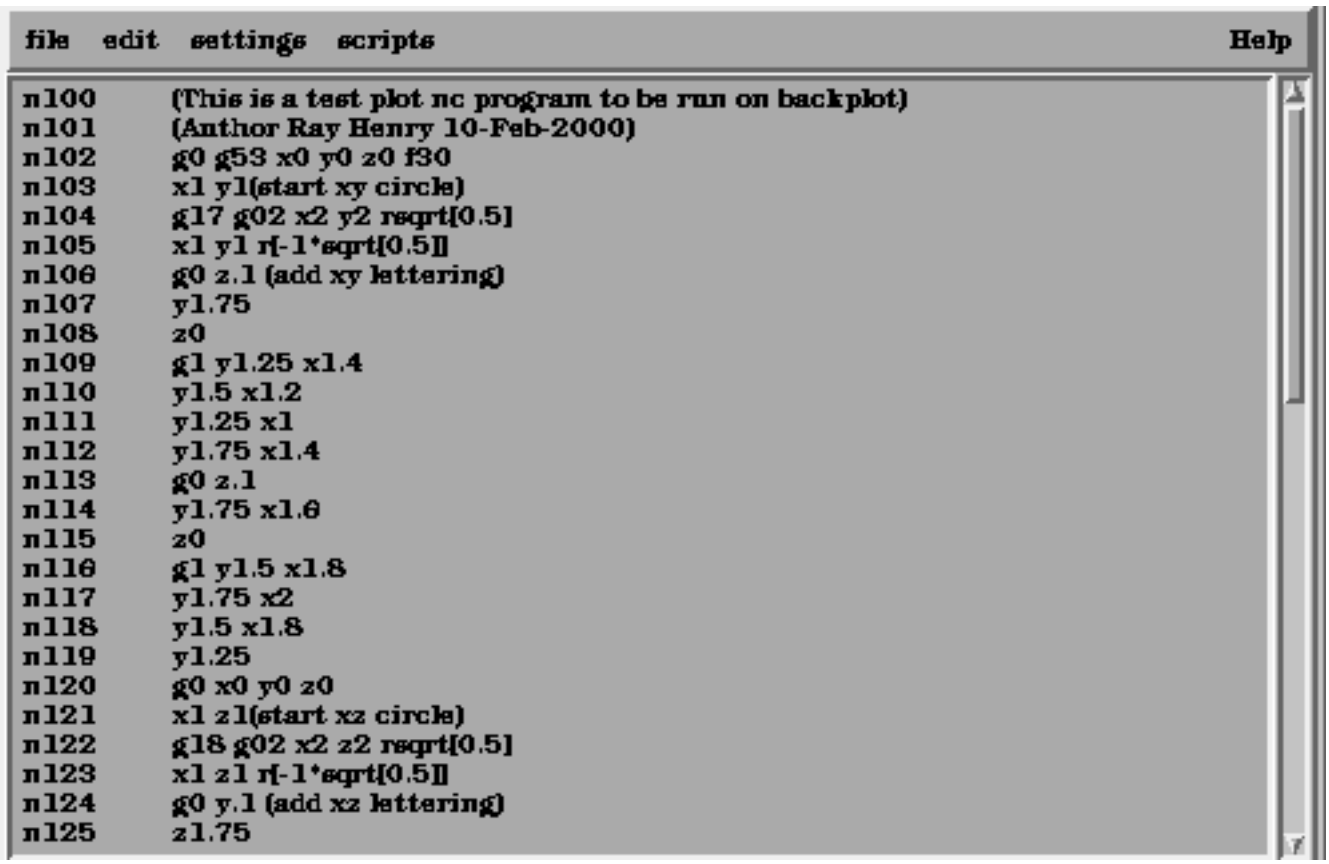



Figure 2.25: Mini Text Editor

The editor is rather limited compared to many modern text editors. It does not have *undo* nor *paste* between windows with the clipboard. These were eliminated because of interaction with a running program. Future releases will replace these functions so

that it will work the way you've come to expect from a text editor. It is included because it has the rather nice feature of being able to number and renumber lines in the way that the interpreter expects of a file. It will also allow you to cut and paste from one part of a file to another. In addition, it will allow you to save your changes and submit them to the LinuxCNC interpreter with the same menu click. You can work on a file in here for a while and then save and load if the LinuxCNC is in Auto mode. If you have been running a file and find that you need to edit it, that file will be placed in the editor when you click on the editor button on the top menu.

#### 2.6.6.2 Backplot Display

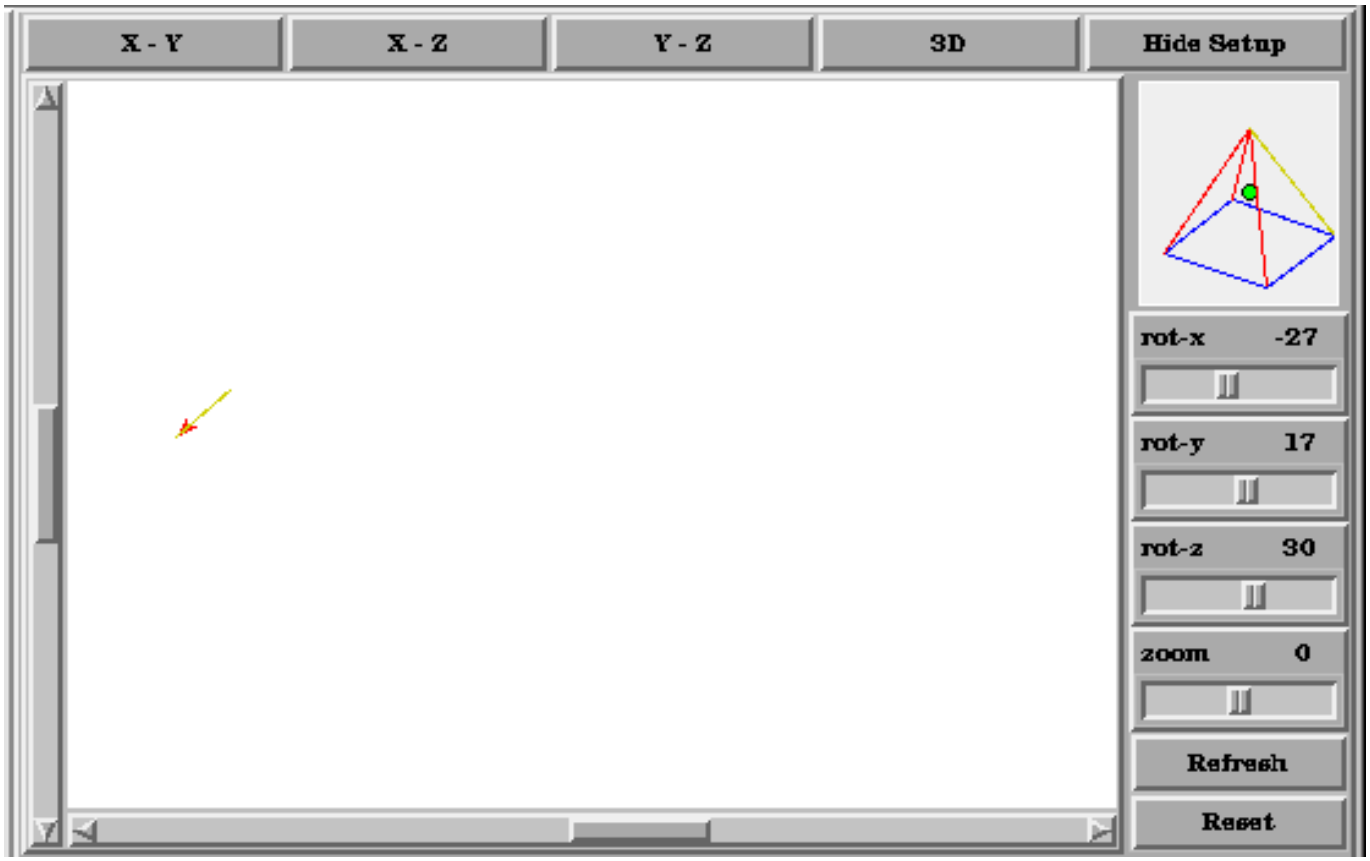



Figure 2.26: Mini Backplotter

Backplot [Backplot] will show the tool path that can be viewed from a chosen direction. *3-D* is the default. Other choices and controls are displayed along the top and right side of the pop-in. If you are in the middle of a cut when you press one of these control buttons the machine will pause long enough to re-compute the view.

Along the right side of the pop-in there is a small pyramid shaped graphic that tries to show the angle you are viewing the tool path from. Below it are a series of sliders that allow you to change the angle of view and the size of the plot. You can rotate the little position angle display with these. They take effect when you press the [Refresh] button. The [Reset] button removes all of the paths from the display and readies it for a new run of the program but retains your settings for that session.

If backplot is started before a program is started, it will try to use some color lines to indicate the kind of motion that was used to make it. A green line is a rapid move. A black line is a feed rate move. Blue and red indicate arcs in counterclockwise and clockwise directions.

The backplotter with Mini allows you to zoom and rotate views after you have run your program but it is not intended to store a tool path for a long period of time.

#### 2.6.6.3 Tool Page

The tool page is pretty much like the others. You can set length and diameter values here and they become effective when you press the [Enter] key. You will need to set up your tool information before you begin to run a program. You can't change tool offsets while the program is running or when the program is paused.

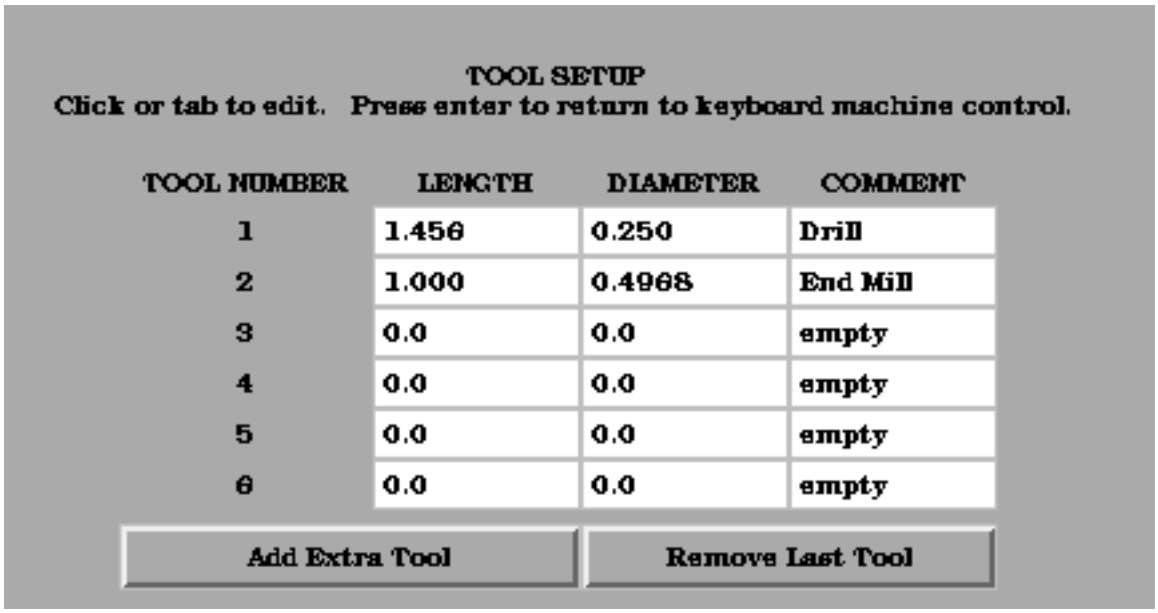



Figure 2.27: Mini Tool Display

The [Add Tools] and [Remove Tools] buttons work on the bottom of the tool list so you will want to fill in tool information in descending order. Once a new tool has been added, you can use it in a program with the usual G-code commands. There is a 32 tool limit in the current LinuxCNC configuration files but you will run out of display space in Mini long before you get there.

**Tip**
You can use Menu > View > Show Popin Full to see more tools if you need.

#### 2.6.6.4 Offset Page

The offset page can be used to display and setup work offsets. The coordinate system is selected along the left hand side of the window. Once you have selected a coordinate system you can enter values or move an axis to a teach position.

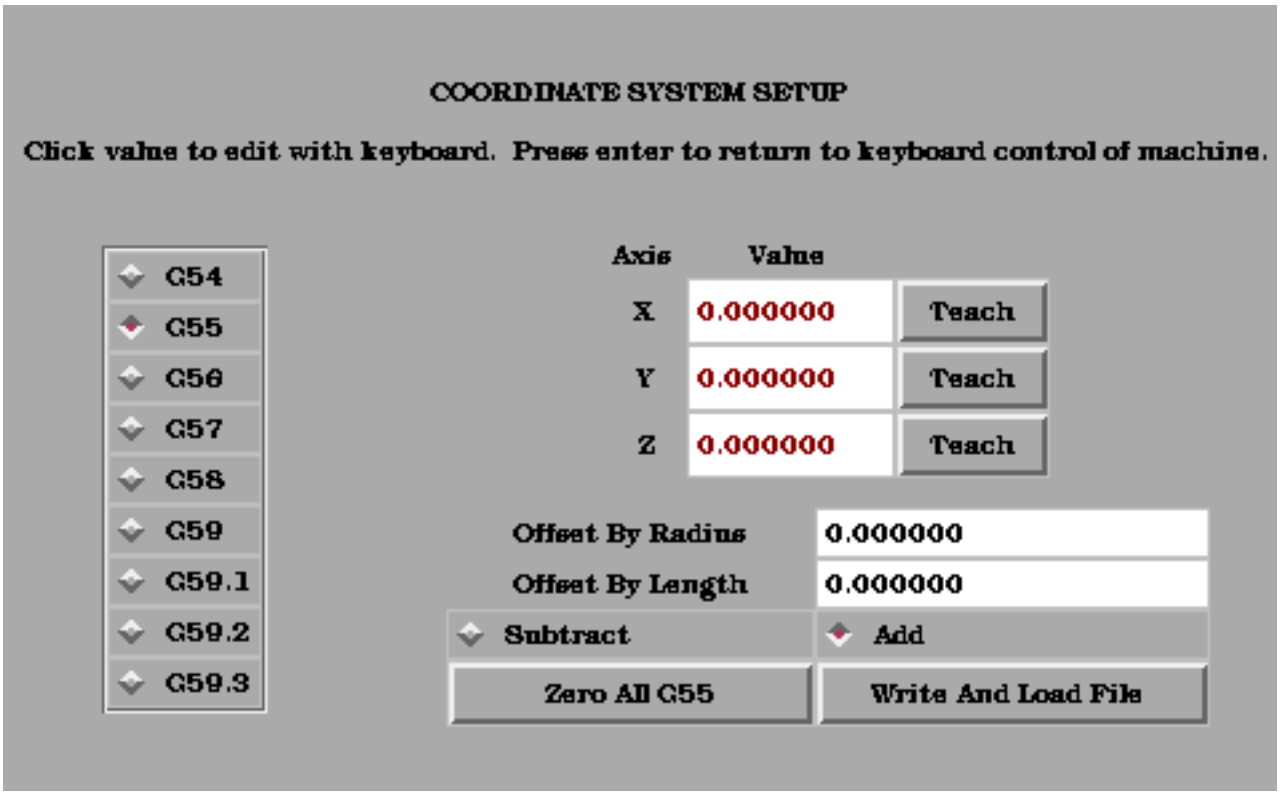



Figure 2.28: Mini Offset Display

You can also teach using an edgefinder by adding the radius and length to the offset_by widgets. When you do this you may need to add or subtract the radius depending upon which surface you choose to touch from. This is selected with the add or subtract radiobuttons below the offset windows.

The zero all for the active coordinate system button will remove any offsets that you have showing but they are not set to zero in the variable file until you press the write and load file button as well. This write and load file button is the one to use when you have set all of the axis values that you want for a coordinate system.

### 2.6.7 Keyboard Bindings

A number of the bindings used with tkLinuxCNC have been preserved with mini. A few of the bindings have been changed to extend that set or to ease the operation of a machine using this interface. Some keys operate the same regradless of the mode. Others change with the mode that LinuxCNC is operating in.

#### 2.6.7.1 Common Keys

- *Pause* - Toggle feedhold
- *Escape* - abort motion
- *F1* - toggle estop/estop reset state
- *F2* - toggle machine off/machine on state
- *F3* - manual mode
- *F4* - auto mode
- *F5* - MDI mode

- *F6* - reset interpreter

The following only work for machines using auxiliary I/O

- *F7* - toggle mist on/mist off
- *F8* - toggle flood on/flood off
- *F9* - toggle spindle forward/off
- *F10* - toggle spindle reverse/off
- *F11* - decrease spindle speed
- *F12* - increase spindle speed

#### 2.6.7.2 Manual Mode

- *1-9 0* - set feed override to 10%-90%, 0 is 100%
- ~ - set feed override to 0 or feedhold
- *x* - select X axis
- *y* - select Y axis
- *z* - select Z axis
- *a* - select A axis
- *b* - select B axis
- *c* - select C axis
- *Left Right Arrow* - jog X axis
- *Up Down Arrow* - jog Y axis
- *Page Up Down* - jog Z axis
- - _ - jog the active axis in the minus direction
- + = - jog the active axis in the plus direction.
- *Home* - home selected axis
- *i I* - toggle through jog increments

The following only work with a machine using auxiliary I/O

- *b* - take spindle brake off
- *Alt-b* - put spindle brake on

#### 2.6.7.3 Auto Mode

- *1-9,0* - set feed override to 10%-90%, 0 is 100%
- ~ - set feed override to 0 or feedhold
- *o/O* - open a program
- *r/R* - run an opened program
- *p/P* - pause an executing program
- *s/S* - resume a paused program
- *a/A* - step one line in a paused program

### 2.6.8 Misc

One of the features of Mini is that it displays any axis above number 2 as a rotary and will display degree units for it. It also converts to degree units for incremental jogs when a rotary axis has the focus.

## 2.7 KEYSTICK GUI

### 2.7.1 Introduction

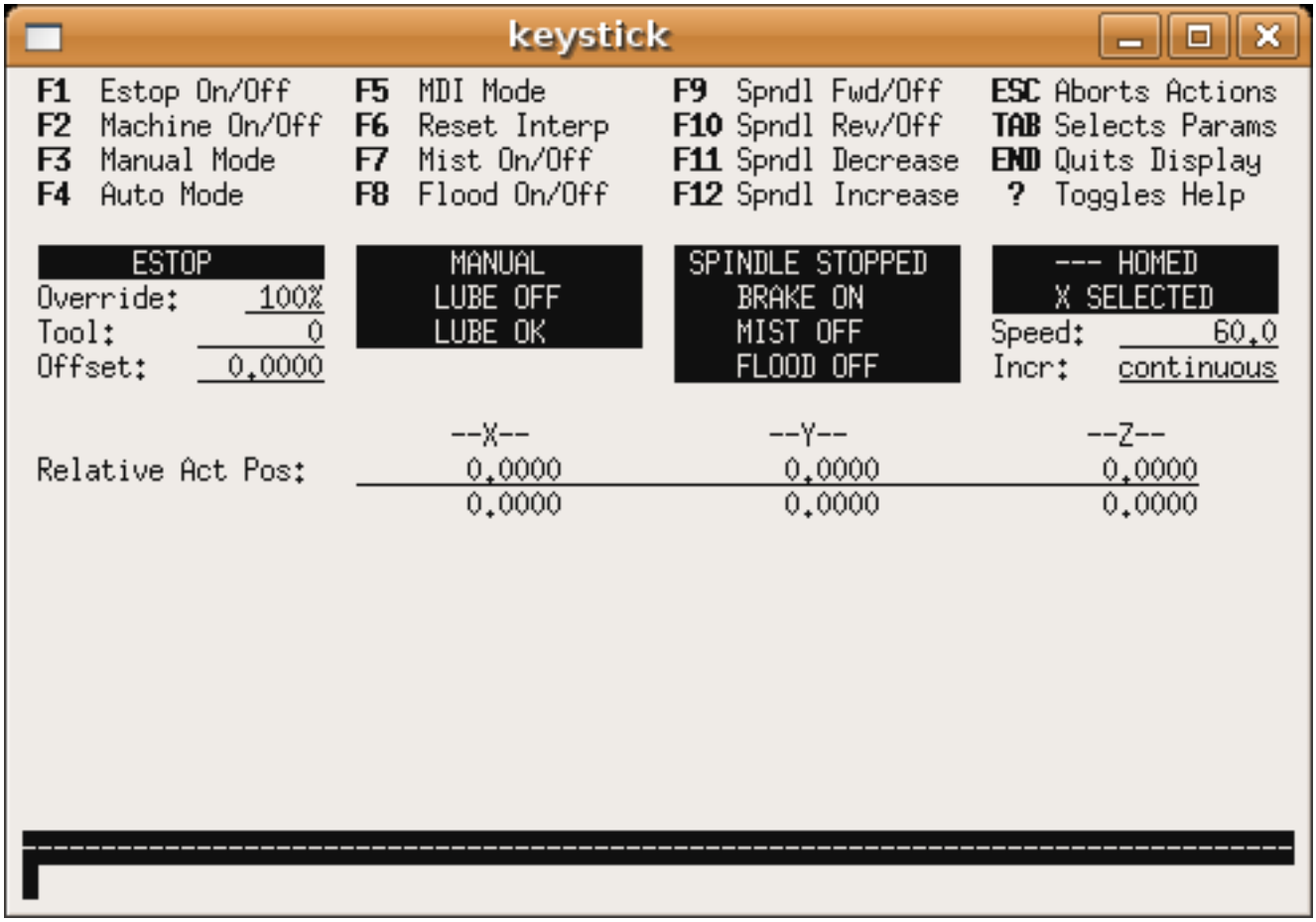



Figure 2.29: The Mini Graphical Interface

Keystick is a minimal text based interface.

### 2.7.2 Installing

To use keystick change the *DISPLAY* setting in the ini file setting to:

```
DISPLAY = keystick
```

### 2.7.3 Using

Keystick is very simple to use. In the MDI Mode you simply start typing the g code and it shows up in the bottom text area. The *?* key toggles help.

# Chapter 3

# Programming

## 3.1 Coordinate Systems

### 3.1.1 Introduction

This chapter introduces you to offsets as they are used by the LinuxCNC. These include:

- Machine Coordinates (G53)
- Nine Coordinate System Offsets (G54-G59.3)
- Global Offsets (G92)

### 3.1.2 Machine Coordinate System

When LinuxCNC is started the postions of each axis is the machine origin. Once an axis homed the the machine orgin for that axis is set to the homed position. The machine origin is the machine coordinate system which all other coordinate systems are based. The G53 G code can be used to move in the machine coordinate system.

### 3.1.3 Coordinate Systems

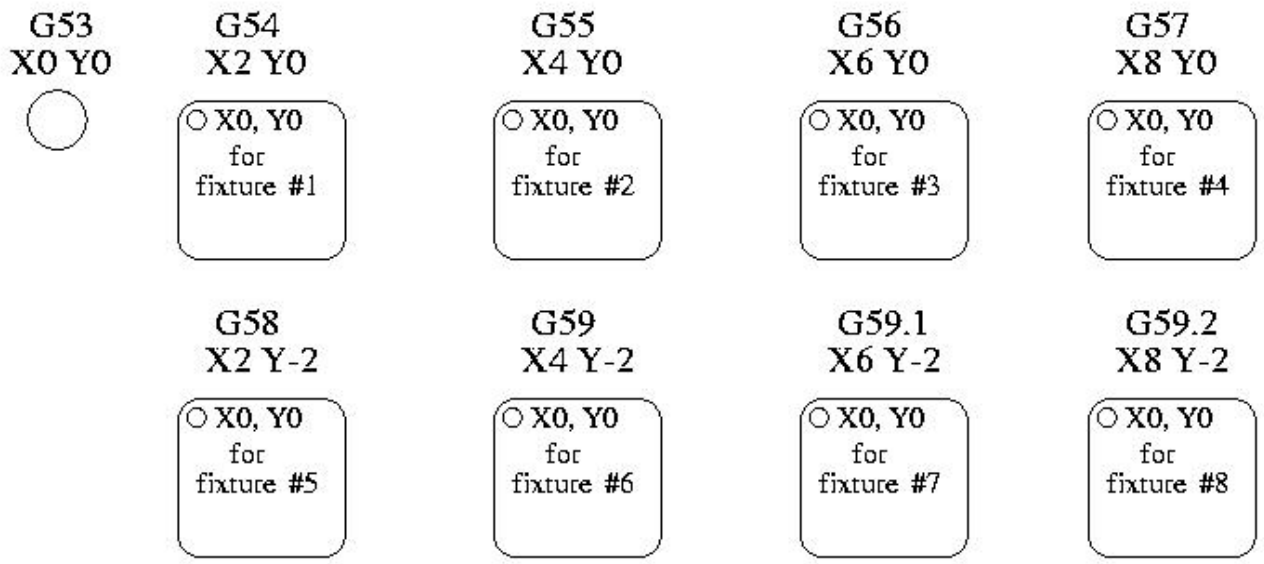



Figure 3.1: Example of Coordinate Systems

COORDINATE SYSTEM OFFSETS

- G54 - use coordinate system 1
- G55 - use coordinate system 2
- G56 - use coordinate system 3
- G57 - use coordinate system 4
- G58 - use coordinate system 5
- G59 - use coordinate system 6
- G59.1 - use coordinate system 7
- G59.2 - use coordinate system 8
- G59.3 - use coordinate system 9

Coordinate system offsets are used to shift the coordinate system from the machine coordinate system. This allows the G code to be programmed for the part without regard to the part location on the machine. Using coordinate system offsets would allow you to machine parts in multiple locations with the same G code.

The values for offsets are stored in the VAR file that is requested by the INI file during the startup of an LinuxCNC.

In the VAR file scheme, the first variable number stores the X offset, the second the Y offset and so on for all nine axes. There are numbered sets like this for each of the coordinate system offsets.

Each of the graphical interfaces has a way to set values for these offsets. You can also set these values by editing the VAR file itself and then restart LinuxCNC so that the LinuxCNC reads the new values however this is not the recommended way. Using G10, G92, G28.1, etc are better ways to set the variables.

Table 3.1: Example of G55 parameters

| Axis | Variable | Value |
|---|---|---|
| X | 5241 | 2.000000 |
| Y | 5242 | 1.000000 |
| Z | 5243 | -2.000000 |
| A | 5244 | 0.000000 |
| B | 5245 | 0.000000 |
| C | 5246 | 0.000000 |
| U | 5247 | 0.000000 |
| V | 5248 | 0.000000 |
| W | 5249 | 0.000000 |

You should read this as moving the zero positions of G55 to X = 2 units, Y= 1 unit, and Z = -2 units away from the absolute zero position.

Once there are values assigned, a call to G55 in a program block would shift the zero reference by the values stored. The following line would then move each axis to the new zero position. Unlike G53, G54 through G59.3 are modal commands. They will act on all blocks of code after one of them has been set. The program that might be run using fixture offsets would require only a single coordinate reference for each of the locations and all of the work to be done there. The following code is offered as an example of making a square using the G55 offsets that we set above.

```
G55 ; use coordinate system 2
G0 X0 Y0 Z0
G1 F2 Z-0.2000
X1
Y1
X0
Y0
G0 Z0
G54 ; use coordinate system 1
G0 X0 Y0 Z0
M2
```

In this example the G54 near the end leaves the G54 coordinate system with all zero offsets so that there is a modal code for the absolute machine based axis positions. This program assumes that we have done that and use the ending command as a command to machine zero. It would have been possible to use G53 and arrive at the same place but that command would not have been modal and any commands issued after it would have returned to using the G55 offsets because that coordinate system would still be in effect.

#### 3.1.3.1 Default Coordinate System

One other variable in the VAR file becomes important when we think about offset systems. This variable is named 5220. In the default files its value is set to 1.00000. This means that when the LinuxCNC starts up it should use the first coordinate system as its default. If you set this to 9.00000 it would use the ninth offset system as its default for start up and reset. Any value other than an integer (decimal really) between 1 and 9, or a missing 5220 variable will cause the LinuxCNC to revert to the default value of 1.00000 on start up.

#### 3.1.3.2 Setting Coordinate System Offsets

The G10 L2x command can be used to set coordinate system offsets:

- *G10 L2 P(1-9)* - Set offset(s) to a value. Current position irrelevant. (see G10 L2 for details)
- *G10 L20 P(1-9)* - Set offset(s) so current position becomes a value. (see G10 L20 for details)

### 3.1.4 Global Offsets

#### 3.1.4.1 The G92 Commands

This set of commands include;

- *G92* - This command, when used with axis names, sets values to offset variables.
- *G92.1* - This command sets zero values to the G92 variables.
- *G92.2* - This command suspends but does not zero out the G92 variables.
- *G92.3* - This command applies offset values that have been suspended.

When the commands are used as described above, they will work pretty much as you would expect.

To make the current point, what ever it is, have the coordinates X0, Y0, and Z0 you would use G92 X0 Y0 Z0. G92 **does not** work from absolute machine coordinates. It works from **current location**.

G92 also works from current location as modified by any other offsets that are in effect when the G92 command is invoked. While testing for differences between work offsets and actual offsets it was found that a G54 offset could cancel out a G92 and thus give the appearance that no offsets were in effect. However, the G92 was still in effect for all coordinates and did produce expected work offsets for the other coordinate systems.

It is a good practice to clear the G92 offsets at the end of their use with G92.1 or G92.2. When starting up LinuxCNC if any offsets are in the G92 variables they will be applied when an axis is homed.

#### 3.1.4.2 Setting G92 Values

G92 commands work from current axis location and add and subtract correctly to give the current axis position the value assigned by the G92 command. The effects work even though previous offsets are in.

So if the X axis is currently showing 2.0000 as its position a *G92 X0* will set an offset of -2.0000 so that the current location of X becomes zero. A *G92 X2* will set an offset of 0.0000 and the displayed position will not change. A *G92 X5.0000* will set an offset of 3.0000 so that the current displayed position becomes 5.0000.

The G92 parameters are:

- 5211 - X Axis Offset
- 5212 - Y Axis Offset
- 5213 - Z Axis Offset
- 5214 - A Axis Offset
- 5215 - B Axis Offset
- 5216 - C Axis Offset
- 5217 - U Axis Offset
- 5218 - V Axis Offset
- 5219 - W Axis Offset

If you are seeing unexpected positions as the result of a commanded move, as a result of storing an offset in a previous program and not clearing them at the end then issue a G92.1 in the MDI window to clear the stored offsets.

If G92 values exist in the VAR file when LinuxCNC starts up, the G92 values in the var file will be applied to the values of the current location of each axis. If this is home position and home position is set as machine zero everything will be correct. Once home has been established using real machine switches, or by moving each axis to a known home position and issuing an axis home command, any G92 offsets will be applied. If you have a G92 X1 in effect when you home the X axis the DRO will read

*X: 1.000* instead of the expected *X: 0.000* because the G92 was applied to the machine origin. If you issue a G92.1 and the DRO now reads all zeros then you had a G92 offset in effect when you last ran LinuxCNC.

Unless your intention is to use the same G92 offsets in the next program, the best practice is to issue a G92.1 at the end of any G Code files where you use G92 offsets.

**Caution**
When a file is aborted during processing that has G92 offsets in effect a startup will cause them to become active again. Always have your preamble to set the environment as you expect it.

### 3.1.5 Sample Program Using Offsets

This sample engraving project mills a set of four .1 radius circles in roughly a star shape around a center circle. We can setup the individual circle pattern like this.

```
G10 L2 P1 X0 Y0 Z0 (ensure that G54 is set to machine zero)
G0 X-0.1 Y0 Z0
G1 F1 Z-0.25
G3 X-0.1 Y0 I0.1 J0
G0 Z0
M2
```

We can issue a set of commands to create offsets for the four other circles like this.

```
G10 L2 P2 X0.5 (offsets G55 X value by 0.5 inch)
G10 L2 P3 X-0.5 (offsets G56 X value by -0.5 inch)
G10 L2 P4 Y0.5 (offsets G57 Y value by 0.5 inch)
G10 L2 P5 Y-0.5 (offsets G58 Y value by -0.5 inch)
```

We put these together in the following program:

```
(a program for milling five small circles in a diamond shape)

G10 L2 P1 X0 Y0 Z0 (ensure that G54 is machine zero)
G10 L2 P2 X0.5 (offsets G55 X value by 0.5 inch)
G10 L2 P3 X-0.5 (offsets G56 X value by -0.5 inch)
G10 L2 P4 Y0.5 (offsets G57 Y value by 0.5 inch)
G10 L2 P5 Y-0.5 (offsets G58 Y value by -0.5 inch)

G54 G0 X-0.1 Y0 Z0 (center circle)
G1 F1 Z-0.25
G3 X-0.1 Y0 I0.1 J0
G0 Z0

G55 G0 X-0.1 Y0 Z0 (first offset circle)
G1 F1 Z-0.25
G3 X-0.1 Y0 I0.1 J0
G0 Z0

G56 G0 X-0.1 Y0 Z0 (second offset circle)
G1 F1 Z-0.25
G3 X-0.1 Y0 I0.1 J0
G0 Z0

G57 G0 X-0.1 Y0 Z0 (third offset circle)
G1 F1 Z-0.25
G3 X-0.1 Y0 I0.1 J0
G0 Z0
```

```
G58 G0 X-0.1 Y0 Z0 (fourth offset circle)
G1 F1 Z-0.25
G3 X-0.1 Y0 I0.1 J0
G54 G0 X0 Y0 Z0

M2
```

Now comes the time when we might apply a set of G92 offsets to this program. You'll see that it is running in each case at Z0. If the mill were at the zero position, a G92 Z1.0000 issued at the head of the program would shift everything an inch. You might also shift the whole pattern around in the XY plane by adding some X and Y offsets with G92. If you do this you should add a G92.1 command just before the M2 that ends the program. If you do not, other programs that you might run after this one will also use that G92 offset. Furthermore it would save the G92 values when you shut down the LinuxCNC and they will be recalled when you start up again.

## 3.2 G Code Overview

### 3.2.1 Overview

The LinuxCNC G Code language is based on the RS274/NGC language. The G Code language is based on lines of code. Each line (also called a *block*) may include commands to do several different things. Lines of code may be collected in a file to make a program.

A typical line of code consists of an optional line number at the beginning followed by one or more *words*. A word consists of a letter followed by a number (or something that evaluates to a number). A word may either give a command or provide an argument to a command. For example, *G1 X3* is a valid line of code with two words. *G1* is a command meaning *move in a straight line at the programmed feed rate to the programmed end point*, and *X3* provides an argument value (the value of X should be 3 at the end of the move). Most LinuxCNC G Code commands start with either G or M (for General and Miscellaneous). The words for these commands are called *G codes* and *M codes.*

The LinuxCNC language has no indicator for the start of a program. The Interpreter, however, deals with files. A single program may be in a single file, or a program may be spread across several files. A file may demarcated with percents in the following way. The first non-blank line of a file may contain nothing but a percent sign, %, possibly surrounded by white space, and later in the file (normally at the end of the file) there may be a similar line. Demarcating a file with percents is optional if the file has an *M2* or *M30* in it, but is required if not. An error will be signaled if a file has a percent line at the beginning but not at the end. The useful contents of a file demarcated by percents stop after the second percent line. Anything after that is ignored.

The LinuxCNC G Code language has two commands (*M2* or *M30*), either of which ends a program. A program may end before the end of a file. Lines of a file that occur after the end of a program are not to be executed. The interpreter does not even read them.

### 3.2.2 Format of a line

A permissible line of input code consists of the following, in order, with the restriction that there is a maximum (currently 256) to the number of characters allowed on a line.

1. an optional block delete character, which is a slash /.
2. an optional line number.
3. any number of words, parameter settings, and comments.
4. an end of line marker (carriage return or line feed or both).

Any input not explicitly allowed is illegal and will cause the Interpreter to signal an error.

Spaces and tabs are allowed anywhere on a line of code and do not change the meaning of the line, except inside comments. This makes some strange-looking input legal. The line *G0X +0. 12 34Y 7* is equivalent to *G0 x+0.1234 Y7*, for example.

Blank lines are allowed in the input. They are to be ignored.

Input is case insensitive, except in comments, i.e., any letter outside a comment may be in upper or lower case without changing the meaning of a line.

#### 3.2.2.1 Block Delete

The optional block delete character the slash / when placed first on a line can be used by some user interfaces to skip lines of code when needed. In Axis the key combination Alt-m-/ toggles block delete on and off. When block delete is on any lines starting with the slash / are skipped.

#### 3.2.2.2 Line Number

A line number is the letter N followed by an unsigned integer, optionally followed by a period and another unsigned integer. For example, *N1234* and *N56.78* are valid line numbers. They may be repeated or used out of order, although normal practice is to avoid such usage. Line numbers may also be skipped, and that is normal practice. A line number is not required to be used, but must be in the proper place if used.

#### 3.2.2.3 Word

A word is a letter other than N followed by a real value.

Words may begin with any of the letters shown in the following Table. The table includes N for completeness, even though, as defined above, line numbers are not words. Several letters (I, J, K, L, P, R) may have different meanings in different contexts. Letters which refer to axis names are not valid on a machine which does not have the corresponding axis.

Table 3.2: Words and their meanings

| Letter | Meaning |
|---|---|
| A | A axis of machine |
| B | B axis of machine |
| C | C axis of machine |
| D | Tool radius compensation number |
| F | Feed rate |
| G | General function (See table Modal Groups) |
| H | Tool length offset index |
| I | X offset for arcs and G87 canned cycles |
| J | Y offset for arcs and G87 canned cycles |
| K | Z offset for arcs and G87 canned cycles. |
| | Spindle-Motion Ratio for G33 synchronized movements. |
| L | generic parameter word for G10, M66 and others |
| M | Miscellaneous function (See table Modal Groups) |
| N | Line number |
| P | Dwell time in canned cycles and with G4. |
| | Key used with G10. |
| Q | Feed increment in G73, G83 canned cycles |
| R | Arc radius or canned cycle plane |
| S | Spindle speed |
| T | Tool selection |
| U | U axis of machine |
| V | V axis of machine |
| W | W axis of machine |
| X | X axis of machine |
| Y | Y axis of machine |
| Z | Z axis of machine |

#### 3.2.2.4 Number

The following rules are used for (explicit) numbers. In these rules a digit is a single character between 0 and 9.

- A number consists of (1) an optional plus or minus sign, followed by (2) zero to many digits, followed, possibly, by (3) one decimal point, followed by (4) zero to many digits - provided that there is at least one digit somewhere in the number.
- There are two kinds of numbers: integers and decimals. An integer does not have a decimal point in it; a decimal does.
- Numbers may have any number of digits, subject to the limitation on line length. Only about seventeen significant figures will be retained, however (enough for all known applications).
- A non-zero number with no sign as the first character is assumed to be positive.

Notice that initial (before the decimal point and the first non-zero digit) and trailing (after the decimal point and the last non-zero digit) zeros are allowed but not required. A number written with initial or trailing zeros will have the same value when it is read as if the extra zeros were not there.

Numbers used for specific purposes in RS274/NGC are often restricted to some finite set of values or some to some range of values. In many uses, decimal numbers must be close to integers; this includes the values of indexes (for parameters and carousel slot numbers, for example), M codes, and G codes multiplied by ten. A decimal number which is supposed be close to an integer is considered close enough if it is within 0.0001 of an integer.

### 3.2.3 Parameters

The RS274/NGC language supports *parameters* - what in other programming languages would be called *variables*. There are several types of parameter of different purpose and appearance, each described in the following sections. The only value type supported by parameters is floating-point; there are no string, boolean or integer types in G-code like in other programming languages. However, logic expressions can be formulated with boolean operators ( *AND*, *OR*, *XOR*, and the comparison operators *EQ*,*NE*,*GT*,*GE*,*LT*,*LE*), and the *MOD*, *ROUND*, *FUP* and *FIX* operators support integer arithmetic.

Parameters differ in syntax, scope, behavior when not yet initialized, mode, persistence and intended use.

**Syntax**

There are three kinds of syntactic appearance:

- *numbered* - #4711
- *named local* - #<localvalue>
- *named global* - #<_globalvalue>

**Scope**

The scope of a parameter is either global, or local within a subroutine. Subroutine parameters and local named variables have local scope. Global named parameters and numbered parameters starting from number 31 are global in scope. RS274/NGC uses *lexical scoping* - in a subroutine only the local variables defined therein, and any global variables are visible. The local variables of a calling procedure are not visible in a called procedure.

**Behavior of uninitialized parameters**

- Unitialized global parameters, and unused subroutine parameters return the value zero when used in an expression.
- Unitialized named parameters signal an error when used in an expression.

**Mode**

Most parameters are read/write and may be assigned to within an assignment statement. However, for many predefined parameters this does not make sense, so they are are read-only - they may appear in expressions, but not on the left-hand side of an assignment statement.

**Persistence**

When LinuxCNC is shut down, volatile parameters lose their values. All parameters except numbered parameters in the current persistent range [1] are volatile. Persistent parameters are saved in the .var file and restored to their previous values when LinuxCNC is started again. Volatile numbered parameters are reset to zero.

**Intended Use**

- user parameters:: numbered parameters in the range 31..5000, and named global and local parameters except predefined parameters. These are available for general-purpose storage of floating-point values, like intermediate results, flags etc, throughout program execution. They are read/write (can be assigned a value).
- subroutine parameters - these are used to hold the actual parameters passed to a subroutine.
- numbered parameters - most of these are used to access offsets of coordinate systems.
- system parameters - used to determine the current running version. They are read-only.

#### 3.2.3.1 Numbered Parameters

A numbered parameter is the pound character # followed by an integer between 1 and (currently) 5602 [2]. The parameter is referred to by this integer, and its value is whatever number is stored in the parameter.

A value is stored in a parameter with the = operator; for example:

```
#3 = 15 (set parameter 3 to 15)
```

A parameter setting does not take effect until after all parameter values on the same line have been found. For example, if parameter 3 has been previously set to 15 and the line *#3=6 G1 X#3* is interpreted, a straight move to a point where X equals 15 will occur and the value of parameter 3 will be 6.

The # character takes precedence over other operations, so that, for example, *#1+2* means the number found by adding 2 to the value of parameter 1, not the value found in parameter 3. Of course, *#[1+2]* does mean the value found in parameter 3. The # character may be repeated; for example *##2* means the value of the parameter whose index is the (integer) value of parameter 2.

- *31-5000* - G code user parameters. These parameters are global in the G code file, and available for general use. Volatile.
- *5061-5069* - Coordinates of a "G38.2" Probe result of X, Y, Z, A, B, C, U, V & W. Volatile.
- *5070* - "G38" probe result - 1 if success, 0 if probe failed to close. Used with G38.3 and G38.5. Volatile.
- *5161-5169* - "G28" Home for X, Y, Z, A, B, C, U, V & W. Persistent.
- *5181-5189* - "G30" Home for X, Y, Z, A, B, C, U, V & W. Persistent.
- *5211-5219* - "G92" offset for X, Y, Z, A, B, C, U, V & W. Persistent.
- *5210* - 1 if "G92" offset is currently applied, 0 otherwise. Persistent.
- *5211-5219* - *G92* offset (X Y Z A B C U V W).
- *5220* - Coordinate System number 1 - 9 for G54 - G59.3. Persistent.
- *5221-5230* - Coordinate System 1, G54 for X, Y, Z, A, B, C, U, V, W & R. R denotes the XY rotation angle around the Z axis. Persistent.
- *5241-5250* - Coordinate System 2, G55 for X, Y, Z, A, B, C, U, V, W & R. Persistent.
- *5261-5270* - Coordinate System 3, G56 for X, Y, Z, A, B, C, U, V, W & R. Persistent.
- *5281-5290* - Coordinate System 4, G57 for X, Y, Z, A, B, C, U, V, W & R. Persistent.

[1] The range of persistent parameters may change as development progresses. This range is currently 5161- 5390. It is defined in the *_required_parameters array* in file the src/emc/rs274ngc/interp_array.cc .

[2] The RS274/NGC interpreter maintains an array of numbered parameters. Its size is defined by the symbol *RS274NGC_MAX_PARAMETERS* in the file src/emc/rs274ngc/interp_internal.hh). This number of numerical parameters may also increase as development adds support for new parameters.

- *5301-5310* - Coordinate System 5, G58 for X, Y, Z, A, B, C, U, V, W & R. Persistent.
- *5321-5330* - Coordinate System 6, G59 for X, Y, Z, A, B, C, U, V, W & R. Persistent.
- *5341-5350* - Coordinate System 7, G59.1 for X, Y, Z, A, B, C, U, V, W & R. Persistent.
- *5361-5370* - Coordinate System 8, G59.2 for X, Y, Z, A, B, C, U, V, W & R. Persistent.
- *5381-5390* - Coordinate System 9, G59.3 for X, Y, Z, A, B, C, U, V, W & R. Persistent.
- *5399* - Result of M66 - Check or wait for input. Volatile.
- *5400* - Tool Number. Volatile.
- *5401-5409* - Tool Offsets for X, Y, Z, A, B, C, U, V & W. Volatile.
- *5410* - Tool Diameter. Volatile.
- *5411* - Tool Front Angle. Volatile.
- *5412* - Tool Back Angle. Volatile.
- *5413* - Tool Orientation. Volatile.
- *5420-5428* - Current relative position in the active coordinate system including all offsets and in the current program units for X, Y, Z, A, B, C, U, V & W, volatile.
- *5599* - Flag for controlling the output of (DEBUG,) statements. 1=output, 0=no output; default=1. Volatile.
- *5600* - Toolchanger fault indicator. Used with the iocontrol-v2 component. 1: toolchanger faulted, 0: normal. Volatile.
- *5601* - Toolchanger fault code. Used with the iocontrol-v2 component. Reflects the value of the *toolchanger-reason* HAL pin if a fault occurred. Volatile.

**Numbered Parameters Persistence** The values of parameters in the persistent range are retained over time, even if the machining center is powered down. LinuxCNC uses a parameter file to ensure persistence. It is managed by the Interpreter. The Interpreter reads the file when it starts up, and writes the file when it exits.

The format of a parameter file is shown in Table Parameter File Format.

The Interpreter expects the file to have two columns. It skips any lines which do not contain exactly two numeric values. The first column is expected to contain an integer value (the parameter's number). The second column contains a floating point number (this parameter's last value). The value is represented as a double-precision floating point number inside the Interpreter, but a decimal point is not required in the file.

Parameters in the user-defined range (31-5000) may be added to this file. Such parameters will be read by the Interpreter and written to the file as it exits.

Missing Parameters in the persistent range will be initialized to zero and written with their current values on the next save operation.

The parameter numbers must be arranged in ascending order. An *Parameter file out of order* error will be signaled if they are not in ascending order.

The original file is saved as a backup file when the new file is written.

Table 3.3: Parameter File Format

| Parameter Number | Parameter Value |
|---|---|
| 5161 | 0.0 |
| 5162 | 0.0 |

#### 3.2.3.2 Subroutine Parameters

- *1-30* Subroutine local parameters of call arguments. These parameters are local to the subroutine. Volatile. See also the chapter on O-Codes.

#### 3.2.3.3 Named Parameters

Named parameters work like numbered parameters but are easier to read. All parameter names are converted to lower case and have spaces and tabs removed, so *<param>* and *<P a R am >* refer to the same parameter. Named parameters must be enclosed with < > marks.

*#<named parameter>* is a local named parameter. By default, a named parameter is local to the scope in which it is assigned. You can't access a local parameter outside of its subroutine. This means that two subroutines can use the same parameter names without fear of one subroutine overwriting the values in another.

*#<_global named parameter>* is a global named parameter. They are accessible from within called subroutines and may set values within subroutines that are accessible to the caller. As far as scope is concerned, they act just like regular numeric parameters. They are not stored in files.

Examples:

**Declaration of named global variable**

```
#<_endmill_dia> = 0.049
```

**Reference to previously declared global variable**

```
#<_endmill_rad> = [#<_endmill_dia>/2.0]
```

**Mixed literal and named parameters**

```
o100 call [0.0] [0.0] [#<_inside_cutout>-#<_endmill_dia>] [#<_Zcut>] [#<_feedrate>]
```

Named parameters spring into existence when they are assigned a value for the first time. Local named parameters vanish when their scope is left: when a subroutine returns, all its local parameters are deleted and cannot be referred to anymore.

It is an error to use a non-existent named parameter within an expression, or at the right-hand side of an assignment. Printing the value of a non-existent named parameter with a DEBUG statement - like *(DEBUG, <no_such_parameter>)* will display the string *#*.

Global parameters, as well as local parameters assigned to at the global level, retain their value once assigned even when the program ends, and have these values when the program is run again.

The EXISTS function tests whether a given named parameter exists.

#### 3.2.3.4 Predefined Named Parameters

The following global read only named parameters are available to access internal state of the interpreter and machine state. They can be used in arbitrary expressions, for instance to control flow of the program with if-then-else statements. Note that new predefined named parameters can be added easily without changes to the source code.

- *#<_vmajor>* - Major package version. If current version was 2.5.2 would return 2.5.
- *#<_vminor>* - Minor package version. If current version was 2.6.2 it would return 0.2.
- *#<_line>* - Sequence number. If running a G-Code file, this returns the current line number.
- *#<_motion_mode>* - Return the interpreter's current motion mode:

| Motion mode | return value |
|---|---|
| G1 | 10 |
| G2 | 20 |
| G3 | 30 |
| G33 | 330 |
| G38.2 | 382 |
| G38.3 | 383 |
| G38.4 | 384 |
| G38.5 | 385 |
| G5.2 | 52 |
| G73 | 730 |
| G76 | 760 |
| G80 | 800 |
| G81 | 810 |
| G82 | 820 |
| G83 | 830 |
| G84 | 840 |
| G85 | 850 |
| G86 | 860 |
| G87 | 870 |
| G88 | 880 |
| G89 | 890 |

- *#<_plane>* - returns the value designating the current plane:

| Plane | return value |
|---|---|
| G17 | 170 |
| G18 | 180 |
| G19 | 190 |
| G17.1 | 171 |
| G18.1 | 181 |
| G19.1 | 191 |

- *#<_ccomp>* - Status of cutter compensation. Return values:

| Mode | return value |
|---|---|
| G40 | 400 |
| G41 | 410 |
| G41.1 | 411 |
| G41 | 410 |
| G42 | 420 |
| G42.1 | 421 |

- *#<_metric>* - Return 1 if G21 is on, else 0.
- *#<_imperial>* - Return 1 if G20 is on, else 0.
- *#<_absolute>* - Return 1 if G90 is on, else 0.
- *#<_incremental>* - Return 1 if G91 is on, else 0.
- *#<_inverse_time>* - Return 1 if inverse feed mode (G93) is on, else 0.

- *#<_units_per_minute>* - Return 1 if Units/minute feed mode (G94) is on, else 0.
- *#<_units_per_rev>* - Return 1 if Units/revolution mode (G95) is on, else 0.
- *#<_coord_system>* - Return a float of the current coordinate system name(G54..G59.3). For example if your in G55 coordinate system the return value is 550.000000 and if your in G59.1 the return value is 591.000000.

| Mode | return value |
|---|---|
| G54 | 0 |
| G55 | 1 |
| G56 | 2 |
| G57 | 3 |
| G58 | 4 |
| G59 | 5 |
| G59.1 | 6 |
| G59.2 | 7 |
| G59.3 | 8 |

- *#<_tool_offset>* - Return 1 if tool offset (G43) is on, else 0.
- *#<_retract_r_plane>* - Return 1 if G98 is set, else 0.
- *#<_retract_old_z>* - Return 1 if G99 is on, else 0.

#### 3.2.3.5 System Parameters

- *#<_spindle_rpm_mode>* - Return 1 if spindle rpm mode (G97) is on, else 0.
- *#<_spindle_css_mode>* - Return 1 if constant surface speed mode (G96) is on, else 0.
- *#<_ijk_absolute_mode>* - Return 1 if Absolute Arc distance mode (G90.1) is on, else 0.
- *#<_lathe_diameter_mode>* - Return 1 if this is a lathe configuration and diameter (G7) mode is on, else 0.
- *#<_lathe_radius_mode>* - Return 1 if this is a lathe configuration and radius (G8) mode is on, else 0.
- *#<_spindle_on>* - Return 1 if spindle currently running (M3 or M4) else 0.
- *#<_spindle_cw>* - Return 1 if spindle direction is clockwise (M3) else 0.
- *#<_mist>* - Return 1 if mist (M7) is on.
- *#<_flood>* - Return 1 if flood (M8) is on.
- *#<_speed_override>* - Return 1 if feed override (M48 or M50 P1) is on, else 0.
- *#<_feed_override>* - Return 1 if feed override (M48 or M51 P1) is on, else 0.
- *#<_adaptive_feed>* - Return 1 if adaptive feed (M52 or M52 P1) is on, else 0.
- *#<_feed_hold>* - Return 1 if feed hold switch is enabled (M53 P1), else 0.
- *#<_feed>* - Return the current value of F, not the actual feed rate.
- *#<_rpm>* - Return the current value of S, not the actual spindle speed.
- *#<_x>* - Return current relative X coordinate including all offsets. Same as #5420.
- *#<_y>* - Return current relative Y coordinate including all offsets. Same as #5421.
- *#<_z>* - Return current relative Z coordinate including all offsets. Same as #5422.

- *#<_a>* - Return current relative A coordinate including all offsets. Same as #5423.
- *#<_b>* - Return current relative B coordinate including all offsets. Same as #5424.
- *#<_c>* - Return current relative C coordinate including all offsets. Same as #5425.
- *#<_u>* - Return current relative U coordinate including all offsets. Same as #5426.
- *#<_v>* - Return current relative V coordinate including all offsets. Same as #5427.
- *#<_w>* - Return current relative W coordinate including all offsets. Same as #5428.
- *#<_current_tool>* - Return number of the current tool in spindle. Same as #5400.
- *#<_current_pocket>* - Return pocket number of the current tool.
- *#<_selected_tool>* - Return number of the selected tool post a T code. Default -1.
- *#<_selected_pocket>* - Return number of the selected pocket post a T code. Default -1 (no pocket selected).
- *#<_value>* - Return value from the last O-word *return* or *endsub*. Default value 0 if no expression after *return* or *endsub*. Initialized to 0 on program start.
- *#<_value_returned>* - 1.0 if the last O-word *return* or *endsub* returned a value, 0 otherwise. Cleared by the next O-word call.
- *#<_task>* - 1.0 if the executing interpreter instance is part of milltask, 0.0 otherwise. Sometimes it is necessary to treat this case specially to retain proper preview, for instance when testing the success of a probe (G38.x) by inspecting #5070, which will always fail in the preview interpreter (e.g. Axis).
- *#<_call_level>* - current nesting level of O-word procedures. For debugging.
- *#<_remap_level>* - current level of the remap stack. Each remap in a block adds one to the remap level. For debugging.

### 3.2.4 Expressions

An expression is a set of characters starting with a left bracket *[* and ending with a balancing right bracket *]* . In between the brackets are numbers, parameter values, mathematical operations, and other expressions. An expression is evaluated to produce a number. The expressions on a line are evaluated when the line is read, before anything on the line is executed. An example of an expression is *[1 + acos[0] - [#3 ** [4.0/2]]]*.

### 3.2.5 Binary Operators

Binary operators only appear inside expressions. There are four basic mathematical operations: addition (+), subtraction (-), multiplication (*), and division (/). There are three logical operations: non-exclusive or (*OR*), exclusive or (*XOR*), and logical and (*AND*). The eighth operation is the modulus operation (*MOD*). The ninth operation is the *power* operation (**) of raising the number on the left of the operation to the power on the right. The relational operators are equality (*EQ*), inequality (*NE*), strictly greater than (*GT*), greater than or equal to (*GE*), strictly less than (*LT*), and less than or equal to (*LE*).

The binary operations are divided into several groups according to their precedence. If operations in different precedence groups are strung together (for example in the expression *[2.0 / 3 * 1.5 - 5.5 / 11.0]*), operations in a higher group are to be performed before operations in a lower group. If an expression contains more than one operation from the same group (such as the first / and * in the example), the operation on the left is performed first. Thus, the example is equivalent to: *[ [ [2.0 / 3] * 1.5] - [5.5 / 11.0] ]* , which is equivalent to to *[1.0 - 0.5]* , which is *0.5*.

The logical operations and modulus are to be performed on any real numbers, not just on integers. The number zero is equivalent to logical false, and any non-zero number is equivalent to logical true.

Table 3.4: (continued)

| Operators | Precedence |
|---|---|

Table 3.4: Operator Precedence

| Operators | Precedence |
|---|---|
| ** | *highest* |
| * / MOD | |
| + - | |
| EQ NE GT GE LT LE | |
| AND OR XOR | *lowest* |

### 3.2.6 Equality and floating-point values

The RS274/NGC language only supports floating-point values of finite precision. Therefore, testing for equality or inequality of two floating-point values is inherently problematic. The interpreter solves this problem by considering values equal if their absolute difference is less than 0.0001 (this value is defined as *TOLERANCE_EQUAL* in src/emc/rs274ngc/interp_internal.hh).

### 3.2.7 Functions

The available functions are shown in following table. Arguments to unary operations which take angle measures (*COS*, *SIN*, and *TAN* ) are in degrees. Values returned by unary operations which return angle measures (*ACOS*, *ASIN*, and *ATAN*) are also in degrees.

Table 3.5: Functions

| Function Name | Function result |
|---|---|
| ATAN[arg]/[arg] | Four quadrant inverse tangent |
| ABS[arg] | Absolute value |
| ACOS[arg] | Inverse cosine |
| ASIN[arg] | Inverse sine |
| COS[arg] | Cosine |
| EXP[arg] | e raised to the given power |
| FIX[arg] | Round down to integer |
| FUP[arg] | Round up to integer |
| ROUND[arg] | Round to nearest integer |
| LN[arg] | Base-e logarithm |
| SIN[arg] | Sine |
| SQRT[arg] | Square Root |
| TAN[arg] | Tangent |
| EXISTS[arg] | Check named Parameter |

The *FIX* function rounds towards the left (less positive or more negative) on a number line, so that *FIX[2.8] =2* and *FIX[-2.8] = -3*.

The *FUP* operation rounds towards the right (more positive or less negative) on a number line; *FUP[2.8] = 3* and *FUP[-2.8] = -2*.

The *EXISTS* function checks for the existence of a single named parameter. It takes only one named parameter and returns 1 if it exists and 0 if it does not exist. It is an error if you use a numbered parameter or an expression. Here is an example for the usage

of the EXISTS function:

```
o<test> sub
o10 if [EXISTS[#<_global>]]
    (debug, _global exists and has the value #<_global>)
o10 else
    (debug, _global does not exist)
o10 endif
o<test> endsub

o<test> call
#<_global> = 4711
o<test> call
m2
```

### 3.2.8 Repeated Items

A line may have any number of G words, but two G words from the same modal group may not appear on the same line See the Modal Groups Section for more information.

A line may have zero to four M words. Two M words from the same modal group may not appear on the same line.

For all other legal letters, a line may have only one word beginning with that letter.

If a parameter setting of the same parameter is repeated on a line, *#3=15 #3=6*, for example, only the last setting will take effect. It is silly, but not illegal, to set the same parameter twice on the same line.

If more than one comment appears on a line, only the last one will be used; each of the other comments will be read and its format will be checked, but it will be ignored thereafter. It is expected that putting more than one comment on a line will be very rare.

### 3.2.9 Item order

The three types of item whose order may vary on a line (as given at the beginning of this section) are word, parameter setting, and comment. Imagine that these three types of item are divided into three groups by type.

The first group (the words) may be reordered in any way without changing the meaning of the line.

If the second group (the parameter settings) is reordered, there will be no change in the meaning of the line unless the same parameter is set more than once. In this case, only the last setting of the parameter will take effect. For example, after the line *#3=15 #3=6* has been interpreted, the value of parameter 3 will be 6. If the order is reversed to *#3=6 #3=15* and the line is interpreted, the value of parameter 3 will be 15.

If the third group (the comments) contains more than one comment and is reordered, only the last comment will be used.

If each group is kept in order or reordered without changing the meaning of the line, then the three groups may be interleaved in any way without changing the meaning of the line. For example, the line *g40 g1 #3=15 (foo) #4=-7.0* has five items and means exactly the same thing in any of the 120 possible orders (such as *#4=-7.0 g1 #3=15 g40 (foo)*) for the five items.

### 3.2.10 Commands and Machine Modes

Many commands cause the controller to change from one mode to another, and the mode stays active until some other command changes it implicitly or explicitly. Such commands are called *modal*. For example, if coolant is turned on, it stays on until it is explicitly turned off. The G codes for motion are also modal. If a G1 (straight move) command is given on one line, for example, it will be executed again on the next line if one or more axis words is available on the line, unless an explicit command is given on that next line using the axis words or canceling motion.

*Non-modal* codes have effect only on the lines on which they occur. For example, G4 (dwell) is non-modal.

### 3.2.11 Polar Coordinates

Polar Coordinates can be used to specify the XY coordinate of a move. The @n is the distance and ˆn is the angle. The advantage of this is for things like bolt hole circles which can be done very simply by moving to a point in the center of the circle, setting the offset and then moving out to the first hole then run the drill cycle. Polar Coordinates always are from the current XY zero position. To shift the Polar Coordinates from machine zero use an offset or select a coordinate system.

In Absolute Mode the distance and angle is from the XY zero position and the angle starts with 0 on the X Positive axis and increases in a CCW direction about the Z axis. The code G1 @1ˆ90 is the same as G1 Y1.

In Relative Mode the distance and angle is also from the XY zero position but it is cumulative. This can be confusing at first how this works in incremental mode.

For example if you have the following program you might expect it to be a square pattern.

```
F100 G1 @.5 ^90
G91 @.5 ^90
@.5 ^90
@.5 ^90
@.5 ^90
G90 G0 X0 Y0 M2
```

You can see from the following figure that the output is not what you might expect. Because we added 0.5 to the distance each time the distance from the XY zero position increased with each line.

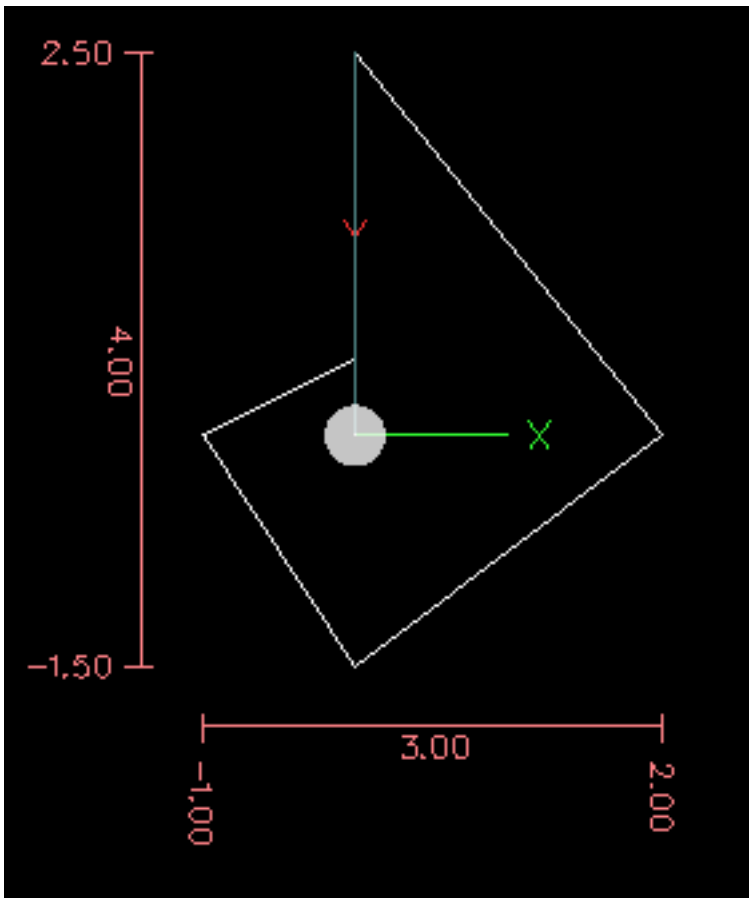



Figure 3.2: Polar Spiral

The following code will produce our square pattern.

```
F100 G1 @.5 ^90
G91 ^90
^90
^90
^90
G90 G0 X0 Y0 M2
```

As you can see by only adding to the angle by 90 degrees each time the end point distance is the same for each line.

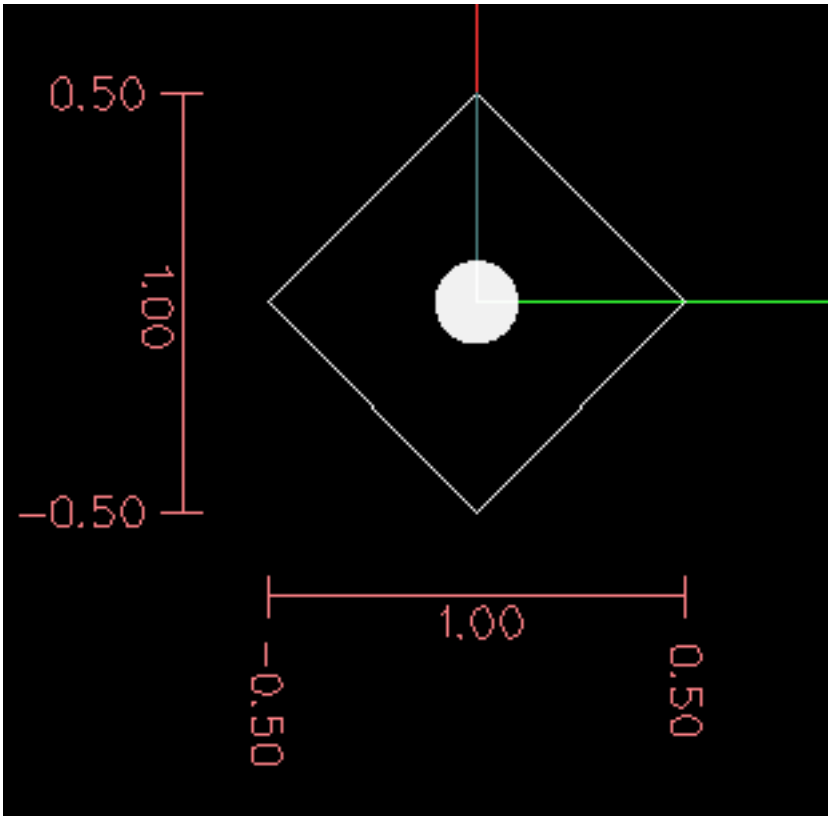



Figure 3.3: Polar Square

It is an error if:

- An incremental move is started at the origin
- A mix of Polar and and X or Y words are used

### 3.2.12 Modal Groups

Modal commands are arranged in sets called *modal groups*, and only one member of a modal group may be in force at any given time. In general, a modal group contains commands for which it is logically impossible for two members to be in effect at the same time - like measure in inches vs. measure in millimeters. A machining center may be in many modes at the same time, with one mode from each modal group being in effect. The modal groups are shown in the following Table.

Table 3.6: G-Code Modal Groups

| Modal Group Meaning | Member Words |
|---|---|
| Non-modal codes (Group 0) | G4, G10 G28, G30, G53 G92, G92.1, G92.2, G92.3, |
| Motion (Group 1) | G0, G1, G2, G3, G33, G38.x, G73, G76, G80, G81 |
| | G82, G83, G84, G85, G86, G87, G88, G89 |
| Plane selection (Group 2) | G17, G18, G19, G17.1, G18.1, G19.1 |
| Distance Mode (Group 3) | G90, G91 |
| Arc IJK Distance Mode (Group 4) | G90.1, G91.1 |
| Feed Rate Mode (Group 5) | G93, G94, G95 |
| Units (Group 6) | G20, G21 |
| Cutter Diameter Compensation (Group 7) | G40, G41, G42, G41.1, G42.1 |
| Tool Length Offset (Group 8) | G43, G43.1, G49 |
| Canned Cycles Return Mode (Group 10) | G98, G99 |
| Coordinate System (Group 12) | G54, G55, G56, G57, G58, G59, G59.1, G59.2, G59.3 |
| Control Mode (Group 13) | G61, G61.1, G64 |
| Spindle Speed Mode (Group 14) | G96, G97 |
| Lathe Diameter Mode (Group 15) | G7, G8 |

Table 3.7: M-Code Modal Groups

| Modal Group Meaning | Member Words |
|---|---|
| Stopping (Group 4) | M0, M1, M2, M30, M60 |
| Spindle (Group 7) | M3, M4, M5 |
| Coolant (Group 8) | (M7 M8 can both be on), M9 |
| Override Switches (Group 9) | M48, M49 |
| User Defined (Group 10) | M100-M199 |

For several modal groups, when a machining center is ready to accept commands, one member of the group must be in effect. There are default settings for these modal groups. When the machining center is turned on or otherwise re-initialized, the default values are automatically in effect.

Group 1, the first group on the table, is a group of G codes for motion. One of these is always in effect. That one is called the current motion mode.

It is an error to put a G-code from group 1 and a G-code from group 0 on the same line if both of them use axis words. If an axis word-using G-code from group 1 is implicitly in effect on a line (by having been activated on an earlier line), and a group 0 G-code that uses axis words appears on the line, the activity of the group 1 G-code is suspended for that line. The axis word-using G-codes from group 0 are G10, G28, G30, and G92.

It is an error to include any unrelated words on a line with *O*- flow control.

### 3.2.13 Comments

Comments can be added to lines of G code to help clear up the intention of the programmer. Comments can be embedded in a line using parentheses () or for the remainder of a line using a semi-colon. The semi-colon is not treated as the start of a comment when enclosed in parentheses.

Comments may appear between words, but not between words and their corresponding parameter. So, *S100(set speed)F200(feed)* is OK while *S(speed)100F(feed)* is not.

```
G0 (Rapid to start) X1 Y1
G0 X1 Y1 (Rapid to start; but don't forget the coolant)
M2 ; End of program.
```

There are several *active* comments which look like comments but cause some action, like *(debug,..)* or *(print,..)*. If there are several comments on a line, only the last comment will be interpreted according to these rules. Hence, a normal comment following an active comment will in effect disable the active comment. For example, *(foo) (debug,#1)* will print the value of parameter *#1*, however *(debug,#1)(foo)* will not.

A comment introduced by a semicolon is by definition the last comment on that line, and will always be interpreted for active comment syntax.

### 3.2.14 Messages

- *(MSG,)* - displays message if *MSG* appears after the left parenthesis and before any other printing characters. Variants of *MSG* which include white space and lower case characters are allowed. The rest of the characters before the right parenthesis are considered to be a message. Messages should be displayed on the message display device of the user interface if provided.

**Message Example**

```
(MSG, This is a message)
```

### 3.2.15 Probe Logging

- *(PROBEOPEN filename.txt)* - will open filename.txt and store the 9-number coordinate consisting of XYZABCUVW of each successful straight probe in it.
- *(PROBECLOSE)* - will close the open probelog file. For more information on probing see the G38 Section.

### 3.2.16 Logging

- *(LOGOPEN,filename.txt)* - opens the named log file. If the file already exists, it is truncated.
- *(LOGAPPEND,filename)* - opens the named log file. If the file already exists, the data is appended.
- *(LOGCLOSE)* - closes an open log file.
- *(LOG,)* - everything past the , is written to the log file if it is open. Supports expansion of parameters as described below.

Examples of logging are in *nc_files/examples/smartprobe.ngc* and in *nc_files/ngcgui_lib/rectange_probe.ngc* sample G code files.

### 3.2.17 Debug Messages

- *(DEBUG,)* - displays a message like *(MSG,)* with the addition of special handling for comment parameters as described below.

### 3.2.18 Print Messages

- *(PRINT,)* - messages are output to *stderr* with special handling for comment parameters as described below.

### 3.2.19 Comment Parameters

In the DEBUG, PRINT and LOG comments, the values of parameters in the message are expanded.

For example: to print a named global variable to stderr (the default console window).

**Parameters Example**

```
(print,endmill dia = #<_endmill_dia>)
(print,value of variable 123 is: #123)
```

Inside the above types of comments, sequences like *#123* are replaced by the value of the parameter 123. Sequences like *#<named parameter>* are replaced by the value of the named parameter. Named parameters will have white space removed from them. So, *#<named parameter>* will be converted to *#<namedparameter>*.

### 3.2.20 File Requirements

A G code file must contain one or more lines of G code and be terminated with a Program End. Any G code past the program end is not evaluated.

If a program end code is not used a pair of percent signs % with the first percent sign on the first line of the file followed by one or more lines of G code and a second percent sign. Any code past the second percent sign is not evaluated.

**Warning**
Using % to wrap a G code file will not do the same thing as using a program end. The machine will be in what ever state the program left it in using %, the spindle and coolant may still be on and things like G90/91 are left as the last program set them. If you don't use a proper preamble the next program could start in a dangerous condition.

**Note**
The file must be created with a text editor like Gedit and not a word processor like Open Office Word Processor.

### 3.2.21 File Size

The interpreter and task are carefully written so that the only limit on part program size is disk capacity. The TkLinuxCNC and Axis interface both load the program text to display it to the user, though, so RAM becomes a limiting factor. In Axis, because the preview plot is drawn by default, the redraw time also becomes a practical limit on program size. The preview can be turned off in Axis to speed up loading large part programs. In Axis sections of the preview can be turned off using preview control comments.

### 3.2.22 G Code Order of Execution

The order of execution of items on a line is defined not by the position of each item on the line, but by the following list:

- O-word commands (optionally followed by a comment but no other words allowed on the same line)
- Comment (including message)
- Set feed rate mode (G93, G94).
- Set feed rate (F).
- Set spindle speed (S).
- Select tool (T).

- HAL pin I/O (M62-M68).
- Change tool (M6) and Set Tool Number (M61).
- Spindle on or off (M3, M4, M5).
- Save State (M70, M73), Restore State (M72), Invalidate State (M71).
- Coolant on or off (M7, M8, M9).
- Enable or disable overrides (M48, M49,M50,M51,M52,M53).
- User-defined Commands (M100-M199).
- Dwell (G4).
- Set active plane (G17, G18, G19).
- Set length units (G20, G21).
- Cutter radius compensation on or off (G40, G41, G42)
- Cutter length compensation on or off (G43, G49)
- Coordinate system selection (G54, G55, G56, G57, G58, G59, G59.1, G59.2, G59.3).
- Set path control mode (G61, G61.1, G64)
- Set distance mode (G90, G91).
- Set retract mode (G98, G99).
- Go to reference location (G28, G30) or change coordinate system data (G10) or set axis offsets (G92, G92.1, G92.2, G94).
- Perform motion (G0 to G3, G33, G38.x, G73, G76, G80 to G89), as modified (possibly) by G53.
- Stop (M0, M1, M2, M30, M60).

### 3.2.23 G Code Best Practices

**Use an appropriate decimal precision** Use at least 3 digits after the decimal when milling in millimeters, and at least 4 digits after the decimal when milling in inches.

**Use consistent white space** G-code is most legible when at least one space appears before words. While it is permitted to insert white space in the middle of numbers, there is no reason to do so.

**Use Center-format arcs** Center-format arcs (which use *I- J- K-* instead of *R-* ) behave more consistently than R-format arcs, particularly for included angles near 180 or 360 degrees.

**Use a Preamble set modal groups** When correct execution of your program depends on modal settings, be sure to set them at the beginning of the part program. Modes can carry over from previous programs and from the MDI commands.

**Example Preamble for a Mill**

```
G17 G20 G40 G49 G54 G80 G90 G94
```

G17 use XY plane, G20 inch mode, G40 cancel diameter compensation, G49 cancel length offset, G54 use coordinate system 1, G80 cancel canned cycles, G90 absolute distance mode, G94 feed/minute mode.

Perhaps the most critical modal setting is the distance units—If you do not include G20 or G21, then different machines will mill the program at different scales. Other settings, such as the return mode in canned cycles may also be important.

**Don't put too many things on one line** Ignore everything in Section Order of Execution, and instead write no line of code that is the slightest bit ambiguous.

**Don't set & use a parameter on the same line** Don't use and set a parameter on the same line, even though the semantics are well defined. Updating a variable to a new value, such as *#1=[#1+#2]* is OK.

**Don't use line numbers** Line numbers offer no benefits. When line numbers are reported in error messages, the numbers refer to the line number in the file, not the N-word value.

### 3.2.24 Linear and Rotary Axis

Because the meaning of an F-word in feed-per-minute mode varies depending on which axes are commanded to move, and because the amount of material removed does not depend only on the feed rate, it may be easier to use G93 inverse time feed mode to achieve the desired material removal rate.

### 3.2.25 Common Error Messages

- *G code out of range* - A G code greater than G99 was used, the scope of G codes in LinuxCNC is 0 - 99. Not every number between 0 and 99 is a valid G code.
- *Unknown g code used* - A G code was used that is not part of the LinuxCNC G code language.
- *i,j,k word with no Gx to use it* - i, j and k words must be used on the same line as the G code.
- *Cannot use axis values without a g code that uses them* - Axis values can not be used on a line without either a modal G code in effect or a G code on the same line.
- *File ended with no percent sign or program end* - Every G code file must end in a M2 or M30 or be wrapped with the percent sign %.

## 3.3 G Codes

### 3.3.1 Conventions

Conventions used in this section

In the G code prototypes the hyphen (-) stands for a real value and (<>) denotes an optional item.

If *L-* is written in a prototype the - will often be referred to as the *L number*, and so on for any other letter.

In the G code prototypes the word *axes* stands for any axis as defined in your configuration.

An optional value will be written like this <*L-*>.

A real value may be:

- An explicit number, *4*
- An expression, *[2+2]*
- A parameter value, *#88*
- A unary function value, *acos[0]*

In most cases, if *axis* words are given (any or all of *X Y Z A B C U V W*), they specify a destination point.

Axis numbers are in the currently active coordinate system, unless explicitly described as being in the absolute coordinate system.

Where axis words are optional, any omitted axes will retain their original value.

Any items in the G code prototypes not explicitly described as optional are required.

The values following letters are often given as explicit numbers. Unless stated otherwise, the explicit numbers can be real values. For example, *G10 L2* could equally well be written *G[2*5] L[1+1]*. If the value of parameter 100 were 2, *G10 L#100* would also mean the same.

If *L-* is written in a prototype the - will often be referred to as the *L number*, and so on for any other letter.

### 3.3.2 G Code Quick Reference Table

| Code | Description |
|---|---|
| G0 | Coordinated Motion at Rapid Rate |
| G1 | Coordinated Motion at Feed Rate |
| G2 G3 | Coordinated Helical Motion at Feed Rate |
| G4 | Dwell |
| G5 | Cubic Spline |
| G5.1 | Quadratic B-Spline |
| G5.2 | NURBS, add control point |
| G7 | Diameter Mode (lathe) |
| G8 | Radius Mode (lathe) |
| G10 L1 | Set Tool Table Entry |
| G10 L10 | Set Tool Table, Calculated, Workpiece |
| G10 L11 | Set Tool Table, Calculated, Fixture |
| G10 L2 | Coordinate System Origin Setting |
| G10 L20 | Coordinate System Origin Setting Calculated |
| G17 - G19.1 | Plane Select |
| G20 G21 | Set Units of Measure |
| G28 - G28.1 | Go to Predefined Position |
| G30 - G30.1 | Go to Predefined Position |
| G33 | Spindle Synchronized Motion |
| G33.1 | Rigid Tapping |
| G38.2 - G38.5 | Probing |
| G40 | Cancel Cutter Compensation |
| G41 G42 | Cutter Compensation |
| G41.1 G42.1 | Dynamic Cutter Compensation |
| G43 | Use Tool Length Offset from Tool Table |
| G43.1 | Dynamic Tool Length Offset |
| G43.2 | Apply additional Tool Length Offset |
| G49 | Cancel Tool Length Offset |
| G53 | Move in Machine Coordinates |
| G54-G59.3 | Select Coordinate System (1 - 9) |
| G61 G61.1 | Path Control Mode |
| G64 | Path Control Mode with Optional Tolerance |
| G73 | Drilling Cycle with Chip Breaking |
| G76 | Multi-pass Threading Cycle (Lathe) |
| G80 | Cancel Motion Modes |
| G81 | Drilling Cycle |
| G82 | Drilling Cycle with Dwell |
| G83 | Drilling Cycle with Peck |
| G85 | Boring Cycle, No Dwell, Feed Out |
| G86 | Boring Cycle, Stop, Rapid Out |
| G89 | Boring Cycle, Dwell, Feed Out |
| G90 G91 | Distance Mode |
| G90.1 G91.1 | Arc Distance Mode |
| G92 | Coordinate System Offset |
| G92.1 G92.2 | Cancel G92 Offsets |
| G92.3 | Restore G92 Offsets |
| G93 G94 G95 | Feed Modes |
| G96 | Spindle Control Mode |
| G98 G99 | Canned Cycle Z Retract Mode |

### 3.3.3 G0 Rapid Move

```
G0 axes
```

For rapid motion, program *G0 axes*, where all the axis words are optional. The *G0* is optional if the current motion mode is *G0*.

This will produce coordinated motion to the destination point at the maximum rapid rate (or slower). *G0* is typically used as a positioning move.

#### 3.3.3.1 Rapid Velocity Rate

The MAX_VELOCITY setting in the ini file [TRAJ] section defines the maximum rapid traverse rate. The maximum rapid traverse rate can be higher than the individual axes MAX_VELOCITY setting during a coordinated move. The maximum rapid traverse rate can be slower than the MAX_VELOCITY setting in the [TRAJ] section if an axis MAX_VELOCITY or trajectory constraints limit it.

**G0 Example**

```
G90 (set absolute distance mode)
G0 X1 Y-2.3 (Rapid linear move from current location to X1 Y-2.3)
M2 (end program)
```

- See G90 & M2 sections for more information.

If cutter compensation is active, the motion will differ from the above; see the Cutter Compensation Section.

If *G53* is programmed on the same line, the motion will also differ; see the G53 Section for more information.

The path of a G0 rapid motion can be rounded at direction changes and depends on the trajectory control settings and maximum acceleration of the axes.

It is an error if:

- An axis letter is without a real value.
- An axis letter is used that is not configured

### 3.3.4 G1 Linear Move

```
G1 axes
```

For linear (straight line) motion at programed feed rate (for cutting or not), program *G1 'axes'*, where all the axis words are optional. The *G1* is optional if the current motion mode is *G1*. This will produce coordinated motion to the destination point at the current feed rate (or slower).

**G1 Example**

```
G90 (set absolute distance mode)
G1 X1.2 Y-3 F10 (linear move at a feed rate of 10 from current position to X1.2 Y-3)
Z-2.3 (linear move at same feed rate from current position to Z-2.3)
Z1 F25 (linear move at a feed rate of 25 from current position to Z1)
M2 (end program)
```

- See G90 & F & M2 sections for more information.

If cutter compensation is active, the motion will differ from the above; see the Cutter Compensation Section.

If *G53* is programmed on the same line, the motion will also differ; see the G53 Section for more information.

It is an error if:

- No feed rate has been set.
- An axis letter is without a real value.
- An axis letter is used that is not configured

### 3.3.5 G2, G3 Arc Move

```
G2 or G3 axes offsets (center format)
G2 or G3 axes R- (radius format)
G2 or G3 offsets|R- <P-> (full circles)
```

A circular or helical arc is specified using either *G2* (clockwise arc) or *G3* (counterclockwise arc) at the current feed rate. The direction (CW, CCW) is as viewed from the positive end of the axis about which the circular motion occurs.

The axis of the circle or helix must be parallel to the X, Y, or Z axis of the machine coordinate system. The axis (or, equivalently, the plane perpendicular to the axis) is selected with *G17* (Z-axis, XY-plane), *G18* (Y-axis, XZ-plane), or *G19* (X-axis, YZ-plane). Planes *17.1*, *18.1*, and *19.1* are not currently supported. If the arc is circular, it lies in a plane parallel to the selected plane.

To program a helix, include the axis word perpendicular to the arc plane: for example, if in the *G17* plane, include a *Z* word. This will cause the *Z* axis to move to the programmed value during the circular *XY* motion.

To program an arc that gives more than one full turn, use the *P* word specifying the number of full turns plus the programmed arc. The *P* word must be an integer. If *P* is unspecified, the behavior is as if *P1* was given: that is, only one full or partial turn will result. For example, if a 180 degree arc is programmed with a P2, the resulting motion will be 1 1/2 rotations. For each P increment above 1 an extra full circle is added to the programmed arc. Multi turn helical arcs are supported and give motion useful for milling holes or threads.

If a line of code makes an arc and includes rotary axis motion, the rotary axes turn at a constant rate so that the rotary motion starts and finishes when the XYZ motion starts and finishes. Lines of this sort are hardly ever programmed.

If cutter compensation is active, the motion will differ from the above; see the Cutter Compensation Section.

The arc center is absolute or relative as set by G90.1 or G91.1 respectively.

Two formats are allowed for specifying an arc: Center Format and Radius Format.

It is an error if:

- No feed rate has been set.
- The P word is not an integer.

#### 3.3.5.1 Center Format Arcs

Center format arcs are more accurate than radius format arcs and are the preferred format to use.

The end point of the arc along with the offset to the center of the arc from the current location are used to program arcs that are less than a full circle. It is OK if the end point of the arc is the same as the current location.

The offset to the center of the arc from the current location and optionally the number of turns are used to program full circles.

When programming arcs an error due to rounding can result from using a precision of less than 4 decimal places (0.0000) for inch and less than 3 decimal places (0.000) for millimeters.

**Incremental Arc Distance Mode** Arc center offsets are a relative distance from the start location of the arc. Incremental Arc Distance Mode is default.

One or more axis words and one or more offsets must be programmed for an arc that is less than 360 degrees.

No axis words and one or more offsets must be programmed for full circles. The *P* word defaults to 1 and is optional.

For more information on 'Incremental Arc Distance Mode see the G91.1 section.

**Absolute Arc Distance Mode** Arc center offsets are the absolute distance from the current 0 position of the axis.

One or more axis words and *both* offsets must be programmed for arcs less than 360 degrees.

No axis words and both offsets must be programmed for full circles. The *P* word defaults to 1 and is optional.

For more information on 'Absolute Arc Distance Mode see the G90.1 section.

**XY-plane (G17)**

```
G2 or G3 <X- Y- Z- I- J- P->
```

- *Z* - helix
- *I* - X offset
- *J* - Y offset
- *P* - number of turns

**XZ-plane (G18)**

```
G2 or G3 <X- Z- Y- I- K- P->
```

- *Y* - helix
- *I* - X offset
- *K* - Z offset
- *P* - number of turns

**YZ-plane (G19)**

```
G2 or G3 <Y- Z- X- J- K- P->
```

- *X* - helix
- *J* - Y offset
- *K* - Z offset
- *P* - number of turns

It is an error if:

- No feed rate is set with the F word.
- No offsets are programmed.
- When the arc is projected on the selected plane, the distance from the current point to the center differs from the distance from the end point to the center by more than (.05 inch/.5 mm) OR ((.0005 inch/.005mm) AND .1% of radius).

Deciphering the Error message *Radius to end of arc differs from radius to start:*

- *start* - the current position
- *center* - the center position as calculated using the i, j, or k words
- *end* - the programmed end point
- *r1* - radius from the start position to the center
- *r2* - radius from the end position to the center

#### 3.3.5.2 Center Format Examples

Calculating arcs by hand can be difficult at times. One option is to draw the arc with a cad program to get the coordinates and offsets. Keep in mind the tolerance mentioned above, you may have to change the precision of your cad program to get the desired results. Another option is to calculate the coordinates and offset using formulas. As you can see in the following figures a triangle can be formed from the current position the end position and the arc center.

In the following figure you can see the start position is X0 Y0, the end position is X1 Y1. The arc center position is at X1 Y0. This gives us an offset from the start position of 1 in the X axis and 0 in the Y axis. In this case only an I offset is needed.

**G2 Example Line**

```
G0 X0 Y0
G2 X1 Y1 I1 F10 (clockwise arc in the XY plane)
```

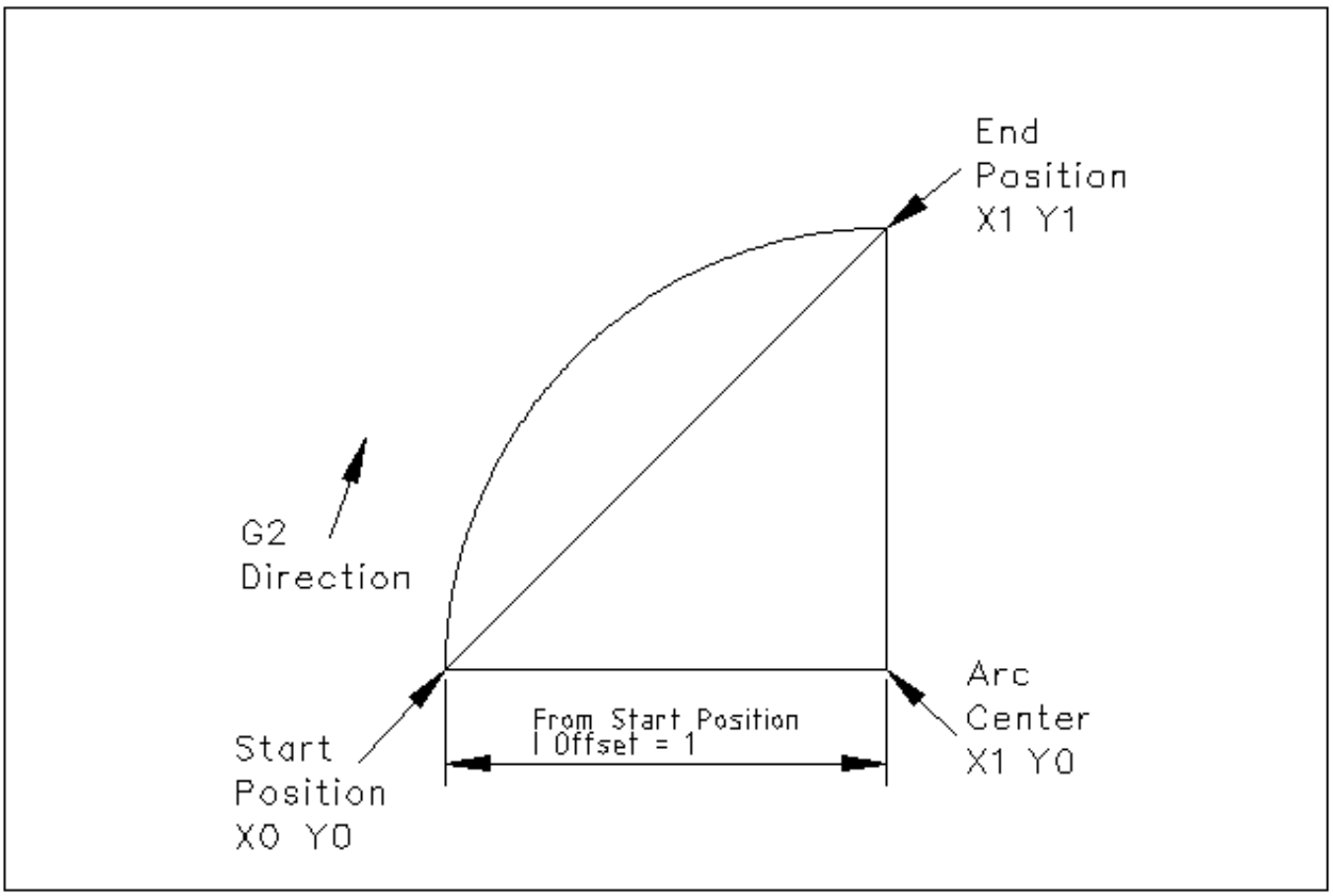



Figure 3.4: G2 Example

In the next example we see the difference between the offsets for Y if we are doing a G2 or a G3 move. For the G2 move the start position is X0 Y0, for the G3 move it is X0 Y1. The arc center is at X1 Y0.5 for both moves. The G2 move the J offset is 0.5 and the G3 move the J offset is -0.5.

**G2-G3 Example Line**

```
G0 X0 Y0
```

```
G2 X0 Y1 I1 J0.5 F25 (clockwise arc in the XY plane)
G3 X0 Y0 I1 J-0.5 F25 (counterclockwise arc in the XY plane)
```

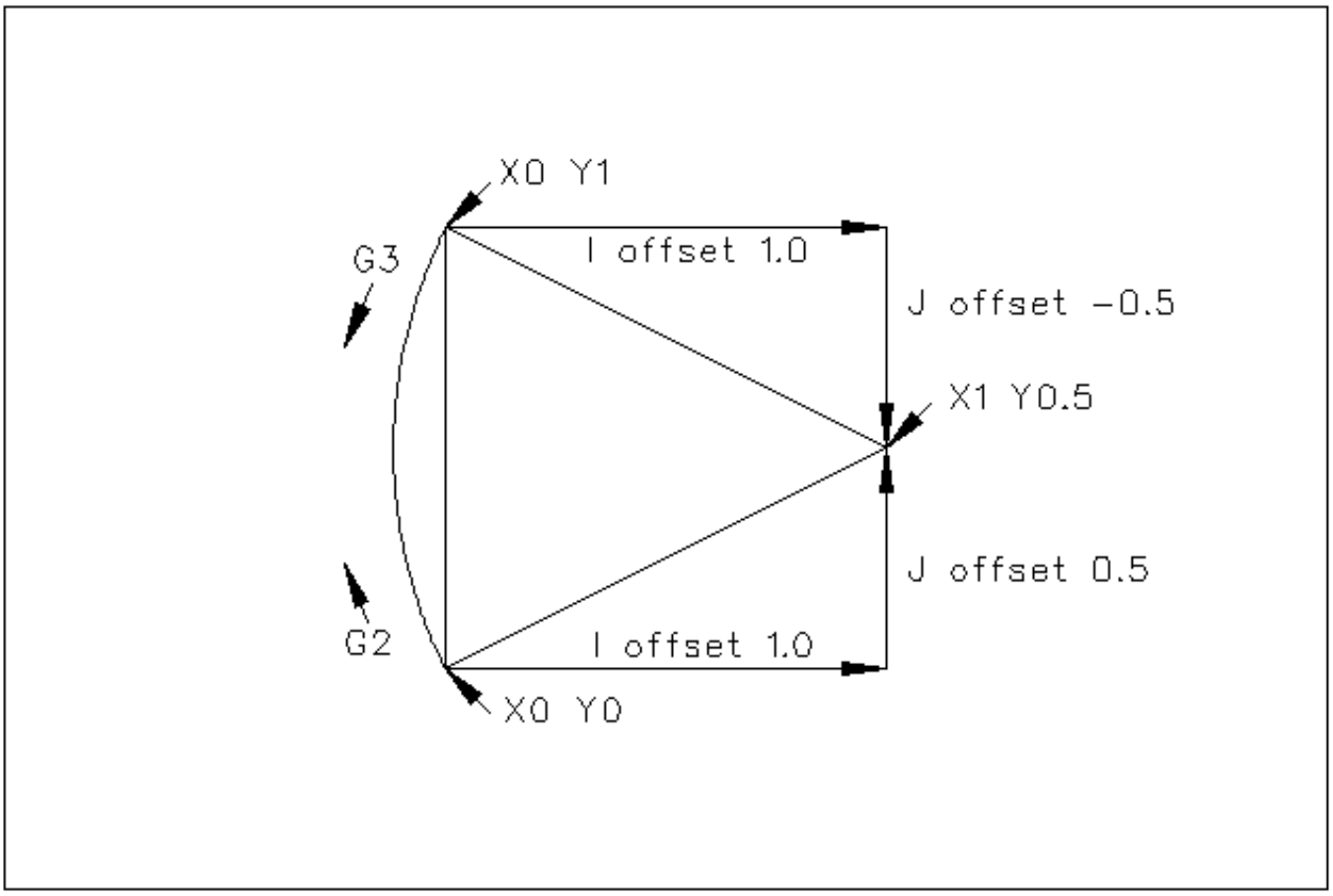



Figure 3.5: G2-G3 Example

In the next example we show how the arc can make a helix in the Z axis by adding the Z word.

**G2 Example Helix**

```
G0 X0 Y0 Z0
G17 G2 X10 Y16 I3 J4 Z-1 (helix arc with Z added)
```

In the next example we show how to make more than one turn using the P word.

**P word Example**

```
G0 X0 Y0 Z0
G2 X0 Y1 Z-1 I1 J0.5 P2 F25
```

In the center format, the radius of the arc is not specified, but it may be found easily as the distance from the center of the circle to either the current point or the end point of the arc.

#### 3.3.5.3 Radius Format Arcs

```
G2 or G3 axes R- <P->
```

- *R* - radius from current position

It is not good practice to program radius format arcs that are nearly full circles or nearly semicircles because a small change in the location of the end point will produce a much larger change in the location of the center of the circle (and, hence, the middle of the arc). The magnification effect is large enough that rounding error in a number can produce out-of-tolerance cuts. For instance, a 1% displacement of the endpoint of a 180 degree arc produced a 7% displacement of the point 90 degrees along the arc. Nearly full circles are even worse. Other size arcs (in the range tiny to 165 degrees or 195 to 345 degrees) are OK.

In the radius format, the coordinates of the end point of the arc in the selected plane are specified along with the radius of the arc. Program *G2 axes R-* (or use *G3* instead of *G2* ). R is the radius. The axis words are all optional except that at least one of the two words for the axes in the selected plane must be used. The R number is the radius. A positive radius indicates that the arc turns through less than 180 degrees, while a negative radius indicates a turn of more than 180 degrees. If the arc is helical, the value of the end point of the arc on the coordinate axis parallel to the axis of the helix is also specified.

It is an error if:

- both of the axis words for the axes of the selected plane are omitted
- the end point of the arc is the same as the current point.

**G2 Example Line**

```
G17 G2 X10 Y15 R20 Z5 (radius format with arc)
```

The above example makes a clockwise (as viewed from the positive Z-axis) circular or helical arc whose axis is parallel to the Z-axis, ending where X=10, Y=15, and Z=5, with a radius of 20. If the starting value of Z is 5, this is an arc of a circle parallel to the XY-plane; otherwise it is a helical arc.

### 3.3.6 G4 Dwell

```
G4 P-
```

- *P* - seconds to dwell (floating point)

The P number is the time in seconds that all axes will remain unmoving. The P number is a floating point number so fractions of a second may be used. G4 does not affect spindle, coolant and any I/O.

**G4 Example Line**

```
G4 P0.5 (wait for 0.5 seconds before proceeding)
```

It is an error if:

- the P number is negative or not specified.

### 3.3.7 G5 Cubic Spline

```
G5 X- Y- <I- J-> P- Q-
```

- *I* - X incremental offset from start point to first control point
- *J* - Y incremental offset from start point to first control point
- *P* - X incremental offset from end point to second control point
- *Q* - Y incremental offset from end point to second control point

G5 creates a cubic B-spline in the XY plane with the X and Y axes only. P and Q must both be specified for every G5 command.

For the first G5 command in a series of G5 commands, I and J must both be specified. For subsequent G5 commands, either both I and J must be specified, or neither. If I and J are unspecified, the starting direction of this cubic will automatically match the ending direction of the previous cubic (as if I and J are the negation of the previous P and Q).

For example, to program a curvy N shape:

**G5 Sample initial cubic spline**

```
G90 G17
G0 X0 Y0
G5 I0 J3 P0 Q-3 X1 Y1
```

A second curvy N that attaches smoothly to this one can now be made without specifying I and J:

**G5 Sample subsequent cubic spline**

```
G5 P0 Q-3 X2 Y2
```

It is an error if:

- P and Q are not both specified
- Just one of I or J are specified
- I or J are unspecified in the first of a series of G5 commands
- An axis other than X or Y is specified
- The active plane is not G17

### 3.3.8 G5.1 Quadratic Spline

```
G5.1 X- Y- I- J-
```

- *I* - X incremental offset from start point to control point
- *J* - Y incremental offset from start point to control point

G5.1 creates a quadratic B-spline in the XY plane with the X and Y axis only. Not specifying I or J gives zero offset for the unspecified axis, so one or both must be given.

For example, to program a parabola, through the origin, from X-2 Y4 to X2 Y4:

**G5.1 Sample quadratic spline**

```
G90 G17
G0 X-2 Y4
G5.1 X2 I2 J-8
```

It is an error if:

- both I and J offset are unspecified or zero
- An axis other than X or Y is specified
- The active plane is not G17

### 3.3.9 G5.2 G5.3 NURBS Block

```
G5.2 <P-> <X- Y-> <L->
X- Y- <P->
...
G5.3
```

Warning: G5.2, G5.3 is experimental and not fully tested.

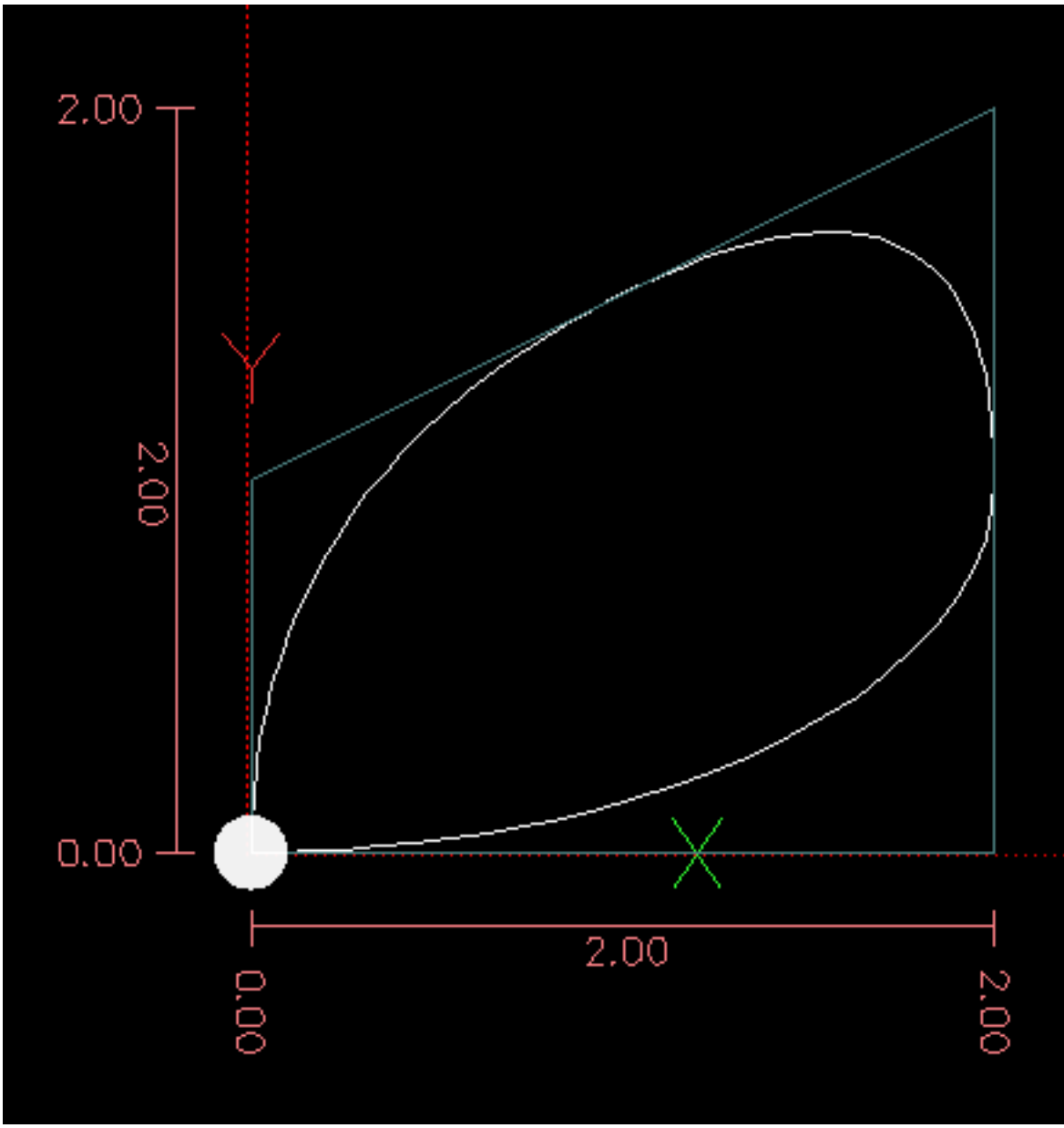
G5.2 is for opening the data block defining a NURBS and G5.3 for closing the data block. In the lines between these two codes the curve control points are defined with both their related *weights* (P) and the parameter (L) which determines the order of the curve.

The current coordinate, before the first G5.2 command, is always taken as the first NURBS control point. To set the weight for this first control point, first program G5.2 P- without giving any X Y.

The default weight if P is unspecified is 1. The default order if L is unspecified is 3.

**G5.2 Example**

```
G0 X0 Y0 (rapid move)
F10 (set feed rate)
G5.2 P1 L3
     X0 Y1 P1
     X2 Y2 P1
     X2 Y0 P1
     X0 Y0 P2
G5.3
; The rapid moves show the same path without the NURBS Block
G0 X0 Y1
   X2 Y2
   X2 Y0
   X0 Y0
M2
```




**Sample NURBS Output**

More information on NURBS can be found here:

http://wiki.linuxcnc.org/cgi-bin/wiki.pl?NURBS

### 3.3.10 G7 Lathe Diameter Mode

```
G7
```

Program G7 to enter the diameter mode for axis X on a lathe. When in the diameter mode the X axis moves on a lathe will be 1/2 the distance to the center of the lathe. For example X1 would move the cutter to 0.500” from the center of the lathe thus giving a 1” diameter part.

### 3.3.11 G8 Lathe Radius Mode

```
G8
```

Program G8 to enter the radius mode for axis X on a lathe. When in Radius mode the X axis moves on a lathe will be the distance from the center. Thus a cut at X1 would result in a part that is 2" in diameter. G8 is default at power up.

### 3.3.12 G10 L1 Set Tool Table

```
G10 L1 P- axes <R- I- J- Q->
```

- *P* - tool number
- *R* - radius of tool
- *I* - front angle (lathe)
- *J* - back angle (lathe)
- *Q* - orientation (lathe)

G10 L1 sets the tool table for the *P* tool number to the values of the words.

A valid G10 L1 rewrites and reloads the tool table.

**G10 L1 Example Line**

```
G10 L1 P1 Z1.5 (set tool 1 Z offset from the machine origin to 1.5)
G10 L1 P2 R0.015 Q3 (lathe example setting tool 2 radius to 0.015 and orientation to 3)
```

It is an error if:

- Cutter Compensation is on
- The P number is unspecified
- The P number is not a valid tool number from the tool table
- The P number is 0

For more information on cutter orientation used by the *Q* word, see the Lathe Tool Orientation diagram.

### 3.3.13 G10 L2 Set Coordinate System

```
G10 L2 P- <axes R->
```

- *P* - coordinate system (0-9)
- *R* - rotation about the Z axis

G10 L2 offsets the origin of the axes in the coordinate system specified to the value of the axis word. The offset is from the machine origin established during homing. The offset value will replace any current offsets in effect for the coordinate system specified. Axis words not used will not be changed.

Program P0 to P9 to specify which coordinate system to change.

Table 3.8: Coordinate System

| P Value | Coordinate System | G code |
|---|---|---|
| 0 | Active | n/a |
| 1 | 1 | G54 |
| 2 | 2 | G55 |
| 3 | 3 | G56 |
| 4 | 4 | G57 |
| 5 | 5 | G58 |
| 6 | 6 | G59 |
| 7 | 7 | G59.1 |
| 8 | 8 | G59.2 |
| 9 | 9 | G59.3 |

Optionally program R to indicate the rotation of the XY axis around the Z axis. The direction of rotation is CCW as viewed from the positive end of the Z axis.

All axis words are optional.

Being in incremental distance mode (G91) has no effect on *G10 L2*.

Important Concepts:

- G10 L2 Pn does not change from the current coordinate system to the one specified by P, you have to use G54-59.3 to select a coordinate system.
- When a rotation is in effect jogging an axis will only move that axis in a positive or negative direction and not along the rotated axis.
- If a *G92* origin offset was in effect before *G10 L2*, it will continue to be in effect afterwards.
- The coordinate system whose origin is set by a *G10* command may be active or inactive at the time the *G10* is executed. If it is currently active, the new coordinates take effect immediately.

It is an error if:

- The P number does not evaluate to an integer in the range 0 to 9.
- An axis is programmed that is not defined in the configuration.

**G10 L2 Example Line**

```
G10 L2 P1 X3.5 Y17.2
```

In the above example the origin of the first coordinate system (the one selected by *G54*) is set to be X=3.5 and Y=17.2. Because only X and Y are specified, the origin point is only moved in X and Y; the other coordinates are not changed.

**G10 L2 Example Line**

```
G10 L2 P1 X0 Y0 Z0 (clear offsets for X,Y & Z axes in coordinate system 1)
```

The above example sets the XYZ coordinates of the coordinate system 1 to the machine origin.

The coordinate system is described in the Coordinate System Section.

### 3.3.14 G10 L10 Set Tool Table

```
G10 L10 P- axes <R- I- J- Q->
```

- *P* - tool number
- *R* - radius of tool
- *I* - front angle (lathe)
- *J* - back angle (lathe)
- *Q* - orientation (lathe)

G10 L10 changes the tool table entry for tool P so that if the tool offset is reloaded, with the machine in its current position and with the current G5x and G92 offsets active, the current coordinates for the given axes will become the given values. The axes that are not specified in the G10 L10 command will not be changed. This could be useful with a probe move as described in the G38 section.

**G10 L10 Example**

```
T1 M6 G43 (load tool 1 and tool length offsets)
G10 L10 P1 Z1.5 (set the current position for Z to be 1.5)
G43 (reload the tool length offsets from the changed tool table)
M2 (end program)
```

- See T & M6, and G43/G43.1 sections for more information.

It is an error if:

- Cutter Compensation is on
- The P number is unspecified
- The P number is not a valid tool number from the tool table
- The P number is 0

### 3.3.15 G10 L11 Set Tool Table

```
G10 L11 P- axes <R- I- J- Q->
```

- *P* - tool number
- *R* - radius of tool
- *I* - front angle (lathe)
- *J* - back angle (lathe)
- *Q* - orientation (lathe)

G10 L11 is just like G10 L10 except that instead of setting the entry according to the current offsets, it is set so that the current coordinates would become the given value if the new tool offset is reloaded and the machine is placed in the G59.3 coordinate system without any G92 offset active.

This allows the user to set the G59.3 coordinate system according to a fixed point on the machine, and then use that fixture to measure tools without regard to other currently-active offsets.

It is an error if:

- Cutter Compensation is on
- The P number is unspecified
- The P number is not a valid tool number from the tool table
- The P number is 0

### 3.3.16 G10 L20 Set Coordinate System

```
G10 L20 P- axes
```

- *P* - coordinate system (0-9)

G10 L20 is similar to G10 L2 except that instead of setting the offset/entry to the given value, it is set to a calculated value that makes the current coordinates become the given value.

**G10 L20 Example Line**

```
G10 L20 P1 X1.5 (set the X axis current location in coordinate system 1 to 1.5)
```

It is an error if:

- The P number does not evaluate to an integer in the range 0 to 9.
- An axis is programmed that is not defined in the configuration.

### 3.3.17 G17 - G19.1 Plane Select

These codes set the current plane as follows:

- *G17* - XY (default)
- *G18* - ZX
- *G19* - YZ
- *G17.1* - UV
- *G18.1* - WU
- *G19.1* - VW

The UV, WU and VW planes do not support arcs.

It is a good idea to include a plane selection in the preamble of each G code file.

The effects of having a plane selected are discussed in Section G2 G3 Arcs and Section G81 G89

### 3.3.18 G20, G21 Units

- *G20* - to use inches for length units.
- *G21* - to use millimeters for length units.

It is a good idea to include units in the preamble of each G code file.

### 3.3.19 G28, G28.1 Go/Set Predefined Position

**Warning**
Only use G28 when your machine is homed to a repeatable position and the desired G28 position has been stored with G28.1.

G28 uses the values stored in parameters 5161-5166 as the X Y Z A B C U V W final point to move to. The parameter values are *absolute* machine coordinates in the native machine *units* as specified in the ini file. All axes defined in the ini file will be moved when a G28 is issued. If no positions are stored with G28.1 then all axes will go to the machine origin.

- *G28* - makes a rapid move from the current position to the *absolute* position of the values in parameters 5161-5166.
- *G28 axes* - makes a rapid move to the position specified by *axes* including any offsets, then will make a rapid move to the *absolute* position of the values in parameters 5161-5166 for all *axes* specified. Any *axis* not specified will not move.
- *G28.1* - stores the current *absolute* position into parameters 5161-5166.

**G28 Example Line**

```
G28 Z2.5 (rapid to Z2.5 then to Z location specified in #5163)
```

It is an error if :

- Cutter Compensation is turned on

### 3.3.20 G30, G30.1 Go/Set Predefined Position

**Warning**
Only use G30 when your machine is homed to a repeatable position and the desired G30 position has been stored with G30.1.

G30 functions the same as G28 but uses the values stored in parameters 5181-5186 as the X Y Z A B C U V W final point to move to. The parameter values are *absolute* machine coordinates in the native machine *units* as specified in the ini file. All axes defined in the ini file will be moved when a G30 is issued. If no positions are stored with G30.1 then all axes will go to the machine origin.

**Note**
G30 parameters will be used to move the tool when a M6 is programmed if TOOL_CHANGE_AT_G30=1 is in the [EMCIO] section of the ini file.

- *G30* - makes a rapid move from the current position to the *absolute* position of the values in parameters 5181-5186.
- *G30 axes* - makes a rapid move to the position specified by *axes* including any offsets, then will make a rapid move to the *absolute* position of the values in parameters 5181-5186 for all *axes* specified. Any *axis* not specified will not move.
- *G30.1* - stores the current absolute position into parameters 5181-5186.

**G30 Example Line**

```
G30 Z2.5 (rapid to Z2.5 then to the Z location specified in #5i83)
```

It is an error if :

- Cutter Compensation is turned on

### 3.3.21 G33 Spindle Synchronized Motion

```
G33 X- Y- Z- K-
```

- *K* - distance per revolution

For spindle-synchronized motion in one direction, code *G33 X- Y- Z- K-* where K gives the distance moved in XYZ for each revolution of the spindle. For instance, if starting at *Z=0*, *G33 Z-1 K.0625* produces a 1 inch motion in Z over 16 revolutions of the spindle. This command might be part of a program to produce a 16TPI thread. Another example in metric, *G33 Z-15 K1.5* produces a movement of 15mm while the spindle rotates 10 times for a thread of 1.5mm.

Spindle-synchronized motion waits for the spindle index and spindle at speed pins, so multiple passes line up. *G33* moves end at the programmed endpoint. G33 could be used to cut tapered threads or a fusee.

All the axis words are optional, except that at least one must be used.

**Note**
K follows the drive line described by *X- Y- Z-*. K is not parallel to the Z axis if X or Y endpoints are used for example when cutting tapered threads.

**Technical Info** At the beginning of each G33 pass, LinuxCNC uses the spindle speed and the machine acceleration limits to calculate how long it will take Z to accelerate after the index pulse, and determines how many degrees the spindle will rotate during that time. It then adds that angle to the index position and computes the Z position using the corrected spindle angle. That means that Z will reach the correct position just as it finishes accelerating to the proper speed, and can immediately begin cutting a good thread.

**HAL Connections** The pins *motion.spindle-at-speed* and the *encoder.n.phase-Z* for the spindle must be connected in your HAL file before G33 will work. See the Integrators Manual for more information on spindle synchronized motion.

**G33 Example**

```
G90 (absolute distance mode)
G0 X1 Z0.1 (rapid to position)
S100 M3 (start spindle turning)
G33 Z-2 K0.125 (move Z axis to -2 at a rate to equal 0.125 per revolution)
G0 X1.25 (rapid move tool away from work)
Z0.1 (rapid move to starting Z position)
M2 (end program)
```

- See G90 & G0 & M2 sections for more information.

It is an error if:

- All axis words are omitted.
- The spindle is not turning when this command is executed
- The requested linear motion exceeds machine velocity limits due to the spindle speed

### 3.3.22 G33.1 Rigid Tapping

```
G33.1 X- Y- Z- K-
```

- *K* - distance per revolution

**Warning**
For Z only tapping preposition the XY location prior to calling G33.1 and only use a Z word in the G33.1. If the coordinates specified are not the current coordinates when calling G33.1 for tapping the move will not be along the Z axis but will be a coordinated, spindle-synchronized move from the current location to the location specified and back.

For rigid tapping (spindle synchronized motion with return), code *G33.1 X- Y- Z- K-* where *K-* gives the distance moved for each revolution of the spindle.

A rigid tapping move consists of the following sequence:

1. A move from the current coordinate to the specified coordinate, synchronized with the spindle at the given ratio and starting from the current coordinate with a spindle index pulse.
2. When reaching the endpoint, a command to reverse the spindle (e.g., from clockwise to counterclockwise).
3. Continued synchronized motion beyond the specified end coordinate until the spindle actually stops and reverses.
4. Continued synchronized motion back to the original coordinate.
5. When reaching the original coordinate, a command to reverse the spindle a second time (e.g., from counterclockwise to clockwise).
6. Continued synchronized motion beyond the original coordinate until the spindle actually stops and reverses.
7. An **unsynchronized** move back to the original coordinate.

Spindle-synchronized motions wait for spindle index, so multiple passes line up.*G33.1* moves end at the original coordinate.

All the axis words are optional, except that at least one must be used.

**G33.1 Example**

```
G90 (set absolute mode)
G0 X1.000 Y1.000 Z0.100 (rapid move to starting position)
S100 M3 (turn on the spindle, 100 RPM)
G33.1 Z-0.750 K0.05 (rigid tap a 20 TPI thread 0.750 deep)
M2 (end program)
```

- See G90 & G0 & M2 sections for more information.

It is an error if:

- All axis words are omitted.
- The spindle is not turning when this command is executed
- The requested linear motion exceeds machine velocity limits due to the spindle speed

### 3.3.23 G38.n Straight Probe

```
G38.x axes
```

- *G38.2* - probe toward workpiece, stop on contact, signal error if failure
- *G38.3* - probe toward workpiece, stop on contact
- *G38.4* - probe away from workpiece, stop on loss of contact, signal error if failure

- *G38.5* - probe away from workpiece, stop on loss of contact

**Important**
You will not be able to use a probe move until your machine has been set up to provide a probe input signal. The probe input signal must be connected to *motion.probe-input* in a .hal file. G38.x uses motion.probe-input to determine when the probe has made (or lost) contact. TRUE for probe contact closed (touching), FALSE for probe contact open.

Program *G38.x axes* to perform a straight probe operation. The axis words are optional, except that at least one of them must be used. The axis words together define the destination point that the probe will move towards, starting from the current location. If the probe is not tripped before the destination is reached G38.2 and G38.4 will signal an error.

The tool in the spindle must be a probe or contact a probe switch.

In response to this command, the machine moves the controlled point (which should be at the center of the probe ball) in a straight line at the current feed rate toward the programmed point. In inverse time feed mode, the feed rate is such that the whole motion from the current point to the programmed point would take the specified time. The move stops (within machine acceleration limits) when the programmed point is reached, or when the requested change in the probe input takes place, whichever occurs first.

After successful probing, parameters 5061 to 5069 will be set to the coordinates of X, Y, Z, A, B, C, U, V, W of the location of the controlled point at the time the probe changed state. After unsuccessful probing, they are set to the coordinates of the programmed point. Parameter 5070 is set to 1 if the probe succeeded and 0 if the probe failed. If the probing operation failed, G38.2 and G38.4 will signal an error by posting an message on screen if the selected GUI supports that. And by halting program execution.

A comment of the form *(PROBEOPEN filename.txt)* will open *filename.txt* and store the 9-number coordinate consisting of XYZABCUVW of each successful straight probe in it. The file must be closed with *(PROBECLOSE)*. For more information see the Comments Section.

An example file *smartprobe.ngc* is included (in the examples directory) to demonstrate using probe moves to log to a file the coordinates of a part. The program *smartprobe.ngc* could be used with *ngcgui* with minimal changes.

It is an error if:

- the current point is the same as the programmed point.
- no axis word is used
- cutter compensation is enabled
- the feed rate is zero
- the probe is already in the target state

### 3.3.24 G40 Compensation Off

- *G40* - turn cutter compensation off. If tool compensation was on the next move must be a linear move and longer than the tool diameter. It is OK to turn compensation off when it is already off.

**G40 Example**

```
; current location is X1 after finishing cutter compensated move
G40 (turn compensation off)
G0 X1.6 (linear move longer than current cutter diameter)
M2 (end program)
```

See G0 & M2 sections for more information.

It is an error if:

- A G2/G3 arc move is programmed next after a G40.
- The linear move after turning compensation off is less than the tool diameter.

### 3.3.25 G41, G42 Cutter Compensation

```
G41 <D-> (left of programmed path)
G42 <D-> (right of programmed path)
```

- *D* - tool number

The D word is optional; if there is no D word the radius of the currently loaded tool will be used (if no tool is loaded and no D word is given, a radius of 0 will be used).

If supplied, the D word is the tool number to use. This would normally be the number of the tool in the spindle (in which case the D word is redundant and need not be supplied), but it may be any valid tool number.

**Note**
*G41/G42 D0* is a little special. Its behavior is different on random tool changer machines and nonrandom tool changer machines (see the Tool Change section). On nonrandom tool changer machines, *G41/G42 D0* applies the TLO of the tool currently in the spindle, or a TLO of 0 if no tool is in the spindle. On random tool changer machines, *G41/G42 D0* applies the TLO of the tool T0 defined in the tool table file (or causes an error if T0 is not defined in the tool table).

To start cutter compensation to the left of the part profile, use G41. G41 starts cutter compensation to the left of the programmed line as viewed from the positive end of the axis perpendicular to the plane.

To start cutter compensation to the right of the part profile, use G42. G42 starts cutter compensation to the right of the programmed line as viewed from the positive end of the axis perpendicular to the plane.

The lead in move must be at least as long as the tool radius. The lead in move can be a rapid move.

Cutter compensation may be performed if the XY-plane or XZ-plane is active.

User M100-M199 commands are allowed when Cutter Compensation is on.

The behavior of the machining center when cutter compensation is on is described in the Cutter Compensation Section along with code examples.

It is an error if:

- The D number is not a valid tool number or 0.
- The YZ plane is active.
- Cutter compensation is commanded to turn on when it is already on.

### 3.3.26 G41.1, G42.1 Dynamic Cutter Compensation

```
G41.1 D- <L-> (left of programmed path)
G42.1 D- <L-> (right of programmed path)
```

- *D* - cutter diameter
- *L* - tool orientation (see lathe tool orientation)

G41.1 & G42.1 function the same as G41 & G42 with the added scope of being able to program the tool diameter. The L word defaults to 0 if unspecified.

It is an error if:

- The YZ plane is active.
- The L number is not in the range from 0 to 9 inclusive.
- The L number is used when the XZ plane is not active.
- Cutter compensation is commanded to turn on when it is already on.

### 3.3.27 G43 Tool Length Offset

```
G43 <H->
```

- *H* - tool number (optional)

G43 enables tool length compensation. G43 changes subsequent motions by offsetting the axis coordinates by the length of the offset. G43 does not cause any motion. The next time a compensated axis is moved, that axis's endpoint is the compensated location.

*G43* without an H word uses the currently loaded tool from the last *Tn M6*.

*G43 Hn* uses the offset for tool n.

**Note**
*G43 H0* is a little special. Its behavior is different on random tool changer machines and nonrandom tool changer machines (see the Tool Changers section). On nonrandom tool changer machines, *G43 H0* applies the TLO of the tool currently in the spindle, or a TLO of 0 if no tool is in the spindle. On random tool changer machines, *G43 H0* applies the TLO of the tool T0 defined in the tool table file (or causes an error if T0 is not defined in the tool table).

#### G43 H- Example Line

```
G43 H1 (set tool offsets using the values from tool 1 in the tool table)
```

It is an error if:

- the H number is not an integer, or
- the H number is negative, or
- the H number is not a valid tool number (though note that 0 is a valid tool number on nonrandom tool changer machines, it means "the tool currently in the spindle")

### 3.3.28 G43.1: Dynamic Tool Length Offset

```
G43.1 axes
```

- *G43.1 axes* - change subsequent motions by replacing the current offset(s) of axes. G43.1 does not cause any motion. The next time a compensated axis is moved, that axis's endpoint is the compensated location.

#### G43.1 Example

```
G90 (set absolute mode)
T1 M6 G43 (load tool 1 and tool length offsets, Z is at machine 0 and DRO shows Z1.500)
G43.1 Z0.250 (offset current tool offset by 0.250, DRO now shows Z1.250)
M2 (end program)
```

- See G90 & T & M6 sections for more information.

It is an error if:

- motion is commanded on the same line as *G43.1*

**Note**
G43.1 does not write to the tool table.

### 3.3.29 G43.2: Apply additional Tool Length Offset

```
G43.2 H-
```

- *H* - tool number

G43.2 applies an additional simultaneous tool offset.

**G43.2 Example**

```
G90 (set absolute mode)
T1 M6 (load tool 1)
G43 (or G43 H1 - replace all tool offsets with T1's offset)
G43.2 H10 (also add in T10's tool offset)
M2 (end program)
```

You can sum together an arbitrary number of offsets by calling G43.2 more times. There are no built-in assumptions about which numbers are geometry offsets and which are wear offsets, or that you should have only one of each.

Like the other G43 commands, G43.2 does not cause any motion. The next time a compensated axis is moved, that axis's endpoint is the compensated location.

It is an error if:

- *H* is unspecified, or
- the given tool number does not exist in the tool table

**Note**
G43.2 does not write to the tool table.

### 3.3.30 G49: Cancel Tool Length Compensation

- *G49* - cancels tool length compensation

It is OK to program using the same offset already in use. It is also OK to program using no tool length offset if none is currently being used.

### 3.3.31 G53 Move in Machine Coordinates

```
G53 axes
```

To move in the machine coordinate system, program *G53* on the same line as a linear move. *G53* is not modal and must be programmed on each line. *G0* or *G1* does not have to be programmed on the same line if one is currently active.

For example *G53 G0 X0 Y0 Z0* will move the axes to the home position even if the currently selected coordinate system has offsets in effect.

**G53 Example**

```
G53 G0 X0 Y0 Z0 (rapid linear move to the machine origin)
G53 X2 (rapid linear move to absolute coordinate X2)
```

- See G0 section for more information.

It is an error if:

- G53 is used without G0 or G1 being active,
- or G53 is used while cutter compensation is on.

### 3.3.32 G54-G59.3 Select Coordinate System

- *G54* - select coordinate system 1
- *G55* - select coordinate system 2
- *G56* - select coordinate system 3
- *G57* - select coordinate system 4
- *G58* - select coordinate system 5
- *G59* - select coordinate system 6
- *G59.1* - select coordinate system 7
- *G59.2* - select coordinate system 8
- *G59.3* - select coordinate system 9

The coordinate systems store the axis values and the XY rotation angle around the Z axis in the parameters shown in the following table.

Table 3.9: Coordinate System Parameters

| Select | CS | X | Y | Z | A | B | C | U | V | W | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| G54 | 1 | 5221 | 5222 | 5223 | 5224 | 5225 | 5226 | 5227 | 5228 | 5229 | 5230 |
| G55 | 2 | 5241 | 5242 | 5243 | 5244 | 5245 | 5246 | 5247 | 5248 | 5249 | 5250 |
| G56 | 3 | 5261 | 5262 | 5263 | 5264 | 5265 | 5266 | 5267 | 5268 | 5269 | 5270 |
| G57 | 4 | 5281 | 5282 | 5283 | 5284 | 5285 | 5286 | 5287 | 5288 | 5289 | 5290 |
| G58 | 5 | 5301 | 5302 | 5303 | 5304 | 5305 | 5306 | 5307 | 5308 | 5309 | 5310 |
| G59 | 6 | 5321 | 5322 | 5323 | 5324 | 5325 | 5326 | 5327 | 5328 | 5329 | 5330 |
| G59.1 | 7 | 5341 | 5342 | 5343 | 5344 | 5345 | 5346 | 5347 | 5348 | 5349 | 5350 |
| G59.2 | 8 | 5361 | 5362 | 5363 | 5364 | 5365 | 5366 | 5367 | 5368 | 5369 | 5370 |
| G59.3 | 9 | 5381 | 5382 | 5383 | 5384 | 5385 | 5386 | 5387 | 5388 | 5389 | 5390 |

It is an error if:

- selecting a coordinate system is used while cutter compensation is on.

See the Coordinate System Section for an overview of coordinate systems.

### 3.3.33 G61, G61.1 Exact Path Mode

- *G61* - Exact path mode, movement exactly as programed. Moves will slow or stop as needed to reach every programed point. If two sequential moves are exactly co-linear movement will not stop.
- *G61.1* - Exact stop mode, movement will stop at the end of each programed segment.

### 3.3.34 G64 Path Blending

```
G64 <P- <Q->>
```

- *P* - motion blending tolerance

- *Q* - naive cam tolerance
- *G64* - best possible speed.
- *G64 P- <Q- >* blending with tolerance.
- *G64* - without P means to keep the best speed possible, no matter how far away from the programmed point you end up.
- *G64 P- Q-* - is a way to fine tune your system for best compromise between speed and accuracy. The P- tolerance means that the actual path will be no more than P- away from the programmed endpoint. The velocity will be reduced if needed to maintain the path. In addition, when you activate G64 P- Q- it turns on the *naive cam detector*; when there are a series of linear XYZ feed moves at the same feed rate that are less than Q- away from being collinear, they are collapsed into a single linear move. On G2/G3 moves in the G17 (XY) plane when the maximum deviation of an arc from a straight line is less than the G64 P- tolerance the arc is broken into two lines (from start of arc to midpoint, and from midpoint to end). those lines are then subject to the naive cam algorithm for lines. Thus, line-arc, arc-arc, and arc-line cases as well as line-line benefit from the *naive cam detector*. This improves contouring performance by simplifying the path. It is OK to program for the mode that is already active. See also the Trajectory Control Section for more information on these modes. If Q is not specified then it will have the same behavior as before and use the value of P-.

**G64 P- Example Line**

```
G64 P0.015 (set path following to be within 0.015 of the actual path)
```

It is a good idea to include a path control specification in the preamble of each G code file.

### 3.3.35 G73 Drilling Cycle with Chip Breaking

```
G73 X- Y- Z- R- Q- <L->
```

- *R* - retract position along the Z axis.
- *Q* - delta increment along the Z axis.
- *L* - repeat

The *G73* cycle is drilling or milling with chip breaking. This cycle takes a Q number which represents a *delta* increment along the Z axis.

1. Preliminary motion.
   - If the current Z position is below the R position, The Z axis does a rapid move to the R position.
   - Move to the X Y coordinates
2. Move the Z-axis only at the current feed rate downward by delta or to the Z position, whichever is less deep.
3. Rapid up a bit.
4. Repeat steps 2 and 3 until the Z position is reached at step 2.
5. The Z axis does a rapid move to the R position.

It is an error if:

- the Q number is negative or zero.
- the R number is not specified

### 3.3.36 G76 Threading Cycle

```
G76 P- Z- I- J- R- K- Q- H- E- L-
```

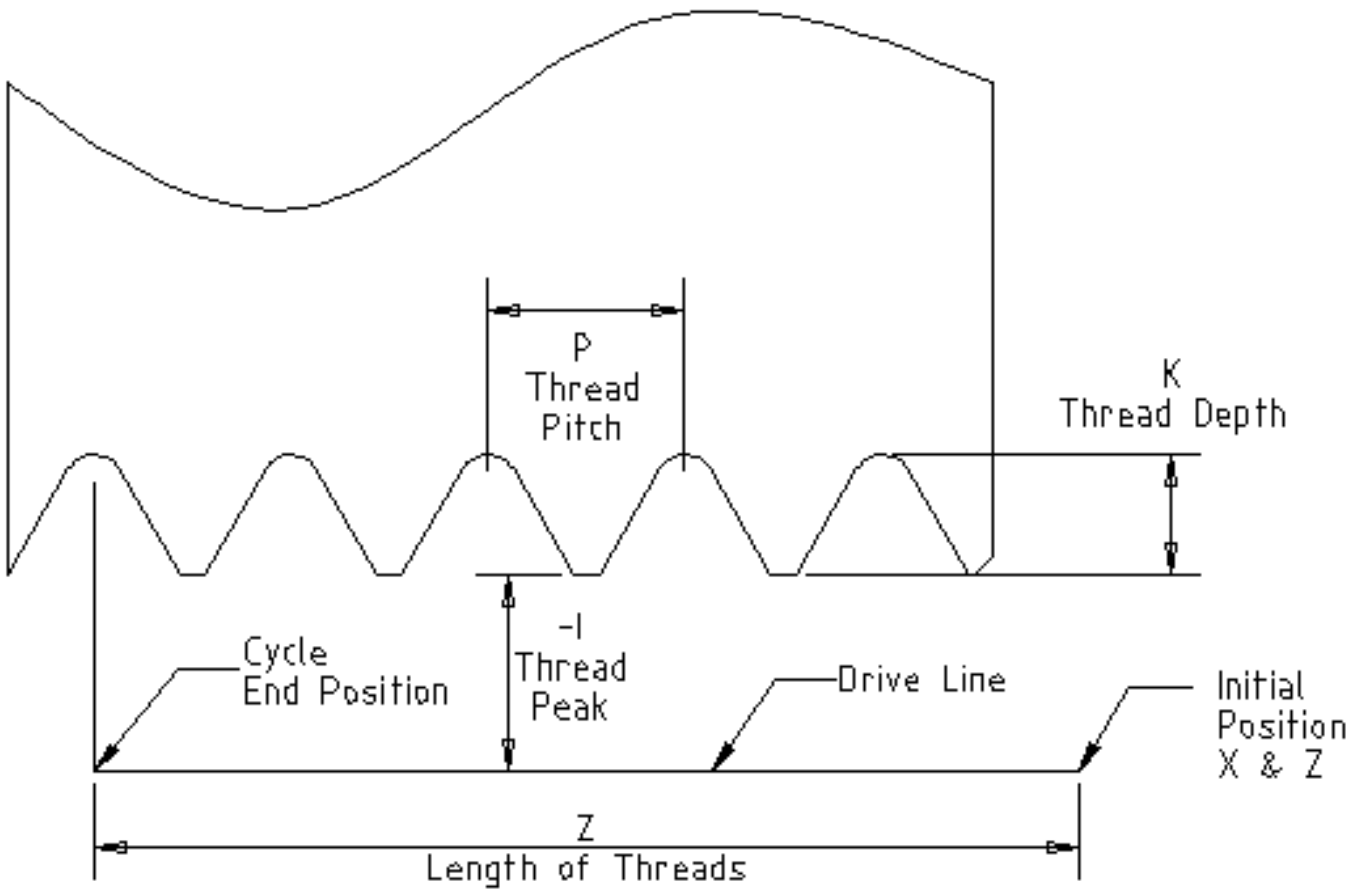



Figure 3.6: G76 Threading

- *Drive Line* - A line through the initial X position parallel to the Z.
- *P-* - The *thread pitch* in distance per revolution.
- *Z-* - The final position of threads. At the end of the cycle the tool will be at this Z position.

---

**Note**
When G7 *Lathe Diameter Mode* is in force the values for *I*, *J* and *K* are diameter measurements. When G8 *Lathe Radius Mode* is in force the values for *I*, *J* and *K* are radius measurements.

---

- *I-* - The *thread peak* offset from the *drive line*. Negative *I* values are external threads, and positive *I* values are internal threads. Generally the material has been turned to this size before the *G76* cycle.
- *J-* - A positive value specifying the *initial cut depth*. The first threading cut will be *J* beyond the *thread peak* position.
- *K-* - A positive value specifying the *full thread depth*. The final threading cut will be *K* beyond the *thread peak* position.

Optional settings

- *R*- - The *depth degression*. *R1.0* selects constant depth on successive threading passes. *R2.0* selects constant area. Values between 1.0 and 2.0 select decreasing depth but increasing area. Values above 2.0 select decreasing area. Beware that unnecessarily high degression values will cause a large number of passes to be used. (degression = a descent by stages or steps.)
- *Q*- - The *compound slide angle* is the angle (in degrees) describing to what extent successive passes should be offset along the drive line. This is used to cause one side of the tool to remove more material than the other. A positive *Q* value causes the leading edge of the tool to cut more heavily. Typical values are 29, 29.5 or 30.
- *H*- - The number of *spring passes*. Spring passes are additional passes at full thread depth. If no additional passes are desired, program *H0*.
- *E*- - Specifies the distance along the drive line used for the taper. The angle of the taper will be so the last pass tapers to the thread crest over the distance specified with E.' E0.2' will give a taper for the first/last 0.2 length units along the thread. For a 45 degree taper program E the same as K
- *L*- - Specifies which ends of the thread get the taper. Program *L0* for no taper (the default), *L1* for entry taper, *L2* for exit taper, or *L3* for both entry and exit tapers. Entry tapers will pause at the drive line to synchronize with the index pulse then move at the feed rate in to the beginning of the taper. No entry taper and the tool will rapid to the cut depth then synchronize and begin the cut.

The tool is moved to the initial X and Z positions prior to issuing the G76. The X position is the *drive line* and the Z position is the start of the threads.

The tool will pause briefly for synchronization before each threading pass, so a relief groove will be required at the entry unless the beginning of the thread is past the end of the material or an entry taper is used.

Unless using an exit taper, the exit move is not synchronized to the spindle speed and will be a rapid move. With a slow spindle, the exit move might take only a small fraction of a revolution. If the spindle speed is increased after several passes are complete, subsequent exit moves will require a larger portion of a revolution, resulting in a very heavy cut during the exit move. This can be avoided by providing a relief groove at the exit, or by not changing the spindle speed while threading.

The final position of the tool will be at the end of the *drive line*. A safe Z move will be needed with an internal thread to remove the tool from the hole.

It is an error if:

- The active plane is not the ZX plane
- Other axis words, such as X- or Y-, are specified
- The *R*- degression value is less than 1.0.
- All the required words are not specified
- *P*-, *J*-, *K*- or *H*- is negative
- *E*- is greater than half the drive line length

**HAL Connections** The pins *motion.spindle-at-speed* and the *encoder.n.phase-Z* for the spindle must be connected in your HAL file before G76 will work. See the spindle pins in the Motion section for more information.

**Technical Info** The G76 canned cycle is based on the G33 Spindle Synchronized Motion. For more information see the G33 Technical Info.

The sample program *g76.ngc* shows the use of the G76 canned cycle, and can be previewed and executed on any machine using the *sim/lathe.ini* configuration.

**G76 Example**

```
G0 Z-0.5 X0.2
G76 P0.05 Z-1 I-.075 J0.008 K0.045 Q29.5 L2 E0.045
```

In the figure the tool is in the final position after the G76 cycle is completed. You can see the entry path on the right from the Q29.5 and the exit path on the left from the L2 E0.045. The white lines are the cutting moves.

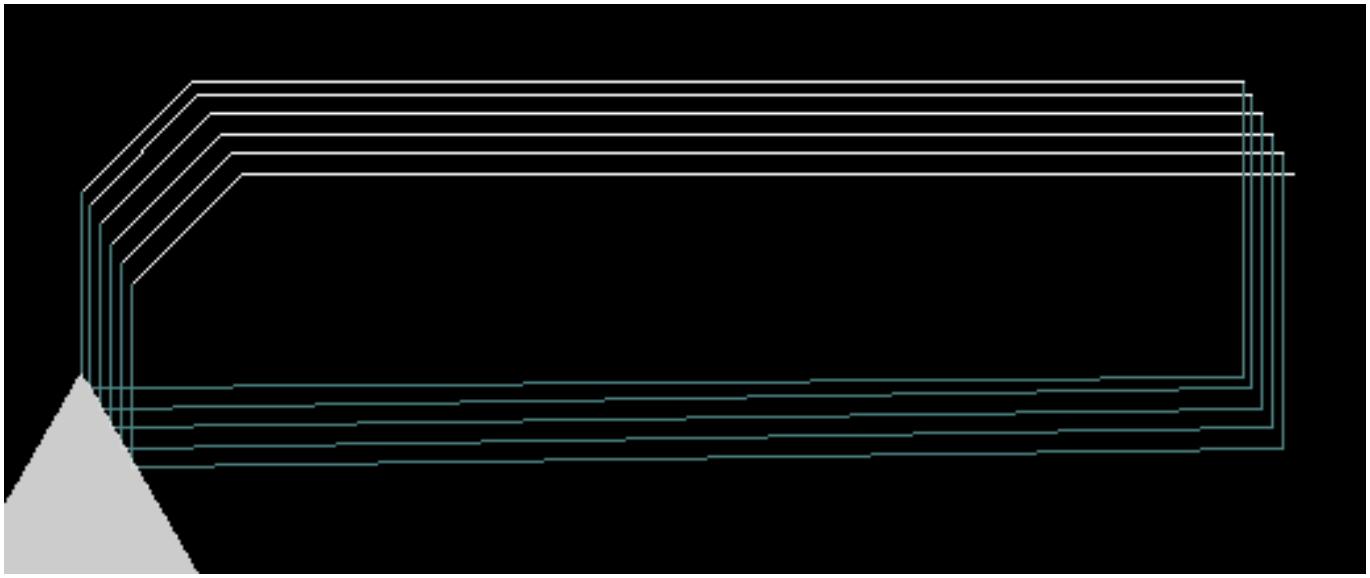


Figure 3.7: G76 Example

### 3.3.37 Canned Cycles

The canned cycles *G81* through *G89* and the canned cycle stop *G80* are described in this section.

All canned cycles are performed with respect to the currently-selected plane. Any of the nine planes may be selected. Throughout this section, most of the descriptions assume the XY-plane has been selected. The behavior is analogous if another plane is selected, and the correct words must be used. For instance, in the *G17.1* plane, the action of the canned cycle is along W, and the locations or increments are given with U and V. In this case substitute U,V,W for X,Y,Z in the instructions below.

Rotary axis words are not allowed in canned cycles. When the active plane is one of the XYZ family, the UVW axis words are not allowed. Likewise, when the active plane is one of the UVW family, the XYZ axis words are not allowed.

#### 3.3.37.1 Common Words

All canned cycles use X, Y, Z, or U, V, W groups depending on the plane selected and R words. The R (usually meaning retract) position is along the axis perpendicular to the currently selected plane (Z-axis for XY-plane, etc.) Some canned cycles use additional arguments.

#### 3.3.37.2 Sticky Words

For canned cycles, we will call a number *sticky* if, when the same cycle is used on several lines of code in a row, the number must be used the first time, but is optional on the rest of the lines. Sticky numbers keep their value on the rest of the lines if they are not explicitly programmed to be different. The R number is always sticky.

In incremental distance mode X, Y, and R numbers are treated as increments from the current position and Z as an increment from the Z-axis position before the move involving Z takes place. In absolute distance mode, the X, Y, R, and Z numbers are absolute positions in the current coordinate system.

#### 3.3.37.3 Repeat Cycle

The L number is optional and represents the number of repeats. L=0 is not allowed. If the repeat feature is used, it is normally used in incremental distance mode, so that the same sequence of motions is repeated in several equally spaced places along a straight line. When L- is greater than 1 in incremental mode with the XY-plane selected, the X and Y positions are determined by adding the given X and Y numbers either to the current X and Y positions (on the first go-around) or to the X and Y positions at the end of the previous go-around (on the repetitions). Thus, if you program *L10* , you will get 10 cycles. The first cycle will be distance X,Y from the original location. The R and Z positions do not change during the repeats. The L number is not sticky. In absolute distance mode, L>1 means *do the same cycle in the same place several times*, Omitting the L word is equivalent to specifying L=1.

#### 3.3.37.4 Retract Mode

The height of the retract move at the end of each repeat (called *clear Z* in the descriptions below) is determined by the setting of the retract mode: either to the original Z position (if that is above the R position and the retract mode is *G98*, OLD_Z), or otherwise to the R position. See the G98 G99 Section.

#### 3.3.37.5 Canned Cycle Errors

It is an error if:

- axis words are all missing during a canned cycle,
- axis words from different groups (XYZ) (UVW) are used together,
- a P number is required and a negative P number is used,
- an L number is used that does not evaluate to a positive integer,
- rotary axis motion is used during a canned cycle,
- inverse time feed rate is active during a canned cycle,
- or cutter compensation is active during a canned cycle.

If the XY plane is active, the Z number is sticky, and it is an error if:

- the Z number is missing and the same canned cycle was not already active,
- or the R number is less than the Z number.

If other planes are active, the error conditions are analogous to the XY conditions above.

#### 3.3.37.6 Preliminary and In-Between Motion

Preliminary motion is a set of motions that is common to all of the milling canned cycles. If the current Z position is below the R position, the Z axis does a rapid move to the R position. This happens only once, regardless of the value of L.

In addition, at the beginning of the first cycle and each repeat, the following one or two moves are made

1. A rapid move parallel to the XY-plane to the given XY-position,
2. The Z-axis make a rapid move to the R position, if it is not already at the R position.

If another plane is active, the preliminary and in-between motions are analogous.

#### 3.3.37.7 Why use a canned cycle?

There are at least two reasons for using canned cycles. The first is the economy of code. A single bore would take several lines of code to execute.

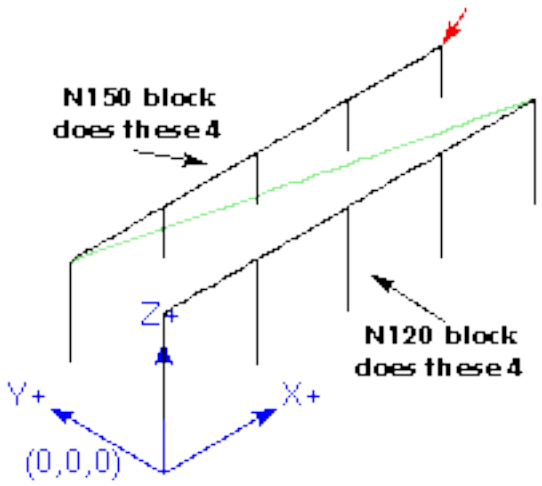
The G81 Example 1 demonstrates how a canned cycle could be used to produce 8 holes with ten lines of G code within the canned cycle mode. The program below will produce the same set of 8 holes using five lines for the canned cycle. It does not follow exactly the same path nor does it drill in the same order as the earlier example. But the program writing economy of a good canned cycle should be obvious.

**Note**
Line numbers are not needed but help clarify these examples

**Eight Holes**

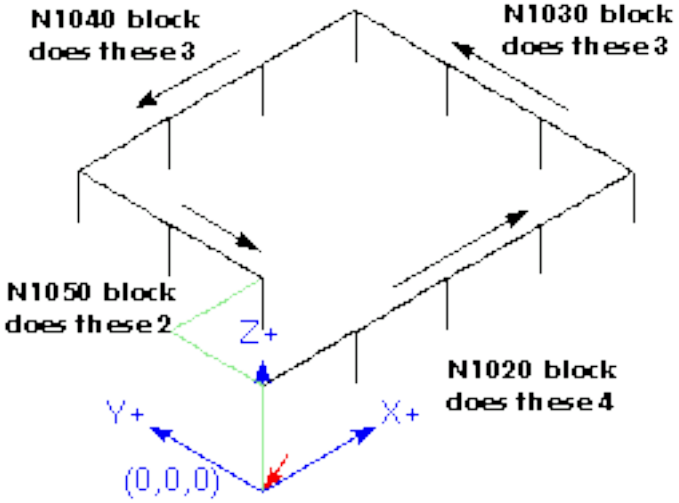
```
N100 G90 G0 X0 Y0 Z0 (move coordinate home)
N110 G1 F10 X0 G4 P0.1
N120 G91 G81 X1 Y0 Z-1 R1 L4(canned drill cycle)
N130 G90 G0 X0 Y1
N140 Z0
N150 G91 G81 X1 Y0 Z-0.5 R1 L4(canned drill cycle)
N160 G80 (turn off canned cycle)
N170 M2 (program end)
```

The G98 to the second line above means that the return move will be to the value of Z in the first line since it is higher that the R value specified.




**Twelve Holes in a Square** This example demonstrates the use of the L word to repeat a set of incremental drill cycles for successive blocks of code within the same G81 motion mode. Here we produce 12 holes using five lines of code in the canned motion mode.

```
N1000 G90 G0 X0 Y0 Z0 (move coordinate home)
N1010 G1 F50 X0 G4 P0.1
N1020 G91 G81 X1 Y0 Z-0.5 R1 L4 (canned drill cycle)
N1030 X0 Y1 R0 L3 (repeat)
N1040 X-1 Y0 L3 (repeat)
N1050 X0 Y-1 L2 (repeat)
N1060 G80 (turn off canned cycle)
N1070 G90 G0 X0 (rapid move home)
N1080 Y0
N1090 Z0
N1100 M2 (program end)
```




The second reason to use a canned cycle is that they all produce preliminary moves and returns that you can anticipate and control regardless of the start point of the canned cycle.

### 3.3.38 G80 Cancel Canned Cycle

- *G80* - cancel canned cycle modal motion. *G80* is part of modal group 1, so programming any other G code from modal group 1 will also cancel the canned cycle.

It is an error if:

- Axis words are programmed when G80 is active.

**G80 Example**

```
G90 G81 X1 Y1 Z1.5 R2.8 (absolute distance canned cycle)
G80 (turn off canned cycle motion)
G0 X0 Y0 Z0 (rapid move to coordinate home)
```

The following code produces the same final position and machine state as the previous code.

**G0 Example**

```
G90 G81 X1 Y1 Z1.5 R2.8 (absolute distance canned cycle)
G0 X0 Y0 Z0 (rapid move to coordinate home)
```

The advantage of the first set is that, the G80 line clearly turns off the G81 canned cycle. With the first set of blocks, the programmer must turn motion back on with G0, as is done in the next line, or any other motion mode G word.

If a canned cycle is not turned off with G80 or another motion word, the canned cycle will attempt to repeat itself using the next block of code that contains an X, Y, or Z word. The following file drills (G81) a set of eight holes as shown in the following caption.

**G80 Example 1**

```
N100 G90 G0 X0 Y0 Z0 (coordinate home)
N110 G1 X0 G4 P0.1
N120 G81 X1 Y0 Z0 R1 (canned drill cycle)
N130 X2
N140 X3
N150 X4
N160 Y1 Z0.5
```

```
N170 X3
N180 X2
N190 X1
N200 G80 (turn off canned cycle)
N210 G0 X0 (rapid move home)
N220 Y0
N230 Z0
N240 M2 (program end)
```

**Note**
Notice the z position change after the first four holes. Also, this is one of the few places where line numbers have some value, being able to point a reader to a specific line of code.

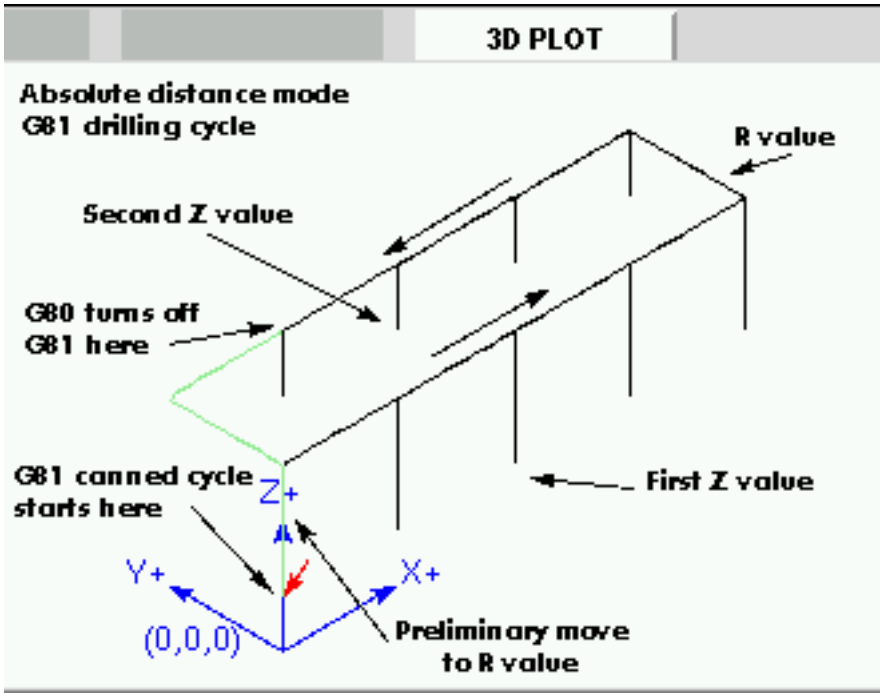



Figure 3.8: G80 Cycle

The use of G80 in line N200 is optional because the G0 on the next line will turn off the G81 cycle. But using the G80 as shown in Example 1, will provide for easier to read canned cycle. Without it, it is not so obvious that all of the blocks between N120 and N200 belong to the canned cycle.

### 3.3.39 G81 Drilling Cycle

```
G81 (X- Y- Z-) or (U- V- W-) R- L-
```

The *G81* cycle is intended for drilling.

The cycle functions as follows:

1. Preliminary motion, as described in the Preliminary and In-Between Motion section.
2. Move the Z-axis at the current feed rate to the Z position.

3. The Z-axis does a rapid move to clear Z.

**Example 1 - Absolute Position G81** Suppose the current position is (X1, Y2, Z3) and the following line of NC code is interpreted.

```
G90 G98 G81 X4 Y5 Z1.5 R2.8
```

This calls for absolute distance mode (G90) and OLD_Z retract mode (G98) and calls for the G81 drilling cycle to be performed once.

The X value and X position are 4.

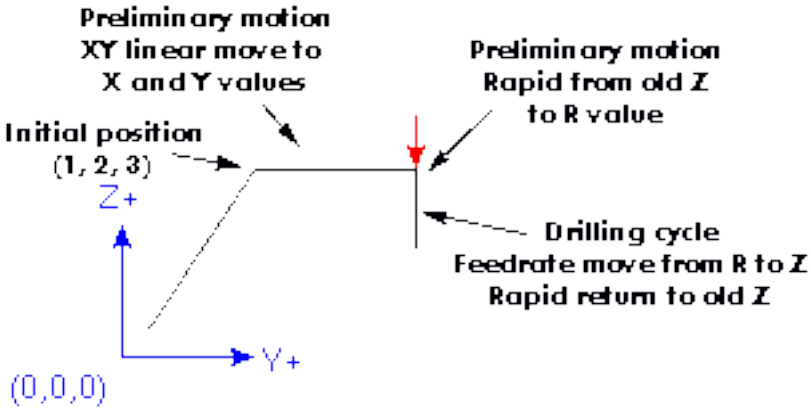
The Y value and Y position are 5.

The Z value and Z position are 1.5.

The R value and clear Z are 2.8. OLD_Z is 3.

The following moves take place:

1. a rapid move parallel to the XY plane to (X4, Y5)
2. a rapid move move parallel to the Z-axis to (Z2.8).
3. move parallel to the Z-axis at the feed rate to (Z1.5)
4. a rapid move parallel to the Z-axis to (Z3)




**Example 2 - Relative Position G81** Suppose the current position is (X1, Y2, Z3) and the following line of NC code is interpreted.

```
G91 G98 G81 X4 Y5 Z-0.6 R1.8 L3
```

This calls for incremental distance mode (G91) and OLD_Z retract mode (G98). It also calls for the G81 drilling cycle to be repeated three times. The X value is 4, the Y value is 5, the Z value is -0.6 and the R value is 1.8. The initial X position is 5 (=1+4), the initial Y position is 7 (=2+5), the clear Z position is 4.8 (=1.8+3), and the Z position is 4.2 (=4.8-0.6). OLD_Z is 3.

The first preliminary move is a maximum rapid move along the Z axis to (X1,Y2,Z4.8), since OLD_Z < clear Z.

The first repeat consists of 3 moves.

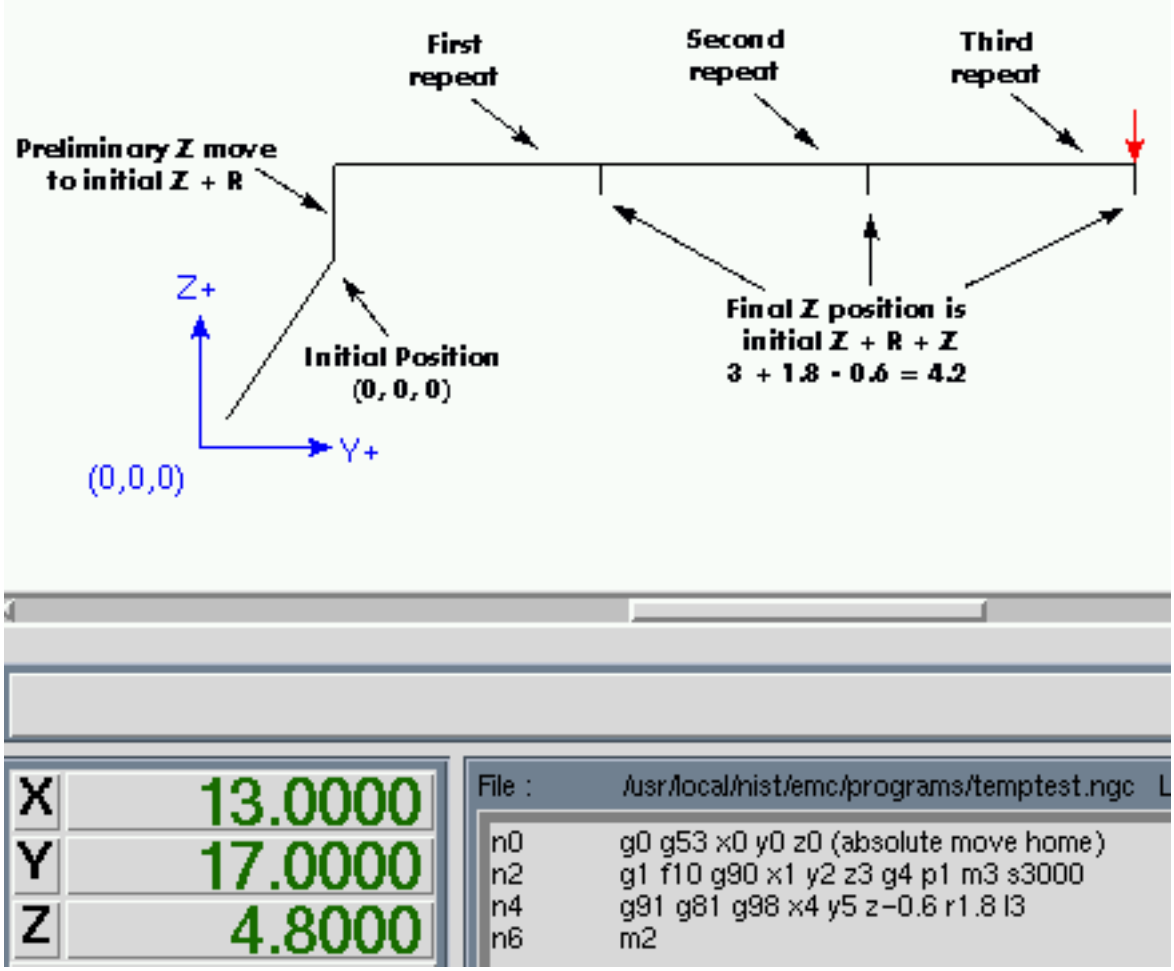
1. a rapid move parallel to the XY-plane to (X5, Y7)
2. move parallel to the Z-axis at the feed rate to (Z4.2)
3. a rapid move parallel to the Z-axis to (X5, Y7, Z4.8)

The second repeat consists of 3 moves. The X position is reset to 9 (=5+4) and the Y position to 12 (=7+5).

1. a rapid move parallel to the XY-plane to (X9, Y12, Z4.8)
2. move parallel to the Z-axis at the feed rate to (X9, Y12, Z4.2)
3. a rapid move parallel to the Z-axis to (X9, Y12, Z4.8)

The third repeat consists of 3 moves. The X position is reset to 13 (=9+4) and the Y position to 17 (=12+5).

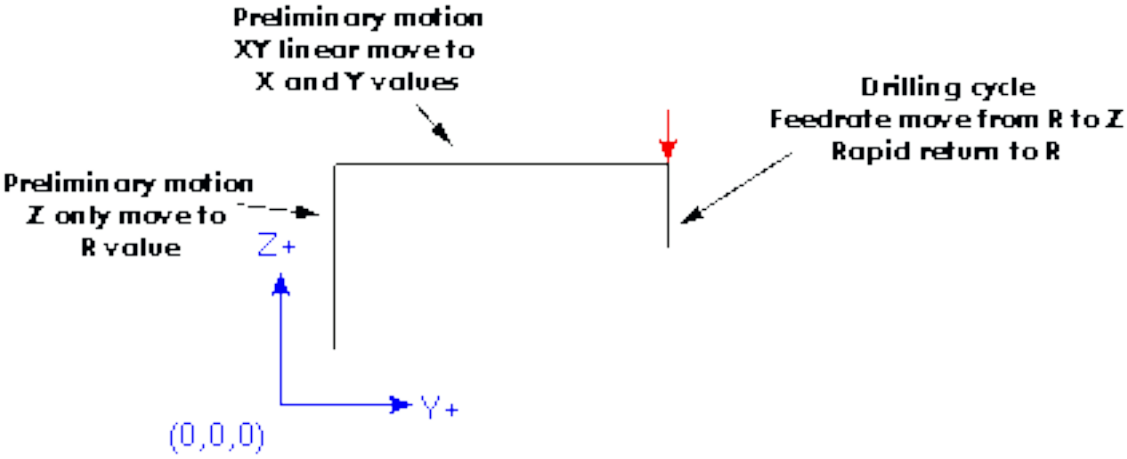
1. a rapid move parallel to the XY-plane to (X13, Y17, Z4.8)
2. move parallel to the Z-axis at the feed rate to (X13, Y17, Z4.2)
3. a rapid move parallel to the Z-axis to (X13, Y17, Z4.8)




**Example 3 - Relative Position G81** Now suppose that you execute the first G81 block of code but from (X0, Y0, Z0) rather than from (X1, Y2, Z3).

```
G90 G98 G81 X4 Y5 Z1.5 R2.8
```

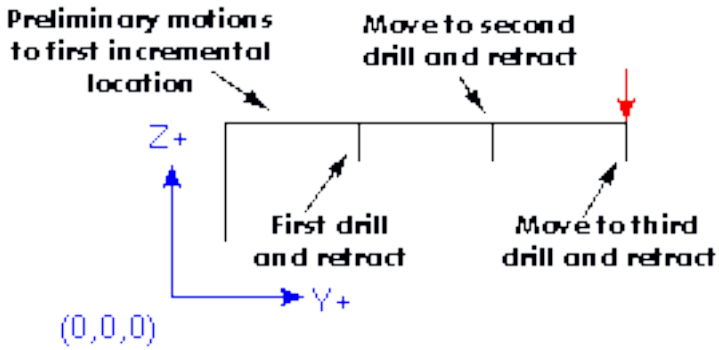
Since OLD_Z is below the R value, it adds nothing for the motion but since the initial value of Z is less than the value specified in R, there will be an initial Z move during the preliminary moves.




**Example 4 - Absolute G81 R > Z** This is a plot of the path of motion for the second g81 block of code.

```
G91 G98 G81 X4 Y5 Z-0.6 R1.8 L3
```

Since this plot starts with (X0, Y0, Z0), the interpreter adds the initial Z0 and R1.8 and rapid moves to that location. After that initial Z move, the repeat feature works the same as it did in example 3 with the final Z depth being 0.6 below the R value.




**Example 5 - Relative position R > Z**

```
G90 G98 G81 X4 Y5 Z-0.6 R1.8
```

Since this plot starts with (X0, Y0, Z0), the interpreter adds the initial Z0 and R1.8 and rapid moves to that location as in *Example 4*. After that initial Z move, the rapid move to X4 Y5 is done. Then the final Z depth being 0.6 below the R value. The repeat function would make the Z move in the same location again.

### 3.3.40 G82 Drilling Cycle, Dwell

```
G82 (X- Y- Z-) or (U- V- W-) R- L- P-
```

The *G82* cycle is intended for drilling with a dwell at the bottom of the hole.

1. Preliminary motion, as described in the Preliminary and In-Between Motion section.
2. Move the Z-axis at the current feed rate to the Z position.

3. Dwell for the P number of seconds.
4. The Z-axis does a rapid move to clear Z.

The motion of a G82 canned cycle looks just like G81 with the addition of a dwell at the bottom of the Z move. The length of the dwell is specified by a *P*- word in the G82 block.

### 3.3.41 G83 Peck Drilling Cycle

```
G83 (X- Y- Z-) or (U- V- W-) R- L- Q-
```

The *G83* cycle (often called peck drilling) is intended for deep drilling or milling with chip breaking. The retracts in this cycle clear the hole of chips and cut off any long stringers (which are common when drilling in aluminum). This cycle takes a Q number which represents a *delta* increment along the Z-axis. The retract before final depth will always be to the *retract* plane even if G98 is in effect. The final retract will honor the G98/99 in effect. G83 functions the same as G81 with the addition of retracts during the drilling operation.

1. Preliminary motion, as described in the Preliminary and In-Between Motion section.
2. Move the Z-axis at the current feed rate downward by delta or to the Z position, whichever is less deep.
3. Rapid move back out to the retract plane specified by the R word.
4. Rapid move back down to the current hole bottom, backed off a bit.
5. Repeat steps 2, 3, and 4 until the Z position is reached at step 2.
6. The Z-axis does a rapid move to clear Z.

It is an error if:

- the Q number is negative or zero.

### 3.3.42 G84 Right-Hand Tapping Cycle

This code is currently unimplemented in LinuxCNC. It is accepted, but the behavior is undefined. See section G33.1

### 3.3.43 G85 Boring Cycle, Feed Out

```
G85 (X- Y- Z-) or (U- V- W-) R- L-
```

The *G85* cycle is intended for boring or reaming, but could be used for drilling or milling.

1. Preliminary motion, as described in the Preliminary and In-Between Motion section.
2. Move the Z-axis only at the current feed rate to the Z position.
3. Retract the Z-axis at the current feed rate to the R plane if it is lower than the initial Z.
4. Retract at the traverse rate to clear Z.

### 3.3.44 G86 Boring Cycle, Spindle Stop, Rapid Move Out

```
G86 (X- Y- Z-) or (U- V- W-) R- L- P-
```

The *G86* cycle is intended for boring. This cycle uses a P number for the number of seconds to dwell.

1. Preliminary motion, as described in the Preliminary and In-Between Motion section.
2. Move the Z-axis only at the current feed rate to the Z position.
3. Dwell for the P number of seconds.
4. Stop the spindle turning.
5. The Z-axis does a rapid move to clear Z.
6. Restart the spindle in the direction it was going.

It is an error if:

- the spindle is not turning before this cycle is executed.

### 3.3.45 G87 Back Boring Cycle

This code is currently unimplemented in LinuxCNC. It is accepted, but the behavior is undefined.

### 3.3.46 G88 Boring Cycle, Spindle Stop, Manual Out

This code is currently unimplemented in LinuxCNC. It is accepted, but the behavior is undefined.

### 3.3.47 G89 Boring Cycle, Dwell, Feed Out

```
G89 (X- Y- Z-) or (U- V- W-) R- L- P-
```

The *G89* cycle is intended for boring. This cycle uses a P number, where P specifies the number of seconds to dwell.

1. Preliminary motion, as described in the Preliminary and In-Between Motion section.
2. Move the Z-axis only at the current feed rate to the Z position.
3. Dwell for the P number of seconds.
4. Retract the Z-axis at the current feed rate to clear Z.

### 3.3.48 G90, G91 Distance Mode

- *G90* - absolute distance mode In absolute distance mode, axis numbers (X, Y, Z, A, B, C, U, V, W) usually represent positions in terms of the currently active coordinate system. Any exceptions to that rule are described explicitly in the G80 G89 Section.
- *G91* - incremental distance mode In incremental distance mode, axis numbers usually represent increments from the current coordinate.

**G90 Example**

```
G90 (set absolute distance mode)
G0 X2.5 (rapid move to coordinate X2.5 including any offsets in effect)
```

**G91 Example**

```
G91 (set incremental distance mode)
G0 X2.5 (rapid move 2.5 from current position along the X axis)
```

- See G0 section for more information.

### 3.3.49 G90.1, G91.1 Arc Distance Mode

- *G90.1* - absolute distance mode for I, J & K offsets. When G90.1 is in effect I and J both must be specified with G2/3 for the XY plane or J and K for the XZ plane or it is an error.
- *G91.1* - incremental distance mode for I, J & K offsets. G91.1 Returns I, J & K to their default behavior.

### 3.3.50 G92 Coordinate System Offset

```
G92 axes
```

G92 makes the current point have the coordinates you want (without motion), where the axis words contain the axis numbers you want. All axis words are optional, except that at least one must be used. If an axis word is not used for a given axis, the coordinate on that axis of the current point is not changed.

When *G92* is executed, the origins of all coordinate systems move. They move such that the value of the current controlled point, in the currently active coordinate system, becomes the specified value. All coordinate system's origins are offset this same distance.

For example, suppose the current point is at X=4 and there is currently no *G92* offset active. Then *G92 x7* is programmed. This moves all origins -3 in X, which causes the current point to become X=7. This -3 is saved in parameter 5211.

Being in incremental distance mode has no effect on the action of *G92*.

*G92* offsets may be already be in effect when the *G92* is called. If this is the case, the offset is replaced with a new offset that makes the current point become the specified value.

It is an error if:

- all axis words are omitted.

LinuxCNC stores the G92 offsets and reuses them on the next run of a program. To prevent this, one can program a G92.1 (to erase them), or program a G92.2 (to remove them - they are still stored).

See the Coordinate System Section for an overview of coordinate systems.

See the Offsets Section for more information.

See the Parameters Section for more information.

### 3.3.51 G92.1, G92.2 Reset G92 Offsets

- *G92.1* - reset G92 offsets to zero and set parameters 5211 - 5219 to zero.
- *G92.2* - reset G92 offsets to zero.

**Note**
G92.1 only clears G92 offsets, to change G53-G59.3 coordinate system offsets in G code use either G10 L2 or G10 L20.

### 3.3.52 G92.3 Restore G92 Offsets

- *G92.3* - set the G92 offset to the values saved in parameters 5211 to 5219

You can set axis offsets in one program and use the same offsets in another program. Program *G92* in the first program. This will set parameters 5211 to 5219. Do not use *G92.1* in the remainder of the first program. The parameter values will be saved when the first program exits and restored when the second one starts up. Use *G92.3* near the beginning of the second program. That will restore the offsets saved in the first program.

### 3.3.53 G93, G94, G95: Feed Rate Mode

- *G93* - is Inverse Time Mode. In inverse time feed rate mode, an F word means the move should be completed in [one divided by the F number] minutes. For example, if the F number is 2.0, the move should be completed in half a minute.

  When the inverse time feed rate mode is active, an F word must appear on every line which has a G1, G2, or G3 motion, and an F word on a line that does not have G1, G2, or G3 is ignored. Being in inverse time feed rate mode does not affect G0 (rapid move) motions.
- *G94* - is Units per Minute Mode. In units per minute feed mode, an F word is interpreted to mean the controlled point should move at a certain number of inches per minute, millimeters per minute, or degrees per minute, depending upon what length units are being used and which axis or axes are moving.
- *G95* - is Units per Revolution Mode In units per revolution mode, an F word is interpreted to mean the controlled point should move a certain number of inches per revolution of the spindle, depending on what length units are being used and which axis or axes are moving. G95 is not suitable for threading, for threading use G33 or G76. G95 requires that motion.spindle-speed-in to be connected.

It is an error if:

- Inverse time feed mode is active and a line with G1, G2, or G3 (explicitly or implicitly) does not have an F word.
- A new feed rate is not specified after switching to G94 or G95

### 3.3.54 G96, G97 Spindle Control Mode

```
G96 <D-> S- (Constant Surface Speed Mode)
G97 (RPM Mode)
```

- *D* - maximum spindle RPM
- *S* - surface speed
- *G96 D- S-* - selects constant surface speed of *S* feet per minute (if G20 is in effect) or meters per minute (if G21 is in effect). D- is optional.

  When using G96, ensure that X0 in the current coordinate system (including offsets and tool lengths) is the center of rotation or LinuxCNC will not give the desired spindle speed. G96 is not affected by radius or diameter mode.
- *G97* selects RPM mode.

**G96 Example Line**

```
G96 D2500 S250 (set CSS with a max rpm of 2500 and a surface speed of 250)
```

It is an error if:

- S is not specified with G96
- A feed move is specified in G96 mode while the spindle is not turning

### 3.3.55 G98, G99 Canned Cycle Return Level

- *G98* - retract to the position that axis was in just before this series of one or more contiguous canned cycles was started.
- *G99* - retract to the position specified by the R word of the canned cycle.

Program a *G98* and the canned cycle will use the Z position prior to the canned cycle as the Z return position if it is higher than the R value specified in the cycle. If it is lower, the R value will be used. The R word has different meanings in absolute distance mode and incremental distance mode.

**G98 Retract to Origin**

```
G0 X1 Y2 Z3
G90 G98 G81 X4 Y5 Z-0.6 R1.8 F10
```

The G98 to the second line above means that the return move will be to the value of Z in the first line since it is higher that the R value specified.

The *initial* (G98) plane is reset any time cycle motion mode is abandoned, whether explicitly (G80) or implicitly (any motion code that is not a cycle). Switching among cycle modes (say G81 to G83) does NOT reset the *initial* plane. It is possible to switch between G98 and G99 during a series of cycles.

## 3.4 M Codes

### 3.4.1 M Code Quick Reference Table

| Code | Description |
|---|---|
| M0 M1 | Program Pause |
| M2 M30 | Program End |
| M60 | Pallet Change Pause |
| M3 M4 M5 | Spindle Control |
| M6 | Tool Change |
| M7 M8 M9 | Coolant Control |
| M19 | Orient Spindle |
| M48 M49 | Feed & Spindle Overrides Enable/Disable |
| M50 | Feed Override Control |
| M51 | Spindle Override Control |
| M52 | Adaptive Feed Control |
| M53 | Feed Stop Control |
| M61 | Set Current Tool Number |
| m62-m65 | Output Control |
| M66 | Input Control |
| M67 | Analog Output Control |
| M68 | Analog Output Control |
| M70 | Save Modal State |
| M71 | Invalidate Stored Modal State |
| M72 | Restore Modal State |
| M73 | Save Autorestore Modal State |
| M100-M199 | User Defined M-Codes |

### 3.4.2 M0, M1 Program Pause

- *M0* - pause a running program temporarily. LinuxCNC remains in the Auto Mode so MDI and other manual actions are not enabled. Pressing the cycle start button will restart the program at the following line.

- *M1* - pause a running program temporarily if the optional stop switch is on. LinuxCNC remains in the Auto Mode so MDI and other manual actions are not enabled. Pressing the cycle start button will restart the program at the following line.

**Note**
It is OK to program *M0* and *M1* in MDI mode, but the effect will probably not be noticeable, because normal behavior in MDI mode is to stop after each line of input anyway.

### 3.4.3 M2, M30 Program End

- *M2* - end the program. Pressing cycle start will start the program at the beginning of the file.
- *M30* - exchange pallet shuttles and end the program. Pressing cycle start will start the program at the beginning of the file.

Both of these commands have the following effects:

1. Change from Auto mode to MDI mode.
2. Origin offsets are set to the default (like *G54*).
3. Selected plane is set to XY plane (like *G17*).
4. Distance mode is set to absolute mode (like *G90*).
5. Feed rate mode is set to units per minute (like *G94*).
6. Feed and speed overrides are set to ON (like *M48*).
7. Cutter compensation is turned off (like *G40*).
8. The spindle is stopped (like *M5*).
9. The current motion mode is set to feed (like *G1*).
10. Coolant is turned off (like *M9*).

**Note**
Lines of code after M2/M30 will not be executed. Pressing cycle start will start the program at the beginning of the file.

**Warning**
Using % to wrap the G code does not do the same thing as a *Program End*. See File Requirements for more information on what using % does not do.

### 3.4.4 M60 Pallet Change Pause

- *M60* - exchange pallet shuttles and then pause a running program temporarily (regardless of the setting of the optional stop switch). Pressing the cycle start button will restart the program at the following line.

### 3.4.5 M3, M4, M5 Spindle Control

- *M3* - start the spindle clockwise at the *S* speed.
- *M4* - start the spindle counterclockwise at the *S* speed.
- *M5* - stop the spindle.

It is OK to use *M3* or *M4* if the S spindle speed is set to zero. If this is done (or if the speed override switch is enabled and set to zero), the spindle will not start turning. If, later, the spindle speed is set above zero (or the override switch is turned up), the spindle will start turning. It is OK to use *M3* or *M4* when the spindle is already turning or to use *M5* when the spindle is already stopped.

### 3.4.6 M6 Tool Change

#### 3.4.6.1 Manual Tool Change

If the HAL component hal_manualtoolchange is loaded, M6 will stop the spindle and prompt the user to change the tool based on the last *T*- number programmed. For more information on hal_manualtoolchange see the Manual Tool Change section.

#### 3.4.6.2 Tool Changer

To change a tool in the spindle from the tool currently in the spindle to the tool most recently selected (using a T word - see Section Select Tool), program *M6*. When the tool change is complete:

- The spindle will be stopped.
- The tool that was selected (by a T word on the same line or on any line after the previous tool change) will be in the spindle.
- If the selected tool was not in the spindle before the tool change, the tool that was in the spindle (if there was one) will be placed back into the tool changer magazine.
- If configured in the .ini file some axis positions may move when a M6 is issued. See the EMCIO section for more information on tool change options.
- No other changes will be made. For example, coolant will continue to flow during the tool change unless it has been turned off by an *M9*.

**Warning**
The tool length offset is not changed by *M6*, use *G43* after the *M6* to change the tool length offset.

The tool change may include axis motion. It is OK (but not useful) to program a change to the tool already in the spindle. It is OK if there is no tool in the selected slot; in that case, the spindle will be empty after the tool change. If slot zero was last selected, there will definitely be no tool in the spindle after a tool change. The tool changer will have to be setup to perform the tool change in hal and possibly classicladder.

### 3.4.7 M7, M8, M9 Coolant Control

- *M7* - turn mist coolant on.
- *M8* - turn flood coolant on.
- *M9* - turn all coolant off.

It is OK to use any of these commands, regardless of the current coolant state.

### 3.4.8 M19 Orient Spindle

- *M19 R- Q- [P-]*
- *R* Position to rotate to from 0, valid range is 0-360 degrees
- *Q* Number of seconds to wait until orient completes. If motion.spindle.is_oriented does not become true within Q timeout an error occurs.
- *P* Direction to rotate to position.
  - *0* rotate for smallest angular movement (default)
  - *1* always rotate clockwise (same as M3 direction)
  - *2* always rotate counterclockwise (same as M4 direction)

M19 is cleared by any of M3,M4,M5.

Spindle orientation requires a quadrature encoder with an index to sense the spindle shaft position and direction of rotation.

INI Settings in the [RS274NGC] section.

ORIENT_OFFSET = 0-360 (fixed offset in degrees added to M19 R word)

HAL Pins

- *motion.spindle-orient-angle* (out float) Desired spindle orientation for M19. Value of the M19 R word parameter plus the value of the [RS274NGC]ORIENT_OFFSET ini parameter.
- *motion.spindle-orient-mode* (out s32) Desired spindle rotation mode. Reflects M19 P parameter word, Default = 0
- *motion.spindle-orient* (out bit) Indicates start of spindle orient cycle. Set by M19. Cleared by any of M3,M4,M5. If spindle-orient-fault is not zero during spindle-orient true, the M19 command fails with an error message.
- *motion.spindle-is-oriented* (in bit) Acknowledge pin for spindle-orient. Completes orient cycle. If spindle-orient was true when spindle-is-oriented was asserted, the spindle-orient pin is cleared and the spindle-locked pin is asserted. Also, the spindle-brake pin is asserted.
- *motion.spindle-orient-fault* (in s32) Fault code input for orient cycle. Any value other than zero will cause the orient cycle to abort.
- *motion.spindle-locked* (out bit) Spindle orient complete pin. Cleared by any of M3,M4,M5.

### 3.4.9 M48, M49 Speed and Feed Override Control

- *M48* - enable the spindle speed and feed rate override controls.
- *M49* - disable both controls.

It is OK to enable or disable the controls when they are already enabled or disabled. See the Feed Rate Section for more details.

### 3.4.10 M50 Feed Override Control

- *M50 <P1>* - enable the feed rate override control. The P1 is optional.
- *M50 P0* - disable the feed rate control.

While disabled the feed override will have no influence, and the motion will be executed at programmed feed rate. (unless there is an adaptive feed rate override active).

### 3.4.11 M51 Spindle Speed Override Control

- *M51 <P1>* - enable the spindle speed override control. The P1 is optional.
- *M51 P0* - disable the spindle speed override control program. While disabled the spindle speed override will have no influence, and the spindle speed will have the exact program specified value of the S-word (described in Spindle Speed Section).

### 3.4.12 M52 Adaptive Feed Control

- *M52 <P1>* - use an adaptive feed. The P1 is optional.
- *M52 P0* - stop using adaptive feed.

When adaptive feed is enabled, some external input value is used together with the user interface feed override value and the commanded feed rate to set the actual feed rate. In LinuxCNC, the HAL pin *motion.adaptive-feed* is used for this purpose. Values on *motion.adaptive-feed* should range from 0 (feed hold) to 1 (full speed).

### 3.4.13 M53 Feed Stop Control

- *M53 <P1>* - enable the feed stop switch. The P1 is optional. Enabling the feed stop switch will allow motion to be interrupted by means of the feed stop control. In LinuxCNC, the HAL pin *motion.feed-hold* is used for this purpose. A *true* value will cause the motion to stop when *M53* is active.
- *M53 P0* - disable the feed stop switch. The state of *motion.feed-hold* will have no effect on feed when M53 is not active.

### 3.4.14 M61 Set Current Tool

- *M61 Q-* - change the current tool number while in MDI or Manual mode. One use is when you power up LinuxCNC with a tool currently in the spindle you can set that tool number without doing a tool change.

It is an error if:

- Q- is not 0 or greater

### 3.4.15 M62 - M65 Digital Output Control

- *M62 P-* - turn on digital output synchronized with motion. The P- word specifies the digital output number.
- *M63 P-* - turn off digital output synchronized with motion. The P- word specifies the digital output number.
- *M64 P-* - turn on digital output immediately. The P- word specifies the digital output number.
- *M65 P-* - turn off digital output immediately. The P- word specifies the digital output number.

The P-word ranges from 0 to a default value of 3. If needed the the number of I/O can be increased by using the num_dio parameter when loading the motion controller. See the Motion Section for more information.

The M62 & M63 commands will be queued. Subsequent commands referring to the same output number will overwrite the older settings. More than one output change can be specified by issuing more than one M62/M63 command.

The actual change of the specified outputs will happen at the beginning of the next motion command. If there is no subsequent motion command, the queued output changes won't happen. It's best to always program a motion G code (G0, G1, etc) right after the M62/63.

M64 & M65 happen immediately as they are received by the motion controller. They are not synchronized with movement, and they will break blending.

**Note**
M62-65 will not function unless the appropriate motion.digital-out-nn pins are connected in your hal file to outputs.

### 3.4.16 M66 Wait on Input

```
M66 P- | E- <L->
```

- *P-* - specifies the digital input number from 0 to 3.
- *E-* - specifies the analog input number from 0 to 3.
- *L-* - specifies the wait mode type.
  - *Mode 0: IMMEDIATE* - no waiting, returns immediately. The current value of the input is stored in parameter #5399
  - *Mode 1: RISE* - waits for the selected input to perform a rise event.
  - *Mode 2: FALL* - waits for the selected input to perform a fall event.
  - *Mode 3: HIGH* - waits for the selected input to go to the HIGH state.
  - *Mode 4: LOW* - waits for the selected input to go to the LOW state.
- *Q-* - specifies the timeout in seconds for waiting. If the timeout is exceeded, the wait is interrupt, and the variable #5399 will be holding the value -1. The Q value is ignored if the L-word is zero (IMMEDIATE). A Q value of zero is an error if the L-word is non-zero.
- Mode 0 is the only one permitted for an analog input.

#### M66 Example Lines

```
M66 P0 L3 Q5 (wait up to 5 seconds for digital input 0 to turn on)
```

M66 wait on an input stops further execution of the program, until the selected event (or the programmed timeout) occurs.

It is an error to program M66 with both a P-word and an E-word (thus selecting both an analog and a digital input). In LinuxCNC these inputs are not monitored in real time and thus should not be used for timing-critical applications.

The number of I/O can be increased by using the num_dio or num_aio parameter when loading the motion controller. See the Motion Section for more information.

**Note**
M66 will not function unless the appropriate motion.digital-in-nn pins or motion.analog-in-nn pins are connected in your hal file to an input.

#### Example HAL Connection

```
net signal-name motion.digital-in-00 <= parport.0.pin10-in
```

### 3.4.17 M67 Analog Output,Synchronized

```
M67 E- Q-
```

- *M67* - set an analog output synchronized with motion.
- *E-* - output number ranging from 0 to 3.
- *Q-* - is the value to set (set to 0 to turn off).

The actual change of the specified outputs will happen at the beginning of the next motion command. If there is no subsequent motion command, the queued output changes won't happen. It's best to always program a motion G code (G0, G1, etc) right after the M67. M67 functions the same as M62-63.

The number of I/O can be increased by using the num_dio or num_aio parameter when loading the motion controller. See the Motion Section for more information.

**Note**
M67 will not function unless the appropriate motion.analog-out-nn pins are connected in your hal file to outputs.

### 3.4.18 M68 Analog Output, Immediate

```
M68 E- Q-
```

- *M68* - set an analog output immediately.
- *E-* - output number ranging from 0 to 3.
- *Q-* - is the value to set (set to 0 to turn off).

M68 output happen immediately as they are received by the motion controller. They are not synchronized with movement, and they will break blending. M68 functions the same as M64-65.

The number of I/O can be increased by using the num_dio or num_aio parameter when loading the motion controller. See the Motion Section for more information.

**Note**
M68 will not function unless the appropriate motion.analog-out-nn pins are connected in your hal file to outputs.

### 3.4.19 M70 Save Modal State

To explicitly save the modal state at the current call level, program *M70*. Once modal state has been saved with *M70*, it can be restored to exactly that state by executing an *M72*.

A pair of *M70* and *M72* instructions will typically be used to protect a program against inadvertant modal changes within subroutines.

The state saved consists of:

- current G20/G21 settings (imperial/metric)
- selected plane (G17/G18/G19 G17.1,G18.1,G19.1)
- status of cutter compensation (G40,G41,G42,G41.1,G42,1)
- distance mode - relative/absolute (G90/G91)
- feed mode (G93/G94,G95)
- current coordinate system (G54-G59.3)
- tool length compensation status (G43,G43.1,G49)
- retract mode (G98,G99)
- spindle mode (G96-css or G97-RPM)

- arc distance mode (G90.1, G91.1)
- lathe radius/diameter mode (G7,G8)
- path control mode (G61, G61.1, G64)
- current feed and speed (*F* and *S* values)
- spindle status (M3,M4,M5) - on/off and direction
- mist (M7) and flood (M8) status
- speed override (M51) and feed override (M50) settings
- adaptive feed setting (M52)
- feed hold setting (M53)

Note that in particular, the motion mode (G1 etc) is NOT restored.

*current call level* means either:

- executing in the main program. There is a single storage location for state at the main program level; if several *M70* instructions are executed in turn, only the most recently saved state is restored when an *M72* is executed.
- executing within a G-code subroutine. The state saved with *M70* within a subroutine behaves exactly like a local named parameter - it can be referred to only within this subroutine invocation with an *M72* and when the subroutine exits, the parameter goes away.

A recursive invocation of a subroutine introduces a new call level.

### 3.4.20 M71 Invalidate Stored Modal State

Modal state saved with an *M70* or by an *M73* at the current call level is invalidated (cannot be restored from anymore).

A subsequent *M72* at the same call level will fail.

If executed in a subroutine which protects modal state by an *M73*, a subsequent return or endsub will **not** restore modal state.

The usefulness of this feature is dubious. It should not be relied upon as it might go away.

### 3.4.21 M72 Restore Modal State

Modal state saved with an M70 code can be restored by executing an *M72*.

The handling of G20/G21 is specially treated as feeds are interpreted differently depending on G20/G21: if length units (mm/in) are about to be changed by the restore operation, 'M72 'will restore the distance mode first, and then all other state including feed to make sure the feed value is interpreted in the correct unit setting.

It is an error to execute an *M72* with no previous *M70* save operation at that level.

The following example demonstrates saving and explicitely restoring modal state around a subroutine call using *M70* and *M72*. Note that the *imperialsub* subroutine is not "aware" of the M7x features and can be used unmodified:

```
O<showstate> sub
(DEBUG, imperial=#<_imperial> absolute=#<_absolute> feed=#<_feed> rpm=#<_rpm>)
O<showstate> endsub

O<imperialsub> sub
g20 (imperial)
g91 (relative mode)
F5 (low feed)
S300 (low rpm)
```

```
(debug, in subroutine, state now:)
o<showstate> call
O<imperialsub> endsub

; main program
g21 (metric)
g90 (absolute)
f200 (fast speed)
S2500 (high rpm)

(debug, in main, state now:)
o<showstate> call

M70 (save caller state in at global level)
O<imperialsub> call
M72 (explicitely restore state)

(debug, back in main, state now:)
o<showstate> call
m2
```

### 3.4.22 M73 Save and Autorestore Modal State

To save modal state within a subroutine, and restore state on subroutine *endsub* or any *return* path, program *M73*.

Aborting a running program in a subroutine which has an *M73* operation will **not** restore state .

Also, the normal end (*M2*) of a main program which contains an *M73* will **not** restore state.

The suggested use is at the beginning of a O-word subroutine as in the following example. Using *M73* this way enables designing subroutines which need to modify modal state but will protect the calling program against inadvertant modal changes. Note the use of predefined named parameters in the *showstate* subroutine.

```
O<showstate> sub
(DEBUG, imperial=#<_imperial> absolute=#<_absolute> feed=#<_feed> rpm=#<_rpm>)
O<showstate> endsub

O<imperialsub> sub
M73 (save caller state in current call context, restore on return or endsub)
g20 (imperial)
g91 (relative mode)
F5 (low feed)
S300 (low rpm)
(debug, in subroutine, state now:)
o<showstate> call

; note - no M72 is needed here - the following endsub or an
; explicit 'return' will restore caller state
O<imperialsub> endsub

; main program
g21 (metric)
g90 (absolute)
f200 (fast speed)
S2500 (high rpm)
(debug, in main, state now:)
o<showstate> call
o<imperialsub> call
(debug, back in main, state now:)
o<showstate> call
m2
```

#### 3.4.22.1 Selectively Restoring Modal State

Executing an *M72* or returning from a subroutine which contains an *M73* will restore all modal state saved.

If only some aspects of modal state should be preserved, an alternative is the usage of predefined named parameters, local parameters and conditional statements. The idea is to remember the modes to be restored at the beginning of the subroutine, and restore these before exiting. Here is an example, based on snippet of *nc_files/tool-length-probe.ngc*:

```
O<measure> sub   (measure reference tool)
;
#<absolute> = #<_absolute>  (remember in local variable if G90 was set)
;
g30 (above switch)
g38.2 z0 f15 (measure)
g91 g0z.2 (off the switch)
#1000=#5063 (save reference tool length)
(print,reference length is #1000)
;
O<restore_abs> if [#<absolute>]
    g90 (restore G90 only if it was set on entry:)
O<restore_abs> endif
;
O<measure> endsub
```

### 3.4.23 M100 - M199 User Defined Commands

```
M1-- <P- Q->
```

- *M1--* - an integer in the range of 100 - 199.
- *P-* - a number passed to the file as the first parameter.
- *Q-* - a number passed to the file as the second parameter.

**Note**
After creating a new *M1nn* file you must restart the GUI so it is aware of the new file, otherwise you will get an *Unkown m code* error.

The external program named *M100* through *M199* (no extension and a capitol M) is executed with the optional P and Q values as its two arguments. Execution of the G code file pauses until the external program exits. Any valid executable file can be used. The file must be located in the search path specificed in the ini file configuration. See the Display Section for more information on search paths.

**Warning**
Do not use a word processor to create or edit the files. A word processor will leave unseen codes that will cause problems and may prevent a bash or python file from working. Use a text editor like Gedit in Ubuntu or Notepad++ in other operating systems to create or edit the files.

The error *Unknown M code used* denotes one of the following

- The specified User Defined Command does not exist
- The file is not an executable file

- The file name has an extension
- The file name does not follow this format M1nn where nn = 00 through 99
- The file name used a lower case M

For example to open and close a collet closer that is controlled by a parallel port pin using a bash script file using M101 and M102. Create two files named M101 and M102. Set them as executable files (typically right click/properties/permissions) before running LinuxCNC. Make sure the parallel port pin is not connected to anything in a HAL file.

**M101 Example File**

```
#!/bin/bash
# file to turn on parport pin 14 to open the collet closer
halcmd setp parport.0.pin-14-out True
exit 0
```

**M102 Example File**

```
#!/bin/bash
# file to turn off parport pin 14 to open the collet closer
halcmd setp parport.0.pin-14-out False
exit 0
```

To pass a variable to a M1nn file you use the P and Q option like this:

```
M100 P123.456 Q321.654
```

**M100 Example file**

```
#!/bin/bash
voltage=$1
feedrate=$2
halcmd setp thc.voltage $voltage
halcmd setp thc.feedrate $feedrate
exit 0
```

To display a graphic message and stop until the message window is closed use a graphic display program like Eye of Gnome to display the graphic file. When you close it the program will resume.

**M110 Example file**

```
#!/bin/bash
eog /home/john/linuxcnc/nc_files/message.png
exit 0
```

To display a graphic message and continue processing the G code file suffix an ampersand to the command.

**M110 Example display and keep going**

```
#!/bin/bash
eog /home/john/linuxcnc/nc_files/message.png &
exit 0
```

## 3.5 O Codes

O-codes provide for flow control in NC programs. Each block has an associated number, which is the number used after O. Care must be taken to properly match the O-numbers. O codes use the letter *O* not the number zero as the first character in the number like O100.

**Numbering Example**

```
o100 sub
(notice that the if-endif block uses a different number)
  o110 if [#2 GT 5]
    (some code here)
  o110 endif
  (some more code here)
o100 endsub
```

The behavior is undefined if:

- The same number is used for more than one block
- Other words are used on a line with an O- word
- Comments are used on a line with an O-word

**Note**
Using the lower case o makes it easier to distinguish from a 0 that might have been mistyped. For example o100 is easier to see than O100 that it is not a 0.

The following statements cause an error message and abort the interpreter:

- a `return` or `endsub` not within a sub defintion
- a label on `repeat` which is defined elsewhere
- a label on `while` which is defińed elsewhere and not referring to a `do`
- a label on `if` defined elsewhere
- a undefined label on `else` or `elseif`
- a label on `else`, `elseif` or `endif` not pointing to a matching `if`
- a label on `break` or `continue` which does not point to a matching `while` or `do`
- a label on `endrepeat` or `endwhile` no referring to a corresponding `while` or `repeat`

To make these errors non-fatal warnings on stderr, set bit 0x20 in the `[RS274NGC]FEATURE=` mask ini option.

### 3.5.1 Subroutines

Subroutines extend from a *O- sub* to an *O- endsub* . The lines between *O- sub* and *O- endsub* are not executed until the subroutine is called with *O- call*.

**Subroutine Example**

```
o100 sub
  G53 G0 X0 Y0 Z0 (rapid move to machine home)
o100 endsub
...
o100 call (call the subroutine here)
M2
```

See G53 & G0 & M2 sections for more information.

**O- Return** Inside a subroutine, *O- return* can be executed. This immediately returns to the calling code, just as though *O- endsub* was encountered.

**O- Return Example**

```
o100 sub
  o110 if [#2 GT 5] (test if parameter #2 is greater than 5)
    o100 return (return to top of subroutine if test is true)
  o110 endif
  (some code here that only gets executed if parameter #2 is less than 5)
o100 endsub
```

See the Binary Operators & Parameters sections for more information.

**O- Call** *O- Call* takes up to 30 optional arguments, which are passed to the subroutine as *#1*, *#2* , . . . , #N. Parameters from #N+1 to #30 have the same value as in the calling context. On return from the subroutine, the values of parameters #1 through #30 (regardless of the number of arguments) will be restored to the values they had before the call. Parameters #1 - #30 are local to the subroutine.

Because *1 2 3* is parsed as the number 123, the parameters must be enclosed in square brackets. The following calls a subroutine with 3 arguments:

**O- Call Example**

```
o200 call [1] [2] [3]
```

Subroutine bodies may not be nested. They may only be called after they are defined. They may be called from other functions, and may call themselves recursively if it makes sense to do so. The maximum subroutine nesting level is 10.

Subroutines do not have *return values*, but they may change the value of parameters above #30 and those changes will be visible to the calling code. Subroutines may also change the value of global named parameters.

### 3.5.2 Looping

The *while loop* has two structures: *while/endwhile*, and *do/while*. In each case, the loop is exited when the *while* condition evaluates to false. The difference is when the test condition is done. The *do/while* loop runs the code in the loop then checks the test condition. The *while/endwhile* loop does the test first.

**While Endwhile Example**

```
(draw a sawtooth shape)
G0 X1 Y0 (move to start position)
#1 = 0 (assign parameter #1 the value of 0)
F25 (set a feed rate)
o101 while [#1 LT 10]
  G1 X0
  G1 Y[#1/10] X1
  #1 = [#1+1] (increment the test counter)
o101 endwhile
M2 (end program)
```

**Do While Example**

```
#1 = 0 (assign parameter #1 the value of 0)
o100 do
  (debug, parameter 1 = #1)
  o110 if [#1 EQ 2]
    #1 = 3 (assign the value of 3 to parameter #1)
    (msg, #1 has been assigned the value of 3)
    o100 continue (skip to start of loop)
  o110 endif
  (some code here)
  #1 = [#1 + 1] (increment the test counter)
o100 while [#1 LT 3]
(msg, Loop Done!)
M2
```

Inside a while loop, *O- break* immediately exits the loop, and *O- continue* immediately skips to the next evaluation of the *while* condition. If it is still true, the loop begins again at the top. If it is false, it exits the loop.

### 3.5.3 Conditional

The *if* conditional consists of a group of statements with the same *o* number that start with *if* and end with *endif*. Optional *elseif* and *else* conditions may be between the starting *if* and the ending *endif*.

If the *if* conditional evaluates to true then the group of statements following the *if* up to the next conditional line are executed.

If the *if* conditional evaluates to false then the *elseif* conditions are evaluated in order until one evaluates to true. If the *elseif* condition is true then the statements following the *elseif* up to the next conditional line are executed. If none of the *if* or *elseif* conditions evaluate to true then the statements following the *else* are executed. When a condition is evaluated to true no more conditions are evaluated in the group.

**If Endif Example**

```
o101 if [#31 EQ 3] (if parameter #31 is equal to 3 set S2000)
  S2000
o101 endif
```

**If ElseIf Else EndIf Example**

```
o102 if [#2 GT 5] (if parameter #2 is greater than 5 set F100)
  F100
o102 elseif [#2 LT 2] (else if parameter #2 is less than 2 set F200)
  F200
o102 else (else if parameter #2 is 2 through 5 set F150)
  F150
o102 endif
```

Several conditons may be tested for by *elseif* statements until the *else* path is finally executed if all preceding conditons are false:

**If Elseif Else Endif Example**

```
O102 if [#2 GT 5] (if parameter #2 is greater than 5 set F100)
  F100
O102 elseif [#2 LT 2] (else if parameter #2 less than 2 set F200)
  F20
O102 else (parameter #2 between 2 and 5)
  F200
O102 endif
```

### 3.5.4 Repeat

The *repeat* will execute the statements inside of the repeat/endrepeat the specified number of times. The example shows how you might mill a diagonal series of shapes starting at the present position.

**Repeat Example**

```
(Mill 5 diagonal shapes)
G91 (Incremental mode)
o103 repeat [5]
... (insert milling code here)
G0 X1 Y1 (diagonal move to next position)
o103 endrepeat
G90 (Absolute mode)
```

### 3.5.5 Indirection

The O-number may be given by a parameter and/or calculation.

**Indirection Example**

```
o[#101+2] call
```

**Computing values in O-words** For more information on computing values see the following sections

- Parameters
- Expressions
- Binary Operators
- Functions

### 3.5.6 Calling Files

To call a separate file with a subroutine name the file the same as your call and include a sub and endsub in the file. The file must be in the directory pointed to by *PROGRAM_PREFIX* or *SUBROUTINE_PATH* in the ini file. The file name can include **lowercase** letters, numbers, dash, and underscore only. A named subroutine file can contain only a single subroutine definition.

**Named File Example**

```
o<myfile> call
```

**Numbered File Example**

```
o123 call
```

In the called file you must include the oxxx sub and endsub and the file must be a valid file.

**Called File Example**

```
(filename myfile.ngc)
o<myfile> sub
  (code here)
o<myfile> endsub
M2
```

**Note**
The file names are lowercase letters only so *o<MyFile>* is converted to *o<myfile>* by the interpreter. More information about the search path and options for the search path are in the INI Configuration Section.

### 3.5.7 Subroutine return values

Subroutines may optionally return a value by an optional expression at an *endsub* or *return* statement.

**Return value example**

```
o123 return [#2 *5]
...
o123 endsub [3 * 4]
```

A subroutine return value is stored in the <*_value*> predefined named parameter , and the <*_value_returned*> predefined parameter is set to 1, to indicate a value was returned. Both parameters are global, and are cleared just before the next subroutine call.

## 3.6 Other Codes

### 3.6.1 F: Set Feed Rate

*Fx* - set the feed rate to *x*. *x* is usually in machine units (inches or millimeters) per minute.

The application of the feed rate is as described in the Feed Rate Section, unless *inverse time feed rate mode* or *feed per revolution mode* are in effect, in which case the feed rate is as described in the G93 G94 G95 Section.

### 3.6.2 S: Set Spindle Speed

*Sx* - set the speed of the spindle to *x* revolutions per minute (RPM).

The spindle will turn at that speed when a *M3* or *M4* is in effect. It is OK to program an S word whether the spindle is turning or not. If the speed override switch is enabled and not set at 100%, the speed will be different from what is programmed. It is OK to program S0, the spindle will not turn if that is done.

It is an error if:

- the S number is negative.

### 3.6.3 T: Select Tool

*Tx* - prepare to change to tool *x*.

The tool is not changed until an *M6* is programmed (see Section M6). The T word may appear on the same line as the *M6* or on a previous line. It is OK if T words appear on two or more lines with no tool change. Only the the most recent T word will take effect at the next tool change.

**Note**
When LinuxCNC is configured for a nonrandom toolchanger (see the entry for RANDOM_TOOLCHANGER in the EMCIO Section), *T0* gets special handling: no tool will be selected. This is useful if you want the spindle to be empty after a tool change.

**Note**
When LinuxCNC is configured for a random toolchanger (see the entry for RANDOM_TOOLCHANGER in the EMCIO Section), *T0* does not get any special treatment: T0 is a valid tool like any other. It is customary to use T0 on a random toolchanger machine to track an empty pocket, so that it behaves like a nonrandom toolchanger machine and unloads the spindle.

It is an error if:

- a negative T number is used,
- T number is used that does not appear in the tool table file (with the exception that T0 on nonrandom toolchangers **is** accepted, as noted above).

On some machines, the carousel will move when a T word is programmed, at the same time machining is occurring. On such machines, programming the T word several lines before a tool change will save time. A common programming practice for such machines is to put the T word for the next tool to be used on the line after a tool change. This maximizes the time available for the carousel to move.

Rapid moves after a *T<n>* will not show on the AXIS preview until after a feed move. This is for machines that travel long distances to change the tool like a lathe. This can be very confusing at first. To turn this feature off for the current tool program a G1 without any move after the *T<n>*.

## 3.7 G Code Examples

After you install LinuxCNC several sample files are placed in the /nc_files folder. Make sure the sample file is appropriate for your machine before running.

### 3.7.1 Mill Examples

#### 3.7.1.1 Helical Hole Milling

File Name: useful-subroutines.ngc

Description: Subroutine for milling a hole using parameters.

#### 3.7.1.2 Slotting

File Name: useful-subroutines.ngc

Description: Subroutine for milling a slot using parameters.

#### 3.7.1.3 Grid Probe

File Name: gridprobe.ngc

Description: Rectangular Probing

This program repeatedly probes in a regular XY grid and writes the probed location to the file *probe-results.txt* in the same directory as the .ini file.

#### 3.7.1.4 Smart Probe

File Name: smartprobe.ngc

Description: Rectangular Probing

This program repeatedly probes in a regular XY grid and writes the probed location to the file *probe-results.txt* in the same directory as the .ini file. This is improved from the grid probe file.

#### 3.7.1.5 Tool Length Probe

File Name: tool-length-probe.ngc

Description: Tool Length Probing

This program shows an example of how to measure tool lengths automatically using a switch hooked to the probe input. This is useful for machines without tool holders, where the length of a tool is different every time it is inserted.

#### 3.7.1.6 Hole Probe

File Name: probe-hole.ngc

Description: Finding the Center and Diameter of a hole.

The program demonstrates how to find the center of a hole, measure the hole diameter and record the results.

#### 3.7.1.7 Cutter Compensation

To be added

### 3.7.2 Lathe Examples

#### 3.7.2.1 Threading

File Name lathe-g76.ngc

Description: Facing, threading and parting off.

This file shows an example of threading on a lathe using parameters.

## 3.8 RS274/NGC Differences

### 3.8.1 Changes from RS274/NGC

DIFFERENCES THAT CHANGE THE MEANING OF RS274/NGC PROGRAMS

**Location after a tool change**
In LinuxCNC, the machine does not return to its original position after a tool change. This change was made because the new tool might be longer than the old tool, and the move to the original machine position could therefore leave the tool tip too low.

**Offset parameters are ini file units**
In LinuxCNC, the values stored in parameters for the G28 and G30 home locations, the P1…P9 coordinate systems, and the G92 offset are in "ini file units". This change was made because otherwise the meaning of a location changed depending on whether G20 or G21 was active when G28, G30, G10 L2, or G92.3 is programmed.

**Tool table lengths/diameters are in ini file units**
In LinuxCNC, the tool lengths (offsets) and diameters in the tool table are specified in ini file units only. This change was made because otherwise the length of a tool and its diameter would change based on whether G20 or G21 was active when initiating G43, G41, G42 modes. This made it impossible to run G code in the machine's non-native units, even when the G code was simple and well-formed (starting with G20 or G21, and didn't change units throughout the program), without changing the tool table.

**G84, G87 not implemented**
G84 and G87 are not currently implemented, but may be added to a future release of LinuxCNC.

**G28, G30 with axis words**
When G28 or G30 is programmed with only some axis words present, LinuxCNC only moves the named axes. This is common on other machine controls. To move some axes to an intermediate point and then move all axes to the predefined point, write two lines of G code:

G0 X- Y- (axes to move to intermediate point) G28 (move all axes to predefined point)

### 3.8.2 Additions to RS274/NGC

DIFFERENCES THAT DO NOT CHANGE THE MEANING OF RS274/NGC PROGRAMS

**G33, G76 threading codes**
These codes are not defined in RS274/NGC.

**G38.2**
The probe tip is not retracted after a G38.2 movement. This retraction move may be added in a future release of LinuxCNC.

**G38.3…G38.5**
These codes are not defined in RS274/NGC

**O-codes**
These codes are not defined in RS274/NGC

**M50…M53 overrides**
These codes are not defined in RS274/NGC

**M61..M66**
These codes are not defined in RS274/NGC

**G43, G43.1**
*Negative Tool Lengths*

The RS274/NGC spec says "it is expected that" all tool lengths will be positive. However, G43 works for negative tool lengths.

*Lathe tools*

G43 tool length compensation can offset the tool in both the X and Z dimensions. This feature is primarily useful on lathes.

*Dynamic tool lengths*

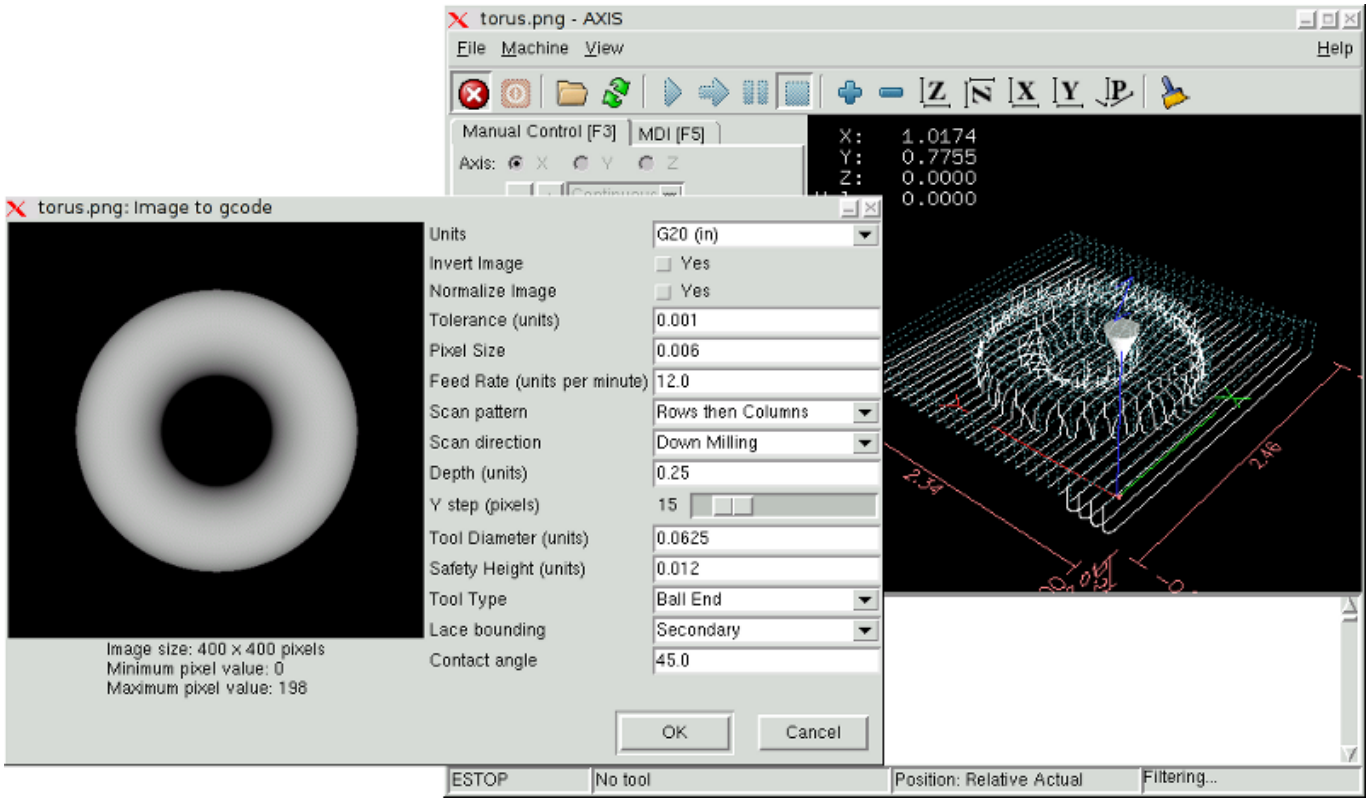
LinuxCNC allows specification of a computed tool length through G43.1 I K.

**G41.1, G42.1**
LinuxCNC allows specification of a tool diameter and, if in lathe mode, orientation in the G code. The format is G41.1/G42.1 D L, where D is diameter and L (if specified) is the lathe tool orientation.

**G43 without H word**
In ngc, this is not allowed. In LinuxCNC, it sets length offsets for the currently loaded tool. If no tool is currently loaded, it is an error. This change was made so the user doesn't have to specify the tool number in two places for each tool change, and because it's consistent with the way G41/G42 work when the D word is not specified.

**U, V, and W axes**
LinuxCNC allows machines with up to 9 axes by defining an additional set of 3 linear axes known as U, V and W

## 3.9 Image to G Code



### 3.9.1 What is a depth map?

A depth map is a greyscale image where the brightness of each pixel corresponds to the depth (or height) of the object at each point.

### 3.9.2 Integrating image-to-gcode with the AXIS user interface

Add the following lines to the *[FILTER]* section of your .ini file to make AXIS automatically invoke image-to-gcode when you open a .png, .gif, or .jpg image

```
PROGRAM_EXTENSION = .png,.gif,.jpg Grayscale Depth Image
png = image-to-gcode
gif = image-to-gcode
jpg = image-to-gcode
```

The standard *sim/axis.ini* configuration file is already configured this way.

### 3.9.3 Using image-to-gcode

Start image-to-gcode either by opening an image file in AXIS, or by invoking image-to-gcode from the terminal, as follows:

```
image-to-gcode torus.png > torus.ngc
```

Verify all the settings in the right-hand column, then press OK to create the gcode. Depending on the image size and options chosen, this may take from a few seconds to a few minutes. If you are loading the image in AXIS, the gcode will automatically be loaded and previewed once image-to-gcode completes. In AXIS, hitting reload will show the image-to-gcode option screen again, allowing you to tweak them.

### 3.9.4 Option Reference

#### 3.9.4.1 Units

Specifies whether to use G20 (inches) or G21 (mm) in the generated g-code and as the units for each option labeled *(units)*.

#### 3.9.4.2 Invert Image

If “no”, the black pixel is the lowest point and the white pixel is the highest point. If “yes”, the black pixel is the highest point and the white pixel is the lowest point.

#### 3.9.4.3 Normalize Image

If *yes*, the darkest pixel is remapped to black, the lightest pixel is remapped to white.

#### 3.9.4.4 Expand Image Border

If *None*, the input image is used as-is, and details which are at the very edges of the image may be cut off. If *White* or *Black*, then a border of pixels equal to the tool diameter is added on all sides, and details which are at the very edges of the images will not be cut off.

#### 3.9.4.5 Tolerance (units)

When a series of points are within *tolerance* of being a straight line, they are output as a straight line. Increasing tolerance can lead to better contouring performance in LinuxCNC, but can also remove or blur small details in the image.

#### 3.9.4.6 Pixel Size (units)

One pixel in the input image will be this many units—usually this number is much smaller than 1.0. For instance, to mill a 2.5x2.5-inch object from a 400x400 image file, use a pixel size of .00625, because 2.5 / 400 = .00625.

#### 3.9.4.7 Plunge Feed Rate (units per minute)

The feed rate for the initial plunge movement.

#### 3.9.4.8 Feed Rate (units per minute)

The feed rate for other parts of the path.

#### 3.9.4.9 Spindle Speed (RPM)

The spindle speed S code that should be put into the gcode file.

#### 3.9.4.10 Scan Pattern

Possible scan patterns are:

- Rows
- Columns
- Rows, then Columns
- Columns, then Rows

#### 3.9.4.11 Scan Direction

Possible scan directions are:

- Positive: Start milling at a low X or Y axis value, and move towards a high X or Y axis value
- Negative: Start milling at a high X or Y axis value, and move towards a low X or Y axis value
- Alternating: Start on the same end of the X or Y axis travel that the last move ended on. This reduces the amount of traverse movements
- Up Milling: Start milling at low points, moving towards high points
- Down Milling: Start milling at high points, moving towards low points

#### 3.9.4.12 Depth (units)

The top of material is always at *Z=0*. The deepest cut into the material is *Z=-depth*.

#### 3.9.4.13 Step Over (pixels)

The distance between adjacent rows or columns. To find the number of pixels for a given units distance, compute *distance/pixel size* and round to the nearest whole number. For example, if *pixel size=.006* and the desired step over *distance=.015*, then use a Step Over of 2 or 3 pixels, because *.015/.006=2.5*'.'

#### 3.9.4.14 Tool Diameter

The diameter of the cutting part of the tool.

#### 3.9.4.15 Safety Height

The height to move to for traverse movements. image-to-gcode always assumes the top of material is at *Z=0*.

#### 3.9.4.16 Tool Type

The shape of the cutting part of the tool. Possible tool shapes are:

- Ball End
- Flat End
- 45 degree “vee”
- 60 degree “vee”

#### 3.9.4.17 Lace bounding

This controls whether areas that are relatively flat along a row or column are skipped. This option only makes sense when both rows and columns are being milled. Possible bounding options are:

- None: Rows and columns are both fully milled.
- Secondary: When milling in the second direction, areas that do not strongly slope in that direction are skipped.
- Full: When milling in the first direction, areas that strongly slope in the second direction are skipped. When milling in the second direction, areas that do not strongly slope in that direction are skipped.

#### 3.9.4.18 Contact angle

When *Lace bounding* is not *None*, slopes greater than *Contact angle* are considered to be *strong* slopes, and slopes less than that angle are considered to be weak slopes.

#### 3.9.4.19 Roughing offset and depth per pass

Image-to-gcode can optionally perform rouging passes. The depth of successive roughing passes is given by *Roughing depth per pass*. For instance, entering 0.2 will perform the first roughing pass with a depth of 0.2, the second roughing pass with a depth of 0.4, and so on until the full Depth of the image is reached. No part of any roughing pass will cut closer than Roughing Offset to the final part. The following figure shows a tall vertical feature being milled. In this image, Roughing depth per pass is 0.2 inches and roughing offset is 0.1 inches.

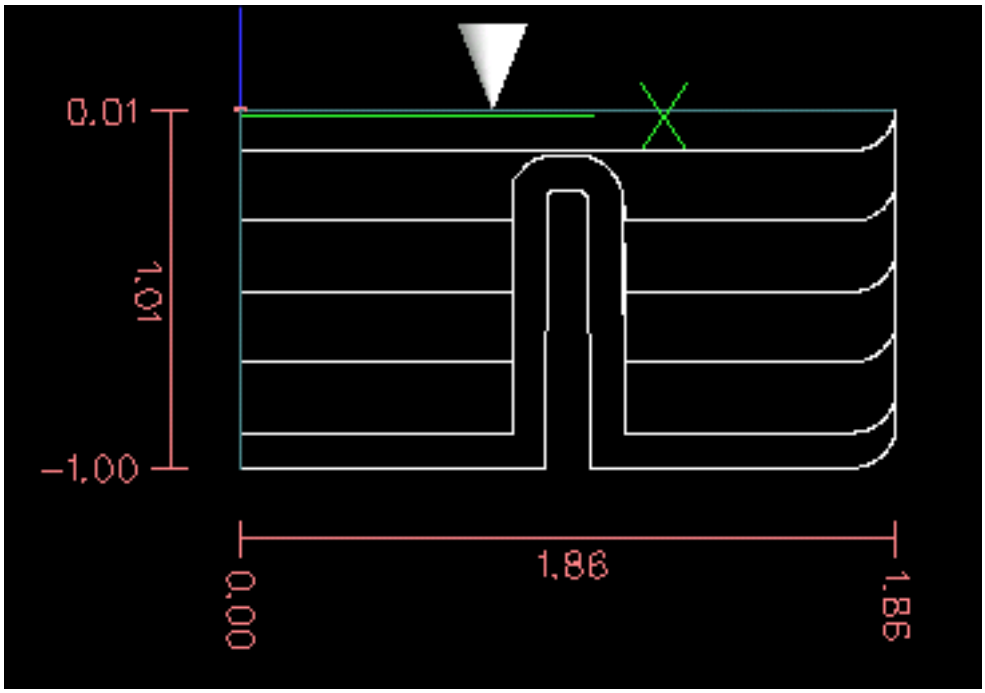



Figure 3.9: Roughing passes and final pass

# Chapter 4

# Tool Compensation

## 4.1 Tool Compensation

### 4.1.1 Tool Length Offsets

#### 4.1.1.1 Touch Off

Using the Touch Off Screen in the AXIS interface you can update the tool table automatically.

Typical steps for updating the tool table:

- After homing load a tool with *Tn M6* where *n* is the tool number.
- Move tool to an established point using a gauge or take a test cut and measure.
- Click the "Touch Off" button in the Manual Control tab (or hit the End button on your keyboard).
- Select *Tool Table* in the Coordinate System drop down box.
- Enter the gauge or measured dimension and select OK.

The Tool Table will be changed with the correct Z length to make the DRO display the correct Z position and a G43 command will be issued so the new tool Z length will be in effect. Tool table touch off is only available when a tool is loaded with *Tn M6*.

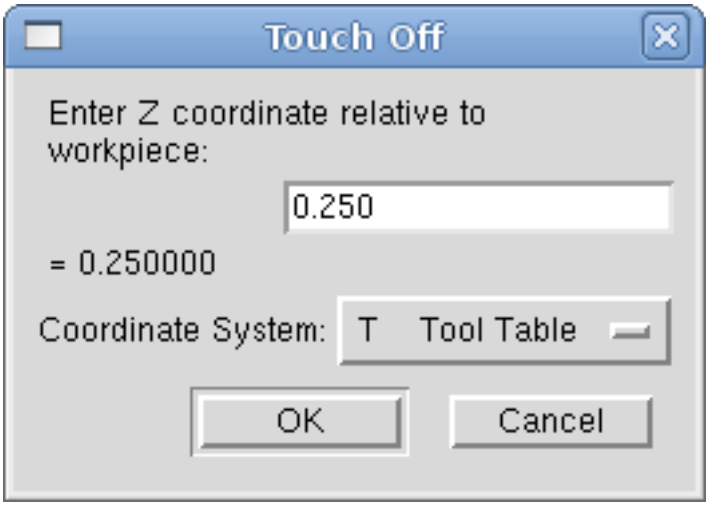



Figure 4.1: Touch Off Tool Table

#### 4.1.1.2 Using G10 L1/L10/L11

The G10 L1/L10/L11 commands can be used to set tool table offsets:

- *G10 L1 Pn* - Set offset(s) to a value. Current position irrelevant. (see G10 L1 for details)
- *G10 L10 Pn* - Set offset(s) so current position w/ fixture 1-8 becomes a value. (see G10 L10 for details)
- *G10 L11 Pn* - Set offset(s) so current position w/ fixture 9 becomes a value. (see G10 L11 for details)

### 4.1.2 Tool Table

The *Tool Table* is a text file that contains information about each tool. The file is located in the same directory as your configuration and is called *tool.tbl*. The tools might be in a tool changer or just changed manually. The file can be edited with a text editor or be updated using G10 L1. See the Lathe Tool Table Section for an example of the lathe tool table format. The maximum number of entries in the tool table is 56. The maximum tool and pocket number is 99999.

The Tool Editor or a text editor can be used to edit the tool table. If you use a text editor make sure you reload the tool table in the GUI.

#### 4.1.2.1 Tool Table Format

Table 4.1: Tool Table Format

| **T#** | **P#** | **X** | **Y** | **Z** | **A** | **B** | **C** | **U** | **V** | **W** | **Dia** | **FA** | **BA** | **Ori** | **Rem** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ; | (no data after opening semicolon) | | | | | | | | | | | | | | |
| T1 | P17 | X0 | Y0 | Z0 | A0 | B0 | C0 | U0 | V0 | W0 | D0 | I0 | J0 | Q0 | ;rem |
| T2 | P5 | X0 | Y0 | Z0 | A0 | B0 | C0 | U0 | V0 | W0 | D0 | I0 | J0 | Q0 | ;rem |
| T3 | P12 | X0 | Y0 | Z0 | A0 | B0 | C0 | U0 | V0 | W0 | D0 | I0 | J0 | Q0 | ;rem |

In general, the new tool table line format is:

- ; - opening semicolon, no data
- T - tool number, 0-99999 (tool numbers must be unique)
- P - pocket number, 1-99999 (pocket numbers must be unique)
- X..W - tool offset on specified axis - floating-point
- D - tool diameter - floating-point, absolute value
- I - front angle (lathe only) - floating-point
- J - back angle (lathe only) - floating-point
- Q - tool orientation (lathe only) - integer, 0-9
- ; - beginning of comment or remark - text

The file consists of one opening semicolon on the first line, followed by up to a maximum of 56 tool entries.

**Note**
Although tool numbers up to 99999 are allowed, the number of entries in the tool table, at the moment, is still limited to a maximum of 56 tools for technical reasons. The LinuxCNC developers plan to remove that limitation eventually. If you have a very large tool changer, please be patient.

Earlier versions of LinuxCNC had two different tool table formats for mills and lathes, but since the 2.4.x release, one tool table format is used for all machines. Just ignore the parts of the tool table that don't pertain to your machine, or which you don't need to use.

Each line of the tool table file after the opening semicolon contains the data for one tool. One line may contain as many as 16 entries, but will likely contain much fewer.

The units used for the length, diameter, etc., are in machine units.

You will probably want to keep the tool entries in ascending order, especially if you are going to be using a randomizing tool changer. Although the tool table does allow for tool numbers in any order.

Each line may have up to 16 entries. The first two entries are required. The last entry (a remark or comment, preceded by a semicolon) is optional. It makes reading easier if the entries are arranged in columns, as shown in the table, but the only format requirement is that there be at least one space or tab after each of the entries on a line and a newline character at the end of each entry.

The meanings of the entries and the type of data to be put in each are as follows.

**Tool Number (required)** The *T* column contains the number (unsigned integer) which represents a code number for the tool. The user may use any code for any tool, as long as the codes are unsigned integers.

**Pocket Number (required)** The *P* column contains the number (unsigned integer) which represents the pocket number (slot number) of the tool changer slot where the tool can be found. The entries in this column must all be different.

The pocket numbers will typically start at 1 and go up to the highest available pocket on your tool changer. But not all tool changers follow this pattern. Your pocket numbers will be determined by the numbers that your tool changer uses to refer to the pockets. So all this is to say that the pocket numbers you use will be determined by the numbering scheme used in your tool changer, and the pocket numbers you use must make sense on your machine.

**Data Offset Numbers (optional)** The *Data Offset* columns (XYZABCUVW) contain real numbers which represent tool offsets in each axis. This number will be used if tool length offsets are being used and this tool is selected. These numbers can be positive, zero, or negative, and are in fact completely optional. Although you will probably want to make at least one entry here, otherwise there would be little point in making an entry in the tool table to begin with.

In a typical mill, you probably want an entry for Z (tool length offset). In a typical lathe, you probably want an entry for X (X tool offset) and Z (Z tool offset). In a typical mill using cutter diameter compensation (cutter comp), you probably also want to add an entry for D (cutter diameter). In a typical lathe using tool nose diameter compensation (tool comp), you probably also want to add an entry for D (tool nose diameter).

A lathe also requires some additional information to describe the shape and orientation of the tool. So you probably want to have entries for I (tool front angle) and J (tool back angle). You probably also want an entry for Q (tool orientation).

See the Lathe User Information chapter for more detail.

The *Diameter* column contains a real number. This number is used only if cutter compensation is turned on using this tool. If the programmed path during compensation is the edge of the material being cut, this should be a positive real number representing the measured diameter of the tool. If the programmed path during compensation is the path of a tool whose diameter is nominal, this should be a small number (positive or negative, but near zero) representing only the difference between the measured diameter of the tool and the nominal diameter. If cutter compensation is not used with a tool, it does not matter what number is in this column.

The *Comment* column may optionally be used to describe the tool. Any type of description is OK. This column is for the benefit of human readers only. The comment must be preceded by a semicolon.

#### 4.1.2.2 Tool Changers

LinuxCNC supports three types of tool changers: *manual*, *random location* and *fixed location*. Information about configuring a LinuxCNC tool changer is in the EMCIO Section of the INI chapter.

**Manual Tool Changer** Manual tool changer (you change the tool by hand) is treated like a fixed location tool changer and the P number is ignored. Using the manual tool changer only makes sense if you have tool holders that remain with the tool (Cat, NMTB, Kwik Switch etc.) when changed thus preserving the location of the tool to the spindle. Machines with R-8 or router collet type tool holders do not preserve the location of the tool and the manual tool changer should not be used.

**Fixed Location Tool Changers** Fixed location tool changers always return the tools to a fixed position in the tool changer. This would also include designs like lathe turrets. When LinuxCNC is configured for a fixed location tool changer the *P* number is ignored (but read, preserved and rewritten) by LinuxCNC, so you can use P for any bookkeeping number you want.

**Random Location Tool Changers** Random location tool changers swap the tool in the spindle with the one in the changer. With this type of tool changer the tool will always be in a different pocket after a tool change. When a tool is changed LinuxCNC rewrites the pocket number to keep track of where the tools are. T can be any number but P must be a number that makes sense for the machine.

### 4.1.3 Cutter Compensation

Cutter Compensation allows the programmer to program the tool path without knowing the exact tool diameter. The only caveat is the programmer must program the lead in move to be at least as long as the largest tool radius that might be used.

There are two possible paths the cutter can take while cutter compensation is on to the left or right side of a line when facing the direction of cutter motion from behind the cutter. To visualize this imagine you were standing on the part walking behind the tool as it progresses across the part. G41 is your left side of the line and G42 is the right side of the line.

The end point of each move depends on the next move. If the next move creates an outside corner the move will be to the end point of the compensated cut line. If the next move creates in an inside corner the move will stop short so to not gouge the part. The following figure shows how the compensated move will stop at different points depending on the next move.

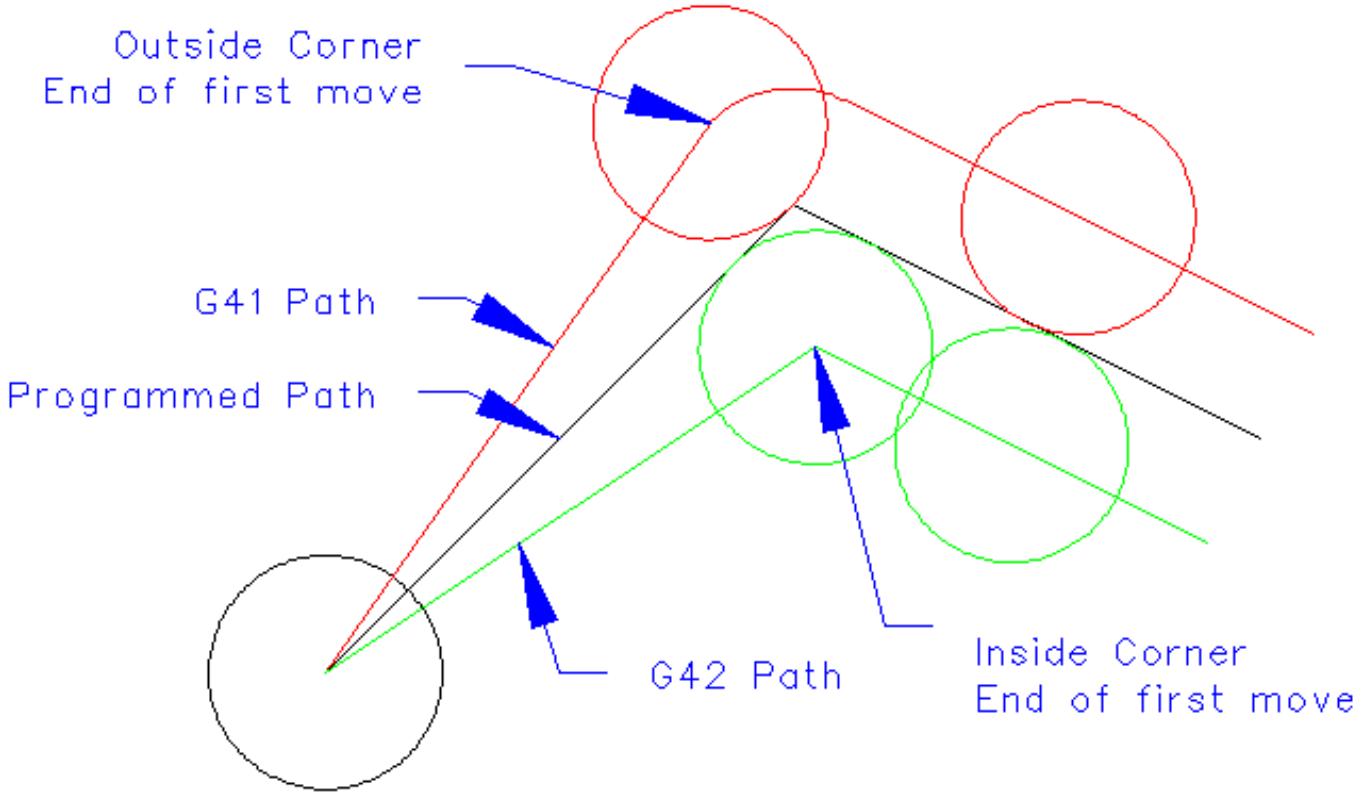



Figure 4.2: Compensation End Point

#### 4.1.3.1 Overview

**Tool Table** Cutter compensation uses the data from the tool table to determine the offset needed. The data can be set at run time with G10 L1.

**Programming Entry Moves** Any move that is long enough to perform the compensation will work as the entry move. The minimum length is the cutter radius. This can be a rapid move above the work piece. If several rapid moves are issued after a G41/42 only the last one will move the tool to the compensated position.

In the following figure you can see that the entry move is compensated to the right of the line. This puts the center of the tool to the right of X0 in this case. If you were to program a profile and the end is at X0 the resulting profile would leave a bump due to the offset of the entry move.

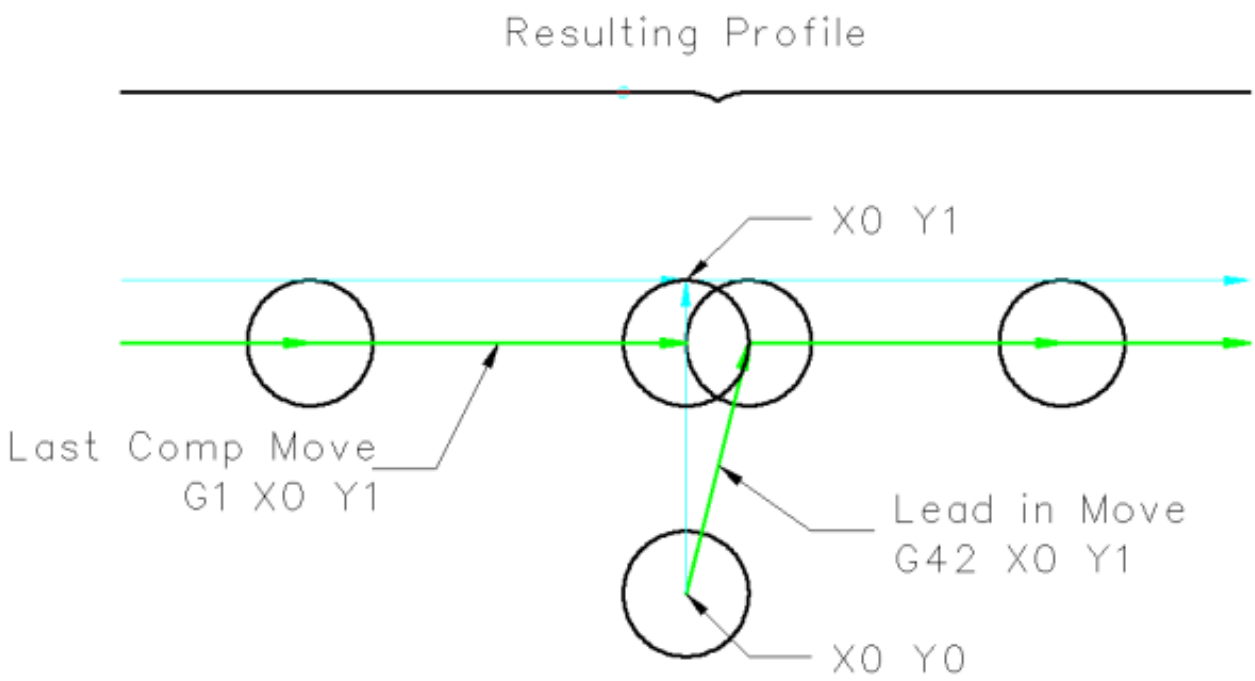



Figure 4.3: Entry Move

**Z Motion** Z axis motion may take place while the contour is being followed in the XY plane. Portions of the contour may be skipped by retracting the Z axis above the part and by extending the Z-axis at the next start point.

**Rapid Moves** Rapid moves may be programed while compensation is turned on.

GOOD PRACTICES

- Start a program with G40 to make sure compensation is off.

#### 4.1.3.2 Examples

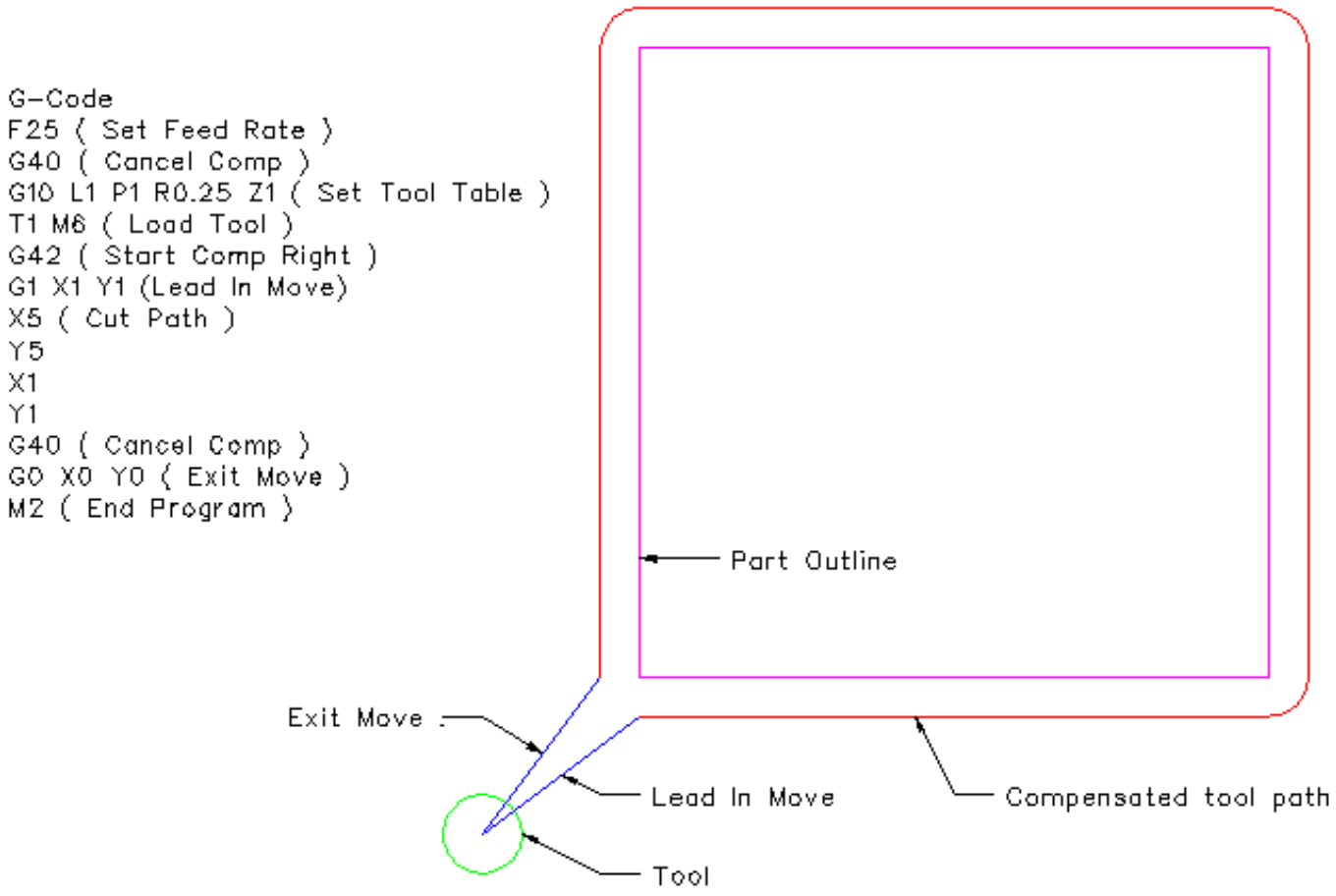



Figure 4.4: Outside Profile

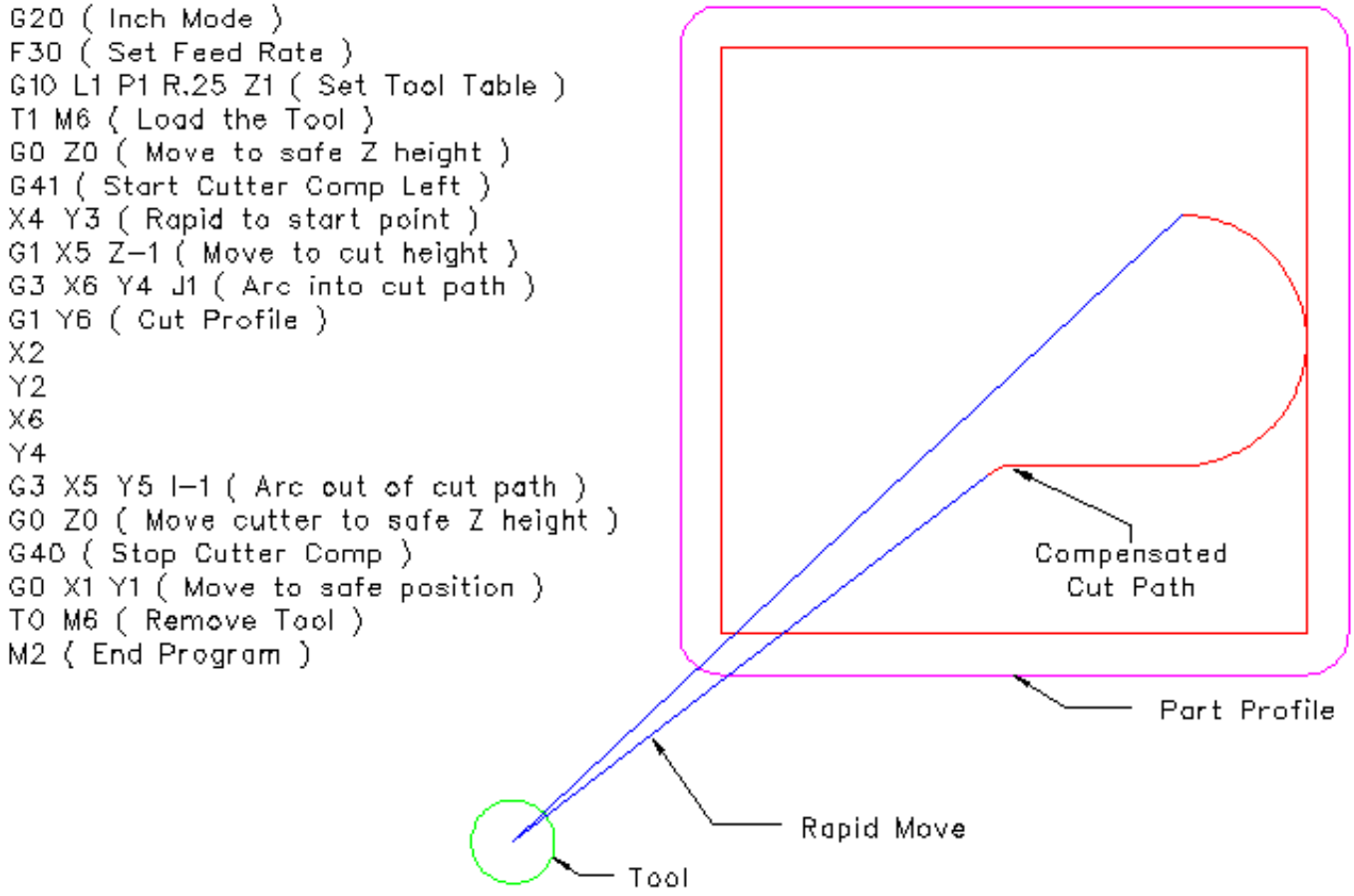
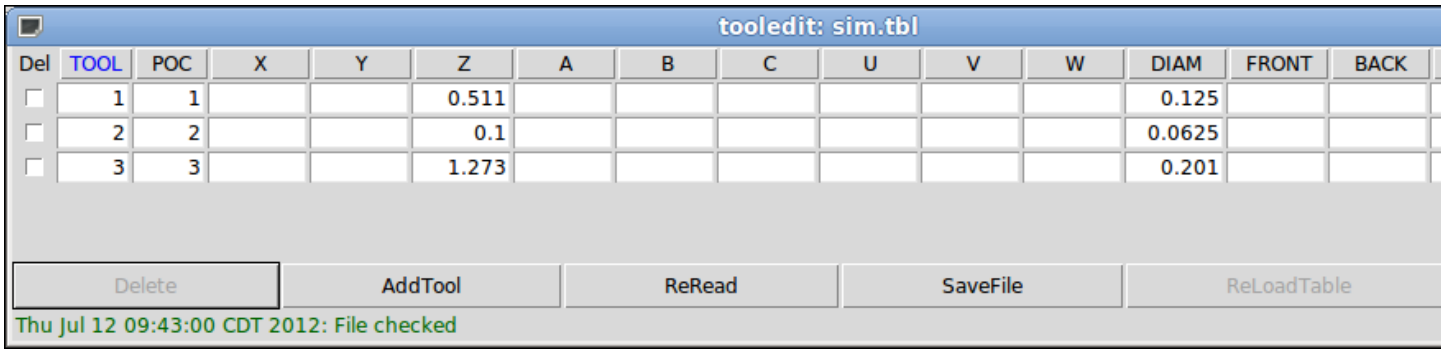



Figure 4.5: Inside Profile

## 4.2 Tool Edit GUI

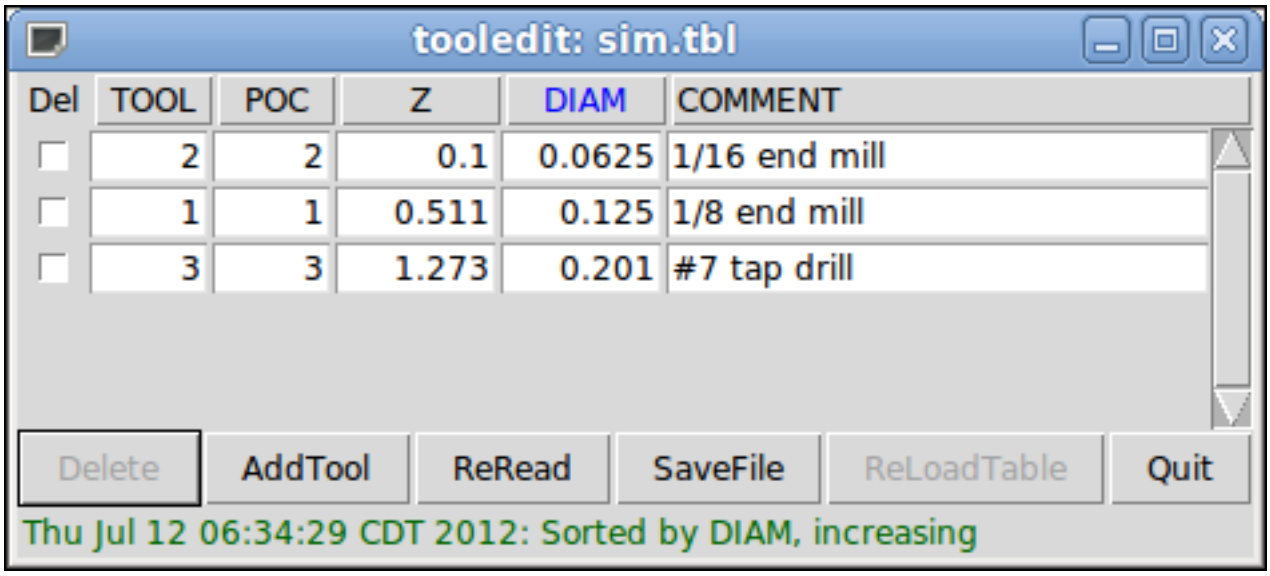
### 4.2.1 Overview




The *tooledit* program can update the tool table file with edited changes by using the SaveFile button. The SaveFile button updates the system file but a separate action is required to update the tool table data used by a running LinuxCNC instance. With the axis GUI, both the file and the current tool table data used by LinuxCNC can be updated with the ReloadTable button. This button is enabled only when the machine is ON and IDLE.

### 4.2.2 Column Sorting

The tool table display can be sorted on any column in ascending order by clicking on the column header. A second click sorts in descending order. Column sorting requires that the machine is configured with the default tcl version >= 8.5.




On Ubuntu Lucid 10.04 tcl/tk8.4 is the default. You can add tcl/tk8.5 with the commands:

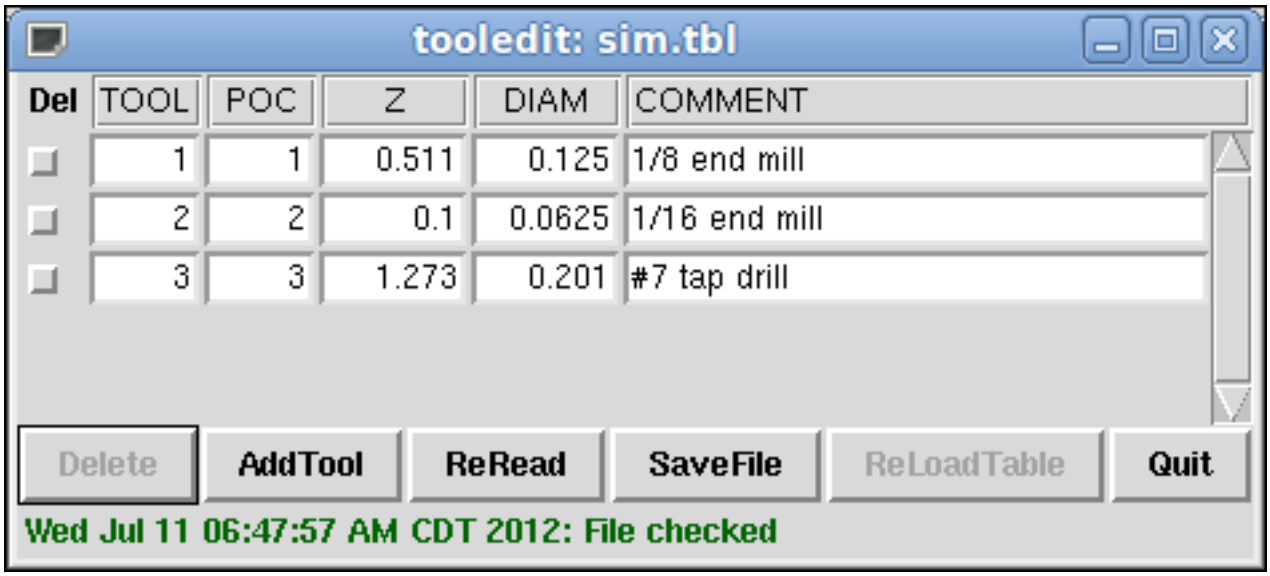
```
sudo apt-get install tcl8.5 tk8.5
```

Depending upon other applications installed on the system, it may be necessary to enable tcl/tk8.5 with the commands:

```
sudo update-alternatives --config tclsh  ;# select the option for tclsh8.5
sudo update-alternatives --config wish   ;# select the option for wish8.5
```

### 4.2.3 Column Selection

By default, the *tooledit* program will display all possible tool table parameter columns. Since few machines use all parameters, the columns displayed can be limited with the following ini file setting:



**INI File Syntax**

```
[DISPLAY]
TOOL_EDITOR = tooledit column_name column_name ...
```

**Example for Z and DIAM columns**

```
[DISPLAY]
TOOL_EDITOR = tooledit Z DIAM
```

### 4.2.4 Stand Alone Use

The *tooledit* program can also be invoked as a standalone program. For example, if the program is in the user PATH, typing *tooledit* will show the usage syntax:

**Stand Alone**

```
tooledit
Usage:
tooledit filename
tooledit [column_1 ... column_n] filename
```

To synchronize a standalone *tooledit* with a running LinuxCNC application, the filename must resolve to the same [EMCIO]TOOL_TABLE filename specified in the LinuxCNC ini file.

When using the program *tooledit* while LinuxCNC is running, gcode command execution or other programs may alter tool table data and the tool table file. File changes are detected by *tooledit* and a message is displayed:

```
Warning: File changed by another process
```

The *tooledit* tool table display can be updated to read the modified file with the ReRead button.

he tool table is specified in the ini file with an entry:

```
[EMCIO]TOOL_TABLE = tool_table_filename
```

The tool table file can be edited with any simple text editor (not a word processor).

The axis GUI can optionally use an ini file setting to specify the tool editor program:

```
[DISPLAY]TOOL_EDITOR = path_to_editor_program
```

By default, the program named *tooledit* is used. This editor supports all tool table parameters, allows addition and deletion of tool entries, and performs a number of validity checks on parameter values.

# Part III

# Configuration

# Chapter 5

# General Info

## 5.1 Integrator Concepts

### 5.1.1 File Locations

LinuxCNC looks for the configuration and G code files in a specific place. The location depends on how you run LinuxCNC.

#### 5.1.1.1 Installed

If your running LinuxCNC from the LiveCD or you installed via a deb and use the configuration picker *LinuxCNC* from the menu LinuxCNC looks in the following directories:

- The LinuxCNC directory is located at */home/user-name/linuxcnc*.
- The Configuration directories are located at */home/user-name/linuxcnc/configs*.
  - Configuration files are located at */home/user-name/linuxcnc/configs/name-of-config*.
- G code files are located at /home/user-name/linuxcnc/nc_files'.

For example for a configuration called Mill and a user name Fred the directory and file structure would look like this.

- */home/fred/linuxcnc*
- */home/fred/linuxcnc/nc_files*
- */home/fred/linuxcnc/configs/mill*
  - */home/fred/linuxcnc/configs/mill/mill.ini*
  - */home/fred/linuxcnc/configs/mill/mill.hal*
  - */home/fred/linuxcnc/configs/mill/mill.var*
  - */home/fred/linuxcnc/configs/mill/tool.tbl*

#### 5.1.1.2 Command Line

If you run LinuxCNC from the command line and specify the name and location of the INI file the file locations can be in a different place. To view the options for running LinuxCNC from the command line run *linuxcnc -h*.

**Note**
Optional locations for some files can be configured in the INI file. See the DISPLAY section and the RS274NGC section.

### 5.1.2 Files

Each configuration directory requires at least the following files:

- An INI file .ini
- A HAL file .hal or HALTCL file .tcl specified in the HAL section of the INI file.

[NOTE]Other files may be required for some GUI's.

Optionally you can also have:

- A Variables file .var
  - If you omit a .var file in a directory but include [RS274NGC] PARAMETER_FILE=somefilename.var, the file will be created for you when LinuxCNC starts.
  - If you omit a .var file and omit the item [RS274NGC] PARAMETER_FILE, a var file named rs274ngc.var will be created when LinuxCNC starts. There may be some confusing messages if [RS274NGC]PARAMETER_FILE is omitted.
- A Tool Table file .tbl if [EMCMOT]TOOL_TABLE has been specified in the INI file. Some configurations do not need a tool table.

### 5.1.3 Stepper Systems

#### 5.1.3.1 Base Period

BASE_PERIOD is the *heartbeat* of your LinuxCNC computer.[1] Every period, the software step generator decides if it is time for another step pulse. A shorter period will allow you to generate more pulses per second, within limits. But if you go too short, your computer will spend so much time generating step pulses that everything else will slow to a crawl, or maybe even lock up. Latency and stepper drive requirements affect the shortest period you can use.

Worst case latencies might only happen a few times a minute, and the odds of bad latency happening just as the motor is changing direction are low. So you can get very rare errors that ruin a part every once in a while and are impossible to troubleshoot.

The simplest way to avoid this problem is to choose a BASE_PERIOD that is the sum of the longest timing requirement of your drive, and the worst case latency of your computer. This is not always the best choice. For example, if you are running a drive with a 20 us direction signal hold time requirement, and your latency test said you have a maximum latency of 11 us , then if you set the BASE_PERIOD to 20+11 = 31 us you get a not-so-nice 32,258 steps per second in one mode and 16,129 steps per second in another mode.

The problem is with the 20 us hold time requirement. That plus the 11 us latency is what forces us to use a slow 31 us period. But the LinuxCNC software step generator has some parameters that let you increase the various times from one period to several. For example, if *steplen* [2] is changed from 1 to 2, then there will be two periods between the beginning and end of the step pulse. Likewise, if *dirhold* [3] is changed from 1 to 3, there will be at least three periods between the step pulse and a change of the direction pin.

If we can use *dirhold* to meet the 20 us hold time requirement, then the next longest time is the 4.5 us high time. Add the 11 us latency to the 4.5 us high time, and you get a minimum period of 15.5 us . When you try 15.5 us , you find that the computer is sluggish, so you settle on 16 us . If we leave *dirhold* at 1 (the default), then the minimum time between step and direction is the 16 us period minus the 11 us latency = 5 us , which is not enough. We need another 15 us . Since the period is 16 us , we need one more period. So we change *dirhold* from 1 to 2. Now the minimum time from the end of the step pulse to the changing direction pin is 5+16=21 us , and we don't have to worry about the drive stepping the wrong direction because of latency.

For more information on stepgen see the stepgen section.

[1] This section refers to using **stepgen**, LinuxCNC's built-in step generator. Some hardware devices have their own step generator and do not use LinuxCNC's built-in one. In that case, refer to your hardware manual.

[2] steplen refers to a parameter that adjusts the performance of LinuxCNC's built-in step generator, *stepgen*, which is a HAL component. This parameter adjusts the length of the step pulse itself. Keep reading, all will be explained eventually.

[3] dirhold refers to a parameter that adjusts the length of the direction hold time.

#### 5.1.3.2 Step Timing

Step Timing and Step Space on some drives are different. In this case the Step point becomes important. If the drive steps on the falling edge then the output pin should be inverted.

### 5.1.4 Servo Systems

#### 5.1.4.1 Basic Operation

Servo systems are capable of greater speed and accuracy than equivalent stepper systems, but are more costly and complex. Unlike stepper systems, servo systems require some type of position feedback device, and must be adjusted or *tuned*, as they don't quite work right out of the box as a stepper system might. These differences exist because servos are a *closed loop* system, unlike stepper motors which are generally run *open loop*. What does *closed loop* mean? Let's look at a simplified diagram of how a servomotor system is connected.

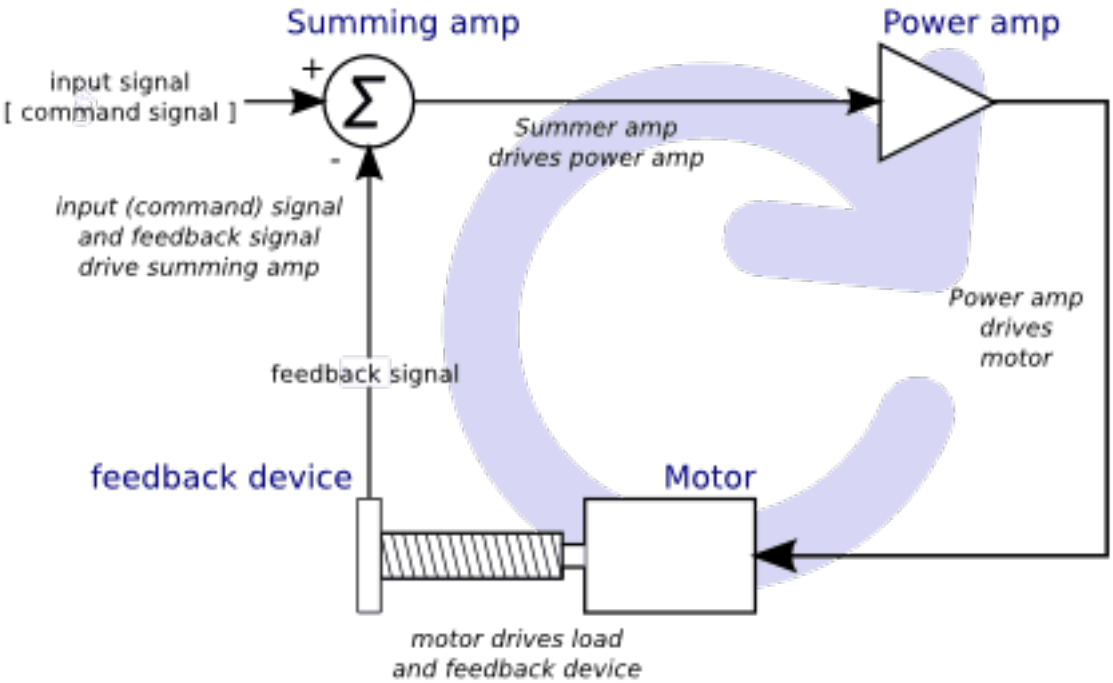



Figure 5.1: Servo Loop

This diagram shows that the input signal (and the feedback signal) drive the summing amplifier, the summing amplifier drives the power amplifier, the power amplifier drives the motor, the motor drives the load (and the feedback device), and the feedback device (and the input signal) drive the motor. This looks very much like a circle (a closed loop) where A controls B, B controls C, C controls D, and D controls A.

If you have not worked with servo systems before, this will no doubt seem a very strange idea at first, especially as compared to more normal electronic circuits, where the inputs proceed smoothly to the outputs, and never go back.[4] If *everything* controls *everything else*, how can that ever work, who's in charge? The answer is that LinuxCNC *can* control this system, but it has to do it by choosing one of several control methods. The control method that LinuxCNC uses, one of the simplest and best, is called PID.

PID stands for Proportional, Integral, and Derivative. The Proportional value determines the reaction to the current error, the Integral value determines the reaction based on the sum of recent errors, and the Derivative value determines the reaction based

[4] If it helps, the closest equivalent to this in the digital world are *state machines*, *sequential machines* and so forth, where what the outputs are doing *now* depends on what the inputs (and the outputs) were doing *before*. If it doesn't help, then nevermind.

on the rate at which the error has been changing. They are three common mathematical techniques that are applied to the task of getting a working process to follow a set point. In the case of LinuxCNC the process we want to control is actual axis position and the set point is the commanded axis position.

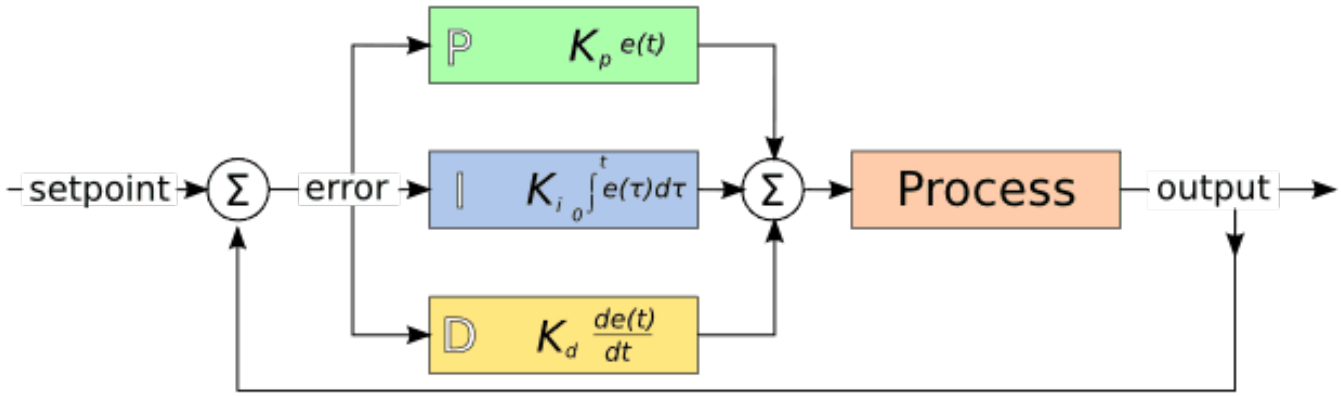



Figure 5.2: PID Loop

By *tuning* the three constants in the PID controller algorithm, the controller can provide control action designed for specific process requirements. The response of the controller can be described in terms of the responsiveness of the controller to an error, the degree to which the controller overshoots the set point and the degree of system oscillation.

#### 5.1.4.2 Proportional term

The proportional term (sometimes called gain) makes a change to the output that is proportional to the current error value. A high proportional gain results in a large change in the output for a given change in the error. If the proportional gain is too high, the system can become unstable. In contrast, a small gain results in a small output response to a large input error. If the proportional gain is too low, the control action may be too small when responding to system disturbances.

In the absence of disturbances, pure proportional control will not settle at its target value, but will retain a steady state error that is a function of the proportional gain and the process gain. Despite the steady-state offset, both tuning theory and industrial practice indicate that it is the proportional term that should contribute the bulk of the output change.

#### 5.1.4.3 Integral term

The contribution from the integral term (sometimes called reset) is proportional to both the magnitude of the error and the duration of the error. Summing the instantaneous error over time (integrating the error) gives the accumulated offset that should have been corrected previously. The accumulated error is then multiplied by the integral gain and added to the controller output.

The integral term (when added to the proportional term) accelerates the movement of the process towards set point and eliminates the residual steady-state error that occurs with a proportional only controller. However, since the integral term is responding to accumulated errors from the past, it can cause the present value to overshoot the set point value (cross over the set point and then create a deviation in the other direction).

#### 5.1.4.4 Derivative term

The rate of change of the process error is calculated by determining the slope of the error over time (i.e. its first derivative with respect to time) and multiplying this rate of change by the derivative gain.

The derivative term slows the rate of change of the controller output and this effect is most noticeable close to the controller set point. Hence, derivative control is used to reduce the magnitude of the overshoot produced by the integral component and improve the combined controller-process stability.

#### 5.1.4.5 Loop tuning

If the PID controller parameters (the gains of the proportional, integral and derivative terms) are chosen incorrectly, the controlled process input can be unstable, i.e. its output diverges, with or without oscillation, and is limited only by saturation or mechanical breakage. Tuning a control loop is the adjustment of its control parameters (gain/proportional band, integral gain/reset, derivative gain/rate) to the optimum values for the desired control response.

#### 5.1.4.6 Manual tuning

A simple tuning method is to first set the I and D values to zero. Increase the P until the output of the loop oscillates, then the P should be set to be approximately half of that value for a *quarter amplitude decay* type response. Then increase I until any offset is correct in sufficient time for the process. However, too much I will cause instability. Finally, increase D, if required, until the loop is acceptably quick to reach its reference after a load disturbance. However, too much D will cause excessive response and overshoot. A fast PID loop tuning usually overshoots slightly to reach the set point more quickly; however, some systems cannot accept overshoot, in which case an *over-damped* closed-loop system is required, which will require a P setting significantly less than half that of the P setting causing oscillation.

### 5.1.5 RTAI

The Real Time Application Interface (RTAI) is used to provide the best Real Time (RT) performance. The RTAI patched kernel lets you write applications with strict timing constraints. RTAI gives you the ability to have things like software step generation which require precise timing.

#### 5.1.5.1 ACPI

The Advanced Configuration and Power Interface (ACPI) has a lot of different functions, most of which interfere with RT performance (for example: power management, CPU power down, CPU frequency scaling, etc). The LinuxCNC kernel (and probably all RTAI-patched kernels) has ACPI disabled. ACPI also takes care of powering down the system after a shutdown has been started, and that's why you might need to push the power button to completely turn off your computer. The RTAI group has been improving this in recent releases, so your LinuxCNC system may shut off by itself after all.

## 5.2 Latency Test

This test is the first test that should be performed on a PC to see if it is able to drive a CNC machine.

Latency is how long it takes the PC to stop what it is doing and respond to an external request. For LinuxCNC the request is BASE_THREAD that makes the periodic *heartbeat* that serves as a timing reference for the step pulses. The lower the latency, the faster you can run the heartbeat, and the faster and smoother the step pulses will be.

Latency is far more important than CPU speed. A lowly Pentium II that responds to interrupts within 10 microseconds each and every time can give better results than the latest and fastest P4 Hyperthreading beast.

The CPU isn't the only factor in determining latency. Motherboards, video cards, USB ports, and a number of other things can hurt the latency. The best way to find out what you are dealing with is to run the RTAI latency test.

Generating step pulses in software has one very big advantage - it's free. Just about every PC has a parallel port that is capable of outputting step pulses that are generated by the software. However, software step pulses also have some disadvantages:

- limited maximum step rate
- jitter in the generated pulses
- loads the CPU

The best way to find out how well your PC will lrun LinuxCNC is to run the HAL latency test. To run the test, open a terminal window (In Ubuntu, from Applications → Accessories → Terminal) and run the following command:

```
latency-test
```

You should see something like this:

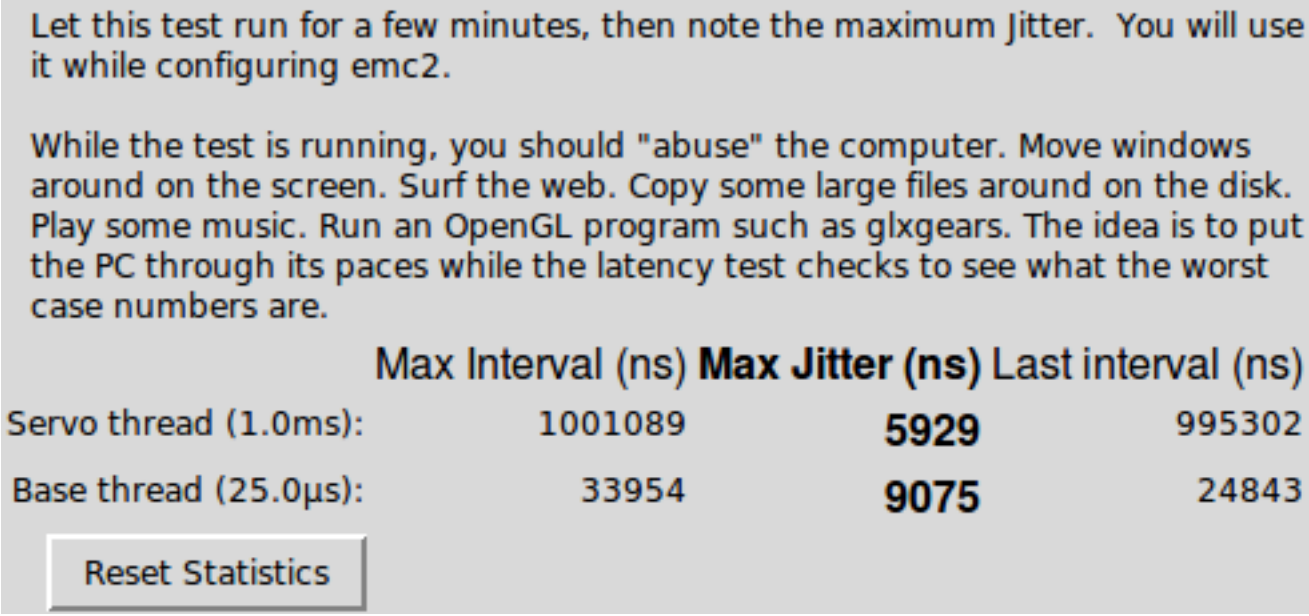



Figure 5.3: HAL Latency Test

While the test is running, you should *abuse* the computer. Move windows around on the screen. Surf the web. Copy some large files around on the disk. Play some music. Run an OpenGL program such as glxgears. The idea is to put the PC through its paces while the latency test checks to see what the worst case numbers are.

**Note**
Do not run LinuxCNC or Stepconf while the latency test is running.

The important numbers are the *max jitter*. In the example above, that is 9075 nanoseconds, or 9.075 microseconds. Record this number, and enter it in Stepconf when it is requested.

In the example above, latency-test only ran for a few seconds. You should run the test for at least several minutes; sometimes the worst case latency doesn't happen very often, or only happens when you do some particular action. For instance, one Intel motherboard worked pretty well most of the time, but every 64 seconds it had a very bad 300 us latency. Fortunately that was fixable, see http://wiki.linuxcnc.org/cgi-bin/wiki.pl?FixingSMIIssues

So, what do the results mean? If your Max Jitter number is less than about 15-20 microseconds (15000-20000 nanoseconds), the computer should give very nice results with software stepping. If the max latency is more like 30-50 microseconds, you can still get good results, but your maximum step rate might be a little disappointing, especially if you use microstepping or have very fine pitch leadscrews. If the numbers are 100 us or more (100,000 nanoseconds), then the PC is not a good candidate for software stepping. Numbers over 1 millisecond (1,000,000 nanoseconds) mean the PC is not a good candidate for LinuxCNC, regardless of whether you use software stepping or not.

Note that if you get high numbers, there may be ways to improve them. Another PC had very bad latency (several milliseconds) when using the onboard video. But a $5 used video card solved the problem.

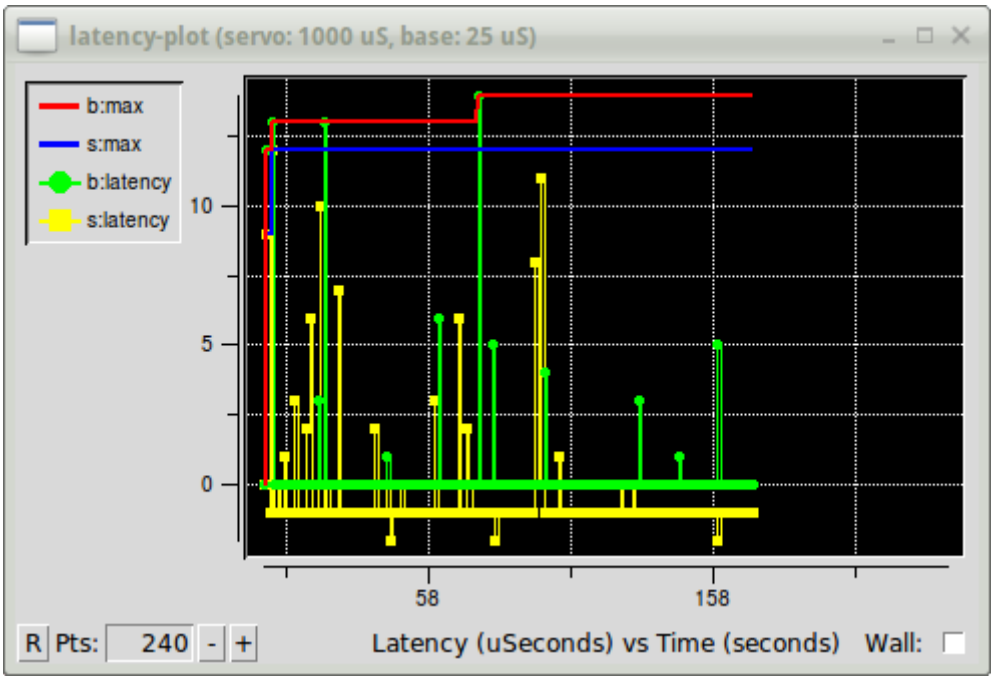
**Note**
LinuxCNC does not require bleeding edge hardware.

For more information on stepper tuning see the Stepper Tuning Chapter.

Additional command line tools are availalbe for examining latency when LinuxCNC is not running.

latency-plot makes a strip chart recording for a base and a servo thread. It may be useful to see spikes in latency when other applications are started or used. Usage:

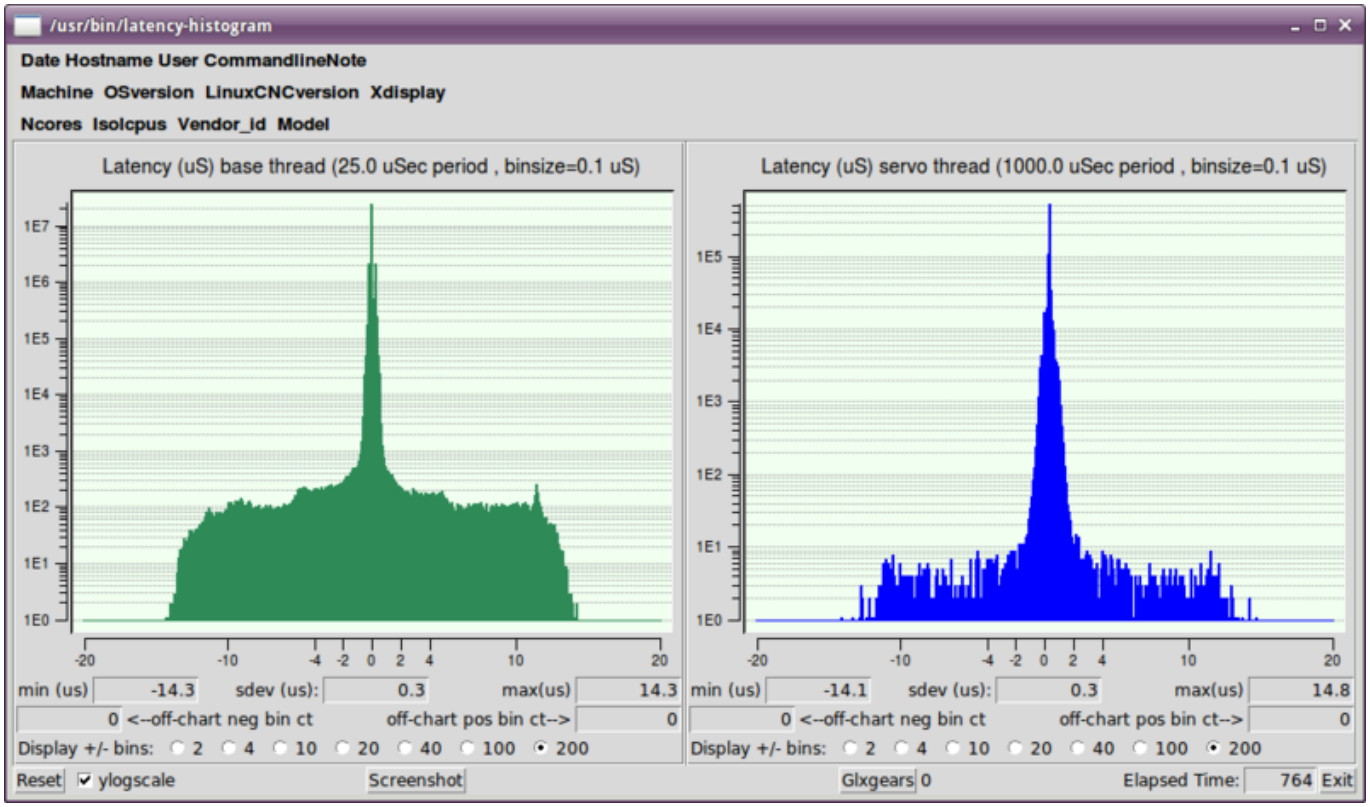
```
latency-plot --help

Usage:
      latency-plot --help | -?      (this)
      latency-plot --hal [Options]

Options:
      --base nS  (base  thread interval, default:   25000)
      --servo nS (servo thread interval, default: 1000000)
      --time mS  (report interval, default: 1000)
      --relative (relative clock time (default))
      --actual   (actual clock time)
```




latency-histogram displays a histogram of latency (jitter) for a base and servo thread. Usage:

```
latency-histogram --help

Usage:
   latency-histogram --help | -?
or
   latency-histogram [Options]

Options:
  --version         (show version and exit)
  --base      nS    (base  thread interval, default:   25000, min:  5000)
  --servo     nS    (servo thread interval, default: 1000000, min: 25000)
  --bbinsize  nS    (base  bin size,  default: 100
  --sbinsize  nS    (servo bin size, default: 100
  --bbins     n     (base  bins, default: 200
  --sbins     n     (servo bins, default: 200
  --logscale  0|1   (y axis log scale, default: 1)
```

```
  --text      note (additional note, default: "" )
  --show           (show count of undisplayed bins)
  --nobase         (servo thread only)
  --verbose        (progress and debug)

Notes:
  Linuxcnc and Hal should not be running, stop with halrun -U.
  Large number of bins and/or small binsizes will slow updates.
  For single thread, specify --nobase (and options for servo thread).
  Measured latencies outside of the +/- bin range are reported
  with special end bars.  Use --show to show count for
  the off-chart [pos|neg] bin
```




## 5.3 Starting LinuxCNC

### 5.3.1 Running LinuxCNC

LinuxCNC is started with the script file *linuxcnc*.

```
linuxcnc [options] [<ini-file>]
```

LINUXCNC SCRIPT OPTIONS

- *-v* = verbose - prints info as it works
- *-d* = echoes script commands to screen for debugging

If the linuxcnc script is passed an ini file it reads the ini file and starts LinuxCNC. The ini file [HAL] section specifies the order of loading up HAL files if more than one is used. Once the HAL=xxx.hal files are loaded then the GUI is loaded then the POSTGUI=.xxx.hal file is loaded. If you create PyVCP or GladeVCP objects with HAL pins you must use the postgui HAL file to make any connections to those pins. See the [HAL] section of the INI configuration for more information.

#### 5.3.1.1 Configuration Selector

If no ini file is passed to the linuxcnc script it loads the configuration selector so you can choose and save a sample configuration. Once a sample configuration has been saved it can be modified to suit your application. The configuration files are saved in linuxcnc/configs directory.

images/configuration-selector.png

Figure 5.4: Configuration Selector

## 5.4 Stepper Tuning

### 5.4.1 Getting the most out of Software Stepping

Generating step pulses in software has one very big advantage - it's free. Just about every PC has a parallel port that is capable of outputting step pulses that are generated by the software. However, software step pulses also have some disadvantages:

- limited maximum step rate
- jitter in the generated pulses
- loads the CPU

This chapter has some steps that can help you get the best results from software generated steps.

#### 5.4.1.1 Run a Latency Test

Run the latency test as described in the Latency Test chapter.

While the test is running, you should *abuse* the computer. Move windows around on the screen. Surf the web. Copy some large files around on the disk. Play some music. Run an OpenGL program such as glxgears. The idea is to put the PC through its paces while the latency test checks to see what the worst case numbers are.

The last number in the column labeled *ovl max* is the most important. Write it down - you will need it later. It contains the worst latency measurement during the entire run of the test. In the example above, that is 10636 nano-seconds, or 10.6 micro-seconds, which is excellent. However the example only ran for a few seconds (it prints one line every second). You should run the test for at least several minutes; sometimes the worst case latency doesn't happen very often, or only happens when you do some particular action. I had one Intel motherboard that worked pretty well most of the time, but every 64 seconds it had a very bad 300 us latency. Fortunately that is fixable, see FixingDapperSMIIssues in the wiki found at wiki.linuxcnc.org.

So, what do the results mean? If your *ovl max* number is less than about 15-20 microseconds (15000-20000 nanoseconds), the computer should give very nice results with software stepping. If the max latency is more like 30-50 microseconds, you can still get good results, but your maximum step rate might be a little disappointing, especially if you use microstepping or have very fine pitch leadscrews. If the numbers are 100 us or more (100,000 nanoseconds), then the PC is not a good candidate for software stepping. Numbers over 1 millisecond (1,000,000 nanoseconds) mean the PC is not a good candidate for EMC, regardless of whether you use software stepping or not.

Note that if you get high numbers, there may be ways to improve them. For example, one PC had very bad latency (several milliseconds) when using the onboard video. But a $5 used Matrox video card solved the problem - EMC does not require bleeding edge hardware.

#### 5.4.1.2 Figure out what your drives expect

Different brands of stepper drives have different timing requirements on their step and direction inputs. So you need to dig out (or Google for) the data sheet that has your drive's specs.

From the Gecko G202 manual:

```
Step Frequency: 0 to 200 kHz
Step Pulse "0" Time: 0.5 us min (Step on falling edge)
Step Pulse "1" Time: 4.5 us min
Direction Setup: 1 us min (20 us min hold time after Step edge)
```

From the Gecko G203V manual:

```
Step Frequency: 0 to 333 kHz
Step Pulse "0" Time: 2.0 us min (Step on rising edge)
Step Pulse "1" Time: 1.0 us min

Direction Setup:
    200 ns (0.2 us) before step pulse rising edge
    200 ns (0.2 us) hold after step pulse rising edge
```

From the Xylotex datasheet:

```
Minimum DIR setup time before rising edge of STEP Pulse 200 ns Minimum
DIR hold time after rising edge of STEP pulse 200 ns
Minimum STEP pulse high time 2.0 us
Minimum STEP pulse low time 1.0 us
Step happens on rising edge
```

Once you find the numbers, write them down too - you need them in the next step.

#### 5.4.1.3 Choose your BASE_PERIOD

BASE_PERIOD is the *heartbeat* of your EMC computer. Every period, the software step generator decides if it is time for another step pulse. A shorter period will allow you to generate more pulses per second, within limits. But if you go too short, your computer will spend so much time generating step pulses that everything else will slow to a crawl, or maybe even lock up. Latency and stepper drive requirements affect the shortest period you can use, as we will see in a minute.

Let's look at the Gecko example first. The G202 can handle step pulses that go low for 0.5 us and high for 4.5 us, it needs the direction pin to be stable 1 us before the falling edge, and remain stable for 20 us after the falling edge. The longest timing requirement is the 20 us hold time. A simple approach would be to set the period at 20 us. That means that all changes on the STEP and DIR lines are separated by 20 us. All is good, right?

Wrong! If there was ZERO latency, then all edges would be separated by 20 us, and everything would be fine. But all computers have some latency. Latency means lateness. If the computer has 11 us of latency, that means sometimes the software runs as much as 11 us later than it was supposed to. If one run of the software is 11 us late, and the next one is on time, the delay from the first to the second is only 9 us. If the first one generated a step pulse, and the second one changed the direction bit, you just violated the 20 us G202 hold time requirement. That means your drive might have taken a step in the wrong direction, and your part will be the wrong size.

The really nasty part about this problem is that it can be very very rare. Worst case latencies might only happen a few times a minute, and the odds of bad latency happening just as the motor is changing direction are low. So you get very rare errors that ruin a part every once in a while and are impossible to troubleshoot.

The simplest way to avoid this problem is to choose a BASE_PERIOD that is the sum of the longest timing requirement of your drive, and the worst case latency of your computer. If you are running a Gecko with a 20 us hold time requirement, and your latency test said you have a maximum latency of 11 us, then if you set the BASE_PERIOD to 20+11 = 31 us (31000 nano-seconds in the ini file), you are guaranteed to meet the drive's timing requirements.

But there is a tradeoff. Making a step pulse requires at least two periods. One to start the pulse, and one to end it. Since the period is 31 us, it takes 2x31 = 62 us to create a step pulse. That means the maximum step rate is only 16,129 steps per second. Not so good. (But don't give up yet, we still have some tweaking to do in the next section.)

For the Xylotex, the setup and hold times are very short, 200 ns each (0.2 us). The longest time is the 2 us high time. If you have 11 us latency, then you can set the BASE_PERIOD as low as 11+2=13 us. Getting rid of the long 20 us hold time really helps! With a period of 13 us, a complete step takes 2x13 = 26 us, and the maximum step rate is 38,461 steps per second!

But you can't start celebrating yet. Note that 13 us is a very short period. If you try to run the step generator every 13 us, there might not be enough time left to run anything else, and your computer will lock up. If you are aiming for periods of less than 25 us, you should start at 25 us or more, run EMC, and see how things respond. If all is well, you can gradually decrease the period. If the mouse pointer starts getting sluggish, and everything else on the PC slows down, your period is a little too short. Go back to the previous value that let the computer run smoothly.

In this case, sppose you started at 25 us, trying to get to 13 us, but you find that around 16 us is the limit - any less and the computer doesn't respond very well. So you use 16 us. With a 16 us period and 11 us latency, the shortest output time will be 16-11 = 5 us. The drive only needs 2 us, so you have some margin. Margin is good - you don't want to lose steps because you cut the timing too close.

What is the maximum step rate? Remember, two periods to make a step. You settled on 16 us for the period, so a step takes 32 us. That works out to a not bad 31,250 steps per second.

#### 5.4.1.4 Use steplen, stepspace, dirsetup, and/or dirhold

In the last section, we got the Xylotex drive to a 16 us period and a 31,250 step per second maximum speed. But the Gecko was stuck at 31 us and a not-so-nice 16,129 steps per second. The Xylotex example is as good as we can make it. But the Gecko can be improved.

The problem with the G202 is the 20 us hold time requirement. That plus the 11 us latency is what forces us to use a slow 31 us period. But the LinuxCNC software step generator has some parameters that let you increase the various time from one period to several. For example, if steplen is changed from 1 to 2, then it there will be two periods between the beginning and end of the step pulse. Likewise, if dirhold is changed from 1 to 3, there will be at least three periods between the step pulse and a change of the direction pin.

If we can use dirhold to meet the 20 us hold time requirement, then the next longest time is the 4.5 us high time. Add the 11 us latency to the 4.5 us high time, and you get a minimum period of 15.5 us. When you try 15.5 us, you find that the computer is sluggish, so you settle on 16 us. If we leave dirhold at 1 (the default), then the minimum time between step and direction is the 16 us period minus the 11 us latency = 5 us, which is not enough. We need another 15 us. Since the period is 16 us, we need one more period. So we change dirhold from 1 to 2. Now the minimum time from the end of the step pulse to the changing direction pin is 5+16=21 us, and we don't have to worry about the Gecko stepping the wrong direction because of latency.

If the computer has a latency of 11 us, then a combination of a 16 us base period, and a dirhold value of 2 ensures that we will always meet the timing requirements of the Gecko. For normal stepping (no direction change), the increased dirhold value has no effect. It takes two periods totalling 32 us to make each step, and we have the same 31,250 step per second rate that we got with the Xylotex.

The 11 us latency number used in this example is very good. If you work through these examples with larger latency, like 20 or 25 us, the top step rate for both the Xylotex and the Gecko will be lower. But the same formulas apply for calculating the optimum BASE_PERIOD, and for tweaking dirhold or other step generator parameters.

#### 5.4.1.5 No Guessing!

For a fast AND reliable software based stepper system, you cannot just guess at periods and other configuration paremeters. You need to make measurements on your computer, and do the math to ensure that your drives get the signals they need.

To make the math easier, I've created an Open Office spreadsheet http://wiki.linuxcnc.org/uploads/StepTimingCalculator.ods You enter your latency test result and your stepper drive timing requirements and the spreadsheet calculates the optimum BASE_PERIOD. Next, you test the period to make sure it won't slow down or lock up your PC. Finally, you enter the actual period, and the spreadsheet will tell you the stepgen parameter settings that are needed to meet your drive's timing requirements. It also calculates the maximum step rate that you will be able to generate.

I've added a few things to the spreadsheet to calculate max speed and stepper electrical calculations.

## 5.5 Stepper Diagnostics

If what you get is not what you expect many times you just got some experience. Learning from the experience increases your understanding of the whole. Diagnosing problems is best done by divide and conquer. By this I mean if you can remove 1/2 of the variables from the equation each time you will find the problem the fastest. In the real world this is not always the case, but it's usually a good place to start.

### 5.5.1 Common Problems

#### 5.5.1.1 Stepper Moves One Step

The most common reason in a new installation for a stepper motor not to move is that the step and direction signals are exchanged. If you press the jog forward and jog backward keys, alternately , and the stepper moves one step each time, and in the same direction, there is your clue.

#### 5.5.1.2 No Steppers Move

Many drives have an enable pin or need a charge pump to enable the output.

#### 5.5.1.3 Distance Not Correct

If you command the axis to move a specific distance and it does not move that distance, then your scale setting is wrong.

### 5.5.2 Error Messages

#### 5.5.2.1 Following Error

The concept of a following error is strange when talking about stepper motors. Since they are an open loop system, there is no position feedback to let you know if you actually are out of range. LinuxCNC calculates if it can keep up with the motion called for, and if not, then it gives a following error. Following errors usually are the result of one of the following on stepper systems.

- FERROR too small
- MIN_FERROR too small
- MAX_VELOCITY too fast
- MAX_ACCELERATION too fast
- BASE_PERIOD set too long
- Backlash added to an axis

Any of the above can cause the real-time pulsing to not be able to keep up the requested step rate. This can happen if you didn't run the latency test long enough to get a good number to plug into the Stepconf Wizard, or if you set the Maximum Velocity or Maximum Acceleration too high.

If you added backlash you need to increase the STEPGEN_MAXACCEL up to double the MAX_ACCELERATION in the AXIS section of the INI file for each axis you added backlash to. LinuxCNC uses "extra acceleration" at a reversal to take up the backlash. Without backlash correction, step generator acceleration can be just a few percent above the motion planner acceleration.

#### 5.5.2.2 RTAPI Error

When you get this error:

```
RTAPI: ERROR: Unexpected realtime delay on task n
```

This error is generated by rtapi based on an indication from RTAI that a deadline was missed. It is usually an indication that the BASE_PERIOD in the [EMCMOT] section of the ini file is set too low. You should run the Latency Test for an extended period of time to see if you have any delays that would cause this problem. If you used the Stepconf Wizard, run it again, and test the Base Period Jitter again, and adjust the Base Period Maximum Jitter on the Basic Machine Information page. You might have to leave the test running for an extended period of time to find out if some hardware causes intermittent problems.

LinuxCNC tracks the number of CPU cycles between invocations of the real-time thread. If some element of your hardware is causing delays or your realtime threads are set too fast you will get this error.

**Note**
This error is only displayed once per session. If you had your BASE_PERIOD too low you could get hundreds of thousands of error messages per second if more than one was displayed.

### 5.5.3 Testing

#### 5.5.3.1 Step Timing

If you are seeing an axis ending up in the wrong location over multiple moves, it is likely that you do not have the correct direction hold times or step timing for your stepper drivers. Each direction change may be losing a step or more. If the motors are stalling, it is also possible you have either the MAX_ACCELERATION or MAX_VELOCITY set too high for that axis.

The following program will test the Z axis configuration for proper setup. Copy the program to your ~/emc2/nc_files directory and name it TestZ.ngc or similar. Zero your machine with Z = 0.000 at the table top. Load and run the program. It will make 200 moves back and forth from 0.5 to 1". If you have a configuration issue, you will find that the final position will not end up 0.500" that the axis window is showing. To test another axis just replace the Z with your axis in the G0 lines.

```
( test program to see if Z axis loses position )
( msg, test 1 of Z axis configuration )
G20 #1000=100 ( loop 100 times )
( this loop has delays after moves )
( tests acc and velocity settings )
o100 while [#1000]
G0 Z1.000
G4 P0.250
G0 Z0.500
G4 P0.250
#1000 = [#1000 - 1]
o100 endwhile
( msg, test 2 of Z axis configuration S to continue)
M1 (stop here)
#1000=100 ( loop 100 times )
( the next loop has no delays after moves )
( tests direction hold times on driver config and also max accel setting )
o101 while [#1000]
G0 Z1.000
G0 Z0.500
#1000 = [#1000 - 1]
o101 endwhile
( msg, Done...Z should be exactly .5" above table )
M2
```

# Chapter 6

# Configuration

## 6.1 Stepper Quickstart

This section assumes you have done a standard install from the Live CD. After installation it is recommended that you connect the computer to the Internet and wait for the update manager to pop up and get the latest updates for LinuxCNC and Ubuntu before continuing.

### 6.1.1 Latency Test

The Latency Test determines how late your computer processor is in responding to a request. Some hardware can interrupt the processing which could cause missed steps when running a CNC machine. This is the first thing you need to do. Follow the instructions here to run the latency test.

### 6.1.2 Sherline

If you have a Sherline several predefined configurations are provided. This is on the main menu CNC/EMC then pick the Sherline configuration that matches yours and save a copy.

### 6.1.3 Xylotex

If you have a Xylotex you can skip the following sections and go straight to the Stepper Config Wizard. LinuxCNC has provided quick setup for the Xylotex machines.

### 6.1.4 Machine Information

Gather the information about each axis of your machine.

Drive timing is in nano seconds. If you're unsure about the timing many popular drives are included in the stepper configuration wizard. Note some newer Gecko drives have different timing than the original one. A list is also on the user maintained LinuxCNC wiki site of more drives.

| Axis | Drive Type | Step Time ns | Step Space ns | Dir. Hold ns | Dir. Setup ns |
|---|---|---|---|---|---|
| X | | | | | |
| Y | | | | | |
| Z | | | | | |
| | | | | | |

### 6.1.5 Pinout Information

Gather the information about the connections from your machine to the PC parallel port.

| Output Pin | Typ. Function | If Different | Input Pin | Typ. Function | If Different |
|---|---|---|---|---|---|
| 1 | E-Stop Out | | 10 | X Limit/Home | |
| 2 | X Step | | 11 | Y Limit/Home | |
| 3 | X Direction | | 12 | Z Limit/Home | |
| 4 | Y Step | | 13 | A Limit/Home | |
| 5 | Y Direction | | 15 | Probe In | |
| 6 | Z Step | | | | |
| 7 | Z Direction | | | | |
| 8 | A Step | | | | |
| 9 | A Direction | | | | |
| 14 | Spindle CW | | | | |
| 16 | Spindle PWM | | | | |
| 17 | Amplifier Enable | | | | |

Note any pins not used should be set to Unused in the drop down box. These can always be changed later by running Stepconf again.

### 6.1.6 Mechanical Information

Gather information on steps and gearing. The result of this is steps per user unit which is used for SCALE in the .ini file.

| Axis | Steps/Rev. | Micro Steps | Motor Teeth | Leadscrew Teeth | Leadscrew Pitch |
|---|---|---|---|---|---|
| X | | | | | |
| Y | | | | | |
| Z | | | | | |
| | | | | | |

- *Steps per revolution* - is how many stepper-motor-steps it takes to turn the stepper motor one revolution. Typical is 200.
- *Micro Steps* - is how many steps the drive needs to move the stepper motor one full step. If microstepping is not used, this number will be 1. If microstepping is used the value will depend on the stepper drive hardware.
- *Motor Teeth and Leadscrew Teeth* - is if you have some reduction (gears, chain, timing belt, etc.) between the motor and the leadscrew. If not, then set these both to 1.
- *Leadscrew Pitch* - is how much movement occurs (in user units) in one leadscrew turn. If you're setting up in inches then it is inches per turn. If you're setting up in millimeters then it is millimeters per turn.

The net result you're looking for is how many CNC-output-steps it takes to move one user unit (inches or mm).

**Example 6.1** Units inches

```
Stepper         = 200 steps per revolution
Drive           =  10 micro steps per step
Motor Teeth     =  20
Leadscrew Teeth =  40
Leadscrew Pitch =   0.2000 inches per turn
```

From the above information, the leadscrew moves 0.200 inches per turn. - The motor turns 2.000 times per 1 leadscrew turn. - The drive takes 10 microstep inputs to make the stepper step once. - The drive needs 2000 steps to turn the stepper one revolution. So the scale needed is:

$$\frac{200 motor steps}{1 motor rev} \times \frac{10 microsteps}{1 motor step} \times \frac{2 motor revs}{1 leadscrew rev} \times \frac{1 leadscrew revs}{0.2000 inch} = \frac{20{,}000 microsteps}{inch}$$

**Example 6.2** Units mm

```
    Stepper         = 200 steps per revolution
    Drive           =   8 micro steps per step
    Motor Teeth     =  30
    Leadscrew Teeth =  90
    Leadscrew Pitch =   5.00 mm per turn
```

From the above information: - The leadscrew moves 5.00 mm per turn. - The motor turns 3.000 times per 1 leadscrew turn. - The drive takes 8 microstep inputs to make the stepper step once. - The drive needs 1600 steps to turn the stepper one revolution. So the scale needed is:

$$\frac{200 \, full \, steps}{1 \, rev} x \frac{8 \, microsteps}{1 \, step} x \frac{3 \, revs}{1 \, leadscrew \, rev} x \frac{1 \, leadscrew \, rev}{5.00 \text{mm}} = \frac{960 \, steps}{1 \, mm}$$

## 6.2 INI Configuration

### 6.2.1 The INI File Components

A typical INI file follows a rather simple layout that includes;

- comments
- sections
- variables

Each of these elements is separated on single lines. Each end of line or newline character creates a new element.

#### 6.2.1.1 Comments

A comment line is started with a ; or a # mark. When the ini reader sees either of these marks at the start a line, the rest of the line is ignored by the software. Comments can be used to describe what an INI element will do.

```
; This is my mill configuration file.
# I set it up on January 12, 2012
```

Comments can also be used to *turn off* a variable. This makes it easier to pick between different variables.

```
DISPLAY = axis
# DISPLAY = touchy
```

In this list, the DISPLAY variable will be set to axis because the other one is commented out. If someone carelessly edits a list like this and leaves two of the lines uncommented, the first one encountered will be used.

Note that inside a variable, the "#" and ";" characters do not denote comments:

```
INCORRECT = value     # and a comment

# Correct Comment
CORRECT = value
```

#### 6.2.1.2 Sections

Related parts of an ini file are separated into sections. A section name is enclosed in brackets like this *[THIS_SECTION]* The order of sections is unimportant. Sections begin at the section name and end at the next section name.

The following sections are used by LinuxCNC:

- *[EMC]* general information
- *[DISPLAY]* settings related to the graphical user interface
- *[FILTER]* settings input filter programs
- *[RS274NGC]* settings used by the g-code interpreter
- *[EMCMOT]* settings used by the real time motion controller
- *[TASK]* settings used by the task controller
- *[HAL]* specifies .hal files
- *[HALUI]* MDI commands used by HALUI
- *[APPLICATIONS]* Other applications to be started by LinuxCNC
- *[TRAJ]* additional settings used by the real time motion controller
- *[AXIS_n]* individual axis variables
- *[EMCIO]* settings used by the I/O Controller

#### 6.2.1.3 Variables

A variable line is made up of a variable name, an equals sign (=), and a value. Everything from the first non-white space character after the = up to the end of the line is passed as the value, so you can embed spaces in string symbols if you want to or need to. A variable name is often called a keyword.

**Variable Example**

```
MACHINE = My Machine
```

A variable line may be extended to multiple lines with a terminal backslash (\) character. A maximum of MAX_EXTEND_LINES (==20) are allowed. There must be no whitespace following the trailing backslash character.

Section identifiers may not be extended to multiple lines.

**Variable with Line extends Example**

```
APP = sim_pin \
ini.0.max_acceleration \
ini.1.max_acceleration \
ini.2.max_acceleration \
ini.0.max_velocity \
ini.1.max_velocity \
ini.2.max_velocity
```

The following sections detail each section of the configuration file, using sample values for the configuration lines.

Variables that are used by LinuxCNC must always use the section names and variable names as shown. In the following example the variable *MACHINE* is assigned the value *My Machine*.

#### 6.2.1.4 Custom Sections and Variables

Most sample configurations use custom sections and variables to put all of the settings into one location for convenience.

To use a custom section variable in your HAL file add the section and variable to the INI file.

**Custom Section Example**

```
[OFFSETS]
OFFSET_1 = 0.1234
```

To add a custom variable to a LinuxCNC section simply include the variable in that section.

**Custom Variable Example**

```
[AXIS_0]
TYPE = LINEAR
...
SCALE = 16000
```

To use the custom variables in your HAL file put the section and variable name in place of the value.

**HAL Example**

```
setp offset.1.offset [OFFSETS]OFFSET_1
setp stepgen.0.position-scale [AXIS_0]SCALE
```

**Note**
The value stored in the variable must match the type specified by the component pin.

#### 6.2.1.5 Include Files

An INI file may include the contents of another file by using a #INCLUDE directive.

**#INCLUDE Format**

```
#INCLUDE filename
```

The filename can be specified as:

- a file in the same directory as the INI file
- a file located relative to the working directory
- an absolute file name (starts with a /)
- a user-home-relative file name (starts with a ~)

Multiple #INCLUDE directives are supported.

**#INCLUDE Examples**

```
#INCLUDE axis_0.inc
#INCLUDE ../parallel/axis_1.inc
#INCLUDE below/axis_2.inc
#INCLUDE /home/myusername/myincludes/display.inc
#INCLUDE ~/linuxcnc/myincludes/rs274ngc.inc
```

The #INCLUDE directives are supported for one level of expansion only — an included file may not include additional files. The recommended file extension is .inc. Do not use a file extension of .ini for included files.

### 6.2.2 INI File Sections

#### 6.2.2.1 [EMC] Section

- *VERSION = $Revision: 1.3 $* - The version number for the INI file. The value shown here looks odd because it is automatically updated when using the Revision Control System. It's a good idea to change this number each time you revise your file. If you want to edit this manually just change the number and leave the other tags alone.
- *MACHINE = My Controller* - This is the name of the controller, which is printed out at the top of most graphical interfaces. You can put whatever you want here as long as you make it a single line long.
- *DEBUG = 0* - Debug level 0 means no messages will be printed when LinuxCNC is run from a terminal. Debug flags are usually only useful to developers. See src/emc/nml_intf/debugflags.h for other settings.

#### 6.2.2.2 [DISPLAY] Section

Different user interface programs use different options, and not every option is supported by every user interface. The main two interfaces for LinuxCNC are AXIS and Touchy. There are several newer interfaces, like gmoccapy and gscreen. Axis is an interface for use with normal computer and monitor, Touchy is for use with touch screens. Gmoccapy can be used both ways and offers also many connections for hardware controls. Descriptions of the interfaces are in the Interfaces section of the User Manual.

- *DISPLAY = axis* - The name of the user interface to use. Valid options may include: *axis*, *touchy*, *gmoccapy*, *gscreen*, *keystick*, *mini*, *tklinuxcnc*, *xemc*,
- *POSITION_OFFSET = RELATIVE* - The coordinate system (RELATIVE or MACHINE) to show on the DRO when the user interface starts. The RELATIVE coordinate system reflects the G92 and G5x coordinate offsets currently in effect.
- *POSITION_FEEDBACK = COMMANDED* - The coordinate value (COMMANDED or ACTUAL) to show on the DRO when the user interface starts. In Axis this can be changed from the View menu. The COMMANDED position is the position requested by LinuxCNC. The ACTUAL position is the feedback position of the motors if they have feedback like most servo systems. Typically the COMMANDED value is used.
- *MAX_FEED_OVERRIDE = 1.2* - The maximum feed override the user may select. 1.2 means 120% of the programmed feed rate.
- *MIN_SPINDLE_OVERRIDE = 0.5* - The minimum spindle override the user may select. 0.5 means 50% of the programmed spindle speed. (This is used to set the minimum spindle speed).
- *MAX_SPINDLE_OVERRIDE = 1.0* - The maximum spindle override the user may select. 1.0 means 100% of the programmed spindle speed.
- *DEFAULT_SPINDLE_SPEED = 100* - The default spindle RPM when the spindle is started in manual mode. if this setting is not present, this defaults to 1 RPM for AXIS and 300 RPM for gmoccapy.
- *PROGRAM_PREFIX = ~/linuxcnc/nc_files* - The default location for g-code files and the location for user-defined M-codes. This location is searched for the file name before the subroutine path and user M path if specified in the [RS274NGC] section.
- *INTRO_GRAPHIC = emc2.gif* - The image shown on the splash screen.
- *INTRO_TIME = 5* - The maximum time to show the splash screen, in seconds.
- *CYCLE_TIME = 0.05* - Cycle time in seconds that display will sleep between polls.

**Note**
The following [DISPLAY] items are used by GladeVCP, see the embedding a tab section of the GladeVCP Chapter.

- *EMBED_TAB_NAME=GladeVCP demo*

•

**Note**
The following [DISPLAY] items are for the AXIS interface only, see the AXIS GUI document for details. Many of them are used also from gmoccapy, see the gmoccapy document for details.

- *DEFAULT_LINEAR_VELOCITY = .25* - The default velocity for linear jogs, in , machine units per second.
- *MIN_VELOCITY = .01* - The approximate lowest value the jog slider.
- *MAX_LINEAR_VELOCITY = 1.0* - The maximum velocity for linear jogs, in machine units per second.
- *MIN_LINEAR_VELOCITY = .01* - The approximate lowest value the jog slider.
- *DEFAULT_ANGULAR_VELOCITY = .25* - The default velocity for angular jogs, in machine units per second.
- *MIN_ANGULAR_VELOCITY = .01* - The approximate lowest value the angular jog slider.
- *MAX_ANGULAR_VELOCITY = 1.0* - The maximum velocity for angular jogs, in machine units per second.
- *INCREMENTS = 1 mm, .5 in, . . .* - Defines the increments available for incremental jogs. The INCREMENTS can be used to override the default. The values can be decimal numbers (e.g., 0.1000) or fractional numbers (e.g., 1/16), optionally followed by a unit (cm, mm, um, inch, in or mil). If a unit is not specified the machine unit is assumed. Metric and imperial distances may be mixed: INCREMENTS = 1 inch, 1 mil, 1 cm, 1 mm, 1 um is a valid entry.
- *GRIDS = 10 mm, 1 in, . . .* - Defines the preset values for grid lines. The value is interpreted the same way as *INCREMENTS*.
- *OPEN_FILE = /full/path/to/file.ngc* - The file to show in the preview plot when AXIS starts. Use a blank string "" and no file will be loaded at start up. gmoccapy will not use this setting, as it offers a coresponding entry on its settings page.
- *EDITOR = gedit* - The editor to use when selecting File > Edit to edit the G code from the AXIS menu. This must be configured for this menu item to work. Another valid entry is gnome-terminal -e vim. This entry does not apply to gmoccapy, as gmoccapy has an integrated editor.
- *TOOL_EDITOR = tooledit* - The editor to use when editing the tool table (for example by selecting "File > Edit tool table. . . " in Axis). Other valid entries are "gedit", "gnome-terminal -e vim", and "gvim". This entry does not apply to gmoccapy, as gmoccapy has an integrated editor.
- *PYVCP = /filename.xml* - The PyVCP panel description file. See the PyVCP Chapter for more information.
- *LATHE = 1* - This displays in lathe mode with a top view and with Radius and Diameter on the DRO.
- *GEOMETRY = XYZABCUVW* - Controls the preview and backplot of rotary motion. This item consists of a sequence of axis letters, optionally preceded by a "-" sign. Only axes defined in *[TRAJ]AXES* should be used. This sequence specifies the order in which the effect of each axis is applied, with a "-" inverting the sense of the rotation. The proper GEOMETRY string depends on the machine configuration and the kinematics used to control it. The example string GEOMETRY=XYZBCUVW is for a 5-axis machine where kinematics causes UVW to move in the coordinate system of the tool and XYZ to move in the coordinate system of the material. The order of the letters is important, because it expresses the order in which the different transformations are applied. For example rotating around C then B is different than rotating around B then C. Geometry has no effect without a rotary axis.
- *ARCDIVISION = 64* - Set the quality of preview of arcs. Arcs are previewed by dividing them into a number of straight lines; a semicircle is divided into **ARCDIVISION** parts. Larger values give a more accurate preview, but take longer to load and result in a more sluggish display. Smaller values give a less accurate preview, but take less time to load and may result in a faster display. The default value of 64 means a circle of up to 3 inches will be displayed to within 1 mil (.03%).
- *MDI_HISTORY_FILE =* - The name of a local MDI history file. If this is not specified Axis will save the MDI history in **.axis_mdi_history** in the user's home directory. This is useful if you have multiple configurations on one computer.
- *USER_COMMAND_FILE = mycommands.py* — The name of an optional, configuration-specific python file sourced by the axis gui instead of the user-specific file `~/.axisrc`.

**Note**
The following [DISPLAY] item is used by the TKLinuxCNC interface only.

- *HELP_FILE = tklinucnc.txt* - Path to help file.

#### 6.2.2.3 [FILTER] Section

AXIS and gmoccapy have the ability to send loaded files through a filter program. This filter can do any desired task: Something as simple as making sure the file ends with M2, or something as complicated as detecting whether the input is a depth image, and generating g-code to mill the shape it defines. The [FILTER] section of the ini file controls how filters work. First, for each type of file, write a PROGRAM_EXTENSION line. Then, specify the program to execute for each type of file. This program is given the name of the input file as its first argument, and must write RS274NGC code to standard output. This output is what will be displayed in the text area, previewed in the display area, and executed by LinuxCNC when Run.

- *PROGRAM_EXTENSION = .extension Description*

If your post processor outputs files in all caps you might want to add the following line:

- *PROGRAM_EXTENSION = .NGC XYZ Post Processor*

The following lines add support for the image-to-G code converter included with LinuxCNC.

- *PROGRAM_EXTENSION = .png,.gif,.jpg Greyscale Depth Image*
  - *png = image-to-gcode*
  - *gif = image-to-gcode*
  - *jpg = image-to-gcode*

An example of a custom G code converter located in the linuxcnc directory.

- 'PROGRAM_EXTENSION = .gcode 3D Printer
  - *gcode = /home/mill/linuxcnc/convert.py*

**Note**
The program file assoiated with an extension must have either the full path to the program or be located in a directory that is on the system path.

It is also possible to specify an interpreter:

- *PROGRAM_EXTENSION = .py Python Script*
  - *py = python*

In this way, any Python script can be opened, and its output is treated as g-code. One such example script is available at nc_files/holecircle.py. This script creates g-code for drilling a series of holes along the circumference of a circle. Many more g-code generators are on the LinuxCNC Wiki site http://wiki.linuxcnc.org/.

If the environment variable AXIS_PROGRESS_BAR is set, then lines written to stderr of the form

- *FILTER_PROGRESS=%d*

sets the AXIS progress bar to the given percentage. This feature should be used by any filter that runs for a long time.

Python filters should use the print function to output the result to Axis.

This example program filters a file and adds a W axis to match the Z axis. It depends on there being a space between each axis word to work.

```
#! /usr/bin/env python

import sys

def main(argv):

  openfile = open(argv[0], 'r')
  file_in = openfile.readlines()
  openfile.close()

  file_out = []
  for line in file_in:
    # print line
    if line.find('Z') != -1:
      words = line.rstrip('\n')
      words = words.split(' ')
      newword = ''
      for i in words:
        if i[0] == 'Z':
          newword = 'W'+ i[1:]
      if len(newword) > 0:
        words.append(newword)
        newline = ' '.join(words)
        file_out.append(newline)
    else:
      file_out.append(line)
  for item in file_out:
    print "%s" % item

if __name__ == "__main__":
   main(sys.argv[1:])
```

#### 6.2.2.4 [RS274NGC] Section

- *PARAMETER_FILE = myfile.var* - The file located in the same directory as the ini file which contains the parameters used by the interpreter (saved between runs).
- *ORIENT_OFFSET = 0* - A float value added to the R word parameter of an M19 Orient Spindle operation. Used to define an arbitrary zero position regardless of encoder mount orientation.
- *RS274NGC_STARTUP_CODE = G17 G20 G40 G49 G64 P0.001 G80 G90 G92 G94 G97 G98* - A string of NC codes that the interpreter is initialized with. This is not a substitute for specifying modal g-codes at the top of each ngc file, because the modal codes of machines differ, and may be changed by g-code interpreted earlier in the session.
- *SUBROUTINE_PATH = ncsubroutines:/tmp/testsubs:lathesubs:millsubs* - Specifies a colon (:) separated list of up to 10 directories to be searched when single-file subroutines are specified in gcode. These directories are searched after searching [DISPLAY]PROGRAM_PREFIX (if it is specified) and before searching [WIZARD]WIZARD_ROOT (if specified). The paths are searched in the order that they are listed. The first matching subroutine file found in the search is used. Directories are specified relative to the current directory for the ini file or as absolute paths. The list must contain no intervening whitespace.
- *USER_M_PATH = myfuncs:/tmp/mcodes:experimentalmcodes* - Specifies a list of colon (:) separated directories for user defined functions. Directories are specified relative to the current directory for the ini file or as absolute paths. The list must contain no intervening whitespace.

  A search is made for each possible user defined function, typically (M100-M199). The search order is:

1. [DISPLAY]PROGRAM_PREFIX (if specified)
2. If [DISPLAY]PROGRAM_PREFIX is not specified, search the default location: nc_files
3. Then search each directory in the list [RS274NGC]USER_M_PATH
   The first executable M1xx found in the search is used for each M1xx.

**Note**
The maximum number of USER_M_PATH directories is defined at compile time (typ: *USER_DEFINED_FUNCTION_MAX_DIRS == 5*).

**Note**
[WIZARD]WIZARD_ROOT is a valid search path but the Wizard has not been fully implemented and the results of using it are unpredictable.

#### 6.2.2.5 [EMCMOT] Section

This section is a custom section and is not used by LinuxCNC directly. Most configurations use values from this section to load the motion controller. For more information on the motion controller see the Motion Section.

- *EMCMOT = motmod* - the motion controller name is typically used here.
- *BASE_PERIOD = 50000* - the *Base* task period in nanoseconds.
- *SERVO_PERIOD = 1000000* - This is the "Servo" task period in nanoseconds.
- *TRAJ_PERIOD = 100000* - This is the *Trajectory Planner* task period in nanoseconds.
- *COMM_TIMEOUT = 1.0* - Number of seconds to wait for Motion (the realtime part of the motion controller) to acknowledge receipt of messages from Task (the non-realtime part of the motion controller).

#### 6.2.2.6 [TASK] Section

- *TASK = milltask* - Specifies the name of the *task* executable. The *task* executable does various things, such as communicate with the UIs over NML, communicate with the realtime motion planner over non-HAL shared memory, and interpret gcode. Currently there is only one task executable that makes sense for 99.9% of users, milltask.
- *CYCLE_TIME = 0.010* - The period, in seconds, at which TASK will run. This parameter affects the polling interval when waiting for motion to complete, when executing a pause instruction, and when accepting a command from a user interface. There is usually no need to change this number.

#### 6.2.2.7 [HAL] section

- *HALFILE = example.hal* - Execute the file *example.hal* at start up. If *HALFILE* is specified multiple times, the files are executed in the order they appear in the ini file. Almost all configurations will have at least one *HALFILE*, and stepper systems typically have two such files, one which specifies the generic stepper configuration (*core_stepper.hal*) and one which specifies the machine pin out (*xxx_pinout.hal*). HALFILES are found using a search. If the named file is found in the directory containing the ini file, it is used. If the named file is not found in this ini file directory, a search is made using a system library of halfiles. A HALFILE may also be specified as an absolute path (when the name starts with a / character). Absolute paths are not recommended as their use may limit relocation of configurations.
- *HALFILE = texample.tcl [arg1 [arg2] . . . ]]* - Execute the tcl file *texample.tcl* at start up with arg1, arg2, etc as ::argv list. Files with a .tcl suffix are processed as above but use haltcl for processing See the HALTCL Chapter for more information.

- *HALFILE = LIB:sys_example.hal* - Execute the system library file *sys_example.hal* at start up. Explicit use of the LIB: prefix causes use of the system library HALFILE without searching the ini file directory.
- *HALFILE = LIB:sys_texample.tcl [arg1 [arg2 . . . ]]* - Execute the system library file *sys_texample.tcl* at start up. Explicit use of the LIB: prefix causes use of the system library HALFILE without searching the ini file directory.

  HALFILE items specify files that loadrt Hal components and make signal connections between component pins. Common mistakes are 1) omission of the addf statement needed to add a component's function(s) to a thread, 2) incomplete signal (net) specifiers. Omission of required addf statements is almost always an error. Signals usually include one or more input connections and a single output (but both are not strictly required). A system library file is provided to make checks for these conditions and report to stdout and in a popup gui:

```
HALFILE = LIB:halcheck.tcl [ nopopup ]
```

**Note**
The LIB:halcheck.tcl line should be the last [HAL]HALFILE. Specify the *nopopup* option to suppress the popup message and allow immediate starting. Connections made using a POSTGUI_HALFILE are not checked.

- *TWOPASS = ON* - Use twopass processing for loading HAL components. With TWOPASS processing, [HAL]HALFILE= lines are processed in two passes. In the first pass (pass0), all HALFILES are read and multiple appearances of loadrt and loadusr commands are accumulated. These accumulated load commands are executed at the end of pass0. This accumulation allows load lines to be specified more than once for a given component (provided the names= names used are unique on each use). In the second pass (pass1), the HALFILES are reread and all commands except the previously executed load commands are executed.
- *TWOPASS = nodelete verbose* - The TWOPASS feature can be activated with any non-null string including the keywords verbose and nodelete. The verbose keyword causes printing of details to stdout. The nodelete keyword preserves temporary files in /tmp.

  For more information see the Hal TWOPASS chapter.
- *HALCMD = command* - Execute *command* as a single HAL command. If *HALCMD* is specified multiple times, the commands are executed in the order they appear in the ini file. *HALCMD* lines are executed after all *HALFILE* lines.
- *SHUTDOWN = shutdown.hal* - Execute the file *shutdown.hal* when LinuxCNC is exiting. Depending on the hardware drivers used, this may make it possible to set outputs to defined values when LinuxCNC is exited normally. However, because there is no guarantee this file will be executed (for instance, in the case of a computer crash) it is not a replacement for a proper physical e-stop chain or other protections against software failure.
- *POSTGUI_HALFILE = example2.hal* - Execute *example2.hal* after the GUI has created its HAL pins. Some GUIs create hal pins and support the use of a postgui halfile to use them. GUIs that support postgui halfiles include Touchy, Axis, Gscreen, and gmoccapy.

  `See section <<sec:pyvcp-with-axis,pyVCP with Axis>> Section for more information.`

- *HALUI = halui* - adds the HAL user interface pins. For more information see the HAL User Interface chapter.

#### 6.2.2.8 [HALUI] section

- *MDI_COMMAND = G53 G0 X0 Y0 Z0* - An MDI command can be executed by using halui.mdi-command-00. Increment the number for each command listed in the [HALUI] section.

#### 6.2.2.9 [APPLICATIONS] Section

LinuxCNC can start other applications before the specified gui is started. The applications can be started after a specified delay to allow for gui-dependent actions (like creating gui-specific hal pins).

- *DELAY = value* - seconds to wait beore starting other applications. A delay may be needed if an application has dependencies on [HAL]POSTGUI_HALFILE actions or gui-created hal pins (default DELAY=0).
- *APP = appname [arg1 [arg2 . . . ]]* - Application to be started. This specification can be included multiple times. The appname can be explicitly named as an absolute or tilde specified filename (first character is / or ~), a relative filename (first characters of filename are ./), or as a file in the inifile directory. If no executable file is found using these names, then the user search PATH is used to find the application.

  `Examples:`

  - Simulate inputs to hal pins for testing (using sim_pin — a simple gui to set inputs to parameters, unconnected pins, or signals with no writers):

    ```
    APP = sim_pin motion.probe-input halui.abort motion.analog-in-00
    ```

  - Invoke halshow with a previuosly saved watchlist. Since linuxcnc sets the working directory to the directory for the inifile, you can refer to files in that directory (example: my.halshow):

    ```
    APP = halshow my.halshow
    ```

  - Alternatively, a watchlist file identified with a full pathname could be specified:

    ```
    APP = halshow ~/saved_shows/spindle.halshow
    ```

  - Open halscope using a previously saved configuration:

    ```
    APP = halscope -i my.halscope
    ```

#### 6.2.2.10 [TRAJ] Section

**Warning**
The new Trajectory Planner (TP) is on by default.
If you have no TP settings in your [TRAJ] section - LinuxCNC defaults to:
ARC_BLEND_ENABLE = 1
ARC_BLEND_FALLBACK_ENABLE = 0
ARC_BLEND_OPTIMIZATION_DEPTH = 50
ARC_BLEND_GAP_CYCLES = 4
ARC_BLEND_RAMP_FREQ = 100

The [TRAJ] section contains general parameters for the trajectory planning module in *motion*.

- *ARC_BLEND_ENABLE = 1* - Turn on new TP. If set to 0 TP uses parabolic blending (1 segment look ahead.) Default value 1.
- *ARC_BLEND_FALLBACK_ENABLE = 0* - Optionally fall back to parabolic blends if the estimated speed is faster. However, this estimate is rough, and it seems that just disabling it gives better performance. Default value 0.
- *ARC_BLEND_OPTIMIZATION_DEPTH = 50* - Look ahead depth in number of segments.

  To expand on this a bit, you can choose this value somewhat arbitrarily. Here's a formula to estimate how much *depth* you need for a particular config:

  # n = v_max / (2.0 * a_max * t_c) # where: # n = optimization depth # v_max = max axis velocity (UU / sec) # a_max = max axis acceleration (UU / sec) # t_c = servo period (seconds)

So, a machine with a maximum axis velocity of 10 IPS, a max acceleration of 100 IPS^2, and a servo period of 0.001 sec would need:

10 / (2.0 * 100 * 0.001) = 50 segments to always reach maximum velocity along the fastest axis.

In practice, this number isn't that important to tune, since the look ahead rarely needs the full depth unless you have lots of very short segments. If during testing, you notice strange slowdowns and can't figure out where they come from, first try increasing this depth using the formula above.

If you still see strange slowdowns, it may be because you have short segments in the program. If this is the case, try adding a small tolerance for Naive CAM detection. A good rule of thumb is this:

# min_length ~= v_req * t_c # where: # v_req = desired velocity in UU / sec # t_c = servo period (seconds)

If you want to travel along a path at 1 IPS = 60 IPM, and your servo period is 0.001 sec, then any segments shorter than min_length will slow the path down. If you set Naive CAM tolerance to around this min length, overly short segments will be combined together to eliminate this bottleneck. Of course, setting the tolerance too high means big path deviations, so you have to play with it a bit to find a good value. I'd start at 1/2 of the min_length, then work up as needed.

- *ARC_BLEND_GAP_CYCLES = 4* How short the previous segment must be before the trajectory planner *consumes* it.

  Often, a circular arc blend will leave short line segments in between the blends. Since the geometry has to be circular, we can't blend over all of a line if the next one is a little shorter. Since the trajectory planner has to touch each segment at least once, it means that very tiny segments will slow things down significantly. My fix to this way to "consume" the short segment by making it a part of the blend arc. Since the line+blend is one segment, we don't have to slow down to hit the very short segment. Likely, you won't need to touch this setting.

- *ARC_BLEND_RAMP_FREQ = 20* - This is a *cutoff* frequency for using ramped velocity.

  *Ramped velocity* in this case just means constant acceleration over the whole segment. This is less optimal than a trapezoidal velocity profile, since the acceleration is not maximized. However, if the segment is short enough, there isn't enough time to accelerate much before we hit the next segment. Recall the short line segments from the previous example. Since they're lines, there's no cornering acceleration, so we're free to accelerate up to the requested speed. However, if this line is between two arcs, then it will have to quickly decelerate again to be within the maximum speed of the next segment. This means that we have a spike of acceleration, then a spike of deceleration, causing a large jerk, for very little performance gain. This setting is a way to eliminate this jerk for short segments.

  Basically, if a segment will complete in less time than 1 / ARC_BLEND_RAMP_FREQ, we don't bother with a trapezoidal velocity profile on that segment, and use constant acceleration. (Setting ARC_BLEND_RAMP_FREQ = 1000 is equivalent to always using trapezoidal acceleration, if the servo loop is 1kHz).

  You can characterize the worst-case loss of performance by comparing the velocity that a trapezoidal profile reaches vs. the ramp:

  # v_ripple = a_max / (4.0 * f) # where: # v_ripple = average velocity "loss" due to ramping # a_max = max axis acceleration # f = cutoff frequency from INI

  For the aforementioned machine, the ripple for a 20Hz cutoff frequency is 100 / (4 * 20) = 1.25 IPS. This seems high, but keep in mind that it is only a worst-case estimate. In reality , the trapezoidal motion profile is limited by other factors, such as normal acceleration or requested velocity, and so the actual performance loss should be much smaller. Increasing the cutoff frequency can squeeze out more performance, but make the motion rougher due to acceleration discontinuities. A value in the range 20Hz to 200Hz should be reasonable to start.

  Finally, no amount of tweaking will speed up a toolpath with lots of small, tight corners, since you're limited by cornering acceleration.

- *COORDINATES = X Y Z* - The names of the axes being controlled. Only X, Y, Z, A, B, C, U, V, W are valid. Only axes named in *COORDINATES* are accepted in g-code. This has no effect on the mapping from G-code axis names (X- Y- Z-) to joint numbers—for *trivial kinematics*, X is always joint 0, A is always joint 3, and U is always joint 6, and so on. It is permitted to write an axis name twice (e.g., X Y Y Z for a gantry machine) but this has no effect.

- *AXES = 3* - One more than the number of the highest joint number in the system. For an XYZ machine, the joints are numbered 0, 1 and 2; in this case AXES should be 3. For an XYUV machine using *trivial kinematics*, the V joint is numbered 7 and therefore AXES should be 8. For a machine with nontrivial kinematics (e.g., scarakins) this will generally be the number of controlled joints.

- *JOINTS = 3* - (This config variable is used by the Axis and gmoccapy (from release 2.0) GUI only, not by the trajectory planner in the motion controller.) Specifies the number of joints (motors) in the system. For example, an XYZ machine with a single motor for each axis has 3 joints. A gantry machine with one motor on each of two of the axes, and two motors on the third axis, has 4 joints.
- *HOME = 0 0 0* - Coordinates of the homed position of each axis. Again for a fourth axis you will need 0 0 0 0. This value is only used for machines with nontrivial kinematics. On machines with trivial kinematics (mill, lathe, gantry types) this value is ignored.
- *LINEAR_UNITS = <units>* - Specifies the *machine units* for linear axes. Possible choices are (in, inch, imperial, metric, mm). This does not affect the linear units in NC code (the G20 and G21 words do this).
- *ANGULAR_UNITS = <units>* - Specifies the *machine units* for rotational axes. Possible choices are *deg*, *degree* (360 per circle), *rad*, *radian* (2pi per circle), *grad*, or *gon* (400 per circle). This does not affect the angular units of NC code. In RS274NGC, A-, B- and C- words are always expressed in degrees.
- *DEFAULT_VELOCITY = 0.0167* - The initial rate for jogs of linear axes, in machine units per second. The value shown in *Axis* equals machine units per minute.
- *DEFAULT_ACCELERATION = 2.0* - In machines with nontrivial kinematics, the acceleration used for "teleop" (Cartesian space) jogs, in *machine units* per second per second.
- *MAX_VELOCITY = 5.0* - The maximum velocity for any axis or coordinated move, in *machine units* per second. The value shown equals 300 units per minute.
- *MAX_ACCELERATION = 20.0* - The maximum acceleration for any axis or coordinated axis move, in *machine units* per second per second.
- *POSITION_FILE = position.txt* - If set to a non-empty value, the joint positions are stored between runs in this file. This allows the machine to start with the same coordinates it had on shutdown. This assumes there was no movement of the machine while powered off. If unset, joint positions are not stored and will begin at 0 each time LinuxCNC is started. This can help on smaller machines without home switches.
- *NO_FORCE_HOMING = 1* - The default behavior is for LinuxCNC to force the user to home the machine before any MDI command or a program is run. Normally, only jogging is allowed before homing. Setting NO_FORCE_HOMING = 1 allows the user to make MDI moves and run programs without homing the machine first. Interfaces without homing ability will need to have this option set to 1.

**Warning**
LinuxCNC will not know your axis travel limits when using *NO_FORCE_HOMING = 1*.

#### 6.2.2.11 [AXIS_<num>] Section

The [AXIS_0], [AXIS_1], etc. sections contains general parameters for the individual components in the axis control module. The axis section names begin numbering at 0, and run through the number of axes specified in the [TRAJ] AXES entry minus 1.

Typically (but not always):

- AXIS_0 = X
- AXIS_1 = Y
- AXIS_2 = Z
- AXIS_3 = A
- AXIS_4 = B

- AXIS_5 = C
- AXIS_6 = U
- AXIS_7 = V
- AXIS_8 = W
- *TYPE = LINEAR* - The type of axes, either LINEAR or ANGULAR.
- *WRAPPED_ROTARY = 1* - When this is set to 1 for an ANGULAR axis the axis will move 0-359.999 degrees. Positive Numbers will move the axis in a positive direction and negative numbers will move the axis in the negative direction.
- *LOCKING_INDEXER = 1* - When this is set to 1, a G0 move for this axis will initiate an unlock with axis.N.unlock pin then wait for the axis.N.is-unlocked pin then move the axis at the rapid rate for that axis. After the move, the axis.N.unlock will be false and motion will wait for axis.N.is-unlocked to go false. Moving with other axes is not allowed when moving a locked rotary axis. If motion is disabled (estop for instance), while the axis.N.unlock pin is set, the axis.N.unlock pin is reset to false.
- *UNITS = INCH* - If specified, this setting overrides the related [TRAJ] UNITS setting. (e.g., [TRAJ]LINEAR_UNITS if the TYPE of this axis is LINEAR, [TRAJ]ANGULAR_UNITS if the TYPE of this axis is ANGULAR)
- *MAX_VELOCITY = 1.2* - Maximum velocity for this axis in machine units per second.
- *MAX_ACCELERATION = 20.0* - Maximum acceleration for this axis in machine units per second squared.
- *BACKLASH = 0.0000* - Backlash in machine units. Backlash compensation value can be used to make up for small deficiencies in the hardware used to drive an axis. If backlash is added to an axis and you are using steppers the STEPGEN_MAXACCEL must be increased to 1.5 to 2 times the MAX_ACCELERATION for the axis.
- *COMP_FILE = file.extension* - A file holding compensation structure for the axis. The file could be named xscrew.comp, for example, for the X axis. File names are case sensitive and can contain letters and/or numbers. The values are triplets per line separated by a space. The first value is nominal (where it should be). The second and third values depend on the setting of COMP_FILE_TYPE. Currently the limit inside LinuxCNC is for 256 triplets per axis. If COMP_FILE is specified, BACKLASH is ignored. Compensation file values are in machine units.
- *COMP_FILE_TYPE = 0 or 1* -
  - *If 0:* The second and third values specify the forward position (where the axis is while traveling forward) and the reverse position (where the axis is while traveling reverse), positions which correspond to the nominal position.'
  - *If 1:* The second and third values specify the forward trim (how far from nominal while traveling forward) and the reverse trim (how far from nominal while traveling in reverse), positions which correspond to the nominal position.

  ```
  Example triplet with COMP_FILE_TYPE = 0: 1.00 1.01 0.99 +
  Example triplet with COMP_FILE_TYPE = 1: 1.00 0.01 -0.01
  ```

- *MIN_LIMIT = -1000* - The minimum limit for axis motion, in machine units. When this limit is reached, the controller aborts axis motion. The axis must be homed before MIN_LIMIT is in force. For a rotary axis with unlimited rotation having no MIN_LIMIT for that axis in the [AXIS_n] section then the value -1e99 is used.
- *MAX_LIMIT = 1000* - The maximum limit for axis motion, in machine units. When this limit is reached, the controller aborts axis motion. The axis must be homed before MAX_LIMIT is in force. For a rotary axis with unlimited rotation having no MAX_LIMIT for that axis in the [AXIS_n] section then the value 1e99 is used.
- *MIN_FERROR = 0.010* - This is the value in machine units by which the axis is permitted to deviate from commanded position at very low speeds. If MIN_FERROR is smaller than FERROR, the two produce a ramp of error trip points. You could think of this as a graph where one dimension is speed and the other is permitted following error. As speed increases the amount of following error also increases toward the FERROR value.
- *FERROR = 1.0* - FERROR is the maximum allowable following error, in machine units. If the difference between commanded and sensed position exceeds this amount, the controller disables servo calculations, sets all the outputs to 0.0, and disables the amplifiers. If MIN_FERROR is present in the .ini file, velocity-proportional following errors are used. Here, the maximum allowable following error is proportional to the speed, with FERROR applying to the rapid rate set by [TRAJ]MAX_VELOCITY, and proportionally smaller following errors for slower speeds. The maximum allowable following error will always be greater than MIN_FERROR. This prevents small following errors for stationary axes from inadvertently aborting motion. Small following errors will always be present due to vibration, etc.

**Homing** These parameters are Homing related, for a better explanation read the Homing Configuration Chapter.

- *HOME = 0.0* - The position that the joint will go to upon completion of the homing sequence.
- *HOME_OFFSET = 0.0* - The axis position of the home switch or index pulse, in machine units. When the home point is found during the homing process, this is the position that is assigned to that point. When sharing home and limit switches and using a home sequence that will leave the home/limit switch in the toggled state the home offset can be used define the home switch position to be other than 0 if your HOME position is desired to be 0.
- *HOME_SEARCH_VEL = 0.0* - Initial homing velocity in machine units per second. Sign denotes direction of travel. A value of zero means assume that the current location is the home position for the machine. If your machine has no home switches you will want to leave this value at zero.
- *HOME_LATCH_VEL = 0.0* - Homing velocity in machine units per second to the home switch latch position. Sign denotes direction of travel.
- *HOME_FINAL_VEL = 0.0* - Velocity in machine units per second from home latch position to home position. If left at 0 or not included in the axis rapid velocity is used. Must be a positive number.
- *HOME_USE_INDEX = NO* - If the encoder used for this axis has an index pulse, and the motion card has provision for this signal you may set it to yes. When it is yes, it will affect the kind of home pattern used. Currently, you can't home to index with steppers unless you're using stepgen in velocity mode and PID.
- *HOME_IGNORE_LIMITS = NO* - When you use the limit switch as a home switch and the limit switch this should be set to YES. When set to YES the limit switch for this axis is ignored when homing. You must configure your homing so that at the end of your home move the home/limit switch is not in the toggled state you will get a limit switch error after the home move.
- *HOME_IS_SHARED = <n>* - If the home input is shared by more than one axis set <n> to 1 to prevent homing from starting if the one of the shared switches is already closed. Set <n> to 0 to permit homing if a switch is closed.
- *HOME_SEQUENCE = <n>* - Used to define the "Home All" sequence. <n> starts at 0 and no numbers may be skipped. If left out or set to -1 the joint will not be homed by the "Home All" function. More than one axis can be homed at the same time.
- *VOLATILE_HOME = 0* - When enabled (set to 1) this joint will be unhomed if the Machine Power is off or if E-Stop is on. This is useful if your machine has home switches and does not have position feedback such as a step and direction driven machine.

**Servo** These parameters are relevant to axes controlled by servos.

**Warning**
The following are custom INI file entries that you may find in a sample INI file or a wizard generated file. These are not used by the LinuxCNC software. They are only there to put all the settings in one place. For more information on custom INI file entries see the Custom Sections and Variables subsection.

The following items might be used by a PID component and the assumption is that the output is volts.

- *DEADBAND = 0.000015* - How close is close enough to consider the motor in position, in machine units. This is often set to a distance equivalent to 1, 1.5, 2, or 3 encoder counts, but there are no strict rules. Looser (larger) settings allow less servo *hunting* at the expense of lower accuracy. Tighter (smaller) settings attempt higher accuracy at the expense of more servo *hunting*. Is it really more accurate if it's also more uncertain? As a general rule, it's good to avoid, or at least limit, servo *hunting* if you can.

Be careful about going below 1 encoder count, since you may create a condition where there is no place that your servo is happy. This can go beyond *hunting* (slow) to *nervous* (rapid), and even to *squealing* which is easy to confuse with oscillation caused by improper tuning. Better to be a count or two loose here at first, until you've been through *gross tuning* at least.

Example of calculating machine units per encoder pulse to use in deciding DEADBAND value:

$$\frac{1\,revolution}{1000\,lines} X \frac{1\,line}{4\,pulse/line} X \frac{0.2\,units}{1\,revolution} = \frac{0.200\,units}{4000\,pulses} = \frac{0.00005\,units}{1\,pulse}$$

- *BIAS = 0.000* - This is used by hm2-servo and some others. Bias is a constant amount that is added to the output. In most cases it should be left at zero. However, it can sometimes be useful to compensate for offsets in servo amplifiers, or to balance the weight of an object that moves vertically. bias is turned off when the PID loop is disabled, just like all other components of the output.
- *P = 50* - The proportional gain for the axis servo. This value multiplies the error between commanded and actual position in machine units, resulting in a contribution to the computed voltage for the motor amplifier. The units on the P gain are volts per machine unit, e.g., $\frac{volts}{unit}$
- *I = 0* - The integral gain for the axis servo. The value multiplies the cumulative error between commanded and actual position in machine units, resulting in a contribution to the computed voltage for the motor amplifier. The units on the I gain are volts per machine unit second, e.g., $\frac{volts}{unit\ second}$
- *D = 0* - The derivative gain for the axis servo. The value multiplies the difference between the current and previous errors, resulting in a contribution to the computed voltage for the motor amplifier. The units on the D gain are volts per machine unit per second, e.g., $\frac{volts}{unit\ second}$
- *FF0 = 0* - The 0th order feed forward gain. This number is multiplied by the commanded position, resulting in a contribution to the computed voltage for the motor amplifier. The units on the FF0 gain are volts per machine unit, e.g., $\frac{volts}{unit}$
- *FF1 = 0* - The 1st order feed forward gain. This number is multiplied by the change in commanded position per second, resulting in a contribution to the computed voltage for the motor amplifier. The units on the FF1 gain are volts per machine unit per second, e.g., $\frac{volts}{unit\ second}$
- *FF2 = 0* - The 2nd order feed forward gain. This number is multiplied by the change in commanded position per second per second, resulting in a contribution to the computed voltage for the motor amplifier. The units on the FF2 gain are volts per machine unit per second per second, e.g., $\frac{volts}{unit\ second^2}$
- *OUTPUT_SCALE = 1.000* -
- *OUTPUT_OFFSET = 0.000* - These two values are the scale and offset factors for the axis output to the motor amplifiers. The second value (offset) is subtracted from the computed output (in volts), and divided by the first value (scale factor), before being written to the D/A converters. The units on the scale value are in true volts per DAC output volts. The units on the offset value are in volts. These can be used to linearize a DAC. Specifically, when writing outputs, the LinuxCNC first converts the desired output in quasi-SI units to raw actuator values, e.g., volts for an amplifier DAC. This scaling looks like: $raw = \frac{output - offset}{scale}$

The value for scale can be obtained analytically by doing a unit analysis, i.e., units are [output SI units]/[actuator units]. For example, on a machine with a velocity mode amplifier such that 1 volt results in 250 mm/sec velocity.

$$amplifier[volts] = (output[\frac{mm}{sec}] - offset[\frac{mm}{sec}])/250\,\frac{mm}{secvolt}$$

Note that the units of the offset are in machine units, e.g., mm/sec, and they are pre-subtracted from the sensor readings. The value for this offset is obtained by finding the value of your output which yields 0.0 for the actuator output. If the DAC is linearized, this offset is normally 0.0.

The scale and offset can be used to linearize the DAC as well, resulting in values that reflect the combined effects of amplifier gain, DAC non-linearity, DAC units, etc.

To do this, follow this procedure.

1. Build a calibration table for the output, driving the DAC with a desired voltage and measuring the result.
2. Do a least-squares linear fit to get coefficients a, b such that $measured = a * raw + b$
3. Note that we want raw output such that our measured result is identical to the commanded output. This means
   a. $command = a * raw + b$
   b. $raw = (command - b) / a$
4. As a result, the a and b coefficients from the linear fit can be used as the scale and offset for the controller directly.

See the following table for an example of voltage measurements.

Table 6.1: Output Voltage Measurements

| Raw | Measured |
|---|---|
| -10 | -9.93 |
| -9 | -8.83 |
| 0 | -0.03 |
| 1 | 0.96 |
| 9 | 9.87 |
| 10 | 10.87 |

- *MAX_OUTPUT = 10* - The maximum value for the output of the PID compensation that is written to the motor amplifier, in volts. The computed output value is clamped to this limit. The limit is applied before scaling to raw output units. The value is applied symmetrically to both the plus and the minus side.
- *INPUT_SCALE = 20000* - in Sample configs
- *ENCODER_SCALE = 20000* - in PNCconf built configs Specifies the number of pulses that corresponds to a move of one machine unit as set in the [TRAJ] section. For a linear axis one machine unit will be equal to the setting of LINEAR_UNITS. For an angular axis one unit is equal to the setting in ANGULAR_UNITS. A second number, if specified, is ignored. For example, on a 2000 counts per rev encoder, and 10 revs/inch gearing, and desired units of inch, we have:

$$input\,scale = 2000 \frac{counts}{rev} * 10 \frac{rev}{inch} = 20000 \frac{counts}{inch}$$

**Stepper** These parameters are relevant to axes controlled by steppers.

**Warning**
The following are custom INI file entries that you may find in a sample INI file or a wizard generated file. These are not used by the LinuxCNC software. They are only there to put all the settings in one place. For more information on custom INI file entries see the Custom Sections and Variables subsection.

The following items might be used by a stepgen component.

- *SCALE = 4000* - in Sample configs
- *STEP_SCALE = 4000* - in PNCconf built configs Specifies the number of pulses that corresponds to a move of one machine unit as set in the [TRAJ] section. For stepper systems, this is the number of step pulses issued per machine unit. For a linear axis one machine unit will be equal to the setting of LINEAR_UNITS. For an angular axis one unit is equal to the setting in ANGULAR_UNITS. For servo systems, this is the number of feedback pulses per machine unit. A second number, if specified, is ignored.

For example, on a 1.8 degree stepper motor with half-stepping, and 10 revs/inch gearing, and desired machine units of inch, we have:

$$input\,scale = \frac{2\,steps}{1.8\,degrees} * 360\,\frac{degree}{rev} * 10\,\frac{rev}{inch} = 4000\,\frac{steps}{inch}$$

- *ENCODER_SCALE = 20000* (Optionally used in PNCconf built configs) - Specifies the number of pulses that corresponds to a move of one machine unit as set in the [TRAJ] section. For a linear axis one machine unit will be equal to the setting of LINEAR_UNITS. For an angular axis one unit is equal to the setting in ANGULAR_UNITS. A second number, if specified, is ignored. For example, on a 2000 counts per rev encoder, and 10 revs/inch gearing, and desired units of inch, we have:

$$input\,scale = 2000\,\frac{counts}{rev} * 10\,\frac{rev}{inch} = 20000\,\frac{counts}{inch}$$

- *STEPGEN_MAXACCEL = 21.0* - Acceleration limit for the step generator. This should be 1% to 10% larger than the axis MAX_ACCELERATION. This value improves the tuning of stepgen's "position loop". If you have added backlash compensation to an axis then this should be 1.5 to 2 times greater than MAX_ACCELERATION.
- *STEPGEN_MAXVEL = 1.4* - Older configuration files have a velocity limit for the step generator as well. If specified, it should also be 1% to 10% larger than the axis MAX_VELOCITY. Subsequent testing has shown that use of STEPGEN_MAXVEL does not improve the tuning of stepgen's position loop.

#### 6.2.2.12 [EMCIO] Section

- *EMCIO = io* - Name of IO controller program
- *CYCLE_TIME = 0.100* - The period, in seconds, at which EMCIO will run. Making it 0.0 or a negative number will tell EMCIO not to sleep at all. There is usually no need to change this number.
- *TOOL_TABLE = tool.tbl* - The file which contains tool information, described in the User Manual.
- *TOOL_CHANGE_POSITION = 0 0 2* - Specifies the XYZ location to move to when performing a tool change if three digits are used. Specifies the XYZABC location when 6 digits are used. Specifies the XYZABCUVW location when 9 digits are used. Tool Changes can be combined. For example if you combine the quill up with change position you can move the Z first then the X and Y.
- *TOOL_CHANGE_WITH_SPINDLE_ON = 1* - The spindle will be left on during the tool change when the value is 1. Useful for lathes or machines where the material is in the spindle, not the tool.
- *TOOL_CHANGE_QUILL_UP = 1* - The Z axis will be moved to machine zero prior to the tool change when the value is 1. This is the same as issuing a G0 G53 Z0.
- *TOOL_CHANGE_AT_G30 = 1* - The machine is moved to reference point defined by parameters 5181-5186 for G30 if the value is 1. For more information see the Parameters Section and the <<gcode:g30-g30.1.
- *RANDOM_TOOLCHANGER = 1* - This is for machines that cannot place the tool back into the pocket it came from. For example, machines that exchange the tool in the active pocket with the tool in the spindle.

## 6.3 Homing Configuration

### 6.3.1 Overview

Homing seems simple enough - just move each joint to a known location, and set LinuxCNC's internal variables accordingly. However, different machines have different requirements, and homing is actually quite complicated.

### 6.3.2 Homing Sequence

There are four possible homing sequences defined by the sign of HOME_SEARCH_VEL and HOME_LATCH_VEL, along with the associated configuration parameters as shown in the following table. Two basic conditions exist, HOME_SEARCH_VEL and HOME_LATCH_VEL are the same sign or they are opposite signs. For a more detailed description of what each configuration parameter does, see the following section.

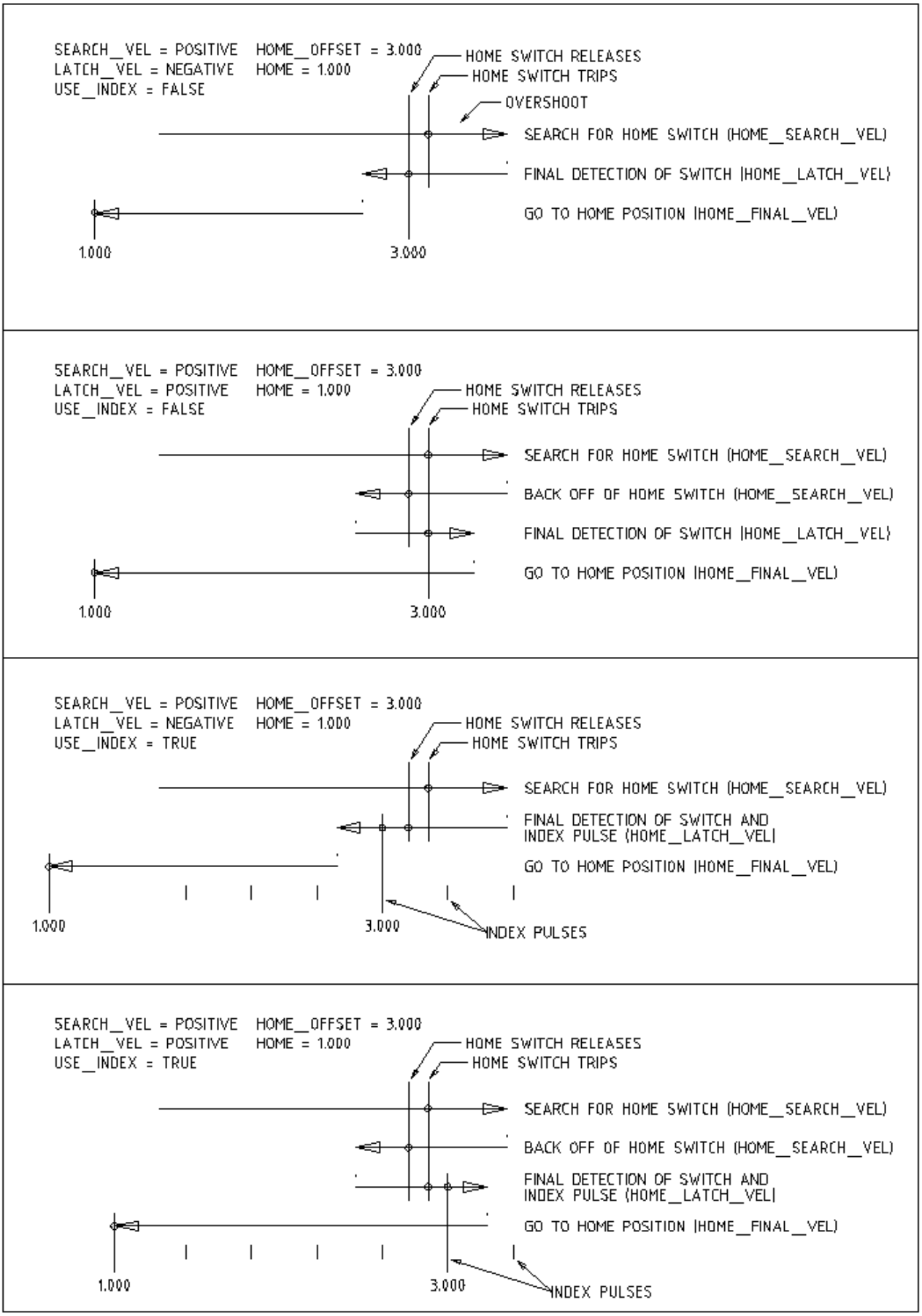



Figure 6.1: Homing Sequences

### 6.3.3 Configuration

The following determines exactly how the home sequence behaves. They are defined in an [AXIS] section of the inifile.

| Homing Type | HOME_SEARCH_VEL | HOME_LATCH_VEL | HOME_USE_INDEX |
|---|---|---|---|
| Immediate | 0 | 0 | NO |
| Index-only | 0 | nonzero | YES |
| Switch-only | nonzero | nonzero | NO |
| Switch and Index | nonzero | nonzero | YES |

**Note**
Any other combinations may result in an error.

#### 6.3.3.1 HOME_SEARCH_VEL

This variable has units of machine-units per second.

The default value is zero. A value of zero causes LinuxCNC to assume that there is no home switch; the search stage of homing is skipped.

If HOME_SEARCH_VEL is non-zero, then LinuxCNC assumes that there is a home switch. It begins by checking whether the home switch is already tripped. If tripped it backs off the switch at HOME_SEARCH_VEL. The direction of the back-off is opposite the sign of HOME_SEARCH_VEL. Then it searches for the home switch by moving in the direction specified by the sign of HOME_SEARCH_VEL, at a speed determined by its absolute value. When the home switch is detected, the joint will stop as fast as possible, but there will always be some overshoot. The amount of overshoot depends on the speed. If it is too high, the joint might overshoot enough to hit a limit switch or crash into the end of travel. On the other hand, if HOME_SEARCH_VEL is too low, homing can take a long time.

#### 6.3.3.2 HOME_LATCH_VEL

This variable has units of machine-units per second.

Specifies the speed and direction that LinuxCNC uses when it makes its final accurate determination of the home switch (if present) and index pulse location (if present). It will usually be slower than the search velocity to maximize accuracy. If HOME_SEARCH_VEL and HOME_LATCH_VEL have the same sign, then the latch phase is done while moving in the same direction as the search phase. (In that case, LinuxCNC first backs off the switch, before moving towards it again at the latch velocity.) If HOME_SEARCH_VEL and HOME_LATCH_VEL have opposite signs, the latch phase is done while moving in the opposite direction from the search phase. That means LinuxCNC will latch the first pulse after it moves off the switch. If HOME_SEARCH_VEL is zero (meaning there is no home switch), and this parameter is nonzero, LinuxCNC goes ahead to the index pulse search. If HOME_SEARCH_VEL is non-zero and this parameter is zero, it is an error and the homing operation will fail. The default value is zero.

#### 6.3.3.3 HOME_FINAL_VEL

This variable has units of machine-units per second.

It specifies the speed that LinuxCNC uses when it makes its move from HOME_OFFSET to the HOME position. If the HOME_FINAL_VEL is missing from the ini file, then the maximum joint speed is used to make this move. The value must be a positive number.

#### 6.3.3.4 HOME_IGNORE_LIMITS

Can hold the values YES / NO. The default value for this parameter is NO. This flag determines whether LinuxCNC will ignore the limit switch input for this axis while homing. Setting this to YES will not ignore limit inputs for other axes. If you do not have a separate home switch set this to YES and case connect the limit switch signal to the home switch input in HAL. LinuxCNC will ignore the limit switch input for this axis while homing. To use only one input for all homing and limits you will have to block the limit signals of the axes not homing in HAL and home one axis at a time.

#### 6.3.3.5 HOME_USE_INDEX

Specifies whether or not there is an index pulse. If the flag is true (HOME_USE_INDEX = YES), LinuxCNC will latch on the rising edge of the index pulse. If false, LinuxCNC will latch on either the rising or falling edge of the home switch (depending on the signs of HOME_SEARCH_VEL and HOME_LATCH_VEL). The default value is NO.

**Note**
HOME_USE_INDEX requires connections in your hal file to axis.n.index-enable from the encoder.n.index-enable.

#### 6.3.3.6 HOME_OFFSET

Contains the location of the home switch or index pulse, in joint coordinates. It can also be treated as the distance between the point where the switch or index pulse is latched and the zero point of the joint. After detecting the index pulse, LinuxCNC sets the joint coordinate of the current point to HOME_OFFSET. The default value is zero.

**Note**
The home switch location, as indicated by the HOME_OFFSET variable, must be strictly within the soft limits. That is, `MIN_LIMIT < HOME_OFFSET < MAX_LIMIT`.

#### 6.3.3.7 HOME

The position that the joint will go to upon completion of the homing sequence. After detecting the home switch or home switch then index pulse (depending on configuation), and setting the coordinate of that point to HOME_OFFSET, LinuxCNC makes a move to HOME as the final step of the homing process. The default value is zero. Note that even if this parameter is the same as HOME_OFFSET, the joint will slightly overshoot the latched position as it stops. Therefore there will always be a small move at this time (unless HOME_SEARCH_VEL is zero, and the entire search/latch stage was skipped). This final move will be made at the joint's maximum velocity unless HOME_FINAL has been set.

**Note**
The distinction between *HOME_OFFSET* and *HOME* is that *HOME_OFFSET* first establishes the scale location on the machine by applying the *HOME_OFFSET* value to the location where home was found, and then *HOME* says where the joint should move to on that scale.

#### 6.3.3.8 HOME_IS_SHARED

If there is not a separate home switch input for this axis, but a number of momentary switches wired to the same pin, set this value to 1 to prevent homing from starting if one of the shared switches is already closed. Set this value to 0 to permit homing even if the switch is already closed.

#### 6.3.3.9 HOME_SEQUENCE

Used to define a multi-axis homing sequence HOME ALL and enforce homing order (e.g., Z may not be homed if X is not yet homed). An axis may be homed after all axes with a lower HOME_SEQUENCE have already been homed and are at the HOME_OFFSET. If two axes have the same HOME_SEQUENCE, they may be homed at the same time. If HOME_SEQUENCE is -1 or not specified then this joint will not be homed by the HOME ALL sequence. HOME_SEQUENCE numbers start with 0 and there may be no unused numbers.

#### 6.3.3.10 VOLATILE_HOME

If this setting is true, this axis becomes unhomed whenever the machine transitions into the OFF state. This is appropriate for any axis that does not maintain position when the axis drive is off. Some stepper drives, especially microstep drives, may need this.

#### 6.3.3.11 LOCKING_INDEXER

If this axis is a locking rotary indexer, it will unlock before homing, and lock afterward.

#### 6.3.3.12 Immediate Homing

If an axis does not have home switches or does not have a logical home position like a rotary axis and you want that axis to home at the current position when the "Home All" button is pressed in Axis the following ini entries for that axis are needed.

1. HOME_SEARCH_VEL = 0
2. HOME_LATCH_VEL = 0
3. HOME_USE_INDEX = NO
4. HOME_SEQUENCE = 0

## 6.4 Lathe Configuration

### 6.4.1 Default Plane

When LinuxCNC's interpreter was first written, it was designed for mills. That is why the default plane is XY (G17). A normal lathe only uses the XZ plane (G18). To change the default plane place the following line in the .ini file in the RS274NGC section.

```
RS274NGC_STARTUP_CODE = G18
```

The above can be overwritten in a g code program so always set important things in the preamble of the g code file.

### 6.4.2 INI Settings

The following .ini settings are needed for lathe mode in Axis in addition to or replacing normal settings in the .ini file. Gmoccapy uses also the mentioned settings, but does offer additional settings, check the gmoccapy Section for details.

```
[DISPLAY]
DISPLAY = axis
LATHE = 1

[TRAJ]
AXES = 3
COORDINATES = X Z
[AXIS_0]
...
[AXIS_2]
...
```

## 6.5 HALTCL Files

The halcmd language excels in specifiying components and connections but offers no computational capabilities. As a result, ini files are limited in the clarity and brevity that is possible with higher level languages.

The haltcl facility provides a means to use tcl scripting and its features for computation, looping, branching, procedures, etc. in ini files. To use this functionality, you use the tcl language and the extension .tcl for halfiles.

The .tcl extension is understood by the main script (linuxcnc) that processes ini files. Haltcl files are identified in the the HAL section of ini files (just like .hal files).

**Example**

```
[HAL]
HALFILE = conventional_file.hal
HALFILE = tcl_based_file.tcl
```

With appropriate care, .hal and .tcl files can be intermixed.

### 6.5.1 Compatibility

The halcmd language used in .hal files has a simple syntax that is actually a subset of the more powerful general-purpose tcl scripting language.

### 6.5.2 Haltcl Commands

Haltcl files use the tcl scripting language augmented with the specific commands of the LinuxCNC hardware abstraction layer (HAL). The hal-specific commands are.

```
addf, alias,
delf, delsig,
getp, gets
ptype,
stype,
help,
linkpp, linkps, linksp, list, loadrt, loadusr, lock,
net, newsig,
save, setp, sets, show, source, start, status, stop,
unalias, unlinkp, unload, unloadrt, unloadusr, unlock,
waitusr
```

Two special cases occur for the *gets* and *list* commands due to conflicts with tcl builtin commands. For haltcl, these commands must be preceded with the keyword *hal*.

```
halcmd   haltcl
------   ------
gets     hal gets
list     hal list
```

### 6.5.3 Haltcl Inifile variables

Inifile variables are accessible by both halcmd and haltcl but with differing syntax.

LinuxCNC ini files use SECTION and ITEM specifiers to identify configuration items.

```
[SECTION_A]
ITEM1 = value_1
ITEM2 = value_2
...
[SECTION_B]
...
```

The ini file values are accessible by text substition in .hal files using the form.

```
[SECTION]ITEM
```

The same ini file values are accessible in .tcl files using the form of a tcl global array variable.

```
$::SECTION(ITEM)
```

For example, an ini file item like:

```
[AXIS_0]
MAX_VELOCITY = 4
```

is expressed as [AXIS_0]MAX_VELOCITY in .hal files for halcmd
and as $::AXIS_0(MAX_VELOCITY) in .tcl files for haltcl

Because inifiles can repat the same ITEM in the same SECTION multiple times, $::SECTION(ITEM) is actually a Tcl list of each individual value.

When there is just one value and it is a simple value (all values that are just letters and numbers without whitespace are in this group), then it is possible to treat $::SECTION(ITEM) as though it is not a list.

When the value could contain special characters—quote characters, curly-brace characters, embedded whitespace, and other characters that have special meaning in Tcl—it is necessary to distinguish between the list of values and the initial (and possibly only) value in the list.

In Tcl, this is written `[lindex $::SECTION(ITEM) 0]`.

For example: given the following ini values

```
[HOSTMOT2]
DRIVER=hm2_eth
IPADDR="10.10.10.10"
BOARD=7i92
CONFIG="num_encoders=0 num_pwmgens=0 num_stepgens=6"
```

And this loadrt command:

```
loadrt $::HOSTMOT2(DRIVER) board_ip=$::HOSTMOT2(IPADDR) config=$::HOSTMOT2(CONFIG)
```

Here is the actual command that is run:

```
loadrt hm2_eth board_ip={"10.10.10.10"} config={"num_encoders=0 num_pwmgens=0 num_stepgens ↩
    =6"}
```

This fails because loadrt does not recognize the braces.

So to get the values just as entered in the ini file, re-write the loadrt line like this:

```
loadrt $::HOSTMOT2(DRIVER) board_ip=[lindex $::HOSTMOT2(IPADDR) 0] config=[lindex $:: ↩
    HOSTMOT2(CONFIG) 0]
```

### 6.5.4 Converting .hal files to .tcl files

Existing .hal files can be converted to .tcl files by hand editing to adapt to the differences mentioned above. The process can be automated with scripts that convert using these substitutions.

```
[SECTION]ITEM ---> $::SECTION(ITEM)
gets          ---> hal gets
list          ---> hal list
```

### 6.5.5 Haltcl Notes

In haltcl, the value argument for the *sets* and *setp* commands is implicitly treated as an expression in the tcl language.

**Example**

```
# set gain to convert deg/sec to units/min for AXIS_0 radius
setp scale.0.gain 6.28/360.0*$::AXIS_0(radius)*60.0
```

Whitespace in the bare expression is not allowed, use quotes for that:

```
setp scale.0.gain "6.28 / 360.0 * $::AXIS_0(radius) * 60.0"
```

In other contexts, such as *loadrt*, you must explicitly use the tcl expr command ([expr {}]) for computational expressions.

**Example**

```
loadrt motion base_period=[expr {500000000/$::TRAJ(MAX_PULSE_RATE)}]
```

### 6.5.6 Haltcl Examples

Consider the topic of *stepgen headroom*. Software stepgen runs best with an acceleration constraint that is "a bit higher" than the one used by the motion planner. So, when using halcmd files, we force inifiles to have a manually calculated value.

```
[AXIS_0]
MAXACCEL = 10.0
STEPGEN_MAXACCEL = 10.5
```

With haltcl, you can use tcl commands to do the computation and eliminate the STEPGEN_MAXACCEL inifile item altogether.

```
setp stepgen.0.maxaccel $::AXIS_0(MAXACCEL)*1.05
```

Another haltcl feature is looping and testing. For example, many simulator configurations use "core_sim.hal" or "core_sim9.hal" hal files. These differ because of the requirement to connect more or fewer axes. The following haltcl code would work for any combination of axes in a trivkins machine.

```
# Create position, velocity and acceleration signals for each axis
set ddt 0
foreach axis {X Y Z A B C U V W} axno {0 1 2 3 4 5 6 7 8} {
  # 'list pin' returns an empty list if the pin doesn't exist
  if {[hal list pin axis.$axno.motor-pos-cmd] == {}} {
    continue
  }
  net ${axis}pos axis.$axno.motor-pos-cmd => axis.$axno.motor-pos-fb \
                                          => ddt.$ddt.in
  net ${axis}vel <= ddt.$ddt.out
  incr ddt
  net ${axis}vel => ddt.$ddt.in
  net ${axis}acc <= ddt.$ddt.out
  incr ddt
}
puts [show sig *vel]
puts [show sig *acc]
```

### 6.5.7 Haltcl Interactive

The halrun command recognizes haltcl files. With the -T option, haltcl can be run interaactively as a tcl interpreter. This capability is useful for testing and for standalone hal applications.

**Example**

```
$ halrun -T haltclfile.tcl
```

### 6.5.8 Haltcl Distribution Examples (sim)

The configs/sim/axis/simtcl directory includes an ini file that uses a .tcl file to demonstrate a haltcl configuration in conjunction with the usage of twopass processing. The example shows the use of tcl procedures, looping, the use of comments, and output to the terminal.

## 6.6 Remap Extending G code

### 6.6.1 Introduction: Extending the RS274NGC Interpreter by Remapping Codes

#### 6.6.1.1 A Definition: Remapping Codes

By *remapping codes* we mean one of the following:

1. define the semantics of new - that is, currently unallocated - M- or G-codes
2. redefine the semantics of a - currently limited - set of existing codes.

#### 6.6.1.2 Why would you want to extend the RS274NGC Interpreter?

The set of codes (M,G,T,S,F) currently understood by the RS274NGC interpreter is fixed and cannot be extended by configuration options.

In particular, some of these codes implement a fixed sequence of steps to be executed. While some of these, like M6, can be moderately configured by activating or skipping some of these steps through ini file options, overall the behavior is fairly rigid. So - if your are happy with this situation, then this manual section is not for you.

In many cases, this means that supporting a non *out of the box* configuration or machine is either cumbersome or impossible, or requires resorting to changes at the *C/C+\+* language level. The latter is unpopular for good reasons - changing internals requires in-depth understanding of interpreter internals, and moreover brings its own set of support issues. While it is conceivable that certain patches might make their way into the main LinuxCNC distribution, the result of this approach is a hodge-podge of special-case solutions.

A good example for this deficiency is tool change support in LinuxCNC: while random tool changers are well supported, it is next to impossible to reasonably define a configuration for a manual-tool change machine with, for example, an automatic tool length offset switch being visited after a tool change, and offsets set accordingly. Also, while a patch for a very specific rack tool changer exists, it has not found its way back into the main code base.

However, many of these things may be fixed by using an O-word procedure instead of a built in code - whenever the - insufficient - built in code is to be executed, call the O-word procedure instead. While possible, it is cumbersome - it requires source-editing of NGC programs, replacing all calls to the deficient code by a an O-word procedure call.

In it's simplest form a remapped code isn't much more than a spontaneous call to an O-word procedure. This happens behind the scenes - the procedure is visible at the configuration level, but not at the NGC program level.

Generally, the behavior of a remapped code may be defined in the following ways:

- you define a O-word subroutine which implements the desired behavior
- alternatively, you may employ a Python function which extends the interpreter's behavior.

**How to glue things together** M- and G-codes, and O-words subroutine calls have some fairly different syntax.

O-word procedures, for example, take positional parameters with a specific syntax like so:

```
o<test> call [1.234] [4.65]
```

whereas M- or G-codes typically take required or optional *word* parameters. For instance, G76 (threading) requires the P,Z,I,J and K words, and optionally takes the R,Q,H, E and L words.

So it isn't simply enough to say *whenever you encounter code X, please call procedure Y* - at least some checking and conversion of parameters needs to happen. This calls for some *glue code* between the new code, and its corresponding NGC procedure to execute before passing control to the NGC procedure.

This glue code is impossible to write as an O-word procedure itself since the RS274NGC language lacks the introspective capabilities and access into interpreter internal data structures to achieve the required effect. Doing the glue code in - again - *C/C+\+* would be an inflexible and therefore unsatisfactory solution.

**How Embedded Python fits in** To make a simple situation easy and a complex situation solvable, the glue issue is addressed as follows:

- for simple situations, a built-in glue procedure (`argspec`) covers most common parameter passing requirements
- for remapping T,M6,M61,S,F there is some standard Python glue which should cover most situations, see Standard Glue
- for more complex situations, one can write your own Python glue to implement new behavior.

Embedded Python functions in the Interpreter started out as glue code, but turned out very useful well beyond that. Users familiar with Python will likely find it easier to write remapped codes, glue, O-word procedures etc in pure Python, without resorting to the somewhat cumbersome RS274NGC language at all.

**A Word on Embedded Python** Many people are familiar with *extending* the Python interpreter by *C/C++* modules, and this is heavily used in LinuxCNC to access Task, HAL and and Interpreter internals from Python scripts. *Extending Python* basically means: your Python script executes as *it is in the driver seat*, and may access non-Python code by importing and using extension modules written in *C/C+\+*. Examples for this are the LinuxCNC `hal`, `gcode` and `emc` modules.

Embedded Python is a bit different and and less commonly known: The main program is written in C/C++ and may use Python like a subroutine. This is powerful extension mechanism and the basis for the *scripting extensions* found in many successful software packages. Embedded Python code may access *C/C+\+* variables and functions through a similar extension module method.

### 6.6.2 Getting started

Defining a code involves the following steps:

- pick a code - either use an unallocated code, or redefine an existing code
- deciding how parameters are handled
- decide if and how results are handled
- decide about the execution sequencing.

#### 6.6.2.1 Picking a code

Note that currently only a few existing codes may be redefined, whereas there are many *free* codes which might be made available by remapping. When developing a redefined existing code, it might be a good idea to start with an unallocated G- or M-code so both the existing and new behavior can be exercised. When done, redefine the existing code to use your remapping setup.

- the current set of unused M-codes open to user definition can be found here,
- unallocated G-codes are listed here.
- Existing codes which may be remapped are listed here.

#### 6.6.2.2 Parameter handling

Let's assume the new code will be defined by an NGC procedure, and needs some parameters, some of which might be required, others might be optional. We have the following options to feed values to the procedure:

1. extracting words from the current block and pass them to the procedure as parameters (like `X22.34` or `P47`)
2. referring to ini file variables
3. referring to global variables (like `#2200 = 47.11` or `#<_global_param> = 315.2`

The first method is preferred for parameters of dynamic nature, , like positions. You need to define which words on the current block have any meaning for your new code, and specify how that is passed to the NGC procedure. Any easy way is to use the argspec statement. A custom prolog might provide better error messages.

Using to ini file variables is most useful for referring to setup information for your machine, for instance a fixed position like a tool-length sensor position. The advantage of this method is that the parameters are fixed for your configuration regardless which NGC file you're currently executing.

Referring to global variables is always possible, but they are easily overlooked.

Note there's a limited supply of words which may be used as parameters, so one might need to fall back to the second and third methods if many parameters are needed.

#### 6.6.2.3 Handling results

Your new code might succeed or fail, for instance if passed an invalid parameter combination. Or you might choose to *just execute* the procedure and disregard results, in which case there isn't much work to do.

Epilog handlers help in processing results of remap procedures - see the reference section.

#### 6.6.2.4 Execution sequencing

Executable G-code words are classified into modal groups, which also defines their relative execution behavior.

If a G-code block contains several executable words on a line, these words are executed in a predefined order of execution, not in the order they appear in block.

When you define a new executable code, the interpreter does not yet know where your code fits into this scheme. For this reason, you need to choose an appropriate modal group for your code to execute in.

#### 6.6.2.5 An minimal example remapped code

To give you an idea how the pieces fit together, let's explore a fairly minimal but complete remapped code definition. We choose an unallocated M-code and add the following option to the ini file:

```
[RS274NGC]
REMAP=M400  modalgroup=10 argspec=Pq ngc=myprocedure
```

In a nutshell, this means:

- The `M400` code takes a required parameter `P` and an optional parameter `Q`. Other words in the current block are ignored with respect to the `M400` code. If the `P` word is not present, fail execution with an error.
- when an `M400` code is encountered, execute `myprocedure.ngc` along the other modal group 10 M-codes as per order of execution.
- the value of `P`, and `Q` are available in the procedure as local named parameters. The may be referred to as `#<P>` and `#<Q>`. The procedure may test whether the `Q` word was present with the EXISTS built in function.

The file `myprocedure.ngc` is expected to exists in the `[DISPLAY]NC_FILES` or `[RS274NGC]SUBROUTINE_PATH` directory.

A detailed discussion of REMAP parameters is found in the reference section below.

### 6.6.3 Configuring Remapping

#### 6.6.3.1 The REMAP statement

To remap a code, define it using the `REMAP` option in `RS274NG` section of your ini file. Use one `REMAP` line per remapped code.

The syntax of the `REMAP` is:

**`REMAP=`*<code>* *<options>***
: where *<code>* may be one of `T`,`M6`,`M61`,`S`,`F` (existing codes) or any of the unallocated M-codes or G-codes.

It is an error to omit the *<code>* parameter.

The options of the REMAP statement are separated by whitespace. The options are keyword-value pairs and currently are:

**`modalgroup=`*<modal group>***

**G-codes**
: the only currently supported modal group is 1, which is also the default value if no group is given. Group 1 means *execute alongside other G-codes*.

**M-codes**
: currently supported modal groups are: 5,6,7,8,9,10. If no modalgroup is give, it defaults to 10 (*execute after all other words in the block*).

**T,S,F**
: for these the modal group is fixed and any `modalgroup=` option is ignored.

**`argspec=`*<argspec>***
: See description of the argspec parameter options. Optional.

**`ngc=`*<ngc_basename>***
: Basename of an O-word subroutine file name. Do not specify an .ngc extension. Searched for in the directories specified in the directory specified in `[DISPLAY]PROGRAM_PREFIX`, then in `[RS274NGC]SUBROUTINE_PATH`. Mutually exclusive with `python=`. It is an error to omit both `ngc=` and `python=`.

**`python=`*<Python function name>***
: Instead of calling an ngc O-word procedure call a Python function. The function is expected to be defined in the `module_basename.oword` module. Mutually exclusive with `ngc=`.

**`prolog=`*<Python function name>***
: Before executing an ngc procedure, call this Python function. The function is expected to be defined in the `module_basename.` module. Optional.

**`epilog=`*<Python function name>***
: After executing an ngc procedure, call this Python function. The function is expected to be defined in the `module_basename.r` module. Optional.

The `python`, `prolog` and `epilog` options require the Python Interpreter plugin to be configured, and appropriate Python functions to be defined there so they can be referred to with these options.

The syntax for defining a new code, and redefining an existing code is identical.

#### 6.6.3.2 Useful REMAP option combinations

Note that while many combinations of argspec options are possible, not all of them make sense. The following combinations are useful idioms:

**`argspec=`*<words>* `ngc=`*<procname>* `modalgroup=`*<group>***
: The recommended way to call an NGC procedure with a standard argspec parameter conversion. Used if argspec is good enough. Note it's not good enough for remapping the Tx and M6/M61 tool change codes.

**prolog=<*pythonprolog*> ngc=<*procname*> epilog=<*pythonepilog*> modalgroup=<*group*>**
Call a Python prolog function to take any preliminary steps, then call the NGC procedure. When done, call the Python epilog function to do any cleanup or result extraction work which cannot be handled in G-code. The most flexible way of remapping a code to an NGC procedure, since almost all of the Interpreter internal variables, and some internal functions may be accessed from the prolog and epilog handlers. Also, a longer rope to hang yourselves.

**python=<*pythonfunction*> modalgroup=<*group*>**
Directly call to a Python function without any argument conversion. The most powerful way of remapping a code and going straight to Python. Use this if you don't need an NGC procedure, or NGC is just getting in your way.

**argspec=<*words*> python=<*pythonfunction*> modalgroup=<*group*>**
Convert the argspec words and pass them to a Python function as keyword argument dictionary. Use it when you're too lazy to investigate words passed on the block yourself.

Note that if all you want to achieve is to call some Python code from G-code, there is the somewhat easier way of calling Python functions like O-word procedures.

#### 6.6.3.3 The argspec parameter

The argument specification (keyword `argspec`) describes required and optional words to be passed to an ngc procedure, as well as optional preconditions for that code to execute.

An argspec consists of 0 or more characters of the class `[@A-KMNP-Za-kmnp-z^>]` . It can by empty (like `argspec=`).

An empty argspec, or no argspec argument at all implies the remapped code does not receive any parameters from the block. It will ignore any extra parameters present.

Note that RS274NGC rules still apply - for instance you may use axis words (eg X,Y,Z) only in the context of a G-code.

**ABCDEFGHIJKMPQRSTUVWXYZ**
Defines a required word parameter: an uppercase letter specifies that the corresponding word **must** be present in the current block. The word`s value will be passed as a local named parameter with a corresponding name. If the `@` character is present in the argspec, it will be passed as positional parameter, see below.

**abcdefghijkmpqrstuvwxyz**
Defines an optional word parameter: a lowercase letter specifies that the corresponding word **may** be present in the current block. If the word is present, the word's value will be passed as a local named parameter. If the `@` character is present in the argspec, it will be passed as positional parameter, see below.

**@**
The `@` (at-sign) tells argspec to pass words as positional parameters, in the order defined following the `@` option. Note that when using positional parameter passing, a procedure cannot tell whether a word was present or not, see example below.

> **Tip**
> this helps with packaging existing NGC procedures as remapped codes. Existing procedures do expect positional parameters. With the `@` option, you can avoid rewriting them to refer to local named parameters.

**^**
The `^` (caret) character specifies that the current spindle speed must be greater than zero (spindle running), otherwise the code fails with an appropriate error message.

**>**
The > (greater-than) character specifies that the current feed must be greater than zero, otherwise the code fails with an appropriate error message.

**n**
The `n` (greater-than) character specifies to pass the current line number in the `n`local named parameter.

By default, parameters are passed as local named parameter to an NGC procedure. These local parameters appear as *already set* when the procedure starts executing, which is different from existing semantics (local variables start out with value 0.0 and need to be explicitly assigned a value).

Optional word parameters may be tested for presence by the `EXISTS(#<word>)` idiom.

**Example for named parameter passing to NGC procedures** Assume the code is defined as

`REMAP=M400 modalgroup=10 argspec=Pq ngc=m400`

and `m400.ngc` looks as follows:

```
o<m400> sub
(P is required since it's uppercase in the argspec)
(debug, P word=#<P>)
(the q argspec is optional since its lowercase in the argspec. Use as follows:)
o100 if [EXISTS[#<q>]]
    (debug, Q word set: #<q>)
o100 endif
o<m400> endsub
M2
```

- executing `M400` will fail with the message `user-defined M400:  missing:  P`
- executing `M400 P123` will display `P word=123.000000`
- executing `M400 P123 Q456` will display `P word=123.000000` and `Q word set:  456.000000`

**Example for positional parameter passing to NGC procedures** Assume the code is defined as

`REMAP=M410 modalgroup=10 argspec=@PQr ngc=m410`

and `m410.ngc` looks as follows:

```
o<m410> sub
(debug, [1]=#1 [2]=#2 [3]=#3)
o<m410> endsub
M2
```

- executing `M410 P10` will display `m410.ngc:  [1]=10.000000 [2]=0.000000`
- executing `M410 P10 Q20` will display `m410.ngc:  [1]=10.000000 [2]=20.000000`

NB: you lose the capability to distinguish more than one optional parameter word, and you cannot tell whether an optional parameter was present but had the value 0, or was not present at all.

**Simple example for named parameter passing to a Python function** It's possible to define new codes *without* any NGC procedure. Here's a simple first example, a more complex one can be found in the next section.

Assume the code is defined as

`REMAP=G88.6 modalgroup=1 argspec=XYZp python=g886`

This instructs the interpreter to execute the Python function `g886` in the `module_basename.remap` module which might look like so:

```python
from interpreter import INTERP_OK
from emccanon import MESSAGE

def g886(self, **words):
    for key in words:
        MESSAGE("word '%s' = %f" % (key, words[key]))
    if words.has_key('p'):
        MESSAGE("the P word was present")
    MESSAGE("comment on this line: '%s'" % (self.blocks[self.remap_level].comment))
    return INTERP_OK
```

Try this with out with: g88.6 x1 y2 z3 g88.6 x1 y2 z3 p33 (a comment here)

You'll notice the gradual introduction of the embedded Python environment - see here for details. Note that with Python remapping functions, it make no sense to have Python prolog or epilog functions since it's executing a Python function in the first place.

**Advanced example: Remapped codes in pure Python** The `interpreter` and `emccanon` modules expose most of the Interpreter and some Canon internals, so many things which so far required coding in *C/C+\+* can be now be done in Python.

The following example is based on the `nc_files/involute.py` script - but canned as a G-code with some parameter extraction and checking. It also demonstrates calling the interpreter recursively (see `self.execute()`).

Assuming a definition like so (NB: this does not use argspec):

```
REMAP=G88.1 modalgroup=1 py=involute
```

The `involute` function in `python/remap.py` listed below does all word extraction from the current block directly. Note that interpreter errors can be translated to Python exceptions. Remember this is *readahead time* - execution time errors cannot be trapped this way.

```
import sys
import traceback
from math import sin,cos

from interpreter import *
from emccanon import MESSAGE
from util import lineno, call_pydevd
# raises InterpreterException if execute() or read() fails
throw_exceptions = 1

def involute(self, **words):
    """ remap function with raw access to Interpreter internals """

    if self.debugmask & 0x20000000: call_pydevd() # USER2 debug flag

    if equal(self.feed_rate,0.0):
        return "feedrate > 0 required"

    if equal(self.speed,0.0):
        return "spindle speed > 0 required"

    plunge = 0.1 # if Z word was given, plunge - with reduced feed

    # inspect controlling block for relevant words
    c = self.blocks[self.remap_level]
    x0 = c.x_number if c.x_flag else 0
    y0 = c.y_number if c.y_flag else 0
    a  = c.p_number if c.p_flag else 10
    old_z = self.current_z

    if self.debugmask & 0x10000000:
        print "x0=%f y0=%f a=%f old_z=%f" % (x0,y0,a,old_z)

    try:
        #self.execute("G3456")  # would raise InterpreterException
        self.execute("G21",lineno())
        self.execute("G64 P0.001",lineno())
        self.execute("G0 X%f Y%f" % (x0,y0),lineno())

        if c.z_flag:
            feed = self.feed_rate
            self.execute("F%f G1 Z%f" % (feed * plunge, c.z_number),lineno())
            self.execute("F%f" % (feed),lineno())
```

```
        for i in range(100):
            t = i/10.
            x = x0 + a * (cos(t) + t * sin(t))
            y = y0 + a * (sin(t) - t * cos(t))
            self.execute("G1 X%f Y%f" % (x,y),lineno())

        if c.z_flag: # retract to starting height
            self.execute("G0 Z%f" % (old_z),lineno())

    except InterpreterException,e:
        msg = "%d: '%s' - %s" % (e.line_number,e.line_text, e.error_message)
  return msg

    return INTERP_OK
```

The examples described so far can be found in *configs/sim/axis/remap/getting-started* with complete working configurations.

### 6.6.4 Upgrading an existing configuration for remapping

The minimal prerequisites for using `REMAP` statements are as follows:

- the Python plug in must be activated by specifying a `[PYTHON]TOPLEVEL=<path-to-toplevel-script>` in the ini file.
- the toplevel script needs to import the `remap` module, which can be initially empty, but the import needs to be in place.
- The Python interpreter needs to find the remap.py module above, so the path to the directory where your Python modules live needs to be added with `[PYTHON]PATH_APPEND=<path-to-your-local-Python-directory>`
- Recommended: import the `stdglue` handlers in the `remap` module. In this case Python also needs to find `stdglue.py` - we just copy it from the distribution so you can make local changes as needed. Depending on your installation the path to `stdglue.py` might vary.

Assuming your configuration lives under `/home/user/xxx` and the ini file is `/home/user/xxx/xxx.ini`, execute the following commands.

```
$ cd /home/user/xxx
$ mkdir python
$ cd python
$ cp /usr/share/linuxcnc/ncfiles/remap_lib/python-stdglue/stdglue.py .
$ echo 'from stdglue import *' >remap.py
$ echo 'import remap' >toplevel.py
```

Now edit `/home/user/xxx/xxx.ini` and add the following:

```
[PYTHON]
TOPLEVEL=/home/user/xxx/python/toplevel.py
PATH_APPEND=/home/user/xxx/python
```

Now verify that LinuxCNC comes up with no error messages - from a terminal window execute:

```
$ cd /home/user/xxx
$ linuxcnc xxx.ini
```

### 6.6.5 Remapping tool change-related codes: T, M6, M61

#### 6.6.5.1 Overview

If you are unfamiliar with LinuxCNC internals, first read the How tool change currently works section (dire but necessary).

Note than when remapping an existing code, we completely disable this codes' built in functionality of the interpreter.

So our remapped code will need to do a bit more than just generating some commands to move the machine as we like - it will also need to replicate those steps from this sequence which are needed to keep the interpreter and task happy.

However, this does **not** affect the processing of tool change-related commands in task and iocontrol. This means when we execute step 6b this will still cause iocontrol to do its thing.

Decisions, decisions:

- Do we want to use an O-word procedure or do it all in Python code?
- Is the iocontrol HAL sequence (tool-prepare/tool-prepared and tool-change/tool-changed pins) good enough or do we need a different kind of HAL interaction for our tool changer (for example: more HAL pins involved with a different interaction sequence)?

Depending on the answer, we have four different scenarios:

- When using an O-word procedure, we need prolog and epilog functions
- if using all Python code and no O-word procedure, a Python function is enough
- when using the iocontrol pins, our O-word procedure or Python code will contain mostly moves
- when we need a more complex interaction than offered by iocontrol, we need to completely define our own interaction, using `motion.digital*` and `motion.analog*` pins, and essentially ignore the iocontrol pins by looping them.

---

**Note**
If you hate O-word procedures and love Python, you're free to do it all in Python, in which case you would just have a `python=<function>` spec in the REMAP statement. But assuming most folks would be interested in using O-word procedures because they are more familiar with that, we'll do that as the first example.

---

So the overall approach for our first example will be:

1. we'd like to do as much as possible with G-code in an O-word procedure for flexibility. That includes all HAL interaction which would normally be handled by iocontrol - because we rather would want to do clever things with moves, probes, HAL pin I/O and so forth.
2. we'll try to minimize Python code to the extent needed to keep the interpreter happy, and cause task to actually do anything. That will go into the `prolog` and `epilog` Python functions.

#### 6.6.5.2 Understanding the role of iocontrol with remapped tool change codes

Iocontrol provides two HAL interaction sequences we might or might not use:

- when the NML message queued by a SELECT_POCKET() canon command is executed, this triggers the "raise tool-prepare and wait for tool-prepared to become high" HAL sequence in iocontrol, besides setting the XXXX pins
- when the NML message queued by the CHANGE_TOOL() canon command is executed, this triggers the "raise tool-change and wait for tool-changed to become high" HAL sequence in iocontrol, besides setting the XXXX pins

What you need to decide is whether the existing iocontrol HAL sequences are sufficient to drive your changer. Maybe you need a different interaction sequence - for instance more HAL pins, or maybe a more complex interaction. Depending on the answer, we might continue to use the existing iocontrol HAL sequences, or define our own ones.

For the sake of documentation, we'll disable these iocontrol sequences, and roll our own - the result will look and feel like the existing interaction, but now we have complete control over them because they are executed in our own O-word procedure.

So what we'll do is use some `motion.digital-*` and `motion.analog-*` pins, and the associated `M62` .. `M68` commands to do our own HAL interaction in our O-word procedure, and those will effectively replace the iocontrol *tool-prepare/tool-prepared* and *tool-change/tool-changed* sequences. So we'll define our pins replacing existing iocontrol pins functionally, and go ahead and make the iocontrol interactions a loop. We'll use the following correspondence in our example:

Iocontrol pin correspondence in the examples

| **iocontrol.0 pin** | **motion pin** |
|---|---|
| tool-prepare | digital-out-00 |
| tool-prepared | digital-in-00 |
| tool-change | digital-out-01 |
| tool-changed | digital-in-01 |
| tool-prep-number | analog-out-00 |
| tool-prep-pocket | analog-out-01 |
| tool-number | analog-out-02 |

Let us assume you want to redefine the M6 command, and replace it by an O-word procedure, but other than that things *should continue to work*.

So what our O-word procedure would do is to replace the steps outlined here. Looking through these steps you'll find that NGC code can be used for most of them, but not all. So the stuff NGC cant handle will be done in Python prolog and epilog functions.

#### 6.6.5.3 Specifying the M6 replacement

To convey the idea, we just replace the built in M6 semantics with our own. Once that works, you may go ahead and place any actions you see fit into the O-word procedure.

Going through the steps, we find:

1. check for T command already executed - **execute in Python prolog**
2. check for cutter compensation being active - **execute in Python prolog**
3. stop the spindle if needed - **can be done in NGC**
4. quill up - **can be done in NGC**
5. if TOOL_CHANGE_AT_G30 was set:
   a. move the A, B and C indexers if applicable - **can be done in NGC**
   b. generate rapid move to the G30 position - **can be done in NGC**
6. send a CHANGE_TOOL Canon command to task - **execute in Python epilog**
7. set the numberer parameters 5400-5413 according to the new tool - **execute in Python epilog**
8. signal to task to stop calling the interpreter for readahead until tool change complete - **execute in Python epilog**

So we need a prolog, and an epilog. Lets assume our ini file incantation of the M6 remap looks as follows:

```
REMAP=M6   modalgroup=6  prolog=change_prolog ngc=change epilog=change_epilog
```

So the prolog covering steps 1 and 2 would look like so - we decide to pass a few variables to the remap procedure which can be inspected and changed there, or used in a message. Those are: `tool_in_spindle`, `selected_tool` (tool numbers) and their respective pockets `current_pocket` and `selected_pocket`:

```
def change_prolog(self, **words):
    try:
  if self.selected_pocket < 0:
            return "M6: no tool prepared"

  if self.cutter_comp_side:
            return "Cannot change tools with cutter radius compensation on"

  self.params["tool_in_spindle"] = self.current_tool
  self.params["selected_tool"] = self.selected_tool
  self.params["current_pocket"] = self.current_pocket
        self.params["selected_pocket"] = self.selected_pocket
        return INTERP_OK
    except Exception, e:
        return "M6/change_prolog: %s" % (e)
```

You will find that most prolog functions look very similar: first test that all preconditions for executing the code hold, then prepare the environment - inject variables and/or do any preparatory processing steps which cannot easily be done in NGC code; then hand off to the NGC procedure by returning INTERP_OK.

Our first iteration of the O-word procedure is unexciting - just verify we got parameters right, and signal success by returning a positive value; steps 3-5 would eventually be covered here (see here for the variables referring to ini file settings):

```
O<change> sub
(debug, change: current_tool=#<current_tool>)
(debug, change: selected_pocket=#<selected_pocket>)
;
; insert any g-code which you see fit here, eg:
; G0  #<_ini[setup]tc_x>  #<_ini[setup]tc_y>  #<_ini[setup]tc_z>
;
O<change> endsub [1]
m2
```

Assuming success of `change.ngc`, we need to mop up steps 6-8:

```
def change_epilog(self, **words):
    try:
        if self.return_value > 0.0:
            # commit change
            self.selected_pocket =  int(self.params["selected_pocket"])
            emccanon.CHANGE_TOOL(self.selected_pocket)
            # cause a sync()
            self.tool_change_flag = True
            self.set_tool_parameters()
            return INTERP_OK
        else:
            return "M6 aborted (return code %.1f)" % (self.return_value)

    except Exception, e:
        return "M6/change_epilog: %s" % (e)
```

This replacement M6 is compatible with the built in code, except steps 3-5 need to be filled in with your NGC code.

Again, most epilogs have a common scheme: first, determine whether things went right in the remap procedure, then do any commit and cleanup actions which cant be done in NGC code.

#### 6.6.5.4 Configuring iocontrol with a remapped M6

Note that the sequence of operations has changed: we do everything required in the O-word procedure - including any HAL pin setting/reading to get a changer going, and to acknowledge a tool change - likely with `motion.digital-*` and `motion-analog-*` IO pins. When we finally execute the `CHANGE_TOOL()` command, all movements and HAL interactions are already completed.

Normally only now iocontrol would do its thing as outlined here. However, we don't need the HAL pin wiggling anymore - all iocontrol is left to do is to accept we're done with prepare and change.

This means that the corresponding iocontrol pins have no function any more. Therefore, we configure iocontrol to immediately acknowledge a change by configuring like so:

```
# loop change signals when remapping M6
net tool-change-loop iocontrol.0.tool-change iocontrol.0.tool-changed
```

If you for some reason want to remap `Tx` (prepare), the corresponding iocontrol pins need to be looped as well.

#### 6.6.5.5 Writing the change and prepare O-word procedures

The standard prologs and epilogs found in `ncfiles/remap_lib/python-stdglue/stdglue.py` pass a few *exposed parameters* to the remap procedure.

An *exposed parameter* is a named local variable visible in a remap procedure which corresponds to interpreter-internal variable which is relevant for the current remap. Exposed parameters are set up in the respective prolog, and inspected in the epilog. They can be changed in the remap procedure and the change will be picked up in the epilog. The exposed parameters for remappable built in codes are:

- `T` (prepare_prolog): `#<tool>` , `#<pocket>`
- `M6` (change_prolog): `#<tool_in_spindle>`, `#<selected_tool>`, `#<current_pocket>`, `#<selected_pocket>`
- `M61` (settool_prolog): `#<tool>` , `#<pocket>`
- `S` (setspeed_prolog): `#<speed>`
- `F` (setfeed_prolog): `#<feed>`

If you have specific needs for extra parameters to be made visible, that can simply be added to the prolog - practically all of the interpreter internals are visible to Python.

#### 6.6.5.6 Making minimal changes to the built in codes, including M6

Remember that normally remapping a code completely disables all internal processing for that code.

However, in some situations it might be sufficient to add a few codes around the existing `M6` built in implementation, like a tool length probe, but other than that retain the behavior of the built in `M6`.

Since this might be a common scenario, the built in behavior of remapped codes has been made available within the remap procedure. The interpreter detects that you are referring to a remapped code within the procedure which is supposed to redefine its behavior. In this case, the built in behavior is used - this currently is enabled for the set: `M6`, `M61`,`T`, `S`, `F`). Note that otherwise referring to a code within its own remap procedure would be a error - a `remapping recursion`.

Slightly twisting a built in would look like so (in the case of `M6`):

```
REMAP=M6   modalgroup=6  ngc=mychange
```

```
o<mychange> sub
M6 (use built in M6 behavior)
(.. move to tool length switch, probe and set tool length..)
o<mychange> endsub
m2
```

**Caution**
when redefining a built in code, **do not specify any leading zeroes in G- or M-codes** - for example, say `REMAP=M1 ..`, not `REMAP=M01 ....`

See the `configs/sim/axis/remap/extend-builtins` directory for a complete configuration which is the recommended starting point for own work when extending built in codes.

#### 6.6.5.7 Specifying the T (prepare) replacement

If you're confident with the default implementation, you wouldn't need to do this. But remapping is also a way to work around deficiencies in the current implementation, for instance to not block until the "tool-prepared" pin is set.

What you could do, for instance, is: - in a remapped T, just set the equivalent of the "tool-prepare" pin, but **not** wait for "tool-prepared" here - in the corresponding remapped M6, wait for the "tool-prepared" at the very beginning of the O-word procedure.

Again, the iocontrol tool-prepare/tool-prepared pins would be unused and replaced by `motion.*` pins, so those would pins must be looped:

```
# loop prepare signals when remapping T
net tool-prep-loop iocontrol.0.tool-prepare iocontrol.0.tool-prepared
```

So, here's the setup for a remapped T:

```
REMAP=T  prolog=prepare_prolog epilog=prepare_epilog ngc=prepare
```

```
def prepare_prolog(self,**words):
    try:
        cblock = self.blocks[self.remap_level]
        if not cblock.t_flag:
            return "T requires a tool number"

        tool  = cblock.t_number
        if tool:
            (status, pocket) = self.find_tool_pocket(tool)
            if status != INTERP_OK:
                return "T%d: pocket not found" % (tool)
        else:
            pocket = -1 # this is a T0 - tool unload

        # these variables will be visible in the ngc oword sub
        # as #<tool> and #<pocket> local variables, and can be
        # modified there - the epilog will retrieve the changed
        # values
        self.params["tool"] = tool
        self.params["pocket"] = pocket

        return INTERP_OK
    except Exception, e:
        return "T%d/prepare_prolog: %s" % (int(words['t']), e)
```

The minimal ngc prepare procedure again looks like so:

```
o<prepare> sub
; returning a positive value to commit:
o<prepare> endsub [1]
m2
```

And the epilog:

```
def prepare_epilog(self, **words):
    try:
        if self.return_value > 0:
            self.selected_tool = int(self.params["tool"])
            self.selected_pocket = int(self.params["pocket"])
            emccanon.SELECT_POCKET(self.selected_pocket, self.selected_tool)
            return INTERP_OK
        else:
            return "T%d: aborted (return code %.1f)" % (int(self.params["tool"]),self. ↩
                return_value)

    except Exception, e:
        return "T%d/prepare_epilog: %s" % (tool,e)
```

prepare_prolog and prepare_epilog are part of the *standard glue* provided by *nc_files/remap_lib/python-stdglue/stdglue.py*. This module is intended to cover most standard remapping situations in a common way.

#### 6.6.5.8 Error handling: dealing with abort

The built in tool change procedure has some precautions for dealing with a program abort (e.g. hitting Escape in Axis during a change). Your remapped function has none of this, therefore some explicit cleanup might be needed if a remapped code is aborted. In particular, a remap procedure might establish modal settings which are undesirable to have active after an abort. For instance, if your remap procedure has motion codes (G0,G1,G38..) and the remap is aborted, then the last modal code will remain active. However, you very likely want to have any modal motion canceled when the remap is aborted.

The way to do this is by using the `[RS274NGC]ON_ABORT_COMMAND` feature. This ini option specifies a O-word procedure call which is executed if task for some reason aborts program execution.

```
[RS274NGC]
ON_ABORT_COMMAND=O <on_abort> call
```

The suggested on_abort procedure would look like so (adapt to your needs):

```
o<on_abort> sub

G54 (origin offsets are set to the default)
G17 (select XY plane)
G90 (absolute)
G94 (feed mode: units/minute)
M48 (set feed and speed overrides)
G40 (cutter compensation off)
M5  (spindle off)
G80 (cancel modal motion)
M9  (mist and coolant off)

o<on_abort> endsub
m2
```

**Caution**
Never use an `M2` in a O-word subroutine, including this one. It will cause hard-to-find errors. For instance, using an `M2` in a subroutine will not end the subroutine properly and will leave the subroutine NGC file open, not your main program.

Make sure `on_abort.ngc` is along the interpreter search path (recommended location: `SUBROUTINE_PATH` so as not to clutter your `NC_FILES` directory with internal procedures). `on_abort` receives a single parameter indicating the cause for calling the abort procedure, which might be used for conditional cleanup.

Statements in that procedure typically would assure that post-abort any state has been cleaned up, like HAL pins properly reset. For an example, see `configs/sim/axis/remap/rack-toolchange`.

Note that terminating a remapped code by returning INTERP_ERROR from the epilog (see previous section) will also cause the `on_abort` procedure to be called.

#### 6.6.5.9 Error handling: failing a remapped code NGC procedure

If you determine in your handler procedure that some error condition occurred, do not use `M2` to end your handler - see above:

If displaying an operator error message and stopping the current program is good enough, use the `(abort, <message>)` feature to terminate the handler with an error message. Note that you can substitute numbered, named, ini and HAL parameters in the text like in this example (see also `tests/interp/abort-hot-comment/test.ngc`):

```
o100 if [..] (some error condition)
     (abort, Bad Things! p42=#42 q=#<q> ini=#<_ini[a]x> pin=#<_hal[component.pin])
o100 endif
```

NB: ini and HAL variable expansion need explicit enabling with FEATURE.

If more fine grained recovery action is needed, use the idiom laid out in the previous example:

- define an epilog function, even if it's just to signal an error condition
- pass a negative value from the handler to signal the error
- inspect the return value in the epilog function.
- take any recovery action needed
- return the error message string from the handler, which will set the interpreter error message and abort the program (pretty much like `(abort, message=`

This error message will be displayed in the UI, and returning INTERP_ERROR will cause this error handled like any other runtime error.

Note that both `(abort, msg)` and returning INTERP_ERROR from an epilog will cause any ON_ABORT handler to be called as well if defined (see previous section).

### 6.6.6 Remapping other existing codes: S, M0, M1, M60

#### 6.6.6.1 Automatic gear selection be remapping S (set spindle speed)

A potential use for a remapped S code would be *automatic gear selection* depending on speed. In the remap procedure one would test for the desired speed attainable given the current gear setting, and change gears appropriately if not.

#### 6.6.6.2 Adjusting the behavior of M0, M1, M60

A use case for remapping M0/M1 would be to customize the behavior of the existing code. For instance, it could be desirable to turn off the spindle, mist and flood during an M0 or M1 program pause, and turn these settings back on when the program is resumed.

For a complete example doing just that, see *configs/sim/axis/remap/extend-builtins/*, which adapts M1 as laid out above.

### 6.6.7 Creating new G-code cycles

A G-code cycle as used here is meant to behave as follows:

- On first invocation, the associated words are collected and the G-code cycle is executed.
- If subsequent lines just continue parameter words applicable to this code, but no new G-code, the previous G code is re-executed with the parameters changed accordingly.

An example: Assume you have `G84.3` defined as remapped G code cycle with the following ini segment (see here for a detailed description of `cycle_prolog` and `cycle_epilog`):

```
[RS274NGC]
# A cycle with an oword procedure: G84.3 <X- Y- Z- Q- P->
REMAP=G84.3 argspec=xyzabcuvwpr prolog=cycle_prolog ngc=g843 epilog=cycle_epilog modalgroup ↩
    =1
```

Executing the following lines:

```
g17
(1)   g84.3 x1 y2 z3  r1
(2)   x3 y4 p2
(3)   x6 y7 z5
(4)   G80
```

causes the following (note *R* is sticky, and *Z* is sticky since the plane is *XY*):

1. `g843.ngc` is called with words x=1, y=2, z=3, r=1
2. `g843.ngc` is called with words x=3, y=4, z=3, p=2, r=1
3. `g843.ngc` is called with words x=6, y=7, z=3, r=1
4. The `G84.3` cycle is canceled.

Besides creating new cycles, this provides an easy method for repackaging existing G-codes which do not behave as cycles. For instance, the `G33.1` Rigid Tapping code does not behave as a cycle. With such a wrapper, a new code can be easily created which uses `G33.1` but behaves as a cycle.

See *configs/sim/axis/remap/cycle* for a complete example of this feature. It contains two cycles, one with an NGC procedure like above, and a cycle example using just Python.

### 6.6.8 Configuring Embedded Python

The Python plugin serves both the interpreter, and task if so configured, and hence has its own section `PYTHON` in the ini file.

#### 6.6.8.1 Python plugin : ini file configuration

`[PYTHON]`

**TOPLEVEL=<*filename*>**
: filename of the initial Python script to execute on startup. This script is responsible for setting up the package name structure, see below.

**PATH_PREPEND=<*directory*>**
: prepend this directory to `PYTHON_PATH`. A repeating group.

**PATH_APPEND=<*directory*>**
: append this directory to `PYTHON_PATH`. A repeating group.

**LOG_LEVEL=<*integer*>**
log level of plugin-related actions. Increase this if you suspect problems. Can be very verbose.

**RELOAD_ON_CHANGE=*[0|1]***
reload the *TOPLEVEL* script if the file was changed. Handy for debugging but currently incurs some runtime overhead. Turn this off for production configurations.

**PYTHON_TASK=*[0|1]***
Start the Python task plug in. Experimental. See xxx.

#### 6.6.8.2 Executing Python statements from the interpreter

For ad-hoc execution of commands the Python *hot comment* has been added. Python output by default goes to stdout, so you need to start LinuxCNC from a terminal window to see results. Example (eg. in the MDI window):

```
;py,print 2*3
```

Note that the interpreter instance is available here as `self`, so you could also run:

```
;py,print self.tool_table[0].toolno
```

The `emcStatus` structure is accessible, too:

```
;py,from emctask import *
;py,print emcstat.io.aux.estop
```

### 6.6.9 Programming Embedded Python in the RS274NGC Interpreter

#### 6.6.9.1 The Python plugin namespace

The namespace is expected to be laid out as follows:

**oword**
Any callables in this module are candidates for Python O-word procedures. Note that the Python `oword` module is checked **before** testing for a NGC procedure with the same name - in effect names in `oword` will hide NGC files of the same basename.

**remap**
Python callables referenced in an argspec `prolog`,`epilog` or `python` option are expected to be found here.

**namedparams**
Python funtcions int this module extend or redefine the namespace of predefined named parameters, see adding predefined parameters.

**task**
Task-related callables are expected here.

#### 6.6.9.2 The Interpreter as seen from Python

The interpreter is an existing C++ class (*Interp*) defined in *src/emc/rs274ngc*. Conceptually all `oword.<function>` and `remap.<function>` Python calls are methods of this Interp class, although there is no explicit Python definition of this class (it's a *Boost.Python* wrapper instance) and hence receive the as the first parameter `self` which can be used to access internals.

#### 6.6.9.3 The Interpreter `__init__` and `__delete__` functions

If the `TOPLEVEL` module defines a function `__init__`, it will be called once the interpreter is fully configured (ini file read, and state synchronized with the world model).

If the `TOPLEVEL` module defines a function `__delete__`, it will be called once before the interpreter is shutdown and after the persistent parameters have been saved to the `PARAMETER_FILE`.

Note_ at this time, the `__delete__` handler does not work for interpreter instances created by importing the `gcode` module. If you need an equivalent functionality there (which is quite unlikely), please consider the Python `atexit` module.

```
# this would be defined in the TOPLEVEL module

def __init__(self):
    # add any one-time initialization here
    if self.task:
  # this is the milltask instance of interp
  pass
    else:
  # this is a non-milltask instance of interp
        pass

def __delete__(self):
    # add any cleanup/state saving actions here
    if self.task: # as above
  pass
    else:
        pass
```

This function may be used to initialize any Python-side attributes which might be needed later, for instance in remap or oword functions, and save or restore state beyond what `PARAMETER_FILE` provides.

If there are setup or cleanup actions which are to happen only in the milltask Interpreter instance (as opposed to the interpreter instance which sits in the `gcode` Python module and serves preview/progress display purposes but nothing else), this can be tested for by evaluating self.task.

An example use of `__init__` and `__delete__` can be found in *configs/sim/axis/remap/cycle/python/toplevel.py* initialising attributes needed to handle cycles in *ncfiles/remap_lib/python-stdglue/stdglue.py* (and imported into *configs/sim/axis/remap/cycle/python/remap.py*).

#### 6.6.9.4 Calling conventions: NGC to Python

Python code is called from NGC in the following situations:

- during normal program execution:
  - when an O-word call like `O<proc> call` is executed and the name `oword.proc` is defined and callable
  - when a comment like `;py,<Python statement>` is executed
- during execution of a remapped code: any `prolog=`, `python=` and `epilog=` handlers.

**Calling O-word Python subroutines**

Arguments:

**`self`**
: the interpreter instance

**`*args`**
: the list of actual positional parameters. Since the number of actual parameters may vary, it is best to use this style of declaration:

```
# this would be defined in the oword module
def mysub(self, *args):
    print "number of parameters passed:", len(args)
    for a in args:
  print a
```

**Return values of O-word Python subroutines** Just as NGC procedures may return values, so do O-word Python subroutines. They are expected to either:

- return no value (no `return` statement or the value `None`)
- a float or int value
- a string, this means *this is an error message, abort the program*. Works like `(abort, msg)`.

Any other return value type will raise a Python exception.

In a calling NGC environment, the following predefined named parameters are available:

**#<_value>**
: value returned by the last procedure called. Initialized to 0.0 on startup. Exposed in Interp as `self.return_value` (float).

**#<_value_returned>**
: indicates the last procedure called did `return`or `endsub` with an explicit value. 1.0 if true. Set to 0.0 on each `call`. Exposed in Interp was `self.value_returned` (int).

See also `tests/interp/value-returned` for an example.

**Calling conventions for prolog= and epilog= subroutines** Arguments are:

**self**
: the interpreter instance

**words**
: keyword parameter dictionary. If an argspec was present, words are collected from the current block accordingly and passed in the dictionary for convenience (the words could as well be retrieved directly from the calling block, but this requires more knowledge of interpreter internals). If no argspec was passed, or only optional values were specified and none of these was present in the calling block, this dict is empty. Word names are converted to lowercase.

Example call:

```
def minimal_prolog(self, **words): # in remap module
    print len(words)," words passed"
    for w in words:
        print "%s: %s" % (w, words[w])
    if words['p'] < 78: # NB: could raise an exception if p were optional
       return "failing miserably"
    return INTERP_OK
```

Return values:

**INTERP_OK**
: return this on success. You need to import this from `interpreter`.

**"a message text"**
: returning a string from a handler means *this is an error message, abort the program*. Works like `(abort, msg)`.

.

**Calling conventions for python= subroutines** Arguments are:

**self**
: the interpreter instance

**words**
: keyword parameter dictionary. the same kwargs dictionary as prologs and epilogs (see above).

The minimum `python=` function example:

```
def useless(self,  **words): # in remap module
    return INTERP_OK
```

Return values:

**INTERP_OK**
: return this on success

**"a message text"**
: returning a string from a handler means *this is an error message, abort the program*. Works like `(abort, msg)`.

If the handler needs to execute a *queuebuster operation* (tool change, probe, HAL pin reading) it is supposed to suspend execution with the following statement:

**yield INTERP_EXECUTE_FINISH**
: This signals task to stop read ahead, execute all queued operations, execute the *queue-buster* operation, synchronize interpreter state with machine state, and then signal the interpreter to continue. At this point the function is resumed at the statement following the `yield ..` statement.

**Dealing with queue-buster: Probe, Tool change and waiting for a HAL pin** Queue busters interrupt a procedure at the point where such an operation is called, hence the procedure needs to be restarted after the interpreter synch(). When this happens the procedure needs to know if it is restarted, and where to continue. The Python generator method is used to deal with procedure restart.

This demonstrates call continuation with a single point-of-restart:

```
def read_pin(self,*args):
    # wait 5secs for digital-input 00 to go high
    emccanon.WAIT(0,1,2,5.0)
    # cede control after executing the queue buster:
    yield INTERP_EXECUTE_FINISH
    # post-sync() execution resumes here:
    pin_status = emccanon.GET_EXTERNAL_DIGITAL_INPUT(0,0);
    print "pin status=",pin_status
```

**Warning**
The *yield* feature is fragile. The following restrictions apply to the usage of *yield INTERP_EXECUTE_FINISH*:

- Python code executing a *yield INTERP_EXECUTE_FINISH* must be part of a remap procedure. Yield does not work in a Python oword procedure.
- A Python remap subroutine containing *yield INTERP_EXECUTE_FINISH* statement may not return a value, as with normal Python yield statements.
- Code following a yield may not recursively call the interpreter, like with self.execute("<mdi command>"). This is an architectural restriction of the interpreter and is not fixable without a major redesign.

#### 6.6.9.5 Calling conventions: Python to NGC

NGC code is executed from Python when:

- the method `self.execute(<NGC code>[,<line number>])` is executed
- during execution of a remapped code, if a `prolog=` function is defined, the NGC procedure given in `ngc=` is executed immediately thereafter.

The prolog handler does not call the handler, but it prepares its call environment, for instance by setting up predefined local parameters.

**Inserting parameters in a prolog, and retrieving them in an epilog** Conceptually a prolog and an epilog execute at the same call level like the O-word procedure, that is: after the subroutine call is set up, and before the subroutine endsub or return.

This means that any local variable created in a prolog will be a local variable in the O-word procedure, and any local variables created in the O-word procedure are still accessible when the epilog executes.

The `self.params` array handles reading and setting numbered and named parameters. If a named parameter begins with _ (underscore), it is assumed to be a global parameter; if not, it is local to the calling procedure. Also, numbered parameters in the range 1..30 are treated like local variables; their original values are restored on return/endsub from an O-word procedure.

Here is an example remapped code demonstrating insertion and extraction of parameters into/from the O-word procedure:

```
REMAP=m300 prolog=insert_param ngc=testparam epilog=retrieve_param modalgroup=10
```

```
def insert_param(self, **words): # in the remap module
    print "insert_param call level=",self.call_level
    self.params["myname"] = 123
    self.params[1] = 345
    self.params[2] = 678
    return INTERP_OK

def retrieve_param(self, **words):
    print "retrieve_param call level=",self.call_level
    print "#1=", self.params[1]
    print "#2=", self.params[2]
    try:
        print "result=", self.params["result"]
    except Exception,e:
  return "testparam forgot to assign #<result>"
    return INTERP_OK
```

```
o<testparam> sub
(debug, call_level=#<_call_level> myname=#<myname>)
; try commenting out the next line and run again
#<result> = [#<myname> * 3]
#1 = [#1 * 5]
#2 = [#2 * 3]
o<testparam> endsub
m2
```

`self.params()` returns a list of all variable names currently defined. Since `myname` is local, it goes away after the epilog finishes.

**Calling the interpreter from Python** You can recursively call the interpreter from Python code as follows:

```
self.execute(<NGC code>[,<line number>])
```

Examples:

```
self.execute("G1 X%f Y%f" % (x,y))
self.execute("O <myprocedure> call", currentline)
```

You might want to test for the return value being < `INTERP_MIN_ERROR`. If you're using lots of execute() statements, it's probably easier to trap InterpreterException as per below.

**Caution**
The parameter insertion/retrieval method described in the previous section does not work in this case. It is good enough for just executing simple NGC commands or a procedure call and advanced introspection into the procedure, and passing of local named parameters is not needed. The recursive call feature is fragile.

**Interpreter Exception during execute()** if `interpreter.throw_exceptions` is nonzero (default 1), and self.execute() returns an error, the exception `InterpreterException` is raised. InterpreterException has the following attributes:

**`line_number`**
: where the error occurred

**`line_text`**
: the NGC statement causing the error

**`error_message`**
: the interpreter's error message

Errors can be trapped in the following Pythonic way:

```
import interpreter
interpreter.throw_exceptions = 1
   ...
   try:
        self.execute("G3456")  #  raise InterpreterException

   except InterpreterException,e:
        msg = "%d: '%s' - %s" % (e.line_number,e.line_text, e.error_message)
        return msg  # replace builtin error message
```

**Canon** The canon layer is practically all free functions. Example:

```
import emccanon
def example(self,*args):
    ....
    emccanon.STRAIGHT_TRAVERSE(line,x0,y0,z0,0,0,0,0,0,0)
    emccanon.STRAIGHT_FEED(line,x1,y1,z1,0,0,0,0,0,0)
    ...
    return INTERP_OK
```

The actual canon functions are declared in `src/emc/nml_intf/canon.hh` and implemented in `src/emc/task/emccanon.cc`
The implementation of the Python functions can be found in `src/emc/rs274ncg/canonmodule.cc`.

#### 6.6.9.6 Built in modules

The following modules are built in:

**`interpreter`**
: exposes internals of the Interp class. See `src/emc/rs274ngc/interpmodule.cc`, and the `tests/remap/introspect` regression test.

**emccanon**
exposes most calls of `src/emc/task/emccanon.cc`.

**emctask**
exposes the `emcStatus` class instance. See `src/emc/task/taskmodule.cc`. Not present when using the `gcode` module used for user interfaces - only present in the milltask instance of the interpreter.

### 6.6.10 Adding Predefined Named Parameters

The interpreter comes with a set of predefined named parameters for accessing internal state from the NGC language level. These parameters are read-only and global, and hence cannot be assigned to.

Additional parameters may be added by defining a function in the `namedparams` module. The name of the function defines the name of the new predefined named parameter, which now can be referenced in arbitrary expressions.

To add or redefine a named parameter:

- add a `namedparams` module so it can be found by the interpreter
- define new parameters by functions (see below). These functions receive `self` (the interpreter instance) as parameter and so can access aribtrary state. Arbitrary Python capabilities can be used to return a value.
- import that module from the `TOPLEVEL` script

```
# namedparams.py
# trivial example
def _pi(self):
    return 3.1415926535
```

```
#<circumference> = [2 * #<radius> * #<_pi>]
```

Functions in `namedparams.py` are expected to return a float or int value. If a string is returned, this sets the interpreter error message and aborts execution.

Ònly functions with a leading underscore are added as parameters, since this is the RS274NGC convention for globals.

It is possible to redefine an existing predefined parameter by adding a Python function of the same name to the `namedparams` module. In this case, a warning is generated during startup.

While the above example isnt terribly useful, note that pretty much all of the interpreter internal state is accessible from Python, so arbitrary predicates may be defined this way. For a slightly more advanced example, see `tests/remap/predefined-named-para`

### 6.6.11 Standard Glue routines

Since many remapping tasks are very similar, I've started collecting working prolog and epilog routines in a single Python module. These can currently be found in *ncfiles/remap_lib/python-stdglue/stdglue.py* and provide the following routines:

#### 6.6.11.1 T: `prepare_prolog` and `prepare_epilog`

These wrap a NGC procedure for Tx Tool Prepare.

**Actions of prepare_prolog** The following parameters are made visible to the NGC procedure:

- `#<tool>` - the parameter of the `T` word
- `#<pocket>` - the corresponding pocket

If tool number zero is requested (meaning Tool unload), the corresponding pocket is passed as -1.

It is an error if:

- no tool number is given as T parameter
- the tool cannot be found in the tool table.

Note that unless you set the `[EMCIO] RANDOM_TOOLCHANGER=1` parameter, tool and pocket number are identical, and the pocket number from the tool table is ignored. This is currently a restriction.

ACTIONS OF `PREPARE_EPILOG`

- The NGC procedure is expected to return a positive value, otherwise and error message containing the return value is given and the interpreter aborts.
- In case the NGC procedure executed the T command (which then refers to the built in T behavior), no further action is taken. This can be used for instance to minimally adjust the built in behavior be preceding or following it with some other statements.
- Otherwise, the `#<tool>` and `#<pocket>` parameters are extracted from the subroutine's parameter space. This means that the NGC procedure could change these values, and the epilog takes the changed values in account.
- then, the Canon command `SELECT_POCKET(#<pocket>,#<tool>)` is executed.

#### 6.6.11.2 M6: `change_prolog` and `change_epilog`

These wrap a NGC procedure for M6 Tool Change.

ACTIONS OF `CHANGE_PROLOG`

- The following three steps are applicable only if the `iocontrol-v2` component is used:
  - If parameter 5600 (fault indicator) is greater than zero, this indicates a Toolchanger fault, which is handled as follows:
  - if parameter 5601 (error code) is negative, this indicates a hard fault and the prolog aborts with an error message.
  - if parameter 5601 (error code) is greater equal zero, this indicates a soft fault. An informational message is displayed and the prolog continues.
- If there was no preceding T command which caused a pocket to be selected, the prolog aborts with an error message.
- If cutter radius compensation is on, the prolog aborts with an error message.

Then, the following parameters are exported to the NGC procedure:

- `#<tool_in_spindle>` : the tool number of the currently loaded tool
- `#<selected_tool>` : the tool number selected
- `#<selected_pocket>` : the selected tool's pocket number

ACTIONS OF `CHANGE_EPILOG`

- The NGC procedure is expected to return a positive value, otherwise and error message containing the return value is given and the interpreter aborts.
- If parameter 5600 (fault indicator) is greater than zero, this indicates a Toolchanger fault, which is handled as follows (`iocontrol-v2`-only):
  - if parameter 5601 (error code) is negative, this indicates a hard fault and the epilog aborts with an error message.
  - if parameter 5601 (error code) is greater equal zero, this indicates a soft fault. An informational message is displayed and the epilog continues.
- In case the NGC procedure executed the M6 command (which then refers to the built in M6 behavior), no further action is taken. This can be used for instance to minimally adjust the built in behavior be preceding or following it with some other statements.

- Otherwise, the `#<selected_pocket>` parameter is extracted from the subroutine's parameter space, and used to set the interpreter's `current_pocket` variable. Again, the procedure could change this value, and the epilog takes the changed value in account.
- then, the Canon command `CHANGE_TOOL(#<selected_pocket>)` is executed.
- The new tool parameters (offsets, diameter etc) are set.

#### 6.6.11.3 G code Cycles: `cycle_prolog` and `cycle_epilog`

These wrap a NGC procedure so it can act as a cycle, meaning the motion code is retained after finishing execution. If the next line just contains parameter words (e.g. new X,Y values), the code is executed again with the new parameter words merged into the set of the parameters given in the first invocation.

These routines are designed to work in conjunction with an argspec=<words> parameter. While this is easy to use, in a realistic scenario you would avoid argspec and do a more thorough investigation of the block manually in order to give better error messages.

The suggested argspec is as follows:

```
REMAP=G<somecode> argspec=xyzabcuvwqplr prolog=cycle_prolog ngc=<ngc procedure> epilog= ↩
    cycle_epilog modalgroup=1
```

This will permit `cycle_prolog` to determine the compatibility of any axis words give in the block, see below.

ACTIONS OF `CYCLE_PROLOG`

- Determine whether the words passed in from the current block fulfill the conditions outlined under Canned Cycle Errors.
  - export the axis words as `<x>`, `#<y>` etc; fail if axis words from different groups (XYZ) (UVW) are used together, or any of (ABC) is given.
  - export *L*- as `#<l>`; default to 1 if not given.
  - export *P*- as `#<p>`; fail if p less than 0.
  - export *R*- as `#<r>`; fail if r not given, or less equal 0 if given.
  - fail if feed rate is zero, or inverse time feed or cutter compensation is on.
- Determine whether this is the first invocation of a cycle G code, if so:
  - Add the words passed in (as per argspec) into a set of sticky parameters, which is retained across several invocations.
- If not (a continuation line with new parameters):
  - merge the words passed in into the existing set of sticky parameters.
- export the set of sticky parameters to the NGC procedure.

ACTIONS OF `CYCLE_EPILOG`

- Determine if the current code was in fact a cycle, if so:
  - retain the current motion mode so a continuation line without a motion code will execute the same motion code.

#### 6.6.11.4 S (Set Speed) : `setspeed_prolog` and `setspeed_epilog`

TBD

#### 6.6.11.5 F (Set Feed) : `setfeed_prolog` and `setfeed_epilog`

TBD

#### 6.6.11.6 M61 Set tool number : `settool_prolog` and `settool_epilog`

TBD

### 6.6.12 Remapped code execution

#### 6.6.12.1 NGC procedure call environment during remaps

Normally, an O-word procedure is called with positional parameters. This scheme is very limiting in particular in the presence of optional parameters. Therefore, the calling convention has been extended to use something remotely similar to the Python keyword arguments model.

see LINKTO gcode/main Subroutines: sub, endsub, return, call.

#### 6.6.12.2 Nested remapped codes

Remapped codes may be nested just like procedure calls - that is, a remapped code whose NGC procedure refers to some other remapped code will execute properly.

The maximum nesting level remaps is currently 10.

#### 6.6.12.3 Sequence number during remaps

Sequence numbers are propagated and restored like with O-word calls. See `tests/remap/nested-remaps/word` for the regression test, which shows sequence number tracking during nested remaps three levels deep.

#### 6.6.12.4 Debugging flags

The following flags are relevant for remapping and Python - related execution:

```
EMC_DEBUG_OWORD             0x00002000  traces execution of O-word subroutines
EMC_DEBUG_REMAP             0x00004000  traces execution of remap-related code
EMC_DEBUG_PYTHON            0x00008000  calls to the Python plug in
EMC_DEBUG_NAMEDPARAM        0x00010000  trace named parameter access
EMC_DEBUG_PYTHON_TASK       0x00040000  trace the task Python plug in
EMC_DEBUG_USER1             0x10000000  user-defined - not interpreted by LinuxCNC
EMC_DEBUG_USER2             0x20000000  user-defined - not interpreted by LinuxCNC
```

*or* these flags into the `[EMC]DEBUG` variable as needed. For a current list of debug flags see *src/emc/nml_intf/debugflags.h*.

#### 6.6.12.5 Debugging Embedded Python code

Debugging of embedded Python code is harder than debugging normal Python scripts, and only a limited supply of debuggers exists. A working open-source based solution is to use the Eclipse IDE, and the PydDev Eclipse plug in and its remote debugging feature.

To use this approach:

- install Eclipse via the the *Ubuntu Software Center* (choose first selection)

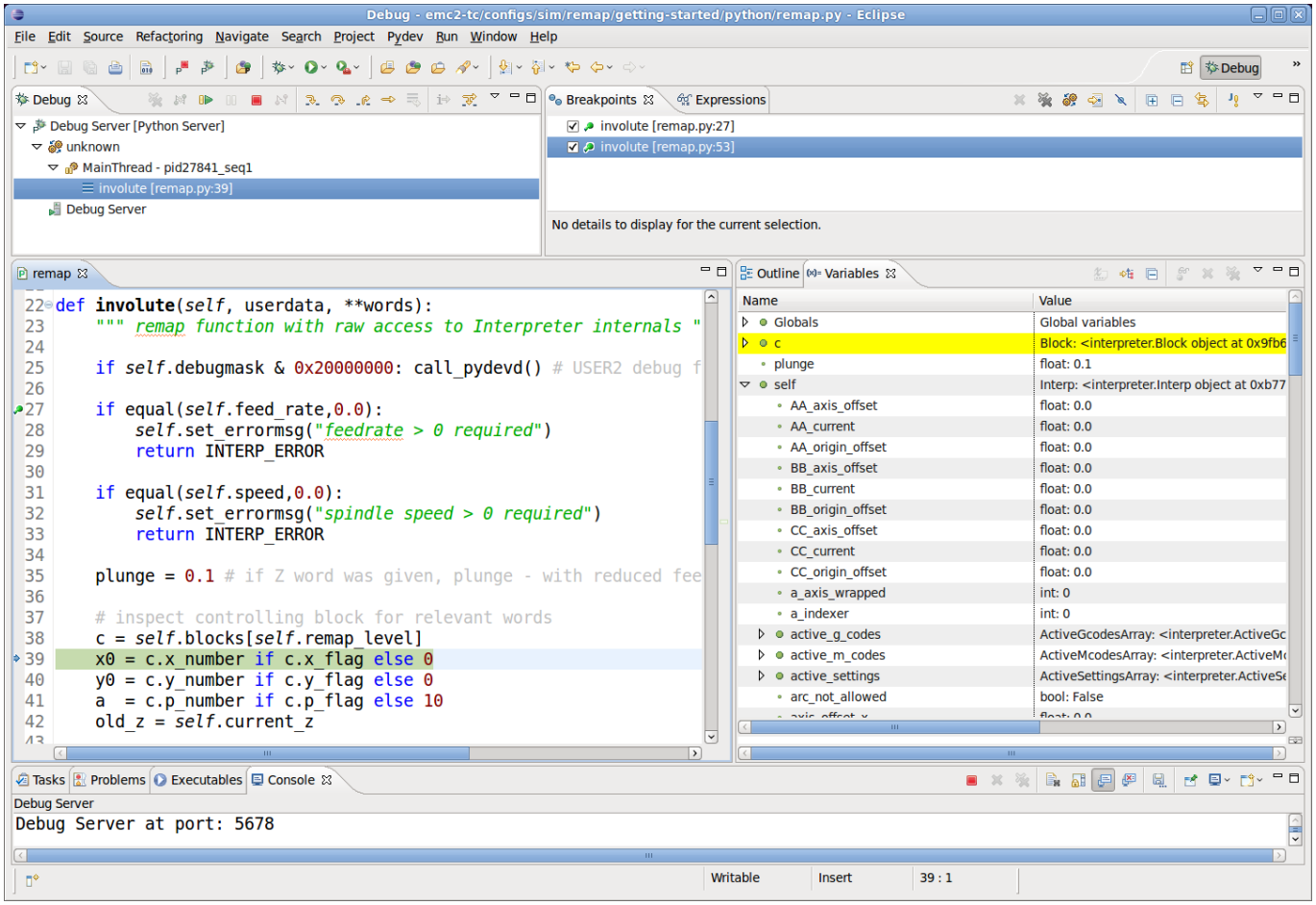
- install the PyDev plug in from the Pydev Update Site
- setup the LinuxCNC source tree as an Eclipse project
- start the Pydev Debug Server in Eclipse
- make sure the embedded Python code can find the `pydevd.py` module which comes with that plug in - it's buried somewhere deep under the Eclipse install directory. Set the the `pydevd` variable in `util.py` to reflect this directory location.
- `import pydevd` in your Python module - see example `util.py` and `remap.py`
- call `pydevd.settrace()` in your module at some point to connect to the Eclipse Python debug server - here you can set breakpoints in your code, inspect variables, step etc as usual.

**Caution**

`pydevd.settrace()` will block execution if Eclipse and the Pydev debug server have not been started.

To cover the last two steps: the `o<pydevd>` procedure helps to get into the debugger from MDI mode. See also the `call_pydevd` function in `util.py` and its usage in `remap.involute` to set a breakpoint.

Here's a screen-shot of Eclipse/PyDevd debugging the `involute` procedure from above:




See the Python code in `configs/sim/axis/remap/getting-started/python` for details.

### 6.6.13 Axis Preview and Remapped code execution

For complete preview of a remapped code's tool path some precautions need to be taken. To understand what is going on, let's review the preview and execution process (this covers the Axis case, but others are similar):

First, note that there are **two** independent interpreter instances involved:

- one instance in the milltask program, which executes a program when you hit the *Start* button, and actually makes the machine move
- a second instance in the user interface whose primary purpose is to generate the tool path preview. This one *executes* a program once it is loaded, but it doesn't actually cause machine movements.

Now assume that your remap procedure contains a G38 probe operation, for example as part of a tool change with automatic tool length touch off. If the probe fails, that would clearly be an error, so you'd display a message and abort the program.

Now, what about preview of this procedure? At preview time, of course it's not known whether the probe succeeds or fails - but you would likely want to see what the maximum depth of the probe is, and assume it succeeds and continues execution to preview further movements. Also, there is no point in displaying a *probe failed* message and aborting **during preview**.

The way to address this issue is to test in your procedure whether it executes in preview or execution mode. This can be checked for by testing the `#<_task>` predefined named parameter - it will be 1 during actual execution and 0 during preview. See *configs/sim/axis/remap/manual-toolchange-with-tool-length-switch/nc_subroutines/manual_change.ngc* for a complete usage example.

Within Embedded Python, the task instance can be checked for by testing *self.task* - this will be 1 in the milltask instance, and 0 in the preview instance(s).

### 6.6.14 Remappable Codes

#### 6.6.14.1 Existing codes which can be remapped

The current set of **existing** codes open to redefinition is:

- Tx (Prepare)
- M6 (Change tool)
- M61 (Set tool number)
- M0 (pause a running program temporarily)
- M1 (pause a running program temporarily if the optional stop switch is on)
- M60 (exchange pallet shuttles and then pause a running program temporarily)
- S (set spindle speed)
- F (set feed)

Note that the use of M61 currently requires the use of iocontrol-v2.

#### 6.6.14.2 Currently unallocated G-codes:

These codes are currently undefined in the current implementation of LinuxCNC and may be used to define new G-codes:

FIXTHIS too verbose

G0.1 G0.2 G0.3 G0.4 G0.5 G0.6 G0.7 G0.8 G0.9 G1.1 G1.2 G1.3 G1.4 G1.5 G1.6 G1.7 G1.8 G1.9 G2.1 G2.2 G2.3 G2.4 G2.5 G2.6 G2.7 G2.8 G2.9 G3.1 G3.2 G3.3 G3.4 G3.5 G3.6 G3.7 G3.8 G3.9 G4.1 G4.2 G4.3 G4.4 G4.5 G4.6 G4.7 G4.8 G4.9 G5.4 G5.5 G5.6 G5.7 G5.8 G5.9 G6 G6.1 G6.2 G6.3 G6.4 G6.5 G6.6 G6.7 G6.8 G6.9 G7.1 G7.2 G7.3 G7.4 G7.5 G7.6 G7.7 G7.8

G7.9 G8.1 G8.2 G8.3 G8.4 G8.5 G8.6 G8.7 G8.8 G8.9 G9 G9.1 G9.2 G9.3 G9.4 G9.5 G9.6 G9.7 G9.8 G9.9 G10.1 G10.2 G10.3 G10.4 G10.5 G10.6 G10.7 G10.8 G10.9 G11 G11.1 G11.2 G11.3 G11.4 G11.5 G11.6 G11.7 G11.8 G11.9 G12 G12.1 G12.2 G12.3 G12.4 G12.5 G12.6 G12.7 G12.8 G12.9 G13 G13.1 G13.2 G13.3 G13.4 G13.5 G13.6 G13.7 G13.8 G13.9 G14 G14.1 G14.2 G14.3 G14.4 G14.5 G14.6 G14.7 G14.8 G14.9 G15 G15.1 G15.2 G15.3 G15.4 G15.5 G15.6 G15.7 G15.8 G15.9 G16 G16.1 G16.2 G16.3 G16.4 G16.5 G16.6 G16.7 G16.8 G16.9 G17.2 G17.3 G17.4 G17.5 G17.6 G17.7 G17.8 G17.9 G18.2 G18.3 G18.4 G18.5 G18.6 G18.7 G18.8 G18.9 G19.2 G19.3 G19.4 G19.5 G19.6 G19.7 G19.8 G19.9 G20.1 G20.2 G20.3 G20.4 G20.5 G20.6 G20.7 G20.8 G20.9 G21.1 G21.2 G21.3 G21.4 G21.5 G21.6 G21.7 G21.8 G21.9 G22 G22.1 G22.2 G22.3 G22.4 G22.5 G22.6 G22.7 G22.8 G22.9 G23 G23.1 G23.2 G23.3 G23.4 G23.5 G23.6 G23.7 G23.8 G23.9 G24 G24.1 G24.2 G24.3 G24.4 G24.5 G24.6 G24.7 G24.8 G24.9 G25 G25.1 G25.2 G25.3 G25.4 G25.5 G25.6 G25.7 G25.8 G25.9 G26 G26.1 G26.2 G26.3 G26.4 G26.5 G26.6 G26.7 G26.8 G26.9 G27 G27.1 G27.2 G27.3 G27.4 G27.5 G27.6 G27.7 G27.8 G27.9 G28.2 G28.3 G28.4 G28.5 G28.6 G28.7 G28.8 G28.9 G29 G29.1 G29.2 G29.3 G29.4 G29.5 G29.6 G29.7 G29.8 G29.9 G30.2 G30.3 G30.4 G30.5 G30.6 G30.7 G30.8 G30.9 G31 G31.1 G31.2 G31.3 G31.4 G31.5 G31.6 G31.7 G31.8 G31.9 G32 G32.1 G32.2 G32.3 G32.4 G32.5 G32.6 G32.7 G32.8 G32.9 G33.2 G33.3 G33.4 G33.5 G33.6 G33.7 G33.8 G33.9 G34 G34.1 G34.2 G34.3 G34.4 G34.5 G34.6 G34.7 G34.8 G34.9 G35 G35.1 G35.2 G35.3 G35.4 G35.5 G35.6 G35.7 G35.8 G35.9 G36 G36.1 G36.2 G36.3 G36.4 G36.5 G36.6 G36.7 G36.8 G36.9 G37 G37.1 G37.2 G37.3 G37.4 G37.5 G37.6 G37.7 G37.8 G37.9 G38 G38.1 G38.6 G38.7 G38.8 G38.9 G39 G39.1 G39.2 G39.3 G39.4 G39.5 G39.6 G39.7 G39.8 G39.9 G40.1 G40.2 G40.3 G40.4 G40.5 G40.6 G40.7 G40.8 G40.9 G41.2 G41.3 G41.4 G41.5 G41.6 G41.7 G41.8 G41.9 G42.2 G42.3 G42.4 G42.5 G42.6 G42.7 G42.8 G42.9 G43.2 G43.3 G43.4 G43.5 G43.6 G43.7 G43.8 G43.9 G44 G44.1 G44.2 G44.3 G44.4 G44.5 G44.6 G44.7 G44.8 G44.9 G45 G45.1 G45.2 G45.3 G45.4 G45.5 G45.6 G45.7 G45.8 G45.9 G46 G46.1 G46.2 G46.3 G46.4 G46.5 G46.6 G46.7 G46.8 G46.9 G47 G47.1 G47.2 G47.3 G47.4 G47.5 G47.6 G47.7 G47.8 G47.9 G48 G48.1 G48.2 G48.3 G48.4 G48.5 G48.6 G48.7 G48.8 G48.9 G49.1 G49.2 G49.3 G49.4 G49.5 G49.6 G49.7 G49.8 G49.9 G50 G50.1 G50.2 G50.3 G50.4 G50.5 G50.6 G50.7 G50.8 G50.9 G51 G51.1 G51.2 G51.3 G51.4 G51.5 G51.6 G51.7 G51.8 G51.9 G52 G52.1 G52.2 G52.3 G52.4 G52.5 G52.6 G52.7 G52.8 G52.9 G53.1 G53.2 G53.3 G53.4 G53.5 G53.6 G53.7 G53.8 G53.9 G54.1 G54.2 G54.3 G54.4 G54.5 G54.6 G54.7 G54.8 G54.9 G55.1 G55.2 G55.3 G55.4 G55.5 G55.6 G55.7 G55.8 G55.9 G56.1 G56.2 G56.3 G56.4 G56.5 G56.6 G56.7 G56.8 G56.9 G57.1 G57.2 G57.3 G57.4 G57.5 G57.6 G57.7 G57.8 G57.9 G58.1 G58.2 G58.3 G58.4 G58.5 G58.6 G58.7 G58.8 G58.9 G59.4 G59.5 G59.6 G59.7 G59.8 G59.9 G60 G60.1 G60.2 G60.3 G60.4 G60.5 G60.6 G60.7 G60.8 G60.9 G61.2 G61.3 G61.4 G61.5 G61.6 G61.7 G61.8 G61.9 G62 G62.1 G62.2 G62.3 G62.4 G62.5 G62.6 G62.7 G62.8 G62.9 G63 G63.1 G63.2 G63.3 G63.4 G63.5 G63.6 G63.7 G63.8 G63.9 G64.1 G64.2 G64.3 G64.4 G64.5 G64.6 G64.7 G64.8 G64.9 G65 G65.1 G65.2 G65.3 G65.4 G65.5 G65.6 G65.7 G65.8 G65.9 G66 G66.1 G66.2 G66.3 G66.4 G66.5 G66.6 G66.7 G66.8 G66.9 G67 G67.1 G67.2 G67.3 G67.4 G67.5 G67.6 G67.7 G67.8 G67.9 G68 G68.1 G68.2 G68.3 G68.4 G68.5 G68.6 G68.7 G68.8 G68.9 G69 G69.1 G69.2 G69.3 G69.4 G69.5 G69.6 G69.7 G69.8 G69.9 G70 G70.1 G70.2 G70.3 G70.4 G70.5 G70.6 G70.7 G70.8 G70.9 G71 G71.1 G71.2 G71.3 G71.4 G71.5 G71.6 G71.7 G71.8 G71.9 G72 G72.1 G72.2 G72.3 G72.4 G72.5 G72.6 G72.7 G72.8 G72.9 G73.1 G73.2 G73.3 G73.4 G73.5 G73.6 G73.7 G73.8 G73.9 G74 G74.1 G74.2 G74.3 G74.4 G74.5 G74.6 G74.7 G74.8 G74.9 G75 G75.1 G75.2 G75.3 G75.4 G75.5 G75.6 G75.7 G75.8 G75.9 G76.1 G76.2 G76.3 G76.4 G76.5 G76.6 G76.7 G76.8 G76.9 G77 G77.1 G77.2 G77.3 G77.4 G77.5 G77.6 G77.7 G77.8 G77.9 G78 G78.1 G78.2 G78.3 G78.4 G78.5 G78.6 G78.7 G78.8 G78.9 G79 G79.1 G79.2 G79.3 G79.4 G79.5 G79.6 G79.7 G79.8 G79.9 G80.1 G80.2 G80.3 G80.4 G80.5 G80.6 G80.7 G80.8 G80.9 G81.1 G81.2 G81.3 G81.4 G81.5 G81.6 G81.7 G81.8 G81.9 G82.1 G82.2 G82.3 G82.4 G82.5 G82.6 G82.7 G82.8 G82.9 G83.1 G83.2 G83.3 G83.4 G83.5 G83.6 G83.7 G83.8 G83.9 G84.1 G84.2 G84.3 G84.4 G84.5 G84.6 G84.7 G84.8 G84.9 G85.1 G85.2 G85.3 G85.4 G85.5 G85.6 G85.7 G85.8 G85.9 G86.1 G86.2 G86.3 G86.4 G86.5 G86.6 G86.7 G86.8 G86.9 G87.1 G87.2 G87.3 G87.4 G87.5 G87.6 G87.7 G87.8 G87.9 G88.1 G88.2 G88.3 G88.4 G88.5 G88.6 G88.7 G88.8 G88.9 G89.1 G89.2 G89.3 G89.4 G89.5 G89.6 G89.7 G89.8 G89.9 G90.2 G90.3 G90.4 G90.5 G90.6 G90.7 G90.8 G90.9 G91.2 G91.3 G91.4 G91.5 G91.6 G91.7 G91.8 G91.9 G92.4 G92.5 G92.6 G92.7 G92.8 G92.9 G93.1 G93.2 G93.3 G93.4 G93.5 G93.6 G93.7 G93.8 G93.9 G94.1 G94.2 G94.3 G94.4 G94.5 G94.6 G94.7 G94.8 G94.9 G95.1 G95.2 G95.3 G95.4 G95.5 G95.6 G95.7 G95.8 G95.9 G96.1 G96.2 G96.3 G96.4 G96.5 G96.6 G96.7 G96.8 G96.9 G97.1 G97.2 G97.3 G97.4 G97.5 G97.6 G97.7 G97.8 G97.9 G98.1 G98.2 G98.3 G98.4 G98.5 G98.6 G98.7 G98.8 G98.9 G99.1 G99.2 G99.3 G99.4 G99.5 G99.6 G99.7 G99.8 G99.9

#### 6.6.14.3 Currently unallocated M-codes:

These codes are currently undefined in the current implementation of LinuxCNC and may be used to define new M-codes:

```
M10
M11 M12 M13 M14 M15 M16 M17 M18 M20
M21 M22 M23 M24 M25 M26 M27 M28 M29 M31 M32 M33 M34 M35 M36 M37 M38 M39 M40
M41 M42 M43 M44 M45 M46 M47 M54 M55 M56 M57 M58 M59 M74 M75 M76 M77 M78 M79 M80
M81 M82 M83 M84 M85 M86 M87 M88 M89 M90
M91 M92 M93 M94 M95 M96 M97 M98 M99
```

All codes between `M199` and `M999`.

#### 6.6.14.4 readahead time and execution time

foo

#### 6.6.14.5 plugin/pickle hack

foo

#### 6.6.14.6 Module, methods, classes, etc reference

foo

### 6.6.15 Introduction: Extending Task Execution

foo

#### 6.6.15.1 Why would you want to change Task Execution?

foo

#### 6.6.15.2 A diagram: task, interp, iocontrol, UI (??)

foo

### 6.6.16 Models of Task execution

foo

#### 6.6.16.1 Traditional iocontrol/iocontrolv2 execution

foo

#### 6.6.16.2 Redefining IO procedures

foo

#### 6.6.16.3 Execution-time Python procedures

foo

### 6.6.17 A short survey of LinuxCNC program execution

To understand remapping of codes it might be helpful to survey the execution of task and interpreter as far as it relates to remapping.

#### 6.6.17.1 Interpreter state

Conceptually, the interpreter's state consist of variables which fall into the following categories:

1. configuration information (typically from INI file)
2. the *World model* - a representation of actual machine state
3. modal state and settings
4. interpreter execution state

(3) refers to state which is *carried over* between executing individual NGC codes - for instance, once the spindle is turned on and the speed is set, it remains at this setting until turned off. The same goes for many codes, like feed, units, motion modes (feed or rapid) and so forth.

(4) holds information about the block currently executed, whether we are in a subroutine, interpreter variables etc.

Most of this state is aggregated in a - fairly unsystematic - `structure _setup` (see interp_internals.hh).

#### 6.6.17.2 Task and Interpreter interaction, Queuing and Read-Ahead

The task part of LinuxCNC is responsible for coordinating actual machine commands - movement, HAL interactions and so forth. It does not by itself handle the RS274NGC language. To do so, task calls upon the interpreter to parse and execute the next command - either from MDI or the current file.

The interpreter execution generates canonical machine operations, which actually move something. These are, however, not immediately executed but put on a queue. The actual execution of these codes happens in the task part of LinuxCNC: canon commands are pulled off that interpreter queue, and executed resulting in actual machine movements.

This means that typically the interpreter is far ahead of the actual execution of commands - the parsing of the program might well be finished before any noticeable movement starts. This behavior is called *read-ahead*.

#### 6.6.17.3 Predicting the machine position

To compute canonical machine operations in advance during read ahead, the interpreter must be able to predict the machine position after each line of Gcode, and that is not always possible.

Let's look at a simple example program which does relative moves (G91), and assume the machine starts at x=0,y=0,z=0. Relative moves imply that the outcome of the next move relies on the position of the previous one:

```
N10 G91
N20 G0 X10 Y-5 Z20
N30 G1 Y20 Z-5
N40 G0 Z30
N50 M2
```

Here the interpreter can clearly predict machine positions for each line:

After N20: x=10 y=-5 z=20; after N30: x=10 y=15 z=15; after N40: x=10 y=15 z=45

and so can parse the whole program and generate canonical operations well in advance.

#### 6.6.17.4 Queue-busters break position prediction

However, complete read ahead is only possible when the interpreter can predict the position impact for **every** line in the program in advance. Let's look at a modified example:

```
N10 G91
N20 G0 X10 Y-5 Z20
N30 G38.3 Z-10
N40 O100 if [#5070 EQ 0]
N50    G1 Y20 Z-5
N60 O100 else
N70    G0 Z30
N80 O100 endif
N90 G1 Z10
N95 M2
```

To pre-compute the move in N90, the interpreter would need to know where the machine is after line N80 - and that depends on whether the probe command succeeded or not, which is not known until it's actually executed.

So, some operations are incompatible with further read-ahead. These are called *queue busters*, and they are:

- reading a HAL pin's value with M66: value of HAL pin not predictable
- loading a new tool with M6: tool geometry not predictable
- executing a probe with G38.x: final position and success/failure not predictable

#### 6.6.17.5 How queue-busters are dealt with

Whenever the interpreter encounters a queue-buster, it needs to stop read ahead and wait until the relevant result is available. The way this works is:

- when such a code is encountered, the interpreter returns a special return code to task (*INTERP_EXECUTE_FINISH*).
- this return code signals to task to stop read ahead for now, execute all queued canonical commands built up so far (including the last one, which is the queue buster), and then *synchronize the interpreter state with the world model*. Technically, this means updating internal variables to reflect HAL pin values, reload tool geometries after an M6, and convey results of a probe.
- The interpreter's *synch()* method is called by task and does just that - read all the world model *actual* values which are relevant for further execution.
- at this point, task goes ahead and calls the interpreter for more read ahead - until either the program ends or another queue-buster is encountered.

#### 6.6.17.6 Word order and execution order

One or several *words* may be present on an NGC *block* if they are compatible (some are mutually exclusive and must be on different lines). The execution model however prescribes a strict ordering of execution of codes, regardless of their appearance on the source line (G-Code Order of Execution).

#### 6.6.17.7 Parsing

Once a line is read (in either MDI mode, or from the current NGC file), it is parsed and flags and parameters are set in a *struct block* (struct _setup, member block1). This struct holds all information about the current source line, but independent of different ordering of codes on the current line: as long as several codes are compatible, any source ordering will result in the same variables set in the struct block. Right after parsing, all codes on a block are checked for compatibility.

#### 6.6.17.8 Execution

After successful parsing the block is executed by execute_block(), and here the different items are handled according to execution order.

If a "queue buster" is found, a corresponding flag is set in the interpreter state (toolchange_flag, input_flag, probe_flag) and the interpreter returns an INTERP_EXECUTE_FINISH return value, signaling *stop readahead for now, and resynch* to the caller (*task*). If no queue busters are found after all items are executed, INTERP_OK is returned, signalling that read-ahead may continue.

When read ahead continues after the synch, task starts executing interpreter read() operations again. During the next read operation, the above mentioned flags are checked and corresponding variables are set (because the a synch() was just executed, the values are now current). This means that the next command already executes in the properly set variable context.

#### 6.6.17.9 Procedure execution

O-word procedures complicate handling of queue busters a bit. A queue buster might be found somewhere in a nested procedure, resulting in a semi-finished procedure call when INTERP_EXECUTE_FINISH is returned. Task makes sure to synchronize the world model, and continue parsing and execution as long as there is still a procedure executing (call_level > 0).

#### 6.6.17.10 How tool change currently works

The actions happening in LinuxCNC are a bit involved, but it's necessary to get the overall idea what currently happens before you set out to adapt those workings to your own needs.

Note that remapping an existing code completely disables all internal processing for that code. That means that beyond your desired behavior - probably described through an NGC Oword or Python procedure, you need to replicate those internal actions of the interpreter which together result in a complete replacement of the existing code. The prolog and epilog code is the place to do this.

**How tool information is communicated** Several processes are *interested* in tool information: task and its interpreter, as well as the user interface. Also, the *halui* process.

Tool information is held in the *emcStatus* structure, which is shared by all parties. One of its fields is the *toolTable* array, which holds the description as loaded from the tool table file (tool number, diameter, frontangle, backangle and orientation for lathe, tool offset information).

The authoritative source and only process actually *setting* tool information in this structure is the *iocontrol* process. All others processes just consult this structure. The interpreter holds actually a local copy of the tool table.

For the curious, the current emcStatus structure can be accessed by Python statements. The interpreter's perception of the tool currently loaded for instance is accessed by:

```
;py,from interpreter import *
;py,print this.tool_table[0]
```

To see fields in the global emcStatus structure, try this:

```
;py,from emctask import *
;py,print emcstat.io.tool.pocketPrepped
;py,print emcstat.io.tool.toolInSpindle
;py,print emcstat.io.tool.toolTable[0]
```

You need to have LinuxCNC started from a terminal window to see the results.

#### 6.6.17.11 How Tx (Prepare Tool) works

**Interpreter action on a Tx command**

All the interpreter does is evaluate the toolnumber parameter, looks up its corresponding pocket, remembers it in the `selected_pocke` variable for later, and queues a canon command (SELECT_POCKET). See *Interp::convert_tool_select* in *src/emc/rs274/interp_execute.c*

**Task action on SELECT_POCKET** When task gets around to handle a SELECT_POCKET, it sends a EMC_TOOL_PREPARE message to the iocontrol process, which handles most tool-related actions in LinuxCNC.

In the current implementation, task actually waits for iocontrol to complete the changer positioning operation, which is not necessary IMO - it defeats the idea that changer preparation and code execution can proceed in parallel.

**Iocontrol action on EMC_TOOL_PREPARE** When iocontrol sees the select pocket command, it does the related HAL pin wiggling - it sets the "tool-prep-number" pin to indicate which tool is next, raises the "tool-prepare" pin, and waits for the "tool-prepared" pin to go high.

When the changer responds by asserting "tool-prepared", it considers the prepare phase to be completed and signals task to continue. (again, this *wait* istn strictly necessary IMO)

**Building the prolog and epilog for Tx** See the Python functions `prepare_prolog` and `prepare_epilog` in `configs/sim/ax`

#### 6.6.17.12 How M6 (Change tool) works

You need to understand this fully before you can adapt it. It is very relevant to writing a prolog and epilog handler for a remapped M6. Remapping an existing codes means you disable the internal steps taken normally, and replicate them as far as needed for your own purposes.

Even if you are not familiar with C, I suggest you look at the *Interp::convert_tool_change* code in *src/emc/rs274/interp_convert.cc*.

**Interpreter action on a M6 command**

When the interpreter sees an M6, it:

1. checks whether a T command has already been executed (test *settings->selected_pocket* to be >= 0) and fail with *Need tool prepared -Txx- for toolchange* message if not.
2. check for cutter compensation being active, and fail with *Cannot change tools with cutter radius compensation on* if so.
3. stop the spindle except if the "TOOL_CHANGE_WITH_SPINDLE_ON" ini option is set.
4. generate a rapid *Z up* move if if the "TOOL_CHANGE_QUILL_UP" ini option is set.
5. if TOOL_CHANGE_AT_G30 was set:
   a. move the A, B and C indexers if applicable
   b. generate rapid move to the G30 position
6. execute a CHANGE_TOOL canon command,with the selected pocket as parameter. CHANGE_TOOL will:
   a. generate a rapid move to TOOL_CHANGE_POSITION if so set in ini
   b. enqueue an EMC_TOOL_LOAD NML message to task.
7. set the numberer parameters 5400-5413 according to the new tool
8. signal to task to stop calling the interpreter for readahead by returning INTERP_EXECUTE_FINISH since M6 is a queue buster.

**What task does when it sees a CHANGE_TOOL command** Again, not much more than passing the buck to iocontrol by sending it an EMC_TOOL_LOAD message, and waiting until iocontrol has done its thing.

IOCONTROL ACTION ON EMC_TOOL_LOAD

1. it asserts the "tool-change" pin
2. it waits for the "tool-changed" pin to become active
3. when that has happened:
   a. deassert "tool-change"
   b. set "tool-prep-number" and "tool-prep-pocket" pins to zero
   c. execute the *load_tool()* function with the pocket as parameter.

The last step actually sets the tooltable entries in the *emcStatus* structure. The actual action taken depends on whether the RANDOM_TOOLCHANGER ini option was set, but at the end of the process *toolTable[0]* reflects the tool currently in the spindle.

When that has happened:

1. iocontrol signals task to go ahead
2. task tells the interpreter to execute a *synch()* operation, to see what has changed
3. the interpreter *synch()* pulls all information from the world model needed, among it the changed tool table.

From there on, the interpreter has complete knowledge of the world model and continues with read ahead.

**Building the prolog and epilog for M6** See the Python functions `change_prolog` and `change_epilog` in `configs/sim/axis`

#### 6.6.17.13 How M61 (Change tool number) works

M61 requires a non-negative `Q`parameter (tool number). If zero, this means *unload tool*, else *set current tool number to Q*.

**Building the replacement for M61** An example Python redefinition for M61 can be found in the `set_tool_number` function in `configs/sim/axis/remap/toolchange/python/toolchange.py`.

### 6.6.18 Optional Interpreter features: ini file configuration

There are some interpreter features in this branch which are experimental, and not backwards compatible, which is why they need to be enabled explicitly. They are specified as follows:

```
[RS274NGC]
FEATURES = <feature mask>
```

Mask bits are:

**Retain G43: 1 (experimental)**
: When set, you can turn on G43 after loading the first tool, and then not worry about it through the program. When you finally unload the last tool, G43 mode is canceled. This is experimental as it changes the operation of legal ngc program, but it could be argued that those programs are buggy or likely to be not what the author intended.

**add n_args parameter: 2**
: A called subroutine can determine the number of actual positional parameters passed by inspecting the `#<n_args>` parameter.

**enable #<_ini[section]name> read only variables: 4**
: if set, the interpreter will fetch read-only values from the ini file through this special variable syntax.

**enable #<_hal[Hal item]> read only variables: 8**
: if set, the interpreter will fetch read-only values from HAL file through this special variable syntax.

**preserve case in O-word names within comments: 16**
: if set, enables reading of mixed-case HAL items in structured comments like *(debug, #<_hal[MixedCaseItem])*. Really a kludge which should go away.

### 6.6.19 Named parameters and inifile variables

To access ini file values from G-code, use the following named parameter syntax:

```
#<_ini[section]name>
```

For example, if the ini file looks like so:

```
[SETUP]
XPOS = 3.145
YPOS = 2.718
```

you may refer to the O-word named parameters `#<_ini[setup]xpos>` and `#<_ini[setup]ypos>` within G-code.

`EXISTS` can be used to test for presence of a given ini file variable:

```
o100 if [EXISTS[#<_ini[setup]xpos>]]
  (debug, [setup]xpos exists: #<_ini[setup]xpos>)
o100 else
  (debug, [setup]xpos does not exist)
o100 endif
```

The value is read from the inifile once, and cached in the interpreter. These parameters are read-only - assigning a value will cause a runtime error. The names are not case sensitive - they are converted to uppercase before consulting the ini file.

Permanent setup information is usually stored in the ini file. While ini variables can be easily accessed from the shell, Python and C code, so far there was no way to refer to ini file variables from G-code. This release enables such access. The feature was motivated by the need to replace ini variables which are currently used in the hard-coded tool change process, like the `[EMCIO]TOOL_CHANGE_POSITION` parameter.

**Caution**
this section doesn't really belong here but since it comes with the same branch, here it rests for now until its clear this will be merged. It should go into the gcode/overview Named Parameters section.

### 6.6.20 Named parameters and HAL items

The variables are read during read-ahead and should not be used for run time evaluation of *current position* or other execution time variables.

To read arbitrary HAL pins, signals and parameters from G-code, use the following named parameter syntax:

```
#<_hal[hal_name]>
```

where `hal_name` may be a pin, parameter or signal name.

Example:

```
(debug, #<_hal[motion-controller.time]>)
```

Access of HAL items is read-only. Currently, only all-lowercase HAL names can be accessed this way.

`EXISTS` can be used to test for the presence of a given HAL item:

```
o100 if [EXISTS[#<_hal[motion-controller.time]>]]
  (debug, [motion-controller.time] exists: #<_hal[motion-controller.time]>)
o100 else
  (debug, [motion-controller.time] does not exist)
o100 endif
```

This feature was motivated by the desire for stronger coupling between user interface components like `GladeVCP` and `PyVCP` to act as parameter source for driving NGC file behavior. The alternative - going through the M6x pins and wiring them - has a limited, non-mmemonic namespace and is unnecessary cumbersome just as a UI/Interpreter communications mechanism.

**Note**
The values are are only updated when the G code is not running.

**Caution**
this section doesn't really belong here but since it comes with the same branch, here it rests for now until its clear this will be merged. It should go into the gcode/overview Named Parameters section.

### 6.6.21 Status

1. the RELOAD_ON_CHANGE feature is fairly broken. Restart after changing a Python file.
2. M61 (remapped or not) is broken in iocontrol and requires iocontrol-v2 to actually work.

### 6.6.22 Changes

- the method to return error messages and fail used to be *self.set_errormsg(text)* followed by *return INTERP_ERROR*. This has been replaced by merely returning a string from a Python handler or oword subroutine. This sets the error message and aborts the program. Previously there was no clean way to abort a Python oword subroutine.

### 6.6.23 Debugging

In the *[EMC]* section of the ini file the *DEBUG* parameter can be changed to get various levels of debug messages when LinuxCNC is started from a terminal.

```
Debug level, 0 means no messages. See src/emc/nml_intf/debugflags.h for others
DEBUG = 0x00000002 # configuration
DEBUG = 0x7FFFDEFF # no interp,oword
DEBUG = 0x00008000 # py only
DEBUG = 0x0000E000 # py + remap + Oword
DEBUG = 0x0000C002 # py + remap + config
DEBUG = 0x0000C100 # py + remap + Interpreter
DEBUG = 0x0000C140 # py + remap + Interpreter + NML msgs
DEBUG = 0x0000C040 # py + remap + NML
DEBUG = 0x0003E100 # py + remap + Interpreter + oword + signals + namedparams
DEBUG = 0x10000000 # EMC_DEBUG_USER1 - trace statements
DEBUG = 0x20000000 # EMC_DEBUG_USER2 - trap into Python debugger
DEBUG = 0x10008000 # USER1, PYTHON
DEBUG = 0x30008000 # USER1,USER2, PYTHON # USER2 will cause involute to try to connect to ↩
    pydev
DEBUG = 0x7FFFFFFF # All debug messages
```

## 6.7 Moveoff Component

The moveoff Hal component is a Hal-only method for implementing offsets. See the manpage (man moveoff) for the IMPORTANT limitations and warnings.

The moveoff component is used to offset joint positions using custom Hal connections. Implementing an offset-while-program-is-paused functionality is supported with appropriate connections for the input pins. Nine joints are supported.

The axis offset pin values (offset-in-M) are continuously applied (respecting limits on value, velocity, and acceleration) to the output pins (offset-current-M, pos-plusoffset-M, fb-minusoffset-M) when both enabling input pins (apply-offsets and move-enable) are TRUE. The two enabling inputs are anded internally. A *warning pin* is set and a message issued if the apply-offsets pin is deasserted while offsets are applied. The warning pin remains TRUE until the offsets are removed or the apply-offsets pin is set.

Typically, the move-enable pin is connected to external controls and the apply-offsets pin is connected to halui.program.is-paused (for offsets only while paused) or set to TRUE (for continuously applied offsets).

Applied offsets are *automatically returned* to zero (respecting limits) when either of the enabling inputs is deactivated. The zero value tolerance is specified by the epsilon input pin value.

Waypoints are recorded when the moveoff componenent is enabled. Waypoints are managed with the waypoint-sample-secs and waypoint-threshold pins. When the backtrack-enable pin is TRUE, the auto-return path follows the recorded waypoints. When the memory available for waypoints is exhausted, offsets are frozen and the waypoint-limit pin is asserted. This restriction applies regardless of the state of the backtrack-enable pin. An enabling pin must be deasserted to allow a return to the original (non-offset position).

Backtracking through waypoints results in *slower* movement rates as the moves are point-to-point respecting velocity and acceleration settings. The velocity and acceleration limit pins can be managed dynamically to control offsets at all times.

When backtrack-enable is FALSE, the auto-return move is **NOT** coordinated, each axis returns to zero at its own rate. If a controlled path is wanted in this condition, each axis should be manually returned to zero before deasserting an enabling pin.

The waypoint-sample-secs, waypoint-threshold, and epsilon pins are evaluated only when the component is idle.

The offsets-applied output pin is provided to indicate the current state to a GUI so that program resumption can be managed. If the offset(s) are non-zero when the apply-offsets pin is deasserted (for example when resuming a program when offsetting during a pause), offsets are returned to zero (respecting limits) and an *Error* message is issued.

**Caution**

If offsets are enabled and applied and the machine is turned off for any reason, any *external* Hal logic that manages the enabling pins and the offset-in-M inputs is responsible for their state when the machine is subsequently turned on again.

This Hal-only means of offsetting is typically not known to LinuxCNC nor available in GUI preview displays. **No protection is provided** for offset moves that exceed soft limits managed by LinuxCNC. Since soft limits are not honored, an offset move may encounter hard limits (or **CRASH** if there are no limit switches). Use of the offset-min-M and offset-max-M inputs to limit travel is recommended. Triggering a hard limit will turn off the machine — see **Caution** above.

The offset-in-M values may be set with inifile settings, controlled by a GUI, or managed by other Hal components and connections. Fixed values may be appropriate in simple cases where the direction and amount of offset is well-defined but a control method is required to deactivate an enabling pin in order to return offsets to zero. GUIs may provide means for users to set, increment, decrement, and accumulate offset values for each axis and may set offset-in-M values to zero before deasserting an enabling pin.

The default values for accel, vel, min, max, epsilon, waypoint-sample-secs, and waypoint-threshold may not be suitable for any particular application. This Hal component is unaware of limits enforced elsewhere by LinuxCNC. Users should test usage in a simulator application and understand all hazards before use on hardware.

Sim configurations that demonstrate the component and a gui (moveoff_gui) are located in:

- configs/sim/axis/moveoff (axis-ui)
- configs/sim/touchy/ngcgui (touchy-ui)

### 6.7.1 Modifying an existing configuration

A system-provided halfile (LIB:hookup_moveoff.tcl) can be used to adapt an existing configuration to use the moveoff component. Additional ini file settings support the use of a simple gui (moveoff_gui) for controlling offsets.

When the system halfile (LIB:hookup_moveoff.tcl) is properly specified in a configuration ini file, it will:

1. Disconnect the original axis.N.motor-pos-cmd and axis.N.motor-pos-fb pin connections
2. Load (loadrt) the moveoff component (using the name mv) with a personality set to accomodate all axes identified in the ini file
3. Add (addf) the moveoff component functions in the required sequence
4. Reconnect the axis.N.motor-pos-cmd and axis.N.motor-pos-fb pins to use the moveoff component
5. Set the moveoff component operating parameters and limits for each axis in accordance with additional ini file settings

Modify an existing configuration as follows:

Make sure there is an ini file entry for [HAL]HALUI and create a new [HAL]HALFILE entry for LIB:hookup_moveoff.tcl. The entry for LIB:hookup_moveoff.tcl should follow all HALFILE= entries for halfiles that connect the pins for axis.N.motor-pos-cmd, axis.N.motor-pos-fb, and any components connected to these pins (pid and encoder components in a servo system for instance).

```
[HAL]
HALUI   = halui
HALFILE = existing_configuration_halfile_1
...
HALFILE = existing_configuration_halfile_n
HALFILE = LIB:hookup_moveoff.tcl
```

Add ini file entries for the per-axis settings for each axis in use (if an entry is not defined, the corresponding entry from the [AXIS_n] section will be used, if no entry is found, then the moveoff component default will be used. Note: Using component defaults or [AXIS_n] section values for per-axis offset settings is NOT recommended.

```
[MOVEOFF_n]
MAX_LIMIT =
MIN_LIMIT =
MAX_VELOCITY =
MAX_ACCELERATION =
```

Add ini file entries for moveoff component settings (omit to use moveoff defaults):

```
[MOVEOFF]
EPSILON =
WAYPOINT_SAMPLE_SECS =
WAYPOINT_THRESHOLD =
```

The moveoff_gui is used to make additional required connections and provide a popup gui to:

1. Provide a control togglebutton to Enable/Disable offsets
2. Provide a control togglebutton to Enable/Disable backtracking
3. Provide control pushbuttons to Increment/Decrement/Zero each axis offset
4. Display each axis offset current value
5. Display current offset status (disabled, active, removing, etc)

The provided control buttons are optional depending upon the state of the moveoff component move-enable pin. Both a display and controls for enabling offsetting are provided if the pin mv.move-enable is NOT connected when the moveoff_gui is started. For this case, the moveoff_gui manages the moveoff component move-enable pin (named mv.move-enable) as well as the offsets (mv.move-offset-in-M) and the backtracking enable (mv.backtrack-enable)

If the mv.move-enable pin IS connected when the moveoff_gui is started, the moveoff_gui will provide a display but NO controls. This mode supports configurations that use a jog wheel or other methods of controlling the offset inputs and the enable pins (mv.offset-in-M, mv.move-enable, mv.backtrack-enable).

The moveoff_gui makes the required connections for the moveoff component pins: mv.power_on and mv.apply-offsets. The mv.power_on pin is connected to the motion.motion-enabled pin (a new signal is automatically created if necessary). The mv.apply-offsets is connected to halui.program.is-paused or set to 1 depending upon the command line option -mode [ onpause | always ]. A new signal is automatically created if necessary.

To use the moveoff_gui, add an entry in the ini file [APPLICATIONS] section as follows:

```
[APPLICATIONS]
# Note: a delay (specified in seconds) may be required if connections
# are made using postgui halfiles ([HAL]POSTGUI_HALFILE=)
DELAY = 0
APP = moveoff_gui option1 option2 ...
```

When the halfile LIB:hookup_moveoff.tcl is used to load and connect the moveoff component, the mv.move-enable pin will not be connected and local controls provided by the moveoff_gui will be used. This is the simplest method to test or demonstrate the moveoff component when modifying an existing ini configuration.

To enable external controls while using the moveoff_gui display for offset values and status, halfiles that follow LIB:hookup_moveoff.tcl must make additional connections. For example, the supplied demonstration configs (configs/sim/axis/moveoff/*.ini) use a simple system halfile (named LIB:moveoff_external.hal) to connect the mv.move-enable,mv.offset-in-M, and mv.bactrack-enable pins to signals:

```
[HAL]
HALUI = halui
...
HALFILE = LIB:hookup_moveoff.tcl
HALFILE = LIB:moveoff_external.hal
```

The connections made by LIB:moveoff_external.hal (for a three axis configuration) are:

```
net external_enable    mv.move-enable

net external_offset_0 mv.offset-in-0
net external_offset_1 mv.offset-in-1
net external_offset_2 mv.offset-in-2

net external_backtrack_en mv.backtrack-enable
```

These signals (external_enable, external_offset_M, external_backtrack_en) may be managed by subsequent HALFILES (including POSTGUI_HALFILEs) to provide customized control of the component while using the moveoff_gui display for current offset values and offset status.

The moveoff_gui is configured with command line options. For details on the operation of moveoff_gui, see the man page:

```
$ man moveoff_gui
```

For a brief listing of command line options for moveoff_gui, use the command line help option:

```
$ moveoff_gui --help

Usage:
moveoff_gui [Options]

Options:
```

```
    [--help | -? | -- -h ]  (This text)

    [-mode [onpause | always]]  (default: onpause)
                                  (onpause: show gui when program paused)
                                  (always:  show gui always)

    [-axes axisnames]       (default: xyz (no spaces))
                              (letters from set of: x y z a b c u v w)
                              (example: -axes z)
                              (example: -axes xz)
                              (example: -axes xyz)
    [-inc incrementvalue]   (default: 0.001 0.01 0.10 1.0 )
                              (specify one per -inc (up to 4) )
                              (example: -inc 0.001 -inc 0.01 -inc 0.1 )
    [-size integer]         (default: 14
                              (Overall gui popup size is based on font size)
    [-loc center|+x+y]      (default: center)
                              (example: -loc +10+200)
    [-autoresume]           (default: notused)
                              (resume program when move-enable deasserted)
    [-delay delay_secs]     (default: 5 (resume delay))

Options for special cases:
    [-noentry]              (default: notused)
                              (don't create entry widgets)
    [-no_resume_inhibit]    (default: notused)
                              (do not use a resume-inhibit-pin)
    [-no_pause_requirement] (default: notused)
                              (no check for halui.program.is-paused)
    [-no_cancel_autoresume] (default: notused)
                              (useful for retraact offsets with simple)
                              (external control                       )
    [-no_display]           (default: notused)
                              (Use when both external controls and displays)
                              (are in use (see Note))                      )

Note: If the moveoff move-enable pin (mv.move-enable) is connected when
      moveoff_gui is started, external controls are required and only
      displays are provided.
```

## 6.8 Stepper Configuration

### 6.8.1 Introduction

The preferred way to set up a standard stepper machine is with the Step Configuration Wizard. See the Stepper Configuration Wizard Chapter.

This chapter describes some of the more common settings for manually setting up a stepper based system. These systems are using stepper motors with drives that accept step & direction signals.

It is one of the simpler setups, because the motors run open-loop (no feedback comes back from the motors), yet the system needs to be configured properly so the motors don't stall or lose steps.

Most of this chapter is based on a sample config released along with LinuxCNC. The config is called stepper_inch, and can be found by running the Configuration Picker.

### 6.8.2 Maximum step rate

With software step generation, the maximum step rate is one step per two BASE_PERIODs for step-and-direction output. The maximum requested step rate is the product of an axis' MAX_VELOCITY and its INPUT_SCALE. If the requested step rate is not attainable, following errors will occur, particularly during fast jogs and G0 moves.

If your stepper driver can accept quadrature input, use this mode. With a quadrature signal, one step is possible for each BASE_PERIOD, doubling the maximum step rate.

The other remedies are to decrease one or more of: the BASE_PERIOD (setting this too low will cause the machine to become unresponsive or even lock up), the INPUT_SCALE (if you can select different step sizes on your stepper driver, change pulley ratios, or leadscrew pitch), or the MAX_VELOCITY and STEPGEN_MAXVEL.

If no valid combination of BASE_PERIOD, INPUT_SCALE, and MAX_VELOCITY is acceptable, then consider using hardware step generation (such as with the LinuxCNC-supported Universal Stepper Controller, Mesa cards, and others.)

### 6.8.3 Pinout

One of the major flaws in LinuxCNC was that you couldn't specify the pinout without recompiling the source code. LinuxCNC is far more flexible, and now (thanks to the Hardware Abstraction Layer) you can easily specify which signal goes where. See the Basic HAL Tutorial for more information on HAL.

As it is described in the HAL Introduction and tutorial, we have signals, pins and parameters inside the HAL.

**Note**
We are presenting one axis to keep it short, all others are similar.

The ones relevant for our pinout are:

```
signals: Xstep, Xdir & Xen
pins: parport.0.pin-XX-out & parport.0.pin-XX-in
```

Depending on what you have chosen in your .ini file you are using either standard_pinout.hal or xylotex_pinout.hal. These are two files that instruct the HAL how to link the various signals & pins. Further on we'll investigate the standard_pinout.hal.

#### 6.8.3.1 Standard Pinout HAL

This file contains several HAL commands, and usually looks like this:

```
# standard pinout config file for 3-axis steppers
# using a parport for I/O
#
# first load the parport driver
loadrt hal_parport cfg="0x0378"
#
# next connect the parport functions to threads
# read inputs first
addf parport.0.read base-thread 1
# write outputs last
addf parport.0.write base-thread -1
#
# finally connect physical pins to the signals
net Xstep => parport.0.pin-03-out
net Xdir  => parport.0.pin-02-out
net Ystep => parport.0.pin-05-out
net Ydir  => parport.0.pin-04-out
net Zstep => parport.0.pin-07-out
net Zdir  => parport.0.pin-06-out
```

```
# create a signal for the estop loopback
net estop-loop iocontrol.0.user-enable-out iocontrol.0.emc-enable-in

# create signals for tool loading loopback
net tool-prep-loop iocontrol.0.tool-prepare iocontrol.0.tool-prepared
net tool-change-loop iocontrol.0.tool-change iocontrol.0.tool-changed

# connect "spindle on" motion controller pin to a physical pin
net spindle-on motion.spindle-on => parport.0.pin-09-out

###
### You might use something like this to enable chopper drives when machine ON
### the Xen signal is defined in core_stepper.hal
###

# net Xen => parport.0.pin-01-out

###
### If you want active low for this pin, invert it like this:
###

# setp parport.0.pin-01-out-invert 1

###
### A sample home switch on the X axis (axis 0).  make a signal,
### link the incoming parport pin to the signal, then link the signal
### to LinuxCNC’s axis 0 home switch input pin
###

# net Xhome parport.0.pin-10-in => axis.0.home-sw-in

###
### Shared home switches all on one parallel port pin?
### that’s ok, hook the same signal to all the axes, but be sure to
### set HOME_IS_SHARED and HOME_SEQUENCE in the ini file.
###

# net homeswitches <= parport.0.pin-10-in
# net homeswitches => axis.0.home-sw-in
# net homeswitches => axis.1.home-sw-in
# net homeswitches => axis.2.home-sw-in

###
### Sample separate limit switches on the X axis (axis 0)
###

# net X-neg-limit parport.0.pin-11-in => axis.0.neg-lim-sw-in
# net X-pos-limit parport.0.pin-12-in => axis.0.pos-lim-sw-in

###
### Just like the shared home switches example, you can wire together
### limit switches.  Beware if you hit one, LinuxCNC will stop but can’t tell
### you which switch/axis has faulted.  Use caution when recovering from this.
###

# net Xlimits parport.0.pin-13-in => axis.0.neg-lim-sw-in axis.0.pos-lim-sw-in
```

The lines starting with # are comments, and their only purpose is to guide the reader through the file.

#### 6.8.3.2 Overview

There are a couple of operations that get executed when the standard_pinout.hal gets executed/interpreted:

- The Parport driver gets loaded (see the Parport Chapter for details)
- The read & write functions of the parport driver get assigned to the base thread [1]
- The step & direction signals for axes X,Y,Z get linked to pins on the parport
- Further I/O signals get connected (estop loopback, toolchanger loopback)
- A spindle-on signal gets defined and linked to a parport pin

#### 6.8.3.3 Changing the standard_pinout.hal

If you want to change the standard_pinout.hal file, all you need is a text editor. Open the file and locate the parts you want to change.

If you want for example to change the pin for the X-axis Step & Directions signals, all you need to do is to change the number in the *parport.0.pin-XX-out* name:

```
net Xstep parport.0.pin-03-out
net Xdir  parport.0.pin-02-out
```

can be changed to:

```
net Xstep parport.0.pin-02-out
net Xdir  parport.0.pin-03-out
```

or basically any other *out* pin you like.

Hint: make sure you don't have more than one signal connected to the same pin.

#### 6.8.3.4 Changing polarity of a signal

If external hardware expects an "active low" signal, set the corresponding *-invert* parameter. For instance, to invert the spindle control signal:

```
setp parport.0.pin-09-invert TRUE
```

#### 6.8.3.5 Adding PWM Spindle Speed Control

If your spindle can be controlled by a PWM signal, use the *pwmgen* component to create the signal:

```
loadrt pwmgen output_type=0
addf pwmgen.update servo-thread
addf pwmgen.make-pulses base-thread
net spindle-speed-cmd motion.spindle-speed-out => pwmgen.0.value
net spindle-on motion.spindle-on => pwmgen.0.enable
net spindle-pwm pwmgen.0.pwm => parport.0.pin-09-out
setp pwmgen.0.scale 1800 # Change to your 'spindles top speed in RPM
```

This assumes that the spindle controller's response to PWM is simple: 0% PWM gives 0 RPM, 10% PWM gives 180 RPM, etc. If there is a minimum PWM required to get the spindle to turn, follow the example in the *nist-lathe* sample configuration to use a *scale* component.

[1] the fastest thread in the LinuxCNC setup, usually the code gets executed every few tens of microseconds

#### 6.8.3.6 Adding an enable signal

Some amplifiers (drives) require an enable signal before they accept and command movement of the motors. For this reason there are already defined signals called *Xen*, *Yen*, *Zen*.

To connect them use the following example:

```
net Xen parport.0.pin-08-out
```

You can either have one single pin that enables all drives; or several, depending on the setup you have. Note, however, that usually when one axis faults, all the other drives will be disabled as well, so having only one enable signal / pin for all drives is a common practice.

#### 6.8.3.7 External ESTOP button

The standard_pinout.hal file assumes no external ESTOP button. For more information on an external E-Stop see the estop_latch man page.

# Chapter 7

# Control Panels

## 7.1 Python Virtual Control Panel

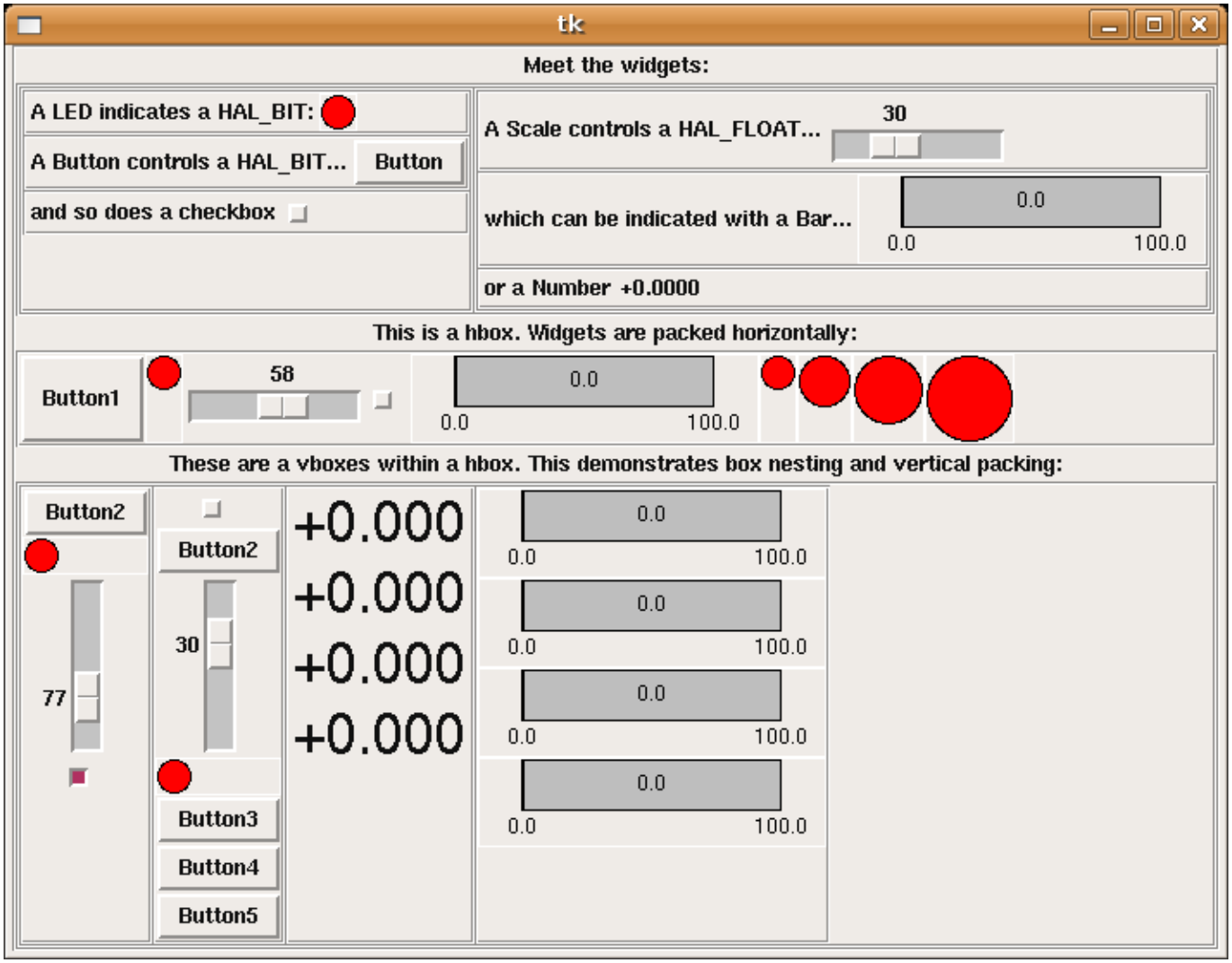
### 7.1.1 Introduction

**Python Virtual Control Panel** The PyVCP (Python Virtual Control Panel) is designed to give the integrator the ability to customize the AXIS interface with buttons and indicators to do special tasks.

Hardware machine control panels can use up a lot of I/O pins and can be expensive. That is where Virtual Control Panels have the advantage as well as it cost nothing to build a PyVCP.

Virtual Control Panels can be used for testing or monitoring things to temporarily replace real I/O devices while debugging ladder logic, or to simulate a physical panel before you build it and wire it to an I/O board.

The following graphic displays many of the PyVCP widgets.




### 7.1.2 Panel Construction

The layout of a PyVCP panel is specified with an XML file that contains widget tags between <pyvcp> and </pyvcp>. For example:

```
<pyvcp>
    <label text="This is a LED indicator"/>
    <led/>
</pyvcp>
```

If you place this text in a file called tiny.xml, and run

```
halrun -I loadusr pyvcp -c mypanel tiny.xml
```

PyVCP will create the panel for you, which includes two widgets, a Label with the text *This is a LED indicator*, and a LED, used for displaying the state of a HAL BIT signal. It will also create a HAL component named *mypanel* (all widgets in this panel are connected to pins that start with *mypanel.*). Since no <halpin> tag was present inside the <led> tag, PyVCP will automatically name the HAL pin for the LED widget mypanel.led.0

For a list of widgets and their tags and options, see the widget reference below.

Once you have created your panel, connecting HAL signals to and from the PyVCP pins is done with the halcmd:

```
net <signal-name> <pin-name> <opt-direction> <opt-pin-name>signal-name
```

If you are new to HAL, the HAL basics chapter in the Integrator Manual is a good place to start.

### 7.1.3 Security

Parts of PyVCP files are evaluated as Python code, and can take any action available to Python programs. Only use PyVCP .xml files from a source that you trust.

### 7.1.4 AXIS

Since AXIS uses the same GUI toolkit (Tkinter) as PyVCP, it is possible to include a PyVCP panel on the right side of the normal AXIS user interface. A typical example is explained below.

Place your PyVCP XML file describing the panel in the same directory where your .ini file is. Say we we want to display the current spindle speed using a Bar widget. Place the following in a file called spindle.xml:

```
<pyvcp>
    <label>
        <text>"Spindle speed:"</text>
    </label>
    <bar>
        <halpin>"spindle-speed"</halpin>
        <max_>5000</max_>
    </bar>
</pyvcp>
```

Here we've made a panel with a Label and a Bar widget, specified that the HAL pin connected to the Bar should be named *spindle-speed*, and set the maximum value of the bar to 5000 (see widget reference below for all options). To make AXIS aware of this file, and call it at start up, we need to specify the following in the [DISPLAY] section of the .ini file:

```
PYVCP = spindle.xml
```

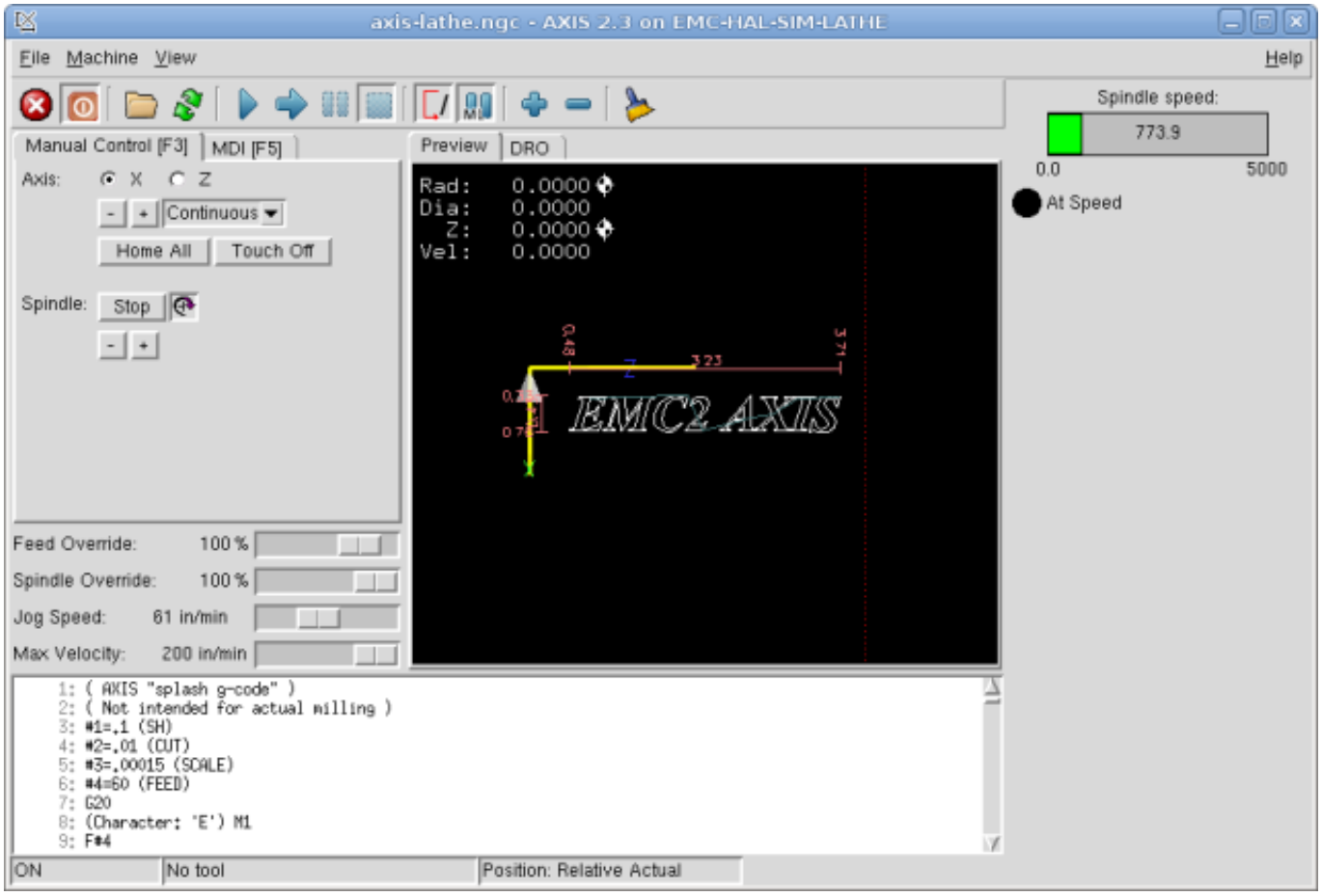
To make our widget actually display the spindle-speed it needs to be hooked up to the appropriate HAL signal. A .hal file that will be run once AXIS and PyVCP have started can be specified in the [HAL] section of the .ini file:

```
POSTGUI_HALFILE = spindle_to_pyvcp.hal
```

This change will run the HAL commands specified in *spindle_to_pyvcp.hal*. In our example the contents could look like this:

```
net spindle-rpm-filtered => pyvcp.spindle-speed
```

assuming that a signal called *spindle-rpm-filtered* already exists. Note that when running together with AXIS, all PyVCP widget HAL pins have names that start with *pyvcp.*.




This is what the newly created PyVCP panel should look like in AXIS. The *sim/lathe* configuration is already configured this way.

### 7.1.5 Stand Alone

This section describes how PyVCP panels can be displayed on their own with or without LinuxCNC's machine controller.

To load a stand alone PyVCP panel with LinuxCNC use these commands:

```
loadusr -Wn mypanel pyvcp -g WxH+X+Y -c mypanel <path/>panel_file.xml
```

You would use this if you wanted a floating panel or a panel with a GUI other than AXIS.

- *-Wn panelname* - makes HAL wait for the component *panelname* to finish loading (*become ready* in HAL speak) before processing more HAL commands. This is important because PyVCP panels export HAL pins, and other HAL components will need them present to connect to them. Note the capital W and lowercase n. If you use the -Wn option you must use the -c option to name the panel.
- *pyvcp < -g> < -c> panel.xml* - builds the panel with the optional geometry and/or panelname from the xml panel file. The panel.xml can be any name that ends in .xml. The .xml file is the file that describes how to build the panel. You must add the path name if the panel is not in the directory that the HAL script is in.
- *-g <WxH><+X+Y>* - specifies the geometry to be used when constructing the panel. The syntax is *Width x Height + X Anchor + Y Anchor*. You can set the size or position or both. The anchor point is the upper left corner of the panel. An example is -g 250x500+800+0 This sets the panel at 250 pixels wide, 500 pixels tall, and anchors it at X800 Y0.
- *-c panelname* - tells PyVCP what to call the component and also the title of the window. The panelname can be any name without spaces.

To load a *stand alone* PyVCP panel without LinuxCNC use this command:

```
loadusr -Wn mypanel pyvcp -g 250x500+800+0 -c mypanel mypanel.xml
```

The minimum command to load a pyvcp panel is:

```
loadusr pyvcp mypanel.xml
```

You would use this if you want a panel without LinuxCNC's machine controller such as for testing or a standalone DRO.

The loadusr command is used when you also load a component that will stop HAL from closing until it's done. If you loaded a panel and then loaded Classic Ladder using *loadusr -w classicladder*, CL would hold HAL open (and the panel) until you closed CL. The *-Wn* above means wait for the component *-Wn "name"* to become ready. (*name* can be any name. Note the capital W and lowercase n.) The -c tells PyVCP to build a panel with the name *panelname* using the info in *panel_file_name.xml*. The name *panel_file_name.xml* can be any name but must end in .xml - it is the file that describes how to build the panel. You must add the path name if the panel is not in the directory that the HAL script is in.

An optional command to use if you want the panel to stop HAL from continuing commands / shutting down. After loading any other components you want the last HAL command to be:

```
waituser panelname
```

This tells HAL to wait for component *panelname* to close before continuing HAL commands. This is usually set as the last command so that HAL shuts down when the panel is closed.

### 7.1.6 Widgets

HAL signals come in two variants, bits and numbers. Bits are off/on signals. Numbers can be *float*, *s32* or *u32*. For more information on HAL data types see the HAL Data section. The PyVCP widget can either display the value of the signal with an indicator widget, or modify the signal value with a control widget. Thus there are four classes of PyVCP widgets that you can connect to a HAL signal. A fifth class of helper widgets allow you to organize and label your panel.

1. Widgets for indicating *bit* signals: led, rectled
2. Widgets for controlling *bit* signals: button, checkbutton, radiobutton
3. Widgets for indicating *number* signals: number, s32, u32, bar, meter
4. Widgets for controlling *number* signals: spinbox, scale, jogwheel
5. Helper widgets: hbox, vbox, table, label, labelframe

#### 7.1.6.1 Syntax

Each widget is described briefly, followed by the markup used, and a screen shot. All tags inside the main widget tag are optional.

#### 7.1.6.2 General Notes

At the present time, both a tag-based and an attribute-based syntax are supported. For instance, the following XML fragments are treated identically:

```
<led halpin="my-led"/>
```

and

```
<led><halpin>"my-led"</halpin></led>
```

When the attribute-based syntax is used, the following rules are used to turn the attributes value into a Python value:

1. If the first character of the attribute is one of the following, it is evaluated as a Python expression: *{([*"
2. If the string is accepted by int(), the value is treated as an integer
3. If the string is accepted by float(), the value is treated as floating-point
4. Otherwise, the string is accepted as a string.

When the tag-based syntax is used, the text within the tag is always evaluated as a Python expression.

The examples below show a mix of formats.

**Comments** To add a comment use the xml syntax for a comment.

```
<!-- My Comment -->
```

**Editing the XML file** Edit the XML file with a text editor. In most cases you can right click on the file and select *open with text editor* or similar.

**Colors**

Colors can be specified using the X11 rgb colors by name *gray75* or hex *#0000ff*. A complete list is located here http://sedition.com/-perl/rgb.html.

Common Colors (colors with numbers indicate shades of that color)

- white
- black
- blue and blue1 - 4
- cyan and cyan1 - 4
- green and green1 - 4
- yellow and yellow1 - 4
- red and red1 - 4
- purple and purple1 - 4
- gray and gray0 - 100

**HAL Pins** HAL pins provide a means to *connect* the widget to something. Once you create a HAL pin for your widget you can *connect* it to another HAL pin with a *net* command in a .hal file. For more information on the *net* command see the HAL Commands section.

#### 7.1.6.3 Label

A label is a way to add text to your panel.

- *<label></label>* - creates a label
- *<text>"text"</text>* - the text to put in your label, a blank label can be used as a spacer to align other objects.
- *<font>("Helvetica",20)</font>* - specificy the font and size of the text
- *<relief>FLAT</relief>* - specificy the border around the label (*FLAT*, *RAISED*, *SUNKEN*) default is *FLAT*
- *<bd>n</bd>* - where *n* is the border width when *RAISED* or *SUNKEN* borders are used.
- *<padx>n</padx>* - where *n* is the amount of extra horizontal extra space.
- *<pady>n</pady>* - where *n* is the amount of extra vertical extra space.

The label has an optional disable pin that is created when you add *<disable_pin>True</disable_pin>*.

```
<label>
    <text>"This is a Label:"</text>
    <font>("Helvetica",20)</font>
</label>
```

The above code produced this example.

#### 7.1.6.4 Multi_Label

An extention of the text label.

Selectable text label, can display up to 6 label legends when associated bit pin is activated.

Attach each legend pin to a signal and get a descriptive label when the signal is TRUE.

If more than one legend pin is TRUE, the highest numbered *TRUE* legend will be displayed.

If a disable pin is created with *<disable_pin>True</disable_pin>* and that pin is set to true the lable changes to a grayed out state.

```
<multilabel>
    <legends>["Label1", "Label2", "Label3", "Label4", "Label5", "Label6"]</legends>
    <font>("Helvetica",20)</font>
    <disable_pin>True</disable_pin>
</multilabel>
```

The above example would create the following pins.

```
pyvcp.multilabel.0.disable
pyvcp.multilabel.0.legend0
pyvcp.multilabel.0.legend1
pyvcp.multilabel.0.legend2
pyvcp.multilabel.0.legend3
pyvcp.multilabel.0.legend4
pyvcp.multilabel.0.legend5
```

If you have more than one multilabel the pins created would increment the number like this *pyvcp.multilabel.1.legend1*.

#### 7.1.6.5 LEDs

A LED is used to indicate the status of a *bit* halpin. The LED color will be on_color when the halpin is true, and off_color otherwise.

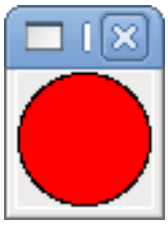
- *<led></led>* - makes a round LED
- *<rectled></rectled>* - makes a rectangle LED
- *<halpin>name</halpin>* - *name* of the pin, default is *led.n*, where n is an integer that is incremented for each LED.
- *<size>n</size>* - *n* is the size of the led in pixels, default is 20
- *<on_color>color</on_color>* - sets the color of the LED when the pin is true. default is *green*. See colors for more info.

- *<off_color>color</off_color>* - sets the color of the LED when the pin is false. default is *red*
- *<height>n<height>* - sets the height of the LED in pixels
- *<width>n<width>* - sets the width of the LED in pixels
- *<disable_pin>false<disable_pin>* - when true adds a disable pin to the led.
- *<disabled_color>color<disabled_color>* - sets the color of the LED when the pin is disabled.

**Round LED**

```
<led>
    <halpin>"my-led"</halpin>
    <size>50</size>
    <on_color>"green"</on_color>
    <off_color>"red"</off_color>
</led>
```

The above code produced this example.



**Rectangle LED** This is a variant of the *led* widget.

```
<vbox>
    <relief>RIDGE</relief>
    <bd>6</bd>
    <rectled>
        <halpin>"my-led"</halpin>
        <height>"50"</height>
        <width>"100"</width>
        <on_color>"green"</on_color>
        <off_color>"red"</off_color>
    </rectled>
</vbox>
```

The above code produced this example. Also showing a vertical box with relief.

#### 7.1.6.6 Buttons

A button is used to control a BIT pin. The pin will be set True when the button is pressed and held down, and will be set False when the button is released. Buttons can use the following optional options.

- *<padx>n</padx>* - where *n* is the amount of extra horizontal extra space.
- *<pady>n</pady>* - where *n* is the amount of extra vertical extra space.
- *<activebackground>"color"</activebackground>* - the cursor over color.
- *<fg> "color"</fg>* - the forground color.
- *<bg>"color"</bg>* - the background color.
- *<disable_pin>True</disable_pin>* - disable pin.

**Text Button** A text button controls a *bit* halpin. The halpin is false until the button is pressed then it is true. The button is a momentary button.

The text button has an optional disable pin that is created when you add <disable_pin>True</disable_pin>.

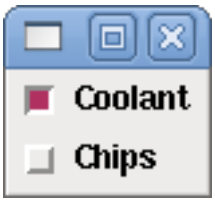
```
<button>
    <halpin>"ok-button"</halpin>
    <text>"OK"</text>
</button>
<button>
    <halpin>"abort-button"</halpin>
    <text>"Abort"</text>
</button
```

The above code produced this example.

**Checkbutton** A checkbutton controls a bit halpin. The halpin will be set True when the button is checked, and false when the button is unchecked. The checkbutton is a toggle type button. The Checkbuttons may be set initially as TRUE or FALSE the initval field A pin called changepin is also created automatically, which can toggle the Checkbutton via HAL, if the value linked is changed, to update the display remotely.

```
<checkbutton>
    <halpin>"coolant-chkbtn"</halpin>
    <text>"Coolant"</text>
    <initval>1</initval>
</checkbutton>
<checkbutton>
    <halpin>"chip-chkbtn"</halpin>
    <text>"Chips    "</text>
    <initval>0</initval>
</checkbutton>
```

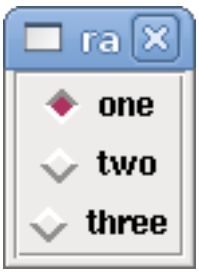
The above code produced this example. The coolant checkbutton is checked. Notice the extra spaces in the Chips text to keep the checkbuttons aligned.



**Radiobutton** A radiobutton will set one of the halpins true. The other pins are set false. The initval field may be set to choose the default selection when the panel displays. Only one radio button may be set to TRUE (1) or only the highest number pin set TRUE will have that value.

```
<radiobutton>
    <choices>["one","two","three"]</choices>
    <halpin>"my-radio"</halpin>
    <initval>0</initval>
</radiobutton>
```

The above code produced this example.




Note that the HAL pins in the example above will me named my-radio.one, my-radio.two, and my-radio.three. In the image above, *one* is the selected value.

#### 7.1.6.7 Number Displays

Number displays can use the following formatting options

- <font>("Font Name",n)</font> where *n* is the font size
- <width>n</width> where *n* is the overall width of the space used
- <justify>pos</justify> where *pos* is LEFT, CENTER, or RIGHT (doesn't work)
- <padx>n</padx> where *n* is the amount of extra horizontal extra space
- <pady>n</pady> where *n* is the amount of extra vertical extra space

**Number** The number widget displays the value of a float signal.

```
<number>
    <halpin>"my-number"</halpin>
    <font>("Helvetica",24)</font>
    <format>"+4.4f"</format>
</number>
```

The above code produced this example.

- *<font>* - is a Tkinter font type and size specification. One font that will show up to at least size 200 is *courier 10 pitch*, so for a really big Number widget you could specify:

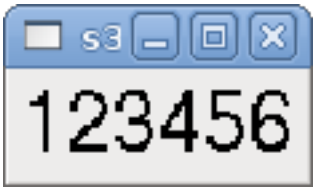
```
<font>("courier 10 pitch",100)</font>
```

- *<format>* - is a *C-style* format specified that determines how the number is displayed.

**s32 Number** The s32 number widget displays the value of a s32 number. The syntax is the same as *number* except the name which is <s32>. Make sure the width is wide enough to cover the largest number you expect to use.

```
<s32>
    <halpin>"my-number"</halpin>
    <font>("Helvetica",24)</font>
    <format>"6d"</format>
    <width>6</width>
</s32>
```

The above code produced this example.




**u32 Number** The u32 number widget displays the value of a u32 number. The syntax is the same as *number* except the name which is <u32>.

**Bar** A bar widget displays the value of a FLOAT signal both graphically using a bar display and numerically. The color of the bar can be set as one color throughout its range (default using fillcolor) or set to change color dependent upon the value of the halpin (range1, range2 range3 must all be set, if you only want 2 ranges, set 2 of them to the same color).

```
<bar>
    <halpin>"my-bar"</halpin>
    <min_>0</min_>
    <max_>150</max_>
    <bgcolor>"grey"</bgcolor>
    <fillcolor>"red"</fillcolor>
    <range1>0,100,"green"</range1>
    <range2>101,135,"orange"</range2>
    <range3>136, 150,"red"</range3>
</bar>
```

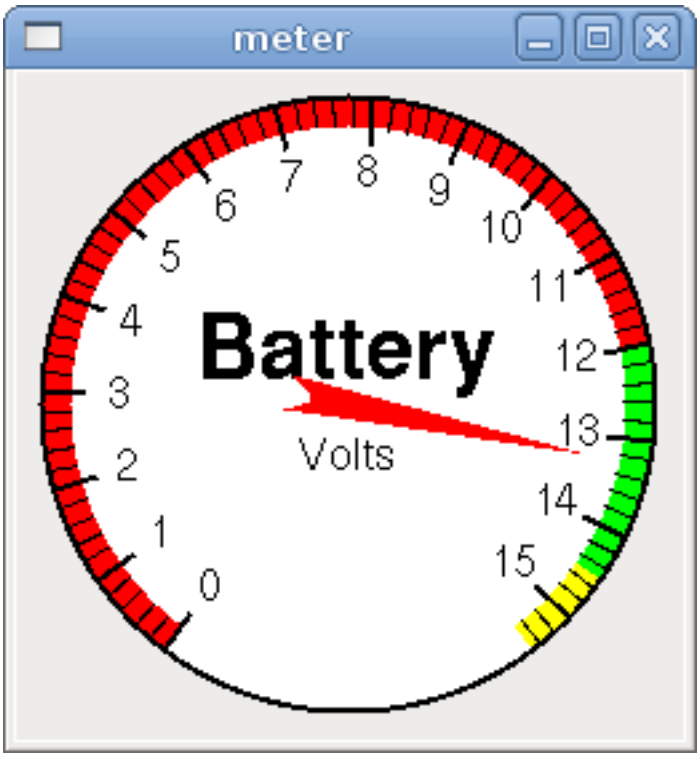
The above code produced this example.

**Meter** Meter displays the value of a FLOAT signal using a traditional dial indicator.

```
<meter>
    <halpin>"mymeter"</halpin>
    <text>"Battery"</text>
    <subtext>"Volts"</subtext>
    <size>250</size>
```

```
    <min_>0</min_>
    <max_>15.5</max_>
    <majorscale>1</majorscale>
    <minorscale>0.2</minorscale>
    <region1>(14.5,15.5,"yellow")</region1>
    <region2>(12,14.5,"green")</region2>
    <region3>(0,12,"red")</region3>
</meter>
```

The above code produced this example.




#### 7.1.6.8 Number Inputs

**Spinbox** Spinbox controls a FLOAT pin. You increase or decrease the value of the pin by either pressing on the arrows, or pointing at the spinbox and rolling your mouse-wheel. If the param_pin field is set TRUE(1), a pin will be created that can be used to set the spinbox to an initial value and to remotely alter its value without HID input.

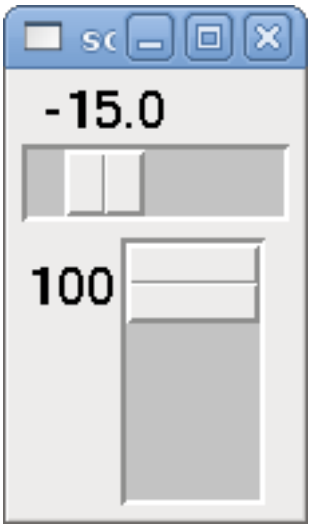
```
<spinbox>
    <halpin>"my-spinbox"</halpin>
    <min_>-12</min_>
    <max_>33</max_>
    <initval>0</initval>
    <resolution>0.1</resolution>
    <format>"2.3f"</format>
    <font>("Arial",30)</font>
    <param_pin>1</param_pin>
</spinbox>
```

The above code produced this example.

**Scale** Scale controls a float or a s32 pin. You increase or decrease the value of the pin be either dragging the slider, or pointing at the scale and rolling your mouse-wheel. The *halpin* will have both *-f* and *-i* added to it to form the float and s32 pins. Width is the width of the slider in vertical and the height of the slider in horizontal orientation. If the param_pin field is set TRUE(1), a pin will be created that can be used to set the spinbox to an initial value and to remotely alter its value without HID input.

```
<scale>
    <font>("Helvetica",16)</font>
    <width>"25"</width>
    <halpin>"my-hscale"</halpin>
    <resolution>0.1</resolution>
    <orient>HORIZONTAL</orient>
    <initval>-15</initval>
    <min_>-33</min_>
    <max_>26</max_>
    <param_pin>1</param_pin>
</scale>
<scale>
    <font>("Helvetica",16)</font>
    <width>"50"</width>
    <halpin>"my-vscale"</halpin>
    <resolution>1</resolution>
    <orient>VERTICAL</orient>
    <min_>100</min_>
    <max_>0</max_>
    <param_pin>1</param_pin>
</scale>
```

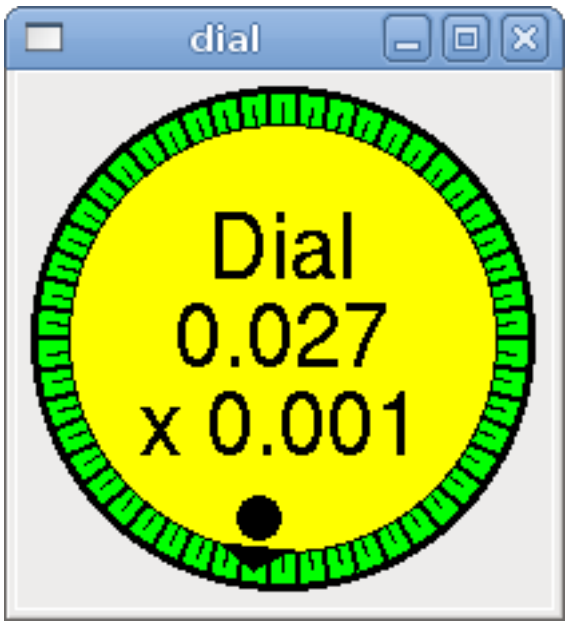
The above code produced this example.




**Dial** The Dial outputs a HAL float and reacts to both mouse wheel and dragging. Double left click to increase the resolution and double right click to reduce the resolution by one digit. The output is capped by the min and max values. The <cpr> is how many tick marks are on the outside of the ring (beware of high numbers). If the param_pin field is set TRUE(1), a pin will be created that can be used to set the spinbox to an initial value and to remotely alter its value without HID input.

```
<dial>
    <size>200</size>
    <cpr>100</cpr>
    <min_>-15</min_>
    <max_>15</max_>
    <text>"Dial"</text>
    <initval>0</initval>
    <resolution>0.001</resolution>
    <halpin>"anaout"</halpin>
    <dialcolor>"yellow"</dialcolor>
    <edgecolor>"green"</edgecolor>
    <dotcolor>"black"</dotcolor>
    <param_pin>1</param_pin>
</dial>
```

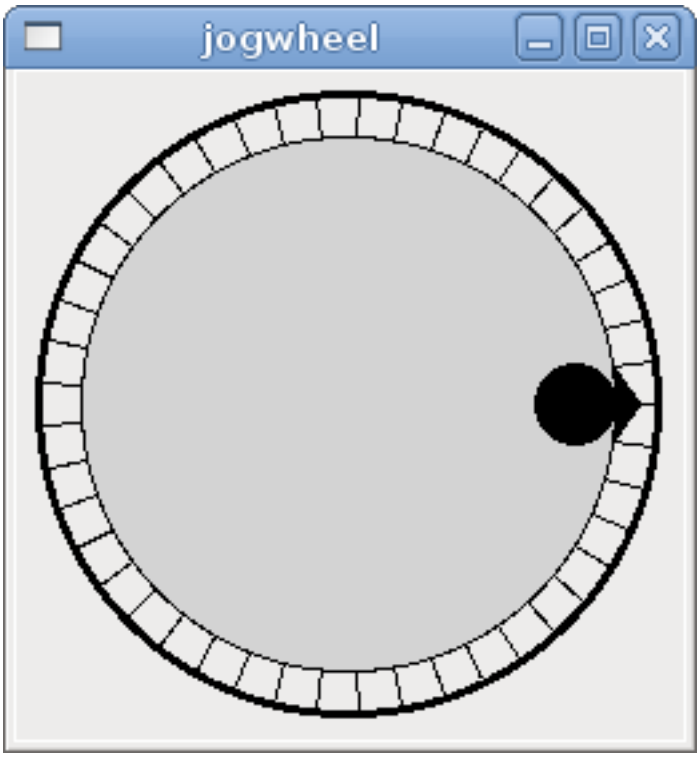
The above code produced this example.




**Jogwheel** Jogwheel mimics a real jogwheel by outputting a FLOAT pin which counts up or down as the wheel is turned, either by dragging in a circular motion, or by rolling the mouse-wheel.

```
<jogwheel>
    <halpin>"my-wheel"</halpin>
    <cpr>45</cpr>
    <size>250</size>
</jogwheel>
```

The above code produced this example.




#### 7.1.6.9 Images

Image displays use only .gif image format. All of the images must be the same size. The images must be in the same directory as your ini file (or in the current directory if running from the command line with halrun/halcmd).

**Image Bit** The *image_bit* toggles between two images by setting the halpin to true or false.

```
<image name='fwd' file='fwd.gif'/>
<image name='rev' file='rev.gif'/>
<vbox>
    <image_bit halpin='selectimage' images='fwd rev'/>
</vbox>
```

This example was produced from the above code. Using the two image files fwd.gif and rev.gif. FWD is displayed when *selectimage* is false and REV is displayed when *selectimage* is true.

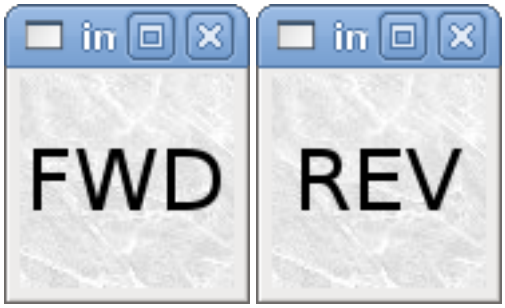



**Image u32** The *image_u32* is the same as *image_bit* except you have essentially an unlimited number of images and you *select* the image by setting the halpin to a integer value with 0 for the first image in the images list and 1 for the second image etc.

```
<image name='stb' file='stb.gif'/>
<image name='fwd' file='fwd.gif'/>
<image name='rev' file='rev.gif'/>
<vbox>
    <image_u32 halpin='selectimage' images='stb fwd rev'/>
</vbox>
```

The above code produced the following example by adding the stb.gif image.

Notice that the default is the min even though it is set higher than max unless there is a negative min.

#### 7.1.6.10 Containers

Containers are widgets that contain other widgets. Containers are used to group other widgets.

**Borders** Container borders are specified with two tags used together. The <relief> tag specifies the type of border and the <bd> specifies the width of the border.

- *<relief>type</relief>* - Where *type* is FLAT, SUNKEN, RAISED, GROOVE, or RIDGE
- *<bd>n</bd>* - Where *n* is the width of the border.

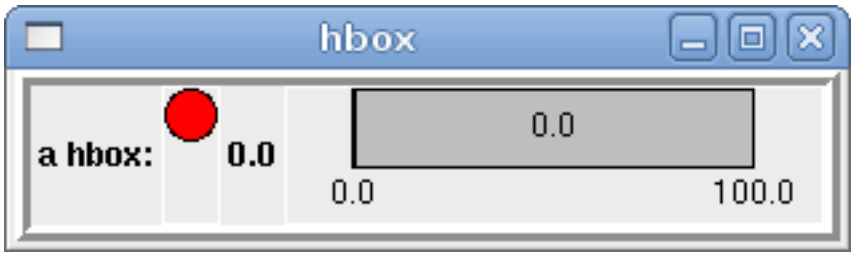
```
<hbox>
    <button>
        <relief>FLAT</relief>
        <text>"FLAT"</text>
        <bd>3</bd>
    </button>
    <button>
        <relief>SUNKEN</relief>
        <text>"SUNKEN"</text>
        <bd>3</bd>
    </button>
    <button>
        <relief>RAISED</relief>
        <text>"RAISED"</text>
        <bd>3</bd>
    </button>
    <button>
        <relief>GROOVE</relief>
        <text>"GROOVE"</text>
        <bd>3</bd>
    </button>
    <button>
        <relief>RIDGE</relief>
        <text>"RIDGE"</text>
        <bd>3</bd>
    </button>
</hbox>
```

The above code produced this example.

**Hbox** Use an Hbox when you want to stack widgets horizontally next to each other.

```
<hbox>
    <relief>RIDGE</relief>
    <bd>6</bd>
    <label><text>"a hbox:"</text></label>
    <led></led>
    <number></number>
    <bar></bar>
</hbox>
```

The above code produced this example.

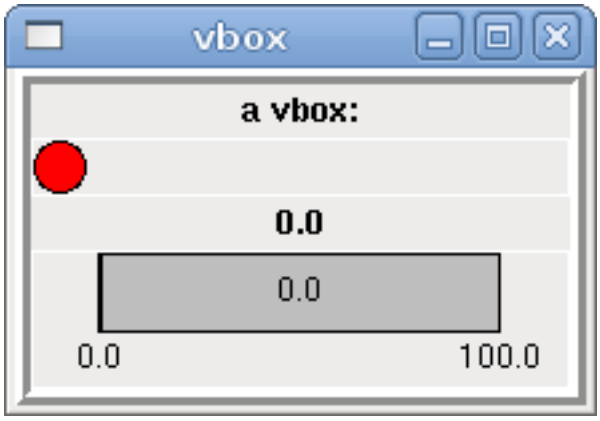



Inside an Hbox, you can use the *<boxfill fill=""/>*, *<boxanchor anchor=""/>* , and *<boxexpand expand=""/>* tags to choose how items in the box behave when the window is re-sized. For details of how fill, anchor, and expand behave, refer to the Tk *pack* manual page, *pack(3tk)*. By default, *fill="y"*, *anchor="center"*, *expand="yes"*.

**Vbox** Use a Vbox when you want to stack widgets vertically on top of each other.

```
<vbox>
    <relief>RIDGE</relief>
    <bd>6</bd>
    <label><text>"a vbox:"</text></label>
    <led></led>
    <number></number>
    <bar></bar>
</vbox>
```

The above code produced this example.




Inside a Hbox, you can use the *<boxfill fill=""/>*, *<boxanchor anchor=""/>* , and *<boxexpand expand=""/>* tags to choose how items in the box behave when the window is re-sized. For details of how fill, anchor, and expand behave, refer to the Tk *pack* manual page, *pack(3tk)*. By default, *fill="x"*, *anchor="center"*, *expand="yes"*.

**Labelframe** A labelframe is a frame with a groove and a label at the upper-left corner.

```
<labelframe text="Group Title">
    <font>("Helvetica",16)</font>
    <hbox>
    <led/>
    <led/>
    </hbox>
</labelframe>
```

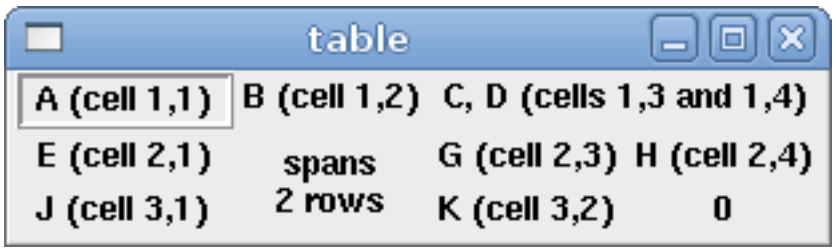
The above code produced this example.

**Table** A table is a container that allows layout in a grid of rows and columns. Each row is started by a *<tablerow/>* tag. A contained widget may span rows or columns through the use of the *<tablespan rows= cols=/>* tag. The sides of the cells to which the contained widgets "stick" may be set through the use of the *<tablesticky sticky=/>* tag. A table expands on its flexible rows and columns.

Example:

```
<table flexible_rows="[2]" flexible_columns="[1,4]">
<tablesticky sticky="new"/>
<tablerow/>
    <label>
        <text>" A (cell 1,1) "</text>
        <relief>RIDGE</relief>
        <bd>3</bd>
    </label>
    <label text="B (cell 1,2)"/>
    <tablespan columns="2"/>
    <label text="C, D (cells 1,3 and 1,4)"/>
<tablerow/>
    <label text="E (cell 2,1)"/>
    <tablesticky sticky="nsew"/>
    <tablespan rows="2"/>
    <label text="'spans\n2 rows'"/>
    <tablesticky sticky="new"/>
    <label text="G (cell 2,3)"/>
    <label text="H (cell 2,4)"/>
<tablerow/>
    <label text="J (cell 3,1)"/>
    <label text="K (cell 3,2)"/>
    <u32 halpin="test"/>
</table>
```

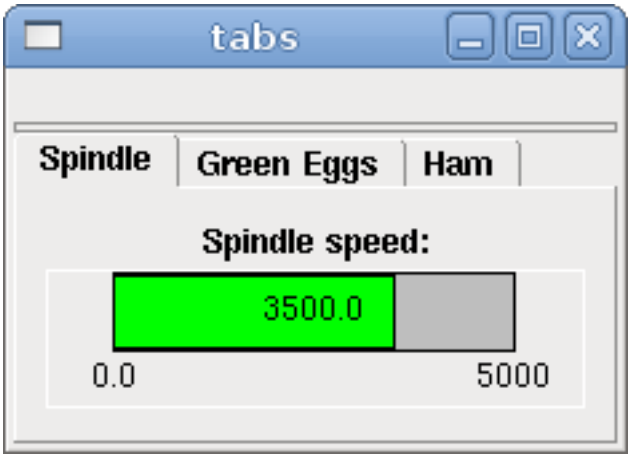
The above code produced this example.

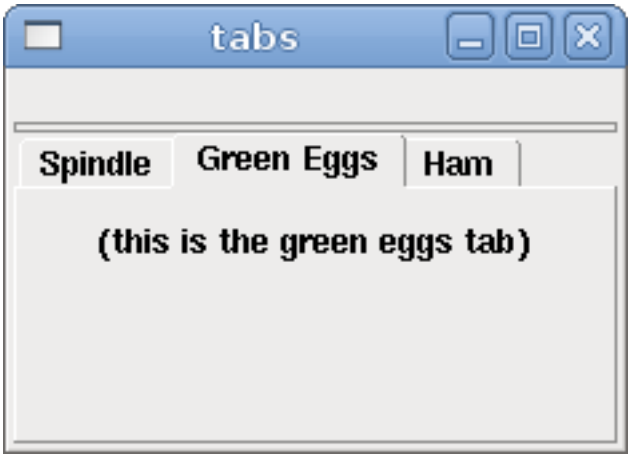


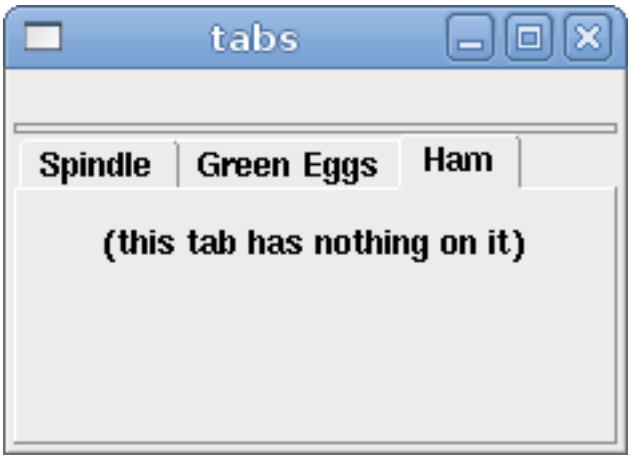
**Tabs** A tabbed interface can save quite a bit of space.

```
<tabs>
    <names> ["spindle","green eggs"]</names>
</tabs>
<tabs>
    <names>["Spindle", "Green Eggs", "Ham"]</names>
    <vbox>
        <label>
            <text>"Spindle speed:"</text>
        </label>
        <bar>
            <halpin>"spindle-speed"</halpin>
            <max_>5000</max_>
        </bar>
    </vbox>
    <vbox>
        <label>
            <text>"(this is the green eggs tab)"</text>
        </label>
    </vbox>
    <vbox>
        <label>
            <text>"(this tab has nothing on it)"</text>
        </label>
    </vbox>
</tabs>
```

The above code produced this example showing each tab selected.






## 7.2 PyVCP Examples

### 7.2.1 AXIS

To create a PyVCP panel to use with the AXIS interface that is attached to the right of AXIS you need to do the following basic things.

- Create an .xml file that contains your panel description and put it in your config directory.
- Add the PyVCP entry to the [DISPLAY] section of the ini file with your .xml file name.
- Add the POSTGUI_HALFILE entry to the [HAL] section of the ini file with the name of your postgui HAL file name.
- Add the links to HAL pins for your panel in the postgui.hal file to *connect* your PyVCP panel to LinuxCNC.

### 7.2.2 Floating

To create floating PyVCP panels that can be used with any interface you need to do the following basic things.

- Create an .xml file that contains your panel description and put it in your config directory.
- Add a loadusr line to your .hal file to load each panel.
- Add the links to HAL pins for your panel in the postgui.hal file to *connect* your PyVCP panel to LinuxCNC.

The following is an example of a loadusr command to load two PyVCP panels and name each one so the connection names in HAL will be known.

```
loadusr -Wn btnpanel pyvcp -c btnpanel panel1.xml
loadusr -Wn sppanel pyvcp -c sppanel panel2.xml
```

The -Wn makes HAL *Wait for name* to be loaded before proceeding. The pyvcp -c makes PyVCP name the panel.

The HAL pins from panel1.xml will be named btnpanel.<pin name>

The HAL pins from panel2.xml will be named sppanel.<pin name>

Make sure the loadusr line is before any nets that make use of the PyVCP pins.

### 7.2.3 Jog Buttons

In this example we will create a PyVCP panel with jog buttons for X, Y, and Z. This configuration will be built upon a Stepconf Wizard generated configuration. First we run the Stepconf Wizard and configure our machine, then on the Advanced Configuration Options page we make a couple of selections to add a blank PyVCP panel as shown in the following figure. For this example we named the configuration *pyvcp_xyz* on the Basic Machine Information page of the Stepconf Wizard.

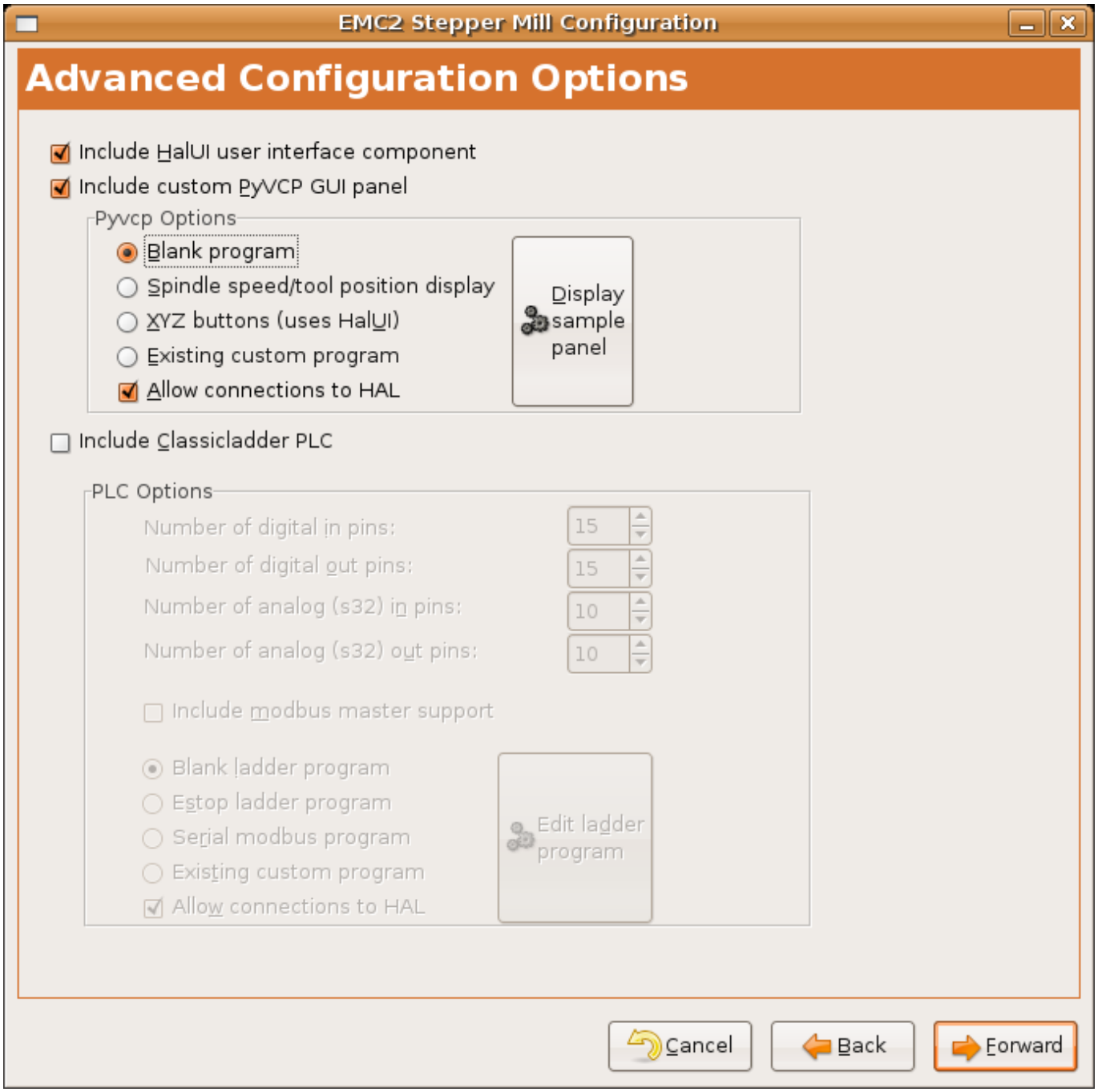



Figure 7.1: XYZ Wizard Configuration

The Stepconf Wizard will create several files and place them in the linuxcnc/configs/pyvcp_xyz directory. If you left the create link checked you will have a link to those files on your desktop.

#### 7.2.3.1 Create the Widgets

Open up the custompanel.xml file by right clicking on it and selecting *open with text editor*. Between the <pyvcp></pyvcp> tags we will add the widgets for our panel.

Look in the PyVCP Widgets Reference section of the manual for more detailed information on each widget.

In your custompanel.xml file we will add the description of the widgets.

```
<pyvcp>
  <labelframe text="Jog Buttons">
    <font>("Helvetica",16)</font>

      <!-- the X jog buttons -->
      <hbox>
      <relief>RAISED</relief>
      <bd>3</bd>
      <button>
        <font>("Helvetica",20)</font>
        <width>3</width>
        <halpin>"x-plus"</halpin>
        <text>"X+"</text>
      </button>
      <button>
        <font>("Helvetica",20)</font>
        <width>3</width>
        <halpin>"x-minus"</halpin>
        <text>"X-"</text>
      </button>
      </hbox>

      <!-- the Y jog buttons -->
      <hbox>
      <relief>RAISED</relief>
      <bd>3</bd>
      <button>
        <font>("Helvetica",20)</font>
        <width>3</width>
        <halpin>"y-plus"</halpin>
        <text>"Y+"</text>
      </button>
      <button>
        <font>("Helvetica",20)</font>
        <width>3</width>
        <halpin>"y-minus"</halpin>
        <text>"Y-"</text>
      </button>
      </hbox>

      <!-- the Z jog buttons -->
      <hbox>
      <relief>RAISED</relief>
      <bd>3</bd>
      <button>
        <font>("Helvetica",20)</font>
        <width>3</width>
        <halpin>"z-plus"</halpin>
        <text>"Z+"</text>
      </button>
      <button>
        <font>("Helvetica",20)</font>
        <width>3</width>
        <halpin>"z-minus"</halpin>
```

```
        <text>"Z-"</text>
      </button>
      </hbox>

      <!-- the jog speed slider -->
      <vbox>
      <relief>RAISED</relief>
      <bd>3</bd>
      <label>
        <text>"Jog Speed"</text>
        <font>("Helvetica",16)</font>
      </label>
      <scale>
        <font>("Helvetica",14)</font>
        <halpin>"jog-speed"</halpin>
        <resolution>1</resolution>
        <orient>HORIZONTAL</orient>
        <min_>0</min_>
        <max_>80</max_>
      </scale>
      </vbox>
  </labelframe>
</pyvcp>
```

After adding the above you now will have a PyVCP panel that looks like the following attached to the right side of AXIS. It looks nice but it does not do anything until you *connect* the buttons to halui. If you get an error when you try and run scroll down to the bottom of the pop up window and usually the error is a spelling or syntax error and it will be there.

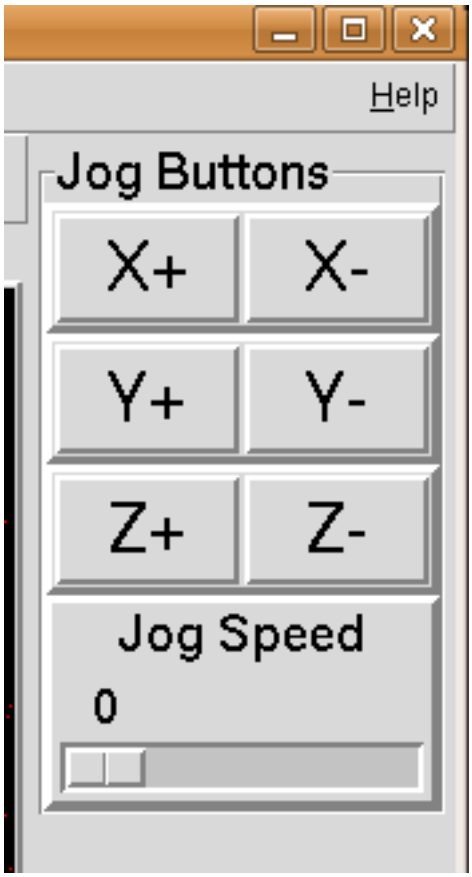



Figure 7.2: Jog Buttons

#### 7.2.3.2 Make Connections

To make the connections needed open up your custom_postgui.hal file and add the following.

```
# connect the X PyVCP buttons
net my-jogxminus halui.jog.0.minus <= pyvcp.x-minus
net my-jogxplus halui.jog.0.plus <= pyvcp.x-plus

# connect the Y PyVCP buttons
net my-jogyminus halui.jog.1.minus <= pyvcp.y-minus
net my-jogyplus halui.jog.1.plus <= pyvcp.y-plus

# connect the Z PyVCP buttons
net my-jogzminus halui.jog.2.minus <= pyvcp.z-minus
net my-jogzplus halui.jog.2.plus <= pyvcp.z-plus

# connect the PyVCP jog speed slider
  net my-jogspeed halui.jog-speed <= pyvcp.jog-speed-f
```

After resetting the E-Stop and putting it into jog mode and moving the jog speed slider in the PyVCP panel to a value greater than zero the PyVCP jog buttons should work. You can not jog when running a g code file or while paused or while the MDI tab is selected.

### 7.2.4 Port Tester

This example shows you how to make a simple parallel port tester using PyVCP and HAL.

First create the ptest.xml file with the following code to create the panel description.

```
<!-- Test panel for the parallel port cfg for out -->
<pyvcp>
  <hbox>
    <relief>RIDGE</relief>
    <bd>2</bd>
    <button>
      <halpin>"btn01"</halpin>
      <text>"Pin 01"</text>
    </button>
    <led>
      <halpin>"led-01"</halpin>
      <size>25</size>
      <on_color>"green"</on_color>
      <off_color>"red"</off_color>
    </led>
  </hbox>
  <hbox>
    <relief>RIDGE</relief>
    <bd>2</bd>
    <button>
      <halpin>"btn02"</halpin>
      <text>"Pin 02"</text>
    </button>
    <led>
      <halpin>"led-02"</halpin>
      <size>25</size>
      <on_color>"green"</on_color>
      <off_color>"red"</off_color>
    </led>
  </hbox>
  <hbox>
    <relief>RIDGE</relief>
```

```
    <bd>2</bd>
    <label>
      <text>"Pin 10"</text>
      <font>("Helvetica",14)</font>
    </label>
    <led>
      <halpin>"led-10"</halpin>
      <size>25</size>
      <on_color>"green"</on_color>
      <off_color>"red"</off_color>
    </led>
  </hbox>
  <hbox>
    <relief>RIDGE</relief>
    <bd>2</bd>
    <label>
      <text>"Pin 11"</text>
      <font>("Helvetica",14)</font>
    </label>
    <led>
      <halpin>"led-11"</halpin>
      <size>25</size>
      <on_color>"green"</on_color>
      <off_color>"red"</off_color>
    </led>
  </hbox>
</pyvcp>
```

This will create the following floating panel which contains a couple of in pins and a couple of out pins.

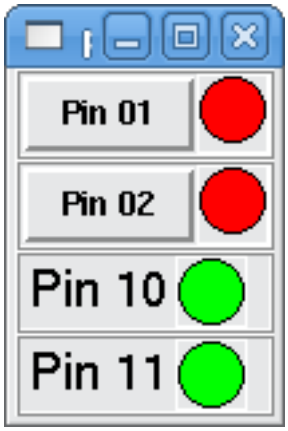



Figure 7.3: Port Tester Panel

To run the HAL commands that we need to get everything up and running we put the following in our ptest.hal file.

```
loadrt hal_parport cfg="0x378 out"
loadusr -Wn ptest pyvcp -c ptest ptest.xml
loadrt threads name1=porttest period1=1000000
addf parport.0.read porttest
addf parport.0.write porttest
net pin01 ptest.btn01 parport.0.pin-01-out ptest.led-01
net pin02 ptest.btn02 parport.0.pin-02-out ptest.led-02
net pin10 parport.0.pin-10-in ptest.led-10
net pin11 parport.0.pin-11-in ptest.led-11
start
```

To run the HAL file we use the following command from a terminal window.

```
~$ halrun -I -f ptest.hal
```

The following figure shows what a complete panel might look like.

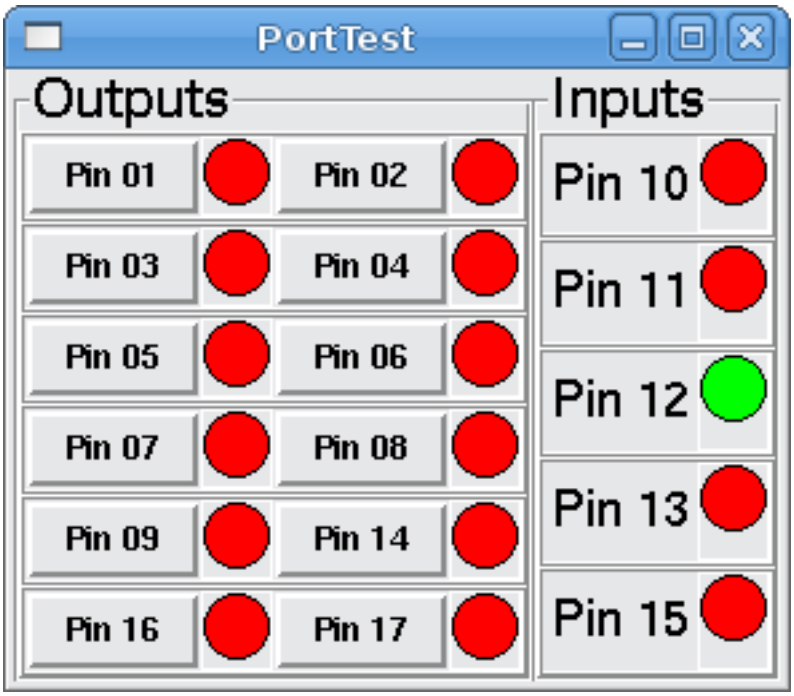



Figure 7.4: Port Tester Complete

To add the rest of the parallel port pins just modify the .xml and .hal files.

To show the pins after running the HAL script use the following command at the halcmd prompt:

```
halcmd: show pin
Component Pins:
Owner Type  Dir Value  Name
    2 bit   IN  FALSE  parport.0.pin-01-out <== pin01
    2 bit   IN  FALSE  parport.0.pin-02-out <== pin02
    2 bit   IN  FALSE  parport.0.pin-03-out
    2 bit   IN  FALSE  parport.0.pin-04-out
    2 bit   IN  FALSE  parport.0.pin-05-out
    2 bit   IN  FALSE  parport.0.pin-06-out
    2 bit   IN  FALSE  parport.0.pin-07-out
    2 bit   IN  FALSE  parport.0.pin-08-out
    2 bit   IN  FALSE  parport.0.pin-09-out
    2 bit   OUT TRUE   parport.0.pin-10-in ==> pin10
    2 bit   OUT FALSE  parport.0.pin-10-in-not
    2 bit   OUT TRUE   parport.0.pin-11-in ==> pin11
    2 bit   OUT FALSE  parport.0.pin-11-in-not
    2 bit   OUT TRUE   parport.0.pin-12-in
    2 bit   OUT FALSE  parport.0.pin-12-in-not
    2 bit   OUT TRUE   parport.0.pin-13-in
    2 bit   OUT FALSE  parport.0.pin-13-in-not
    2 bit   IN  FALSE  parport.0.pin-14-out
    2 bit   OUT TRUE   parport.0.pin-15-in
    2 bit   OUT FALSE  parport.0.pin-15-in-not
    2 bit   IN  FALSE  parport.0.pin-16-out
```

```
    2 bit    IN  FALSE  parport.0.pin-17-out
    4 bit    OUT FALSE  ptest.btn01 ==> pin01
    4 bit    OUT FALSE  ptest.btn02 ==> pin02
    4 bit    IN  FALSE  ptest.led-01 <== pin01
    4 bit    IN  FALSE  ptest.led-02 <== pin02
    4 bit    IN  TRUE   ptest.led-10 <== pin10
    4 bit    IN  TRUE   ptest.led-11 <== pin11
```

This will show you what pins are IN and what pins are OUT as well as any connections.

### 7.2.5 GS2 RPM Meter

The following example uses the Automation Direct GS2 VDF driver and displays the RPM and other info in a PyVCP panel. This example is based on the GS2 example in the Hardware Examples section this manual.

#### 7.2.5.1 The Panel

To create the panel we add the following to the .xml file.

```
<pyvcp>

  <!-- the RPM meter -->
  <hbox>
    <relief>RAISED</relief>
    <bd>3</bd>
    <meter>
      <halpin>"spindle_rpm"</halpin>
      <text>"Spindle"</text>
      <subtext>"RPM"</subtext>
      <size>200</size>
      <min_>0</min_>
      <max_>3000</max_>
      <majorscale>500</majorscale>
      <minorscale>100</minorscale>
      <region1>0,10,"yellow"</region1>
    </meter>
  </hbox>

  <!-- the On Led -->
  <hbox>
  <relief>RAISED</relief>
  <bd>3</bd>
  <vbox>
  <relief>RAISED</relief>
  <bd>2</bd>
  <label>
  <text>"On"</text>
  <font>("Helvetica",18)</font>
  </label>
  <width>5</width>
   <hbox>
  <label width="2"/> <!-- used to center the led -->
  <rectled>
  <halpin>"on-led"</halpin>
  <height>"30"</height>
  <width>"30"</width>
  <on_color>"green"</on_color>
  <off_color>"red"</off_color>
  </rectled>
```

```
  </hbox>
  </vbox>

  <!-- the FWD Led -->
  <vbox>
    <relief>RAISED</relief>
    <bd>2</bd>
    <label>
      <text>"FWD"</text>
      <font>("Helvetica",18)</font>
      <width>5</width>
    </label>
    <label width="2"/>
    <rectled>
      <halpin>"fwd-led"</halpin>
      <height>"30"</height>
      <width>"30"</width>
      <on_color>"green"</on_color>
      <off_color>"red"</off_color>
    </rectled>
  </vbox>

  <!-- the REV Led -->
  <vbox>
  <relief>RAISED</relief>
  <bd>2</bd>
    <label>
      <text>"REV"</text>
      <font>("Helvetica",18)</font>
       <width>5</width>
    </label>
    <label width="2"/>
    <rectled>
      <halpin>"rev-led"</halpin>
      <height>"30"</height>
      <width>"30"</width>
      <on_color>"red"</on_color>
      <off_color>"green"</off_color>
    </rectled>
  </vbox>
  </hbox>
</pyvcp>
```

The above gives us a PyVCP panel that looks like the following.

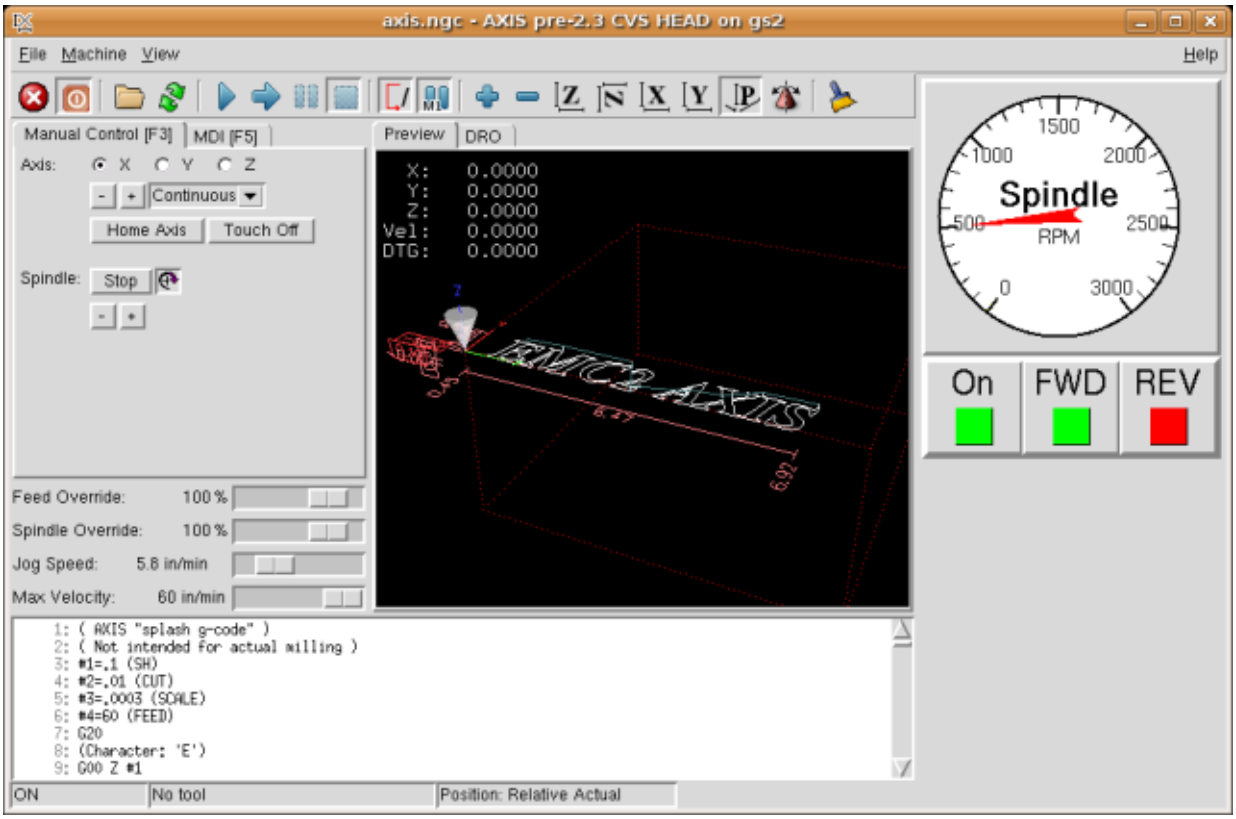



Figure 7.5: GS2 Panel

#### 7.2.5.2 The Connections

To make it work we add the following code to the custom_postgui.hal file.

```
# display the rpm based on freq * rpm per hz
loadrt mult2
addf mult2.0 servo-thread
setp mult2.0.in1 28.75
net cypher_speed mult2.0.in0 <= spindle-vfd.frequency-out
net speed_out pyvcp.spindle_rpm <= mult2.0.out

# run led
net gs2-run => pyvcp.on-led

# fwd led
net gs2-fwd => pyvcp.fwd-led

# rev led
net running-rev spindle-vfd.spindle-rev => pyvcp.rev-led
```

Some of the lines might need some explanations. The fwd led line uses the signal created in the custom.hal file whereas the rev led needs to use the spindle-rev bit. You can't link the spindle-fwd bit twice so you use the signal that it was linked to.

## 7.3 Glade Virtual Control Panel

### 7.3.1 What is GladeVCP?

GladeVCP is an LinuxCNC component which adds the ability to add a new user interface panel to LinuxCNC user interfaces like:

```
-Axis
-Touchy
-Gscreen
-Gmoccapy
```

Unlike PyVCP, GladeVCP is not limited to displaying and setting HAL pins, as arbitrary actions can be executed in Python code - in fact, a complete LinuxCNC user interface could be built with GladeVCP and Python.

GladeVCP uses the Glade WYSIWYG user interface editor, which makes it easy to create visually pleasing panels. It relies on the PyGTK bindings to the rich GTK+ widget set, and in fact all of these may be used in a GladeVCP application - not just the specialized widgets for interacting with HAL and LinuxCNC, which are documented here.

#### 7.3.1.1 PyVCP versus GladeVCP at a glance

Both support the creation of panels with *HAL widgets* - user interface elements like LED's, buttons, sliders etc whose values are linked to a HAL pin, which in turn interfaces to the rest of LinuxCNC.

**PyVCP:**

- widget set: uses TkInter widgets
- user interface creation: "edit XML file / run result / evaluate looks" cycle
- no support for embedding user-defined event handling
- no LinuxCNC interaction beyond HAL pin I/O supported

**GladeVCP:**

- widget set: relies on the GTK+ widget set.
- user interface creation: uses the Glade WYSIWYG user interface editor
- any HAL pin change may be directed to call back into a user-defined Python event handler
- any GTK signal (key/button press, window, I/O, timer, network events) may be associated with user-defined handlers in Python
- direct LinuxCNC interaction: arbitrary command execution, like initiating MDI commands to call a G-code subroutine, plus support for status change operations through Action Widgets
- several independent GladeVCP panels may be run in different tabs
- separation of user interface appearance and functionality: change appearance without touching any code

### 7.3.2 A Quick Tour with the Example Panel

GladeVCP panel windows may be run in three different setups:

- always visible integrated into Axis at the right side, exactly like PyVCP panels
- as a tab in Axis,Touchy, Gscreen, or Gmoccapy; in Axis this would create a third tab besides the Preview and DRO tabs which must be raised explicitly
- as a standalone toplevel window, which can be iconifyed/deiconified independent of the main window.

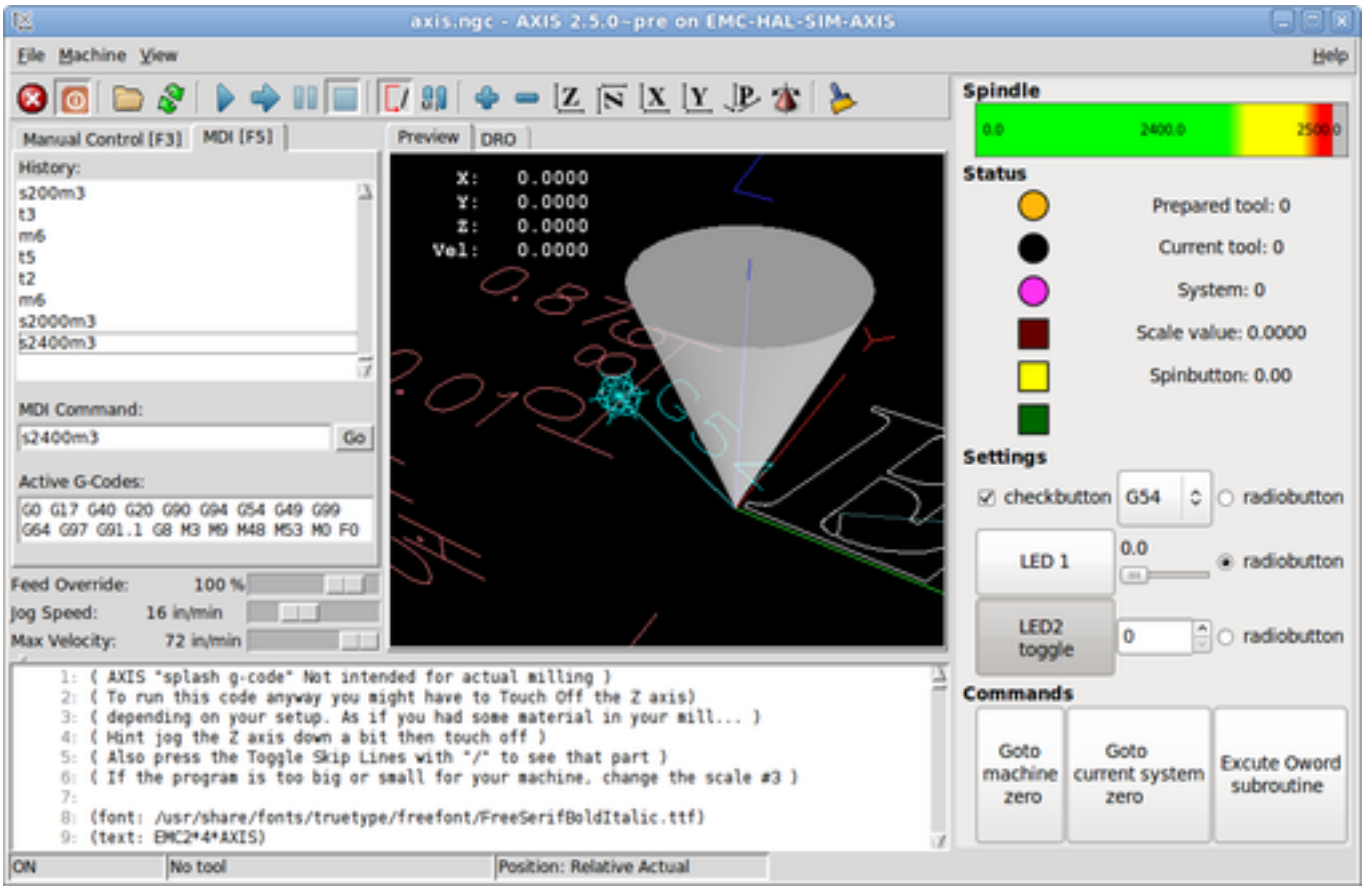
**Installed LinuxCNC** If you're using an installed version of LinuxCNC the examples shown below are in the configuration picker in the *Sample Configurations* > *apps* > *gladevcp* branch.

**Git Checkout** The following instructions only apply if you're using a git checkout. Open a terminal and change to the directory created by git then issue the commands as shown.

> **Note**
> For the following commands to work on your git checkout you must first run *make* then run *sudo make setuid* then run *. ./scripts/rip-environment*. More information about a git checkout is on the linuxcnc wiki page.

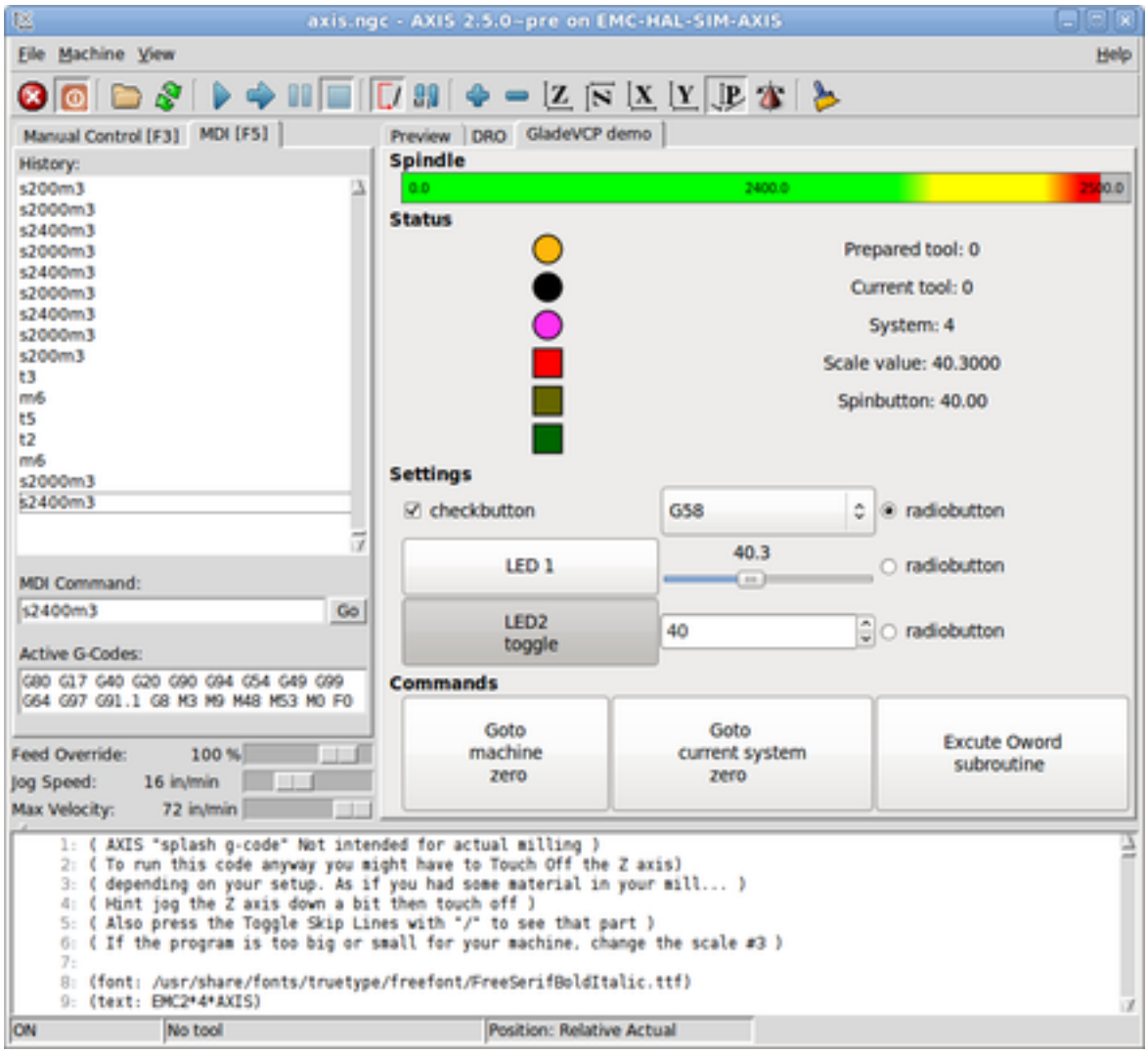
Run the sample GladeVCP panel integrated into Axis like PyVCP as follows:

```
$ cd configs/sim/axis/gladevcp
$ linuxcnc gladevcp_panel.ini
```



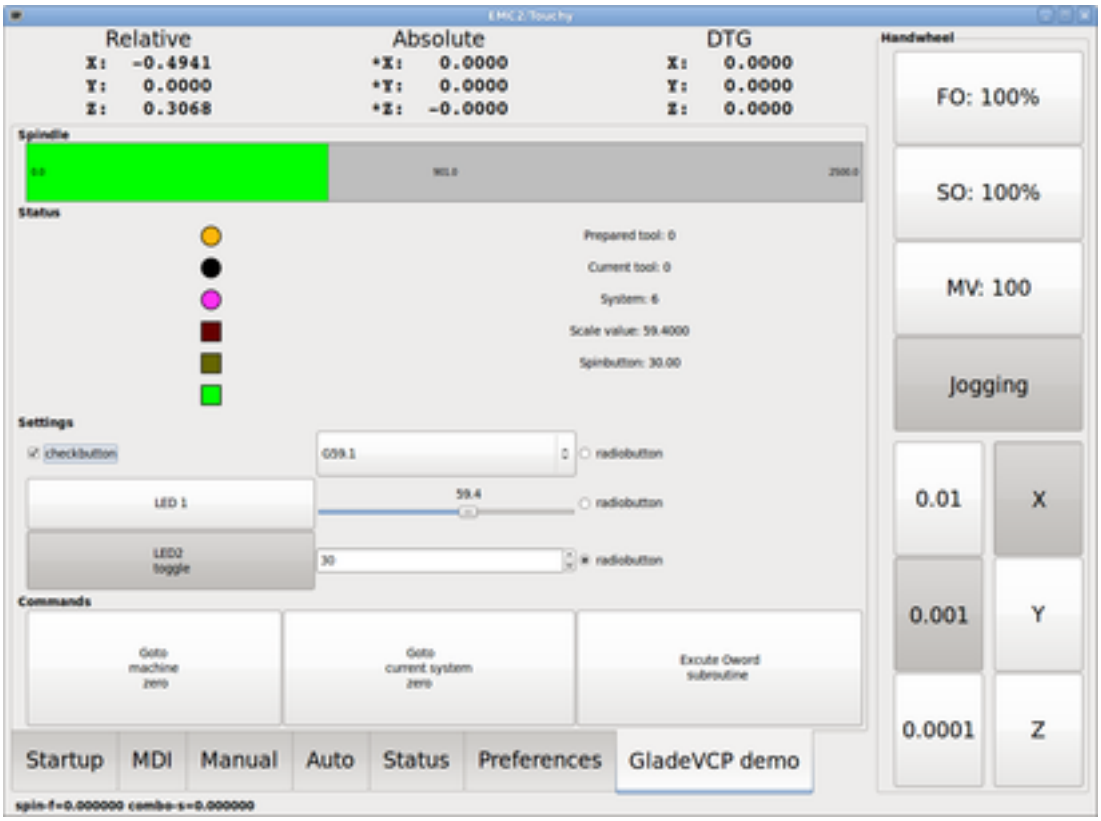

Run the same panel, but as a tab inside Axis:

```
$ cd configs/sim/axis/gladevcp
$ linuxcnc gladevcp_tab.ini
```




To run this panel inside *Touchy*:

```
$ cd configs/sim/touchy/gladevcp
$ linuxcnc gladevcp_touchy.ini
```




Functionally these setups are identical - they only differ in screen real estate requirements and visibility. Since it is possible to run several GladeVCP components in parallel (with different HAL component names), mixed setups are possible as well - for instance a panel on the right hand side, and one or more tabs for less-frequently used parts of the interface.

#### 7.3.2.1 Exploring the example panel

While running configs/sim/axis/gladevcp_panel.ini or configs/sim/axis/gladevcp_tab.ini, explore *Show HAL Configuration* - you will find the *gladevcp* HAL component and may observe their pin values while interacting with the widgets in the panel. The HAL setup can be found in *configs/axis/gladevcp/manual-example.hal*.

The example panel has two frames at the bottom. The panel is configured so that resetting ESTOP activates the Settings frame and turning the machine on enables the Commands frame at the bottom. The HAL widgets in the Settings frame are linked to LEDs and labels in the *Status* frame, and to the current and prepared tool number - play with them to see the effect. Executing the *T<toolnumber>* and *M6* commands in the MDI window will change the current and prepared tool number fields.

The buttons in the *Commands* frame are *MDI Action widgets* - pressing them will execute an MDI command in the interpreter. The third button *Execute Oword subroutine* is an advanced example - it takes several HAL pin values from the *Settings* frame, and passes them as parameters to the Oword subroutine. The actual parameters received by the routine are displayed by *(DEBUG, )* commands - see *../../nc_files/oword.ngc* for the subroutine body.

To see how the panel is integrated into Axis, see the *[DISPLAY]GLADEVCP* statement in configs/sim/axis/gladevcp/gladevcp_panel.ini, the *[DISPLAY]EMBED** statement in configs/sim/axis/gladevcp/gladevcp_tab.ini and *[HAL]POSTGUI_HALFILE* statements in both configs/sim/axis/gladevcp/gladevcp_tab.ini and configs/sim/axis/gladevcp/gladevcp_panel.ini

#### 7.3.2.2 Exploring the User Interface description

The user interface is created with the glade UI editor - to explore it, you need to have glade installed. To edit the user interface, run the command

```
$ glade configs/axis/gladevcp/manual-example.ui
```

(The required glade program may be named glade-gtk2 on more recent systems.)

The center window shows the appearance of the UI. All user interface objects and support objects are found in the right top window, where you can select a specific widget (or by clicking on it in the center window). The properties of the selected widget are displayed, and can be changed, in the right bottom window.

To see how MDI commands are passed from the MDI Action widgets, explore the widgets listed under *Actions* in the top right window, and in the right bottom window, under the *General* tab, the *MDI command* property.

#### 7.3.2.3 Exploring the Python callback

See how a Python callback is integrated into the example:

- in glade, see the `hits` label widget (a plain GTK+ widget)
- in the `button1` widget, look at the *Signals* tab, and find the signal *pressed* associated with the handler *on_button_press*
- in hitcounter.py, see the method *on_button_press* and see how it sets the label property in the *hits* object

The is just touching upon the concept - the callback mechanism will be handled in more detail in the GladeVCP Programming section.

### 7.3.3 Creating and Integrating a Glade user interface

#### 7.3.3.1 Prerequisite: Glade installation

To view or modify Glade UI files, you need glade 3.8.0 installed - it is not needed just to run a GladeVCP panel. If the glade command is missing, install it with the command:

```
$ sudo apt-get install glade-gtk2
```

Verify the version number to be 3.8.0 or less

```
 $ glade-gtk2 --version
glade3 3.8.0
```

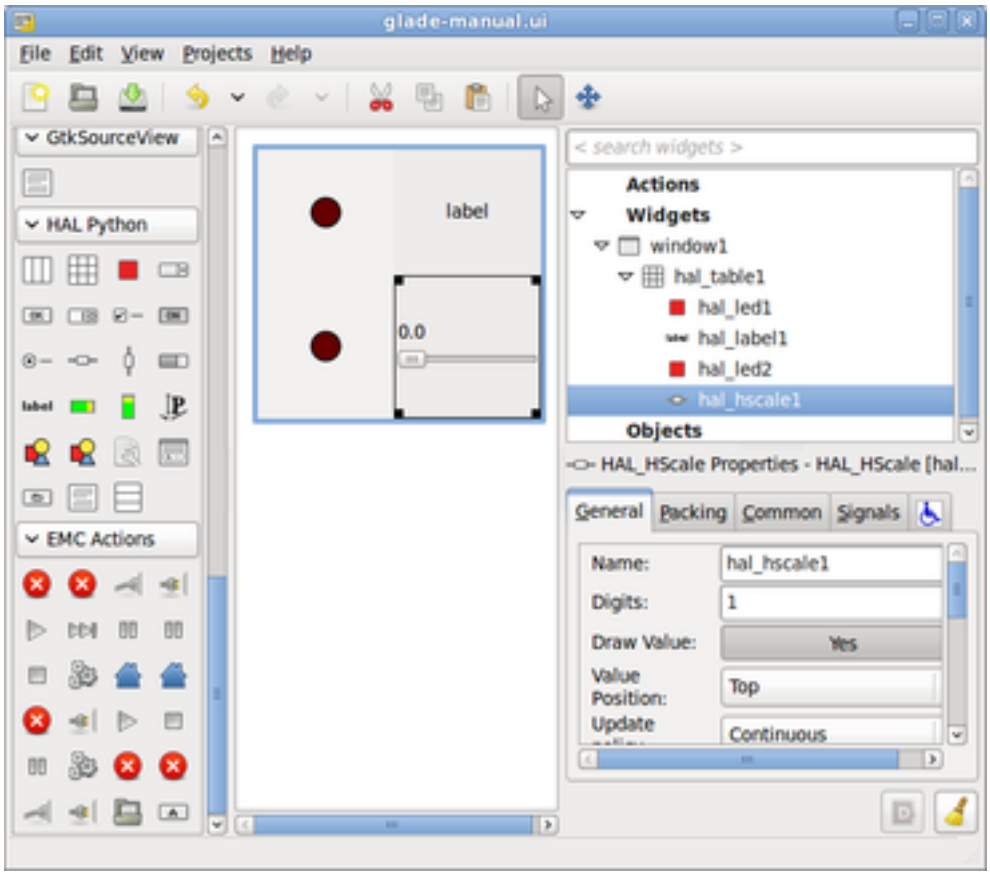
#### 7.3.3.2 Running Glade to create a new user interface

This section just outlines the initial LinuxCNC-specific steps. For more information and a tutorial on glade, see http://glade.gnome.org. Some glade tips & tricks may also be found on youtube.

Either modify an existing UI component by running `glade <file>.ui` or start a new one by just running the `glade` command from the shell.

- If LinuxCNC was not installed from a package, the LinuxCNC shell environment needs to be set up with `. <linuxcncdir>/scr` otherwise glade won't find the LinuxCNC-specific widgets.
- When asked for unsaved Preferences, just accept the defaults and hit *Close*.
- From *Toplevel* (left pane), pick *Window* (first icon) as top level window, which by default will be named *window1*. Do not change this name - GladeVCP relies on it.
- In the left tab, scroll down and expand *HAL Python* and *VCP Actions*.
- add a container like a HAL_Box or a HAL_Table from *HAL Python* to the frame
- pick and place some elements like LED, button, etc. within a container

This will look like so:




Glade tends to write a lot of messages to the shell window, which mostly can be ignored. Select *File→Save as*, give it a name like *myui.ui* and make sure it's saved as *GtkBuilder* file (radio button left bottom corner in Save dialog). GladeVCP will also process the older *libglade* format correctly but there is no point in using it. The convention for GtkBuilder file extension is *.ui*.

#### 7.3.3.3 Testing a panel

You're now ready to give it a try (while LinuxCNC, e.g. Axis is running) it with:

```
gladevcp myui.ui
```

GladeVCP creates a HAL component named like the basename of the UI file - *myui* in this case - unless overridden by the `-c <component name>` option. If running Axis, just try *Show HAL configuration* and inspect its pins.

You might wonder why widgets contained a *HAL_Hbox* or *HAL_Table* appear greyed out (inactive). HAL containers have an associated HAL pin which is off by default, which causes all contained widgets to render inactive. A common use case would be to associate these container HAL pins with `halui.machine.is-on` or one of the `halui.mode.` signals, to assure some widgets appear active only in a certain state.

To just activate a container, execute the HAL command `setp gladevcp.<container-name> 1`.

#### 7.3.3.4 Preparing the HAL command file

The suggested way of linking HAL pins in a GladeVCP panel is to collect them in a separate file with extension `.hal`. This file is passed via the `POSTGUI_HALFILE=` option in the `HAL` section of your ini file.

**Caution**
Do not add the GladeVCP HAL command file to the Axis `[HAL]HALFILE=` ini section, this will not have the desired effect - see the following sections.

#### 7.3.3.5 Integrating into Axis like PyVCP

Place the GladeVCP panel in the righthand side panel by specifying the following in the ini file:

```
[DISPLAY]
# add GladeVCP panel where PyVCP used to live:
GLADEVCP= -u ./hitcounter.py ./manual-example.ui

[HAL]
# HAL commands for GladeVCP components in a tab must be executed via POSTGUI_HALFILE
POSTGUI_HALFILE =  ./manual-example.hal

[RS274NGC]
# gladevcp Demo specific Oword subs live here
SUBROUTINE_PATH = ../../nc_files/gladevcp_lib
```

The HAL component name of a GladeVCP application started with the GLADEVCP option is fixed: `gladevcp`. The command line actually run by Axis in the above configuration is as follows:

```
halcmd loadusr -Wn gladevcp gladevcp -c gladevcp -x {XID} <arguments to GLADEVCP>
```

This means you may add arbitrary gladevcp options here, as long as they dont collide with the above command line options.

**Note**
The file specifiers like ./hitcounter.py, ./manual-example.ui, etc. indicate that the files are located in the same directory as the ini file. You might have to copy them to you directory (alternatively, specify a correct absolute or relative path to the file(s))

**Note**
The `[RS274NGC]SUBROUTINE_PATH=` option is only set so the example panel will find the Oword subroutine (oword.ngc) for the MDI Command widget. It might not be needed in your setup. The relative path specifier ../../nc_files/gladevcp_lib is constructed to work with directories copied by the configuration picker and when using a run-in-place setup.

#### 7.3.3.6 Embeding as a Tab

To do so, edit your .ini file and add to the DISPLAY and HAL sections of ini file as follows:

```
[DISPLAY]
# add GladeVCP panel as a tab next to Preview/DRO:
EMBED_TAB_NAME=GladeVCP demo
EMBED_TAB_COMMAND=halcmd loadusr -Wn gladevcp gladevcp -c gladevcp -x {XID} -u ./gladevcp/ ↩
    hitcounter.py ./gladevcp/manual-example.ui

[HAL]
# HAL commands for GladeVCP components in a tab must be executed via POSTGUI_HALFILE
POSTGUI_HALFILE =  ./gladevcp/manual-example.hal

[RS274NGC]
# gladevcp Demo specific Oword subs live here
SUBROUTINE_PATH = ../../nc_files/gladevcp_lib
```

Note the *halcmd loadusr* way of starting the tab command - this assures that *POSTGUI_HALFILE* will only be run after the HAL component is ready. In rare cases you might run a a command here which uses a tab but does not have an associated HAL component. Such a command can be started without *halcmd loadusr*, and this signifies to Axis that it does not have to wait for a HAL component since there is none.

When changing the component name in the above example, note that the names used in `-Wn <component>` and `-c <component>` must be identical.

Try it out by running Axis - there should be a new tab called *GladeVCP demo* near the DRO tab. Select that tab, you should see the example panel nicely fit within Axis.

**Note**
Make sure the UI file is the last option passed to GladeVCP in both the `GLADEVCP=` and `EMBED_TAB_COMMAND=` statements.

#### 7.3.3.7 Integrating into Touchy

To do add a GladeVCP tab to *Touchy*, edit your .ini file as follows:

```
[DISPLAY]
# add GladeVCP panel as a tab
EMBED_TAB_NAME=GladeVCP demo
EMBED_TAB_COMMAND=gladevcp -c gladevcp -x {XID} -u ./hitcounter.py -H ./gladevcp-touchy.hal ↩
      ./manual-example.ui

[RS274NGC]
# gladevcp Demo specific Oword subs live here
SUBROUTINE_PATH = ../../nc_files/gladevcp_lib
```

**Note**
The file specifiers like ./hitcounter.py, ./manual-example.ui, etc. indicate that the files are located in the same directory as the ini file. You might have to copy them to you directory (alternatively, specify a correct absolute or relative path to the file(s))

Note the following differences to the Axis tab setup:

- The HAL command file is slightly modified since *Touchy* does not use the *halui* components so its signals are not available and some shortcuts have been taken.
- there is no *POSTGUI_HALFILE=* ini option, but passing the HAL command file on the *EMBED_TAB_COMMAND=* line is ok
- the *halcmd loaduser -Wn ...* incantation is not needed.

### 7.3.4 GladeVCP command line options

See also *man gladevcp* . These are the gladevcp command line options:

Usage: gladevcp [options] myfile.ui

Options:

**-h, --help**
show this help message and exit

**-c NAME**
Set component name to NAME. Default is base name of UI file

**-d**
Enable debug output

**-g GEOMETRY**
Set geometry WIDTHxHEIGHT+XOFFSET+YOFFSET. Values are in pixel units, XOFFSET/YOFFSET is referenced from top left of screen. Use -g WIDTHxHEIGHT for just setting size or -g +XOFFSET+YOFFSET for just position

**-H FILE**
execute hal statements from FILE with halcmd after the component is set up and ready

**-m MAXIMUM**
force panel window to maximize. Together with the -g geometry option one can move the panel to a second monitor and force it to use all of the screen

**-t THEME**
set gtk theme. Default is system theme. Different panels can have different themes. An example theme can be found in the EMC Wiki.

**-x XID**
Re-parent GladeVCP into an existing window XID instead of creating a new top level window

**-u FILE**
Use File's as additional user defined modules with handlers

**-U USEROPT**
pass USEROPTs to Python modules

### 7.3.5 Understanding the gladeVCP startup process

The integration steps outlined above look a bit tricky, and they are. It does therefore help to understand the startup process of LinuxCNC and how this relates to gladeVCP.

The normal LinuxCNC startup process does the following:

- the realtime environment is started
- all HAL components are loaded
- the HAL components are linked together through the .hal cmd scripts
- task, iocontrol and eventually the user interface is started
- pre-gladeVCP the assumption was: by the time the UI starts, all of HAL is loaded, plumbed and ready to go

The introduction of gladeVCP brought the following issue:

- gladeVCP panels need to be embedded in a master GUI window setup, e.g. Axis, or Touchy, Gscreen, or Gmoccapy (embedded window or as an embedded tab)
- this requires the master GUI to run before the gladeVCP window can be hooked into the master GUI
- however gladeVCP is also a HAL component, and creates HAL pins of its own.
- as a consequence, all HAL plumbing involving gladeVCP HAL pins as source or destination must be run **after** the GUI has been set up

This is the purpose of the `POSTGUI_HALFILE`. This ini option is inspected by the GUIs. If a GUI detects this option, it runs the corresponding HAl file after any embedded gladVCP panel is set up. However, it does not check whether a gladeVCP panel is actually used, in which case the HAL cmd file is just run normally. So if you do NOT start gladeVCP through `GLADEVCP` or `EMBED_TAB` etc, but later in a separate shell window or some other mechanism, a HAL command file in `POSTGUI_HALFILE` will be executed too early. Assuming gladeVCP pins are referenced herein, this will fail with an error message indicating that the gladeVCP HAL component is not available.

So, in case you run gladeVCP from a separate shell window (i.e. not started by the GUI in an embedded fashion):

- you cannot rely on the `POSTGUI_HALFILE` ini option causing the HAL commands being run *at the right point in time*, so comment that out in the ini file
- explicitly pass the HAL command file which refers to gladeVCP pins to gladeVCP with the *-H <halcmd file>* option (see previous section).

### 7.3.6 HAL Widget reference

GladeVcp includes a collection of Gtk widgets with attached HAL pins called HAL Widgets, intended to control, display or otherwise interact with the LinuxCNC HAL layer. They are intended to be used with the Glade user interface editor. With proper installation, the HAL Widgets should show up in Glade's *HAL Python* widget group. Many HAL specific fields in the Glade *General* section have an associated mouse-over tool tip.

HAL signals come in two variants, bits and numbers. Bits are off/on signals. Numbers can be "float", "s32" or "u32". For more information on HAL data types see the HAL manual. The GladeVcp widgets can either display the value of the signal with an indicator widget, or modify the signal value with a control widget. Thus there are four classes of GladeVcp widgets that you can connect to a HAL signal. Another class of helper widgets allow you to organize and label your panel.

- Widgets for indicating "bit" signals: HAL_LED
- Widgets for controlling "bit" signals: HAL_Button HAL_RadioButton HAL_CheckButton
- Widgets for indicating "number" signals: HAL_Label, HAL_ProgressBar, HAL_HBar and HAL_VBar, HAL_Meter
- Widgets for controlling "number" signals: HAL_SpinButton, HAL_HScale and HAL_VScale, Jog Wheel, Speed Control
- Sensitive control widgets: State_Sensitive_Table HAL_Table and HAL_HBox
- Tool Path preview: HAL_Gremlin
- Widgets to show axis positions: DRO Widget, Combi DRO Widget
- Widgets for file handling: IconView File Selection
- Widgets for display/edit of all axes offsets: OffsetPage
- Widgets for display/edit of all tool offsets: Tooloffset editor
- Widget for Gcode display and edit: HAL_Sourceview
- widget for MDI input and history display: MDI History

#### 7.3.6.1 Widget and HAL pin naming

Most HAL widgets have a single associated HAL pin with the same HAL name as the widget (glade: General→Name).

Exceptions to this rule currently are.

- *HAL_Spinbutton* and *HAL_ComboBox*, which have two pins: a `<widgetname>-f` (float) and a `<widgetname>-s` (s32) pin
- *HAL_ProgressBar*, which has a `<widgetname>-value` input pin, and a `<widgetname>-scale` input pin.

#### 7.3.6.2 Python attributes and methods of HAL Widgets

HAL widgets are instances of GtKWidgets and hence inherit the methods, properties and signals of the applicable GtkWidget class. For instance, to figure out which GtkWidget-related methods, properties and signals a *HAL_Button* has, lookup the description of GtkButton in the PyGtk Reference Manual.

An easy way to find out the inheritance relationship of a given HAL widget is as follows: run glade, place the widget in a window, and select it; then choose the *Signals* tab in the *Properties* window. For example, selecting a *HAL_LED* widget, this will show that a *HAL_LED* is derived from a *GtkWidget*, which in turn is derived from a *GtkObject*, and eventually a *GObject*.

HAL Widgets also have a few HAL-specific Python attributes:

**hal_pin**
: the underlying HAL pin Python object in case the widget has a single pin type

**hal_pin_s, hal_pin_f**
: the S32 and float pins of the *HAL_Spinbutton* and *HAL_ComboBox* widgets - note these widgets do not have a *hal_pin* attribute!

**hal_pin_scale**
: the float input pin of *HAL_ProgressBar* widget representing the maximum absolute value of input.

The are several HAL-specific methods of HAL Widgets, but the only relevant method is:

**<halpin>.get()**
: Retrieve the value of the current HAL pin, where <*halpin*> is the applicable HAL pin name listed above.

#### 7.3.6.3 Setting pin and widget values

As a general rule, if you need to set a HAL output widget's value from Python code, do so by calling the underlying Gtk *setter* (e.g. `set_active()`, `set_value()`) - do not try to set the associated pin's value by `halcomp[pinname] = value` directly because the widget will not take notice of the change!.

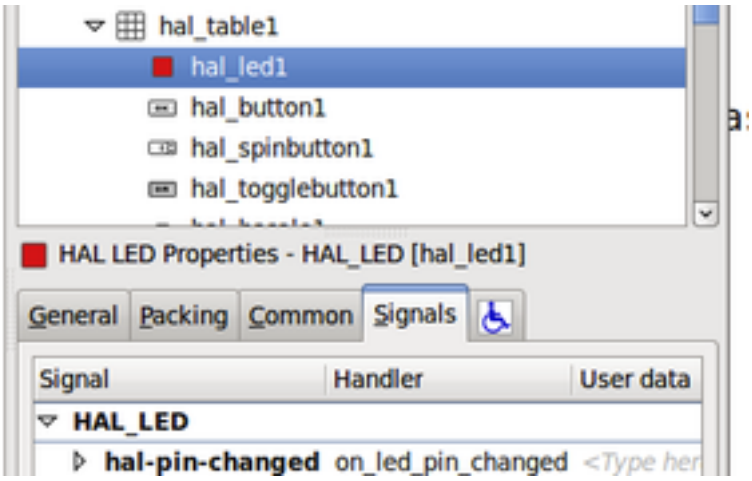
It might be tempting to *set HAL widget input pins* programmatically. Note this defeats the purpose of an input pin in the first place - it should be linked to, and react to signals generated by other HAL components. While there is currently no write protection on writing to input pins in HAL Python, this doesn't make sense. You might use setp pinname value in the associated halfile for testing though.

It is perfectly OK to set an output HAL pin's value with `halcomp[pinname] = value` provided this HAL pin is not associated with a widget, that is, has been created by the `hal_glib.GPin(halcomp.newpin(<name>,<type>,<direction>)` method (see GladeVCP Programming for an example).

#### 7.3.6.4 The hal-pin-changed signal

Event-driven programming means that the UI tells your code when "something happens" - through a callback, like when a button was pressed. The output HAL widgets (those which display a HAL pin's value) like LED, Bar, VBar, Meter etc, support the *hal-pin-changed* signal which may cause a callback into your Python code when - well, a HAL pin changes its value. This means there's no more need for permanent polling of HAL pin changes in your code, the widgets do that in the background and let you know.

Here is an example how to set a `hal-pin-changed` signal for a HAL_LED in the Glade UI editor:

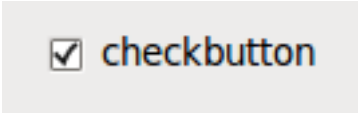


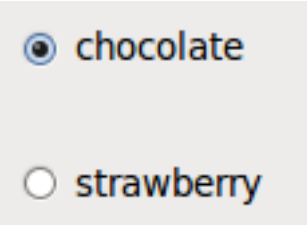

The example in `configs/apps/gladevcp/complex` shows how this is handled in Python.

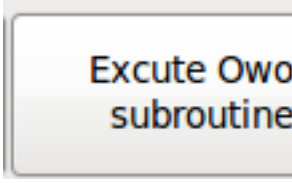
#### 7.3.6.5 Buttons

This group of widgets are derived from various Gtk buttons and consists of HAL_Button, HAL_ToggleButton, HAL_RadioButton and CheckButton widgets. All of them have a single output BIT pin named identical to the widget. Buttons have no additional properties compared to their base Gtk classes.

- HAL_Button: instantaneous action, does not retain state. Important signal: `pressed`
- HAL_ToggleButton, HAL_CheckButton: retains on/off state. Important signal: `toggled`
- HAL_RadioButton: a one-of-many group. Important signal: `toggled` (per button).
- Important common methods: `set_active()`, `get_active()`
- Important properties: `label`, `image`

Check button: 

Radio buttons: 

Toggle button: 

.

> **Tip**
> Defining radio button groups in Glade:

+ - decide on default active button

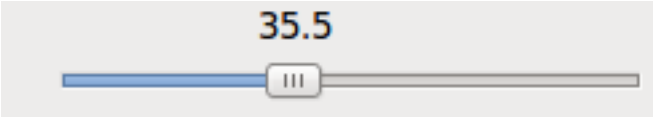
+ - in the other button's *General→Group* select the default active button's name in the *Choose a Radio Button in this project* dialog.

+ See `configs/apps/gladevcp/by-widget/` for a GladeVCP applications and UI file for working with radio buttons.

#### 7.3.6.6 Scales

HAL_HScale and HAL_VScale are derived from the GtkHScale and GtkVScale respectively. They have one output FLOAT pin with name equal to widget name. Scales have no additional properties.

To make a scale useful in Glade, add an *Adjustment* (General→Adjustment→New or existing adjustment) and edit the adjustment object. It defines the default/min/max/increment values. Also, set adjustment *Page size* and *Page increment* to zero to avoid warnings.

Example HAL_HScale: 

.

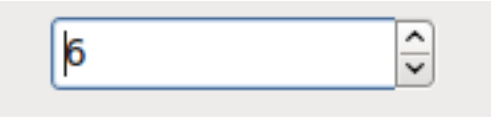
#### 7.3.6.7 SpinButton

HAL SpinButton is derived from GtkSpinButton and holds two pins:

**<widgetname>-f**
out FLOAT pin

**<widgetname>-s**
out S32 pin

To be useful, Spinbuttons need an adjustment value like scales, see above.

Example SpinButton: 


#### 7.3.6.8 Hal_Dial

The hal_dial widget simulates a jogwheel or adjustment dial.
It can be operated with the mouse. You can just use the mouse wheel, while the mouse cursor is over the Hal_Dial widget, or you hold the left mouse button and move the cursor in circular direction to increase or degrease the counts.
By double clicking the left or right button the scale factor can be increased or decreased.

- Counterclockwise = reduce counts
- Clockwise = increase counts
- Wheel up = increase counts
- Wheel down = reduce counts
- left Double Click = x10 scale
- Right Double Click = /10 scale

```
Hal_Dial exports it's count value as hal pins:

<widgetname>::
   out S32 pin
<widgetname>-scaled::
   out FLOAT pin
<widgetname>-delta-scaled::
   out FLOAT pin
```

It has the following properties:

**cpr**
Sets the Counts per Revolution, allowed values are in the range from 25 to 360
default = 100

**show_counts**
Set this to False, if you want to hide the counts display in the middle of the widget.
default = True

**label**
Set the content of the label witch may be shown over the counts value.
If the label given is longer than 15 Characters, it will be cut to 15 Characters.
default = blank

**center_color**
This allows one to change the color of the wheel. It uses a GDK color string.
default = #bdefbdefbdef (gray)

**count_type_shown**
There are three counts available 0) Raw CPR counts 1) Scaled counts 2) Delta scaled counts.
default = 1

- count is based on the CPR selected - it will count positive and negative. It is available as a S32 pin.
- Scaled-count is CPR count times the scale - it can be positive and negative.
  If you change the scale the output will immediately reflect the change. It is available as a FLOAT pin.
- Delta-scaled-count is cpr count CHANGE, times scale.
  If you change the scale, only the counts after that change will be scaled and then added to the current value.
  It is available as a FLOAT pin.

**scale_adjustable**
Set this to False if you want to disallow scale changes by double clicking the widget.
If this is false the scale factor will not show on the widget.
default = True

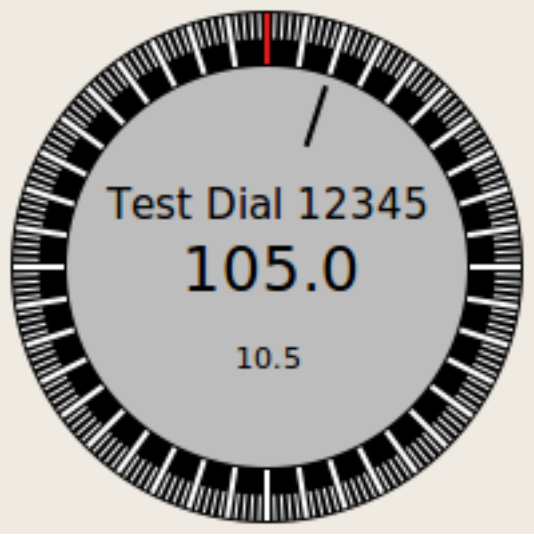
**scale**
Set this to scale the counts.
default = 1.0

**Direct program control**
There are ways to directly control the widget using Python.

```
Using goobject to set the above listed properties:
    [widget name].set_property("cpr",int(value))
    [widget name].set_property("show_counts, True)
    [widget name].set_property("center_color",gtk.gdk.Color('#bdefbdefbdef'))
    [widget name].set_property('label', 'Test Dial 12345')
    [widget name].set_property('scale_adjustable', True)
    [widget name].set_property('scale', 10.5)
    [widget name].set_property('count_type_shown', 0)

There are python methods:
    [widget name].get_value()
        Will return the counts value as a s32 integer
    [widget name].get_scaled_value()
        Will return the counts value as a float
```

```
        [widget name].get_delta_scaled_value()
            Will return the counts value as a float
        [widget name].set_label("string")
            Sets the label content with "string"

    There are two GObject signals emitted:
        count_changed
            emitted when the widget's count changes eg. from being wheel scrolled.
        scale_changed
            emitted when the widget's scale changes eg. from double clicking. +
        connect to these like so:
            [widget name].connect('count_changed', [count function name])
            [widget name].connect('scale_changed', [scale function name]) +
        The callback functions would use this pattern:
            def [count function name](widget, count,scale,delta_scale):
        This will return: the widget, the current count, scale and delta scale of that wid
```

Example Hal_Dial:




#### 7.3.6.9 Jog Wheel

The jogwheel widget simulates a real jogwheel.
It can be operated with the mouse. You can just use the mouse wheel, while the mouse cursor is over the JogWheel widget, or you push the left mouse button and move the cursor in circular direction to increase or degrease the counts.

- Counterclockwise = reduce counts
- Clockwise = increase counts
- Wheel up = increase counts
- Wheel down = reduce counts

As moving the mouse the drag and drop way may be faster than the widget can update itself, you may loose counts turning to fast. It is recommended to use the mouse wheel, and only for very rough movements the drag and drop way.

JogWheel exports it's count value as hal pin:

**<widgetname>-s**
out S32 pin

It has the following properties:

**size**
Sets the size in pixel of the widget, allowed values are in the range of 100 to 500 default = 200

**cpr**
Sets the Counts per Revolution, allowed values are in the range from 25 to 100 default = 40

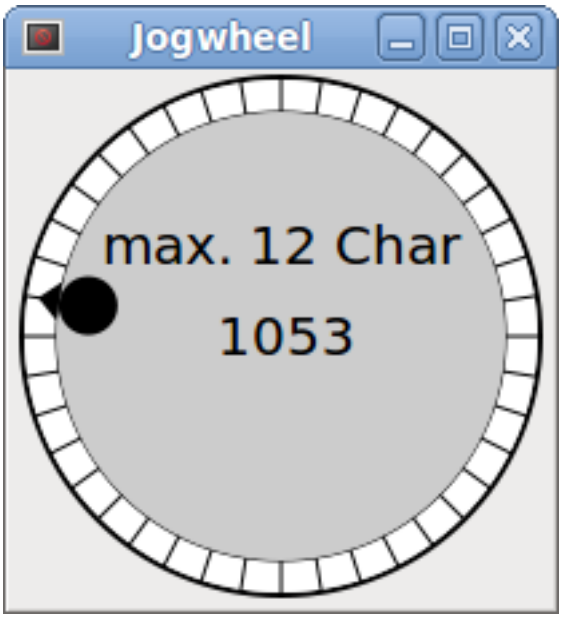
**show_counts**
Set this to False, if you want to hide the counts display in the middle of the widget.

**label**
Set the content of the label witch may be shown over the counts value. The purpose is to give the user an idea about the usage of that jogwheel. If the label given is longer than 12 Characters, it will be cut to 12 Characters.

**Direct program control**
There a couple ways to directly control the widget using Python.

```
Using gobject to set the above listed properties:
    [widget name].set_property("size",int(value))
    [widget name].set_property("cpr",int(value))
    [widget name].set_property("show_counts, True)

There are two python methods:
    [widget name].get_value()
    Will return the counts value as integer
    [widget name].set_label("string")
    Sets the label content with "string"
```

Example JogWheel:



#### 7.3.6.10 Speed Control

SpeedControl is a widget specially made to control an adjustment with a touch screen. It is a replacement to the normal scale widget witch is difficult to slide on a touch screen.

The value is controlled with two button to increase or decrease the value. The Increment will change as long a button is pressed. The value of each increment as well as the time between two changes can be set using the widget properties.

SpeedControl offers some hal pin:

**<widgetname>-value**
: out float pin The shown value of the widget

**<widgetname>-scaled-value**
: out float pin The shown value divided by the scale value, this is very useful, if the velocity is shown in units / min, but linuxcnc expects it to be in units / second

**<widgetname>-scale**
: in float pin The scale to apply Default is 60

**<widgetname>-increase**
: in bit pin As long as the pin is true, the value will increase Very handy with connected momentary switch

**<widgetname>-decrease**
: in bit pin As long as the pin is true, the value will decrease Very handy with connected momentary switch

It has the following properties:

**height**
: integer The height of the widget in pixel allowed values are 24 to 96 default is 36

**value**
: float The start value to set allowed values are in the range from 0.001 to 99999.0 default is 10.0

**min**
: float The min allowed value allowed values are 0.0 to 99999.0 default is 0.0 If you change this value, the increment will be reseted to default, so it might be necessary to set afterwards a new increment.

**max**
: float The max allowed value allowed values are 0.001 to 99999.0 default is 100.0 If you change this value, the increment will be reseted to default, so it might be necessary to set afterwards a new increment.

**increment**
: float sets the applied increment per mouse click allowed values are 0.001 to 99999.0 and -1 default is -1 resulting in 100 increments from min to max

**inc_speed**
: integer Sets the timer delay for the increment speed holding pressed the buttons allowed values are 20 to 300 default is 100

**unit**
: string Sets the unit to be shown in the bar after the value any string is allowed default is ""

**color**
: Color Sets the color of the bar any hex color is allowed default is "#FF8116"

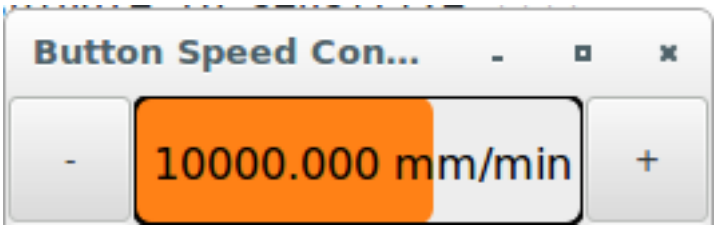
**template**
: String Text template to display the value Python formatting is used Any allowed format default is "%.1f"

**do_hide_button**
: Boolean Whether to show or hide the increment an decrement button True or False Default = False

**Direct program control**
: There a couple ways to directly control the widget using Python.

```
Using gobject to set the above listed properties:
    [widget name].set_property("do_hide_button",bool(value))
    [widget name].set_property("color","#FF00FF")
    [widget name].set_property("unit", "mm/min")
            etc.

There are also python methods to modify the widget:
    [widget name].set_adjustment(gtk-adjustment)
    You can assign a existing adjustment to the control, that way it is easy to replac
    existing sliders without many code changes. Be aware, that after changing the adju
    you may need to set a new increment, as it will be reseted to its default (100 ste
    [widget name].get_value()
    Will return the counts value as float
    [widget name].set_value(float(value))
    Sets the widget to the commanded value
    [widget name].set_digits(int(value))
    Sets the digits of the value to be used
    [widget name].hide_button(bool(value))
    Hide or show the button
```

Example Speedcontrol:




#### 7.3.6.11 Label

HAL_Label is a simple widget based on GtkLabel which represents a HAL pin value in a user-defined format.

**label_pin_type**
The pin's HAL type (0:S32, 1:float, 2:U32), see also the tooltip on 'General→HAL pin type '(note this is different from PyVCP which has three label widgets, one for each type).

**text_template**
Determines the text displayed - a Python format string to convert the pin value to text. Defaults to `%s` (values are converted by the str() function) but may contain any legit as an argument to Pythons format() method.
Example: `Distance:  %.03f` will display the text and the pin value with 3 fractional digits padded with zeros for a FLOAT pin.

#### 7.3.6.12 Containers

- HAL_HideTable
- HAL_Table State_Sensitive_Table
- HAL_HBox

These containers are meant to be used to sensitize (grey out) or hide their children.
Insensitived children will not respond to input.
HAL_HideTable has one HAL BIT input pin which controls if it's child widgets are hidden or not.
If the pin is low then child widgets are visible which is the default state.

HAL_Table and HAL_Hbox have one HAL BIT input pin which controls if their child widgets are sensitive or not.
If the pin is low then child widgets are inactive which is the default state.
State_Sensitive_table responds to the state to linuxcnc's interpreter.
optionally selectable to respond to *must-be-all-homed*,*must-be-on* and *must-be-idle*
You can combine them. It will always be insensitive at Estop.

* HAL_Hbox is depreceiated - use HAL_Table.
If current panels use it it won't fail. You just won't find it in the GLADE editor anymore.
Future vesions of gladeVCP may remove this widget completely and then you will need to update the panel.

**Tip**
If you find some part of your GladeVCP application is *grayed out* (insensitive), see whether a HAL_Table pin is unset or unconnected.

#### 7.3.6.13 LED

The hal_led simulates a real indicator LED.
It has a single input BIT pin which controls it's state: ON or OFF.
LEDs have several properties which control their look and feel:

**on_color**
: a String defining ON color of LED. May be any valid gtk.gdk.Color name. Not working on Ubuntu 8.04.

**off_color**
: String defining OFF color of LED. May be any valid gtk.gdk.Color name or special value `dark`. `dark` means that OFF color will be set to 0.4 value of ON color. Not working on Ubuntu 8.04.

**pick_color_on, pick_color_off**
: Colors for ON and OFF states may be represented as `#RRRRGGGGBBBB` strings. These are optional properties which have precedence over `on_color` and `off_color`.

**led_size**
: LED radius (for square - half of LED's side)

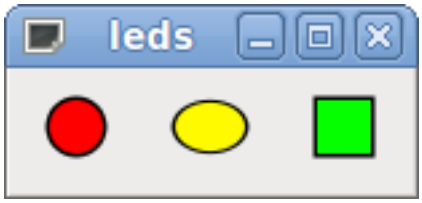
**led_shape**
: LED Shape. Valid values are 0 for round, 1 for oval and 2 for square shapes.

**led_blink_rate**
: if set and LED is ON then it's blinking. Blink period is equal to "led_blink_rate" specified in milliseconds.

**create hal pin**
: select/deselect making of HAL pin to control LED. With no HAL pin created LED can be controlled with a python function. As an input widget, LED also supports the `hal-pin-changed signal`. If you want to get a notification in your code when the LED's HAL pin was changed, then connect this signal to a handler, for example `on_led_pin_changed` and provide the handler as follows:

```
def on_led_pin_changed(self,hal_led,data=None):
    print "on_led_pin_changed() - HAL pin value:",hal_led.hal_pin.get()
```

This will be called at any edge of the signal and also during program start up to report the current value.

Example LEDs: 

#### 7.3.6.14 ProgressBar

**Note**
This widget might go away. Use the HAL_HBar and HAL_VBar widgets instead.

The HAL_ProgressBar is derived from gtk.ProgressBar and has two float HAL input pins:

**<widgetname>**
the current value to be displayed

**<widgetname>-scale**
the maximum absolute value of input

It has the following properties:

**scale**
value scale. set maximum absolute value of input. Same as setting the <widgetname>.scale pin. A float, range from $-2^{24}$ to $+2^{24}$.

**green_limit**
green zone limit lower limit

**yellow_limit**
yellow zone limit lower limit

**red_limit**
red zone limit lower limit

**text_template**
Text template to display the current value of the `<widgetname>` pin. Python formatting may be used for dict `{"value":value`

Example HAL_ProgressBar:

#### 7.3.6.15 ComboBox

HAL_ComboBox is derived from gtk.ComboBox. It enables choice of a value from a dropdown list.

It exports two HAL pins:

**<widgetname>-f**
the current value, type FLOAT

**<widgetname>-s**
the current value, type S32

It has the following property which can be set in Glade:

**column**
the column index, type S32, defaults to -1, range from -1..100 .

In default mode this widgets sets the pins to the index of the chosen list entry. So if your widget has three labels, it may only assume values 0,1 and 2.

In column mode (column > -1), the value reported is chosen from the ListStore array as defined in Glade. So typically your widget definition would have two columns in the ListStore , one with text displayed in the dropdown, and an int or float value to use for that choice.

There's an example in `configs/apps/by-widget/combobox.{py,ui}` which uses column mode to pick a float value from the ListStore.

If you're confused like me about how to edit ComboBox ListStores and CellRenderer, see http://www.youtube.com/watch?v=Z5_F-rW2cL8.

#### 7.3.6.16 Bars

HAL Bar and VBar widgets for horizontal and vertical bars representing float values. They have one input FLOAT hal pin. Both bars have the following properties:

**invert**
: Swap min and max direction. An inverted HBar grows from right to left, an inverted VBar from top to bottom.

**min, max**
: Minimum and maximum value of desired range. It is not an error condition if the current value is outside this range.

**show limits**
: Used to select/deselect the limits text on bar.

**zero**
: Zero point of range. If it's inside of min/max range then the bar will grow from that value and not from the left (or right) side of the widget. Useful to represent values that may be both positive or negative.

**force_width, force_height**
: Forced width or height of widget. If not set then size will be deduced from packing or from fixed widget size and bar will fill whole area.

**text_template**
: Like in Label sets text format for min/max/current values. Can be used to turn off value display.

**value**
: Sets the bar display to the value entered: used only for testing in GLADE editor. The vaue will be set from A HAL pin.

**target value**
: Sets the target line to the value entered: used only for testing in GLADE editor. The value will can be set in a Python function

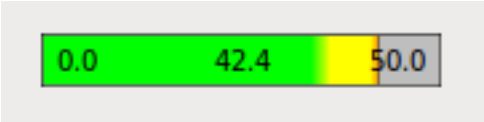
**target_width**
: Width of the line that marks the target value.

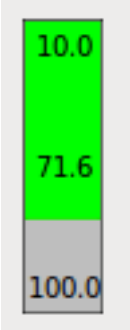
**bg_color**
: Background (inactive) color of bar.

**target_color**
: Color of the the target line.

**z0_color, z1_color, z2_color**
: Colors of different value zones. Defaults are `green`, `yellow` and `red`. For description of zones see `z*_border` properties.

**z0_border, z1_border**
: Define up bounds of color zones. By default only one zone is enabled. If you want more then one zone set `z0_border` and `z1_border` to desired values so zone 0 will fill from 0 to first border, zone 1 will fill from first to second border and zone 2 — from last border to 1. Borders are set as fractions, values from 0 to 1.

Horizontal bar: 

Vertical bar: 

.

#### 7.3.6.17 Meter

HAL Meter is a widget similar to PyVCP meter - it represents a float value and has one input FLOAT hal pin. HAL Meter has the following properties:

**min, max**
: Minimum and maximum value of desired range. It is not an error condition if the current value is outside this range.

**force_size**
: Forced diameter of widget. If not set then size will be deduced from packing or from fixed widget size and meter will fill all available space with respect to aspect ratio.

**text_template**
: Like in Label sets text format for current value. Can be used to turn off value display.

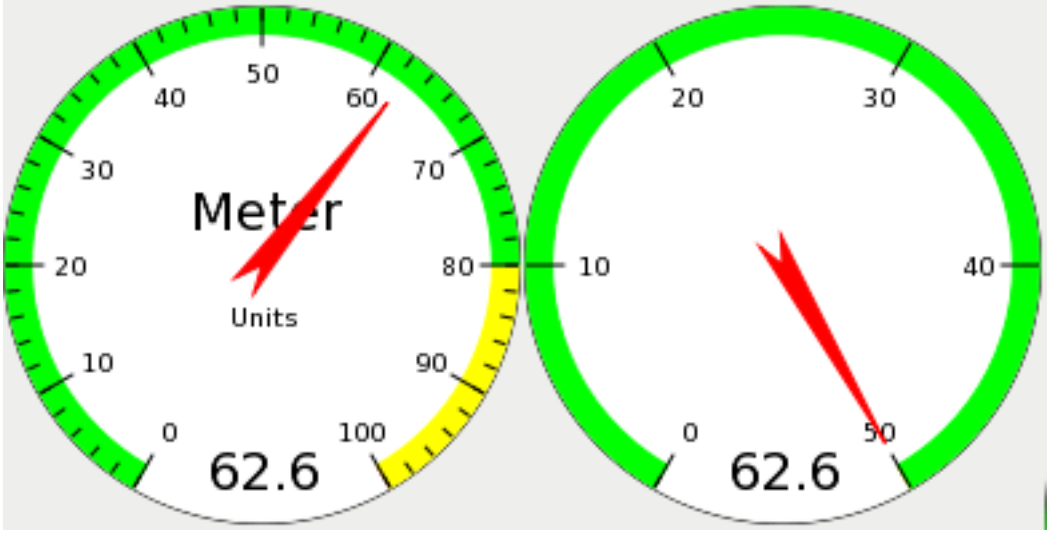
**label**
: Large label above center of meter.

**sublabel**
: Small label below center of meter.

**bg_color**
: Background color of meter.

**z0_color, z1_color, z2_color**
: Colors of different value zones. Defaults are `green`, `yellow` and `red`. For description of zones see `z*_border` properties.

**z0_border, z1_border**
: Define up bounds of color zones. By default only one zone is enabled. If you want more then one zone set `z0_border` and `z1_border` to desired values so zone 0 will fill from min to first border, zone 1 will fill from first to second border and zone 2 — from last border to max. Borders are set as values in range min-max.

Example HAL Meters: 


#### 7.3.6.18 HAL_Graph

This widget is for plotting values over time.

#### 7.3.6.19 Gremlin tool path preview for .ngc files

Gremlin is a plot preview widget similar to the Axis preview window. It assumes a running LinuxCNC environment like Axis or Touchy. To connect to it, inspects the INI_FILE_NAME environment variable. Gremlin displays the current .ngc file - it does monitor for changes and reloads the ngc file if the file name in Axis/Touchy changes. If you run it in a GladeVCP application when LinuxCNC is not running, you might get a traceback because the Gremlin widget can't find LinuxCNC status, like the current file name.

Gremlin does not export any HAL pins. It has the following properties:

**show tool speed**
: This displays the tool speed. Defaults true

**show commanded**
: This selects the DRO to use commanded or actual values. Defaults true

**use metric units**
: This selects the DRO to use metric or imperial units. Defaults true

**show rapids**
: This tells the plotter to show the rapid moves. Defaults true

**show DTG**
: This selects the DRO to display the distance-to-go value. Defaults true

**show relative**
: This selects the DRO to show values relative to user system or machine cordinates. Defaults true

**show live plot**
: This tells the plotter to draw or not. Defaults true

**show limits**
: This tells the plotter to show the machine's limits. Defaults true

**show lathe radius**
: This selects the DRO to display the X axis in radius or diameter, if in lathe mode (selectable in the INI file with LATHE = 1). Defaults false

**show extents**
This tells the plotter to show the extents. Defaults true

**show tool**
This tells the plotter to draw the tool. Defaults true

**show program**
TODO

**use joints mode**
Used in non trivialkins machines (eg robots). Defaults false

**grid size**
Sets the size of the grid. which is only visible in the X, Y and Z view. Defaults to 0

**use default mouse controls**
This disables the default mouse controls. This is most useful when using a touchscreen as the default controls do not work well. You can programically add controls using python and the handler file technique. Defaults to *True*

**view**
may be any of `x`, `y`, *y2* , `z`, *z2* , `p` (perspective) . Defaults to `z` view.

**enable_dro**
boolean; whether to draw a DRO on the plot or not. Defaults to `True`

**mouse_btn_mode**
integer; mouse button handling, leads to different functions of the button 0 = default: left rotate, middle move, right zoom 1 = left zoom, middle move, right rotate 2 = left move, middle rotate, right zoom 3 = left zoom, middle rotate, right move 4 = left move, middle zoom, right rotate 5 = left rotate, middle zoom, right move 6 = left move, middle zoom, right zoom

```
mode 6 is reccomended for plasmas and lathes, as rotation is not needed for such machi
```

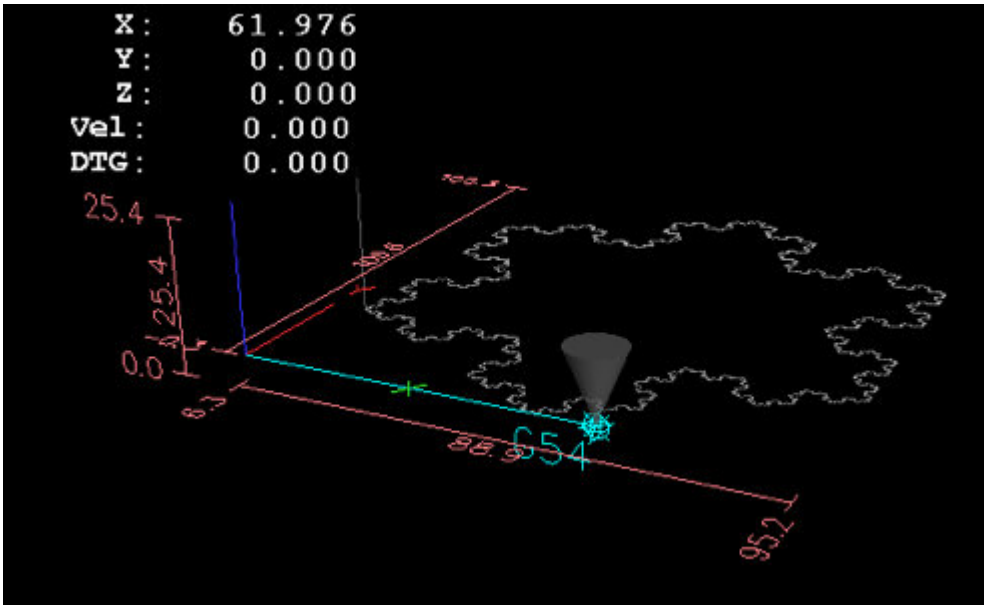
**Direct program control**
There a couple ways to directly control the widget using Python.

```
Using goobject to set the above listed properties:
    [widget name].set_property('view','P')
    [widget name].set_property('metric_units',False)
    [widget name].set_property('use_default_controls',False)
    [widget name].set_property('enable_dro' False))
    [widget name].set_property('show_program', False)
    [widget name].set_property('show_limits', False)
    [widget name].set_property('show_extents_option', False)
    [widget name].set_property('show_live_plot', False)
    [widget name].set_property('show_tool', False)
    [widget name].set_property('show_lathe_radius',True)
    [widget name].set_property('show_dtg',True)
    [widget name].set_property('show_velocity',False)
    [widget name].set_property('mouse_btn_mode', 4)

There are python methods:
    [widget name].show_offsets = True
    [widget name].grid_size =  .75
    [widget name].select_fire(event.x,event.y)
    [widget name].select_prime(event.x,event.y)
    [widget name].start_continuous_zoom(event.y)
    [widget name].set_mouse_start(0,0)
    [widget name].gremlin.zoom_in()
    [widget name].gremlin.zoom_out()
```

```
[widget name].get_zoom_distance()
[widget name].set_zoom_distance(dist)
[widget name].clear_live_plotter()
[widget name].rotate_view(x,y)
[widget name].pan(x,y)
```

**Hints**

- If you set all the plotting options false but show_offsets true you get an offsets page instead of a graphics plot.
- If you get the zoom distance before changing the view then reset the zoom distance, it's much more user friendly.
- if you select an element in the preview, the selected element will be used as rotation center point

Example: 


#### 7.3.6.20 HAL_Offset

The HAL_Offset widget is used to display the offset of a single axis. It has the following properties:

**Joint Number**

Used to select which axis (technically which joint) is displayed. On a trivialkins machine (mill, lathe, router) axis vrs joint number are:

```
    0:X  1:Y  2:Z  3:A  4:B  5:C  6:U  7:V  8:W
Text template for metric units::
    You can use python formatting to display the position with different precision.
Text template for imperial units::
    You can use python formatting to display the position with different precision.
Reference Type::
    0:G5x 1:tool 2:G92 3:Rotation around Z
```

#### 7.3.6.21 DRO widget

The DRO widget is used to display the current axis position. It has the following properties:

**Actual Position**

select actual (feedback) position or commanded position.

**Text template for metric units**
You can use python formatting to display the position with different precision.

**Text template for imperial units**
You can use python formatting to display the position with different precision.

**Reference Type**
Absolute (machine origin), Relative (to current user coordinate origin - G5x) or Distance-to-go (relative to current user coordinate origin)

**Joint Number**
Used to select which axis (technically which joint) is displayed. On a trivialkins machine (mill, lathe, router) axis vrs joint number are:

```
0:X  1:Y  2:Z  3:A  4:B  5:C  6:U  7:V  8:W
```

**Display units**
Used to toggle the display units between metric and imperial.

**Hints**

- If you want the display to be right justified, set the X align to 1.0
- If you want different colors or size or text change the attributes in the glade editor (eg scale is a good way to change the size of the text)
- The background of the widget is actually see through - so if you place if over an image the DRO numbers will show on top of it with no backgroud. There is a special technique to do this. See the animated function diagrams below.
- The DRO widget is a modified gtk label widget. As such much or what can be done to a gtk label can be done to DRO widget.

**Direct program control**
There a couple ways to directly control the widget using Python.

```
Using goobject to set the above listed properties:
    [widget name].set_property("display_units_mm",True)
    [widget name].set_property("actual",True)
    [widget name].set_property("mm_text_template","%f")
    [widget name].set_property("imperial_text_template","%f")
    [widget name].set_property("Joint_number",3)
    [widget name].set_property("reference_type",3)

There are two python methods:
    [widget name].set_dro_inch()
    [widget name].set_dro_metric()
```

#### 7.3.6.22 Combi_DRO widget

The Combi_DRO widget is used to display the current , the relative axis position and the distance to go in one DRO.
By clicking on the DRO the Order of the DRO will toggle around.
In Relative Mode the actual coordinate system will be displayed.

It has the following properties:

**joint_number**
Used to select which axis (technically which joint) is displayed.
On a trivialkins machine (mill, lathe, router) axis vrs. joint number are:
*0:X 1:Y 2:Z etc*

**actual**
select actual (feedback) or commanded position.

**metric_units**
Used to toggle the display units between metric and imperial.

**auto_units**
Units will toggle between metric and imperial according to the active gcode being G20 or G21
default is TRUE

**diameter**
Whether to display position as diameter or radius, in diameter mode the DRO will display the joint value multiplied by 2

**mm_text_template**
You can use python formatting to display the position with different precision.
default is "%10.3f"

**imperial_text_template**
You can use python formatting to display the position with different precision.
default is "%9.4f"

**homed_color**
The foreground color of the DRO numbers if the joint is homed
default is green

**unhomed_color**
The foreground color of the DRO numbers if the joint is not homed
default is red

**abs_color**
the background color of the DRO, if main DRO shows absolute coordinates
default is blue

**rel_color**
the background color of the DRO, if main DRO shows relative coordinates
default is black

**dtg_color**
the background color of the DRO, if main DRO shows distance to go
default is yellow

**font_size**
The font size of the big numbers, the small ones will be 2.5 times smaller, the value must be an integer in the range of 8 to 96,
default is 25

**toggle_readout**
A left mouse click will toggle the DRO readout through the different modes ["Rel", "Abs", "DTG"].
By unchecking the box you can disable that behavior. The toggling can still be done with [widget name].toggle_readout()
Value must be bool
default is TRUE

**cycle_time**
The time the DRO waits between two polls, the value must be an integer in the range of 100 to 1000,
default is 150, this setting should only be changed if you use more
than 5 DRO at the same time, i.e. on a 6 axis config, to avoid, that
the DRO slows down the main application too much.

**Direct program control**
Using gobject to set the above listed properties:

```
[widget name].set_property(property, value)
```

There are several python methods to control the widget:

```
[widget name].set_to_inch(state)
    sets the DRO to show imperial units
    state = boolean (True or False)

[widget name].set_auto_units(state)
    if True the DRO will change units according to active gcode (G20 / G21)
    state = boolean (True or False)
    Default is True

[widget name].set_to_diameter(state)
    if True the DRO will show the diameter not the radius, specially needed for lathes
    the DRO will display the axis value multiplied by 2
    state = boolean (True or False)
    Default is False

[widget name].toggle_readout()
    toggles the order of the DRO in the widget

[widget name].change_axisletter(letter)
    changes the automatically given axis letter
    very useful to change an lathe DRO from X to R or D
    letter = string

[widget name].get_order()
    returns the order of the DRO in the widget mainly used to maintain them consistent
    the order will also be transmitted with the clicked signal
    returns a list containing the order

[widget name].set_order(order)
    sets the order of the DRO, mainly used to maintain them consistent
    order = list object, must be one of
      ["Rel", "Abs", "DTG"]
      ["DTG", "Rel", "Abs"]
      ["Abs", "DTG", "Rel"]
    Default = ["Rel", "Abs", "DTG"]

[widget name].get_position()
    returns the position of the DRO as a list of floats
    the order is independent of the order shown on the DRO
    and will be given as [Absolute , relative , DTG]
    Absolute = the machine coordinates, depends on the actual property
             will give actual or commanded position
    Relative = will be the coordinates of the actual coordinate system
    DTG = the distance to go, will mostly be 0, as this function should not be used
        while the machine is moving, because of time delays
```

The widget will emit the following signals:

```
clicked
    This signal is emitted, when the user has clicked on the Combi_DRO widget,
    it will send the following data:
    widget = widget object = The widget object that sends the signal
    joint_number = integer = The joint number of the DRO, where '0:X  1:Y  2:Z  etc'
    order = list object = the order of the DRO in that widget
                          the order may be used to set other Combi_DRO widgets to the same
```

```
units_changed
    This signal is emitted, if the DRO units are changed, it will send the following data:
    widget = widget object = The widget object that sends the signal
    metric_units = boolean = True if the DRO does display metric units, False in case of i

system_changed
    This signal is emitted, if the DRO units are changed, it will send the following data:
    widget = widget object = The widget object that sends the signal
    system = string = The actual coordinate system. Will be one of
                      G54 G55 G56 G57 G58 G59 G59.1 G59.2 G59.3
                      or Rel if non has been selected at all, what will only happen in Gla
```

There are some information you can get through commands, witch may be of interest for you:

```
[widget name].system
    The actual system, as mentioned in the system_changed signal

[widget name].homed
    True if the joint is homed

[widget name].machine_units
    0 if Imperial, 1 if Metric
```

Example, Three Combi_DRO in a window
X = Relative Mode
Y = Absolute Mode
Z = DTG Mode

#### 7.3.6.23 IconView (File Select)

This is touch screen friendly widget to select a file and to change directories.

The widget has the following properties:

**icon_size**
: Sets the size of the displayed icon.
Allowed values are integers in the range from 12 to 96
default is 48

**start_dir**
Sets the directory to start in when the widget is shown first time,
must be a string, containing a valid directory path,
default is "/"

**jump_to_dir**
Sets the directory "jump to" directory, witch is selected by the corresponding button in the bottom button list, the 5th button counting from the left,
must be a string, containing a valid directory path,
default is "~"

**filetypes**
Sets the file filter for the objects to be shown
Must be a string containing a comma separated list of extensions to be shown
Default is "ngc,py"

**sortorder**
Sets the sorting order of the displayed icon must be an integer value from 0 to 3, where
0 = ASCENDING (sorted according to file names)
1 = DESCENDING (sorted according to file names)
2 = FOLDERFIRST (show the folders first, then the files)
3 = FILEFIRST (show the files first, then the folders),
Default = 2 = FOLDERFIRST

**Direct program control**
Using goobject to set the above listed properties:

```
[widget name].set_property(property,Value)
```

There are python methods to control the widget:

```
[widget name].show_buttonbox(state)
    if False the bottom button box will be hidden, this is helpful in custom screens,
    with special buttons layouts to not alter the layout of the GUI, good example
    for that is gmoccapy
    state = boolean (True or False)
    Default is True

[widget name].show_filelabel(state)
    if True the file label (between the IconView window and the bottom button box will be
    Hiding this label may save place, but showing it is very useful for debugging reasons,
    state = boolean (True or False)
    Default is True

[widget name].set_icon_size(iconsize)
    sets the icon size
    must be an integer in the range from 12 to 96
    Default = 48

[widget name].set_directory(directory)
    Allows to set an directory to be shown
    directory = string (a valid file path)

[widget name].set_filetypes(filetypes)
    sets the file filter to be used, only files with the given extensions will be shown
    filetypes = string containing a comma separated list of extensions
    Default = "ngc,py"
```

```
[widget name].get_selected()
    Returns the path of the selected file, or None if an directory has been selected

[widget name].refresh_filelist()
    Refreshes the filelist, needed if you add a file without changing the directory
```

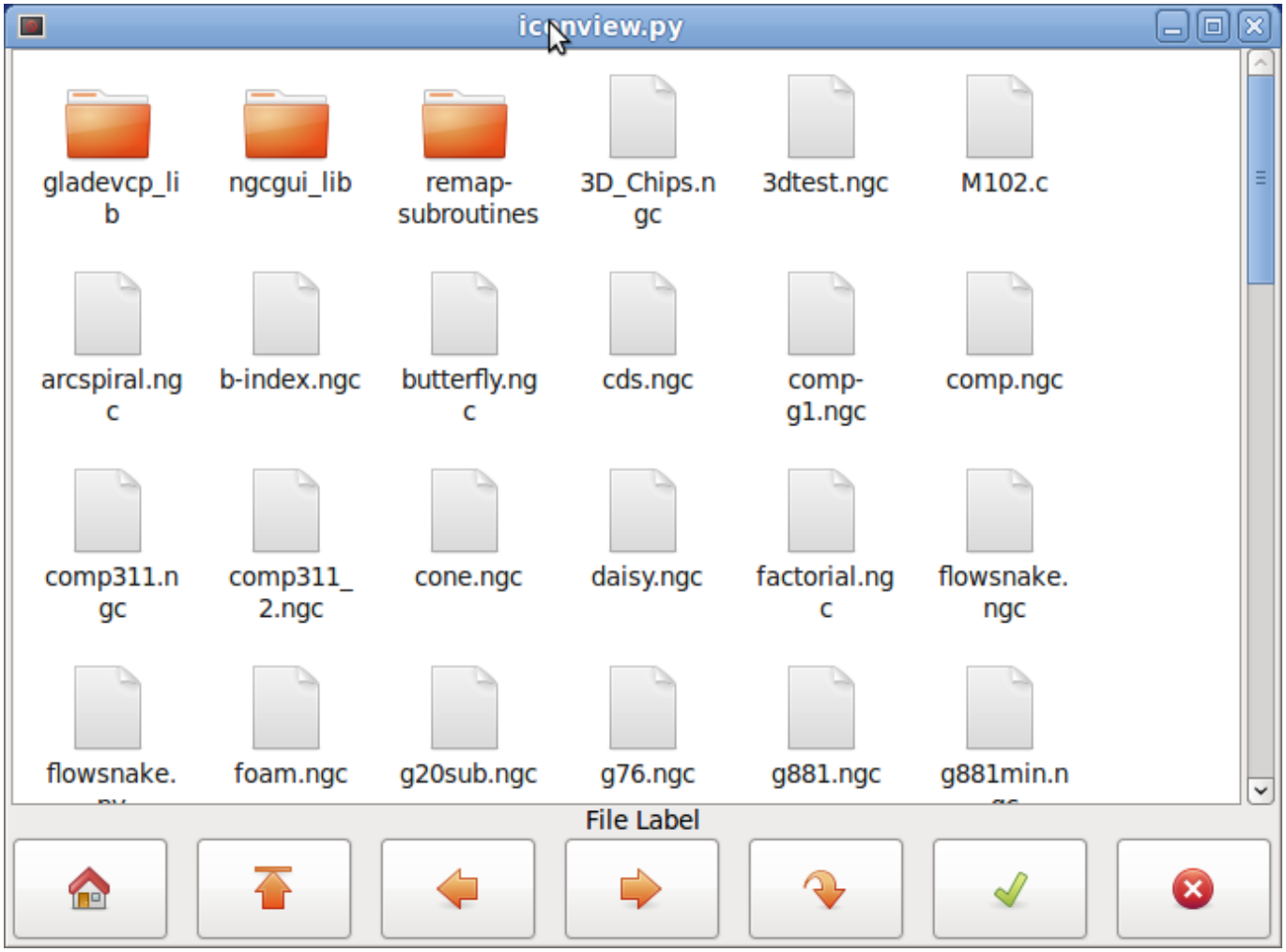
If the button box has been hidden, you can reach the functions of this button through it's clicked signals like so:

```
[widget name].btn_home.emit("clicked")
[widget name].btn_jump_to.emit("clicked")
[widget name].btn_sel_prev.emit("clicked")
[widget name].btn_sel_next.emit("clicked")
[widget name].btn_get_selected.emit("clicked")
[widget name].btn_dir_up.emit("clicked")
[widget name].btn_exit.emit("clicked")
```

The widget will emit the following signals:

```
selected
    This signal is emitted, when the user selects an icon, it will return a string contain
    file path if a file has been selected, or None if an directory has been selected
sensitive
    This signal is emitted, when the buttons change there state from sensitive to not sens
    This signal is useful to maintain surrounding GUI synchronized with the button of the
    It will return the buttonname and the new state. Buttonname is one of "btn_home", "btn
    "btn_sel_next", "btn_jump_to" or "btn_select". State is a boolean and will be True or
exit
    This signal is Emmit, when the exit button has been pressed to close the IconView
    mostly needed if the application is started as stand alone.
```

Example:




#### 7.3.6.24 Calculator widget

This is a simple calculator widget, that can be used for numerical input.
You can preset the display and retrieve the result or that preset value.
It has the following properties:

**Is editable**
This allows the entry display to be typed into from a keyboard.

**Set Font**
This allows you to set the font of the display.

**Direct program control**
There a couple ways to directly control the widget using Python.

```
Using goobject to set the above listed properties:
    [widget name].set_property("is_editable",True)
    [widget name].set_property("font","sans 25")

There are python methods:
   [widget name].set_value(2.5)
        This presets the display and is recorded.
   [widget name].set_font("sans 25")
   [widget name].set_editable(True)
```

```
[widget name].get_value()
     Returns the calculated value - a float.
[widget name].set_editable(True)
[widget name].get_preset_value()
     Returns the recorded value: a float.
```

#### 7.3.6.25 Tooleditor widget

This is a tooleditor widget for displaying and modifying a tool editor file.
It checks the current file once a second to see if linuxcnc updated it.
It has the following properties:

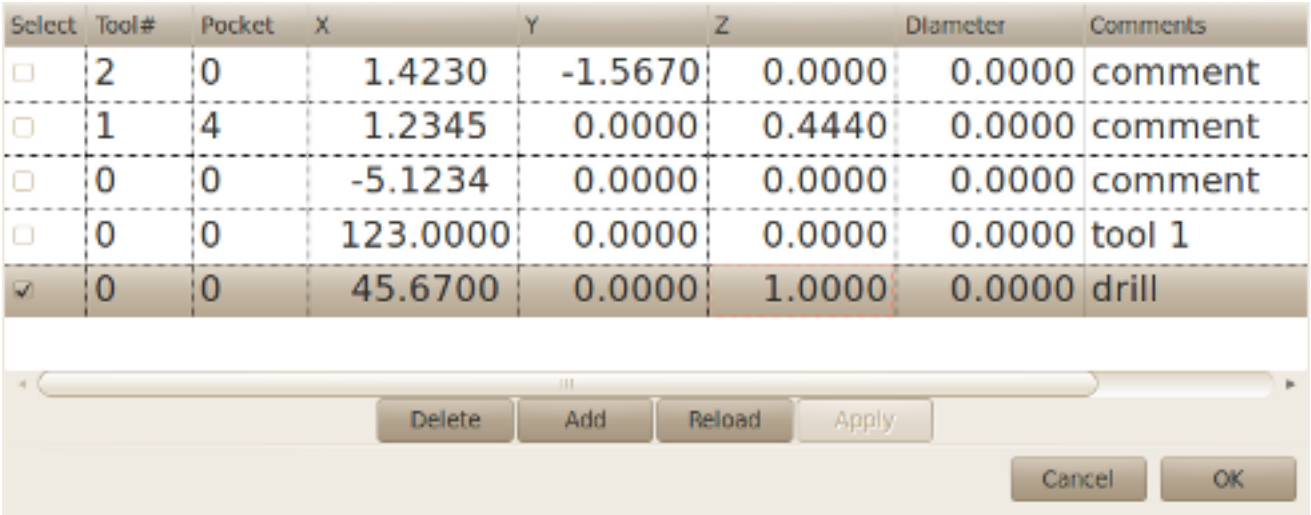
**Hidden Columns**

This will hide the given columns: The columns are designated (in order) as such:
s,t,p,x,y,z,a,b,c,u,v,w,d,i,j,q,;
You can hide any number of columns including the select and comments

**Direct program control**

There a couple ways to directly control the widget using Python.

```
using goobject to set the above listed properties:
    [widget name].set_properties('hide_columns','uvwijq')
        This would hide the uvwij and q columns and show all others.

There are python methods:
    [widget name].set_visible("ijq",False)
        Would hide ij and Q columns and leave the rest as they were.
    [widget name].set_filename(path_to_file)
        Sets and loads the tool file.
    [widget name].reload(None)
        Reloads the current toolfile
```




#### 7.3.6.26 Offsetpage

The Offsetpage widget is used to display/edit the offsets of all the axes.
It has convenience buttons for zeroing G92 and Rotation-Around-Z offsets.
It will only allow you to select the edit mode when the machine is on and idle.

You can directly edit the offsets in the table at this time. Unselect the edit button to allow the OffsetPage to reflect changes.

It has the following properties:

**Hidden Columns**
A no-space list of columns to hide: The columns are designated (in order) as such:
xyzabcuvwt
You can hide any of the columns.

**Hidden Rows**
A no-space list of rows to hide: the rows are designated (in order) as such
0123456789abc
You can hide any of the rows.

**Pango Font**
Sets text font type and size

**HighLight color**
when editing this is the high light color

**Active color**
when OffsetPage detects an active user coordinate system it will use this color for the text

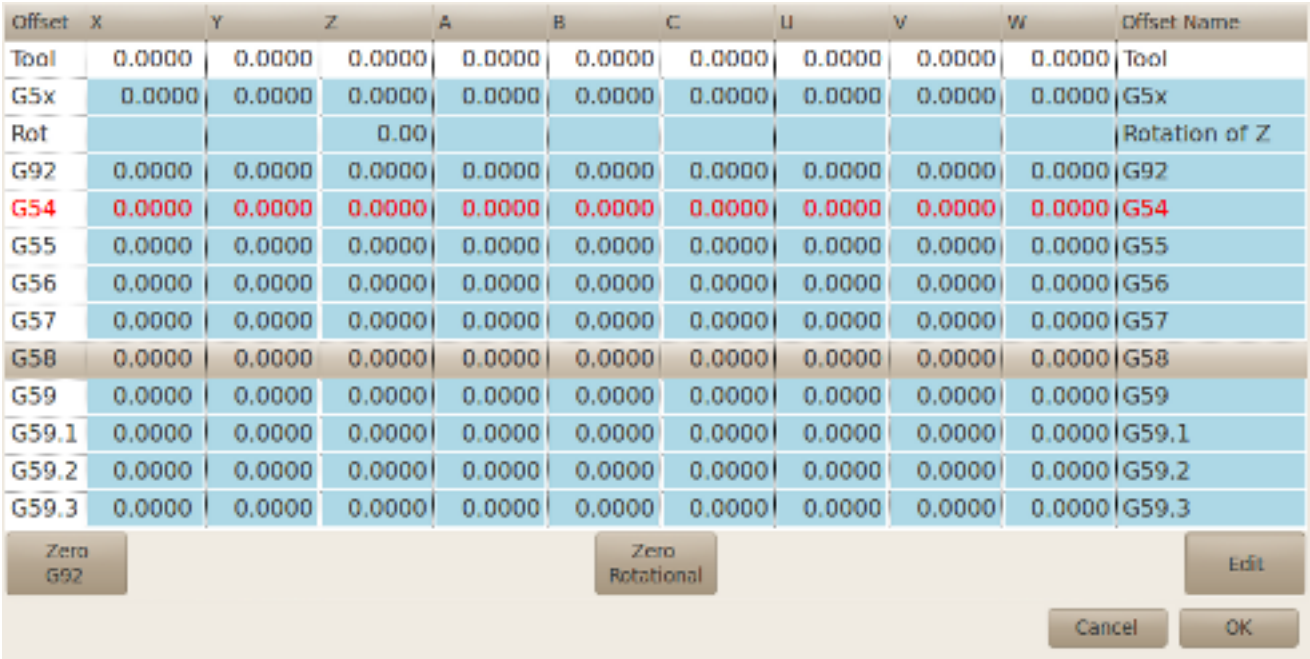
**Text template for metric units**
You can use python formatting to display the position with different precision.

**Text template for imperial units**
You can use python formatting to display the position with different precision.

**Direct program control**
There a couple ways to directly control the widget using Python.

```
Using goobject to set the above listed properties:
[widget name].set_property("highlight_color",gtk.gdk.Color('blue'))
[widget name].set_property("foreground_color",gtk.gdk.Color('black'))
[widget name].set_property("hide_columns","xyzabcuvwt")
[widget name].set_property("hide_rows","123456789abc")
[widget name].set_property("font","sans 25")

There are python methods to control the widget:
[widget name].set_filename("../../../configs/sim/gscreen/gscreen_custom/sim.var")
[widget name].set_col_visible("Yabuvw",False)
[widget name].set_row_visible("456789abc",False)
[widget name].set_to_mm()
[widget name].set_to_inch()
[widget name].hide_button_box(True)
[widget name].set_font("sans 20")
[widget name].set_highlight_color("violet")
[widget name].set_foreground_color("yellow")
[widget name].mark_active("G55")
    Allows you to directly set a row to highlight.
    (eg in case you wish to use your own navigation controls.
    See <<cha:gmoccapy,Gmoccapy Chapter>>
[widget name].selection_mask = ("Tool","Rot","G5x")
    These rows are NOT selectable in edit mode.
[widget name].set_names([['G54','Default'],["G55","Vice1"],['Rot','Rotational']])
    This allows you to set the text of the 'T' column of each/any row.
```

```
    This is a list of a list of offset-name/user-name pairs.
    The default text is the same as the offset name.
[widget name].get_names()
    This returns a list of a list of row-keyword/user-name pairs.
    The user name column is editable, so saving this list is user friendly.
    see set_names above.
```




#### 7.3.6.27 HAL_sourceview widget

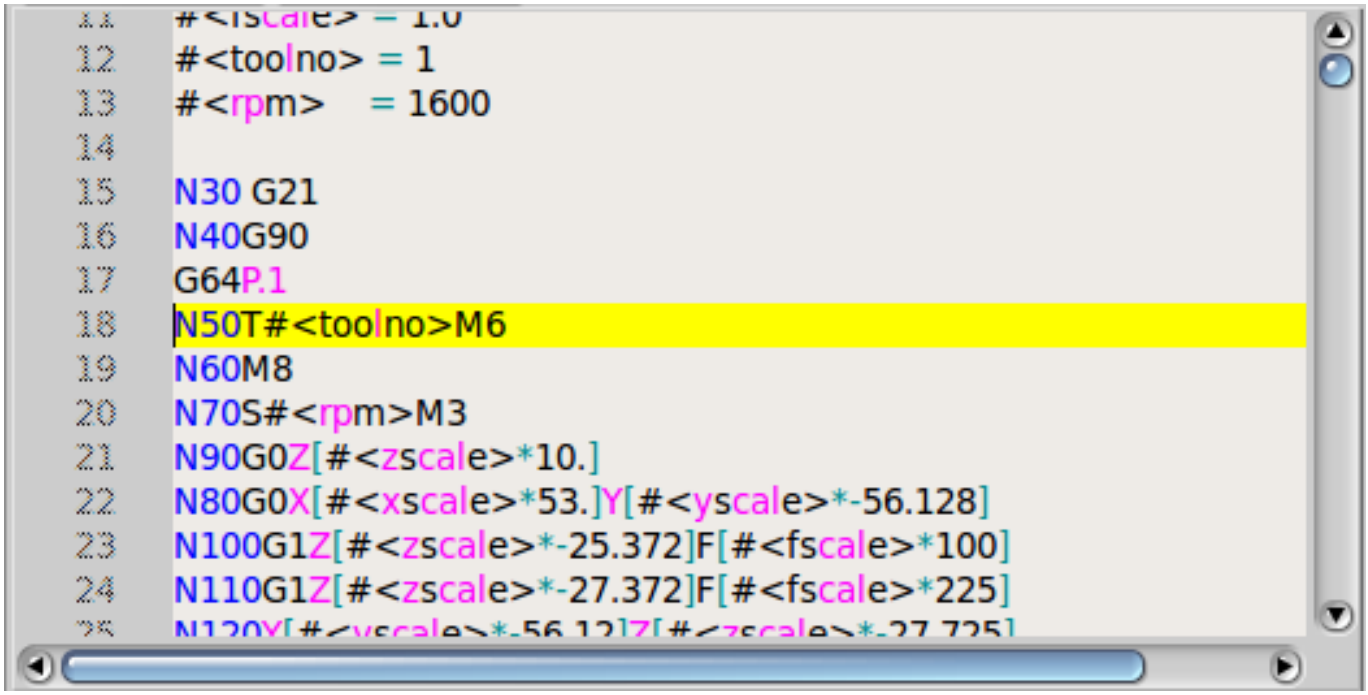
This is for displaying and simple editing of Gcode.
It looks for .ngc highlight specs in ~/share/gtksourceview-2.0/language-specs/ The current running line will be highlighted.
With external python glue code:
*It can search for text, undo and redo changes.
*It can be used for program line selection.

**Direct program control**

There are python methods to control the widget:

```
[widget name].redo()
    redo one level of changes.
[widget name].undo()
    undo one level of changes
[widget name].text_search(direction=True,mixed_case=True,text='G92')
    Searches forward (direction = True) or back, +
    Searches with mixed case (mixed_case = True) or exact match
[widget name].set_line_number(linenumber)
    Sets the line to high light. Uses the sourceview line numbers.
[widget name].get_line_number()
    returns the currently high lighted line.
[widget name].line_up()
    Moves the High lighted line up one line
[widget name].line_down()
    Moves the High lighted line down one line
```

```
[widget name].load_file('filename')
    loads a file. Using None (not a filename string) will reload the same program.
[widget name].get_filename()
```



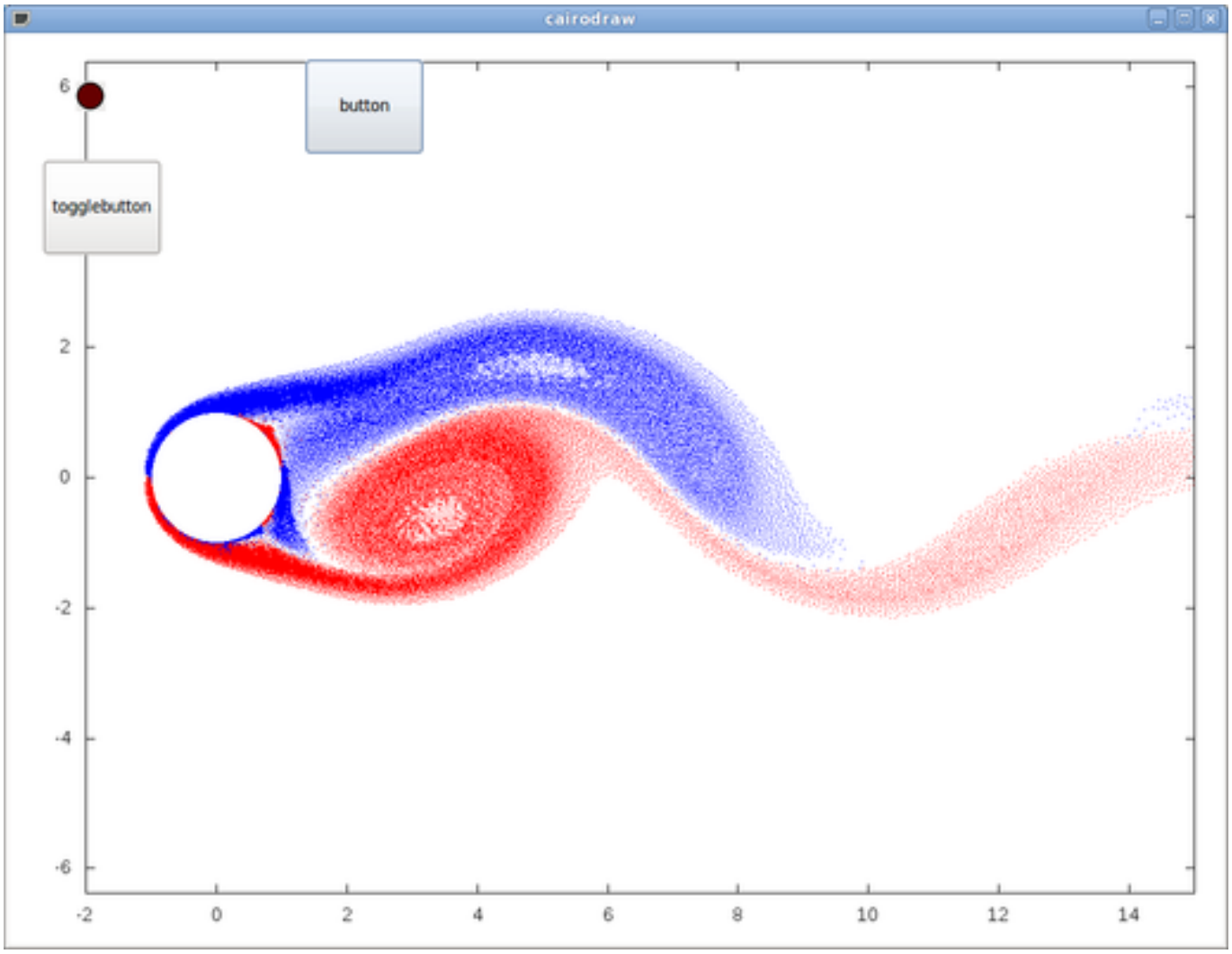

#### 7.3.6.28 MDI history

This is for displaying and entering MDI codes.
It will automatically gray out when MDI is not available.
Eg during Estop and program running.

#### 7.3.6.29 Animated function diagrams: HAL widgets in a bitmap

For some applications it might be desirable to have background image - like a functional diagram - and position widgets at appropriate places in that diagram. A good combination is setting a bitmap background image, like from a .png file, making the gladevcp window fixed-size, and use the glade Fixed widget to position widgets on this image.

The code for the below example can be found in `configs/apps/gladevcp/animated-backdrop`:



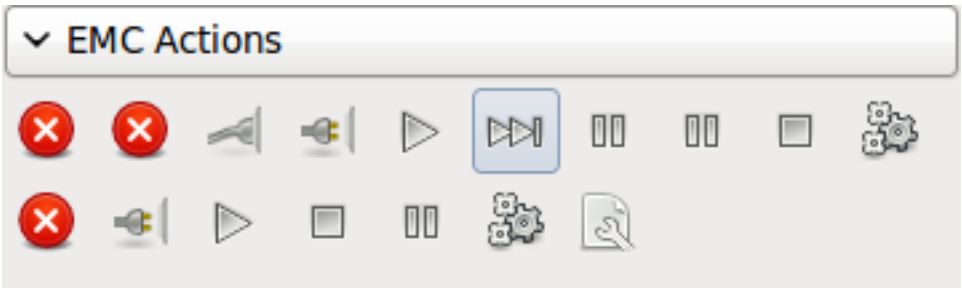

### 7.3.7 Action Widgets reference

GladeVcp includes a collection of "canned actions" called VCP Action Widgets for the Glade user interface editor. Other than HAL widgets, which interact with HAL pins, VCP Actions interact with LinuxCNC and the G-code interpreter.

VCP Action Widgets are derived from the Gtk.Action widget. The Action widget in a nutshell:

- it is an object available in Glade
- it has no visual appearance by itself
- it's purpose: associate a visible, sensitive UI component like menu, toolbutton, button with a command. See these widget's *General→Related Action* property.
- the "canned action" will be executed when the associated UI component is triggered (button press, menu click..)
- it provides an easy way to execute commands without resorting to Python programming.

The appearance of VCP Actions in Glade is roughly as follows:




Tooltip hovers provide a description.

#### 7.3.7.1 VCP Action widgets

VCP Action widgets are one-shot type widgets. They implement a single action and are for use in simple buttons, menu entries or radio/check groups.

#### 7.3.7.2 VCP ToggleAction widgets

These are bi-modal widgets. They implement two actions or use a second (usually pressed) state to indicate that currently an action is running. Toggle actions are aimed for use in ToggleButtons, ToggleToolButtons or toggling menu items. A simplex example is the ESTOP toggle button.

Currently the following widgets are available:

- The ESTOP toggle sends ESTOP or ESTOP_RESET commands to LinuxCNC depending on it's state.
- The ON/OFF toggle sends STATE_ON and STATE_OFF commands.
- Pause/Resume sends AUTO_PAUSE or AUTO_RESUME commands.

The following toggle actions have only one associated command and use the *pressed* state to indicate that the requested operation is running:

- The Run toggle sends an AUTO_RUN command and waits in the pressed state until the interpreter is idle again.
- The Stop toggle is inactive until the interpreter enters the active state (is running G-code) and then allows user to send AUTO_ABORT command.
- The MDI toggle sends given MDI command and waits for its completion in *pressed* inactive state.

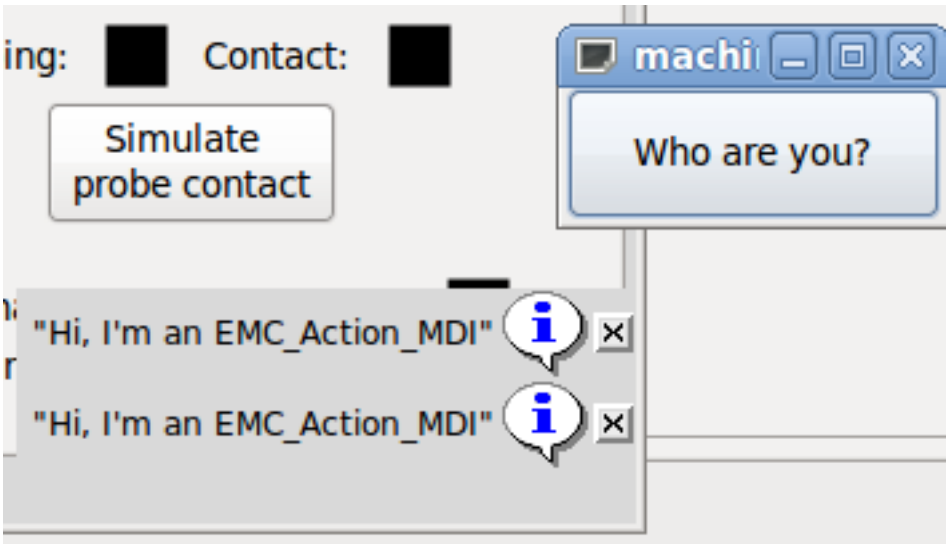
#### 7.3.7.3 The Action_MDI Toggle and Action_MDI widgets

These widgets provide a means to execute arbitrary MDI commands. The Action_MDI widget does not wait for command completion as the Action_MDI Toggle does, which remains disabled until command complete.

#### 7.3.7.4 A simple example: Execute MDI command on button press

`configs/apps/gladevcp/mdi-command-example/whoareyou.ui` is a Glade UI file which conveys the basics:

Open it in Glade and study how it's done. Start Axis, and then start this from a terminal window with `gladevcp whoareyou.ui`. See the `hal_action_mdi1` Action and it's `MDI command` property - this just executes `(MSG, "Hi, I'm an VCP_Action_M` so there should be a message popup in Axis like so:



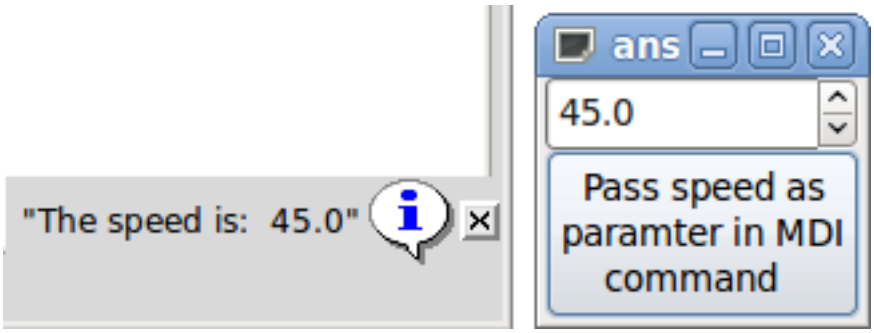

You'll notice that the button associated with the Action_MDI action is grayed out if the machine is off, in E-Stop or the interpreter is running. It will automatically become active when the machine is turned on and out of E-Stop, and the program is idle.

#### 7.3.7.5 Parameter passing with Action_MDI and ToggleAction_MDI widgets

Optionally, *MDI command* strings may have parameters substituted before they are passed to the interpreter. Parameters currently may be names of HAL pins in the GladeVCP component. This is how it works:

- assume you have a *HAL SpinBox* named `speed`, and you want to pass it's current value as a parameter in an MDI command.
- The HAL SpinBox will have a float-type HAL pin named speed-f (see HalWidgets description).
- To substitute this value in the MDI command, insert the HAL pin name enclosed like so: `${pin-name}`
- for the above HAL SpinBox, we could use `(MSG, "The speed is: ${speed-f}")` just to show what's happening.

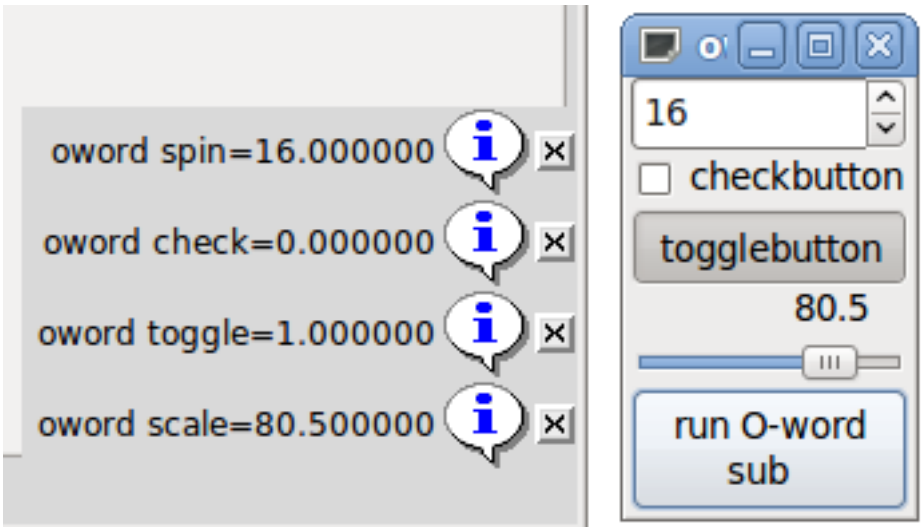
The example UI file is `configs/apps/gladevcp/mdi-command-example/speed.ui`. Here's what you get when running it:




#### 7.3.7.6 An advanced example: Feeding parameters to an O-word subroutine

It's perfectly OK to call an O-word subroutine in an MDI command, and pass HAL pin values as actual parameters. An example UI file is in `configs/apps/gladevcp/mdi-command-example/owordsub.ui`.

Place `nc_files/gladevcp_lib/oword.ngc` so Axis can find it, and run `gladevcp owordsub.ui` from a terminal window. This looks like so:




#### 7.3.7.7 Preparing for an MDI Action, and cleaning up afterwards

The LinuxCNC G-Code interpreter has a single global set of variables, like feed, spindle speed, relative/absolute mode and others. If you use G code commands or O-word subs, some of these variables might get changed by the command or subroutine - for example, a probing subroutine will very likely set the feed value quite low. With no further precautions, your previous feed setting will be overwritten by the probing subroutine's value.

To deal with this surprising and undesirable side effect of a given O-word subroutine or G-code statement executed with an LinuxCNC ToggleAction_MDI, you might associate pre-MDI and post-MDI handlers with a given LinuxCNC ToggleAction_MDI. These handlers are optional and provide a way to save any state before executing the MDI Action, and to restore it to previous values afterwards. The signal names are `mdi-command-start` and `mdi-command-stop`; the handler names can be set in Glade like any other handler.

Here's an example how a feed value might be saved and restored by such handlers (note that LinuxCNC command and status channels are available as `self.linuxcnc` and `self.stat` through the VCP_ActionBase class:

```
    def on_mdi_command_start(self, action, userdata=None):
        action.stat.poll()
        self.start_feed = action.stat.settings[1]

    def on_mdi_command_stop(self, action, userdata=None):
        action.linuxcnc.mdi('F%.1f' % (self.start_feed))
        while action.linuxcnc.wait_complete() == -1:
            pass
```

Only the Action_MDI Toggle widget supports these signals.

**Note**
In a later release of LinuxCNC, the new M-codes M70-M72 are available which make it saving state before a subroutine call, and restoring state on return much easier.

#### 7.3.7.8 Using the LinuxCNC Stat object to deal with status changes

Many actions depend on LinuxCNC status - is it in manual, MDI or auto mode? is a program running, paused or idle? You cannot start an MDI command while a G-code program is running, so this needs to be taken care of. Many LinuxCNC actions take care of this themselves, and related buttons and menu entries are deactivated when the operation is currently impossible.

When using Python event handlers - which are at a lower level than Actions - one needs to take care of dealing with status dependencies oneself. For this purpose, there's the LinuxCNC Stat widget: to associate LinuxCNC status changes with event handlers.

LinuxCNC Stat has no visible component - you just add it to your UI with Glade. Once added, you can associate handlers with its following signals:

- state-related: emitted when E-Stop condition occurs, is reset, machine is turned on, or is turned off
  - `state-estop`
  - `state-estop-reset`
  - `state-on`,
  - `state-off`
- mode-related: emitted when LinuxCNC enters that particular mode
  - `mode-manual`
  - `mode-mdi`
  - `mode-auto`
- interpreter-related: emitted when the G-code interpreter changes into that mode
  - `interp-run`
  - `interp-idle`
  - `interp-paused`
  - `interp-reading`
  - `interp-waiting`
  - `file-loaded`
  - `line-changed`
- homing-related: emitted when linuxcnc is homed or not
  - `all-homed`
  - `not-all-homed`

### 7.3.8 GladeVCP Programming

#### 7.3.8.1 User Defined Actions

Most widget sets, and their associated user interface editors, support the concept of callbacks - functions in user-written code which are executed when *something happens* in the UI - events like mouse clicks, characters typed, mouse movement, timer events, window hiding and exposure and so forth.

HAL output widgets typically map input-type events like a button press to a value change of the associated HAL pin by means of such a - predefined - callback. Within PyVCP, this is really the only type of event handling supported - doing something more complex, like executing MDI commands to call a G-code subroutine, is not supported.

Within GladeVCP, HAL pin changes are just one type of the general class of events (called signals) in GTK+. Most widgets may originate such signals, and the Glade editor supports associating such a signal with a Python method or function name.

If you decide to use user-defined actions, your job is to write a Python module whose class methods - or in the simple case, just functions - can be referred to in Glade as event handlers. GladeVCP provides a way to import your module(s) at startup and will automatically link your event handlers with the widget signals as set in the Glade UI description.

#### 7.3.8.2 An example: adding custom user callbacks in Python

This is just a minimal example to convey the idea - details are laid out in the rest of this section.

GladeVCP can not only manipulate or display HAL pins, you can also write regular event handlers in Python. This could be used, among others, to execute MDI commands. Here's how you do it:

Write a Python module like so and save as e.g. handlers.py:

```
nhits = 0
def on_button_press(gtkobj,data=None):
    global nhits nhits += 1 gtkobj.set_label("hits: %d" % nhits)
```

In Glade, define a button or HAL button, select the *Signals* tab, and in the GtkButton properties select the *pressed* line. Enter *on_button_press* there, and save the Glade file.

Then add the option *-u handlers.py* to the gladevcp command line. If your event handlers are spread over several files, just add multiple *-u <pyfilename>* options.

Now, pressing the button should change its label since it's set in the callback function.

What the `-u` flag does: all Python functions in this file are collected and setup as potential callback handlers for your Gtk widgets - they can be referenced from Glade *Signals* tabs. The callback handlers are called with the particular object instance as parameter, like the GtkButton instance above, so you can apply any GtkButton method from there.

Or do some more useful stuff, like calling an MDI command!

#### 7.3.8.3 HAL value change events

HAL input widgets, like a LED, automatically associate their HAL pin state (on/off) with the optical appearance of the widget (LED lit/dark).

Beyond this built-in functionality, one may associate a change callback with any HAL pin, including those of predefined HAL widgets. This fits nicely with the event-driven structure of a typical widget application: every activity, be it mouse click, key, timer expired, or the change of a HAL pin's value, generates a callback and is handled by the same orthogonal mechanism.

For user-defined HAL pins not associated with a particular HAL widget, the signal name is *value-changed*. See the Adding HAL pins section below for details.

HAL widgets come with a pre-defined signal called *hal-pin-changed*. See the Hal Widgets section for details.

#### 7.3.8.4 Programming model

The overall approach is as follows:

- design your UI with Glade, and set signal handlers where you want actions associated with a widget
- write a Python module which contains callable objects (see *handler models* below)
- pass your module's path name to gladevcp with the *-u <module>* option
- gladevcp imports the module, inspects it for signal handlers and connects them to the widget tree
- the main event loop is run.

**The simple handler model** For simple tasks it's sufficient to define functions named after the Glade signal handlers. These will be called when the corresponding event happens in the widget tree. Here's a trivial example - it assumes that the *pressed* signal of a Gtk Button or HAL Button is linked to a callback called *on_button_press*:

```
nhits = 0
def on_button_press(gtkobj,data=None):
    global nhits
    nhits += 1
    gtkobj.set_label("hits: %d" % nhits)
```

Add this function to a Python file and run as follows:

```
gladevcp -u <myhandler>.py mygui.ui
```

Note communication between handlers has to go through global variables, which does not scale well and is positively unpythonic. This is why we came up with the class-based handler model.

**The class-based handler model** The idea here is: handlers are linked to class methods. The underlying class(es) are instantiated and inspected during GladeVCP startup and linked to the widget tree as signal handlers. So the task now is to write:

- one or more several class definition(s) with one or several methods, in one module or split over several modules,
- a function *get_handlers* in each module which will return a list of class instances to GladeVCP - their method names will be linked to signal handlers

Here is a minimum user-defined handler example module:

```
class MyCallbacks :
    def on_this_signal(self,obj,data=None):
        print "this_signal happened, obj=",obj

def get_handlers(halcomp,builder,useropts):
    return [MyCallbacks ()]
```

Now, *on_this_signal* will be available as signal handler to your widget tree.

**The get_handlers protocol** If during module inspection GladeVCP finds a function `get_handlers`, it calls it as follows:

```
get_handlers(halcomp,builder,useropts)
```

the arguments are:

- halcomp - refers to the HAL component under construction
- builder - widget tree - result of reading the UI definition (either referring to a GtkBuilder or libglade-type object)
- useropts - a list of strings collected from the gladevcp command line `-U <useropts>` option

GladeVCP then inspects the list of class instances and retrieves their method names. Qualifying method names are connected to the widget tree as signal handlers. Only method names which do not begin with an _ (underscore) are considered.

Note that regardless whether you're using the libglade or the new GtkBuilder format for your Glade UI, widgets can always be referred to as `builder.get_object(<widgetname>)`. Also, the complete list of widgets is available as `builder.get_object` regardless of UI format.

#### 7.3.8.5 Initialization sequence

It is important to know in which state of affairs your `get_handlers()` function is called so you know what is safe to do there and what not. First, modules are imported and initialized in command line order. After successful import, `get_handlers()` is called in the following state:

- the widget tree is created, but not yet realized (no toplevel `window.show()` has been executed yet)
- the halcomp HAL component is set up and all HAL widget's pins have already been added to it
- it is safe to add more HAL pins because `halcomp.ready()` has not yet been called at this point, so you may add your own pins, for instance in the class `__init__()` method.

Once all modules have been imported and method names extracted, the following steps happen:

- all qualifying method names will be connected to the widget tree with `connect_signals()/signal_autoconnect()` (depending on the type of UI imported - GtkBuilder vs the old libglade format).
- the HAL component is finalized with halcomp.ready()
- if a window ID was passed as argument, the widget tree is re-parented to run in this window, and Glade's toplevel window1 is abandoned (see FAQ)
- if a HAL command file was passed with `-H halfile`, it is executed with halcmd
- the Gtk main loop is run.

So when your handler class is initialized, all widgets are existent but not yet realized (displayed on screen). And the HAL component isn't ready as well, so its unsafe to access pins values in your `__init__()` method.

If you want to have a callback to execute at program start after it is safe to access HAL pins, then a connect a handler to the realize signal of the top level window1 (which might be its only real purpose). At this point GladeVCP is done with all setup tasks, the halfile has been run, and GladeVCP is about to enter the Gtk main loop.

#### 7.3.8.6 Multiple callbacks with the same name

Within a class, method names must be unique. However, it is OK to have multiple class instances passed to GladeVCP by get_handlers() with identically named methods. When the corresponding signal occurs, these methods will be called in definition order - module by module, and within a module, in the order class instances are returned by `get_handlers()`.

#### 7.3.8.7 The GladeVCP `-U <useropts>` flag

Instead of extending GladeVCP for any conceivable option which could potentially be useful for a handler class, you may use the -U <useroption> flag (repeatedly if you wish). This flag collects a list of <useroption> strings. This list is passed to the get_handlers() function (useropts argument). Your code is free to interpret these strings as you see fit. An possible usage would be to pass them to the Python exec function in your `get_handlers()` as follows:

```
debug = 0
...
def get_handlers(halcomp,builder,useropts):
    ...
    global debug # assuming there's a global var
    for cmd in useropts:
        exec cmd in globals()
```

This way you can pass arbitrary Python statements to your module through the gladevcp -U option, for example:

```
gladevcp -U debug=42 -U "print 'debug=%d' % debug" ...
```

This should set debug to 2 and confirm that your module actually did it.

#### 7.3.8.8 Persistent variables in GladeVCP

A annoying aspect of GladeVCP in its earlier form and pyvcp is the fact that you may change values and HAL pins through text entry, sliders, spin boxes, toggle buttons etc, but their settings are not saved and restored at the next run of LinuxCNC - they start at the default value as set in the panel or widget definition.

GladeVCP has an easy-to-use mechanism to save and restore the state of HAL widgets, and program variables (in fact any instance attribute of type int, float, bool or string).

This mechanism uses the popular *.ini* file format to save and reload persistent attributes.

**Persistence, program versions and the signature check** Imagine renaming, adding or deleting widgets in Glade: an .ini file lying around from a previous program version, or an entirely different user interface, would be not be able to restore the state properly since variables and types might have changed.

GladeVCP detects this situation by a signature which depends on all object names and types which are saved and to be restored. In the case of signature mismatch, a new .ini file with default settings is generated.

#### 7.3.8.9 Using persistent variables

If you want any of Gtk widget state, HAL widgets output pin's values and/or class attributes of your handler class to be retained across invocations, proceed as follows:

- import the `gladevcp.persistence` module
- decide which instance attributes, and their default values you want to have retained, if any
- decide which widgets should have their state retained
- describe these decisions in your handler class' `init()` method through a nested dictionary as follows:

```
def __init__(self, halcomp,builder,useropts):
    self.halcomp = halcomp
    self.builder = builder
    self.useropts = useropts
    self.defaults = {
        # the following names will be saved/restored as method attributes
        # the save/restore mechanism is strongly typed - the variables type will be derived ↩
            from the type of the
        # initialization value. Currently supported types are: int, float, bool, string
        IniFile.vars : { 'nhits' : 0, 'a': 1.67, 'd': True ,'c' : "a string"},
        # to save/restore all widget's state which might remotely make sense, add this:
        IniFile.widgets : widget_defaults(builder.get_objects())
        # a sensible alternative might be to retain only all HAL output widgets' state:
        # IniFile.widgets: widget_defaults(select_widgets(self.builder.get_objects(),  ↩
            hal_only=True,output_only = True)),
    }
```

Then associate an .ini file with this descriptor:

```
self.ini_filename = __name__ + '.ini'
self.ini = IniFile(self.ini_filename,self.defaults,self.builder)
self.ini.restore_state(self)
```

After `restore_state()`, self will have attributes set if as running the following:

```
self.nhits = 0
self.a = 1.67
self.d = True
self.c = "a string"
```

Note that types are saved and preserved on restore. This example assumes that the ini file didn't exist or had the default values from self.defaults.

After this incantation, you can use the following IniFil methods:

**ini.save_state(obj)**
: saves objs's attributes as per IniFil.vars dictionary and the widget state as described in IniFile.widgets in self.defaults

**ini.create_default_ini()**
: create a .ini file with default values

**ini.restore_state(obj)**
: restore HAL out pins and obj's attributes as saved/initialized to default as above

#### 7.3.8.10 Saving the state on Gladvcp shutdown

To save the widget and/or variable state on exit, proceed as follows:

- select some interior widget (type is not important, for instance a table).
- in the *Signals* tab, select *GtkObject*. It should show a *destroy* signal in the first column.
- add the handler name, e.g. *on_destroy* to the second column.
- add a Python handler like below:

```
import gtk
...
def on_destroy(self,obj,data=None):
    self.ini.save_state(self)
```

This will save state and shutdown GladeVCP properly, regardless whether the panel is embedded in Axis, or a standalone window.

**Caution**
Do not use `window1` (the toplevel window) to connect a `destroy` event. Due to the way a GladeVCP panel interacts with Axis if a panel is embedded within Axis, **window1 will not receive destroy events properly**. However, since on shutdown all widgets are destroyed, anyone will do. Recommended: use a second-level widget - for instance, if you have a table container in your panel, use that.

Next time you start the GladeVCP application, the widgets should come up in the state when the application was closed.

**Caution**
The *GtkWidget* line has a similarly sounding *destroy-event* - **dont use that to connect to the *on_destroy* handler, it wont work** - make sure you use the *destroy* event from the *GtkObject* line.

#### 7.3.8.11 Saving state when Ctrl-C is pressed

By default, the reaction of GladeVCP to a Ctrl-C event is to just exit - `without` saving state. To make sure that this case is covered, add a handler call `on_unix_signal` which will be automatically be called on Ctrl-C (actuall on the SIGINT and SIGTERM signals). Example

```
def on_unix_signal(self,signum,stack_frame):
    print "on_unix_signal(): signal %d received, saving state" % (signum)
    self.ini.save_state(self)
```

#### 7.3.8.12 Hand-editing .ini files

You can do that, but note that the values in self.defaults override your edits if there is a syntax or type error in your edit. The error is detected, a console message will hint about that happened, and the bad inifile will be renamed to have the .BAD suffix. Subsequent bad ini files overwrite earlier .BAD files.

#### 7.3.8.13 Adding HAL pins

If you need HAL pins which are not associated with a specific HAL widget, add them as follows:

```
import hal_glib
...
# in your handler class __init__():
self.example_trigger = hal_glib.GPin(halcomp.newpin('example-trigger', hal.HAL_BIT, hal. ↩
    HAL_IN))
```

To get a callback when this pin's value changes, associate a `value-change` callback with this pin, add:

```
self.example_trigger.connect('value-changed', self._on_example_trigger_change)
```

and define a callback method (or function, in this case leave out the `self` parameter):

```
# note '_' - this method will not be visible to the widget tree
def _on_example_trigger_change(self,pin,userdata=None):
    print "pin value changed to:" % (pin.get())
```

#### 7.3.8.14 Adding timers

Since GladeVCP uses Gtk widgets which rely on the GObject base class, the full glib functionally is available. Here is an example for a timer callback:

```
def _on_timer_tick(self,userdata=None):
    ...
    return True # to restart the timer; return False for on-shot
...
# demonstrate a slow background timer - granularity is one second
# for a faster timer (granularity 1 ms), use this:
# glib.timeout_add(100, self._on_timer_tick,userdata) # 10Hz
glib.timeout_add_seconds(1, self._on_timer_tick)
```

#### 7.3.8.15 Setting HAL widget properties programmatically

With glade, widget properties are typically set fixed while editing. You can, however, set widget properties at runtime, for instance from ini file values, which would typically be done in the handler initialization code. Setting properties from HAL pin values is possible, too.

In the following example (assuming a HAL Meter widget called `meter`), the meter's min value is set from an INI file parameter at startup, and the max value is set via a HAL pin, which causes the widget's scale to readjust dynamically:

```
import linuxcnc
import os
import hal
import hal_glib

class HandlerClass:

    def _on_max_value_change(self,hal_pin,data=None):
        self.meter.max = float(hal_pin.get())
        self.meter.queue_draw() # force a widget redraw

    def __init__(self, halcomp,builder,useropts):
  self.builder = builder

        # hal pin with change callback.
        # When the pin's value changes the callback is executed.
```

```
        self.max_value = hal_glib.GPin(halcomp.newpin('max-value',  hal.HAL_FLOAT, hal. ↩
            HAL_IN))
        self.max_value.connect('value-changed', self._on_max_value_change)

  inifile = linuxcnc.ini(os.getenv("INI_FILE_NAME"))
  mmin = float(inifile.find("METER", "MIN") or 0.0)
        self.meter = self.builder.get_object('meter')
        self.meter.min = mmin


def get_handlers(halcomp,builder,useropts):
    return [HandlerClass(halcomp,builder,useropts)]
```

#### 7.3.8.16 Examples, and rolling your own GladeVCP application

Visit `linuxcnc_root_directory/configs/apps/gladevcp` for running examples and starters for your own projects.

### 7.3.9 FAQ

1. *I get an unexpected unmap event in my handler function right after startup. What's this?*

   This is a consequence of your Glade UI file having the window1 Visible property set to True, together with re-parenting the GladeVCP window into Axis or touchy. The GladeVCP widget tree is created, including a top level window, and then *reparented into Axis*, leaving that toplevel window laying around orphaned. To avoid having this useless empty window hanging around, it is unmapped (made invisible), which is the cause of the unmap signal you get. Suggested fix: set window1.visible to False, and ignore an initial unmap event.

2. *My GladeVCP program starts, but no window appears where I expect it to be?*

   The window Axis allocates for GladeVCP will obtain the *natural size* of all its child widgets combined. It's the child widget's job to request a size (width and/or height). However, not all widgets do request a width greater than 0, for instance the Graph widget in its current form. If there's such a widget in your Glade file and it's the one which defines the layout you might want to set its width explicitly. Note that setting the window1 width and height properties in Glade does not make sense because this window will be orphaned during re-parenting and hence its geometry will have no impact on layout (see above). The general rule is: if you manually run a UI file with *gladevcp <uifile>* and its window has reasonable geometry, it should come up in Axis properly as well.

3. *I want a blinking LED, but it wont blink*

   I ticked the checkbutton to let it blink with 100 msec interval. It wont blink, and I get a startup warning: Warning: value "0" of type 'gint' is invalid or out of range for property 'led-blink-rate' of type 'gint'? This seems to be a glade bug. Just type over the blink rate field, and save again - this works for me.

4. *My gladevcp panel in Axis doesn't save state when I close Axis, although I defined an on_destroy handler linked to the window destroy signal*

   Very likely this handler is linked to window1, which due to reparenting isn't usable for this purpose. Please link the on_destroy handler to the destroy signal of an interior window. For instance, I have a notebook inside window1, and linked on_destroy to the notebooks destroy signal, and that works fine. It doesn't work for window1.

5. *I want to set the background color or text of a HAL_Label widget depending on its HAL pin value*

   See the example in configs/apps/gladevcp/colored-label. Setting the background color of a GtkLabel widget (and HAL_Label is derived from GtkLabel) is a bit tricky. The GtkLabel widget has no window object of its own for performance reasons, and only window objects can have a background color. The solution is to enclose the Label in an EventBox container, which has a window but is otherwise invisible - see the coloredlabel.ui file.

6. *I defined a `hal_spinbutton` widget in glade, and set a default `value` property in the corresponding adjustment. It comes up with zero?*

this is due to a bug in the old Gtk version distributed with Ubuntu 8.04 and 10.04, and is likely to be the case for all widgets using adjustment. The workaround mentione for instance in http://osdir.com/ml/gtk-app-devel-list/2010-04/-msg00129.html does not reliably set the HAL pin value, it is better to set it explicitly in an `on_realize` signal handler during widget creation. See the example in `configs/apps/gladevcp/by-widget/spinbutton.{ui,py}`.

### 7.3.10 Troubleshooting

- make sure you have the development version of LinuxCNC installed. You don't need the axisrc file any more, this was mentioned in the old GladeVcp wiki page.
- run GladeVCP or Axis from a terminal window. If you get Python errors, check whether there's still a `/usr/lib/python2.6/dis` file lying around besides the newer `/usr/lib/python2.6/dist-packages/_hal.so` (note underscore); if yes, remove the `hal.so` file. It has been superseded by hal.py in the same directory and confuses the import mechanism.
- if you're using run-in-place, do a *make clean* to remove any accidentally left over hal.so file, then *make*.
- if you're using *HAL_table* or *HAL_HBox* widgets, be aware they have an HAL pin associated with it which is off by default. This pin controls whether these container's children are active or not.

### 7.3.11 Implementation note: Key handling in Axis

We believe key handling works OK, but since it is new code, we're telling about it you so you can watch out for problems; please let us know of errors or odd behavior. This is the story:

Axis uses the TkInter widget set. GladeVCP applications use Gtk widgets and run in a separate process context. They are hooked into Axis with the Xembed protocol. This allows a child application like GladeVCP to properly fit in a parent's window, and - in theory - have integrated event handling.

However, this assumes that both parent and child application properly support the Xembed protocol, which Gtk does, but TkInter doesn't. A consequence of this is that certain keys would not be forwarded from a GladeVCP panel to Axis properly under all circumstances. One of these situations was the case when an Entry, or SpinButton widget had focus: in this case, for instance an Escape key would not have been forwarded to Axis and cause an abort as it should, with potentially disastrous consequences.

Therefore, key events in GladeVCP are explicitly handled, and selectively forwarded to Axis, to assure that such situations cannot arise. For details, see the `keyboard_forward()` function in `lib/python/gladevcp/xembed.py`.

### 7.3.12 Adding Custom Widgets

The LinuxCNC Wiki has information on adding custom widgets to GladeVCP. GladeVCP Custom Widgets

### 7.3.13 Auxiliary Gladevcp Applications

Support is provided for independently installed gladevcp applications that conform to system directory placements as defined by the LINUXCNC_AUX_GLADEVCP and LINUXCNC_AUX_EXAMPLES items reported by the script linuxcnc_var:

```
$ linuxcnc_var LINUXCNC_AUX_GLADEVCP
/usr/share/linuxcnc/aux_gladevcp
$ linuxcnc_var LINUXCNC_AUX_EXAMPLES
/usr/share/linuxcnc/aux_examples
```

The system directory defined by LINUXCNC_AUX_GLADEVCP (/usr/share/linuxcnc/aux_gladevcp) specifies the location for a gladevcp-compatible python file(s) and related subdirectories. The python file is imported at gladevcp startup and made available to subsequent gladevcp applications including embedded usage in supporting guis.

The system directory defined by LINUXCNC_AUX_EXAMPLES (/usr/share/linuxcnc/aux_examples) specifies the location of example configuration subdirectories used for auxiliary applications. See the getting-started/running-linuxcnc section for *Adding Configuration Selection Items*.

For testing, a runtime specification of auxiliary applications may be specified using the exported environmental variable: GLADEVCP_EXTRAS. This variable should be a path list of one or more configuration directories separated by a (:). Typically, this variable would be set in a shell starting linuxcnc or in a user's ~/.profile startup script. Example:

```
export GLADEVCP_EXTRAS=~/mygladevcp:/opt/othergladevcp
```

Files found in directories specified with the environmental variable GLADEVCP_EXTRAS supersede identically-named files within subdirectories of the system directory specified by LINUXNC_AUX_GLADEVCP (e.g., /usr/share/linuxcnc/aux_gladevcp). This provision allows a developer to test an application by exporting GLADEVCP_EXTRAS to specify a private application directory without removing a system-installed application directory. Messages inidicating rejected duplicates are printed to stdout.

**Note**
Support for auxiliary gladevcp applications requires a python module named *importlib*. This module may not be available in older installations like Ubuntu-Lucid.

## 7.4 Gscreen

### 7.4.1 Intro

Gscreen is an infrastructure to display a custom screen to control LinuxCNC. Gscreen borrows heavily from GladeVCP. GladeVCP uses the GTK widget editor GLADE to build virtual control panels (VCP) by point and click. Gscreen combines this with python programming to create a GUI screen for running a CNC machine.

Gscreen is customizable if you want different buttons and status LEDs. Gscreen supports GladeVCP which is used to add controls and indicators. To customize Gscreen you use the Glade editor. Gscreen is not restricted to adding a custom panel on the right or a custom tab it is fully editable.

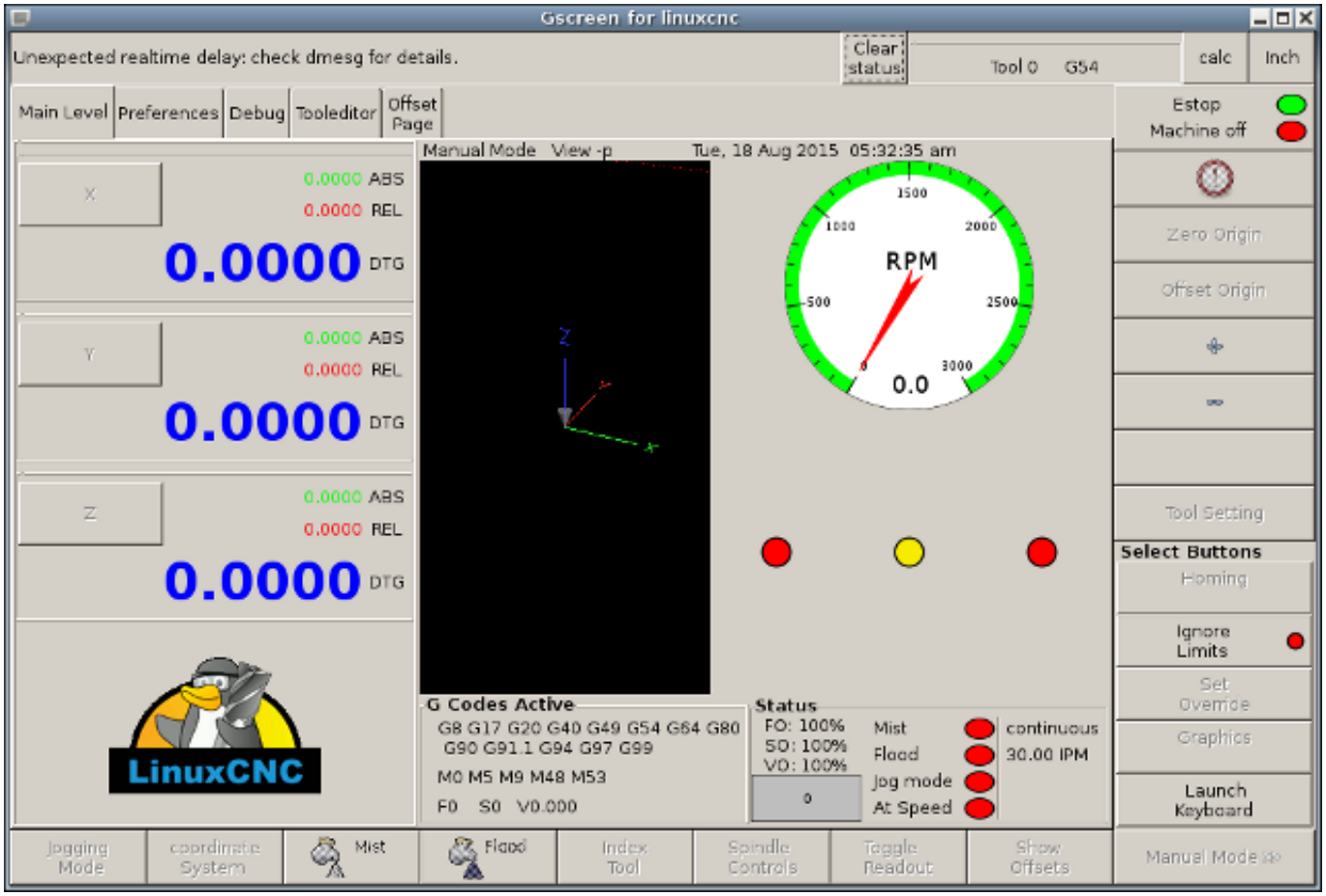



Figure 7.6: Gscreen Mill Base Screen

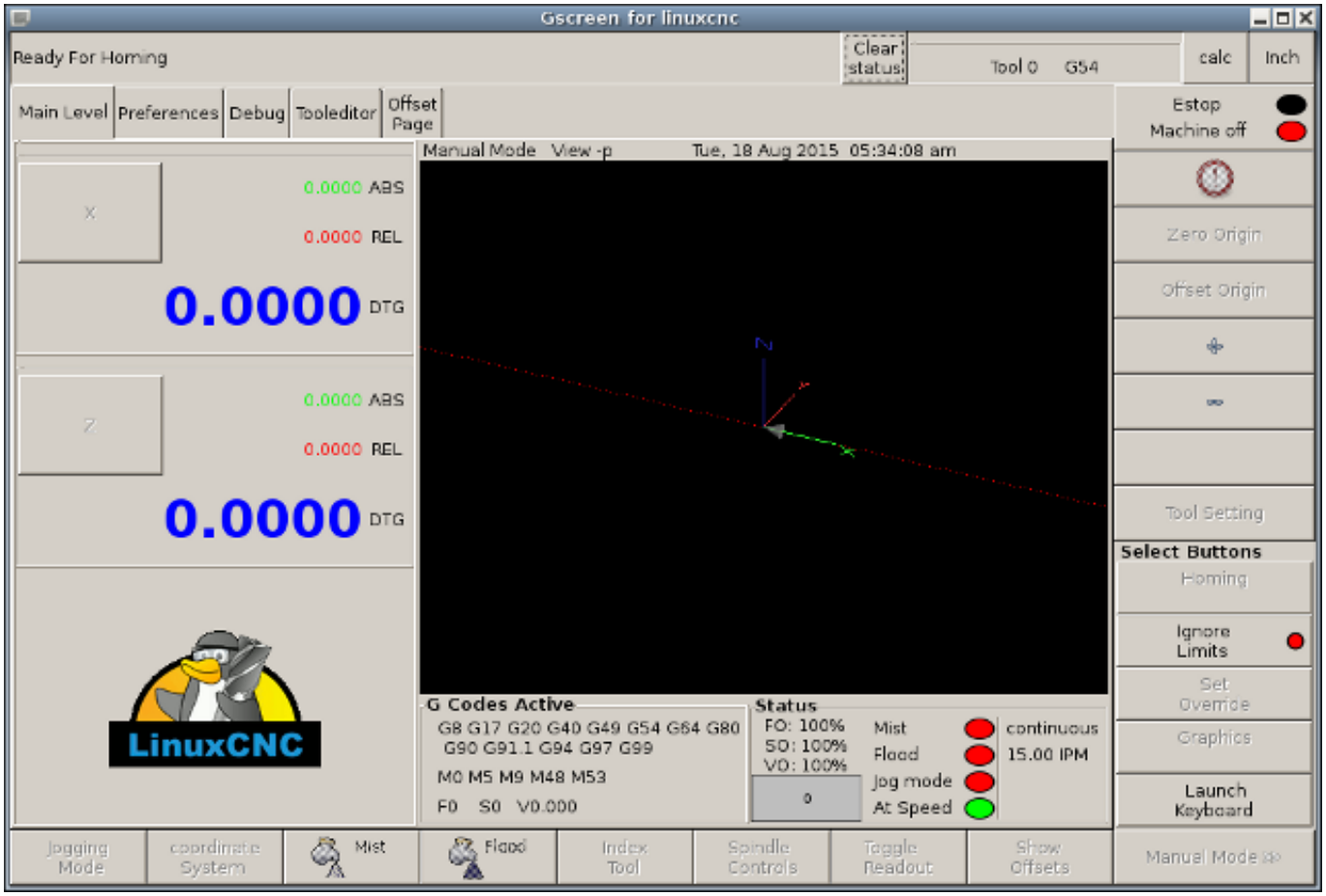



Figure 7.7: Gscreen Lathe Base Screen

Gscreen is based on *Glade* (the editor), *PyGTK* (the widget toolkit), and *GladeVCP* (LinuxCNC's connection to Glade and PyGTK). GladeVCP has some special widgets and actions added just for LinuxCNC A widget is just the generic name used for the buttons, sliders, labels etc of the PyGTK toolkit.

#### 7.4.1.1 Glade File

A Glade file is a text file organized in the XML standard that describes the layout and the widgets of the screen. PyGTK uses this file to actually display and react to those widgets. The Glade editor makes it relatively easy to build and edit this file You must use the Glade 3.8.0 editor that uses the GTK2 widgets. Newer versions of Linux use the newest Glade editor that uses GTK3 widgets. In that case you must download the older Glade editor from their repository.

#### 7.4.1.2 PyGTK

PyGTK is the python binding to GTK. GTK is the *toolkit* of visual widgets, it's programmed in C. PyGTK uses Python to *bind* with GTK.

### 7.4.2 GladeVCP

GladeVCP binds LinuxCNC, HAL, PyGTK and Glade all together. LinuxCNC requires some special widgets so GladeVCP supplies them. Many are just HAL extensions to existing PyGTK widgets. GladeVCP creates the HAL pins for the special widgets described in the Glade file. GladeVCP also allows one to add python commands to interact with the widgets, to make them do things not available in their default form If you can build a GladeVCP panel you can customize Gscreen!

#### 7.4.2.1 Overview

There are two files that can be used, individually or in combination to add customizations. Local glade files and handler files. Normally Gscreen uses the stock Glade file and possibly a handler file (if using a sample *skin*) You can specify Gscreen to use *local* Glade and handler files. Gscreen looks in the folder that holds all the configuration files for the configuration you selected.

**Local Glade Files** If present, local glade files in the configuration folder will be loaded instead of the stock Glade files. Local Glade files allow you to use your customized designs rather then the default screens. There is a switch in the INI file to set set the base name: -c name so Gscreen looks for MYNAME.glade and MYNAME_handler.py.

You can tell Gscreen to just load the Glade file and not connect it's internal signals to it. This allows gscreen to load any GTK builder saved Glade file This means you can display a completely custom screen, but also requires you to use a handler file. Gscreen uses the Glade file to define the widgets, so it can show and interact with them. Many of them have specific names, others have Glade given generic names. If the widget will be displayed but never changed then a generic name is fine. If one needs to control or interact with the widget then a hopefully purposeful name is given (all names must be unique). Widgets can also have signals defined for them in the GLADE editor. It defines what signal is given and what method to call.

**Modifying Stock Skins** If you change the name of a widget, Gscreen might not be able to find it. If this widget is referenced to from python code, at best this makes the widget not work anymore at worst it will cause an error when loading Gscreen's default screens don't use many signals defined in the editor, it defines them in the python code. If you move (cut and paste) a widget with signals, the signals will not be copied. You must add them again manually.

**Handler Files** A handler file is a file containing python code, which Gscreen adds to it's default routines. A handler file allows one to modify defaults, or add logic to a Gscreen skin without having to modify Gscreen proper. You can combine new functions with Gscreen's function to modify behavior as you like. You can completely bypass all of Gscreen's functions and make if work completely differently. If present a handler file named gscreen_handler.py (or MYNAME_handler.py if using the INI switch) will be loaded and registered only one file is allowed Gscreen looks for the handler file, if found it will look for specific function names and call them instead of the default ones. If adding widgets you can set up signal calls from the Glade editor to call routines you have written in the handler file. In this way you can have custom behavior. Handler routines can call Gscreen's default routines, either before or after running it's own code. In this way you can tack on extra behavior such as adding a sound. Please see the GladeVCP Chapter for the basics to GladeVCP handler files. Gscreen uses a very similar technique.

**Themes** Gscreen uses the PyGTK toolkit to display the screen. Pygtk is the Python language binding to GTK. GTK supports *themes*. Themes are a way to modify the look and feel of the widgets on the screen. For instance the color or size of buttons and sliders can be changed using themes. There are many GTK2 themes available on the web. Themes can also be customized to modifiy visuals of particular named widgets. This ties the theme file to the glade file more tightly. Some of the sample screen skins allow the user to select any of the themes on the system. The sample *gscreen* is an example. Some will load the theme that is the same name in the config file. The sample *gscreen-gaxis* is an example. This is done by putting the theme folder in the config folder that has the INI and HAL files and naming it: SCREENNAME_theme (SCREENNAME being the base name of the files eg. gaxis_theme) Inside this folder is another folder call gtk-2.0, inside that is the theme files. If you add this file, Gscreen will default to this theme on start up. gscreen-gaxis has a sample custom theme that looks for certain named widgets and changes the visual behaivor of those specific widgets. The Estop and machine-on buttons use different colors then the rest of the buttons so that they stand out. This is done in the handler file by giving them specific names an by adding specific commands in the theme's gtkrc file. For some info on GTK2 theming (The sample theme uses the pixmap theme engine) See:

GTK Themes

Pixmap Theme Engine

#### 7.4.2.2 Build a GladeVCP Panel

Gscreen is just a big complicated GladeVCP panel, with python code to control it. To customize it we need the Glade file loaded in the Glade editor.

**Installed LinuxCNC** If you have LinuxCNC 2.6+ installed on Ubuntu 10.04 just start the Glade editor from the applications menu or from the terminal. Newer versions of Linux will require you to install Glade 3.8.0

**RIP compiled commands** Using a compiled from source version of LinuxCNC open a terminal and cd to the top of the LinuxCNC folder. Set up the environment by entering . *./scripts/rip-environment* now enter *glade*, you see a bunch of warnings in the terminal that you can ignore and the editor should open. The stock Gscreen Glade file is in: src/emc/usr_intf/gscreen/ sample

skins are in /share/gscreen/skins/. This should be copied to a configuration folder. Or you can make a clean-sheet Glade file by saving it in a configuration folder.

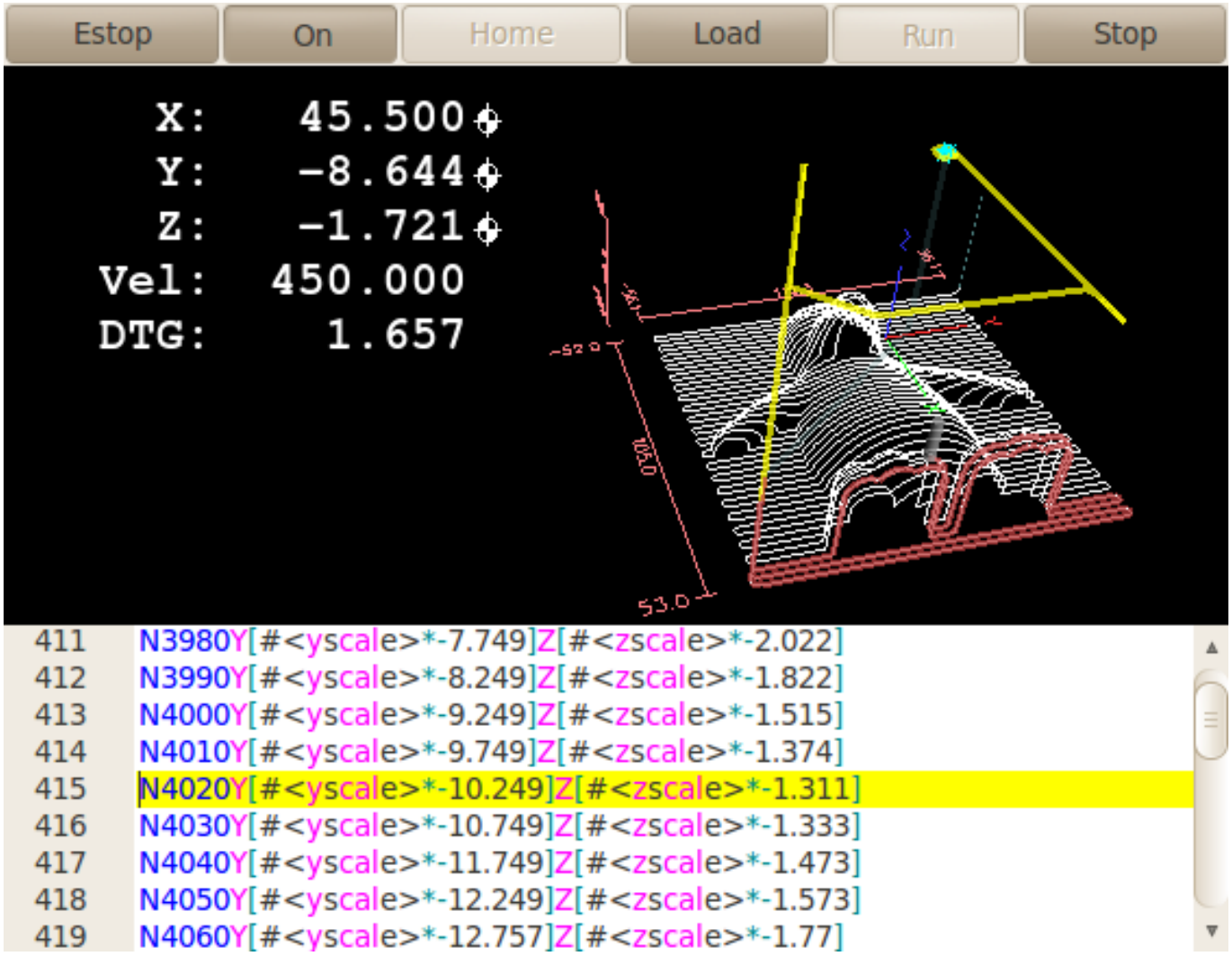
Ok you have loaded the stock Glade file and now can edit it. The first thing you notice is it does not look in the editor like what it's displayed like Gscreen uses some tricks, such as hiding all boxes of buttons except one and changing that one depending on the mode. The same goes for notebooks, some screens use notebooks with the tabs not shown. To change pages in the editor you need to temporarily show those tabs.

When making changes it is far easier to add widgets then subtract widgets and still have the screen work properly making objects *not visible* is one way to change the display without getting errors. This won't always work some widgets will be set visible again. Changing the names of Gscreen's regular widgets is probably not gonna work well without changing the python code, but moving a widget while keeping the name is usually workable.

Gscreen leverages GladeVCP widgets as much as possible, to avoid adding python code. Learning about GladeVCP widgets is a prerequisite. If the existing widgets give you the function you want or need then no python code needs be added, just save the Glade file in your configuration folder. If you need something more custom then you must do some python programming. The name of the parent window needs to be window1. Gscreen assumes this name.

Remember, if you use a custom screen option YOU are responsible for fixing it (if required) when updating LinuxCNC.

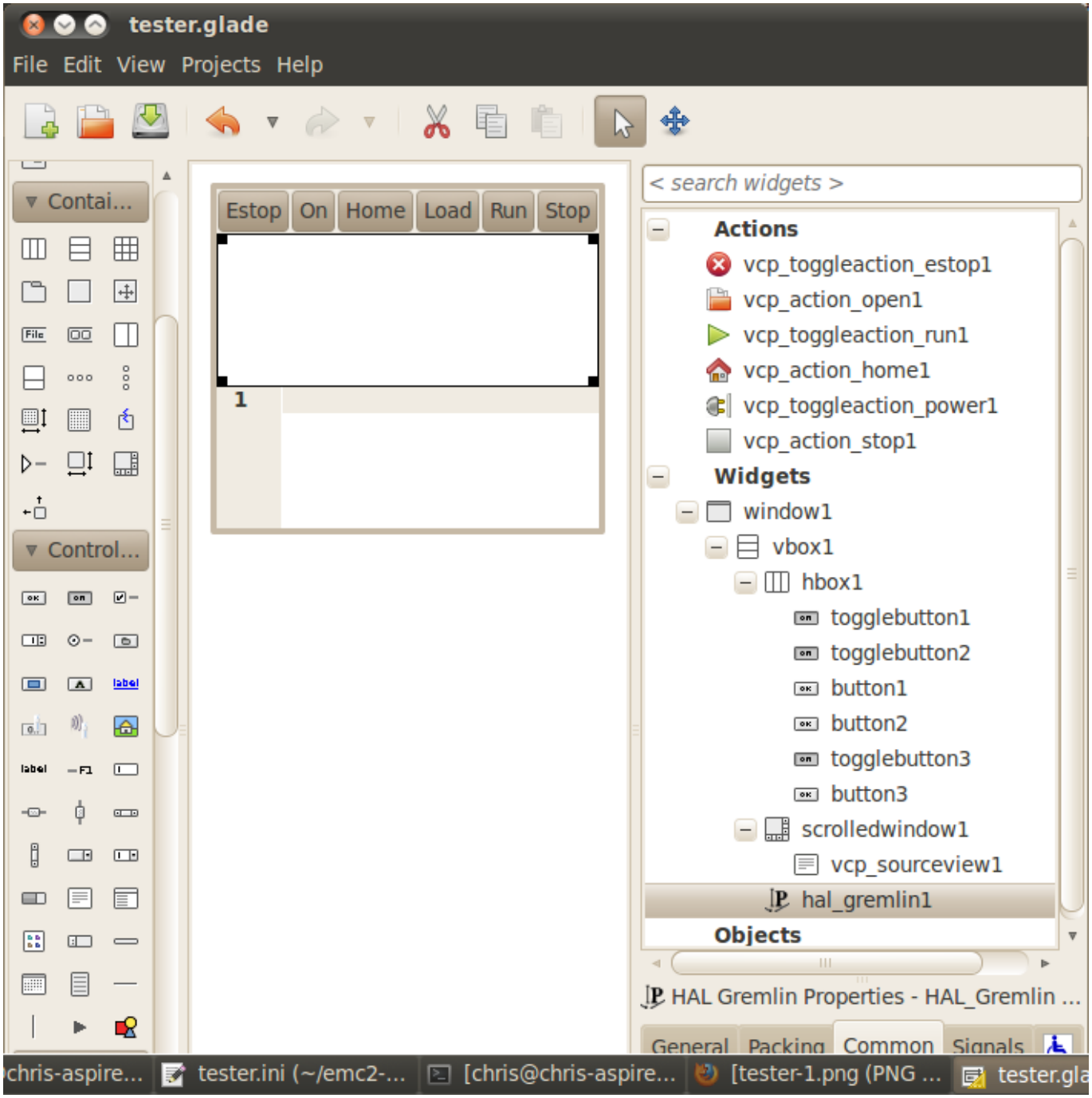
### 7.4.3 Building a simple clean-sheet custom screen




Lets build a simple usable screen. Build this in the Glade editor (if using a RIP package run it from a terminal after using . scripts/rip-environment ).

THINGS TO NOTE:

- The top level window must be called the default name, *window1* - Gscreen relies on this.
- Add actions by clicking the one you want and then clicking somewhere on the window, they don't add anything visual to the window but are added to the right most action list. Add all the ones you see on the top right.
- After adding the actions we must link the buttons to the actions for them to work (see below).
- The gremlin widget doesn't have a default size so setting a requested size is helpful (see below).
- The sourceview widget will try to use the whole window so adding it to a scrolled window will cover this (This is already been done in the example).
- The buttons will expand as the window is made larger which is ugly so we will set the box they are in, to not expand (see below).
- The button types to use depend on the VCP_action used -eg vcp_toggle_action usually require toggle buttons (Follow the example for now).
- The buttons in this example are regular buttons not HAL buttons. We don't need the HAL pins.




In this screen we are using VCP_actions to communicate to LinuxCNC the actions we want. This allows us standard functions without adding python code in the handler file. Let's link the estop toogle button to the estop action Select the estop toggle button and under the general tab look for *Related Action* and click the button beside it. Now select the toggle estop action. Now the button will toggle estop on and off when clicked. Under the general tab you can change the text of the button's label to describe it's function. Do this for all the buttons.

Select the gremlin widget click the common tab and set the requested height to 100 and click the checkbox beside it.

Click the horizontal box that holds the buttons. Click the packing tab and click *expand* to *No*.

Save it as tester.glade and save it in sim/gscreen/gscreen_custom/ folder. Now launch LinuxCNC and click to sim/gscreen/gscreen_custor and start it. If all goes well our screen will pop up and the buttons will do their job This works because the tester.ini tells gscreen to look for and load tester.glade and tester_handler.py. The tester_handler.py file is included in that folder and is coded just show the screen and not much else. Since the special widgets directly communicate with LinuxCNC you can still do useful things. If

your screen needs are covered by the available special widgets then this is as far as you need to go to build a screen. If you want something more there are still many tricks available from just adding *function calls* to get canned behaviour. To coding your own python code to customize exactly what you want. But that means learning about handler files.

### 7.4.4 Handler file example

There are special functions Gscreen checks the handler file for. If you add these in you handler file Gscreen will call them instead of gscreen's internal same-named functions.

- initialize_preferences(self): You can install new preference routines.
- initialize_keybindings(self) You can install new keybinding routines. In most cases you won't want to do this, you will want to override the individual keybinding calls. You can also add more keybindings that will call an arbitrary function.
- initialize_pins(self): makes / initializes HAL pins
- connect_signals(self,handlers): If you are using a completely different screen the default Gscreen you must add this or gscreen will try to connect signals to widgets that are not there. Gscreen's default function is called with self.gscreen.connect_signals(handlers) If you wish to just add extra signals to your screen but still want the default ones call this first then add more signals. If you signals are simple (no user data passed) then you can also use the Glade signal selection in the Glade editor.
- initialize_widgets(self): You can use this to set up any widgets. Gscreen usually calls *self.gscreen.initialize_widgets()* which actually calls many separate functions. If you wish to incorporate some of those widgets then just call those functions directly. or add self.gscreen.init_show_windows() so widgets are just shown. Then if desired, initialize/adjust your new widgets.
- initialize_manual_toolchange(self): allows a complete revamp of the manual toolchange system.
- set_restart_line(self.line):
- timer_interrupt(self): allows one to complete redefine the interrupt routine This is used for calling periodic() and checking for errors from linuxcnc.status.
- check_mode(self): used to check what mode the screen is in. Returns a list[] 0 -manual 1- mdi 2- auto 3- jog.
- on_tool_change(self,widget): You can use this to override the manual tool change dialog -this is called when *gscreen.toolchange* changes state.
- dialog_return(self,dialog_widget,displaytype,pinname): Use this to override any user message or manual tool change dialog. Called when the dialog is closed.
- periodic(self): This is called every (default 100) milliseconds. Use it to update your widgets/HAL pins. You can call Gscreen regular periodic afterwards too, self.gscreen.update_position() or just add pass to not update anything. Gscreen's update_position() actually calls many separate functions. If you wish to incorporate some of those widgets then just call those functions directly.

You can also add you own functions to be called in this file. Usually you would add a signal to a widget to call your function.

#### 7.4.4.1 Adding Features

Our tester example would be more useful if it responded to keyboard commands. There is a function called keybindings() that tries to set this up. While you can override it completely, we didn't - but it assumes some things. It assumes the estop toggle button is call *button_estop* and that F1 key controls it. It assumes the power button is called *button_machine_on* and the F2 key controls it. These are easily fixed by renaming the buttons in the Glade editor to match. But instead we are going to override the standard calls and add our own.

```
Add these command to the handler file:
  # override Gscreen Functions
  # keybinding calls
  def on_keycall_ESTOP(self,state,SHIFT,CNTRL,ALT):
    if state: # only if pressed, not released
```

```
      self.widgets.togglebutton1.emit('activate')
      self.gscreen.audio.set_sound(self.data.alert_sound)
      self.gscreen.audio.run()
      return True # do not let the signal continue to ther widgets
  def on_keycall_POWER(self,state,SHIFT,CNTRL,ALT):
    if state:
      self.widgets.togglebutton2.emit('activate')
      return True
  def on_keycall_ABORT(self,state,SHIFT,CNTRL,ALT):
    if state:
      self.widgets.button3.emit('activate')
      return True
```

So now we have overridden Gscreen's function calls of the same name and deal with them in our handler file. We now reference the widgets by the name we used in the Glade editor. We also added a built in gscreen function to make a sound when Estop changes. Note that we we call Gscreen's built in functions we must use self.gscreen.[FUNCTION NAME]() . If we used self.[FUNCTION NAME]() it would call the function in our handler file.

Lets add another key binding that loads halmeter when F4 is pressed.

```
In the handler file under def initialize_widgets(self): change to:
  def initialize_widgets(self):
    self.gscreen.init_show_windows()
    self.gscreen.keylookup.add_conversion('F4','TEST','on_keycall_HALMETER')

Then add:
  def on_keycall_HALMETER(self,state,SHIFT,CNTRL,ALT):
    if state:
      self.gscreen.on_halmeter()
      return True
```

This adds a keybinding conversion that directs gscreen to call on_keycall_HALMETER when F4 is pressed. Then we add the function to the handle file to call a Gscreen builtin function to start halmeter.

### 7.4.5 Gscreen Start Up

Gscreen is really just infrastructure to load a custom GladeVCP file and interact with it.

1. Gscreen reads the options it was started with.
2. Gscreen sets the debug mode and set the optional skin name.
3. Gscreen checks to see if there are *local* XML, handler and/or locale files in the configuration folder. It will use them instead of the default ones (in share/gscreen/skins/) (There can be two separate screens displayed).
4. The main screen is loaded and translations set up. If present the second screen will be loaded and translations set up.
5. Optional Audio is initialized if available.
6. It reads some of the INI file to initialize the units, and the number/type of axes.
7. Initializes Python's binding to HAL to build a userspace component with the Gscreen name.
8. GladeVCP's makepins is called to parse the XML file to build HAL pins for the HAL widgets and register the LinuxCNC connected widgets.
9. Checks for a *local* handler file in the configuration folder or else uses the stock one from the skin folder.
10. If there is a handler file gscreen parses it, and registers the function calls into Gscreen's namespace.
11. Glade matches/registers all signal calls to functions in gscreen and the handler file.

12. Gscreen checks the INI file for an option preference file name otherwise it uses *.gscreen_preferences* =.
13. Gscreen checks to see if there is a preference function call (*initialize_preferences(self)*) in the handler file otherwise it uses the stock Gscreen one.
14. Gscreen checks for classicladder realtime component.
15. Gscreen checks for the system wide GTK theme.
16. Gscreen collects the jogging increments from the INI file.
17. Gscreen collects the angular jogging increments from the INI file.
18. Gscreen collects the default and max jog rate from the INI.
19. Gscreen collects the max velocity of any axes from the INI's TRAJ section.
20. Gscreen checks to see if there is angular axes then collects the default and max velocity from the INI file.
21. Gscreen collect all the override setting from the INI.
22. Gscreen checks if its a lathe configuration from the INI file.
23. Gscreen finds the name of the tool_table,tool editor and param file from the INI.
24. Gscreen checks the handler file for keybindings function (*initialize_keybindings(self)*) or else use Gscreen stock one.
25. Gscreen checks the handler file for pins function (*initialize_pins(self)*) or else use Gscreen stock one.
26. Gscreen checks the handler file for manual_toolchange function (*initialize_manual_toolchange(self)*) or else use Gscreen stock one.
27. Gscreen checks the handler file for connect_signals function (*initialize_connect_signals(self)*) or else use Gscreen stock one.
28. Gscreen checka the handler file for widgets function (*initialize_widgets(self)*) or else use Gscreen stock one.
29. Gscreen seta up messages specified in the INI file.
30. Gscreen tells HAL the Gscreen HAL component is finished making pins and is ready. If there is a terminal widget in the screen it will print all the Gscreen pins to it.
31. Gscreen sets the display cycle time based on the INI file.
32. Gscreen checks the handler file for *timer_interupt(self)* function call otherwise use Gscreen's default function call.

### 7.4.6 INI Settings

Under the [DISPLAY] heading:

```
DISPLAY = gscreen -c tester
  options:
   -d debugging on
   -v verbose debugging on
```

The -c switch allows one to select a *skin*. Gscreen assumes the Glade file and the handler file use this same name. The optional second screen will be the same name with a 2 (eg. tester2.glade) There is no second handler file allowed. It will only be loaded if it is present. Gscreen will search the LinuxCNC configuration file that was launched first for the files, then in the systen skin folder.

# Chapter 8

# User Interfaces

## 8.1 HAL User Interface

### 8.1.1 Introduction

Halui is a HAL based user interface for LinuxCNC, it connects HAL pins to NML commands. Most of the functionality (buttons, indicators etc.) that is provided by a traditional GUI (mini, Axis, etc.), is provided by HAL pins in Halui.

The easiest way to add halui is to add the following to the [HAL] section of the ini file.

```
HALUI = halui
```

An alternate way to invoke it is to include the following in your .hal file. Make sure you use the actual path to your ini file.

```
loadusr halui -ini /path/to/inifile.ini
```

### 8.1.2 Halui pin reference

ABORT

- *halui.abort* (bit, in) - pin to send an abort message (clears out most errors)

AXIS

- *halui.axis.n.pos-commanded* (float, out) - Commanded axis position in machine coordinates
- *halui.axis.n.pos-feedback* (float, out) - Feedback axis position in machine coordinates
- *halui.axis.n.pos-relative* (float, out) - Commanded axis position in relative coordinates

E-STOP

- *halui.estop.activate* (bit, in) - pin for requesting E-Stop
- *halui.estop.is-activated* (bit, out) - indicates E-stop reset
- *halui.estop.reset* (bit, in) - pin for requesting E-Stop reset

FEED OVERRIDE

- *halui.feed-override.count-enable* (bit, in) - must be true for *counts* or *direct-value* to work.

- *halui.feed-override.counts* (s32, in) - counts * scale = FO percentage. Can be used with an encoder or *direct-value*.
- *halui.feed-override.decrease* (bit, in) - pin for decreasing the FO (-=scale)
- *halui.feed-override.increase* (bit, in) - pin for increasing the FO (+=scale)
- *halui.feed-override.direct-value* (bit, in) - false when using encoder to change counts, true when setting counts directly. The *count-enable* pin must be true.
- *halui.feed-override.scale* (float, in) - pin for setting the scale for increase and decrease of *feed-override*.
- *halui.feed-override.value* (float, out) - current FO value

MIST

- *halui.mist.is-on* (bit, out) - indicates mist is on
- *halui.mist.off* (bit, in) - pin for requesting mist off
- *halui.mist.on* (bit, in) - pin for requesting mist on

FLOOD

- *halui.flood.is-on* (bit, out) - indicates flood is on
- *halui.flood.off* (bit, in) - pin for requesting flood off
- *halui.flood.on* (bit, in) - pin for requesting flood on

HOMING

- *halui.home-all* (bit, in) - pin for requesting all axis to home. This pin will only be there if HOME_SEQUENCE is set in the ini file.

**Jog** <n> is a number between 0 and 8 and *selected*.

- *halui.jog-deadband* (float, in) - deadband for analog jogging (smaller jogging speed requests are not performed)
- *halui.jog-speed* (float, in) - pin for setting jog speed for minus/plus jogging
- *halui.jog.<n>.analog* (float, in) - analog velocity input for jogging (useful with joysticks or other analog devices)
- *halui.jog.<n>.increment* (float,in) - pin for setting the jog increment for axis <n> when using increment-minus or increment-plus to jog.
- *halui.jog.<n>.increment-minus* (bit, in) - pin for moving the <n> axis one increment in the minus direction for each off to on transition.
- *halui.jog.<n>.increment-plus* (bit, in) - pin for moving the <n> axis one increment in the plus direction for each off to on transition.
- *halui.jog.<n>.minus* (bit, in) - pin for jogging axis <n> in negative direction at the halui.jog.speed velocity
- *halui.jog.<n>.plus* (bit, in) - pin for jogging axis <n> in positive direction at the halui.jog.speed velocity
- *halui.jog.selected.increment* (float,in) - pin for setting the jog increment for the selected axis when using increment-minus or incremet-plus to jog.
- *halui.jog.selected.increment-minus* (bit, in) - pin for moving the selected axis one increment in the minus direction for each off to on transition.
- *halui.jog.selected.increment-plus* (bit, in) - pin for moving the selected axis one increment in the plus direction for each off to on transition.

- *halui.jog.selected.minus* (bit, in) - pin for jogging the selected axis in negative direction at the halui.jog.speed velocity
- *halui.jog.selected.plus* (bit, in) - pin for jogging the selected axis in positive direction at the halui.jog.speed velocity

**Joint** <n> is a number between 0 and 8 and *selected*.

- *halui.joint.<n>.has-fault* (bit, out) - status pin telling the joint has a fault
- *halui.joint.<n>.home* (bit, in) - pin for homing the specific joint
- *halui.joint.<n>.is-homed* (bit, out) - status pin telling that the joint is homed
- *halui.joint.<n>.is-selected bit* (bit, out) - status pin a joint is selected* internal halui
- *halui.joint.<n>.on-hard-max-limit* (bit, out) - status pin telling joint <n> is on the positive hardware limit switch
- *halui.joint.<n>.on-hard-min-limit* (bit, out) - status pin telling joint <n> is on the negative hardware limit switch
- *halui.joint.<n>.on-soft-max-limit* (bit, out) - status pin telling joint <n> is at the positive software limit
- *halui.joint.<n>.on-soft-min-limit* (bit, out) - status pin telling joint <n> is at the negative software limit
- *halui.joint.<n>.select* (bit, in) - select joint (0..8) - internal halui
- *halui.joint.<n>.unhome* (bit, in) - unhomes this joint
- *halui.joint.selected* (u32, out) - selected joint (0..8) - internal halui
- *halui.joint.selected.has-fault* (bit, out) - status pin telling that the joint <n> has a fault
- *halui.joint.selected.home* (bit, in) - pin for homing the selected joint
- *halui.joint.selected.is-homed* (bit, out) - status pin telling that the selected joint is homed
- *halui.joint.selected.on-hard-max-limit* (bit, out) - status pin telling that the selected joint is on the positive hardware limit
- *halui.joint.selected.on-hard-min-limit* (bit, out) - status pin telling that the selected joint is on the negative hardware limit
- *halui.joint.selected.on-soft-max-limit* (bit, out) - status pin telling that the selected joint is on the positive software limit
- *halui.joint.selected.on-soft-min-limit* (bit, out) - status pin telling that the selected joint is on the negative software limit
- *halui.joint.selected.unhome* (bit, in) - pin for unhoming the selected joint.

LUBE

- *halui.lube.is-on* (bit, out) - indicates lube is on
- *halui.lube.off* (bit, in) - pin for requesting lube off
- *halui.lube.on* (bit, in) - pin for requesting lube on

MACHINE

- *halui.machine.is-on* (bit, out) - indicates machine on
- *halui.machine.off* (bit, in) - pin for requesting machine off
- *halui.machine.on* (bit, in) - pin for requesting machine on

**Max Velocity** The maximum linear velocity can be adjusted from 0 to the MAX_VELOCITY that is set in the [TRAJ] section of the ini file.

- *halui.max-velocity.count-enable* (bit, in) - must be true for *counts* or *direct-value* to work.

- *halui.max-velocity.counts* (s32, in) - counts * scale = MV percentage. Can be used with an encoder or *direct-value*.
- *halui.max-velocity.direct-value* (bit, in) - false when using encoder to change counts, true when setting counts directly. The *count-enable* pin must be true.
- *halui.max-velocity.decrease* (bit, in) - pin for decreasing max velocity
- *halui.max-velocity.increase* (bit, in) - pin for increasing max velocity
- *halui.max-velocity.scale* (float, in) - the amount applied to the current maximum velocity with each transition from off to on of the increase or decrease pin in machine units per second.
- *halui.max-velocity.value* (float, out) - is the maximum linear velocity in machine units per second.

**MDI** Sometimes the user wants to add more complicated tasks to be performed by the activation of a HAL pin. This is possible using the following MDI commands scheme:

- The MDI_COMMAND is added to the ini file in the [HALUI] section.

```
[HALUI]
MDI_COMMAND = G0 X0
```

- When halui starts it will read the MDI_COMMAND fields in the ini, and export pins from 00 to the number of MDI_COMMAND's found in the ini up to a maximum of 64 commands.
- *halui.mdi-command-<nn>* (bit, in) - halui will try to send the MDI command defined in the ini. This will not always succeed, depending on the operating mode LinuxCNC is in (e.g. while in AUTO halui can't successfully send MDI commands). If the command succeeds then it will place LinuxCNC in the MDI mode and then back to Manual mode.

### Joint Selection

- *halui.joint.select* (u32, in) - select joint (0..8) - internal halui
- *halui.joint.selected* (u32, out) - joint (0..8) selected* internal halui
- *halui.joint.x.select bit* (bit, in) - pins for selecting a joint* internal halui
- *halui.joint.x.is-selected bit* (bit, out) - indicates joint selected* internal halui

### Mode

- *halui.mode.auto* (bit, in) - pin for requesting auto mode
- *halui.mode.is-auto* (bit, out) - indicates auto mode is on
- *halui.mode.is-joint* (bit, out) - indicates joint by joint jog mode is on
- *halui.mode.is-manual* (bit, out) - indicates manual mode is on
- *halui.mode.is-mdi* (bit, out) - indicates mdi mode is on
- *halui.mode.is-teleop* (bit, out) - indicates coordinated jog mode is on
- *halui.mode.joint* (bit, in) - pin for requesting joint by joint jog mode
- *halui.mode.manual* (bit, in) - pin for requesting manual mode
- *halui.mode.mdi* (bit, in) - pin for requesting mdi mode
- *halui.mode.teleop* (bit, in) - pin for requesting coordinated jog mode

### Program

- *halui.program.block-delete.is-on* (bit, out) - status pin telling that block delete is on
- *halui.program.block-delete.off* (bit, in) - pin for requesting that block delete is off
- *halui.program.block-delete.on* (bit, in) - pin for requesting that block delete is on
- *halui.program.is-idle* (bit, out) - status pin telling that no program is running
- *halui.program.is-paused* (bit, out) - status pin telling that a program is paused
- *halui.program.is-running* (bit, out) - status pin telling that a program is running
- *halui.program.optional-stop.is-on* (bit, out) - status pin telling that the optional stop is on
- *halui.program.optional-stop.off* (bit, in) - pin requesting that the optional stop is off
- *halui.program.optional-stop.on* (bit, in) - pin requesting that the optional stop is on
- *halui.program.pause* (bit, in) - pin for pausing a program
- *halui.program.resume* (bit, in) - pin for resuming a paused program
- *halui.program.run* (bit, in) - pin for running a program
- *halui.program.step* (bit, in) - pin for stepping in a program
- *halui.program.stop* (bit, in) - pin for stopping a program

### Spindle Override

- *halui.spindle-override.count-enable* (bit, in) - must be true for *counts* or *direct-value* to work.
- *halui.spindle-override.counts* (s32, in) - counts * scale = SO percentage
- *halui.spindle-override.decrease* (bit, in) - pin for decreasing the SO (-=scale)
- *halui.spindle-override.direct-value* (bit, in) - false when using encoder to change counts, true when setting counts directly. The *count-enable* pin must be true.
- *halui.spindle-override.increase* (bit, in) - pin for increasing the SO (+=scale)
- *halui.spindle-override.scale* (float, in) - pin for setting the scale on changing the SO
- *halui.spindle-override.value* (float, out) - current SO value

### Spindle

- *halui.spindle.brake-is-on* (bit, out) - indicates brake is on
- *halui.spindle.brake-off* (bit, in) - pin for deactivating spindle/brake
- *halui.spindle.brake-on* (bit, in) - pin for activating spindle-brake
- *halui.spindle.decrease* (bit, in) - decreases spindle speed
- *halui.spindle.forward* (bit, in) - starts the spindle with CW motion
- *halui.spindle.increase* (bit, in)- increases spindle speed
- *halui.spindle.is-on* (bit, out) - indicates spindle is on (either direction)
- *halui.spindle.reverse* (bit, in)- starts the spindle with a CCW motion
- *halui.spindle.runs-backward* (bit, out) - indicates spindle is on, and in reverse

- *halui.spindle.runs-forward* (bit, out) - indicates spindle is on, and in forward
- *halui.spindle.start* (bit, in) - starts the spindle
- *halui.spindle.stop* (bit, in) - stops the spindle

TOOL

- *halui.tool.length-offset* (float, out) - indicates current applied tool-length-offset
- *halui.tool.number* (u32, out) - indicates current selected tool

## 8.2 Halui Examples

For any Halui examples to work you need to add the following line to the [HAL] section of the ini file.

```
HALUI = halui
```

### 8.2.1 Remote Start

To connect a remote program start button to LinuxCNC you use the `halui.program.run` pin and the `halui.mode.auto` pin. You have to insure that it is OK to run first by using the `halui.mode.is-auto` pin. You do this with an `and2` component. The following figure shows how this is done. When the Remote Run Button is pressed it is connected to both `halui.mode.auto` and `and2.0.in0`. If it is OK for auto mode the pin `halui.mode.is-auto` will be on. If both the inputs to the `and2.0` component are on the `and2.0.out` will be on and this will start the program.

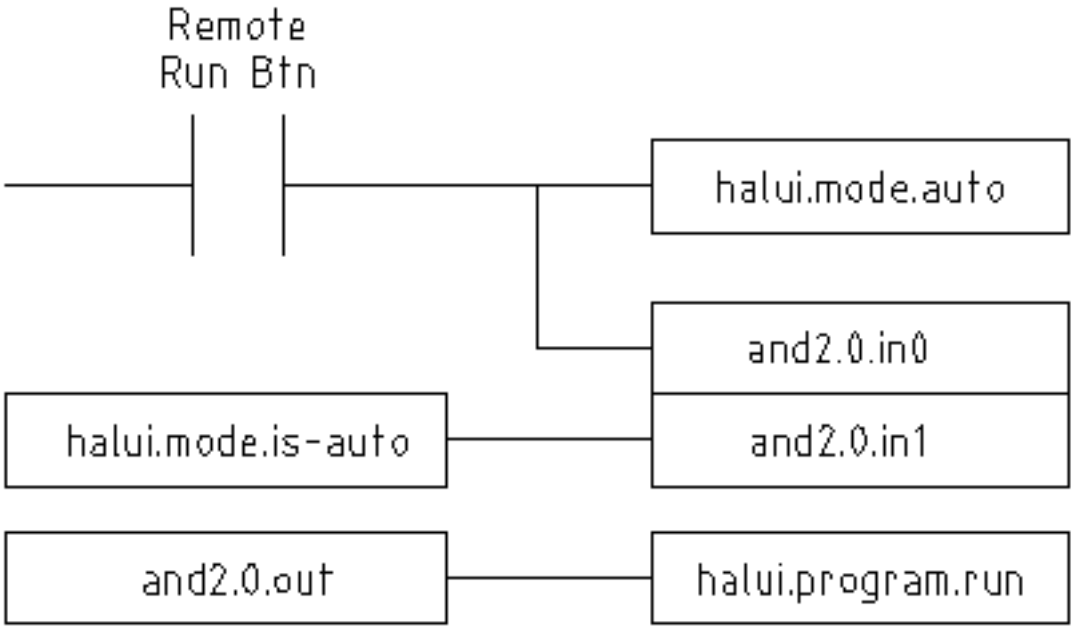



Figure 8.1: Remote Start Example

The hal commands needed to accomplish the above are:

```
net program-start-btn halui.mode.auto and2.0.in0 <= <your input pin>
net program-run-ok and2.0.in1 <= halui.mode.is-auto
net remote-program-run halui.program.run <= and2.0.out
```

Notice on line one that there are two reader pins, this can also be split up to two lines like this:

```
net program-start-btn halui.mode.auto <= <your input pin>
net program-start-btn and2.0.in0
```

### 8.2.2 Pause & Resume

This example was developed to allow LinuxCNC to move a rotary axis on a signal from an external machine. The coordination between the two systems will be provided by two Halui components:

- halui.program.is-paused
- halui.program.resume

In your customized hal file, add the following two lines that will be connected to your I/O to turn on the program pause or to resume when the external system wants LinuxCNC to continue.

```
net ispaused halui.program.is paused => "your output pin"
net resume halui.program.resume <= "your input pin"
```

Your input and output pins are connected to the pins wired to the other controller. They may be parallel port pins or any other I/O pins that you have access to.

This system works in the following way. When an M0 is encountered in your G-code, the `halui.program.is-paused` signal goes true. This turns on your output pin so that the external controller knows that LinuxCNC is paused.

To resume the LinuxCNC gcode program, when the external controller is ready it will make its output true. This will signal LinuxCNC that it should resume executing Gcode.

Difficulties in timing

- The "resume" input return signal should not be longer than the time required to get the g-code running again.
- The "is-paused" output should no longer be active by the time the "resume" signal ends.

These timing problems could be avoided by using ClassicLadder to activate the "is-paused" output via a monostable timer to deliver one narrow output pulse. The "resume" pulse could also be received via a monostable timer.

## 8.3 Python Interface

This documentation describes the *linuxcnc* python module, which provides a python API for talking to LinuxCNC.

### 8.3.1 The linuxcnc Python module

User interfaces control LinuxCNC activity by sending NML messages to the LinuxCNC task controller, and monitor results by observing the LinuxCNC status structure, as well as the error reporting channel.

Programmatic access to NML is through a C++ API; however, the most important parts of the NML interface to LinuxCNC are also available to Python programs through the `linuxcnc` module.

Beyond the NML interface to the command, status and error channels, the `linuxcnc` module also contains:

- support for reading values from ini files
- support for position logging (???)

### 8.3.2 Usage Patterns for the LinuxCNC NML interface

The general pattern for `linuxcnc` usage is roughly like this:

- import the `linuxcnc` module
- establish connections to the command, status and error NML channels as needed
- poll the status channel, either periodically or as needed
- before sending a command, determine from status whether it is in fact OK to do so (for instance, there is no point in sending a *Run* command if task is in the ESTOP state, or the interpreter is not idle)
- send the command by using one of the `linuxcnc` command channel methods

To retrieve messages from the error channel, poll the error channel periodically, and process any messages retrieved.

- poll the status channel, either periodically or as needed
- print any error message FIXME: explore the exception code

`linuxcnc` also defines the `error` Python exception type to support error reporting.

### 8.3.3 Reading LinuxCNC status

Here is a Python fragment to explore the contents of the `linuxcnc.stat` object which contains some 80+ values (run while linuxcnc is running for typical values):

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import sys
import linuxcnc
try:
        s = linuxcnc.stat() # create a connection to the status channel
        s.poll() # get current values
except linuxcnc.error, detail:
        print "error", detail
        sys.exit(1)
for x in dir(s):
        if not x.startswith("_"):
                print x, getattr(s,x)
```

Linuxcnc uses the default compiled-in path to the NML configuration file unless overridden, see Reading ini file values for an example.

#### 8.3.3.1 linuxcnc.stat attributes

**acceleration**
*(returns float)* - default acceleration, reflects the ini entry [TRAJ] DEFAULT_ACCELERATION.

**active_queue**
*(returns integer)* - number of motions blending.

**actual_position**
*(returns tuple of floats)* - current trajectory position, (x y z a b c u v w) in machine units.

**adaptive_feed_enabled**
*(returns boolean)* - status of adaptive feedrate override (0/1).

**ain**
*(returns tuple of floats)* - current value of the analog input pins.

**angular_units**
*(returns float)* - reflects [TRAJ] ANGULAR_UNITS ini value.

**aout**
*(returns tuple of floats)* - current value of the analog output pins.

**axes**
*(returns integer)* - reflects [TRAJ] AXES ini value.

**axis**
*(returns tuple of dicts)* - reflecting current axis values. See The axis dictionary.

**axis_mask**
*(returns integer)* - mask of axis available as defined by [TRAJ] COORDINATES in the ini file. Returns the sum of the axes X=1, Y=2, Z=4, A=8, B=16, C=32, U=64, V=128, W=256.

**block_delete**
*(returns boolean)* - block delete curren status.

**command**
*(returns string)* - currently executing command.

**current_line**
*(returns integer)* - currently executing line, int.

**current_vel**
*(returns float)* - current velocity in user units per second.

**cycle_time**
*(returns float)* - thread period

**debug**
*(returns integer)* - debug flag from the ini file.

**delay_left**
*(returns float)* - remaining time on dwell (G4) command, seconds.

**din**
*(returns tuple of integers)* - current value of the digital input pins.

**distance_to_go**
*(returns float)* - remaining distance of current move, as reported by trajectory planner.

**dout**
*(returns tuple of integers)* - current value of the digital output pins.

**dtg**
*(returns tuple of floats)* - remaining distance of current move for each axis, as reported by trajectory planner.

**echo_serial_number**
*(returns integer)* - The serial number of the last completed command sent by a UI to task. All commands carry a serial number. Once the command has been executed, its serial number is reflected in `echo_serial_number`.

**enabled**
*(returns boolean)* - trajectory planner enabled flag.

**estop**
*(returns integer)* - Returns either STATE_ESTOP or not.

**exec_state**
*(returns integer)* - task execution state. One of EXEC_ERROR, EXEC_DONE, EXEC_WAITING_FOR_MOTION, EXEC_WAITING_FOR_MOTION_QUEUE, EXEC_WAITING_FOR_PAUSE,EXEC_WAITING_FOR_MOTION_AND_IO, EXEC_WAITING_FOR_DELAY, EXEC_WAITING_FOR_SYSTEM_CMD.

**feed_hold_enabled**
*(returns boolean)* - enable flag for feed hold.

**feed_override_enabled**
*(returns boolean)* - enable flag for feed override.

**feedrate**
*(returns float)* - current feedrate override, 1.0 = 100%.

**file**
*(returns string)* - currently loaded gcode filename with path.

**flood**
*(returns integer)* - Flood status, either *FLOOD_OFF* or *FLOOD_ON*.

**g5x_index**
*(returns integer)* -

- replace with constants list currently active coordinate system, G54=1, G55=2 etc.

**g5x_offset**
*(returns tuple of floats)* - offset of the currently active coordinate system.

**g92_offset**
*(returns tuple of floats)* - pose of the current g92 offset.

**gcodes**
*(returns tuple of integers)* - Active G-codes for each modal group. G code constants G_0, G_1, G_2, G_3, G_4, G_5, G_5_1, G_5_2, G_5_3, G_7, G_8, G_100, G_17, G_17_1, G_18, G_18_1, G_19, G_19_1, G_20, G_21, G_28, G_28_1, G_30, G_30_1, G_33, G_33_1, G_38_2, G_38_3, G_38_4, G_38_5, G_40, G_41, G_41_1, G_42, G_42_1, G_43, G_43_1, G_43_2, G_49, G_50, G_51, G_53, G_54, G_55, G_56, G_57, G_58, G_59, G_59_1, G_59_2, G_59_3, G_61, G_61_1, G_64, G_73, G_76, G_80, G_81, G_82, G_83, G_84, G_85, G_86, G_87, G_88, G_89, G_90, G_90_1, G_91, G_91_1, G_92, G_92_1, G_92_2, G_92_3, G_93, G_94, G_95, G_96, G_97, G_98, G_99

**homed**
*(returns tuple of integers)* - 0 = not homed, 1 = homed.

**id**
*(returns integer)* - currently executing motion id.

**inpos**
*(returns boolean)* - machine-in-position flag.

**input_timeout**
*(returns boolean)* - flag for M66 timer in progress.

**interp_state**
*(returns integer)* - current state of RS274NGC interpreter. One of INTERP_IDLE, INTERP_READING, INTERP_PAUSED, INTERP_WAITING.

**interpreter_errcode**
*(returns integer)* - current RS274NGC interpreter return code. One of INTERP_OK, INTERP_EXIT, INTERP_EXECUTE_FINIS INTERP_ENDFILE, INTERP_FILE_NOT_OPEN, INTERP_ERROR. see src/emc/nml_intf/interp_return.hh

**joint_actual_position**
*(returns tuple of floats)* - actual joint positions.

**joint_position**
*(returns tuple of floats)* - Desired joint positions.

**kinematics_type**
*(returns integer)* - identity=1, serial=2, parallel=3, custom=4 .

**limit**
*(returns tuple of integers)* - axis limit masks. minHardLimit=1, maxHardLimit=2, minSoftLimit=4, maxSoftLimit=8.

**linear_units**
*(returns float)* -

reflects [TRAJ]LINEAR_UNITS ini value.

**lube**
*(returns integer)* - *lube on* flag.

**lube_level**
*(returns integer)* - reflects *iocontrol.0.lube_level*.

**max_acceleration**
*(returns float)* - maximum acceleration. reflects [TRAJ] MAX_ACCELERATION.

**max_velocity**
*(returns float)* - maximum velocity. reflects [TRAJ] MAX_VELOCITY.

**mcodes**
*(returns tuple of 10 integers)* - currently active M-codes.

**mist**
*(returns integer)* - Mist status, either MIST_OFF' or *MIST_ON*

**motion_line**
*(returns integer)* - source line number motion is currently executing. Relation to `id` unclear.

**motion_mode**
*(returns integer)* - This is the mode of the Motion controller. One of TRAJ_MODE_COORD, TRAJ_MODE_FREE, TRAJ_MODE_TELEOP.

**motion_type**
*(returns integer)* - The type of the currently executing motion. One of:

- MOTION_TYPE_TRAVERSE
- MOTION_TYPE_FEED
- MOTION_TYPE_ARC
- MOTION_TYPE_TOOLCHANGE
- MOTION_TYPE_PROBING
- MOTION_TYPE_INDEXROTARY
- Or 0 if no motion is currently taking place.

**optional_stop**
*(returns integer)* - option stop flag.

**paused**
*(returns boolean)* - `motion paused` flag.

**pocket_prepped**
*(returns integer)* - A Tx command completed, and this pocket is prepared. -1 if no prepared pocket.

**poll()**
-*(built-in function)* method to update current status attributes.

**position**
*(returns tuple of floats)* - trajectory position.

**probe_tripped**
*(returns boolean)* - flag, True if probe has tripped (latch)

**probe_val**
*(returns integer)* - reflects value of the `motion.probe-input` pin.

**probed_position**
*(returns tuple of floats)* - position where probe tripped.

**probing**
*(returns boolean)* - flag, True if a probe operation is in progress.

**program_units**

default velocity. reflects [TRAJ] DEFAULT_VELOCITY.

#### 8.3.3.2 The `axis` dictionary

The axis configuration and status values are available through a list of per-axis dictionaries. Here's an example how to access an attribute of a particular axis:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import linuxcnc
s = linuxcnc.stat()
s.poll()
print "Axis 1 homed: ", s.axis[1]["homed"]
```

For each axis, the following dictionary keys are available:

**axisType**
*(returns integer)* - type of axis configuration parameter, reflects [AXIS_x]TYPE. LINEAR=1, ANGULAR=2. See Axis ini configuration for details.

**backlash**
*(returns float)* - Backlash in machine units. configuration parameter, reflects [AXIS_x]BACKLASH.

**enabled**
*(returns integer)* - non-zero means enabled.

**fault**
*(returns integer)* - non-zero means axis amp fault.

**ferror_current**
*(returns float)* - current following error.

**ferror_highmark**
*(returns float)* - magnitude of max following error.

**homed**
*(returns integer)* - non-zero means has been homed.

**homing**
*(returns integer)* - non-zero means homing in progress.

**inpos**
*(returns integer)* - non-zero means in position.

**input**
*(returns float)* - current input position.

**max_ferror**
*(returns float)* - maximum following error. configuration parameter, reflects [AXIS_x]FERROR.

**max_hard_limit**
*(returns integer)* - non-zero means max hard limit exceeded.

**max_position_limit**
*(returns float)* - maximum limit (soft limit) for axis motion, in machine units.configuration parameter, reflects [AXIS_x]MAX_LIM

**max_soft_limit**
non-zero means `max_position_limit` was exceeded, int

**min_ferror**
*(returns float)* - configuration parameter, reflects [AXIS_x]MIN_FERROR.

**min_hard_limit**
*(returns integer)* - non-zero means min hard limit exceeded.

**min_position_limit**
*(returns float)* - minimum limit (soft limit) for axis motion, in machine units.configuration parameter, reflects [AXIS_x]MIN_LIMI

**min_soft_limit**
*(returns integer)* - non-zero means `min_position_limit` was exceeded.

**output**
*(returns float)* - commanded output position.

**override_limits**
*(returns integer)* - non-zero means limits are overridden.

**units**
*(returns float)* - units per mm, deg for linear, angular

**velocity**
*(returns float)* - current velocity.

### 8.3.4 Preparing to send commands

Some commands can always be sent, regardless of mode and state; for instance, the `linuxcnc.command.abort()` method can always be called.

Other commands may be sent only in appropriate state, and those tests can be a bit tricky. For instance, an MDI command can be sent only if:

- ESTOP has not been triggered, and
- the machine is turned on and
- the axes are homed and
- the interpreter is not running and
- the mode is set to `MDI mode`

so an appropriate test before sending an MDI command through `linuxcnc.command.mdi()` could be:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import linuxcnc
s = linuxcnc.stat()
c = linuxcnc.command()

def ok_for_mdi():
        s.poll()
        return not s.estop and s.enabled and s.homed and (s.interp_state == linuxcnc. ↩
            INTERP_IDLE)

if ok_for_mdi():
        c.mode(linuxcnc.MODE_MDI)
        c.wait_complete() # wait until mode switch executed
        c.mdi("G0 X10 Y20 Z30")
```

### 8.3.5 Sending commands through `linuxcnc.command`

Before sending a command, initialize a command channel like so:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import linuxcnc
c = linuxcnc.command()

# Usage examples for some of the commands listed below:
c.abort()

c.auto(linuxcnc.AUTO_RUN, program_start_line)
c.auto(linuxcnc.AUTO_STEP)
c.auto(linuxcnc.AUTO_PAUSE)
c.auto(linuxcnc.AUTO_RESUME)

c.brake(linuxcnc.BRAKE_ENGAGE)
c.brake(linuxcnc.BRAKE_RELEASE)

c.flood(linuxcnc.FLOOD_ON)
c.flood(linuxcnc.FLOOD_OFF)

c.home(2)

c.jog(linuxcnc.JOG_STOP, axis)
c.jog(linuxcnc.JOG_CONTINUOUS, axis, speed)
c.jog(linuxcnc.JOG_INCREMENT, axis, speed, increment)

c.load_tool_table()

c.maxvel(200.0)

c.mdi("G0 X10 Y20 Z30")

c.mist(linuxcnc.MIST_ON)
c.mist(linuxcnc.MIST_OFF)

c.mode(linuxcnc.MODE_MDI)
c.mode(linuxcnc.MODE_AUTO)
c.mode(linuxcnc.MODE_MANUAL)

c.override_limits()

c.program_open("foo.ngc")
c.reset_interpreter()

c.tool_offset(toolno, z_offset,  x_offset, diameter, frontangle, backangle, orientation)
```

#### 8.3.5.1 `linuxcnc.command` attributes

**`serial`**
the current command serial number

#### 8.3.5.2 `linuxcnc.command` methods:

**`abort()`**
send EMC_TASK_ABORT message.

**auto(int[, int])**
run, step, pause or resume a program.

**brake(int)**
engage or release spindle brake.

**debug(int)**
set debug level via EMC_SET_DEBUG message.

**feedrate(float)**
set the feedrate.

**flood(int)**
turn on/off flooding.

**home(int)**
home a given axis.

**jog(int, int, [, int[,int]])**
Syntax:
jog(command, axis[, velocity[, distance]])
jog(linuxcnc.JOG_STOP, axis)
jog(linuxcnc.JOG_CONTINUOUS, axis, velocity)
jog(linuxcnc.JOG_INCREMENT, axis, velocity, distance)
Constants:
JOG_STOP (0)
JOG_CONTINUOUS (1)
JOG_INCREMENT (2)

**load_tool_table()**
reload the tool table.

**maxvel(float)**
set maximum velocity

**mdi(string)**
send an MDI command. Maximum 255 chars.

**mist(int)**
turn on/off mist.
Syntax:
mist(command)
mist(linuxcnc.MIST_ON) [(1)]
mist(linuxcnc.MIST_OFF) [(0)]
Constants:
MIST_ON (1)
MIST_OFF (0)

**mode(int)**
set mode (MODE_MDI, MODE_MANUAL, MODE_AUTO).

**override_limits()**
set the override axis limits flag.

**program_open(string)**
open an NGC file.

**reset_interpreter()**
reset the RS274NGC interpreter

**set_adaptive_feed(int)**
set adaptive feed flag

**set_analog_output(int, float)**
set analog output pin to value

**set_block_delete(int)**
set block delete flag

**set_digital_output(int, int)**
set digital output pin to value

**set_feed_hold(int)**
set feed hold on/off

**set_feed_override(int)**
set feed override on/off

**set_max_limit(int, float)**
set max position limit for a given axis

**set_min_limit()**
set min position limit for a given axis

**set_optional_stop(int)**
set optional stop on/off

**set_spindle_override(int)**
set spindle override flag

**spindle(int)**
set spindle direction. Argument one of SPINDLE_FORWARD, SPINDLE_REVERSE, SPINDLE_OFF, SPINDLE_INCREASE, SPINDLE_DECREASE, or SPINDLE_CONSTANT.

**spindleoverride(float)**
set spindle override factor

**state(int)**
set the machine state. Machine state should be STATE_ESTOP, STATE_ESTOP_RESET, STATE_ON, or STATE_OFF

**teleop_enable(int)**
enable/disable teleop mode.

**teleop_vector(float, float, float [,float, float, float])**
set teleop destination vector

**tool_offset(int, float, float, float, float, float, int)**
set the tool offset. See usage example above.

**traj_mode(int)**
set trajectory mode. Mode is one of MODE_FREE, MODE_COORD, or MODE_TELEOP.

**unhome(int)**
unhome a given axis.

**wait_complete([float])**
wait for completion of the last command sent. If timeout in seconds not specified, default is 5 seconds. Return -1 if timed out, return RCS_DONE or RCS_ERROR according to command execution status.

### 8.3.6 Reading the error channel

To handle error messages, connect to the error channel and periodically poll() it.

Note that the NML channel for error messages has a queue (other than the command and status channels), which means that the first consumer of an error message deletes that message from the queue; whether your another error message consumer (e.g. Axis) will *see* the message is dependent on timing. It is recommended to have just one error channel reader task in a setup.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import linuxcnc
e = linuxcnc.error_channel()

error = e.poll()

if error:
        kind, text = error
        if kind in (linuxcnc.NML_ERROR, linuxcnc.OPERATOR_ERROR):
                typus = "error"
        else:
                typus = "info"
        print typus, text
```

### 8.3.7 Reading ini file values

Here's an example for reading values from an ini file through the `linuxcnc.ini` object:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# run as:
# python ini-example.py ~/emc2-dev/configs/sim/axis/axis_mm.ini

import sys
import linuxcnc

inifile = linuxcnc.ini(sys.argv[1])

# inifile.find() returns None if the key wasnt found - the
# following idiom is useful for setting a default value:

machine_name = inifile.find("EMC", "MACHINE") or "unknown"
print "machine name: ", machine_name

# inifile.findall() returns a list of matches, or an empty list
# if the key wasnt found:

extensions = inifile.findall("FILTER", "PROGRAM_EXTENSION")
print "extensions: ", extensions

# override default NML file by ini parameter if given
nmlfile = inifile.find("EMC", "NML_FILE")
if nmlfile:
    linuxcnc.nmlfile = os.path.join(os.path.dirname(sys.argv[1]), nmlfile)
```

### 8.3.8 The `linuxcnc.positionlogger` type

Some usage hints can be gleaned from `src/emc/usr_intf/gremlin/gremlin.py`.

#### 8.3.8.1 members

**`npts`**
number of points.

#### 8.3.8.2 methods

**`start(float)`**
start the position logger and run every ARG seconds

**`clear()`**
clear the position logger

**`stop()`**
stop the position logger

**`call()`**
Plot the backplot now.

**`last([int])`**
Return the most recent point on the plot or None ,

# Chapter 9

# Drivers

## 9.1 Parallel Port Driver

The hal_parport component is a driver for the traditional PC parallel port. The port has a total of 17 physical pins. The original parallel port divided those pins into three groups: data, control, and status. The data group consists of 8 output pins, the control group consists of 4 pins, and the status group consists of 5 input pins.

In the early 1990's, the bidirectional parallel port was introduced, which allows the data group to be used for output or input. The HAL driver supports the bidirectional port, and allows the user to set the data group as either input or output. If configured as *out*, a port provides a total of 12 outputs and 5 inputs. If configured as *in*, it provides 4 outputs and 13 inputs.

In some parallel ports, the control group pins are open collectors, which may also be driven low by an external gate. On a board with open collector control pins. If configured as *x*, it provides 8 outputs, and 9 inputs.

In some parallel ports, the control group has push-pull drivers and cannot be used as an input.

**HAL and Open Collectors**
HAL cannot automatically determine if the *x* mode bidirectional pins are actually open collectors (OC). If they are not, they cannot be used as inputs, and attempting to drive them LOW from an external source can damage the hardware.
To determine whether your port has *open collector* pins, load hal_parport in *x* mode. With no device attached, HAL should read the pin as TRUE. Next, insert a 470 ohm resistor from one of the control pins to GND. If the resulting voltage on the control pin is close to 0V, and HAL now reads the pin as FALSE, then you have an OC port. If the resulting voltage is far from 0V, or HAL does not read the pin as FALSE, then your port cannot be used in *x* mode.
The external hardware that drives the control pins should also use open collector gates (e.g., 74LS05).
On some computers, BIOS settings may affect whether *x* mode can be used. *SPP* mode is most likely to work.

No other combinations are supported, and a port cannot be changed from input to output once the driver is installed.

The parport driver can control up to 8 ports (defined by MAX_PORTS in hal_parport.c). The ports are numbered starting at zero.

### 9.1.1 Loading

The hal_parport driver is a real time component so it must be loaded into the real time thread with *loadrt*. The configuration string consists of at least one port index or port address and optional pin direction specifiers. If the configuration string does not contain at least one address, it is an error.

```
loadrt hal_parport cfg="configuration string"
```

**Using the Port Index** I/O addresses below 16 are treated as port indexes. This is the simplest way to install the parport driver and cooperates with the Linux parport_pc driver if it is loaded. This will use the address Linux has detected for parport 0.

```
loadrt hal_parport cfg="0"
```

**Using the Port Address** The port address is specified using the hex notation *0x* then the address.

```
loadrt hal_parport cfg="0x378"
```

**Direction** The direction is *in*, *out*, or *x* and that determines the direction of the physical pins of the parallel port. If the direction is not specified, the default is *out*.

```
loadrt hal_parport cfg="0 in"
```

Table 9.1: Parallel Port Direction

| Pin | in | out | x |
|---|---|---|---|
| 1 | out | out | in |
| 2 | in | out | out |
| 3 | in | out | out |
| 4 | in | out | out |
| 5 | in | out | out |
| 6 | in | out | out |
| 7 | in | out | out |
| 8 | in | out | out |
| 9 | in | out | out |
| 10 | in | in | in |
| 11 | in | in | in |
| 12 | in | in | in |
| 13 | in | in | in |
| 14 | in | out | in |
| 15 | in | in | in |
| 16 | out | out | in |
| 17 | out | out | in |

**Functions** You must also direct LinuxCNC to run the *read* and *write* functions.

```
addf parport.0.read base-thread
addf parport.0.write base-thread
```

### 9.1.2 PCI Port Address

One good PCI parport card is made with the Netmos 9815 chipset. It has good +5V signals, and can come in a single or dual ports.

To find the I/O addresses for PCI cards open a terminal window and use the list pci command:

```
lspci -v
```

Look for the entry with "Netmos" in it. Example of a 2-port card:

```
0000:01:0a.0 Communication controller: \
      Netmos Technology PCI 9815 Multi-I/O Controller (rev 01)
Subsystem: LSI Logic / Symbios Logic 2POS (2 port parallel adapter)
Flags: medium devsel, IRQ 5
I/O ports at b800 [size=8]
```

```
I/O ports at bc00 [size=8]
I/O ports at c000 [size=8]
I/O ports at c400 [size=8]
I/O ports at c800 [size=8]
I/O ports at cc00 [size=16]
```

From experimentation, I've found the first port (the on-card port) uses the third address listed (c000), and the second port (the one that attaches with a ribbon cable) uses the first address listed (b800). The following example shows the onboard parallel port and a PCI parallel port using the default out direction.

```
loadrt hal_parport cfg="0x378 0xc000"
```

Please note that your values will differ. The Netmos cards are Plug-N-Play, and might change their settings depending on which slot you put them into, so if you like to 'get under the hood' and re-arrange things, be sure to check these values before you start LinuxCNC.

### 9.1.3 Pins

- *parport.<p>.pin-<n>-out* (bit) Drives a physical output pin.
- *parport.<p>.pin-<n>-in* (bit) Tracks a physical input pin.
- *parport.<p>.pin-<n>-in-not* (bit) Tracks a physical input pin, but inverted.

For each pin, *<p>* is the port number, and *<n>* is the physical pin number in the 25 pin D-shell connector.

For each physical output pin, the driver creates a single HAL pin, for example: *parport.0.pin-14-out*.

For each physical input pin, the driver creates two HAL pins, for example: *parport.0.pin-12-in* and *parport.0.pin-12-in-not*.

The *-in* HAL pin is TRUE if the physical pin is high, and FALSE if the physical pin is low. The *-in-not* HAL pin is inverted and is FALSE if the physical pin is high.

### 9.1.4 Parameters

- *parport.<p>.pin-<n>-out-invert* (bit) Inverts an output pin.
- *parport.<p>.pin-<n>-out-reset* (bit) (only for *out* pins) TRUE if this pin should be reset when the *-reset* function is executed.
- parport.<p>.reset-time' (U32) The time (in nanoseconds) between a pin is set by *write* and reset by the *reset* function if it is enabled.

The *-invert* parameter determines whether an output pin is active high or active low. If *-invert* is FALSE, setting the HAL *-out* pin TRUE drives the physical pin high, and FALSE drives it low. If *-invert* is TRUE, then setting the HAL *-out* pin TRUE will drive the physical pin low.

### 9.1.5 Functions

- *parport.<p>.read* (funct) Reads physical input pins of port *<portnum>* and updates HAL *-in* and *-in-not* pins.
- *parport.read-all* (funct) Reads physical input pins of all ports and updates HAL *-in* and *-in-not* pins.
- *parport.<p>.write* (funct) Reads HAL *-out* pins of port *<p>* and updates that port's physical output pins.
- *parport.write-all* (funct) Reads HAL *-out* pins of all ports and updates all physical output pins.
- *parport.<p>.reset* (funct) Waits until *reset-time* has elapsed since the associated *write*, then resets pins to values indicated by *-out-invert* and *-out-invert* settings. *reset* must be later in the same thread as *write. 'If '-reset* is TRUE, then the *reset* function will set the pin to the value of *-out-invert*. This can be used in conjunction with stepgen's *doublefreq* to produce one step per period. The stepgen stepspace for that pin must be set to 0 to enable doublefreq.

The individual functions are provided for situations where one port needs to be updated in a very fast thread, but other ports can be updated in a slower thread to save CPU time. It is probably not a good idea to use both an *-all* function and an individual function at the same time.

### 9.1.6 Common problems

If loading the module reports

```
insmod: error inserting '/home/jepler/emc2/rtlib/hal_parport.ko':
-1 Device or resource busy
```

then ensure that the standard kernel module *parport_pc* is not loaded[1] and that no other device in the system has claimed the I/O ports.

If the module loads but does not appear to function, then the port address is incorrect.

### 9.1.7 Using DoubleStep

To setup DoubleStep on the parallel port you must add the function parport.n.reset after parport.n.write and configure stepspace to 0 and the reset time wanted. So that step can be asserted on every period in HAL and then toggled off by parport after being asserted for time specificed by parport.n.reset-time.

For example:

```
loadrt hal_parport cfg="0x378 out"
setp parport.0.reset-time 5000
loadrt stepgen step_type=0,0,0
addf parport.0.read base-thread
addf stepgen.make-pulses base-thread
addf parport.0.write base-thread
addf parport.0.reset base-thread
addf stepgen.capture-position servo-thread
...
setp stepgen.0.steplen 1
setp stepgen.0.stepspace 0
```

More information on DoubleStep can be found on the wiki.

## 9.2 AX5214H Driver

The Axiom Measurement & Control AX5214H is a 48 channel digital I/O board. It plugs into an ISA bus, and resembles a pair of 8255 chips. In fact it may be a pair of 8255 chips, but I'm not sure. If/when someone starts a driver for an 8255 they should look at the ax5214 code, much of the work is already done.

### 9.2.1 Installing

```
loadrt hal_ax5214h cfg="<config-string>"
```

The config string consists of a hex port address, followed by an 8 character string of "I" and "O" which sets groups of pins as inputs and outputs. The first two character set the direction of the first two 8 bit blocks of pins (0-7 and 8-15). The next two set blocks of 4 pins (16-19 and 20-23). The pattern then repeats, two more blocks of 8 bits (24-31 and 32-39) and two blocks of 4 bits (40-43 and 44-47). If more than one board is installed, the data for the second board follows the first. As an example, the string *"0x220 IIIOIIOO 0x300 OIOOIOIO"* installs drivers for two boards. The first board is at address 0x220, and has 36 inputs (0-19 and 24-39) and 12 outputs (20-23 and 40-47). The second board is at address 0x300, and has 20 inputs (8-15, 24-31, and 40-43) and 28 outputs (0-7. 16-23, 32-39, and 44-47). Up to 8 boards may be used in one system.

[1] In the LinuxCNC packages for Ubuntu, the file /etc/modprobe.d/emc2 generally prevents *parport_pc* from being automatically loaded.

### 9.2.2 Pins

- *(bit) ax5214.<boardnum>.out-<pinnum>* — Drives a physical output pin.
- *(bit) ax5214.<boardnum>.in-<pinnum>* — Tracks a physical input pin.
- *(bit) ax5214.<boardnum>.in-<pinnum>-not* — Tracks a physical input pin, inverted.

For each pin, <boardnum> is the board number (starts at zero), and <pinnum> is the I/O channel number (0 to 47).

Note that the driver assumes active LOW signals. This is so that modules such as OPTO-22 will work correctly (TRUE means output ON, or input energized). If the signals are being used directly without buffering or isolation the inversion needs to be accounted for. The in- HAL pin is TRUE if the physical pin is low (OPTO-22 module energized), and FALSE if the physical pin is high (OPTO-22 module off). The in-<pinnum>-not HAL pin is inverted — it is FALSE if the physical pin is low (OPTO-22 module energized). By connecting a signal to one or the other, the user can determine the state of the input.

### 9.2.3 Parameters

- *(bit) ax5214.<boardnum>.out-<pinnum>-invert* — Inverts an output pin.

The -invert parameter determines whether an output pin is active high or active low. If -invert is FALSE, setting the HAL out- pin TRUE drives the physical pin low, turning ON an attached OPTO-22 module, and FALSE drives it high, turning OFF the OPTO-22 module. If -invert is TRUE, then setting the HAL out- pin TRUE will drive the physical pin high and turn the module OFF.

### 9.2.4 Functions

- *(funct) ax5214.<boardnum>.read* — Reads all digital inputs on one board.
- *(funct) ax5214.<boardnum>.write* — Writes all digital outputs on one board.

## 9.3 GS2 VFD Driver

This is a userspace HAL program for the GS2 series of VFD's at Automation Direct.

This component is loaded using the halcmd "loadusr" command:

```
loadusr -Wn spindle-vfd gs2_vfd -n spindle-vfd
```

The above command says: loadusr, wait for named to load, component gs2_vfd, named spindle-vfd

### 9.3.1 Command Line Options

- *-b or --bits <n>* (default 8) Set number of data bits to <n>, where n must be from 5 to 8 inclusive
- *-d or --device <path>* (default /dev/ttyS0) Set the name of the serial device node to use
- *-g or --debug* Turn on debugging messages. This will also set the verbose flag. Debug mode will cause all modbus messages to be printed in hex on the terminal.
- *-n or --name <string>* (default gs2_vfd) Set the name of the HAL module. The HAL comp name will be set to <string>, and all pin and parameter names will begin with <string>.
- *-p or --parity {even,odd,none}* (default odd) Set serial parity to even, odd, or none.
- *-r or --rate <n>* (default 38400) Set baud rate to <n>. It is an error if the rate is not one of the following: 110, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200

- *-s or --stopbits {1,2}* (default 1) Set serial stop bits to 1 or 2
- *-t or --target <n>* (default 1) Set MODBUS target (slave) number. This must match the device number you set on the GS2.
- *-v or --verbose* Turn on debug messages.
- *-A or --accel-seconds* <n> (default 10.0) Seconds to accelerate the spindle from 0 to Max RPM.
- *-D or --decel-seconds* <n> (default 0.0) Seconds to decelerate the spindle from Max RPM to 0. If set to 0.0 the spindle will be allowed to coast to a stop without controlled deceleration.
- *-R or --braking-resistor* This argument should be used when a braking resistor is installed on the GS2 VFD (see Appendix A of the GS2 manual). It disables deceleration over-voltage stall prevention (see GS2 modbus Parameter 6.05), allowing the VFD to keep braking even in situations where the motor is regenerating high voltage. The regenerated voltage gets safely dumped into the braking resistor.

**Note**
That if there are serial configuration errors, turning on verbose may result in a flood of timeout errors.

### 9.3.2 Pins

Where <n> is gs2_vfd or the name given during loading with the -n option.

- *<n>.DC-bus-volts* (float, out) The DC bus voltage of the VFD
- *<n>.at-speed* (bit, out) when drive is at commanded speed
- *<n>.err-reset* (bit, in) reset errors sent to VFD
- *<n>.firmware-revision* (s32, out) from the VFD
- *<n>.frequency-command* (float, out) from the VFD
- *<n>.frequency-out* (float, out) from the VFD
- *<n>.is-stopped* (bit, out) when the VFD reports 0 Hz output
- *<n>.load-percentage* (float, out) from the VFD
- *<n>.motor-RPM* (float, out) from the VFD
- *<n>.output-current* (float, out) from the VFD
- *<n>.output-voltage* (float, out) from the VFD
- *<n>.power-factor* (float, out) from the VFD
- *<n>.scale-frequency* (float, out) from the VFD
- *<n>.speed-command* (float, in) speed sent to VFD in RPM It is an error to send a speed faster than the Motor Max RPM as set in the VFD
- *<n>.spindle-fwd* (bit, in) 1 for FWD and 0 for REV sent to VFD
- *<n>.spindle-rev* (bit, in) 1 for REV and 0 if off
- *<n>.spindle-on* (bit, in) 1 for ON and 0 for OFF sent to VFD
- *<n>.status-1* (s32, out) Drive Status of the VFD (see the GS2 manual)
- *<n>.status-2* (s32, out) Drive Status of the VFD (see the GS2 manual)

**Note**
The status value is a sum of all the bits that are on. So a 163 which means the drive is in the run mode is the sum of 3 (run) + 32 (freq set by serial) + 128 (operation set by serial).

### 9.3.3 Parameters

Where <n> is gs2_vfd or the name given during loading with the -n option.

- *<n>.error-count* (s32, RW)
- *<n>.loop-time* (float, RW) how often the modbus is polled (default 0.1)
- *<n>.nameplate-HZ* (float, RW) Nameplate Hz of motor (default 60)
- *<n>.nameplate-RPM* (float, RW) Nameplate RPM of motor (default 1730)
- *<n>.retval* (s32, RW) the return value of an error in HAL
- *<n>.tolerance* (s32, RW) speed tolerance (default 0.01)
- *<n>.ack-delay* (s32, RW) number of read/write cycles before checking at-speed (default 2)

For an example of using this component to drive a spindle see the GS2 Spindle example.

## 9.4 Mesa HostMot2 Driver

### 9.4.1 Introduction

HostMot2 is an FPGA configuration developed by Mesa Electronics for their line of *Anything I/O* motion control cards. The firmware is open source, portable and flexible. It can be configured (at compile-time) with zero or more instances (an object created at runtime) of each of several Modules: encoders (quadrature counters), PWM generators, and step/dir generators. The firmware can be configured (at run-time) to connect each of these instances to pins on the I/O headers. I/O pins not driven by a Module instance revert to general-purpose bi-directional digital I/O.

### 9.4.2 Firmware Binaries

**50 Pin Header FPGA cards** Several pre-compiled HostMot2 firmware binaries are available for the different Anything I/O boards. (This list is incomplete, check the hostmot2-firmware distribution for up-to-date firmware lists.)

- 3x20 (144 I/O pins): using hm2_pci module
  - 24-channel servo
  - 16-channel servo plus 24 step/dir generators
- 5i22 (96 I/O pins): using hm2_pci module
  - 16-channel servo
  - 8-channel servo plus 24 step/dir generators
- 5i20, 5i23, 4i65, 4i68 (72 I/O pins): using hm2_pci module
  - 12-channel servo
  - 8-channel servo plus 4 step/dir generators
  - 4-channel servo plus 8 step/dir generators
- 7i43 (48 I/O pins): using hm2_7i43 module
  - 8-channel servo (8 PWM generators & 8 encoders)
  - 4-channel servo plus 4 step/dir generators

**DB25 FPGA cards** The 5i25 Superport FPGA card is preprogrammed when purchased and does not need a firmware binary.

### 9.4.3 Installing Firmware

Depending on how you installed LinuxCNC you may have to open the Synaptic Package Manager from the System menu and install the package for your Mesa card. The quickest way to find them is to do a search for *hostmot2* in the Synaptic Package Manager. Mark the firmware for installation, then apply.

### 9.4.4 Loading HostMot2

The LinuxCNC support for the HostMot2 firmware is split into a generic driver called *hostmot2* and two low-level I/O drivers for the Anything I/O boards. The low-level I/O drivers are *hm2_7i43* and *hm2_pci* (for all the PCI- and PC-104/Plus-based Anything I/O boards). The hostmot2 driver must be loaded first, using a HAL command like this:

```
loadrt hostmot2
```

See the hostmot2(9) man page for details.

The hostmot2 driver by itself does nothing, it needs access to actual boards running the HostMot2 firmware. The low-level I/O drivers provide this access. The low-level I/O drivers are loaded with commands like this:

```
loadrt hm2_pci config="firmware=hm2/5i20/SVST8_4.BIT
       num_encoders=3 num_pwmgens=3 num_stepgens=1"
```

The config parameters are described in the hostmot2 man page.

### 9.4.5 Watchdog

The HostMot2 firmware may include a watchdog Module; if it does, the hostmot2 driver will use it.

The watchdog must be petted by LinuxCNC periodically or it will bite. The hm2 write function (see below) pets the watchdog.

When the watchdog bites, all the board's I/O pins are disconnected from their Module instances and become high-impedance inputs (pulled high). The state of the HostMot2 firmware modules is not disturbed (except for the configuration of the I/O Pins). Encoder instances keep counting quadrature pulses, and pwm- and step-generators keep generating signals (which are not relayed to the motors, because the I/O Pins have become inputs).

Resetting the watchdog resets the I/O pins to the configuration chosen at load-time.

If the firmware includes a watchdog, the following HAL objects will be exported:

#### 9.4.5.1 Pins:

- *has_bit* - (bit i/o) True if the watchdog has bit, False if the watchdog has not bit. If the watchdog has bit and the has_bit bit is True, the user can reset it to False to resume operation.

#### 9.4.5.2 Parameters:

- *timeout_ns* - (u32 read/write) Watchdog timeout, in nanoseconds. This is initialized to 5,000,000 (5 milliseconds) at module load time. If more than this amount of time passes between calls to the hm2 write function, the watchdog will bite.

### 9.4.6 HostMot2 Functions

- *hm2_<BoardType>.<BoardNum>.read* - Read all inputs, update input HAL pins.
- *hm2_<BoardType>.<BoardNum>.write* - Write all outputs.
- *hm2_<BoardType>.<BoardNum>.read_gpio* - Read the GPIO input pins only. (This function is not available on the 7i43 due to limitations of the EPP bus.)

- *hm2_<BoardType>.<BoardNum>.write_gpio* - Write the GPIO control registers and output pins only. (This function is not available on the 7i43 due to limitations of the EPP bus.)

**Note**
The above *read_gpio* and *write_gpio* functions should not normally be needed, since the GPIO bits are read and written along with everything else in the standard *read* and *write* functions above, which are normally run in the servo thread.
The *read_gpio* and *write_gpio* functions were provided in case some very fast (frequently updated) I/O is needed. These functions should be run in the base thread. If you have need for this, please send an email and tell us about it, and what your application is.

### 9.4.7 Pinouts

The hostmot2 driver does not have a particular pinout. The pinout comes from the firmware that the hostmot2 driver sends to the Anything I/O board. Each firmware has different pinout, and the pinout depends on how many of the available encoders, pwmgens, and stepgens are used. To get a pinout list for your configuration after loading LinuxCNC in the terminal window type:

```
dmesg > hm2.txt
```

The resulting text file will contain lots of information as well as the pinout for the HostMot2 and any error and warning messages.

To reduce the clutter by clearing the message buffer before loading LinuxCNC type the following in the terminal window:

```
sudo dmesg -c
```

Now when you run LinuxCNC and then do a *dmesg > hm2.txt* in the terminal only the info from the time you loaded LinuxCNC will be in your file along with your pinout. The file will be in the current directory of the terminal window. Each line will contain the card name, the card number, the I/O Pin number, the connector and pin, and the usage. From this printout you will know the physical connections to your card based on your configuration.

An example of a 5i20 configuration:

```
[HOSTMOT2]
DRIVER=hm2_pci
BOARD=5i20
CONFIG="firmware=hm2/5i20/SVST8_4.BIT num_encoders=1 num_pwmgens=1 num_stepgens=3"
```

The above configuration produced this printout.

```
[ 1141.053386] hm2/hm2_5i20.0: 72 I/O Pins used:
[ 1141.053394] hm2/hm2_5i20.0: IO Pin 000 (P2-01): IOPort
[ 1141.053397] hm2/hm2_5i20.0: IO Pin 001 (P2-03): IOPort
[ 1141.053401] hm2/hm2_5i20.0: IO Pin 002 (P2-05): Encoder #0, pin B (Input)
[ 1141.053405] hm2/hm2_5i20.0: IO Pin 003 (P2-07): Encoder #0, pin A (Input)
[ 1141.053408] hm2/hm2_5i20.0: IO Pin 004 (P2-09): IOPort
[ 1141.053411] hm2/hm2_5i20.0: IO Pin 005 (P2-11): Encoder #0, pin Index (Input)
[ 1141.053415] hm2/hm2_5i20.0: IO Pin 006 (P2-13): IOPort
[ 1141.053418] hm2/hm2_5i20.0: IO Pin 007 (P2-15): PWMGen #0, pin Out0 (PWM or Up) (Output)
[ 1141.053422] hm2/hm2_5i20.0: IO Pin 008 (P2-17): IOPort
[ 1141.053425] hm2/hm2_5i20.0: IO Pin 009 (P2-19): PWMGen #0, pin Out1 (Dir or Down) ( ↩
    Output)
[ 1141.053429] hm2/hm2_5i20.0: IO Pin 010 (P2-21): IOPort
[ 1141.053432] hm2/hm2_5i20.0: IO Pin 011 (P2-23): PWMGen #0, pin Not-Enable (Output)
<snip>...
[ 1141.053589] hm2/hm2_5i20.0: IO Pin 060 (P4-25): StepGen #2, pin Step (Output)
[ 1141.053593] hm2/hm2_5i20.0: IO Pin 061 (P4-27): StepGen #2, pin Direction (Output)
[ 1141.053597] hm2/hm2_5i20.0: IO Pin 062 (P4-29): StepGen #2, pin (unused) (Output)
[ 1141.053601] hm2/hm2_5i20.0: IO Pin 063 (P4-31): StepGen #2, pin (unused) (Output)
[ 1141.053605] hm2/hm2_5i20.0: IO Pin 064 (P4-33): StepGen #2, pin (unused) (Output)
```

```
[ 1141.053609] hm2/hm2_5i20.0: IO Pin 065 (P4-35): StepGen #2, pin (unused) (Output)
[ 1141.053613] hm2/hm2_5i20.0: IO Pin 066 (P4-37): IOPort
[ 1141.053616] hm2/hm2_5i20.0: IO Pin 067 (P4-39): IOPort
[ 1141.053619] hm2/hm2_5i20.0: IO Pin 068 (P4-41): IOPort
[ 1141.053621] hm2/hm2_5i20.0: IO Pin 069 (P4-43): IOPort
[ 1141.053624] hm2/hm2_5i20.0: IO Pin 070 (P4-45): IOPort
[ 1141.053627] hm2/hm2_5i20.0: IO Pin 071 (P4-47): IOPort
[ 1141.053811] hm2/hm2_5i20.0: registered
[ 1141.053815] hm2_5i20.0: initialized AnyIO board at 0000:02:02.0
```

**Note**
That the I/O Pin nnn will correspond to the pin number shown on the HAL Configuration screen for GPIOs. Some of the Stepgen, Encoder and PWMGen will also show up as GPIOs in the HAL Configuration screen.

### 9.4.8 PIN Files

The default pinout is described in a .PIN file (human-readable text). When you install a firmware package the .PIN files are installed in

```
/usr/share/doc/hostmot2-firmware-<board>/
```

### 9.4.9 Firmware

The selected firmware (.BIT file) and configuration is uploaded from the PC motherboard to the Mesa mothercard on LinuxCNC startup. If you are using Run In Place, you must still install a hostmot2-firmware-<board> package. There is more information about firmware and configuration in the *Configurations* section.

### 9.4.10 HAL Pins

The HAL pins for each configuration can be seen by opening up *Show HAL Configuration* from the Machine menu. All the HAL pins and parameters can be found there. The following figure is of the 5i20 configuration used above.

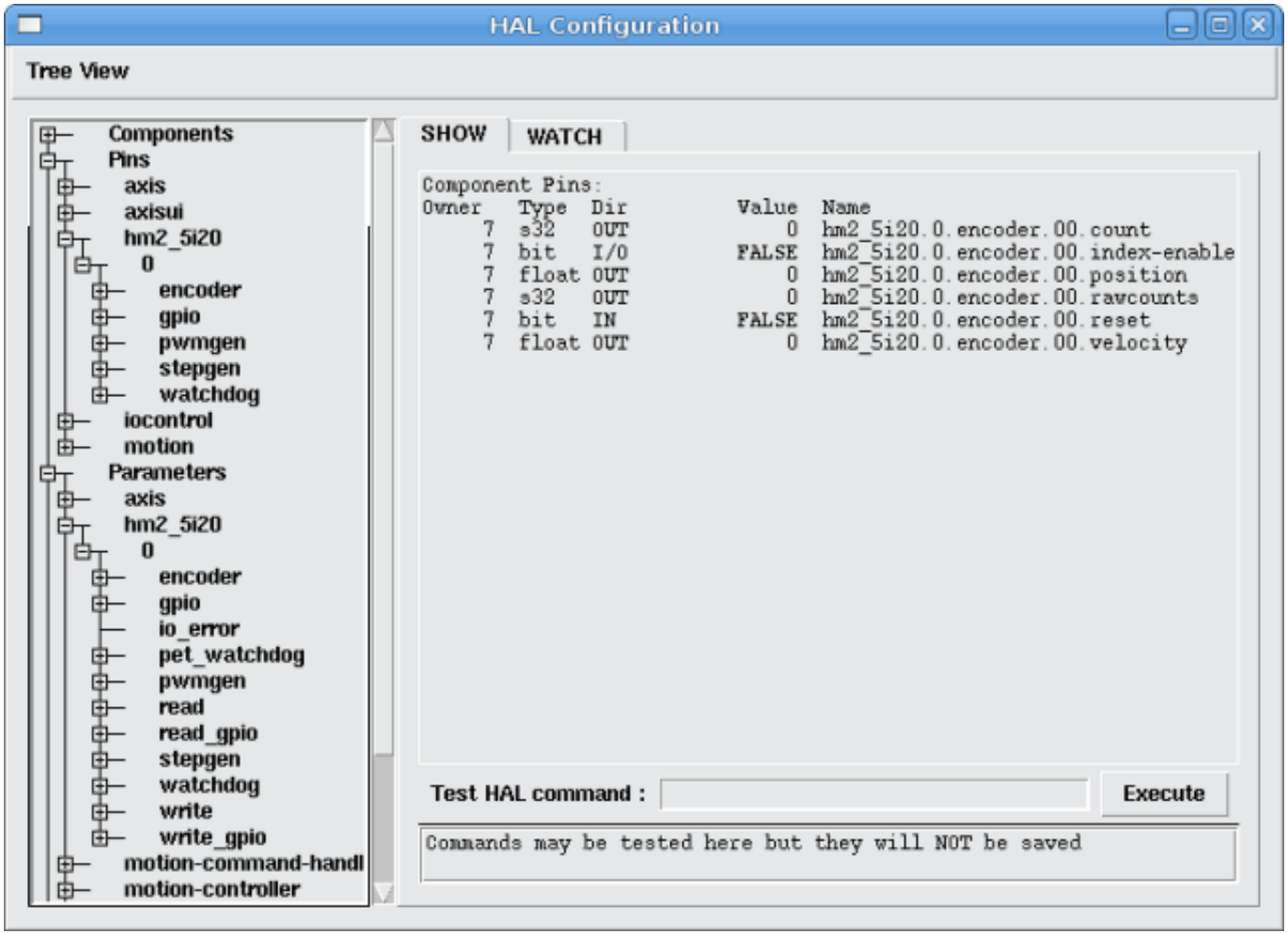



Figure 9.1: 5i20 HAL Pins

### 9.4.11 Configurations

The Hostmot2 firmware is available in several versions, depending on what you are trying to accomplish. You can get a reminder of what a particular firmware is for by looking at the name. Let's look at a couple of examples.

In the 7i43 (two ports), SV8 (*Servo 8*) would be for having 8 servos or fewer, using the *classic* 7i33 4-axis (per port) servo board. So 8 servos would use up all 48 signals in the two ports. But if you only needed 3 servos, you could say *num_encoders=3* and *num_pwmgens=3* and recover 5 servos at 6 signals each, thus gaining 30 bits of GPIO.

Or, in the 5i22 (four ports), SVST8_24 (*Servo 8, Stepper 24*) would be for having 8 servos or fewer (7i33 x2 again), and 24 steppers or fewer (7i47 x2). This would use up all four ports. If you only needed 4 servos you could say *num_encoders=4* and *num_pwmgens=4* and recover 1 port (and save a 7i33). And if you only needed 12 steppers you could say *num_stepgens=12* and free up one port (and save a 7i47). So in this way we can save two ports (48 bits) for GPIO.

Here are tables of the firmwares available in the official packages. There may be additional firmwares available at the Mesanet.com website that have not yet made it into the LinuxCNC official firmware packages, so check there too.

3x20 (6-port various) Default Configurations (The 3x20 comes in 1M, 1.5M, and 2M gate versions. So far, all firmware is available in all gate sizes.)

| Firmware | Encoder | PWMGen | StepGen | GPIO |
|---|---|---|---|---|
| SV24 | 24 | 24 | 0 | 0 |
| SVST16_24 | 16 | 16 | 24 | 0 |

5i22 (4-port PCI) Default Configurations (The 5i22 comes in 1M and 1.5M gate versions. So far, all firmware is available in all gate sizes.)

| Firmware | Encoder | PWM | StepGen | GPIO |
|---|---|---|---|---|
| SV16 | 16 | 16 | 0 | 0 |
| SVST2_4_7I47 | 4 | 2 | 4 | 72 |
| SVST8_8 | 8 | 8 | 8 | 0 |
| SVST8_24 | 8 | 8 | 24 | 0 |

5i23 (3-port PCI) Default Configurations (The 5i23 has 400k gates.)

| Firmware | Encoder | PWM | StepGen | GPIO |
|---|---|---|---|---|
| SV12 | 12 | 12 | 0 | 0 |
| SVST2_8 | 2 | 2 | 8 (tbl5) | 12 |
| SVST2_4_7I47 | 4 | 2 | 4 | 48 |
| SV12_2X7I48_72 | 12 | 12 | 0 | 24 |
| SV12IM_2X7I48_72 | 12 (+IM) | 12 | 0 | 12 |
| SVST4_8 | 4 | 4 | 8 (tbl5) | 0 |
| SVST8_4 | 8 | 8 | 4 (tbl5) | 0 |
| SVST8_4IM2 | 8 (+IM) | 8 | 4 | 8 |
| SVST8_8IM2 | 8 (+IM) | 8 | 8 | 0 |
| SVTP6_7I39 | 6 | 0 (6 BLDC) | 0 | 0 |

5i20 (3-port PCI) Default Configurations (The 5i20 has 200k gates.)

| Firmware | Encoder | PWM | StepGen | GPIO |
|---|---|---|---|---|
| SV12 | 12 | 12 | 0 | 0 |
| SVST2_8 | 2 | 2 | 8 (tbl5) | 12 |
| SVST2_4_7I47 | 4 | 2 | 4 | 48 |
| SV12_2X7I48_72 | 12 | 12 | 0 | 24 |
| SV12IM_2X7I48_72 | 12 (+IM) | 12 | 0 | 12 |
| SVST8_4 | 8 | 8 | 4 (tbl5) | 0 |
| SVST8_4IM2 | 8 (+IM) | 8 | 4 | 8 |

4i68 (3-port PC/104) Default Configurations (The 4i68 has 400k gates.)

| Firmware | Encoder | PWM | StepGen | GPIO |
|---|---|---|---|---|
| SV12 | 12 | 12 | 0 | 0 |
| SVST2_4_7I47 | 4 | 2 | 4 | 48 |
| SVST4_8 | 4 | 4 | 8 | 0 |
| SVST8_4 | 8 | 8 | 4 | 0 |
| SVST8_4IM2 | 8 (+IM) | 8 | 4 | 8 |
| SVST8_8IM2 | 8 (+IM) | 8 | 8 | 0 |

4i65 (3-port PC/104) Default Configurations (The 4i65 has 200k gates.)

| Firmware | Encoder | PWM | StepGen | GPIO |
|---|---|---|---|---|
| SV12 | 12 | 12 | 0 | 0 |
| SVST8_4 | 8 | 8 | 4 | 0 |
| SVST8_4IM2 | 8 (+IM) | 8 | 4 | 8 |

7i43 (2-port parallel) 400k gate versions, Default Configurations

| Firmware | Encoder | PWM | StepGen | GPIO |
|---|---|---|---|---|
| SV8 | 8 | 8 | 0 | 0 |
| SVST4_4 | 4 | 4 | 4 (tbl5) | 0 |
| SVST4_6 | 4 | 4 | 6 (tbl3) | 0 |
| SVST4_12 | 4 | 4 | 12 | 0 |
| SVST2_4_7I47 | 4 | 2 | 4 | 24 |

7i43 (2-port parallel) 200k gate versions, Default Configurations

| Firmware | Encoder | PWM | StepGen | GPIO |
|---|---|---|---|---|
| SV8 | 8 | 8 | 0 | 0 |
| SVST4_4 | 4 | 4 | 4 (tbl5) | 0 |
| SVST4_6 | 4 | 4 | 6 (tbl3) | 0 |
| SVST2_4_7I47 | 4 | 2 | 4 | 24 |

Even though several cards may have the same named .BIT file you cannot use a .BIT file that is not for that card. Different cards have different clock frequencies so make sure you load the proper .BIT file for your card. Custom hm2 firmwares can be created for special applications and you may see some custom hm2 firmwares in the directories with the default ones.

When you load the board-driver (hm2_pci or hm2_7i43), you can tell it to disable instances of the three primary modules (pwmgen, stepgen, and encoder) by setting the count lower. Any I/O pins belonging to disabled module instances become GPIOs.

### 9.4.12 GPIO

General Purpose I/O pins on the board which are not used by a module instance are exported to HAL as *full* GPIO pins. Full GPIO pins can be configured at run-time to be inputs, outputs, or open drains, and have a HAL interface that exposes this flexibility. I/O pins that are owned by an active module instance are constrained by the requirements of the owning module, and have a restricted HAL interface.

GPIOs have names like *hm2_<BoardType>.<BoardNum>.gpio.<IONum>*. IONum. is a three-digit number. The mapping from IONum to connector and pin-on-that-connector is written to the syslog when the driver loads, and it's documented in Mesa's manual for the Anything I/O boards.

The hm2 GPIO representation is modeled after the Digital Inputs and Digital Outputs described in the Canonical Device Interface (part of the HAL General Reference document).

GPIO pins default to input.

#### 9.4.12.1 Pins

- *in* - (Bit, Out) Normal state of the hardware input pin. Both full GPIO pins and I/O pins used as inputs by active module instances have this pin.
- *in_not* - (Bit, Out) Inverted state of the hardware input pin. Both full GPIO pins and I/O pins used as inputs by active module instances have this pin.
- *out* - (Bit, In) Value to be written (possibly inverted) to the hardware output pin. Only full GPIO pins have this pin.

#### 9.4.12.2 Parameters

- *invert_output* - (Bit, RW) This parameter only has an effect if the *is_output* parameter is true. If this parameter is true, the output value of the GPIO will be the inverse of the value on the *out* HAL pin. Only full GPIO pins and I/O pins used as outputs by active module instances have this parameter. To invert an active module pin you have to invert the GPIO pin not the module pin.

- *is_opendrain* - (Bit, RW) This parameter only has an effect if the *is_output* parameter is true. If this parameter is false, the GPIO behaves as a normal output pin: the I/O pin on the connector is driven to the value specified by the *out* HAL pin (possibly inverted), and the value of the *in* and *in_not* HAL pins is undefined. If this parameter is true, the GPIO behaves as an open-drain pin. Writing 0 to the *out* HAL pin drives the I/O pin low, writing 1 to the *out* HAL pin puts the I/O pin in a high-impedance state. In this high-impedance state the I/O pin floats (weakly pulled high), and other devices can drive the value; the resulting value on the I/O pin is available on the *in* and *in_not* pins. Only full GPIO pins and I/O pins used as outputs by active module instances have this parameter.
- *is_output* - (Bit, RW) If set to 0, the GPIO is an input. The I/O pin is put in a high-impedance state (weakly pulled high), to be driven by other devices. The logic value on the I/O pin is available in the *in* and *in_not* HAL pins. Writes to the *out* HAL pin have no effect. If this parameter is set to 1, the GPIO is an output; its behavior then depends on the *is_opendrain* parameter. Only full GPIO pins have this parameter.

### 9.4.13 StepGen

Stepgens have names like *hm2_<BoardType>.<BoardNum>.stepgen.<Instance>.*. *Instance* is a two-digit number that corresponds to the HostMot2 stepgen instance number. There are *num_stepgens* instances, starting with 00.

Each stepgen allocates 2-6 I/O pins (selected at firmware compile time), but currently only uses two: Step and Direction outputs.[2]

The stepgen representation is modeled on the stepgen software component. Stepgen default is active high step output (high during step time low during step space). To invert a StepGen output pin you invert the corresponding GPIO pin that is being used by StepGen. To find the GPIO pin being used for the StepGen output run dmesg as shown above.

Each stepgen instance has the following pins and parameters:

#### 9.4.13.1 Pins

- *control-type* - (Bit, In) Switches between position control mode (0) and velocity control mode (1). Defaults to position control (0).
- *counts* - (s32, Out) Feedback position in counts (number of steps).
- *enable* - (Bit, In) Enables output steps. When false, no steps are generated.
- *position-cmd* - (Float, In) Target position of stepper motion, in user-defined position units.
- *position-fb* - (Float, Out) Feedback position in user-defined position units (counts / position_scale).
- *velocity-cmd* - (Float, In) Target velocity of stepper motion, in user-defined position units per second. This pin is only used when the stepgen is in velocity control mode (control-type=1).
- *velocity-fb* - (Float, Out) Feedback velocity in user-defined position units per second.

#### 9.4.13.2 Parameters

- *dirhold* - (u32, RW) Minimum duration of stable Direction signal after a step ends, in nanoseconds.
- *dirsetup* - (u32, RW) Minimum duration of stable Direction signal before a step begins, in nanoseconds.
- *maxaccel* - (Float, RW) Maximum acceleration, in position units per second per second. If set to 0, the driver will not limit its acceleration.
- *maxvel* - (Float, RW) Maximum speed, in position units per second. If set to 0, the driver will choose the maximum velocity based on the values of steplen and stepspace (at the time that maxvel was set to 0).
- *position-scale* - (Float, RW) Converts from counts to position units. position = counts / position_scale

[2] At present, the firmware supports multi-phase stepper outputs, but the driver doesn't. Interested volunteers are solicited.

- *step_type* - (u32, RW) Output format, like the step_type modparam to the software stegen(9) component. 0 = Step/Dir, 1 = Up/Down, 2 = Quadrature. In Quadrature mode (step_type=2), the stepgen outputs one complete Gray cycle (00 -> 01 -> 11 -> 10 -> 00) for each *step* it takes.
- *steplen* - (u32, RW) Duration of the step signal, in nanoseconds.
- *stepspace* - (u32, RW) Minimum interval between step signals, in nanoseconds.

#### 9.4.13.3 Output Parameters

The Step and Direction pins of each StepGen have two additional parameters. To find which I/O pin belongs to which step and direction output run dmesg as described above.

- *invert_output* - (Bit, RW) This parameter only has an effect if the *is_output* parameter is true. If this parameter is true, the output value of the GPIO will be the inverse of the value on the *out* HAL pin.
- *is_opendrain* - (Bit, RW) If this parameter is false, the GPIO behaves as a normal output pin: the I/O pin on the connector is driven to the value specified by the *out* HAL pin (possibly inverted). If this parameter is true, the GPIO behaves as an open-drain pin. Writing 0 to the *out* HAL pin drives the I/O pin low, writing 1 to the *out* HAL pin puts the I/O pin in a high-impedance state. In this high-impedance state the I/O pin floats (weakly pulled high), and other devices can drive the value; the resulting value on the I/O pin is available on the *in* and *in_not* pins. Only full GPIO pins and I/O pins used as outputs by active module instances have this parameter.

### 9.4.14 PWMGen

PWMgens have names like *hm2_<BoardType>.<BoardNum>.pwmgen.<Instance>.*. *Instance* is a two-digit number that corresponds to the HostMot2 pwmgen instance number. There are *num_pwmgens* instances, starting with 00.

In HM2, each pwmgen uses three output I/O pins: Not-Enable, Out0, and Out1. To invert a PWMGen output pin you invert the corresponding GPIO pin that is being used by PWMGen. To find the GPIO pin being used for the PWMGen output run dmesg as shown above.

The function of the Out0 and Out1 I/O pins varies with output-type parameter (see below).

The hm2 pwmgen representation is similar to the software pwmgen component. Each pwmgen instance has the following pins and parameters:

#### 9.4.14.1 Pins

- *enable* - (Bit, In) If true, the pwmgen will set its Not-Enable pin false and output its pulses. If *enable* is false, pwmgen will set its Not-Enable pin true and not output any signals.
- *value* - (Float, In) The current pwmgen command value, in arbitrary units.

#### 9.4.14.2 Parameters

- *output-type* - (s32, RW) This emulates the output_type load-time argument to the software pwmgen component. This parameter may be changed at runtime, but most of the time you probably want to set it at startup and then leave it alone. Accepted values are 1 (PWM on Out0 and Direction on Out1), 2 (Up on Out0 and Down on Out1), 3 (PDM mode, PDM on Out0 and Dir on Out1), and 4 (Direction on Out0 and PWM on Out1, *for locked antiphase*).
- *scale* - (Float, RW) Scaling factor to convert *value* from arbitrary units to duty cycle: dc = value / scale. Duty cycle has an effective range of -1.0 to +1.0 inclusive, anything outside that range gets clipped.

- *pdm_frequency* - (u32, RW) This specifies the PDM frequency, in Hz, of all the pwmgen instances running in PDM mode (mode 3). This is the *pulse slot frequency*; the frequency at which the pdm generator in the Anything I/O board chooses whether to emit a pulse or a space. Each pulse (and space) in the PDM pulse train has a duration of 1/pdm_frequency seconds. For example, setting the pdm_frequency to 2e6 (2 MHz) and the duty cycle to 50% results in a 1 MHz square wave, identical to a 1 MHz PWM signal with 50% duty cycle. The effective range of this parameter is from about 1525 Hz up to just under 100 MHz. Note that the max frequency is determined by the ClockHigh frequency of the Anything I/O board; the 5i20 and 7i43 both have a 100 MHz clock, resulting in a 100 Mhz max PDM frequency. Other boards may have different clocks, resulting in different max PDM frequencies. If the user attempts to set the frequency too high, it will be clipped to the max supported frequency of the board.
- *pwm_frequency* - (u32, RW) This specifies the PWM frequency, in Hz, of all the pwmgen instances running in the PWM modes (modes 1 and 2). This is the frequency of the variable-duty-cycle wave. Its effective range is from 1 Hz up to 193 KHz. Note that the max frequency is determined by the ClockHigh frequency of the Anything I/O board; the 5i20 and 7i43 both have a 100 MHz clock, resulting in a 193 KHz max PWM frequency. Other boards may have different clocks, resulting in different max PWM frequencies. If the user attempts to set the frequency too high, it will be clipped to the max supported frequency of the board. Frequencies below about 5 Hz are not terribly accurate, but above 5 Hz they're pretty close.

#### 9.4.14.3 Output Parameters

The output pins of each PWMGen have two additional parameters. To find which I/O pin belongs to which output run dmesg as described above.

- *invert_output* - (Bit, RW) This parameter only has an effect if the *is_output* parameter is true. If this parameter is true, the output value of the GPIO will be the inverse of the value on the *out* HAL pin.
- *is_opendrain* - (Bit, RW) If this parameter is false, the GPIO behaves as a normal output pin: the I/O pin on the connector is driven to the value specified by the *out* HAL pin (possibly inverted). If this parameter is true, the GPIO behaves as an open-drain pin. Writing 0 to the *out* HAL pin drives the I/O pin low, writing 1 to the *out* HAL pin puts the I/O pin in a high-impedance state. In this high-impedance state the I/O pin floats (weakly pulled high), and other devices can drive the value; the resulting value on the I/O pin is available on the *in* and *in_not* pins. Only full GPIO pins and I/O pins used as outputs by active module instances have this parameter.

### 9.4.15 Encoder

Encoders have names like *hm2_<BoardType>.<BoardNum>.encoder.<Instance>.*. *Instance* is a two-digit number that corresponds to the HostMot2 encoder instance number. There are *num_encoders* instances, starting with 00.

Each encoder uses three or four input I/O pins, depending on how the firmware was compiled. Three-pin encoders use A, B, and Index (sometimes also known as Z). Four-pin encoders use A, B, Index, and Index-mask.

The hm2 encoder representation is similar to the one described by the Canonical Device Interface (in the HAL General Reference document), and to the software encoder component. Each encoder instance has the following pins and parameters:

#### 9.4.15.1 Pins

- *count* - (s32, Out) Number of encoder counts since the previous reset.
- *index-enable* - (Bit, I/O) When this pin is set to True, the count (and therefore also position) are reset to zero on the next Index (Phase-Z) pulse. At the same time, index-enable is reset to zero to indicate that the pulse has occurred.
- *position* - (Float, Out) Encoder position in position units (count / scale).
- *rawcounts* - (s32, Out) Total number of encoder counts since the start, not adjusted for index or reset.
- *reset* - (Bit, In) When this pin is TRUE, the count and position pins are set to 0. (The value of the velocity pin is not affected by this.) The driver does not reset this pin to FALSE after resetting the count to 0, that is the user's job.
- *velocity* - (Float, Out) Estimated encoder velocity in position units per second.

#### 9.4.15.2 Parameters

- *counter-mode* - (Bit, RW) Set to False (the default) for Quadrature. Set to True for Up/Down or for single input on Phase A. Can be used for a frequency to velocity converter with a single input on Phase A when set to true.
- *filter* - (Bit, RW) If set to True (the default), the quadrature counter needs 15 clocks to register a change on any of the three input lines (any pulse shorter than this is rejected as noise). If set to False, the quadrature counter needs only 3 clocks to register a change. The encoder sample clock runs at 33 MHz on the PCI Anything I/O cards and 50 MHz on the 7i43.
- *index-invert* - (Bit, RW) If set to True, the rising edge of the Index input pin triggers the Index event (if index-enable is True). If set to False, the falling edge triggers.
- *index-mask* - (Bit, RW) If set to True, the Index input pin only has an effect if the Index-Mask input pin is True (or False, depending on the index-mask-invert pin below).
- *index-mask-invert* - (Bit, RW) If set to True, Index-Mask must be False for Index to have an effect. If set to False, the Index-Mask pin must be True.
- *scale* - (Float, RW) Converts from *count* units to *position* units. A quadrature encoder will normally have 4 counts per pulse so a 100 PPR encoder would be 400 counts per revolution. In *.counter-mode* a 100 PPR encoder would have 100 counts per revelution as it only uses the rising edge of A and direction is B.
- *vel-timeout* - (Float, RW) When the encoder is moving slower than one pulse for each time that the driver reads the count from the FPGA (in the hm2_read() function), the velocity is harder to estimate. The driver can wait several iterations for the next pulse to arrive, all the while reporting the upper bound of the encoder velocity, which can be accurately guessed. This parameter specifies how long to wait for the next pulse, before reporting the encoder stopped. This parameter is in seconds.

### 9.4.16 5i25 Configuration

#### 9.4.16.1 Firmware

The 5i25 firmware comes preloaded for the daughter card it is purchased with. So the *firmware=xxx.BIT* is not part of the hm2_pci configuration string when using a 5i25.

#### 9.4.16.2 Configuration

Example configurations of the 5i25/7i76 and 5i25/7i77 cards are included in the Configuration Selector.

If you like to roll your own configuration the following examples show how to load the drivers in the HAL file.

**5i25 + 7i76 Card**

```
# load the generic driver
loadrt hostmot2

# load the PCI driver and configure
loadrt hm2_pci config="num_encoders=1 num_stepgens=5 sserial_port_0=0XXX"
```

**5i25 + 7i77 Card**

```
# load the generic driver
loadrt hostmot2

# load the PCI driver and configure
loadrt hm2_pci config="num_encoders=6 num_pwmgens=6 sserial_port_0=0XXX"
```

#### 9.4.16.3 SSERIAL Configuration

The *sserial_port_0=0XXX* configuration string sets some options for the smart serial daughter card. These options are specific for each daughter card. See the Mesa manual for more information on the exact usuage.

#### 9.4.16.4 7i77 Limits

The minlimit and maxlimit are bounds on the pin value (in this case the analog out value) fullscalemax is the scale factor.

These are by default set to the analog in or analog range (most likely in volts).

So for example on the 7I77 +-10V analog outputs, the default values are:

minlimit -10 maxlimit +10 maxfullscale 10

If you wanted to say scale the analog out of a channel to IPS for a velocity mode servo (say 24 IPS max) you could set the limits like this:

minlimit -24 maxlimit +24 maxfullscale 24

If you wanted to scale the analog out of a channel to RPM for a 0 to 6000 RPM spindle with 0-10V control you could set the limits like this:

minlimit 0 maxlimit 6000 maxfullscale 6000 (this would prevent unwanted negative output voltages from being set)

### 9.4.17 Example Configurations

Several example configurations for Mesa hardware are included with LinuxCNC. The configurations are located in the hm2-servo and hm2-stepper sections of the Configuration Selector. Typically you will need the board installed for the configuration you pick to load. The examples are a good place to start and will save you time. Just pick the proper example from the LinuxCNC Configuration Selector and save a copy to your computer so you can edit it. To see the exact pins and parameters that your configuration gave you, open the Show HAL Configuration window from the Machine menu, or do dmesg as outlined above.

## 9.5 Motenc Driver

Vital Systems Motenc-100 and Motenc-LITE

The Vital Systems Motenc-100 and Motenc-LITE are 8- and 4-channel servo control boards. The Motenc-100 provides 8 quadrature encoder counters, 8 analog inputs, 8 analog outputs, 64 (68?) digital inputs, and 32 digital outputs. The Motenc-LITE has only 4 encoder counters, 32 digital inputs and 16 digital outputs, but it still has 8 analog inputs and 8 analog outputs. The driver automatically identifies the installed board and exports the appropriate HAL objects.

Installing:

```
loadrt hal_motenc
```

During loading (or attempted loading) the driver prints some useful debugging messages to the kernel log, which can be viewed with dmesg.

Up to 4 boards may be used in one system.

### 9.5.1 Pins

In the following pins, parameters, and functions, <board> is the board ID. According to the naming conventions the first board should always have an ID of zero. However this driver sets the ID based on a pair of jumpers on the board, so it may be non-zero even if there is only one board.

- *(s32) motenc.<board>.enc-<channel>-count* - Encoder position, in counts.
- *(float) motenc.<board>.enc-<channel>-position* - Encoder position, in user units.
- *(bit) motenc.<board>.enc-<channel>-index* - Current status of index pulse input.
- *(bit) motenc.<board>.enc-<channel>-idx-latch* - Driver sets this pin true when it latches an index pulse (enabled by latch-index). Cleared by clearing latch-index.

- *(bit) motenc.<board>.enc-<channel>-latch-index* - If this pin is true, the driver will reset the counter on the next index pulse.
- *(bit) motenc.<board>.enc-<channel>-reset-count* - If this pin is true, the counter will immediately be reset to zero, and the pin will be cleared.
- *(float) motenc.<board>.dac-<channel>-value* - Analog output value for DAC (in user units, see -gain and -offset)
- *(float) motenc.<board>.adc-<channel>-value* - Analog input value read by ADC (in user units, see -gain and -offset)
- *(bit) motenc.<board>.in-<channel>* - State of digital input pin, see canonical digital input.
- *(bit) motenc.<board>.in-<channel>-not* - Inverted state of digital input pin, see canonical digital input.
- *(bit) motenc.<board>.out-<channel>* - Value to be written to digital output, seen canonical digital output.
- *(bit) motenc.<board>.estop-in* - Dedicated estop input, more details needed.
- *(bit) motenc.<board>.estop-in-not* - Inverted state of dedicated estop input.
- *(bit) motenc.<board>.watchdog-reset* - Bidirectional, - Set TRUE to reset watchdog once, is automatically cleared.

### 9.5.2 Parameters

- *(float) motenc.<board>.enc-<channel>-scale* - The number of counts / user unit (to convert from counts to units).
- *(float) motenc.<board>.dac-<channel>-offset* - Sets the DAC offset.
- *(float) motenc.<board>.dac-<channel>-gain* - Sets the DAC gain (scaling).
- *(float) motenc.<board>.adc-<channel>-offset* - Sets the ADC offset.
- *(float) motenc.<board>.adc-<channel>-gain* - Sets the ADC gain (scaling).
- *(bit) motenc.<board>.out-<channel>-invert* - Inverts a digital output, see canonical digital output.
- *(u32) motenc.<board>.watchdog-control* - Configures the watchdog. The value may be a bitwise OR of the following values:

| Bit # | Value | Meaning |
|---|---|---|
| 0 | 1 | Timeout is 16ms if set, 8ms if unset |
| 1 | 2 | |
| 2 | 4 | Watchdog is enabled |
| 3 | 8 | |
| 4 | 16 | Watchdog is automatically reset by DAC writes (the HAL dac-write function) |

Typically, the useful values are 0 (watchdog disabled) or 20 (8ms watchdog enabled, cleared by dac-write).

- *(u32) motenc.<board>.led-view* - Maps some of the I/O to onboard LEDs.

### 9.5.3 Functions

- *(funct) motenc.<board>.encoder-read* - Reads all encoder counters.
- *(funct) motenc.<board>.adc-read* - Reads the analog-to-digital converters.
- *(funct) motenc.<board>.digital-in-read* - Reads digital inputs.
- *(funct) motenc.<board>.dac-write* - Writes the voltages to the DACs.
- *(funct) motenc.<board>.digital-out-write* - Writes digital outputs.
- *(funct) motenc.<board>.misc-update* - Updates misc stuff.

## 9.6 MB2HAL

MB2HAL is a generic userspace HAL component to communicate with one or more Modbus devices. So far, there are two options to comunicate with a Modbus device:

1. One option is to create a HAL component as a driver see VFD Modbus.
2. Another option is to use Classic Ladder which has Modbus built in see ClassicLadder.
3. Now there is a third option that consists of a "generic" driver configured by text file, this is called MB2HAL. Why MB2HAL ? Consider using Mb2hal if:
   - You have to write a new driver and you don't know anything about programming.
   - You need to use Classic Ladder "only" to manage the Modbus connections.
   - You have to discover and configure first time the Modbus transactions. Mb2hal have debug levels to facilitate the low level protocol debug.
   - You have more than one device to connect. Mb2hal is very efficient managing multiple devices, transactions and links. Currently i am monitoring two axis drivers using a Rs232 port, a VFD driver using another Rs232 port, and a remote I/O using TCP/IP.
   - You want a protocol to connect your Arduino to HAL. Look the included sample configuration file, sketch and library for Arduino Modbus.

### 9.6.1 Example config file.

```
#This .INI file is also the HELP, MANUAL and HOW-TO file for mb2hal.

#Load the modbus HAL userspace module as the examples below,
#change to match your own HAL_MODULE_NAME and .ini file name
#Using HAL_MODULE_NAME=mb2hal or nothing (default): loadusr -W mb2hal config=config_file. ↩
    ini
#Using HAL_MODULE_NAME=mymodule: loadusr -Wn mymodule mb2hal config=config_file.ini

#Common section

[MB2HAL_INIT]

#OPTIONAL: Debug level of init and INI file parsing.
# 0 = silent.
# 1 = error messages (default).
# 2 = OK confirmation messages.
# 3 = debugging messages.

INIT_DEBUG=3

#OPTIONAL: HAL module (component) name. Defaults to "mb2hal".

HAL_MODULE_NAME=mb2hal

#OPTIONAL: Insert a delay of "FLOAT seconds" between transactions in order
#to not to have a lot of logging and facilitate the debugging.
#Useful when using DEBUG=3 (NOT INIT_DEBUG=3)
#It affects ALL transactions.
#Use "0.0" for normal activity.

SLOWDOWN=0.0

#REQUIRED: The number of total Modbus transactions. There is no maximum.
```

```
TOTAL_TRANSACTIONS=9

#One transaction section is required per transaction, starting at 00 and counting up ↩
    sequentially.
#If there is a new link (not transaction), you must provide the REQUIRED parameters 1st ↩
    time.
#Warning: Any OPTIONAL parameter not specified are copied from the previous transaction.

[TRANSACTION_00]

#REQUIRED: You must specify either a "serial" or "tcp" link for the first transaction.
#Later transaction will use the previous transaction link if not specified.

LINK_TYPE=tcp

#if LINK_TYPE=tcp then REQUIRED (only 1st time): The Modbus slave device ip address.
#if LINK_TYPE=serial then IGNORED

TCP_IP=192.168.2.10

#if LINK_TYPE=tcp then OPTIONAL.
#if LINK_TYPE=serial then IGNORED
#The Modbus slave device tcp port. Defaults to 502.

TCP_PORT=502

#if LINK_TYPE=serial then REQUIRED (only 1st time).
#if LINK_TYPE=tcp then IGNORED
#The serial port.

SERIAL_PORT=/dev/ttyS0

#if LINK_TYPE=serial then REQUIRED (only 1st time).
#if LINK_TYPE=tcp then IGNORED
#The baud rate.

SERIAL_BAUD=115200

#if LINK_TYPE=serial then REQUIRED (only 1st time).
#if LINK_TYPE=tcp then IGNORED
#Data bits. One of 5,6,7,8.

SERIAL_BITS=8

#if LINK_TYPE=serial then REQUIRED (only 1st time).
#if LINK_TYPE=tcp then IGNORED
#Data parity. One of: even, odd, none.

SERIAL_PARITY=none

#if LINK_TYPE=serial then REQUIRED (only 1st time).
#if LINK_TYPE=tcp then IGNORED
#Stop bits. One of 1, 2.

SERIAL_STOP=2

#if LINK_TYPE=serial then OPTIONAL:
#if LINK_TYPE=tcp then IGNORED
#Serial port delay between for this transaction only.
#In ms. Defaults to 0.

SERIAL_DELAY_MS=10
```

```
#REQUIRED (only 1st time).
#Modbus slave number.

MB_SLAVE_ID=1

#REQUIRED: The first element address.

FIRST_ELEMENT=0

#REQUIRED: The number of elements.

NELEMENTS=16

#REQUIRED: Modbus transaction function code (see www.modbus.org specifications).
#    fnct_02_read_discrete_inputs     (02 = 0x02)
#    fnct_03_read_holding_registers   (03 = 0x03)
#    fnct_04_read_input_registers     (04 = 0x04)
#    fnct_15_write_multiple_coils     (15 = 0x0F)
#    fnct_16_write_multiple_registers (16 = 0x10)

#fnct_02_read_discrete_inputs: creates boolean output HAL pins.
#fnct_03_read_holding_registers: creates a floating point output HAL pins.
#                            also creates a u32 output HAL pins.
#fnct_04_read_input_registers: creates a floating point output HAL pins.
#                          also creates a u32 output HAL pins.
#fnct_15_write_multiple_coils: creates boolean input HAL pins.
#fnct_16_write_multiple_registers: creates a floating point input HAL pins.

#The pins are named based on component name, transaction number and order number.
#Example: mb2hal.00.01 (transaction=00, second register=01 (00 is the first one))

MB_TX_CODE=fnct_03_read_holding_registers

#OPTIONAL: Response timeout for this transaction. In INTEGER ms. Defaults to 500 ms.
#This is how much to wait for 1st byte before raise an error.

MB_RESPONSE_TIMEOUT_MS=500

#OPTIONAL: Byte timeout for this transaction. In INTEGER ms. Defaults to 500 ms.
#This is how much to wait from byte to byte before raise an error.

MB_BYTE_TIMEOUT_MS=500

#OPTIONAL: Instead of giving the transaction number, use a name.
#Example: mb2hal.00.01 could become mb2hal.plcin.01
#The name must not exceed 32 characters.
#NOTE: when using names be careful that you dont end up with two transactions
#usign the same name.

HAL_TX_NAME=remoteIOcfg

#OPTIONAL: Maximum update rate in HZ. Defaults to 0.0 (0.0 = as soon as available = infinit ↩
    ).
#NOTE: This is a maximum rate and the actual rate may be lower.
#If you want to calculate it in ms use (1000 / required_ms).
#Example: 100 ms = MAX_UPDATE_RATE=10.0, because 1000.0 ms / 100.0 ms = 10.0 Hz

MAX_UPDATE_RATE=0.0

#OPTIONAL: Debug level for this transaction only.
#See INIT_DEBUG parameter above.
```

```
DEBUG=1

#While DEBUGGING transactions note the returned "ret[]" value correspond to:
#/* Modbus protocol exceptions */
#ILLEGAL_FUNCTION        -0x01 the FUNCTION code received in the query is not allowed or ↩
    invalid.
#ILLEGAL_DATA_ADDRESS    -0x02 the DATA ADDRESS received in the query is not an allowable ↩
    address for the slave or is invalid.
#ILLEGAL_DATA_VALUE      -0x03 a VALUE contained in the data query field is not an ↩
    allowable value or is invalid.
#SLAVE_DEVICE_FAILURE    -0x04 SLAVE (or MASTER) device unrecoverable FAILUER while ↩
    attemping to perform the requested action.
#SERVER_FAILURE          -0x04 (see above).
#ACKNOWLEDGE             -0x05 This response is returned to PREVENT A TIMEOUT in the master ↩
    .
#                              A long duration of time is required to process the request ↩
    in the slave.
#SLAVE_DEVICE_BUSY       -0x06 The slave (or server) is BUSY. Retrasmit the request later.
#SERVER_BUSY             -0x06 (see above).
#NEGATIVE_ACKNOWLEDGE    -0x07 Unsuccessful programming request using function code 13 or ↩
    14.
#MEMORY_PARITY_ERROR     -0x08 SLAVE parity error in MEMORY.
#GATEWAY_PROBLEM_PATH    -0x0A (-10) Gateway path(s) not available.
#GATEWAY_PROBLEM_TARGET  -0x0B (-11) The target device failed to repond (generated by ↩
    master, not slave).
#/* Program or connection */
#COMM_TIME_OUT           -0x0C (-12)
#PORT_SOCKET_FAILURE     -0x0D (-13)
#SELECT_FAILURE          -0x0E (-14)
#TOO_MANY_DATAS          -0x0F (-15)
#INVALID_CRC             -0x10 (-16)
#INVALID_EXCEPTION_CODE  -0x11 (-17)

[TRANSACTION_01]
MB_TX_CODE=fnct_02_read_discrete_inputs
FIRST_ELEMENT=1024
NELEMENTS=24
HAL_TX_NAME=remoteIOin
MAX_UPDATE_RATE=0.0
DEBUG=1

[TRANSACTION_02]
MB_TX_CODE=fnct_15_write_multiple_coils
FIRST_ELEMENT=1280
NELEMENTS=8
HAL_TX_NAME=remoteIOout
MAX_UPDATE_RATE=0.0

[TRANSACTION_03]
LINK_TYPE=serial
SERIAL_PORT=/dev/ttyS0
SERIAL_BAUD=115200
SERIAL_BITS=8
SERIAL_PARITY=none
SERIAL_STOP=2
SERIAL_DELAY_MS=50
MB_SLAVE_ID=1
MB_TX_CODE=fnct_03_read_holding_registers
FIRST_ELEMENT=1
NELEMENTS=2
HAL_TX_NAME=XDrive01
```

```
MAX_UPDATE_RATE=0.0
DEBUG=1

[TRANSACTION_04]
MB_TX_CODE=fnct_03_read_holding_registers
FIRST_ELEMENT=4
NELEMENTS=3
HAL_TX_NAME=XDrive02
MAX_UPDATE_RATE=0.0
DEBUG=1

[TRANSACTION_05]
MB_TX_CODE=fnct_03_read_holding_registers
FIRST_ELEMENT=9
NELEMENTS=1
HAL_TX_NAME=XDrive03
MAX_UPDATE_RATE=0.0

[TRANSACTION_06]
MB_TX_CODE=fnct_03_read_holding_registers
FIRST_ELEMENT=1024
NELEMENTS=1
HAL_TX_NAME=XDrive04
MAX_UPDATE_RATE=0.0

[TRANSACTION_07]
MB_TX_CODE=fnct_03_read_holding_registers
FIRST_ELEMENT=1030
NELEMENTS=2
HAL_TX_NAME=XDrive05
MAX_UPDATE_RATE=0.0

[TRANSACTION_08]
MB_TX_CODE=fnct_03_read_holding_registers
FIRST_ELEMENT=1033
NELEMENTS=1
HAL_TX_NAME=XDrive06
MAX_UPDATE_RATE=0.0
```

## 9.7 Opto22 Driver

PCI AC5 ADAPTER CARD / HAL DRIVER

### 9.7.1 The Adapter Card

This is a card made by Opto22 for adapting the PCI port to solid state relay racks such as their standard or G4 series. It has 2 ports that can control up to 24 points each and has 4 on board LEDs. The ports use 50 pin connectors the same as Mesa boards. Any relay racks/breakout boards thats work with Mesa Cards should work with this card with the understanding any encoder counters, PWM, etc., would have to be done in software. The AC5 does not have any *smart* logic on board, it is just an adapter.

See the manufacturer's website for more info:

http://www.opto22.com/site/pr_details.aspx?cid=4&item=PCI-AC5

I would like to thank Opto22 for releasing info in their manual, easing the writing of this driver!

### 9.7.2 The Driver

This driver is for the PCI AC5 card and will not work with the ISA AC5 card. The HAL driver is a realtime module. It will support 4 cards as is (more cards are possible with a change in the source code). Load the basic driver like so:

```
loadrt opto_ac5
```

This will load the driver which will search for max 4 boards. It will set I/O of each board's 2 ports to a default setting. The default configuration is for 12 inputs then 12 outputs. The pin name numbers correspond to the position on the relay rack. For example the pin names for the default I/O setting of port 0 would be:

- *opto_ac5.0.port0.in-00* - They would be numbered from 00 to 11
- *opto_ac5.0.port0.out-12* - They would be numbered 12 to 23 port 1 would be the same.

### 9.7.3 Pins

- *opto_ac5.[BOARDNUMBER].port[PORTNUMBER].in-[PINNUMBER] OUT bit* -
- *opto_ac5.[BOARDNUMBER].port[PORTNUMBER].in-[PINNUMBER]-not OUT bit* - Connect a HAL bit signal to this pin to read an I/O point from the card. The PINNUMBER represents the position in the relay rack. Eg. PINNUMBER 0 is position 0 in a Opto22 relay rack and would be pin 47 on the 50 pin header connector. The -not pin is inverted so that LOW gives TRUE and HIGH gives FALSE.
- *opto_ac5.[BOARDNUMBER].port[PORTNUMBER].out-[PINNUMBER] IN bit* - Connect a HAL bit signal to this pin to write to an I/O point of the card. The PINNUMBER represents the position in the relay rack.Eg. PINNUMBER 23 is position 23 in a Opto22 relay rack and would be pin 1 on the 50 pin header connector.
- *opto_ac5.[BOARDNUMBER].led[NUMBER] OUT bit* - Turns one of the 4 onboard LEDs on/off. LEDs are numbered 0 to 3.

BOARDNUMBER can be 0-3 PORTNUMBER can be 0 or 1. Port 0 is closest to the card bracket.

### 9.7.4 Parameters

- *opto_ac5.[BOARDNUMBER].port[PORTNUMBER].out-[PINNUMBER]-invert W bit* - When TRUE, invert the meaning of the corresponding -out pin so that TRUE gives LOW and FALSE gives HIGH.

### 9.7.5 FUNCTIONS

- *opto_ac5.0.digital-read* - Add this to a thread to read all the input points.
- *opto_ac5.0.digital-write* - Add this to a thread to write all the output points and LEDs.

For example the pin names for the default I/O setting of port 0 would be:

```
opto_ac5.0.port0.in-00
```

They would be numbered from 00 to 11

```
opto_ac5.0.port0.out-12
```

They would be numbered 12 to 23 port 1 would be the same.

### 9.7.6 Configuring I/O Ports

To change the default setting load the driver something like so:

```
loadrt opto_ac5 portconfig0=0xffff portconfig1=0xff0000
```

Of course changing the numbers to match the I/O you would like. Each port can be set up different.

Here's how to figure out the number: The configuration number represents a 32 bit long code to tell the card which I/O points are output vrs input. The lower 24 bits are the I/O points of one port. The 2 highest bits are for 2 of the on board LEDs. A one in any bit position makes the I/O point an output. The two highest bits must be output for the LEDs to work. The driver will automatically set the two highest bits for you, we won't talk about them.

The easiest way to do this is to fire up the calculator under APPLICATIONS/ACCESSORIES. Set it to scientific (click view). Set it BINARY (radio button Bin). Press 1 for every output you want and/or zero for every input. Remember that HAL pin 00 corresponds to the rightmost bit. 24 numbers represent the 24 I/O points of one port. So for the default setting (12 inputs then 12 outputs) you would push 1 twelve times (thats the outputs) then 0 twelve times (thats the inputs). Notice the first I/O point is the lowest (rightmost) bit. (that bit corresponds to HAL pin 00 .looks backwards) You should have 24 digits on the screen. Now push the Hex radio button. The displayed number (fff000) is the configport number ( put a *0x* in front of it designating it as a HEX number).

Another example: To set the port for 8 outputs and 16 inputs (the same as a Mesa card). Here is the 24 bits represented in a BINARY number. Bit 1 is the rightmost number.

000000000000000011111111

16 zeros for the 16 inputs and 8 ones for the 8 outputs

Which converts to FF on the calculator so 0xff is the number to use for portconfig0 and/or portconfig1 when loading the driver.

### 9.7.7 Pin Numbering

HAL pin 00 corresponds to bit 1 (the rightmost) which represents position 0 on an Opto22 relay rack. HAL pin 01 corresponds to bit 2 (one spot to the left of the rightmost) which represents position 1 on an Opto22 relay rack. HAL pin 23 corresponds to bit 24 (the leftmost) which represents position 23 on an Opto22 relay rack.

HAL pin 00 connects to pin 47 on the 50 pin connector of each port. HAL pin 01 connects to pin 45 on the 50 pin connector of each port. HAL pin 23 connects to pin 1 on the 50 pin connector of each port.

Note that Opto22 and Mesa use opposite numbering systems: Opto22 position 23 = connector pin 1, and the position goes down as the connector pin number goes up. Mesa Hostmot2 position 1 = connector pin 1, and the position number goes up as the connector pin number goes up.

## 9.8 Pico Drivers

Pico Systems has a family of boards for doing analog servo, stepper, and PWM (digital) servo control. The boards connect to the PC through a parallel port working in EPP mode. Although most users connect one board to a parallel port, in theory any mix of up to 8 or 16 boards can be used on a single parport. One driver serves all types of boards. The final mix of I/O depends on the connected board(s). The driver doesn't distinguish between boards, it simply numbers I/O channels (encoders, etc) starting from 0 on the first board. The driver is named hal_ppmc.ko The analog servo interface is also called the PPMC for Parallel Port Motion Control. There is also the Universal Stepper Controller, abbreviated the USC. And the Universal PWM Controller, or UPC.

Installing:

```
loadrt hal_ppmc port_addr=<addr1>[,<addr2>[,<addr3>...]]
```

The *port_addr* parameter tells the driver what parallel port(s) to check. By default, *<addr1>* is 0x0378, and *<addr2>* and following are not used. The driver searches the entire address space of the enhanced parallel port(s) at *port_addr*, looking for any board(s) in the PPMC family. It then exports HAL pins for whatever it finds. During loading (or attempted loading) the driver prints some useful debugging messages to the kernel log, which can be viewed with *dmesg*.

Up to 3 parport busses may be used, and each bus may have up to 8 (or possibly 16 PPMC) devices on it.

### 9.8.1 Command Line Options

There are several options that can be specified on the loadrt command line. First, the USC and UPC can have an 8-bit DAC added for spindle speed control and similar functions. This can be specified with the extradac=*0xnn[,0xmm]* parameter. The part enclosed in [ ] allows you to specify this option on more than one board of the system. The first hex digit tells which EPP bus is being referred to, it corresponds to the order of the port addresses in the port_addr parameter, where <addr1> would be zero here. So, for the first EPP bus, the first USC or UPC board would be described as *0x00*, the second USC or UPC on the same bus would be *0x02*. (Note that each USC or UPC takes up two addresses, so if one is at 00, the next would have to be 02.)

Alternatively, the 8 digital output pins can be used as additional digital outputs, it works the same way as above with the syntax : extradout=0xnn'. The extradac and extradout options are mutually exclusive on each board, you can only specify one.

The UPC and PPMC encoder boards can timestamp the arrival of encoder counts to refine the derivation of axis velocity. This derived velocity can be fed to the PID hal component to produce smoother D term response. The syntax is : timestamp=*0xnn[,0xmm]*, this works the same way as above to select which board is being configured. Default is to not enable the timestamp option. If you put this option on the command line, it enables the option. The first *n* selects the EPP bus, the second one matches the address of the board having the option enabled. The driver checks the revision level of the board to make sure it has firmware supporting the feature, and produces an error message if the board does not support it.

The PPMC encoder board has an option to select the encoder digital filter frequeency. (The UPC has the same ability via DIP switches on the board.) Since the PPMC encoder board doesn't have these extra DIP switches, it needs to be selected via a command-line option. By default, the filter runs at 1 MHz, allowing encoders to be counted up to about 900 KHz (depending on noise and quadrature accuracy of the encoder.) The options are 1, 2.5, 5 and 10 MHz. These are set with a parameter of 1,2,5 and 10 (decimal) which is specified as the hex digit "A". These are specified in a manner similar to the above options, but with the frequency setting to the left of the bus/address digits. So, to set 5 MHz on the encoder board at address 3 on the first EPP bus, you would write : enc_clock=*0x503*

It was recently discovered that some parallel port chips would not work with the ppmc driver. Especially, the Oxford OXPCIe952 chip on the SIIG PCIe parallel port cards had this trouble. The ppmc driver in all LinuxCNC versions starting from 2.7.8 have been corrected for this problem by default. However, this possibly could cause problems with really old EPP parallel port hardware, so there is a command line option to go back to the previous behavior. The new behavior is set by default, or by adding the parameter epp_dir=0 on the command line. To get the old behavior, add epp_dir=1 to the command line. All parallel ports I have here work with the new default behavior. As on the other parameters, it is possible to give a list, like epp_dir=1,0,1 to set different settings for each of up to 3 parallel ports.

### 9.8.2 Pins

In the following pins, parameters, and functions, <port> is the parallel port ID. According to the naming conventions the first port should always have an ID of zero. All the boards have some method of setting the address on the EPP bus. USC and UPC have simple provisions for only two addresses, but jumper foil cuts allow up to 4 boards to be addressed. The PPMC boards have 16 possible addresses. In all cases, the driver enumerates the boards by type and exports the appropriate HAL pins. For instance, the encoders will be enumerated from zero up, in the same order as the address switches on the board specify. So, the first board will have encoders 0 — 3, the second board would have encoders 4 — 7. The first column after the bullet tells which boards will have this HAL pin or parameter associated with it. All means that this pin is available on all three board types. Option means that this pin will only be exported when that option is enabled by an optional parameter in the loadrt HAL command. These options require the board to have a sufficient revision level to support the feature.

- *(All s32 output) ppmc.<port>.encoder.<channel>.count* - Encoder position, in counts.
- *(All s32 output) ppmc.<port>.encoder.<channel>.delta* - Change in counts since last read, in raw encoder count units.
- *(All float output) 'ppmc.<port>.encoder.<channel>.velocity* - Velocity scaled in user units per second. On PPMC and USC this is derived from raw encoder counts per servo period, and hence is affected by encoder granularity. On UPC boards with the 8/21/09 and later firmware, velocity estimation by timestamping encoder counts can be used to improve the smoothness of this velocity output. This can be fed to the PID HAL component to produce a more stable servo response. This function has to be enabled in the HAL command line that starts the PPMC driver, with the timestamp=0x00 option.
- *(All float output) ppmc.<port>.encoder.<channel>.position* - Encoder position, in user units.

- *(All bit bidir) ppmc.<port>.encoder.<channel>.index-enable* - Connect to axis.#.index-enable for home-to-index. This is a bidirectional HAL signal. Setting it to true causes the encoder hardware to reset the count to zero on the next encoder index pulse. The driver will detect this and set the signal back to false.
- *(PPMC float output) ppmc.<port>.DAC.<channel>.value* - sends a signed value to the 16-bit Digital to Analog Converter on the PPMC DAC16 board commanding the analog output voltage of that DAC channel.
- *(UPC bit input) ppmc.<port>.pwm.<channel>.enable* - Enables a PWM generator.
- *(UPC float input) ppmc.<port>.pwm.<channel>.value* - Value which determines the duty cycle of the PWM waveforms. The value is divided by *pwm.<channel>.scale*, and if the result is 0.6 the duty cycle will be 60%, and so on. Negative values result in the duty cycle being based on the absolute value, and the direction pin is set to indicate negative.
- *(USC bit input) ppmc.<port>.stepgen.<channel>.enable* - Enables a step pulse generator.
- *(USC float input) ppmc.<port>.stepgen.<channel>.velocity* - Value which determines the step frequency. The value is multiplied by *stepgen.<channel>.scale* , and the result is the frequency in steps per second. Negative values result in the frequency being based on the absolute value, and the direction pin is set to indicate negative.
- *(All bit output) ppmc.<port>.din.<channel>.in* - State of digital input pin, see canonical digital input.
- *(All bit output) ppmc.<port>.din.<channel>.in-not* - Inverted state of digital input pin, see canonical digital input.
- *(All bit input) ppmc.<port>.dout.<channel>.out* - Value to be written to digital output, see canonical digital output.
- *(Option float input) ppmc.<port>.DAC8-<channel>.value* - Value to be written to analog output, range from 0 to 255. This sends 8 output bits to J8, which should have a Spindle DAC board connected to it. 0 corresponds to zero Volts, 255 corresponds to 10 Volts. The polarity of the output can be set for always minus, always plus, or can be controlled by the state of SSR1 (plus when on) and SSR2 (minus when on). You must specify extradac = 0x00 on the HAL command line that loads the PPMC driver to enable this function on the first USC ur UPC board.
- *(Option bit input) ppmc.<port>.dout.<channel>.out* - Value to be written to one of the 8 extra digital output pins on J8. You must specify extradout = 0x00 on the HAL command line that loads the ppmc driver to enable this function on the first USC or UPC board. extradac and extradout are mutually exclusive features as they use the same signal lines for different purposes. These output pins will be enumerated after the standard digital outputs of the board.

### 9.8.3 Parameters

- *(All float) ppmc.<port>.encoder.<channel>.scale* - The number of counts / user unit (to convert from counts to units).
- *(UPC float) ppmc.<port>.pwm.<channel-range>.freq* - The PWM carrier frequency, in Hz. Applies to a group of four consecutive PWM generators, as indicated by *<channel-range>*. Minimum is 610Hz, maximum is 500KHz.
- *(PPMC float) ppmc.<port>.DAC.<channel>.scale* - Sets scale of DAC16 output channel such that an output value equal to the 1/scale value will produce an output of + or - value Volts. So, if the scale parameter is 0.1 and you send a value of 0.5, the output will be 5.0 Volts.
- *(UPC float) ppmc.<port>.pwm.<channel>.scale* - Scaling for PWM generator. If *scale* is X, then the duty cycle will be 100% when the *value* pin is X (or -X).
- *(UPC float) ppmc.<port>.pwm.<channel>.max-dc* - Maximum duty cycle, from 0.0 to 1.0.
- *(UPC float) ppmc.<port>.pwm.<channel>.min-dc* - Minimum duty cycle, from 0.0 to 1.0.
- *(UPC float) ppmc.<port>.pwm.<channel>.duty-cycle* - Actual duty cycle (used mostly for troubleshooting.)
- *(UPC bit) ppmc.<port>.pwm.<channel>.bootstrap* - If true, the PWM generator will generate a short sequence of pulses of both polarities when E-stop goes false, to reset the shutdown latches on some PWM servo drives.
- *(USC u32) ppmc.<port>.stepgen.<channel-range>.setup-time* - Sets minimum time between direction change and step pulse, in units of 100ns. Applies to a group of four consecutive step generators, as indicated by *<channel-range>*. Values between 200 ns and 25.5 us can be specified.

- *(USC u32) ppmc.<port>.stepgen.<channel-range>.pulse-width* - Sets width of step pulses, in units of 100ns. Applies to a group of four consecutive step generators, as indicated by *<channel-range>*. Values between 200 ns and 25.5 us may be specified.
- *(USC u32) ppmc.<port>.stepgen.<channel-range>.pulse-space-min* - Sets minimum time between pulses, in units of 100ns. Applies to a group of four consecutive step generators, as indicated by *<channel-range>*. Values between 200 ns and 25.5 us can be specified. The maximum step rate is: $\frac{1}{100\text{ns} * (pulsewidth + pulsespacemin)}$
- *(USC float) ppmc.<port>.stepgen.<channel>.scale* - Scaling for step pulse generator. The step frequency in Hz is the absolute value of *velocity* * *scale*.
- *(USC float) ppmc.<port>.stepgen.<channel>.max-vel* - The maximum value for *velocity*. Commands greater than *max-vel* will be clamped. Also applies to negative values. (The absolute value is clamped.)
- *(USC float) ppmc.<port>.stepgen.<channel>.frequency* - Actual step pulse frequency in Hz (used mostly for troubleshooting.)
- *(Option float) ppmc.<port>.DAC8.<channel>.scale* - Sets scale of extra DAC output such that an output value equal to scale gives a magnitude of 10.0 V output. (The sign of the output is set by jumpers and/or other digital outputs.)
- *(Option bit) ppmc.<port>.dout.<channel>.invert* - Inverts a digital output, see canonical digital output.
- *(Option bit) ppmc.<port>.dout.<channel>.invert* - Inverts a digital output pin of J8, see canonical digital output.

### 9.8.4 Functions

- *(All funct) ppmc.<port>.read* - Reads all inputs (digital inputs and encoder counters) on one port. These reads are organized into blocks of contiguous registers to be read in a block to minimize CPU overhead.
- *(All funct) ppmc.<port>.write* - Writes all outputs (digital outputs, stepgens, PWMs) on one port. These writes are organized into blocks of contiguous registers to be written in a block to minimize CPU overhead.

## 9.9 Pluto P Driver

### 9.9.1 General Info

The Pluto-P is a FPGA board featuring the ACEX1K chip from Altera.

#### 9.9.1.1 Requirements

1. A Pluto-P board
2. An EPP-compatible parallel port, configured for EPP mode in the system BIOS or a PCI EPP compatible parallel port card.

**Note**
The Pluto P board requires EPP mode. Netmos98xx chips do not work in EPP mode. The Pluto P board will work on some computers and not on others. There is no known pattern to which computers work and which don't work.

For more information on PCI EPP compatible parallel port cards see the LinuxCNC Supported Hardware page on the wiki.

#### 9.9.1.2 Connectors

- The Pluto-P board is shipped with the left connector presoldered, with the key in the indicated position. The other connectors are unpopulated. There does not seem to be a standard 12-pin IDC connector, but some of the pins of a 16P connector can hang off the board next to QA3/QZ3.
- The bottom and right connectors are on the same .1" grid, but the left connector is not. If OUT2…OUT9 are not required, a single IDC connector can span the bottom connector and the bottom two rows of the right connector.

#### 9.9.1.3 Physical Pins

- Read the ACEX1K datasheet for information about input and output voltage thresholds. The pins are all configured in *LVTTL/LVCMOS* mode and are generally compatible with 5V TTL logic.
- Before configuration and after properly exiting LinuxCNC, all Pluto-P pins are tristated with weak pull-ups (20k-ohms min, 50k-ohms max). If the watchdog timer is enabled (the default), these pins are also tristated after an interruption of communication between LinuxCNC and the board. The watchdog timer takes approximately 6.5ms to activate. However, software bugs in the pluto_servo firmware or LinuxCNC can leave the Pluto-P pins in an undefined state.
- In pwm+dir mode, by default dir is HIGH for negative values and LOW for positive values. To select HIGH for positive values and LOW for negative values, set the corresponding dout-NN-invert parameter TRUE to invert the signal.
- The index input is triggered on the rising edge. Initial testing has shown that the QZx inputs are particularly noise sensitive, due to being polled every 25ns. Digital filtering has been added to filter pulses shorter than 175ns (seven polling times). Additional external filtering on all input pins, such as a Schmitt buffer or inverter, RC filter, or differential receiver (if applicable) is recommended.
- The IN1…IN7 pins have 22-ohm series resistors to their associated FPGA pins. No other pins have any sort of protection for out-of-spec voltages or currents. It is up to the integrator to add appropriate isolation and protection. Traditional parallel port optoisolator boards do not work with pluto_servo due to the bidirectional nature of the EPP protocol.

#### 9.9.1.4 LED

- When the device is unprogrammed, the LED glows faintly. When the device is programmed, the LED glows according to the duty cycle of PWM0 (*LED = UP0 xor DOWN0*) or STEPGEN0 (*LED = STEP0 xor DIR0*).

#### 9.9.1.5 Power

- A small amount of current may be drawn from VCC. The available current depends on the unregulated DC input to the board. Alternately, regulated +3.3VDC may be supplied to the FPGA through these VCC pins. The required current is not yet known, but is probably around 50mA plus I/O current.
- The regulator on the Pluto-P board is a low-dropout type. Supplying 5V at the power jack will allow the regulator to work properly.

#### 9.9.1.6 PC interface

- Only a single pluto_servo or pluto_step board is supported.

#### 9.9.1.7 Rebuilding the FPGA firmware

The *src/hal/drivers/pluto_servo_firmware/* and *src/hal/drivers/pluto_step_firmware/* subdirectories contain the Verilog source code plus additional files used by Quartus for the FPGA firmwares. Altera's Quartus II software is required to rebuild the FPGA firmware. To rebuild the firmware from the .hdl and other source files, open the *.qpf* file and press CTRL-L. Then, recompile LinuxCNC.

Like the HAL hardware driver, the FPGA firmware is licensed under the terms of the GNU General Public License.

The gratis version of Quartus II runs only on Microsoft Windows, although there is apparently a paid version that runs on Linux.

#### 9.9.1.8 For more information

Some additional information about it is available from KNJC LLC and from the developer's blog.

### 9.9.2 Pluto Servo

The pluto_servo system is suitable for control of a 4-axis CNC mill with servo motors, a 3-axis mill with PWM spindle control, a lathe with spindle encoder, etc. The large number of inputs allows a full set of limit switches.

This driver features:

- 4 quadrature channels with 40MHz sample rate. The counters operate in *4x* mode. The maximum useful quadrature rate is 8191 counts per LinuxCNC servo cycle, or about 8MHz for LinuxCNC's default 1ms servo rate.
- 4 PWM channels, *up/down* or *pwm+dir* style. 4095 duty cycles from -100% to +100%, including 0%. The PWM period is approximately 19.5kHz (40MHz / 2047). A PDM-like mode is also available.
- 18 digital outputs: 10 dedicated, 8 shared with PWM functions. (Example: A lathe with unidirectional PWM spindle control may use 13 total digital outputs)
- 20 digital inputs: 8 dedicated, 12 shared with Quadrature functions. (Example: A lathe with index pulse only on the spindle may use 13 total digital inputs)
- EPP communication with the PC. The EPP communication typically takes around 100 us on machines tested so far, enabling servo rates above 1kHz.

#### 9.9.2.1 Pinout

- *UPx* - The *up* (up/down mode) or *pwm* (pwm+direction mode) signal from PWM generator X. May be used as a digital output if the corresponding PWM channel is unused, or the output on the channel is always negative. The corresponding digital output invert may be set to TRUE to make UPx active low rather than active high.
- *DNx* - The *down* (up/down mode) or *direction* (pwm+direction mode) signal from PWM generator X. May be used as a digital output if the corresponding PWM channel is unused, or the output on the channel is never negative. The corresponding digital ouput invert may be set to TRUE to make DNx active low rather than active high.
- *QAx, QBx* - The A and B signals for Quadrature counter X. May be used as a digital input if the corresponding quadrature channel is unused.
- *QZx* - The Z (index) signal for quadrature counter X. May be used as a digital input if the index feature of the corresponding quadrature channel is unused.
- *INx* - Dedicated digital input #x
- *OUTx* - Dedicated digital output #x
- *GND* - Ground
- *VCC* - +3.3V regulated DC

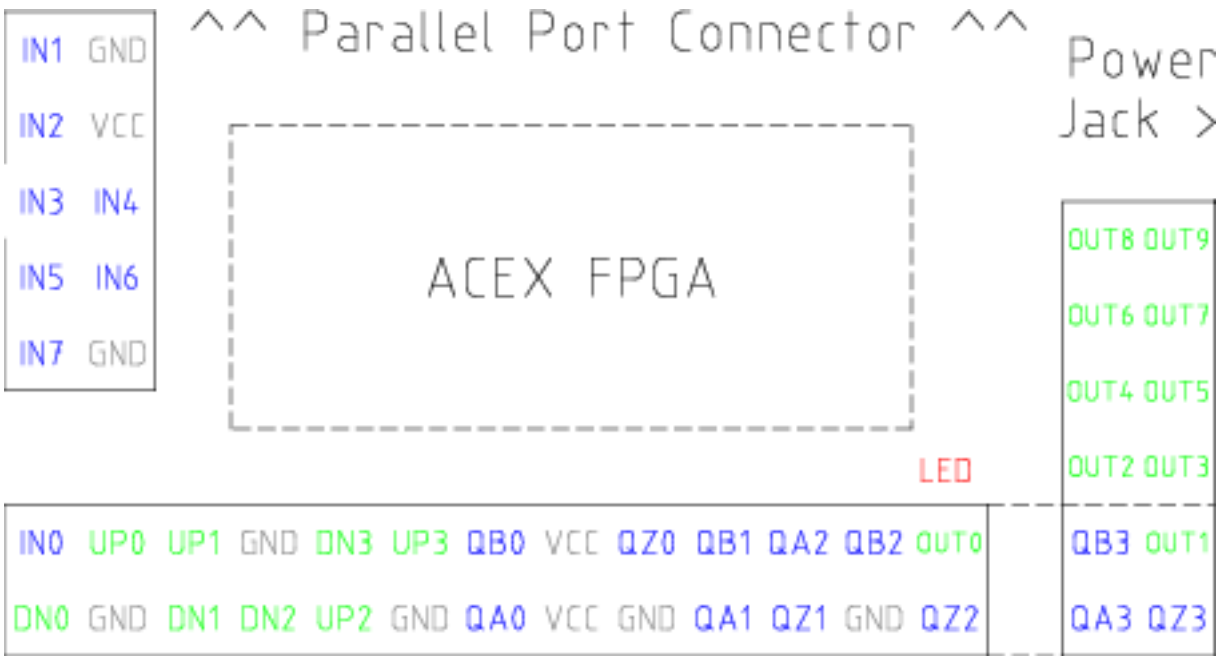



Figure 9.2: Pluto-Servo Pinout

Table 9.2: Pluto-Servo Alternate Pin Functions

| Primary function | Alternate Function | Behavior if both functions used |
|---|---|---|
| **UP0** | PWM0 | When pwm-0-pwmdir is TRUE, this pin is the PWM output |
| | OUT10 | XOR'd with UP0 or PWM0 |
| **UP1** | PWM1 | When pwm-1-pwmdir is TRUE, this pin is the PWM output |
| | OUT12 | XOR'd with UP1 or PWM1 |
| **UP2** | PWM2 | When pwm-2-pwmdir is TRUE, this pin is the PWM output |
| | OUT14 | XOR'd with UP2 or PWM2 |
| **UP3** | PWM3 | When pwm-3-pwmdir is TRUE, this pin is the PWM output |
| | OUT16 | XOR'd with UP3 or PWM3 |
| **DN0** | DIR0 | When pwm-0-pwmdir is TRUE, this pin is the DIR output |
| | OUT11 | XOR'd with DN0 or DIR0 |
| **DN1** | DIR1 | When pwm-1-pwmdir is TRUE, this pin is the DIR output |
| | OUT13 | XOR'd with DN1 or DIR1 |
| **DN2** | DIR2 | When pwm-2-pwmdir is TRUE, this pin is the DIR output |
| | OUT15 | XOR'd with DN2 or DIR2 |
| **DN3** | DIR3 | When pwm-3-pwmdir is TRUE, this pin is the DIR output |
| | OUT17 | XOR'd with DN3 or DIR3 |
| **QZ0** | IN8 | Read same value |
| **QZ1** | IN9 | Read same value |
| **QZ2** | IN10 | Read same value |
| **QZ3** | IN11 | Read same value |
| **QA0** | IN12 | Read same value |

Table 9.2: (continued)

| Primary function | Alternate Function | Behavior if both functions used |
|---|---|---|
| **QA1** | IN13 | Read same value |
| **QA2** | IN14 | Read same value |
| **QA3** | IN15 | Read same value |
| **QB0** | IN16 | Read same value |
| **QB1** | IN17 | Read same value |
| **QB2** | IN18 | Read same value |
| **QB3** | IN19 | Read same value |

#### 9.9.2.2 Input latching and output updating

- PWM duty cycles for each channel are updated at different times.
- Digital outputs OUT0 through OUT9 are all updated at the same time. Digital outputs OUT10 through OUT17 are updated at the same time as the pwm function they are shared with.
- Digital inputs IN0 through IN19 are all latched at the same time.
- Quadrature positions for each channel are latched at different times.

#### 9.9.2.3 HAL Functions, Pins and Parameters

A list of all *loadrt* arguments, HAL function names, pin names and parameter names is in the manual page, *pluto_servo.9*.

#### 9.9.2.4 Compatible driver hardware

A schematic for a 2A, 2-axis PWM servo amplifier board is available from the (the software developer). The L298 H-Bridge can be used for motors up to 4A (one motor per L298) or up to 2A (two motors per L298) with the supply voltage up to 46V. However, the L298 does not have built-in current limiting, a problem for motors with high stall currents. For higher currents and voltages, some users have reported success with International Rectifier's integrated high-side/low-side drivers.

### 9.9.3 Pluto Step

Pluto-step is suitable for control of a 3- or 4-axis CNC mill with stepper motors. The large number of inputs allows for a full set of limit switches.

The board features:

- 4 *step+direction* channels with 312.5kHz maximum step rate, programmable step length, space, and direction change times
- 14 dedicated digital outputs
- 16 dedicated digital inputs
- EPP communication with the PC

#### 9.9.3.1 Pinout

- *STEPx* - The *step* (clock) output of stepgen channel *x*
- *DIRx* - The *direction* output of stepgen channel *x*
- *INx* - Dedicated digital input #x
- *OUTx* - Dedicated digital output #x
- *GND* - Ground
- *VCC* - +3.3V regulated DC

While the *extended main connector* has a superset of signals usually found on a Step & Direction DB25 connector—4 step generators, 9 inputs, and 6 general-purpose outputs—the layout on this header is different than the layout of a standard 26-pin ribbon cable to DB25 connector.

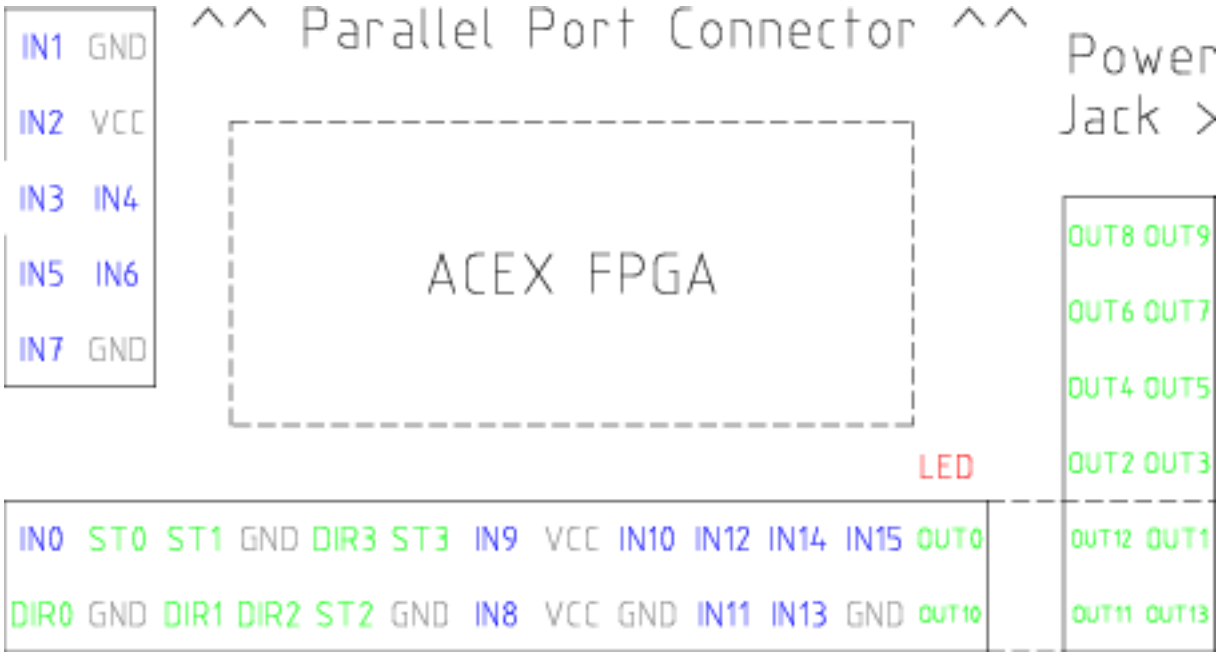



Figure 9.3: Pluto-Step Pinout

#### 9.9.3.2 Input latching and output updating

- Step frequencies for each channel are updated at different times.
- Digital outputs are all updated at the same time.
- Digital inputs are all latched at the same time.
- Feedback positions for each channel are latched at different times.

#### 9.9.3.3 Step Waveform Timings

The firmware and driver enforce step length, space, and direction change times. Timings are rounded up to the next multiple of 1.6µs, with a maximum of 49.6µs. The timings are the same as for the software stepgen component, except that *dirhold* and *dirsetup* have been merged into a single parameter *dirtime* which should be the maximum of the two, and that the same step timings are always applied to all channels.

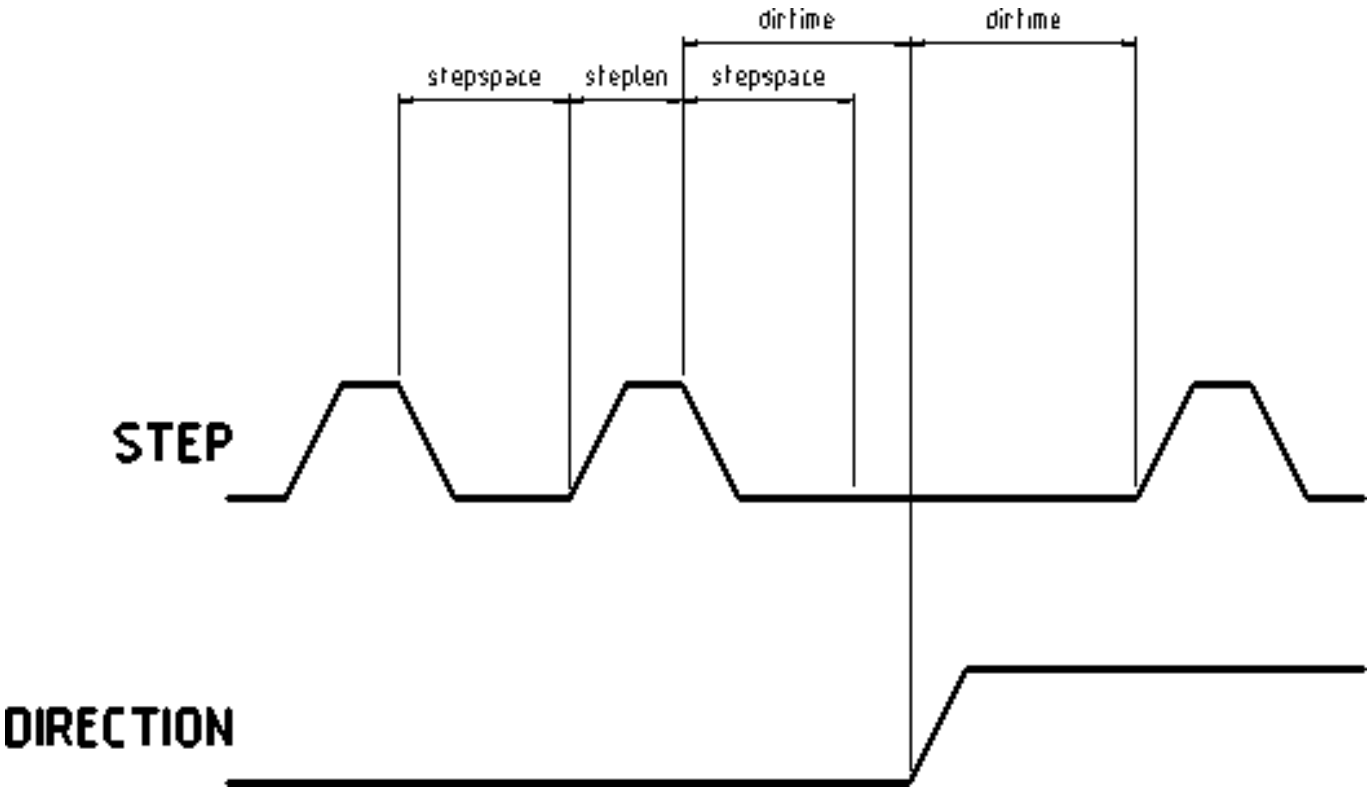



Figure 9.4: Pluto-Step Timings

#### 9.9.3.4 HAL Functions, Pins and Parameters

A list of all *loadrt* arguments, HAL function names, pin names and parameter names is in the manual page, *pluto_step.9*.

## 9.10 Servo To Go Driver

The Servo-To-Go (STG) is one of the first PC motion control cards supported by LinuxCNC. It is an ISA card and it exists in different flavors (all supported by this driver). The board includes up to 8 channels of quadrature encoder input, 8 channels of analog input and output, 32 bits digital I/O, an interval timer with interrupt and a watchdog. For more information see the Servo To Go web page.

**Note**
We have had reports that the opamps on the Servo To Go card do not work with newer ATX power supplies that use modern switch mode DC-DC converters. The failure mode is that STG card outputs a constant voltage regardless of what the driver is commanding it to do. Older ATX power supplies with linear voltage regulators do not have this problem, and work fine with the STG cards.

### 9.10.1 Installing

```
loadrt hal_stg [base=<address>] [num_chan=<nr>] [dio="<dio-string>"] [model=<model>]
```

The base address field is optional; if it's not provided the driver attempts to autodetect the board. The num_chan field is used to specify the number of channels available on the card, if not used the 8 axis version is assumed. The digital inputs/outputs configuration is determined by a config string passed to insmod when loading the module. The format consists of a four character

string that sets the direction of each group of pins. Each character of the direction string is either "I" or "O". The first character sets the direction of port A (Port A - DIO.0-7), the next sets port B (Port B - DIO.8-15), the next sets port C (Port C - DIO.16-23), and the fourth sets port D (Port D - DIO.24-31). The model field can be used in case the driver doesn't autodetect the right card version.

hint: after starting up the driver, *dmesg* can be consulted for messages relevant to the driver (e.g. autodetected version number and base address). For example:

```
loadrt hal_stg base=0x300 num_chan=4 dio="IOIO"
```

This example installs the STG driver for a card found at the base address of 0x300, 4 channels of encoder feedback, DACs and ADCs, along with 32 bits of I/O configured like this: the first 8 (Port A) configured as Input, the next 8 (Port B) configured as Output, the next 8 (Port C) configured as Input, and the last 8 (Port D) configured as Output

```
loadrt hal_stg
```

This example installs the driver and attempts to autodetect the board address and board model, it installs 8 axes by default along with a standard I/O setup: Port A & B configured as Input, Port C & D configured as Output.

### 9.10.2 Pins

- *stg.<channel>.counts* - (s32) Tracks the counted encoder ticks.
- *stg.<channel>.position* - (float) Outputs a converted position.
- *stg.<channel>.dac-value* - (float) Drives the voltage for the corresponding DAC.
- *stg.<channel>.adc-value* - (float) Tracks the measured voltage from the corresponding ADC.
- *stg.in-<pinnum>* - (bit) Tracks a physical input pin.
- *stg.in-<pinnum>-not* - (bit) Tracks a physical input pin, but inverted.
- *stg.out-<pinnum>* - (bit) Drives a physical output pin

For each pin, <channel> is the axis number, and <pinnum> is the logic pin number of the STG if `IIOO` is defined, there are 16 input pins (in-00 .. in-15) and 16 output pins (out-00 .. out-15), and they correspond to PORTs ABCD (in-00 is PORTA.0, out-15 is PORTD.7).

The in-<pinnum> HAL pin is TRUE if the physical pin is high, and FALSE if the physical pin is low. The in-<pinnum>-not HAL pin is inverted — it is FALSE if the physical pin is high. By connecting a signal to one or the other, the user can determine the state of the input.

### 9.10.3 Parameters

- *stg.<channel>.position-scale* - (float) The number of counts / user unit (to convert from counts to units).
- *stg.<channel>.dac-offset* - (float) Sets the offset for the corresponding DAC.
- *stg.<channel>.dac-gain* - (float) Sets the gain of the corresponding DAC.
- *stg.<channel>.adc-offset* - (float) Sets the offset of the corresponding ADC.
- *stg.<channel>.adc-gain* - (float) Sets the gain of the corresponding ADC.
- *stg.out-<pinnum>-invert* - (bit) Inverts an output pin.

The -invert parameter determines whether an output pin is active high or active low. If -invert is FALSE, setting the HAL out-pin TRUE drives the physical pin high, and FALSE drives it low. If -invert is TRUE, then setting the HAL out- pin TRUE will drive the physical pin low.

### 9.10.4 Functions

- *stg.capture-position* - Reads the encoder counters from the axis <channel>.
- *stg.write-dacs* - Writes the voltages to the DACs.
- *stg.read-adcs* - Reads the voltages from the ADCs.
- *stg.di-read* - Reads physical in- pins of all ports and updates all HAL in-<pinnum> and in-<pinnum>-not pins.
- *stg.do-write* - Reads all HAL out-<pinnum> pins and updates all physical output pins.

## 9.11 ShuttleXpress

### 9.11.1 Description

shuttlexpress is a non-realtime HAL component that interfaces Contour Design's ShuttleXpress device with LinuxCNC's HAL.

If the driver is started without command-line arguments, it will probe all /dev/hidraw* device files for ShuttleXpress devices, and use all devices found. If it is started with command-line arguments, it will only probe the devices specified.

The ShuttleXpress has five momentary buttons, a 10 counts/revolution jog wheel with detents, and a 15-position spring-loaded outer wheel that returns to center when released.

**Warning**
The ShuttleXpress device has an internal 8-bit counter for the current jog-wheel position. The shuttlexpress driver can not know this value until the ShuttleXpress device sends its first event. When the first event comes into the driver, the driver uses the device's reported jog-wheel position to initialize counts to 0.
This means that if the first event is generated by a jog-wheel move, that first move will be lost.
Any user interaction with the ShuttleXpress device will generate an event, informing the driver of the jog-wheel position. So if you (for example) push one of the buttons at startup, the jog-wheel will work fine and notice the first click.

### 9.11.2 Setup

The shuttlexpress module needs read permission to the /dev/hidraw* device files. This can be accomplished by adding a file /etc/udev/rules.d/99-shuttlexpress.rules, with the following contents:

```
SUBSYSTEM=="hidraw", ATTRS{idVendor}=="0b33", ATTRS{idProduct}=="0020", MODE="0444"
```

The LinuxCNC Debian package installs an appropriate udev file automatically, but if you are building LinuxCNC from source and are not using the Debian packaging you'll need to install this file by hand.

### 9.11.3 Pins

***shuttlexpress.<DeviceNumber>.button-<ButtonNumber>*** **(bit out)**
The ShuttleXpress has five buttons around the outside of the device, numbered 0 through 4. 0 is the counter-clockwise-most button, 4 is the clockwise-most button. These pins are True (1) when the button is pressed.

***shuttlexpress.<DeviceNumber>.button-<ButtonNumber>-not*** **(bit out)**
These pins have the inverse of the button state, so they're True (1) when the button is not pressed.

***shuttlexpress.<DeviceNumber>.counts*** **(s32 out)**
Accumulated counts from the jog wheel (the inner wheel).

***shuttlexpress.<DeviceNumber>.spring-wheel-s32*** **(s32 out)**
The current deflection of the spring-wheel (the outer wheel). It's 0 at rest, and ranges from -7 at the counter-clockwise extreme to +7 at the clockwise extreme.

***shuttlexpress.<DeviceNumber>.spring-wheel-f*** **(float out)**
The current deflection of the spring-wheel (the outer wheel). It's 0.0 at rest, -1.0 at the counter-clockwise extreme, and +1.0 at the clockwise extreme. (The ShuttleXpress device reports the spring-wheel position as an integer from -7 to +7, so this pin reports only 15 discrete values in it's range.)

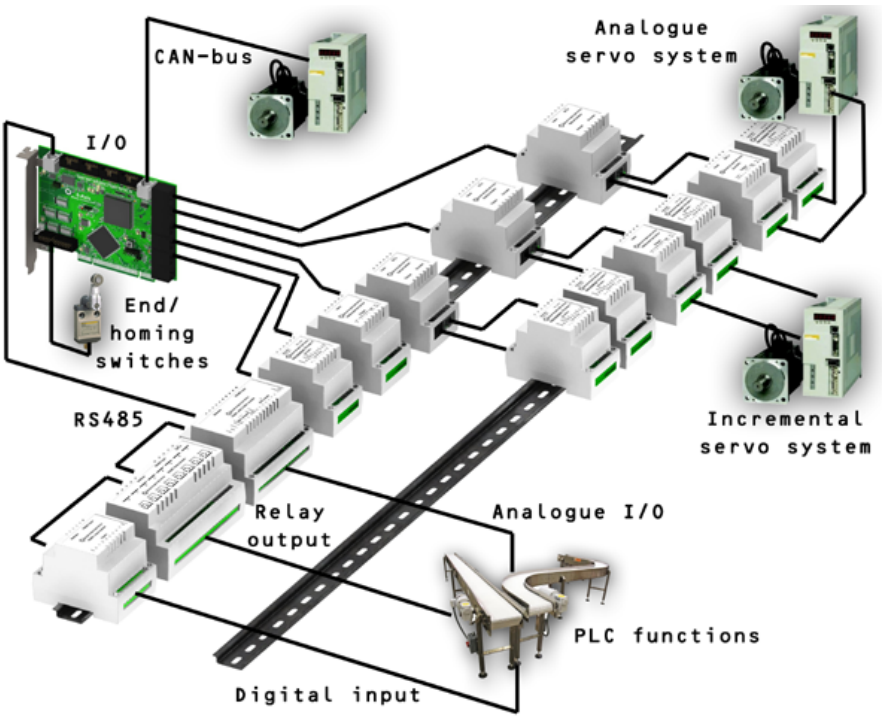
## 9.12 General Mechatronics Driver

General Mechatronics GM6-PCI card based motion control system

For detailed description, please refer to the System integration manual.

The GM6-PCI motion control card is based on an FPGA and a PCI bridge interface ASIC. A small automated manufacturing cell can be controlled, with a short time system integration procedure. The following figure demonstrating the typical connection of devices related to the control system:

- It can control up to six axis, each can be stepper or CAN bus interface or analogue servo.
- GPIO: Four time eight I/O pins are placed on standard flat cable headers.
- RS485 I/O expander modules: RS485 bus was designed for interfacing with compact DIN-rail mounted expander modules. An 8-channel digital input, an 8-channel relay output and an analogue I/O (4x +/-10 Volts output and 8x +/-5 Volts input) modules are available now. Up to 16 modules can be connected to the bus altogether.
- 20 optically isolated input pins: Six times three for the direct connection of two end switch and one homing sensor for each axis. And additionally, two optically isolated E-stop inputs.




Installing:

```
loadrt hal_gm
```

During loading (or attempted loading) the driver prints some useful debugging messages to the kernel log, which can be viewed with dmesg.

Up to 3 boards may be used in one system.

The following connectors can be found on the GM6-PCI card:

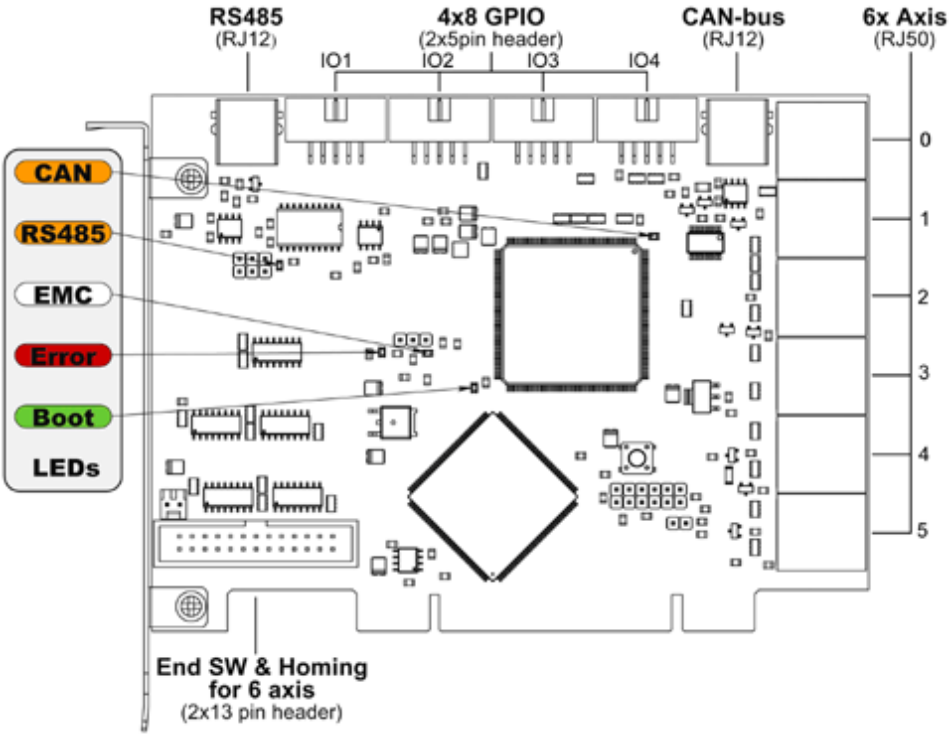



Figure 9.5: GM6-PCI card connectors and LEDs

### 9.12.1 I/O connectors

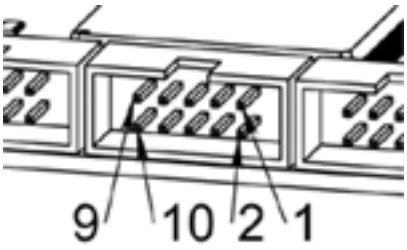



Figure 9.6: Pin numbering of GPIO connectors

Table 9.3: Pinout of GPIO connectors

| 9 | 7 | 5 | 3 | 1 |
|---|---|---|---|---|
| IOx/7 | IOx/5 | IOx/3 | IOx/1 | VCC |

| 10 | 8 | 6 | 4 | 2 |
|---|---|---|---|---|
| GND | IOx/6 | IOx/4 | IOx/2 | IOx/0 |

Each pin can be configured as digital input or output. GM6-PCI motion control card has 4 general purpose I/O (GPIO) connectors, with eight configurable I/O on each. Every GPIO pin and parameter name begins as follows:

```
gm.<nr. of card>.gpio.<nr of gpio con>
```

,where <nr of gpio con> is form 0 to 3. For example:

```
gm.0.gpio.0.in-0
```

indicates the state of the first pin of the first GPIO connector on the GM6-PCI card. Hal pins are updated by function

```
gm.<nr of card>.read
```

#### 9.12.1.1 Pins

Table 9.4: GPIO pins

| Pins | Type and direction | Pin description |
|---|---|---|
| .in-<0-7> | (bit, Out) | Input pin |
| .in-not-<0-7> | (bit, Out) | Negated input pin |
| .out-<0-7> | (bit, In) | Output pin. Used only when GPIO is set to output. |

#### 9.12.1.2 Parameters

Table 9.5: GPIO parameters

| Pins | Type and direction | Parameter description |
|---|---|---|
| .is-out-<0-7> | (bit, R/W) | When True, the corresponding GPIO is set to totem-pole output, other wise set to high impedance input. |
| .invert-out-<0-7> | (bit, R/W) | When True, pin value will be inverted. Used when pin is configured as output. |

### 9.12.2 Axis connectors

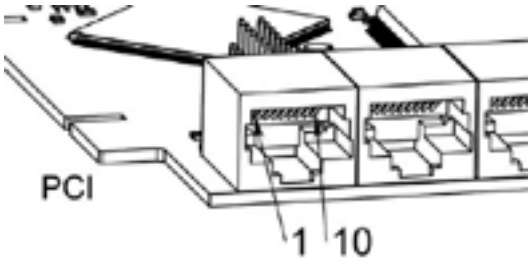



Figure 9.7: Pin numbering of axis connectors

Table 9.6: Pinout of axis connectors

| | |
|---|---|
| 1 | Encoder A |
| 2 | +5 Volt (PC) |
| 3 | Encoder B |
| 4 | Encoder Index |
| 5 | Fault |
| 6 | Power Enabled |
| 7 | Step/CCW/B |
| 8 | Direction/CW/A |
| 9 | Ground (PC) |
| 10 | DAC serial line |

#### 9.12.2.1 Axis interface modules

Small sized DIN rail mounted interface modules gives easy way of connecting different types of servo modules to the axis connectors. Seven different system configurations are presented in the System integration manual for evaluating typical applications. Also the detailed description of the Axis modules can be found in the System integration manual.

For evaluating the appropriate servo-drive structure the modules have to be connected as the following block diagram shows:

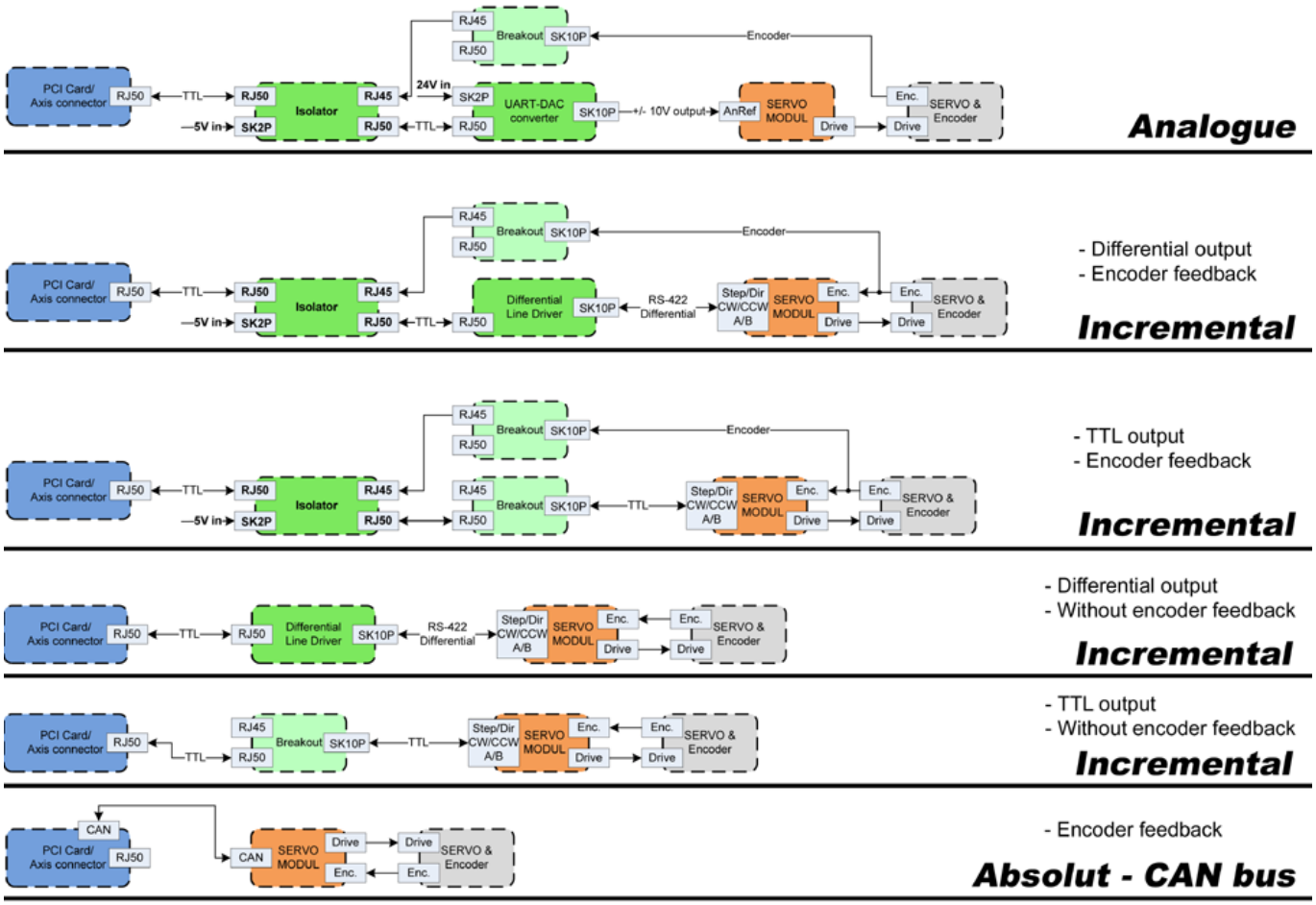



Figure 9.8: Servo axis interfaces

#### 9.12.2.2 Encoder

The GM6-PCI motion control card has six encoder modules. Each encoder module has three channels:

- Channel-A
- Channel-B
- Channel-I (index)

It is able to count quadrature encoder signals or step/dir signals. Each encoder module is connected to the inputs of the corresponding RJ50 axis connector.

Every encoder pin and parameter name begins as follows:

```
gm.<nr. of card>.encoder.<nr of axis>
```

,where <nr of axis> is form 0 to 5. For example:

```
gm.0.encoder.0.position
```

refers to the position of encoder module of axis 0.

The GM6-PCI card counts the encoder signal independently from LinuxCNC. Hal pins are updated by function:

```
gm.<nr of card>.read
```

Table 9.7: Encoder pins

| Pins | Type and direction | Pin description |
|---|---|---|
| .reset | (bit, In) | When True, resets counts and position to zero. |
| .rawcounts | (s32, Out) | The raw count is the counts, but unaffected by reset or the index pulse. |
| .counts | (s32, Out) | Position in encoder counts. |
| .position | (float, Out) | Position in scaled units (=.counts/.position-scale). |
| .index-enabled | (bit, IO) | When True, counts and position are rounded or reset (depends on index-mode) on next rising edge of channel-I. Every time position is reset because of Index, index-enabled pin is set to 0 and remain 0 until connected hal pin does not set it. |
| .velocity | (float, Out) | Velocity in scaled units per second. GM encoder uses high frequency hardware timer to measure time between encoder pulses in order to calculate velocity. It greatly reduces quantization noise as compared to simply differentiating the position output. When the measured velocity is below min-speed-estimate, the velocity output is 0. |

Table 9.8: Encoder parameters

| Parameters | Type and Read/Write | Parameter description |
|---|---|---|
| .counter-mode | (bit, R/W) | When True, the counter counts each rising edge of the channel-A input to the direction determined by channel-B. This is useful for counting the output of a single channel (non-quadrature) or step/dir signal sensor. When false, it counts in quadrature mode. |
| .index-mode | (bit, R/W) | When True and .index-enabled is also true, .counts and .position are rounded (based on .counts-per-rev) at rising edge of channel-I. This is useful to correct few pulses error caused by noise. In round mode, it is essential to set .counts-per-rev parameter correctly. When .index-mode is False and .index-enabled is true, .counts and .position are reset at channel-I pulse. |
| .counts-per-rev | (s32, R/V) | Determine how many counts are between two index pulses. It is used only in round mode, so when both .index-enabled and .index-mode parameters are True. GM encoder process encoder signal in 4x mode, so for example in case of a 500 CPR encoder it should be set to 2000. This parameter can be easily measured by setting .index-enabled True and .index-mode False (so that .counts resets at channel-I pulse), than move axis by hand and see the maximum magnitude of .counts pin in halmeter. |

Table 9.8: (continued)

| Parameters | Type and Read/Write | Parameter description |
|---|---|---|
| .index-invert | (bit, R/W) | When True, channel-I event (reset or round) occur on falling edge of channel-I signal, otherwise on rising edge. |
| .min-speed-estimate | (float, R/W) | Determine the minimum measured velocity magnitude at which .velocity will be set as nonzero. Setting this parameter too low will cause it to take a long time for velocity to go to zero after encoder pulses have stopped arriving. |
| .position-scale | (float, R/W) | Scale in counts per length unit. .position=.counts/.position-scale. For example, if position-scale is 2000, then 1000 counts of the encoder will produce a position of 0.5 units. |

**HAL example** Setting encoder module of axis 0 to receive 500 CPR quadrature encoder signal and use reset to round position.

```
setp gm.0.encoder.0.counter-mode 0         # 0: quad, 1: stepDir
setp gm.0.encoder.0.index-mode 1           # 0: reset pos at index, 1:round pos at index
setp gm.0.encoder.0.counts-per-rev 2000      # GM process encoder in 4x mode, 4x500=2000
setp gm.0.encoder.0.index-invert 0
setp gm.0.encoder.0.min-speed-estimate 0.1 # in position unit/s
setp gm.0.encoder.0.position-scale 20000   # 10 encoder rev cause the machine to
                                             move one position unit (10x2000)
```

Connect encoder position to LinuxCNC position feedback:

```
net Xpos-fb gm.0.encoder.0.position => axis.0.motor-pos-fb
```

#### 9.12.2.3 Stepgen module

The GM6-PCI motion control card has six stepgen modules, one for each axis. Each module has two output signals. It can produce Step/Direction, Up/Down or Quadrature (A/B) pulses. Each stepgen module is connected to the pins of the corresponding RJ50 axis connector.

Every stepgen pin and parameter name begins as follows:

```
gm.<nr. of card>.stepgen.<nr of axis>
```

,where nr of axis is form 0 to 5. For example:

```
gm.0.stepgen.0.position-cmd
```

refers to the position command of stepgen module of axis 0 on card 0.

The GM6-PCI card generates step pulses independently from LinuxCNC. Hal pins are updated by function

```
gm.<nr of card>.write
```

Table 9.9: Stepgen module pins

| Pins | Type and direction | Pin description |
|---|---|---|
| .enable | (bit, In) | Stepgen produces pulses only when this pin is true. |
| .count-fb | (s32, Out) | Position feedback in counts unit. |
| .position-fb | (float, Out) | Position feedback in position unit. |
| .position-cmd | (float, In) | Commanded position in position units. Used in position mode only. |
| .velocity-cmd | (float, In) | Commanded velocity in position units per second. Used in velocity mode only. |

Table 9.10: Stepgen module parameters

| Parameters | Type and Read/Write | Parameter description |
|---|---|---|
| .step-type | (u32, R/W) | When 0, module produces Step/Dir signal. When 1, it produces Up/Down step signals. And when it is 2, it produces quadrature output signals. |
| .control-type | (bit, R/W) | When True, .velocity-cmd is used as reference and velocity control calculate pulse rate output. When False, .position-cmd is used as reference and position control calculate pulse rate output. |
| .invert-step1 | (bit, R/W) | Invert the output of channel 1 (Step signal in StepDir mode) |
| .invert-step2 | (bit, R/W) | Invert the output of channel 2 (Dir signal in StepDir mode) |
| .maxvel | (float, R/W) | Maximum velocity in position units per second. If it is set to 0.0, .maxvel parameter is ignored. |
| .maxaccel | (float, R/W) | Maximum acceleration in position units per second squared. If it is set to 0.0, .maxaccel parameter is ignored. |
| .position-scale | (float, R/W) | Scale in steps per length unit. |
| .steplen | (u32, R/W) | Length of step pulse in nano-seconds. |
| .stepspace | (u32, R/W) | Minimum time between two step pulses in nano-seconds. |
| .dirdelay | (u32, R/W) | Minimum time between step pulse and direction change in nano-seconds. |

For evaluating the appropriate values see the timing diagrams below:

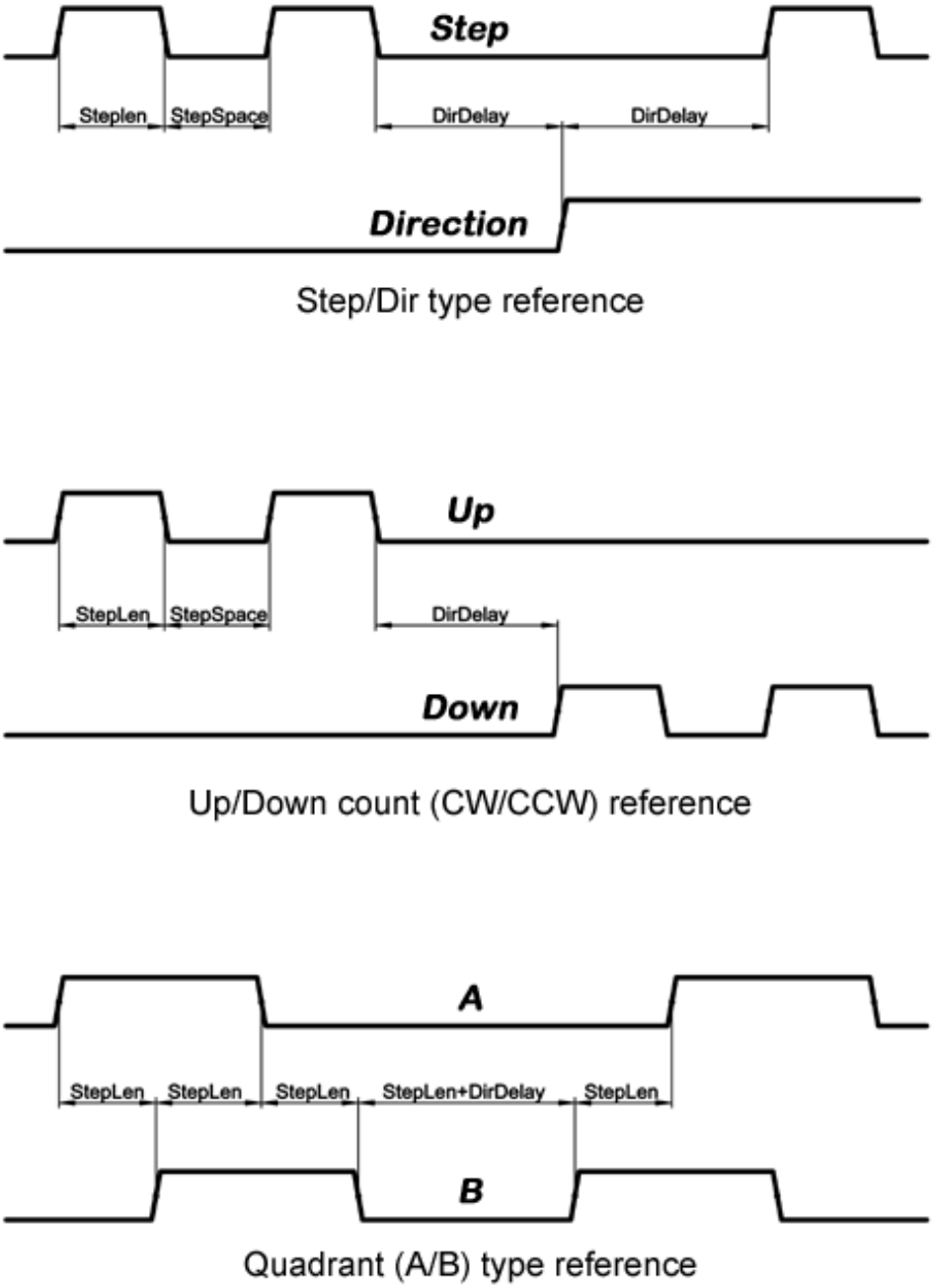



Figure 9.9: Reference signal timing diagrams

**HAL example** Setting stepgen module of axis 0 to generate 1000 step pulse per position unit:

```
setp gm.0.stepgen.0.step-type 0         # 0:stepDir, 1:UpDown, 2:Quad
setp gm.0.stepgen.0.control-type 0      # 0:Pos. control, 1:Vel. Control
setp gm.0.stepgen.0.invert-step1 0
setp gm.0.stepgen.0.invert-step2 0
setp gm.0.stepgen.0.maxvel 0            # do not set maxvel for step
                                        # generator, let interpolator control it.
setp gm.0.stepgen.0.maxaccel 0          # do not set max acceleration for
                                        # step generator, let interpolator control it.
setp gm.0.stepgen.0.position-scale 1000 # 1000 step/position unit
```

```
setp gm.0.stepgen.0.steplen 1000        # 1000 ns = 1 us
setp gm.0.stepgen.0.stepspace1000       # 1000 ns = 1 us
setp gm.0.stepgen.0.dirdelay 2000       # 2000 ns = 2 us
```

Connect stepgen to axis 0 position reference and enable pins:

```
net Xpos-cmd axis.0.motor-pos-cmd => gm.0.stepgen.0.position-cmd
net Xen axis.0.amp-enable-out => gm.0.stepgen.0.enable
```

#### 9.12.2.4 Enable and Fault signals

The GM6-PCI motion control card has one enable output and one fault input HAL pins, both are connected to each RJ50 axis connector and to the CAN connector.

Hal pins are updated by function:

```
gm.<nr of card>.read
```

Table 9.11: Enable and Fault signal pins

| Pins | Type and direction | Pin description |
|---|---|---|
| gm.<nr of card>.power-enable | (bit, In) | If this pin is True,<br>* and Watch Dog Timer is not expired * and there is no power fault Then power enable pins of axis- and CAN connectors are set to high, otherwise set to low. |
| gm.<nr of card>.power-fault | (bit, Out) | Power fault input. |

#### 9.12.2.5 Axis DAC

The GM6-PCI motion control card has six serial axis DAC driver modules, one for each axis. Each module is connected to the pin of the corresponding RJ50 axis connector. Every axis DAC pin and parameter name begins as follows:

```
gm.<nr. of card>.dac.<nr of axis>
```

,where nr of axis is form 0 to 5. For example:

```
gm.0.dac.0.value
```

refers to the output voltage of DAC module of axis 0. Hal pins are updated by function:

```
gm.<nr of card>.write
```

Table 9.12: Axis DAC pins

| Pins | Type and direction | Pin description |
|---|---|---|
| .enable | (bit, In) | Enable DAC output. When enable is false, DAC output is 0.0 V. |
| .value | (float, In) | Value of DAC output in Volts. |

Table 9.13: Axis DAC parameters

| Parameters | Type and direction | Parameter description |
|---|---|---|
| .offset | (float, R/W) | Offset is added to the value before the hardware is updated |
| .high-limit | (float, R/W) | Maximum output voltage of the hardware in volts. |
| .low-limit | (float, R/W) | Minimum output voltage of the hardware in volts. |
| .invert-serial | (float, R/W) | GM6-PCI card is communicating with DAC hardware via fast serial communication to highly reduce time delay compared to PWM. DAC module is recommended to be isolated which is negating serial communication line. In case of isolation, leave this parameter to default (0), while in case of none-isolation, set this parameter to 1. |

### 9.12.3 CAN-bus servo amplifiers

The GM6-PCI motion control card has CAN module to drive CAN servo amplifiers. Implementation of higher level protocols like CANopen is further development. Currently GM produced power amplifiers has upper level driver which export pins and parameters to HAL. They receive position reference and provide encoder feedback via CAN bus.

The frames are standard (11 bit) ID frames, with 4 byte data length. Tha baud rate is 1 Mbit. The position commad IDs for axis 0..5 are 0x10..0x15. The position feedback IDs for axis 0..5 are 0x20..0x25.

These configuration can be changed with the modifivation of hal_gm.c and recompiling LinuxCNC.

Every CAN pin and parameter name begins as follows:

```
gm.<nr. of card>.can-gm.<nr of axis>
```

,where <nr of axis> is form 0 to 5. For example:

```
gm.0.can-gm.0.position
```

refers to the output position of axis 0 in position units.

Hal pins are updated by function:

```
gm.<nr of card>.write
```

#### 9.12.3.1 Pins

Table 9.14: CAN module pins

| Pins | Type and direction | Pin description |
|---|---|---|
| .enable | (bit, In) | Enable sending position references. |
| .position-cmd | (float, In) | Commanded position in position units. |
| .position-fb | (float, In) | Feed back position in position units. |

#### 9.12.3.2 Parameters

Table 9.15: CAN module parameters

| Parameters | Type and direction | Parameter description |
|---|---|---|
| .position-scale | (float, R/W) | Scale in per length unit. |

### 9.12.4 Watchdog timer

Watchdog timer resets at function:

```
gm.<nr of card>.read
```

#### 9.12.4.1 Pins

Table 9.16: Watchdog pins

| Pins | Type and direction | Pin description |
|---|---|---|
| gm.<nr of card>.watchdog-expired | (bit, Out) | Indicates that watchdog timer is expired. |

Watchdog timer overrun causes the set of power-enable to low in hardware.

#### 9.12.4.2 Parameters

Table 9.17: Watchdog parameters

| Parameters | Type and direction | Parameter description |
|---|---|---|
| gm.<nr of card>.watchdog-enable | (bit, R/W) | Enable watchdog timer. It is strongly recommended to enable watchdog timer, because it can disables all the servo amplifiers by pulling down all enable signal in case of PC error. |
| gm.<nr of card>.watchdog-timeout-ns | (float, R/W) | Time interval in within the gm.<nr of card>.read function must be executed. The gm.<nr of card>.read is typically added to servo-thread, so watch timeout is typically set to 3 times of the servo period. |

### 9.12.5 End-, homing- and E-stop switches

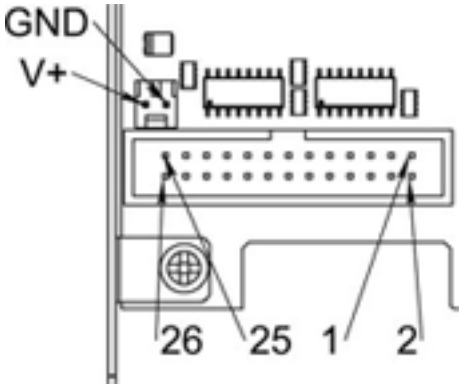



Figure 9.10: Pin numbering of homing & end switch connector

Table 9.18: End- and homing switch connector pinout

| 25 | 23 | 21 | 19 | 17 | 15 | 13 | 11 | 9 | 7 | 5 | 3 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| GND | | 1/End- | 2/End+ | 2/Homing | 3/End- | 4/End+ | 4/Homing | 5/End- | 6/End+ | 6/Homing | E-Stop 2 | V+ (Ext.) |

| 26 | 24 | 22 | 20 | 18 | 16 | 14 | 12 | 10 | 8 | 6 | 4 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| GND | | 1/End+ | 1/Homing | 2/End- | 3/End+ | 3/Homing | 4/End- | 5/End+ | 5/Homing | 6/End- | E-Stop 1 | V+ (Ext.) |

The GM6-PCI motion control card has two limit- and one homing switch input for each axis. All the names of these pins begin as follows:

```
gm.<nr. of card>.axis.<nr of axis>
```

,where nr of axis is form 0 to 5. For example:

```
gm.0.axis.0.home-sw-in
```

indicates the state of the axis 0 home switch.

Hal pins are updated by function:

```
gm.<nr of card>.read
```

#### 9.12.5.1 Pins

Table 9.19: End- and homing switch pins

| Pins | Type and direction | Pin description |
|---|---|---|
| .home-sw-in | (bit, Out) | Home switch input |
| .home-sw-in-not | (bit, Out) | Negated home switch input |
| .neg-lim-sw-in | (bit, Out) | Negative limit switch input |
| .neg-lim-sw-in-not | (bit, Out) | Negated negative limit switch input |
| .pos-lim-sw-in | (bit, Out) | Positive limit switch input |
| .pos-lim-sw-in-not | (bit, Out) | Negated positive limit switch input |

#### 9.12.5.2 Parameters

Table 9.20: E-stop switch parameters

| Parameters | Type and direction | Parameter description |
|---|---|---|
| gm.0.estop.0.in | (bit, Out) | Estop 0 input |
| gm.0.estop.0.in-not | (bit, Out) | Negated Estop 0 input |
| gm.0.estop.1.in | (bit, Out) | Estop 1 input |
| gm.0.estop.1.in-not | (bit, Out) | Negated Estop 1 input |

### 9.12.6 Status LEDs

#### 9.12.6.1 CAN

Color: Orange

- Blink, during data communication.
- On, when any of the buffers are full - communication error.
- Off, when no data communication.

#### 9.12.6.2 RS485

Color: Orange

- Blink, during initialization of modules on the bus
- On, when the data communication is up between all initialized modules.
- Off, when any of the initialized modules dropped off because of an error.

#### 9.12.6.3 EMC

Color: White

- Blink, when LinuxCNC is running.
- Otherwise off.

#### 9.12.6.4 Boot

Color: Green

- On, when system booted successfully.
- Otherwise off.

#### 9.12.6.5 Error

Color: Red

- Off, when there is no fault in the system.
- Blink, when PCI communication error.
- On, when watchdog timer overflowed.

### 9.12.7 RS485 I/O expander modules

These modules were developed for expanding the I/O and function capability along an RS485 line of the GM6-PCI motion control card.

Available module types:

- 8-channel relay output module - gives eight NO-NC relay output on a three pole terminal connector for each channel.
- 8-channel digital input module - gives eight optical isolated digital input pins.
- 8 channel ADC and 4-channel DAC module - gives four digital-to-analogue converter outputs and eight analogue-to-digital inputs. This module is also optically isolated from the GM6-PCI card.

**Automatic node recognizing:**

Each node connected to the bus was recognized by the GM6-PCI card automatically. During starting LinuxCNC, the driver export pins and parameters of all available modules automatically.

**Fault handling:**

If a module does not answer regularly the GM6-PCI card drops down the module. If a module with output do not gets data with correct CRC regularly, the module switch to error sate (green LED blinking), and turns all outputs to error sate.

**Connecting the nodes:**

The modules on the bus have to be connected in serial topology, with termination resistors on the end. The start of the topology is the PCI card, and the end is the last module.

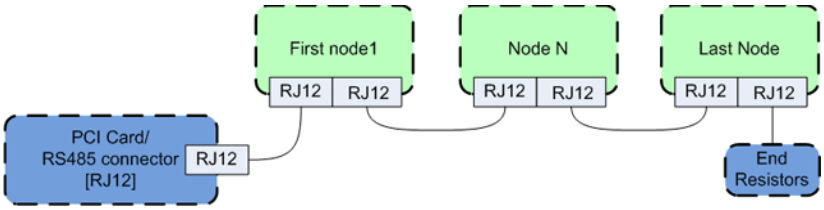



Figure 9.11: Connecting the RS485 nodes to the GM6-PCI card

**Adressing:**

Each node on the bus has a 4 bit unique address that can be set with a red DIP switch.

**Status LED:**

A green LED indicates the status of the module:

- Blink, when the module is only powered, but not jet identified, or when module is dropped down.
- Off, during identification (computer is on, but LinuxCNC not started)
- On, when it communicates continuously.

#### 9.12.7.1 Relay output module

For pinout, connection and electrical charasteristics of the module, please refer to the System integration manual.

All the pins and parameters are updated by the following function:

```
gm.<nr. of card>.rs485
```

It should be added to servo thread or other thread with larger period to avoid CPU overload. Every RS485 module pin and parameter name begins as follows:

```
gm.<nr. of card>.rs485.<modul ID>
```

,where <modul ID> is form 00 to 15.

Table 9.21: Relay output module pins

| Pins | Type and direction | Pin description |
|---|---|---|
| .relay-<0-7> | (bit, Out) | Output pin for relay |

Table 9.22: Relay output module parameters

| Parameters | Type and direction | Parameter description |
|---|---|---|
| .invert-relay-<0-7> | (bit, R/W) | Negate relay output pin |

**HAL example**

```
gm.0.rs485.0.relay-0 # First relay of the node.
gm.0                 # Means the first GM6-PCI motion control card (PCI card address = 0)
.rs485.0             # Select node with address 0 on the RS485 bus
.relay-0             # Select the first relay
```

#### 9.12.7.2 Digital input module

For pinout, connection and electrical charasteristics of the module, please refer to the System integration manual.

All the pins and parameters are updated by the following function:

```
gm.<nr. of card>.rs485
```

It should be added to servo thread or other thread with larger period to avoid CPU overload. Every RS485 module pin and parameter name begins as follows:

```
gm.<nr. of card>.rs485.<modul ID>
```

,where <modul ID> is form 00 to 15.

Table 9.23: Digital input output module pins

| Pins | Type and direction | Pin description |
|---|---|---|
| .in-<0-7> | (bit, Out) | Input pin |
| .in-not-<0-7> | (bit, Out) | Negated input pin |

**HAL example**

```
gm.0.rs485.0.in-0 # First input of the node.
# gm.0     - Means the first GM6-PCI motion control card (PCI card address = 0)
# .rs485.0 - Select node with address 0 on the RS485 bus
# .in-0    - Select the first digital input module
```

#### 9.12.7.3 DAC & ADC module

For pinout, connection and electrical charasteristics of the module, please refer to the System integration manual.

All the pins and parameters are updated by the following function:

```
gm.<nr. of card>.rs485
```

It should be added to servo thread or other thread with larger period to avoid CPU overload. Every RS485 module pin and parameter name begins as follows:

```
gm.<nr. of card>.rs485.<modul ID>
```

,where <modul ID> is form 00 to 15.

Table 9.24: DAC & ADC module pins

| Pins | Type and direction | Pin description |
|---|---|---|
| .adc-<0-7> | (float, Out) | Value of ADC input in Volts. |
| .dac-enable-<0-3> | (bit, In) | Enable DAC output. When enable is false DAC output is set to 0.0 V. |
| .dac-<0-3> | (float, In) | Value of DAC output in Volts. |

Table 9.25: DAC & ADC module parameters

| Parameters | Type and direction | Parameter description |
|---|---|---|
| .adc-scale-<0-7> | (float, R/W) | The input voltage will be multiplied by scale before being output to .adc- pin. |
| .adc-offset-<0-7> | (float, R/W) | Offset is subtracted from the hardware input voltage after the scale multiplier has been applied. |
| .dac-offset-<0-3> | (float, R/W) | Offset is added to the value before the hardware is updated. |
| .dac-high-limit-<0-3> | (float, R/W) | Maximum output voltage of the hardware in volts. |
| .dac-low-limit-<0-3> | (float, R/W) | Minimum output voltage of the hardware in volts. |

**HAL example**

```
gm.0.rs485.0.adc-0 # First analogue channel of the node.
# gm.0     - Means the first GM6-PCI motion control card (PCI card address = 0)
# .rs485.0 - Select node with address 0 on the RS485 bus
# .adc-0   - Select the first analogue input of the module
```

#### 9.12.7.4 Teach Pendant module

For pinout, connection and electrical charasteristics of the module, please refer to the System integration manual.

All the pins and parameters are updated by the following function:

```
gm.<nr. of card>.rs485
```

It should be added to servo thread or other thread with larger period to avoid CPU overload. Every RS485 module pin and parameter name begins as follows:

```
gm.<nr. of card>.rs485.<modul ID>
```

,where <modul ID> is form 00 to 15. Note that on the Teach Pendant module it cannot be changed, and pre-programmed as zero. Upon request it can be delivered with firmware pre-programmed different ID.

Table 9.26: Teach Pendant module pins

| Pins | Type and direction | Pin description |
|---|---|---|
| .adc-<0-5> | (float, Out) | Value of ADC input in Volts. |
| .enc-reset | (bit, In) | When True, resets counts and position to zero. |
| .enc-counts | (s32, Out) | Position in encoder counts. |
| .enc-rawcounts | (s32, Out) | The raw count is the counts, but unaffected by reset. |
| .enc-position | (float, Out) | Position in scaled units (=.enc-counts/.enc-position-scale). |
| .in-<0-7> | (bit, Out) | Input pin |
| .in-not-<0-7> | (bit, Out) | Negated input pin |

Table 9.27: Teach Pendant module parameters

| Parameters | Type and direction | Parameter description |
|---|---|---|
| .adc-scale-<0-5> | (float, R/W) | The input voltage will be multiplied by scale before being output to .adc- pin. |
| .adc-offset-<0-5> | (float, R/W) | Offset is subtracted from the hardware input voltage after the scale multiplier has been applied. |
| .enc-position-scale | (float, R/W) | Scale in per length unit. |

**HAL example**

```
gm.0.rs485.0.adc-0 # First analogue channel of the node.
# gm.0     - Means the first GM6-PCI motion control card (PCI card address = 0)
# .rs485.0 - Select node with address 0 on the RS485 bus
# .adc-0   - Select the first analogue input of the module
```

### 9.12.8 Errata

#### 9.12.8.1 GM6-PCI card Errata

The revision number in this section refers to the revision of the GM6-PCI card device.

REV. 1.2

- Error: The PCI card do not boot, when Axis 1. END B switch is active (low). Found on November 16, 2013.
- Reason: This switch is connected to a boot setting pin of FPGA
- Problem fix/workaround: Use other switch pin, or connect only normally open switch to this switch input pin.

## 9.13 VFS11 VFD Driver

This is a userspace HAL program to control the S11 series of VFD's from Toshiba.

vfs11_vfd supports serial and TCP connections. Serial connections may be RS232 or RS485. RS485 is supported in full- and half-duplex mode. TCP connections may be passive (wait for incoming connection), or active outgoing connections, which may be useful to connect to TCP-based devices or through a terminal server.

Regardless of the connection type, vfs11_vfd operates as a Modbus master.

This component is loaded using the halcmd "loadusr" command:

```
loadusr -Wn spindle-vfd vfs11_vfd -n spindle-vfd
```

The above command says: loadusr, wait for named to load, component vfs11_vfd, named spindle-vfd

### 9.13.1 Command Line Options

*vfs11_vfd* is mostly configured through inifile options. The command line options are:

- *-n or --name <halname>* : set the HAL component name
- *-I or --ini <inifilename>* : take configuration from this ini file. Defaults to environment variable INI_FILE_NAME.
- *-S or --section <section name>* : take configuration from this section in the ini file. Defaults to *VFS11*.
- *-d or --debug* enable debug messages on console output.
- *-m or --modbus-debug* enable modbus messages on console output
- *-r or --report-device* report device properties on console at startup

Debugging can be toggled by sending a USR1 signal to the vfs11_vfd process. Modbus debugging can be toggled by sending a USR2 signal to vfs11_vfd process (example: `kill -USR1 `pidof vfs11_vfd`).

**Note**
That if there are serial configuration errors, turning on verbose may result in a flood of timeout errors.

### 9.13.2 Pins

Where <n> is `vfs11_vfd` or the name given during loading with the -n option.

- *<n>.acceleration-pattern* (bit, in) when true, set acceleration and deceleration times as defined in registers F500 and F501 respecitvely. Used in PID loops to choose shorter ramp times to avoid oscillation.
- *<n>.alarm-code* (s32, out) non-zero if drive is in alarmed state. Bitmap describing alarm information (see register FC91 description). Use err-reset (see below) to clear the alarm.
- *<n>.at-speed* (bit, out) when drive is at commanded speed (see speed-tolerance below)
- *<n>.current-load-percentage* (float, out) reported from the VFD
- *<n>.dc-brake* (bit, in) engage the DC brake. Also turns off spindle-on.
- *<n>.enable* (bit, in) enable the VFD. If false, all operating parameters are still read but control is released and panel control is enabled (subject to VFD setup).
- *<n>.err-reset* (bit, in) reset errors (alarms a.k.a Trip and e-stop status). Resetting the VFD may cause a 2-second delay until it's rebooted and Modbus is up again.
- *<n>.estop* (bit, in) put the VFD into emergency-stopped status. No operation possible until cleared with err-reset or powercycling.

- *<n>.frequency-command* (float, out) current target frequency in HZ as set through speed-command (which is in RPM), from the VFD
- *<n>.frequency-out* (float, out) current output frequency of the VFD
- *<n>.inverter-load-percentage* (float, out) current load report from VFD
- *<n>.is-e-stopped* (bit, out) the VFD is in emergency stop status (blinking "E" on panel). Use err-reset to reboot the VFD and clear the e- stop status.
- *<n>.is-stopped* (bit, out) true when the VFD reports 0 Hz output
- *<n>.max-rpm* (float, R) actual RPM limit based on maximum frequency the VFD may generate, and the motors nameplate values. For instance, if nameplate-HZ is 50, and nameplate-RPM_ is 1410, but the VFD may generate up to 80Hz, then max- rpm would read as 2256 (80*1410/50). The frequency limit is read from the VFD at startup. To increase the upper frequency limit, the UL and FH parameters must be changed on the panel. See the VF-S11 manual for instructions how to set the maximum frequency.
- *<n>.modbus-ok* (bit, out) true when the Modbus session is successfully established and the last 10 transactions returned without error.
- *<n>.motor-RPM* (float, out) estimated current RPM value, from the VFD
- *<n>.output-current-percentage* (float, out) from the VFD
- *<n>.output-voltage-percentage* (float, out) from the VFD
- *<n>.output-voltage* (float, out) from the VFD
- *<n>.speed-command* (float, in) speed sent to VFD in RPM. It is an error to send a speed faster than the Motor Max RPM as set in the VFD
- *<n>.spindle-fwd* (bit, in) 1 for FWD and 0 for REV, sent to VFD
- *<n>.spindle-on* (bit, in) 1 for ON and 0 for OFF sent to VFD, only on when running
- *<n>.spindle-rev* (bit, in) 1 for ON and 0 for OFF, only on when running
- *<n>.jog-mode* (bit, in) 1 for ON and 0 for OFF, enables the VF-S11 *jog mode*. Speed control is disabled, and the output frequency is determined by register F262 (preset to 5Hz). This might be useful for spindle orientation. In normal mode, the VFD shuts off if the frequency drops below 12Hz.
- *<n>.status* (s32, out) Drive Status of the VFD (see the TOSVERT VF-S11 Communications Function Instruction Manual, register FD01). A bitmap.
- *<n>.trip-code* (s32, out) trip code if VF-S11 is in tripped state.
- *<n>.error-count* (s32, out) number of Modbus transactions which returned an error
- *<n>.max-speed* (bit, in) ignore the loop-time parameter and run Modbus at maximum speed, at the expense of higher CPU usage. Suggested use during spindle positioning.

### 9.13.3 Parameters

Where <n> is `vfs11_vfd` or the name given during loading with the -n option.

- *<n>.frequency-limit* (float, RO) upper limit read from VFD setup.
- *<n>.loop-time* (float, RW) how often the Modbus is polled (default interval 0.1 seconds)
- *<n>.nameplate-HZ* (float, RW) Nameplate Hz of motor (default 50). Used to calculate target frequency (together with nameplate-RPM ) for a target RPM value as given by speed-command.
- *<n>.nameplate-RPM* (float, RW) Nameplate RPM of motor (default 1410)

- *<n>.rpm-limit* (float, RW) do-not-exceed soft limit for motor RPM (defaults to nameplate-RPM ).
- *<n>.tolerance* (float, RW) speed tolerance (default 0.01) for determining wether spindle is at speed (0.01 meaning: output frequency is within 1% of target frequency)

### 9.13.4 INI file configuration

This lists all options understood by vfs11_vfd. Typical setups for RS232, RS485 and TCP can be found in *src/hal/user_comps/vfs11_vfd/*

```
[VFS11]
# serial connection
TYPE=rtu

# serial port
DEVICE=/dev/ttyS0

# TCP server - wait for incoming connection
TYPE=tcpserver

# tcp portnumber for TYPE=tcpserver or tcpclient
PORT=1502

# TCP client - active outgoing connection
TYPE=tcpclient

# destination to connect to if TYPE=tcpclient
TCPDEST=192.168.1.1

#---------- meaningful only if TYPE=rtu -------
# serial device detail
# 5 6 7 8
BITS= 5

# even odd none
PARITY=none

# 110, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200
BAUD=19200

# 1 2
STOPBITS=1

#rs232 rs485
SERIAL_MODE=rs485

# up down none
# this feature might not work with a stock Ubuntu
# libmodbus5/libmodbus-dev package, and generate a warning
# execution will continue as if RTS_MODE=up were given.
RTS_MODE=up
#---------------------

# modbus timers in seconds
# inter-character timer
BYTE_TIMEOUT=0.5
# packet timer
RESPONSE_TIMEOUT=0.5

# target modbus ID
TARGET=1
```

```
# on I/O failure, try to reconnect after sleeping
# for RECONNECT_DELAY seconds
RECONNECT_DELAY=1

# misc flags
DEBUG=10
MODBUS_DEBUG=0
POLLCYCLES=10
```

### 9.13.5 HAL example

```
#
# example usage of the VF-S11 VFD driver
#
#
loadusr -Wn spindle-vfd vfs11_vfd -n spindle-vfd

# connect the spindle direction pins to the VFD
net vfs11-fwd spindle-vfd.spindle-fwd <= motion.spindle-forward
net vfs11-rev spindle-vfd.spindle-rev <= motion.spindle-reverse

# connect the spindle on pin to the VF-S11
net vfs11-run spindle-vfd.spindle-on <= motion.spindle-on

# connect the VF-S11 at speed to the motion at speed
net vfs11-at-speed motion.spindle-at-speed <= spindle-vfd.at-speed

# connect the spindle RPM to the VF-S11
net vfs11-RPM spindle-vfd.speed-command <= motion.spindle-speed-out

# connect the VF-S11 DC brake
# since this draws power during spindle off, the dc-brake pin would
# better be driven by a monoflop which triggers on spindle-on falling edge
#net vfs11-spindle-brake motion.spindle-brake => spindle-vfd.dc-brake

# to use the VFS11 jog mode for spindle orient
# see orient.9 and motion.9
net spindle-orient motion.spindle-orient spindle-vfd.max-speed spindle-vfd.jog-mode

# take precedence over control panel
setp spindle-vfd.enable 1
```

### 9.13.6 Panel operation

The vfs11_vfd driver takes precedence over panel control while it is enabled (see *enable* pin), effectively disabling the panel. Clearing the *enable* pin re-enables the panel. Pins and parameters can still be set, but will not be written to the VFD untile the *enable* pin is set. Operating parameters are still read while bus control is disabled. Exiting the vfs11_vfd driver in a controlled way will release the VFD from the bus and restore panel control.

See the EMC2 Integrators Manual for more information. For a detailed register description of the Toshiba VFD's, see the "TOSVERT VF-S11 Communications Function Instruction Manual" (Toshiba document number E6581222) and the "TOSVERT VF-S11 Instruction manual" (Toshiba document number E6581158).

### 9.13.7 Error Recovery

`vfs11_vfd` recovers from I/O errors as follows: First, all HAL pins are set to default values, and the driver will sleep for `RECONNECT_DELAY` seconds (default 1 second).

- Serial (`TYPE=rtu`) mode: on error, close and reopen the serial port.
- TCP server (`TYPE=tcpserver`) mode: on losing the TCP connection, the driver will go back to listen for incoming connections.
- TCP client (`TYPE=tcpclient`) mode: on losing the TCP connection, the driver will reconnect to *TCPDEST:PORTNO*.

### 9.13.8 Configuring the VFS11 VFD for Modbus usage

#### 9.13.8.1 Connecting the Serial Port

The VF-S11 has an RJ-45 jack for serial communication. Unfortunately, it does not have a standard RS-232 plug and logic levels. The Toshiba-recommended way is: connect the USB001Z USB-to-serial conversion unit to the drive, and plug the USB port into the PC. A cheaper alternative is a homebrew interface ( hints from Toshiba support, circuit diagram).

Note: the 24V output from the VFD has no short-circuit protection.

Serial port factory defaults are 9600/8/1/even, the protocol defaults to the proprietary "Toshiba Inverter Protocol".

#### 9.13.8.2 Modbus setup

Several parameters need setting before the VF-S11 will talk to this module. This can either be done manually with the control panel, or over the serial link - Toshiba supplies a Windows application called *PCM001Z* which can read/set parameters in the VFD. Note - PCM001Z only talks the Toshiba inverter protocol. So the last parameter which you'd want to change is the protocol - set from Toshiba Inverter Protocol to Modbus; thereafter, the Windows app is useless.

To increase the upper frequency limit, the UL and FH parameters must be changed on the panel. I increased them from 50 to 80.

See dump-params.mio for a description of non-standard VF-S11 parameters of my setup. This file is for the modio Modbus interactive utility.

### 9.13.9 Programming Note

The vfs11_vfd driver uses the libmodbus version 3 library which is more recent than the version 2 code used in `gs2_vfd`.

The Ubuntu `libmodbus5` and `libmodbus-dev` packages are only available starting from Ubuntu 12 (*Precise Pengolin*). Moreover, these packages lack support for the MODBUS_RTS_MODE_* flags. Therefore, building vfs11_vfd using this library might generate a warning if RTS_MODE= is specified in the ini file.

To use the full functionality on lucid and precise: * remove the libmodbus packages: `sudo apt-get remove libmodbus5 libmodbus-dev` * build and install libmodbus version 3 from source as outlined here.

Libmodbus does not build on Ubuntu Hardy, hence vfs11_vfd is not available on hardy.

# Chapter 10

# Driver Examples

## 10.1 PCI Parallel Port

When you add a second parallel port to your PCI bus you have to find out the address before you can use it with LinuxCNC.

To find the address of your parallel port card open a terminal window and type

```
lspci -v
```

You will see something similar to this as well as info on everything else on the PCI bus:

```
0000:00:10.0 Communication controller: \
        NetMos Technology PCI 1 port parallel adapter (rev 01)
        Subsystem: LSI Logic / Symbios Logic: Unknown device 0010
        Flags: medium devsel, IRQ 11
        I/O ports at a800 [size=8]
        I/O ports at ac00 [size=8]
        I/O ports at b000 [size=8]
        I/O ports at b400 [size=8]
        I/O ports at b800 [size=8]
        I/O ports at bc00 [size=16]
```

In my case the address was the first one so I changed my .hal file from

```
loadrt hal_parport cfg=0x378
```

to

```
loadrt hal_parport cfg="0x378 0xa800 in"
```

(Note the double quotes surrounding the addresses.)

and then added the following lines so the parport will be read and written:

```
addf parport.1.read base-thread
addf parport.1.write base-thread
```

After doing the above then run your config and verify that the parallel port got loaded in Machine/Show HAL Configuration window.

## 10.2 Spindle Control

### 10.2.1 0-10v Spindle Speed

If your spindle speed is controlled by an analog signal, (for example, by a VFD with a 0 to 10 volt signal) and you're using a DAC card like the m5i20 to output the control signal:

First you need to figure the scale of spindle speed to control signal. For this example the spindle top speed of 5000 RPM is equal to 10 volts.

$$\frac{10\,Volts}{5000\,RPM} = \frac{0.002\,Volts}{1\,RPM}$$

We have to add a scale component to the HAL file to scale the motion.spindle-speed-out to the 0 to 10 needed by the VFD if your DAC card does not do scaling.

```
loadrt scale count=1
addf scale.0 servo-thread
setp scale.0.gain 0.002
net spindle-speed-scale motion.spindle-speed-out => scale.0.in
net spindle-speed-DAC scale.0.out => <your DAC pin name>
```

### 10.2.2 PWM Spindle Speed

If your spindle can be controlled by a PWM signal, use the pwmgen component to create the signal:

```
loadrt pwmgen output_type=0
addf pwmgen.update servo-thread
addf pwmgen.make-pulses base-thread
net spindle-speed-cmd motion.spindle-speed-out => pwmgen.0.value
net spindle-on motion.spindle-on => pwmgen.0.enable
net spindle-pwm pwmgen.0.pwm => parport.0.pin-09-out
# Set the spindle's top speed in RPM
setp pwmgen.0.scale 1800
```

This assumes that the spindle controller's response to PWM is simple: 0% PWM gives 0 RPM, 10% PWM gives 180 RPM, etc. If there is a minimum PWM required to get the spindle to turn, follow the example in the nist-lathe sample configuration to use a scale component.

### 10.2.3 Spindle Enable

If you need a spindle enable signal, link your output pin to motion.spindle-on. To link these pins to a parallel port pin put something like the following in your .hal file, making sure you pick the pin that is connected to your control device.

```
net spindle-enable motion.spindle-on => parport.0.pin-14-out
```

### 10.2.4 Spindle Direction

If you have direction control of your spindle the HAL pins motion.spindle-forward and motion.spindle-reverse are controlled by M3 and M4. Spindle speed *Sn* must be set to a positive non-zero value for M3/M4 to turn on spindle motion.

To link these pins to a parallel port pin, put something like the following in your .hal file making sure you pick the pin that is connected to your control device.

```
net spindle-fwd motion.spindle-forward => parport.0.pin-16-out
net spindle-rev motion.spindle-reverse => parport.0.pin-17-out
```

### 10.2.5 Spindle Soft Start

If you need to ramp your spindle speed command and your control does not have that feature it can be done in HAL. Basically you need to hijack the output of motion.spindle-speed-out and run it through a limit2 component with the scale set so it will ramp the rpm from motion.spindle-speed-out to your device that receives the rpm. The second part is to let LinuxCNC know when the spindle is at speed so motion can begin.

In the 0-10 volt example the line *net spindle-speed-scale motion.spindle-speed-out => scale.0.in* is changed as shown in the following example:

**Intro to HAL components limit2 and near:**

In case you have not run across them before, here's a quick introduction to the two HAL components used in the following example.

- A "limit2" is a HAL component (floating point) that accepts an input value and provides an output that has been limited to a max/min range, and also limited to not exceed a specified rate of change.
- A "near" is a HAL component (floating point) with a binary output that says whether two inputs are approximately equal.

More info is available in the documentation for HAL components, or from the man pages, just say *man limit2* or *man near* in a terminal.

```
# load real time a limit2 and a near with names so it is easier to follow
loadrt limit2 names=spindle-ramp
loadrt near names=spindle-at-speed

# add the functions to a thread
addf spindle-ramp servo-thread
addf spindle-at-speed servo-thread

# set the parameter for max rate-of-change
# (max spindle accel/decel in units per second)
setp spindle-ramp.maxv 60

# hijack the spindle speed out and send it to spindle ramp in
net spindle-cmd <= motion.spindle-speed-out => spindle-ramp.in

# the output of spindle ramp is sent to the scale in
net spindle-ramped <= spindle-ramp.out => scale.0.in

# to know when to start the motion we send the near component
# (named spindle-at-speed) to the spindle commanded speed from
# the signal spindle-cmd and the actual spindle speed
# provided your spindle can accelerate at the maxv setting.
net spindle-cmd => spindle-at-speed.in1
net spindle-ramped => spindle-at-speed.in2

# the output from spindle-at-speed is sent to motion.spindle-at-speed
# and when this is true motion will start
net spindle-ready <= spindle-at-speed.out => motion.spindle-at-speed
```

### 10.2.6 Spindle Feedback

#### 10.2.6.1 Spindle Synchronized Motion

Spindle feedback is needed by LinuxCNC to perform any spindle coordinated motions like threading and constant surface speed. The StepConf Wizard can perform the connections for you if you select Encoder Phase A and Encoder Index as inputs.

Hardware assumptions:

- An encoder is connected to the spindle and puts out 100 pulses per revolution on phase A
- The encoder A phase is connected to the parallel port pin 10
- The encoder index pulse is connected to the parallel port pin 11

Basic Steps to add the components and configure them: [1] [2] [3]

```
# add the encoder to HAL and attach it to threads.
loadrt encoder num_chan=1
addf encoder.update-counters base-thread
addf encoder.capture-position servo-thread

# set the HAL encoder to 100 pulses per revolution.
setp encoder.3.position-scale 100

# set the HAL encoder to non-quadrature simple counting using A only.
setp encoder.3.counter-mode true

# connect the HAL encoder outputs to LinuxCNC.
net spindle-position encoder.3.position => motion.spindle-revs
net spindle-velocity encoder.3.velocity => motion.spindle-speed-in
net spindle-index-enable encoder.3.index-enable <=> motion.spindle-index-enable

# connect the HAL encoder inputs to the real encoder.
net spindle-phase-a encoder.3.phase-A <= parport.0.pin-10-in
net spindle-phase-b encoder.3.phase-B
net spindle-index encoder.3.phase-Z <= parport.0.pin-11-in
```

#### 10.2.6.2 Spindle At Speed

To enable LinuxCNC to wait for the spindle to be at speed before executing a series of moves you need to set motion.spindle-at-speed to true when the spindle is at the commanded speed. To do this you need spindle feedback from an encoder. Since the feedback and the commanded speed are not usually *exactly* the same you need to use the *near* component to say that the two numbers are close enough.

The connections needed are from the spindle velocity command signal to near.n.in1 and from the spindle velocity from the encoder to near.n.in2. Then the near.n.out is connected to motion.spindle-at-speed. The near.n.scale needs to be set to say how close the two numbers must be before turning on the output. Depending on your setup you may need to adjust the scale to work with your hardware.

The following is typical of the additions needed to your HAL file to enable Spindle At Speed. If you already have near in your HAL file then increase the count and adjust code to suit. Check to make sure the signal names are the same in your HAL file.

```
# load a near component and attach it to a thread
loadrt near
addf near.0 servo-thread

# connect one input to the commanded spindle speed
net spindle-cmd => near.0.in1

# connect one input to the encoder-measured spindle speed
net spindle-velocity => near.0.in2

# connect the output to the spindle-at-speed input
net spindle-at-speed motion.spindle-at-speed <= near.0.out
```

[1] In this example, we will assume that some encoders have already been issued to axes/joints 0, 1, and 2. So the next encoder available for us to attach to the spindle would be number 3. Your situation may differ.

[2] The HAL encoder index-enable is an exception to the rule in that it behaves as both an input and an output, see the Encoder Section for details

[3] It is because we selected *non-quadrature simple counting...* above that we can get away with *quadrature* counting without having any B quadrature input.

```
# set the spindle speed inputs to agree if within 1%
setp near.0.scale 1.01
```

## 10.3 MPG Pendant

This example is to explain how to hook up the common MPG pendants found on the market today. This example uses an MPG3 pendant and a C22 pendant interface card from CNC4PC connected to a second parallel port plugged into the PCI slot. This example gives you 3 axes with 3 step increments of 0.1, 0.01, 0.001

In your custom.hal file or jog.hal file add the following, making sure you don't have mux4 or an encoder already in use. If you do just increase the counts and change the reference numbers. More information about mux4 and encoder can be found in the HAL manual or the man page.

See the HAL Ini Section of the manual for more information on adding a hal file.

**jog.hal**

```
# Jog Pendant
loadrt encoder num_chan=1
loadrt mux4 count=1
addf encoder.capture-position servo-thread
addf encoder.update-counters base-thread
addf mux4.0 servo-thread

# If your MPG outputs a quadrature signal per click set x4 to 1
# If your MPG puts out 1 pulse per click set x4 to 0
setp encoder.0.x4-mode 0

# For velocity mode, set to 1
# In velocity mode the axis stops when the dial is stopped
# even if that means the commanded motion is not completed,
# For position mode (the default), set to 0
# In position mode the axis will move exactly jog-scale
# units for each count, regardless of how long that might take,
setp axis.0.jog-vel-mode 0
setp axis.1.jog-vel-mode 0
setp axis.2.jog-vel-mode 0

# This sets the scale that will be used based on the input to the mux4
setp mux4.0.in0 0.1
setp mux4.0.in1 0.01
setp mux4.0.in2 0.001

# The inputs to the mux4 component
net scale1 mux4.0.sel0 <= parport.1.pin-09-in
net scale2 mux4.0.sel1 <= parport.1.pin-10-in

# The output from the mux4 is sent to each axis jog scale
net mpg-scale <= mux4.0.out
net mpg-scale => axis.0.jog-scale
net mpg-scale => axis.1.jog-scale
net mpg-scale => axis.2.jog-scale

# The MPG inputs
net mpg-a encoder.0.phase-A <= parport.1.pin-02-in
net mpg-b encoder.0.phase-B <= parport.1.pin-03-in

# The Axis select inputs
net mpg-x axis.0.jog-enable <= parport.1.pin-04-in
net mpg-y axis.1.jog-enable <= parport.1.pin-05-in
```

```
net mpg-z axis.2.jog-enable <= parport.1.pin-06-in

# The encoder output counts to the axis. Only the selected axis will move.
net encoder-counts  <= encoder.0.counts
net encoder-counts => axis.0.jog-counts
net encoder-counts => axis.1.jog-counts
net encoder-counts => axis.2.jog-counts
```

If the machine is capable of high acceleration to smooth out the moves for each click of the MPG use the ilowpass component to limit the acceleration.

**jog.hal with ilowpass**

```
loadrt encoder num_chan=1
loadrt mux4 count=1
addf encoder.capture-position servo-thread
addf encoder.update-counters base-thread
addf mux4.0 servo-thread

loadrt ilowpass
addf ilowpass.0 servo-thread

setp ilowpass.0.scale 1000
setp ilowpass.0.gain 0.01


# If your MPG outputs a quadrature signal per click set x4 to 1
# If your MPG puts out 1 pulse per click set x4 to 0
setp encoder.0.x4-mode 0

# For velocity mode, set to 1
# In velocity mode the axis stops when the dial is stopped
# even if that means the commanded motion is not completed,
# For position mode (the default), set to 0
# In position mode the axis will move exactly jog-scale
# units for each count, regardless of how long that might take,
setp axis.0.jog-vel-mode 0
setp axis.1.jog-vel-mode 0
setp axis.2.jog-vel-mode 0

# The inputs to the mux4 component
net scale1 mux4.0.sel0 <= parport.1.pin-09-in
net scale2 mux4.0.sel1 <= parport.1.pin-10-in

# This sets the scale that will be used based on the input to the mux4
# The scale used here has to be multiplied by the ilowpass scale
setp mux4.0.in0 0.0001
setp mux4.0.in1 0.00001
setp mux4.0.in2 0.000001

# The output from encoder counts is sent to ilowpass
net mpg-out ilowpass.0.in <= encoder.0.counts

# The output from the mux4 is sent to each axis jog scale
net mpg-scale <= mux4.0.out
net mpg-scale => axis.0.jog-scale
net mpg-scale => axis.1.jog-scale
net mpg-scale => axis.2.jog-scale

# The output from the ilowpass is sent to each axis jog count
# Only the selected axis will move.
net encoder-counts  <= ilowpass.0.out
```

```
net encoder-counts => axis.0.jog-counts
net encoder-counts => axis.1.jog-counts
net encoder-counts => axis.2.jog-counts
```

## 10.4 GS2 Spindle

This example shows the connections needed to use an Automation Direct GS2 VFD to drive a spindle. The spindle speed and direction is controlled by LinuxCNC.

Using the GS2 component involves very little to set up. We start with a Stepconf Wizard generated config. Make sure the pins with "Spindle CW" and "Spindle PWM" are set to unused in the parallel port setup screen.

In the custom.hal file we place the following to connect LinuxCNC to the GS2 and have LinuxCNC control the drive.

**GS2 Example**

```
# load the user space component for the Automation Direct GS2 VFD's
loadusr -Wn spindle-vfd gs2_vfd -r 9600 -p none -s 2 -n spindle-vfd

# connect the spindle direction pin to the GS2
net gs2-fwd spindle-vfd.spindle-fwd <= motion.spindle-forward

# connect the spindle on pin to the GS2
net gs2-run spindle-vfd.spindle-on <= motion.spindle-on

# connect the GS2 at speed to the motion at speed
net gs2-at-speed motion.spindle-at-speed <= spindle-vfd.at-speed

# connect the spindle RPM to the GS2
net gs2-RPM spindle-vfd.speed-command <= motion.spindle-speed-out
```

**Note**
The transmission speed might be able to be faster depending on the exact envirnment. Both the drive and the command line options must match. To check for transmission errors add the *-v* command line option and run from a terminal.

On the GS2 drive itself you need to set a couple of things before the modbus communications will work. Other parameters might need to be set based on your physical requirements but these are beyond the scope of this manual. Refer to the GS2 manual that came with the drive for more information on the drive parameters.

- The communications switches must be set to RS-232C
- The motor parameters must be set to match the motor
- P3.00 (Source of Operation Command) must be set to Operation determined by RS-485 interface, 03 or 04
- P4.00 (Source of Frequency Command) must be set to Frequency determined by RS232C/RS485 communication interface, 05
- P9.01 (Transmission Speed) must be set to 9600 baud, 01
- P9.02 (Communication Protocol) must be set to "Modbus RTU mode, 8 data bits, no parity, 2 stop bits", 03

A PyVCP panel based on this example is here.

# Chapter 11

# PLC

## 11.1 Classicladder Introduction

### 11.1.1 History

Classic Ladder is a free implementation of a ladder interpreter, released under the LGPL. It was written by Marc Le Douarain.

He describes the beginning of the project on his website:

> I decided to program a ladder language only for test purposes at the start, in February 2001. It was planned, that I would have to participate to a new product after leaving the enterprise in which I was working at that time. And I was thinking that to have a ladder language in those products could be a nice option to considerate. And so I started to code the first lines for calculating a rung with minimal elements and displaying dynamically it under Gtk, to see if my first idea to realize all this works.
>
> And as quickly I've found that it advanced quite well, I've continued with more complex elements: timer, multiples rungs, etc…
>
> Voila, here is this work… and more: I've continued to add features since then.
>
> — Marc Le Douarain *from "Genesis" at the Classic Ladder website*

Classic Ladder has been adapted to work with LinuxCNC's HAL, and is currently being distributed along with LinuxCNC. If there are issues/problems/bugs please report them to the Enhanced Machine Controller project.

### 11.1.2 Introduction

Ladder logic or the Ladder programming language is a method of drawing electrical logic schematics. It is now a graphical language very popular for programming Programmable Logic Controllers (PLCs). It was originally invented to describe logic made from relays. The name is based on the observation that programs in this language resemble ladders, with two vertical *rails* and a series of horizontal *rungs* between them. In Germany and elsewhere in Europe, the style is to draw the rails horizontally along the top and bottom of the page while the rungs are drawn vertically from left to right.

A program in ladder logic, also called a ladder diagram, is similar to a schematic for a set of relay circuits. Ladder logic is useful because a wide variety of engineers and technicians can understand and use it without much additional training because of the resemblance.

Ladder logic is widely used to program PLCs, where sequential control of a process or manufacturing operation is required. Ladder logic is useful for simple but critical control systems, or for reworking old hardwired relay circuits. As programmable logic controllers became more sophisticated it has also been used in very complex automation systems.

Ladder logic can be thought of as a rule-based language, rather than a procedural language. A *rung* in the ladder represents a rule. When implemented with relays and other electromechanical devices, the various rules *execute* simultaneously and immediately.

When implemented in a programmable logic controller, the rules are typically executed sequentially by software, in a loop. By executing the loop fast enough, typically many times per second, the effect of simultaneous and immediate execution is obtained.

Ladder logic follows these general steps for operation.

- Read Inputs
- Solve Logic
- Update Outputs

### 11.1.3 Example

The most common components of ladder are contacts (inputs), these usually are either NC (normally closed) or NO (normally open), and coils (outputs).

- the NO contact
- the NC contact
- the coil (output)

Of course there are many more components to a full ladder language, but understanding these will help you grasp the overall concept.

The ladder consists of one or more rungs. These rungs are horizontal traces (representing wires), with components on them (inputs, outputs and other), which get evaluated left to right.

This example is the simplest rung:

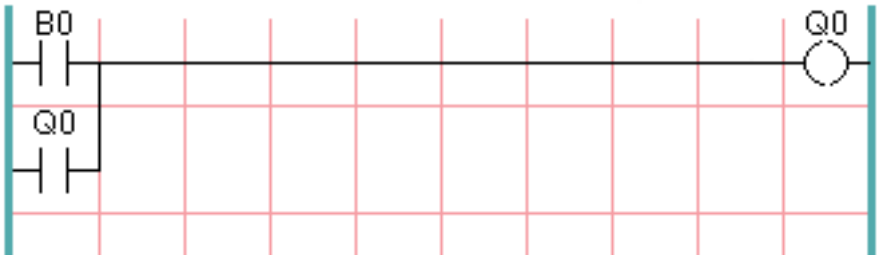
The input on the left, B0, a normally open contact, is connected to the coil (output) on the right, Q0. Now imagine a voltage gets applied to the leftmost end, because the input B0 turns true (e.g. the input is activated, or the user pushed the NO contact). The voltage has a direct path to reach the coil (output) on the right, Q0. As a consequence, the Q0 coil (output) will turn from 0/off/false to 1/on/true. If the user releases B0, the Q0 output quickly returns to 0/off/false.

### 11.1.4 Basic Latching On-Off Circuit

Building on the above example, suppose we add a switch that closes whenever the coil Q0 is active. This would be the case in a relay, where the coil can activate the switch contacts; or in a contactor, where there are often several small auxilliary contacts in addition to the large 3-phase contacts that are the primary feature of the contactor.

Since this auxilliary switch is driven from coil Q0 in our earlier example, we will give it the same number as the coil that drives it. This is the standard practice followed in all ladder programming, although it may seem strange at first to see a switch labeled the same as a coil. So let's call this auxilliary contact Q0 and connect it across the B0 *pushbutton* contact from our earlier example.

Let's take a look at it:

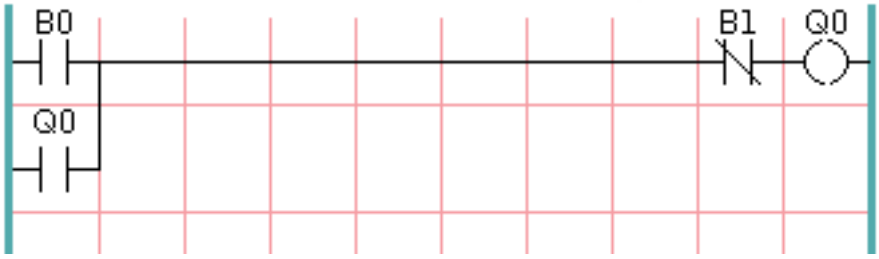



As before, when the user presses pushbutton B0, coil Q0 comes on. And when coil Q0 comes on, switch Q0 comes on. Now the interesting part happens. When the user releases pushbutton B0, coil Q0 does not stop as it did before. This is because switch Q0 of this circuit is effectively holding the user's pushbutton pressed. So we see that switch Q0 is still holding coil Q0 on after the *start* pushbutton has been released.

This type of contact on a coil or relay, used in this way, is often called a *holding contact*, because it *holds on* the coil that it is associated with. It is also occasionally called a *seal* contact, and when it is active it is said that the circuit is *sealed*.

Unfortunately, our circuit so far has little practical use, because, although we have an *on* or *start* button in the form of pushbutton B0, we have no way to shut this circuit off once it is started. But that's easy to fix. All we need is a way to interrupt the power to coil Q0. So let's add a normally-closed (NC) pushbutton just ahead of coil Q0.

Here's how that would look:




Now we have added *off* or *stop* pushbutton B1. If the user pushes it, contact from the rung to the coil is broken. When coil Q0 loses power, it drops to 0/off/false. When coil Q0 goes off, so does switch Q0, so the *holding contact* is broken, or the circuit is *unsealed*. When the user releases the *stop* pushbutton, contact is restored from the rung to coil Q0, but the rung has gone dead, so the coil doesn't come back on.

This circuit has been used for decades on virtually every machine that has a three-phase motor controlled by a contactor, so it was inevitable that it would be adopted by ladder/PLC programmers. It is also a very safe circuit, in that if *start* and *stop* are both pressed at the same time, the *stop* function always wins.

This is the basic building block of much of ladder programming, so if you are new to it, you would do well to make sure that you understand how this circuit operates.

## 11.2 Classicladder Programming

### 11.2.1 Ladder Concepts

Classic Ladder is a type of programming language originally implemented on industrial PLCs (it's called Ladder Programming). It is based on the concept of relay contacts and coils, and can be used to construct logic checks and functions in a manner that is familiar to many systems integrators. Ladder consists of rungs that may have branches and resembles an electrical circuit. It is important to know how ladder programs are evaluated when running.

It seems natural that each line would be evaluated left to right, then the next line down, etc., but it doesn't work this way in ladder logic. Ladder logic *scans* the ladder rungs 3 times to change the state of the outputs.

- the inputs are read and updated

- the logic is figured out
- the outputs are set

This can be confusing at first if the output of one line is read by the input of a another rung. There will be one scan before the second input becomes true after the output is set.

Another gotcha with ladder programming is the "Last One Wins" rule. If you have the same output in different locations of your ladder the state of the last one will be what the output is set to.

### 11.2.2 Languages

The most common language used when working with Classic Ladder is *ladder*. Classic Ladder also supports Sequential Function Chart (Grafcet).

### 11.2.3 Components

There are 2 components to Classic Ladder.

- The real time module classicladder_rt
- The user space module (including a GUI) classicladder

#### 11.2.3.1 Files

Typically classic ladder components are placed in the custom.hal file if your working from a Stepconf generated configuration. These must not be placed in the custom_postgui.hal file or the Ladder Editor menu will be grayed out.

**Note**
Ladder files (.clp) must not contain any blank spaces in the name.

#### 11.2.3.2 Realtime Module

Loading the Classic Ladder real time module (classicladder_rt) is possible from a HAL file, or directly using a halcmd instruction. The first line loads real time the Classic Ladder module. The second line adds the function classicladder.0.refresh to the servo thread. This line makes Classic Ladder update at the servo thread rate.

```
loadrt classicladder_rt
addf classicladder.0.refresh servo-thread
```

The speed of the thread that Classic Ladder is running in directly affects the responsiveness to inputs and outputs. If you can turn a switch on and off faster than Classic Ladder can notice it then you may need to speed up the thread. The fastest that Classic Ladder can update the rungs is one millisecond. You can put it in a faster thread but it will not update any faster. If you put it in a slower than one millisecond thread then Classic Ladder will update the rungs slower. The current scan time will be displayed on the section display, it is rounded to microseconds. If the scan time is longer than one millisecond you may want to shorten the ladder or put it in a slower thread.

#### 11.2.3.3 Variables

It is possible to configure the number of each type of ladder object while loading the Classic Ladder real time module. If you do not configure the number of ladder objects Classic Ladder will use the default values.

Table 11.1: Default Variable Count

| Object Name | Variable Name | Default Value |
|---|---|---|
| Number of rungs | (numRungs) | 100 |
| Number of bits | (numBits) | 20 |
| Number of word variables | (numWords) | 20 |
| Number of timers | (numTimers) | 10 |
| Number of timers IEC | (numTimersIec) | 10 |
| Number of monostables | (numMonostables) | 10 |
| Number of counters | (numCounters) | 10 |
| Number of HAL inputs bit pins | (numPhysInputs) | 15 |
| Number of HAL output bit pins | (numPhysOutputs) | 15 |
| Number of arithmetic expressions | (numArithmExpr) | 50 |
| Number of Sections | (numSections) | 10 |
| Number of Symbols | (numSymbols) | Auto |
| Number of S32 inputs | (numS32in) | 10 |
| Number of S32 outputs | (numS32out) | 10 |
| Number of Float inputs | (numFloatIn) | 10 |
| Number of Float outputs | (numFloatOut) | 10 |

Objects of most interest are numPhysInputs, numPhysOutputs, numS32in, and numS32out.

Changing these numbers will change the number of HAL bit pins available. numPhysInputs and numPhysOutputs control how many HAL bit (on/off) pins are available. numS32in and numS32out control how many HAL signed integers (+- integer range) pins are available.

For example (you don't need all of these to change just a few):

```
loadrt classicladder_rt numRungs=12 numBits=100 numWords=10
numTimers=10 numMonostables=10 numCounters=10 numPhysInputs=10
numPhysOutputs=10 numArithmExpr=100 numSections=4 numSymbols=200
numS32in=5 numS32out=5
```

To load the default number of objects:

```
loadrt classicladder_rt
```

### 11.2.4 Loading the Classic Ladder user module

Classic Ladder HAL commands must executed before the GUI loads or the menu item Ladder Editor will not function. If you used the Stepper Config Wizard place any Classic Ladder HAL commands in the custom.hal file.

To load the user module:

```
loadusr classicladder
```

**Note**
Only one .clp file can be loaded. If you need to divide your ladder use Sections.

To load a ladder file:

```
loadusr classicladder myladder.clp
```

Classic Ladder Loading Options

- *--nogui* - (loads without the ladder editor) normally used after debugging is finished.
- *--modbus_port=port* - (loads the modbus port number)
- *--modmaster* - (initializes MODBUS master) should load the ladder program at the same time or the TCP is default port.
- *--modslave* - (initializes MODBUS slave) only TCP

To use Classic Ladder with HAL without EMC:

```
loadusr -w classicladder
```

The -w tells HAL not to close down the HAL environment until Classic Ladder is finished.

If you first load ladder program with the *--nogui* option then load Classic Ladder again with no options the GUI will display the last loaded ladder program.

In AXIS you can load the GUI from File/Ladder Editor...

### 11.2.5 Classic Ladder GUI

If you load Classic Ladder with the GUI it will display two windows: section display, and section manager.

#### 11.2.5.1 Sections Manager

When you first start up Classic Ladder you get an empty Sections Manager window.

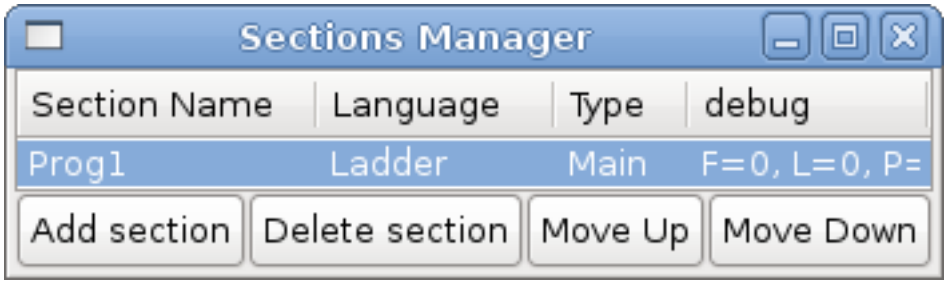



Figure 11.1: Sections Manager Default Window

This window allows you to name, create or delete sections and choose what language that section uses. This is also how you name a subroutine for call coils.

#### 11.2.5.2 Section Display

When you first start up Classic Ladder you get an empty Section Display window. Displayed is one empty rung.

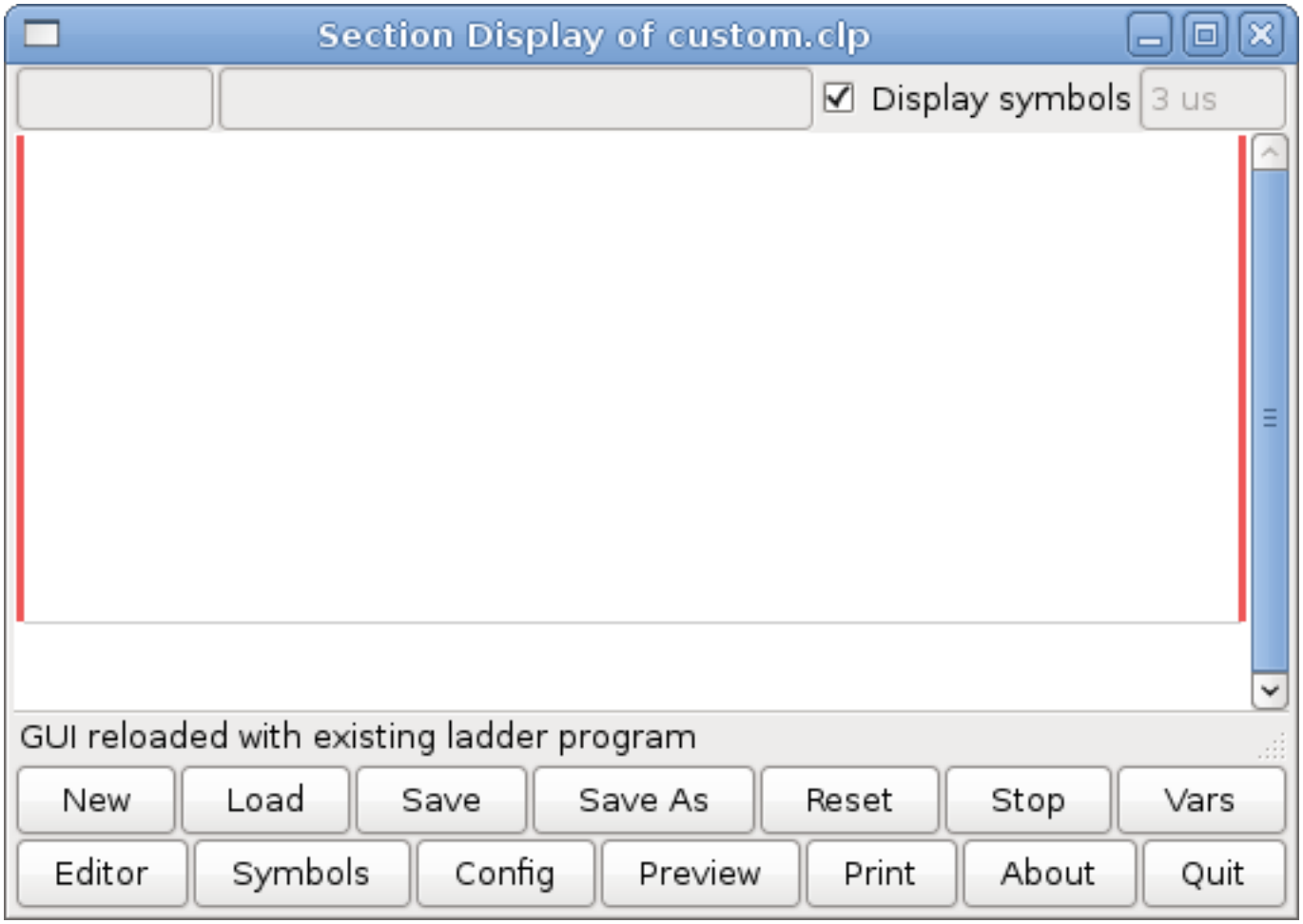



Figure 11.2: Section Display Default Window

Most of the buttons are self explanatory:

The Vars button is for looking at variables, toggle it to display one, the other, both, then none of the windows.

The Config button is used for modbus and shows the max number of ladder elements that was loaded with the real time module.

The Symbols button will display an editable list of symbols for the variables (hint you can name the inputs, outputs, coils etc).

The Quit button will shut down the user program meaning Modbus and the display. The real time ladder program will still run in the background.

The check box at the top right allows you to select whether variable names or symbol names are displayed

You might notice that there is a line under the ladder program display that reads "Project failed to load…" That is the status bar that gives you info about elements of the ladder program that you click on in the display window. This status line will now display HAL signal names for variables %I, %Q and the first %W (in an equation) You might see some funny labels, such as (103) in the rungs. This is displayed (on purpose) because of an old bug- when erasing elements older versions sometimes didn't erase the object with the right code. You might have noticed that the long horizontal connection button sometimes didn't work in the older versions. This was because it looked for the *free* code but found something else. The number in the brackets is the unrecognized code. The ladder program will still work properly, to fix it erase the codes with the editor and save the program.

#### 11.2.5.3 The Variable Windows

This are two variable windows: the Bit Status Window (boolean) and the Watch Window (signed integer). The Vars button is in the Section Display Window, toggle the Vars button to display one, the other, both, then none of the variable windows.

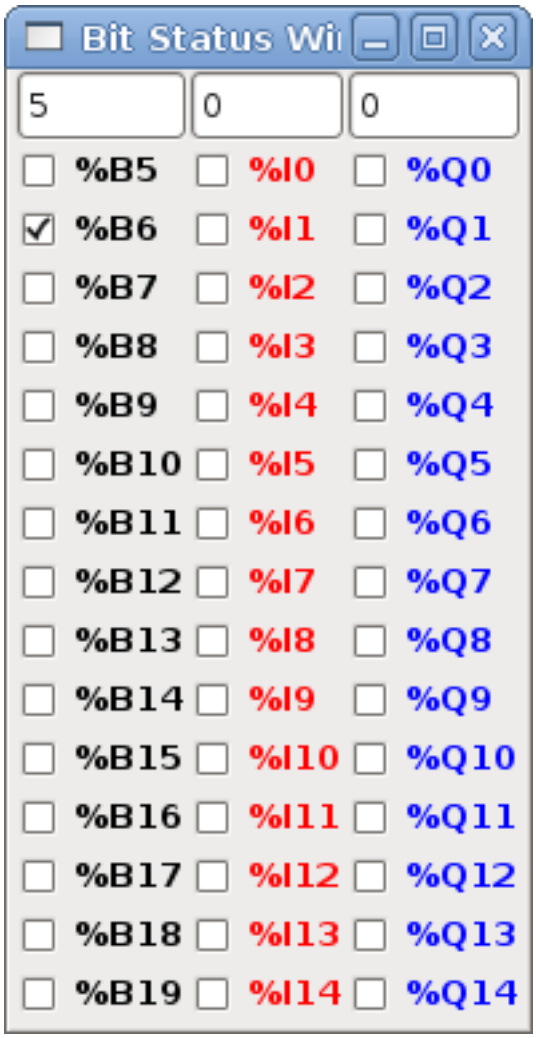



Figure 11.3: Bit Status Window

The Bit Status Window displays some of the boolean (on/off) variable data. Notice all variables start with the % sign. The %I variables represent HAL input bit pins. The %Q represents the relay coil and HAL output bit pins. The %B represents an internal relay coil or internal contact. The three edit areas at the top allow you to select what 15 variables will be displayed in each column. For instance, if the %B Variable column were 15 entries high, and you entered 5 at the top of the column, variables %B5 to %B19 would be displayed. The check boxes allow you to set and unset %B variables manually as long as the ladder program isn't setting them as outputs. Any Bits that are set as outputs by the program when Classic Ladder is running can not be changed and will be displayed as checked if on and unchecked if off.

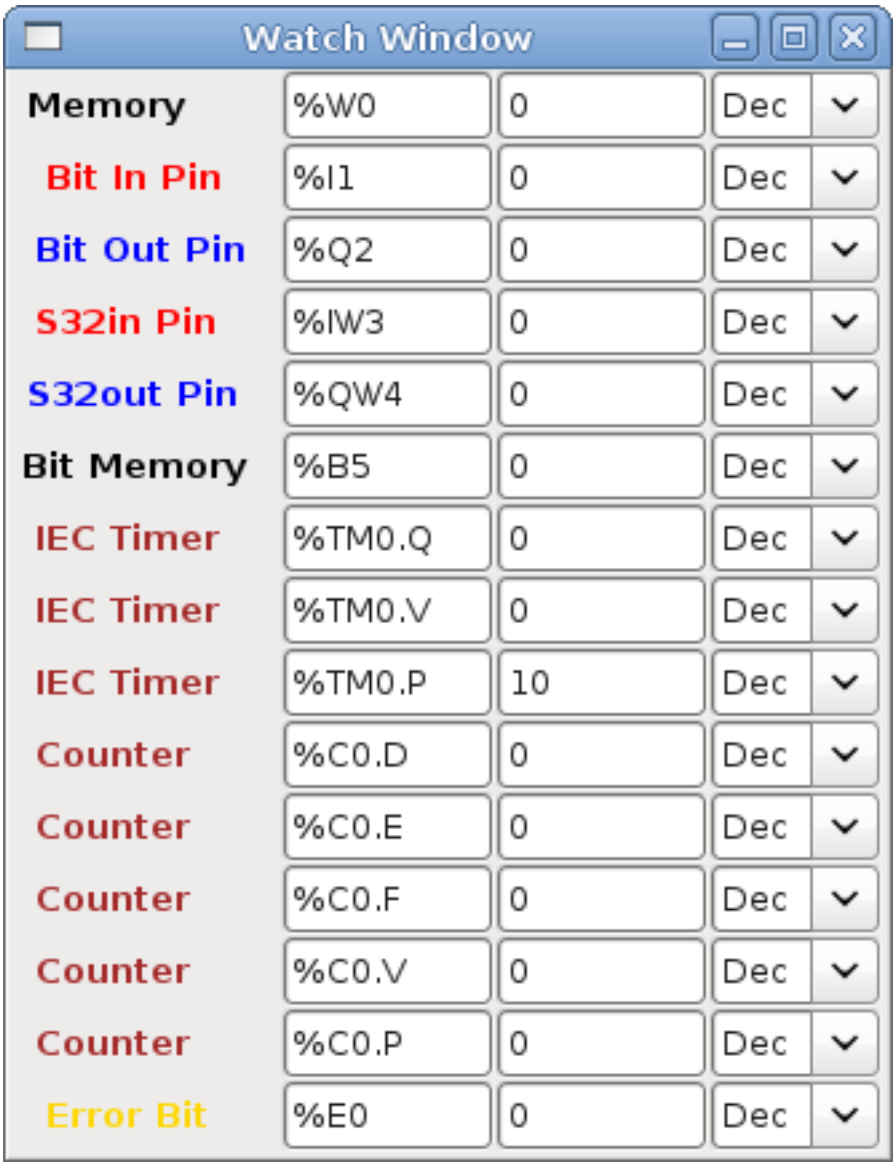



Figure 11.4: Watch Window

The Watch Window displays variable status. The edit box beside it is the number stored in the variable and the drop-down box beside that allow you to choose whether the number to be displayed in hex, decimal or binary. If there are symbol names defined in the symbols window for the word variables showing and the *display symbols* checkbox is checked in the section display window, symbol names will be displayed. To change the variable displayed, type the variable number, e.g. %W2 (if the display symbols check box is not checked) or type the symbol name (if the display symbols checkbox is checked) over an existing variable number/name and press the Enter Key.

#### 11.2.5.4 Symbol Window

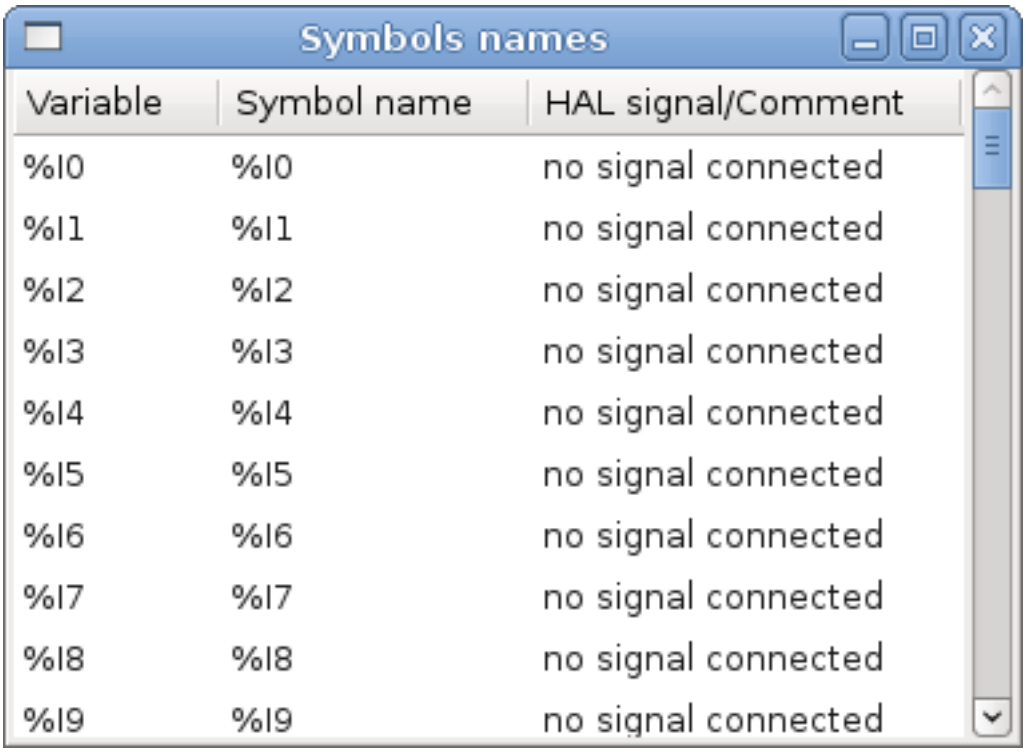



Figure 11.5: Symbol Names window

This is a list of *symbol* names to use instead of variable names to be displayed in the section window when the *display symbols* check box is checked. You add the variable name (remember the % symbol and capital letters), symbol name . If the variable can have a HAL signal connected to it (%I, %Q, and %W-if you have loaded s32 pin with the real time module) then the comment section will show the current HAL signal name or lack thereof. Symbol names should be kept short to display better. Keep in mind that you can display the longer HAL signal names of %I, %Q and %W variable by clicking on them in the section window. Between the two, one should be able to keep track of what the ladder program is connected to!

#### 11.2.5.5 The Editor window

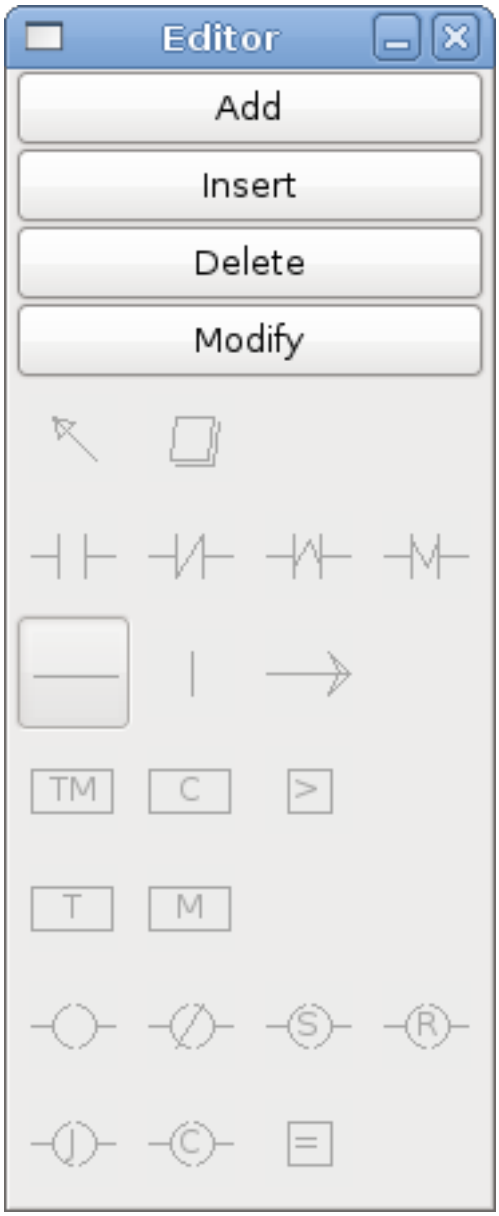



Figure 11.6: Editor Window

- *Add* - adds a rung after the selected rung
- *Insert* - inserts a rung before the selected rung
- *Delete* - deletes the selected rung
- *Modify* - opens the selected rung for editing

Starting from the top left image:

- Object Selector, Eraser
- N.O. Input, N.C. Input, Rising Edge Input , Falling Edge Input

- Horizontal Connection, Vertical Connection , Long Horizontal Connection
- Timer IEC Block, Counter Block, Compare Variable
- Old Timer Block, Old Monostable Block (These have been replaced by the IEC Timer)
- COILS - N.O. Output, N.C. Output, Set Output, Reset Output
- Jump Coil, Call Coil, Variable Assignment

A short description of each of the buttons:

- *Selector* - allows you to select existing objects and modify the information.
- *Eraser* - erases an object.
- *N.O. Contact* - creates a normally open contact. It can be an external HAL-pin (%I) input contact, an internal-bit coil (%B) contact or a external coil (%Q) contact. The HAL-pin input contact is closed when the HAL-pin is true. The coil contacts are closed when the corresponding coil is active (%Q2 contact closes when %Q2 coil is active).
- *N.C. Contact* - creates a normally closed contact. It is the same as the N.O. contact except that the contact is open when the HAL-pin is true or the coil is active.
- 'Rising Edge Contact - creates a contact that is closed when the HAL-pin goes from False to true, or the coil from not-active to active.
- *Falling Edge Contact* - creates a contact that is closed when the HAL-pin goes from true to false or the coil from active to not.
- *Horizontal Connection* - creates a horizontal connection to objects.
- *Vertical Connection* - creates a vertical connection to horizontal lines.
- *Horizontal Running Connection* - creates a horizontal connection between two objects and is a quick way to connect objects that are more than one block apart.
- *IEC Timer* - creates a timer and replaces the *Timer*.
- *Timer* - creates a Timer Module (depreciated use IEC Timer instead).
- *Monostable* - creates a one-shot monostable module
- *Counter* - creates a counter module.
- *Compare* - creates a compare block to compare variable to values or other variables. (eg %W1<=5 or %W1=%W2) Compare cannot be placed in the right most side of the section display.
- *Variable Assignment* - creates an assignment block so you to assign values to variables. (eg %W2=7 or %W1=%W2) ASSIGNMENT functions can only be placed at the right most side of the section display.

#### 11.2.5.6 Config Window

The config window shows the current project status and has the Modbus setup tabs.

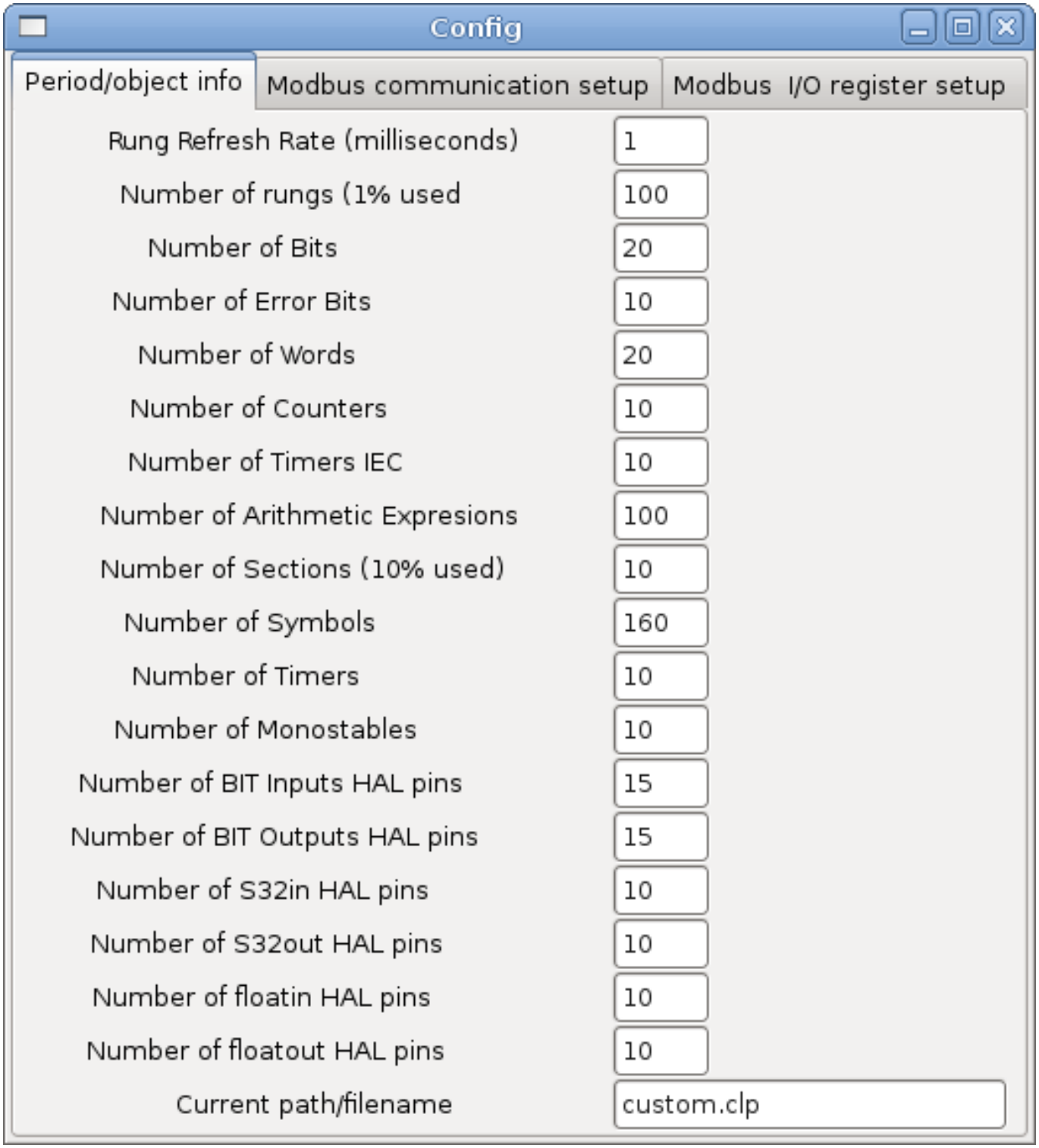



Figure 11.7: Config Window

### 11.2.6 Ladder objects

#### 11.2.6.1 CONTACTS

Represent switches or relay contacts. They are controlled by the variable letter and number assigned to them.

The variable letter can be B, I, or Q and the number can be up to a three digit number eg. %I2, %Q3, or %B123. Variable I is controlled by a HAL input pin with a corresponding number. Variable B is for internal contacts, controlled by a B coil with a corresponding number. Variable Q is controlled by a Q coil with a corresponding number. (like a relay with multiple contacts). E.g. if HAL pin classicladder.0.in-00 is true then %I0 N.O. contact would be on (closed, true, whatever you like to call it). If %B7 coil is *energized* (on, true, etc) then %B7 N.O. contact would be on. If %Q1 coil is *energized* then %Q1 N.O. contact would be on (and HAL pin classicladder.0.out-01 would be true.)

- *N.O. Contact* - (Normally Open) When the variable is false the switch is off.
- *N.C. Contact* - (Normally Closed) When the variable is false the switch is on.
- *Rising Edge Contact* - When the variable changes from false to true, the switch is PULSED on.
- *Falling Edge Contact* - When the variable changes from true to false, the switch is PULSED on.

#### 11.2.6.2 IEC TIMERS

Represent new count down timers. IEC Timers replace Timers and Monostables.

IEC Timers have 2 contacts.

- *I* - input contact
- *Q* - output contact

There are three modes - TON, TOF, TP.

- *TON* - When timer input is true countdown begins and continues as long as input remains true. After countdown is done and as long as timer input is still true the output will be true.
- *TOF* - When timer input is true, sets output true. When the input is false the timer counts down then sets output false.
- *TP* - When timer input is pulsed true or held true timer sets output true till timer counts down. (one-shot)

The time intervals can be set in multiples of 100ms, seconds, or minutes.

There are also Variables for IEC timers that can be read and/or written to in compare or operate blocks.

- *%TMxxx.Q* - timer done (Boolean, read write)
- *%TMxxx.P* - timer preset (read write)
- *%TMxxx.V* - timer value (read write)

#### 11.2.6.3 TIMERS

Represent count down timers. This is deprecated and replaced by IEC Timers.

Timers have 4 contacts.

- *E* - enable (input) starts timer when true, resets when goes false
- *C* - control (input) must be on for the timer to run (usually connect to E)
- *D* - done (output) true when timer times out and as long as E remains true
- *R* - running (output) true when timer is running

The timer base can be multiples of milliseconds, seconds, or minutes.

There are also Variables for timers that can be read and/or written to in compare or operate blocks.

- *%Txx.R* - Timer xx running (Boolean, read only)
- *%Txx.D* - Timer xx done (Boolean, read only)
- *%Txx.V* - Timer xx current value (integer, read only)
- *%Txx.P* - Timer xx preset (integer, read or write)

#### 11.2.6.4 MONOSTABLES

Represent the original one-shot timers. This is now deprecated and replaced by IEC Timers.

Monostables have 2 contacts, I and R.

- *I* - input (input) will start the mono timer running.
- *R* - running (output) will be true while timer is running.

The I contact is rising edge sensitive meaning it starts the timer only when changing from false to true (or off to on). While the timer is running the I contact can change with no effect to the running timer. R will be true and stay true till the timer finishes counting to zero. The timer base can be multiples of milliseconds, seconds, or minutes.

There are also Variables for monostables that can be read and/or written to in compare or operate blocks.

- *%Mxx.R* - Monostable xx running (Boolean, read only)
- *%Mxx.V* - Monostable xx current value (integer, read only)
- *%Mxx.P* - Monostable xx preset (integer, read or write)

#### 11.2.6.5 COUNTERS

Represent up/down counters.

There are 7 contacts:

- *R* - reset (input) will reset the count to 0.
- *P* - preset (input) will set the count to the preset number assigned from the edit menu.
- *U* - up count (input) will add one to the count.
- *D* - down count (input) will subtract one from the count.
- *E* - under flow (output) will be true when the count rolls over from 0 to 9999.
- *D* - done (output) will be true when the count equals the preset.
- *F* - overflow (output) will be true when the count rolls over from 9999 to 0.

The up and down count contacts are edge sensitive meaning they only count when the contact changes from false to true (or off to on if you prefer).

The range is 0 to 9999.

There are also Variables for counters that can be read and/or written to in compare or operate blocks.

- *%Cxx.D* - Counter xx done (Boolean, read only)
- *%Cxx.E* - Counter xx empty overflow (Boolean, read only)
- *%Cxx.F* - Counter xx full overflow (Boolean, read only)
- *%Cxx.V* - Counter xx current value (integer, read or write)
- *%Cxx.P* - Counter xx preset (integer, read or write)

#### 11.2.6.6 COMPARE

For arithmetic comparison. Is variable %XXX = to this number (or evaluated number)

The compare block will be true when comparison is true. you can use most math symbols:

- +, - ,* , /, = (standard math symbols)
- < (less than), > (greater than), <= (less or equal), >= (greater or equal), <> (not equal)
- (, ) separate into groups example %IF1=2,&%IF2<5 in pseudo code translates to if %IF1 is equal to 2 and %IF2 is less than 5 then the comparison is true. Note the comma seperating the two groups of comparisions.
- ˆ (exponent),% (modulus),& (and),| (or),. -
- ABS (absolute), MOY (French for average) ,AVG (average)

For example ABS(%W2)=1, MOY(%W1,%W2)<3.

No spaces are allowed in the comparison equation. For example %C0.V>%C0.P is a valid comparison expression while %C0.V > %CO.P is not a valid expression.

There is a list of Variables down the page that can be used for reading from and writing to ladder objects. When a new compare block is opened be sure and delete the # symbol when you enter a compare.

To find out if word variable #1 is less than 2 times the current value of counter #0 the syntax would be:

```
%W1<2*%C0.V
```

To find out if S32in bit 2 is equal to 10 the syntax would be:

```
%IW2=10
```

Note: Compare uses the arithmetic equals not the double equals that programmers are used to.

#### 11.2.6.7 VARIABLE ASSIGNMENT

For variable assignment, e.g. assign this number (or evaluated number) to this variable %xxx, there are two math functions MINI and MAXI that check a variable for maximum (0x80000000) and minimum values (0x07FFFFFF) (think signed values) and keeps them from going beyond.

When a new variable assignment block is opened be sure to delete the # symbol when you enter an assignment.

To assign a value of 10 to the timer preset of IEC Timer 0 the syntax would be:

```
%TM0.P=10
```

To assign the value of 12 to s32out bit 3 the syntax would be:

```
%QW3=12
```

**Note**
When you assign a value to a variable with the variable assignment block the value is retained until you assign a new value using the variable assignment block. The last value assigned will be restored when LinuxCNC is started.

The following figure shows an Assignment and a Comparison Example. %QW0 is a S32out bit and %IW0 is a S32in bit. In this case the HAL pin classicladder.0.s32out-00 will be set to a value of 5 and when the HAL pin classicladder.0.s32in-00 is 0 the HAL pin classicladder.0.out-00 will be set to True.

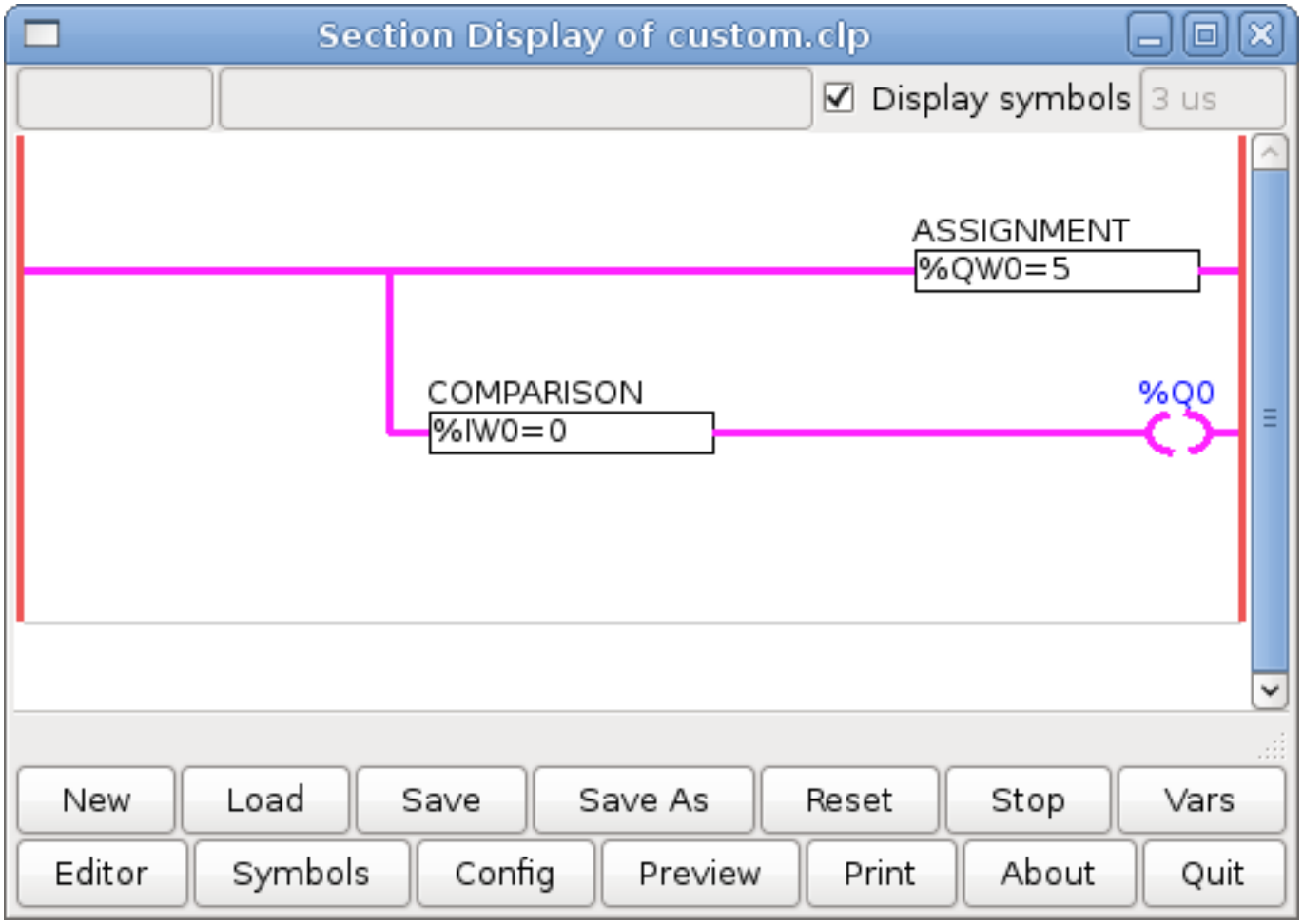



Figure 11.8: Assign/Compare Example

#### 11.2.6.8 COILS

Coils represent relay coils. They are controlled by the variable letter and number assigned to them.

The variable letter can be B or Q and the number can be up to a three digit number eg. %Q3, or %B123. Q coils control HAL out pins, e.g. if %Q15 is energized then HAL pin classicladder.0.out-15 will be true. B coils are internal coils used to control program flow.

- *N.O. COIL* - (a relay coil.) When coil is energized it's N.O. contact will be closed (on, true, etc)
- *N.C. COIL* - (a relay coil that inverses its contacts.) When coil is energized it"s N.O. contact will be open (off, false, etc)
- *SET COIL* - (a relay coil with latching contacts) When coil is energized it's N.O. contact will be latched closed.
- *RESET COIL* - (a relay coil with latching contacts) When coil is energized It's N.0. contact will be latched open.
- *JUMP COIL* - (a *goto* coil) when coil is energized ladder program jumps to a rung (in the CURRENT section) -jump points are designated by a rung label. (Add rung labels in the section display, top left label box)
- *CALL COIL* - (a *gosub* coil) when coil is energized program jumps to a subroutine section designated by a subroutine number -subroutines are designated SR0 to SR9 (designate them in the section manager)

**Warning**
If you use a N.C. contact with a N.C. coil the logic will work (when the coil is energized the contact will be closed) but that is really hard to follow!

**JUMP COIL** A JUMP COIL is used to *JUMP* to another section, like a goto in BASIC programming language.

If you look at the top left of the sections display window you will see a small label box and a longer comment box beside it. Now go to Editor→Modify then go back to the little box, type in a name.

Go ahead and add a comment in the comment section. This label name is the name of this rung only and is used by the JUMP COIL to identify where to go.

When placing a JUMP COIL, add it in the rightmost position and change the label to the rung you want to JUMP to.

**CALL COIL** A CALL COIL is used to go to a subroutine section then return, like a gosub in BASIC programming language.

If you go to the sections manager window hit the add section button. You can name this section, select what language it will use (ladder or sequential), and select what type (main or subroutine).

Select a subroutine number (SR0 for example). An empty section will be displayed and you can build your subroutine.

When you've done that, go back to the section manager and click on the your main section (default name prog1).

Now you can add a CALL COIL to your program. CALL COILs are to be placed at the rightmost position in the rung.

Remember to change the label to the subroutine number you chose before.

### 11.2.7 Classic Ladder Variables

These Variables are used in COMPARE or OPERATE to get information about, or change specs of, ladder objects such as changing a counter preset, or seeing if a timer is done running.

List of variables :

- *%Bxxx* - Bit memory xxx (Boolean)
- *%Wxxx* - Word memory xxx (32 bits signed integer)
- *%IWxxx* - Word memory xxx (S32 in pin)
- *%QWxxx* - Word memory xxx (S32 out pin)
- *%IFxx* - Word memory xx (Float in pin) (**converted to S32 in Classic Ladder**)
- *%QFxx* - Word memory xx (Float out pin) (**converted to S32 in Classic Ladder**)
- *%Txx.R* - Timer xx running (Boolean, user read only)
- *%Txx.D* - Timer xx done (Boolean, user read only)
- *%Txx.V* - Timer xx current value (integer, user read only)
- *%Txx.P* - Timer xx preset (integer)
- *%TMxxx.Q* - Timer xxx done (Boolean, read write)
- *%TMxxx.P* - Timer xxx preset (integer, read write)
- *%TMxxx.V* - Timer xxx value (integer, read write)
- *%Mxx.R* - Monostable xx running (Boolean)
- *%Mxx.V* - Monostable xx current value (integer, user read only)
- *%Mxx.P* - Monostable xx preset (integer)
- *%Cxx.D* - Counter xx done (Boolean, user read only)
- *%Cxx.E* - Counter xx empty overflow (Boolean, user read only)
- *%Cxx.F* - Counter xx full overflow (Boolean, user read only)
- *%Cxx.V* - Counter xx current value (integer)
- *%Cxx.P* - Counter xx preset (integer)
- *%Ixxx* - Physical input xxx (Boolean) (HAL input bit)
- *%Qxxx* - Physical output xxx (Boolean) (HAL output bit)
- *%Xxxx* - Activity of step xxx (sequential language)
- *%Xxxx.V* - Time of activity in seconds of step xxx (sequential language)
- *%Exx* - Errors (Boolean, read write(will be overwritten))
- *Indexed or vectored variables* - These are variables indexed by another variable. Some might call this vectored variables. Example: %W0[%W4] => if %W4 equals 23 it corresponds to %W23

### 11.2.8 GRAFCET Programming

**Warning**
This is probably the least used and most poorly understood feature of Classic Ladder. Sequential programming is used to make sure a series of ladder events always happen in a prescribed order. Sequential programs do not work alone. There is always a ladder program as well that controls the variables. Here are the basic rules governing sequential programs:

- Rule 1 : Initial situation - The initial situation is characterized by the initial steps which are by definition in the active state at the beginning of the operation.There shall be at least one initial step.
- Rule 2 : R2, Clearing of a transition - A transition is either enabled or disabled. It is said to be enabled when all immediately preceding steps linked to its corresponding transition symbol are active, otherwise it is disabled. A transition cannot be cleared unless it is enabled, and its associated transition condition is true.
- Rule 3 : R3, Evolution of active steps - The clearing of a transition simultaneously leads to the active state of the immediately following step(s) and to the inactive state of the immediately preceding step(s).
- Rule 4 : R4, Simultaneous clearing of transitions - All simultaneous cleared transitions are simultaneously cleared.
- Rule 5 : R5, Simultaneous activation and deactivation of a step - If during operation, a step is simultaneously activated and deactivated, priority is given to the activation.

This is the SEQUENTIAL editor window Starting from the top left image: Selector arrow , Eraser Ordinary step , Initial (Starting) step Transition , Step and Transition Transition Link-Downside , Transition Link-Upside Pass-through Link-Downside , Pass-through Link-Upside Jump Link Comment Box [show sequential program]

- *ORDINARY STEP* - has a unique number for each one
- *STARTING STEP* - a sequential program must have one. This is where the program will start.
- *TRANSITION* - This shows the variable that must be true for control to pass through to the next step.
- *STEP AND TRANSITION* - Combined for convenience
- *TRANSITION LINK-DOWNSIDE* - splits the logic flow to one of two possible lines based on which of the next steps is true first (Think OR logic)
- *TRANSITION LINK=UPSIDE* - combines two (OR) logic lines back in to one
- *PASS-THROUGH LINK-DOWNSIDE* - splits the logic flow to two lines that BOTH must be true to continue (Think AND logic)
- *PASS-THROUGH LINK-UPSIDE* - combines two concurrent (AND logic) logic lines back together
- *JUMP LINK* - connects steps that are not underneath each other such as connecting the last step to the first
- *COMMENT BOX* - used to add comments

To use links, you must have steps already placed. Select the type of link, then select the two steps or transactions one at a time. It takes practice!

With sequential programming: The variable %Xxxx (eg. %X5) is used to see if a step is active. The variable %Xxxx.V (eg. %X5.V) is used to see how long the step has been active. The %X and %X.v variables are use in LADDER logic. The variables assigned to the transitions (eg. %B) control whether the logic will pass to the next step. After a step has become active the transition variable that caused it to become active has no control of it anymore. The last step has to JUMP LINK back only to the beginning step.

### 11.2.9 Modbus

Things to consider:

- Modbus is a userspace program so it might have latency issues on a heavily laden computer.
- Modbus is not really suited to Hard real time events such as position control of motors or to control E-stop.
- The Classic Ladder GUI must be running for Modbus to be running.
- Modbus is not fully finished so it does not do all modbus functions.

To get MODBUS to initialize you must specify that when loading the Classic Ladder userspace program.

**Loading Modbus**

```
loadusr -w classicladder --modmaster myprogram.clp
```

The -w makes HAL wait until you close Classic Ladder before closing realtime session. Classic Ladder also loads a TCP modbus slave if you add --*modserver* on command line.

MODBUS FUNCTIONS

- *1* - read coils
- *2* - read inputs
- *3* - read holding registers
- *4* - read input registers
- *5* - write single coils
- *6* - write single register
- *8* - echo test
- *15* - write multiple coils
- *16* - write multiple registers

If you do not specify a -- *modmaster* when loading the Classic Ladder user program this page will not be displayed.

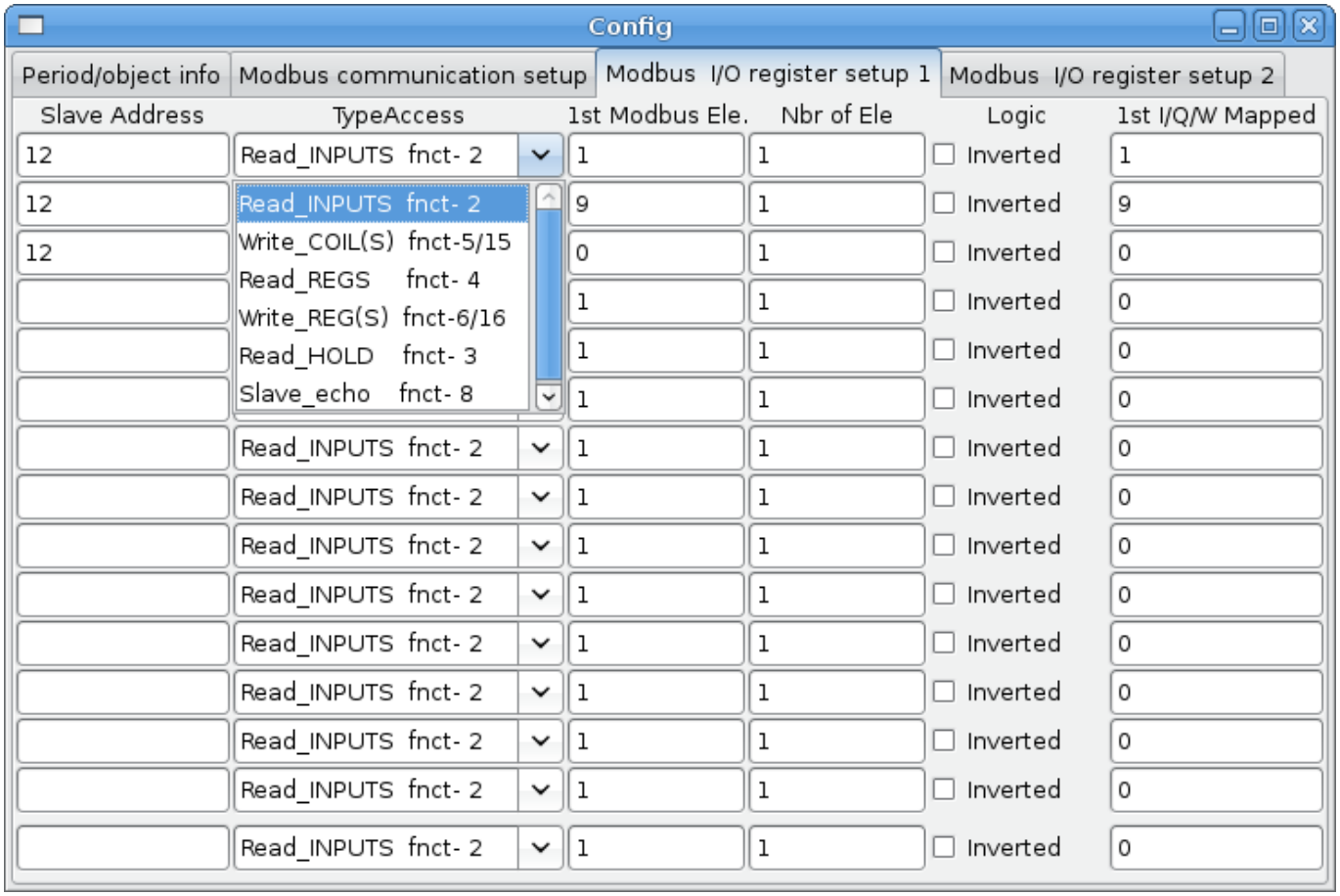



Figure 11.9: Config I/O

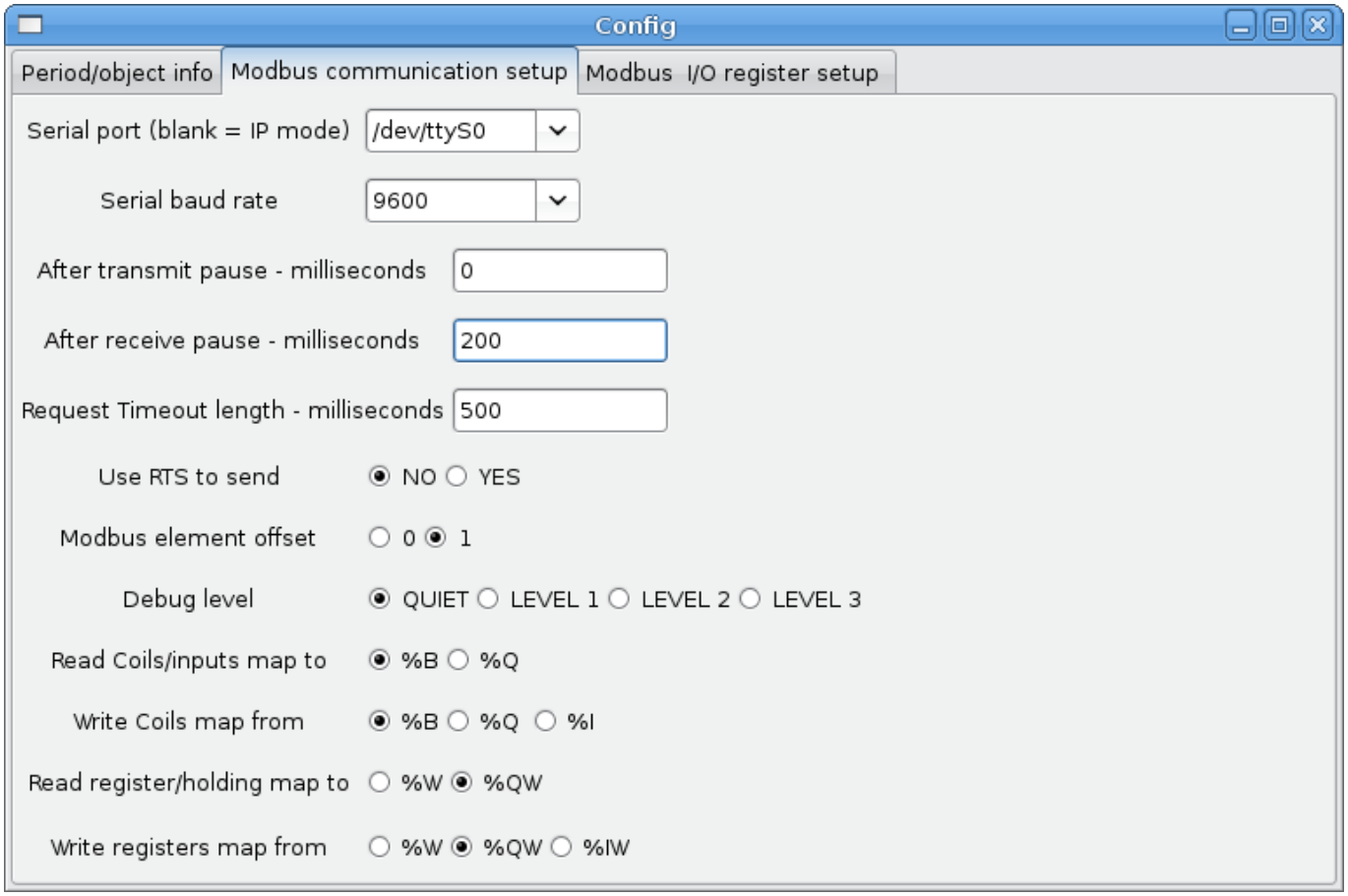



Figure 11.10: Config Coms

- *SERIAL PORT* - For IP blank. For serial the location/name of serial driver eg. /dev/ttyS0 ( or /dev/ttyUSB0 for a USB-to-serial converter).
- *SERIAL SPEED* - Should be set to speed the slave is set for - 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 are supported.
- *PAUSE AFTER TRANSMIT* - Pause (milliseconds) after transmit and before receiving answer, some devices need more time (e.g., USB-to-serial converters).
- *PAUSE INTER-FRAME* - Pause (milliseconds) after receiving answer from slave. This sets the duty cycle of requests (it's a pause for EACH request).
- *REQUEST TIMEOUT LENGTH* - Length (milliseconds) of time before we decide that the slave didn't answer.
- *MODBUS ELEMENT OFFSET* - used to offset the element numbers by 1 (for manufacturers numbering differences).
- *DEBUG LEVEL* - Set this to 0-3 (0 to stop printing debug info besides no-response errors).
- *READ COILS/INPUTS MAP TO* - Select what variables that read coils/inputs will update. (B or Q).
- *WRITE COILS MAP TO* - Select what variables that write coils will updated.from (B,Q,or I).
- *READ REGISTERS/HOLDING* - Select what variables that read registers will update. (W or QW).
- *WRITE REGISTERS MAP TO* - Select what variables that read registers will updated from. (W, QW, or IW).
- *SLAVE ADDRESS* - For serial the slaves ID number usually settable on the slave device (usually 1-256) For IP the slave IP address plus optionally the port number.

- *TYPE ACCESS* - This selects the MODBUS function code to send to the slave (eg what type of request).
- *COILS / INPUTS* - Inputs and Coils (bits) are read from/written to I, B, or Q variables (user selects).
- *REGISTERS (WORDS)* - Registers (Words/Numbers) map to IW, W, or QW variables (user selects).
- *1st MODBUS ELEMENT* - The address (or register number) of the first element in a group. (remember to set MODBUS ELEMENT OFFSET properly).
- *NUMBER OF ELEMENTS* - The number of elements in this group.
- *LOGIC* - You can invert the logic here.
- *1st%I%Q IQ WQ MAPPED* - This is the starting number of %B, %I, %Q, %W, %IW, or %QW variables that are mapped onto/from the modbus element group (starting at the first modbus element number).

In the example above: Port number - for my computer /dev/ttyS0 was my serial port.

The serial speed is set to 9600 baud.

Slave address is set to 12 (on my VFD I can set this from 1-31, meaning I can talk to 31 VFDs maximum on one system).

The first line is set up for 8 input bits starting at the first register number (register 1). So register numbers 1-8 are mapped onto Classic Ladder's %B variables starting at %B1 and ending at %B8.

The second line is set for 2 output bits starting at the ninth register number (register 9) so register numbers 9-10 are mapped onto Classic Ladder's %Q variables starting at %Q9 ending at %Q10.

The third line is set to write 2 registers (16 bits each) starting at the 0th register number (register 0) so register numbers 0-1 are mapped onto Classic Ladder's %W variables starting at %W0 ending at %W1.

It's easy to make an off-by-one error as sometimes the modbus elements are referenced starting at one rather then 0 (actually by the standard that is the way it's supposed to be!) You can use the modbus element offset radio button to help with this.

The documents for your modbus slave device will tell you how the registers are set up- there is no standard way.

The SERIAL PORT, PORT SPEED, PAUSE, and DEBUG level are editable for changes (when you close the config window values are applied, though Radio buttons apply immediately).

To use the echo function select the echo function and add the slave number you wish to test. You don't need to specify any variables.

The number 257 will be sent to the slave number you specified and the slave should send it back. you will need to have Classic Ladder running in a terminal to see the message.

#### 11.2.9.1 MODBUS Settings

Serial:

- Classic Ladder uses RTU protocol (not ASCII).
- 8 data bits, No parity is used, and 1 stop bit is also known as 8-N-1.
- Baud rate must be the same for slave and master. Classic Ladder can only have one baud rate so all the slaves must be set to the same rate.
- Pause inter frame is the time to pause after receiving an answer.
- MODBUS_TIME_AFTER_TRANSMIT is the length of pause after sending a request and before receiving an answer (this apparently helps with USB converters which are slow).

#### 11.2.9.2 MODBUS Info

- Classic Ladder can use distributed inputs/outputs on modules using the modbus protocol ("master": polling slaves).
- The slaves and theirs I/O can be configured in the config window.
- 2 exclusive modes are available : ethernet using Modbus/TCP and serial using Modbus/RTU.
- No parity is used.
- If no port name for serial is set, TCP/IP mode will be used...
- The slave address is the slave address (Modbus/RTU) or the IP address.
- The IP address can be followed per the port number to use (xx.xx.xx.xx:pppp) else the port 9502 will be used per default.
- 2 products have been used for tests: a Modbus/TCP one (Adam-6051, http://www.advantech.com) and a serial Modbus/RTU one (http://www.ipac.ws).
- See examples: adam-6051 and modbus_rtu_serial.
- Web links: http://www.modbus.org and this interesting one: http://www.iatips.com/modbus.html
- MODBUS TCP SERVER INCLUDED
- Classic Ladder has a Modbus/TCP server integrated. Default port is 9502. (the previous standard 502 requires that the application must be launched with root privileges).
- List of Modbus functions code supported are: 1, 2, 3, 4, 5, 6, 15 and 16.
- Modbus bits and words correspondence table is actually not parametric and correspond directly to the %B and %W variables.

More information on modbus protocol is available on the internet.

http://www.modbus.org/

#### 11.2.9.3 Communication Errors

If there is a communication error, a warning window will pop up (if the GUI is running) and %E0 will be true. Modbus will continue to try to communicate. The %E0 could be used to make a decision based on the error. A timer could be used to stop the machine if timed out, etc.

#### 11.2.9.4 MODBUS Bugs

- In compare blocks the function %W=ABS(%W1-%W2) is accepted but does not compute properly. only %W0=ABS(%W1) is currently legal.
- When loading a ladder program it will load Modbus info but will not tell Classic Ladder to initialize Modbus. You must initialize Modbus when you first load the GUI by adding *--modmaster*.
- If the section manager is placed on top of the section display, across the scroll bar and exit is clicked the user program crashes.
- When using *--modmaster* you must load the ladder program at the same time or else only TCP will work.
- reading/writing multiple registers in Modbus has checksum errors.

### 11.2.10 Setting up Classic Ladder

In this section we will cover the steps needed to add Classic Ladder to a Stepconf Wizard generated config. On the advanced Configuration Options page of Stepconf Wizard check off "Include Classic Ladder PLC".

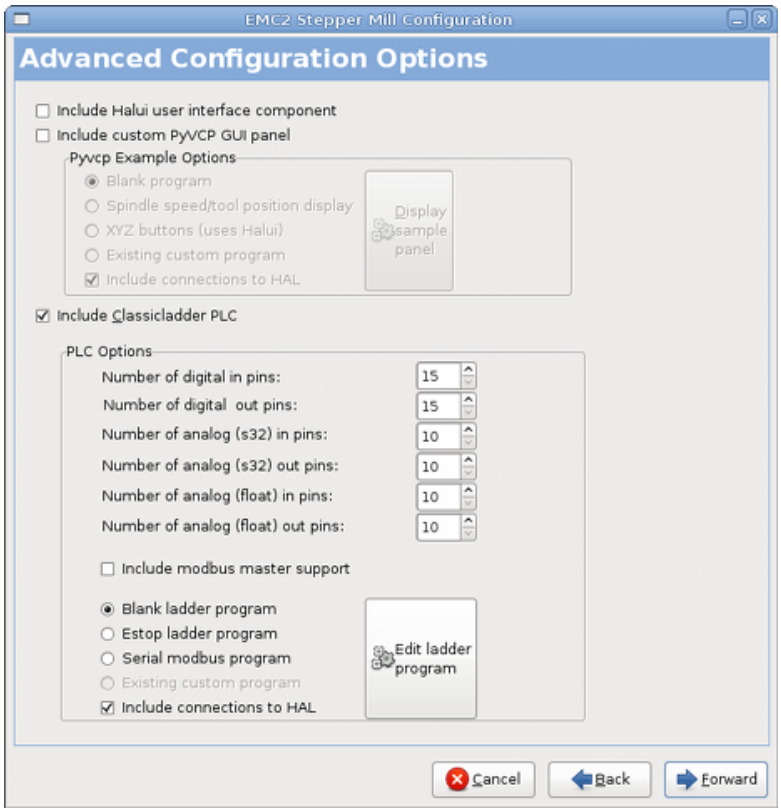



Figure 11.11: Stepconf Classic Ladder

#### 11.2.10.1 Add the Modules

If you used the Stepconf Wizard to add Classic Ladder you can skip this step.

To manually add Classic Ladder you must first add the modules. This is done by adding a couple of lines to the custom.hal file.

This line loads the real time module:

```
loadrt classicladder_rt
```

This line adds the Classic Ladder function to the servo thread:

```
addf classicladder.0.refresh servo-thread
```

#### 11.2.10.2 Adding Ladder Logic

Now start up your config and select "File/Ladder Editor" to open up the Classic Ladder GUI. You should see a blank Section Display and Sections Manager window as shown above. In the Section Display window open the Editor. In the Editor window select Modify. Now a Properties window pops up and the Section Display shows a grid. The grid is one rung of ladder. The rung can contain branches. A simple rung has one input, a connector line and one output. A rung can have up to six horizontal branches. While it is possible to have more than one circuit in a run the results are not predictable.

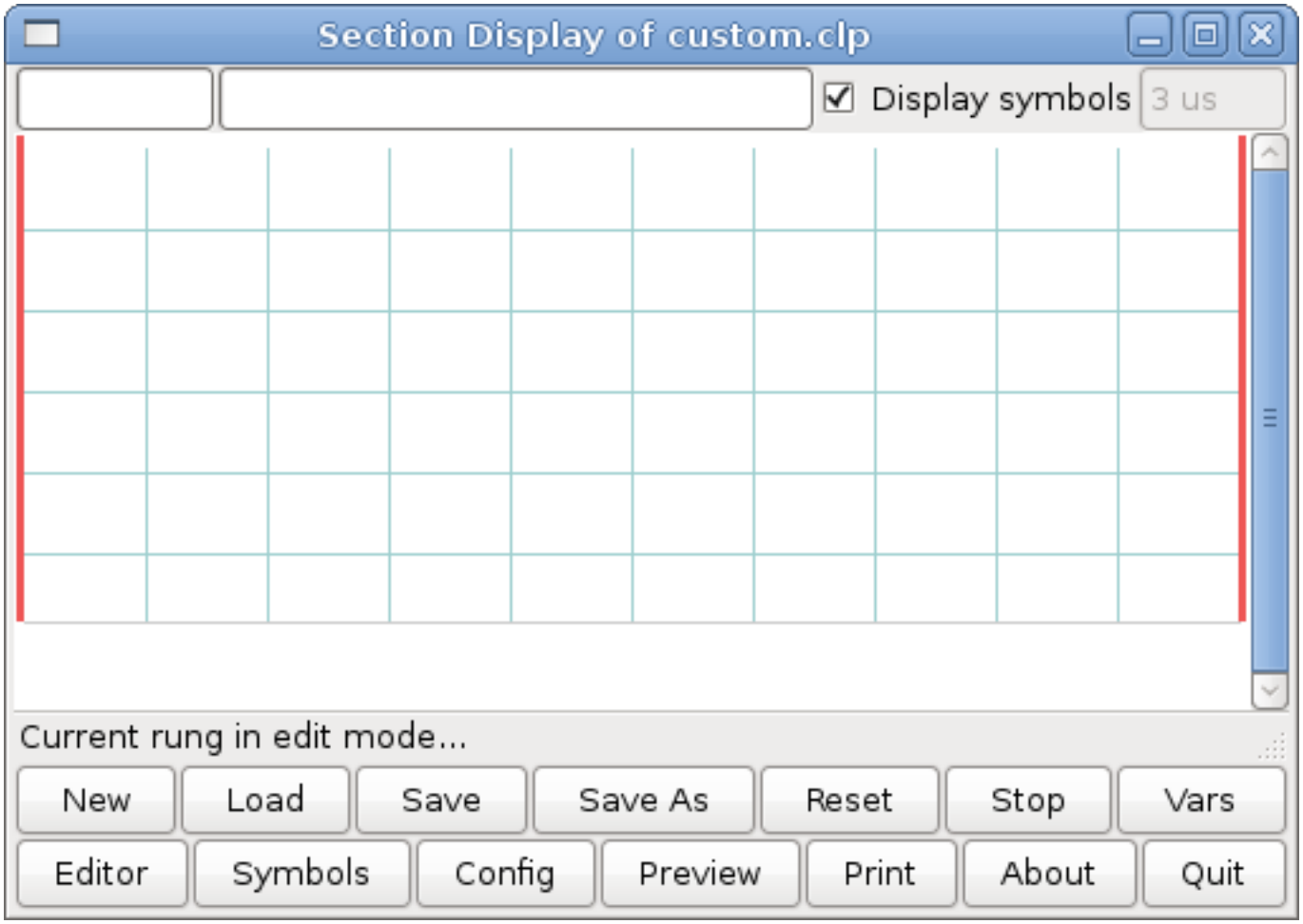



Figure 11.12: Section Display with Grid

Now click on the N.O. Input in the Editor Window.

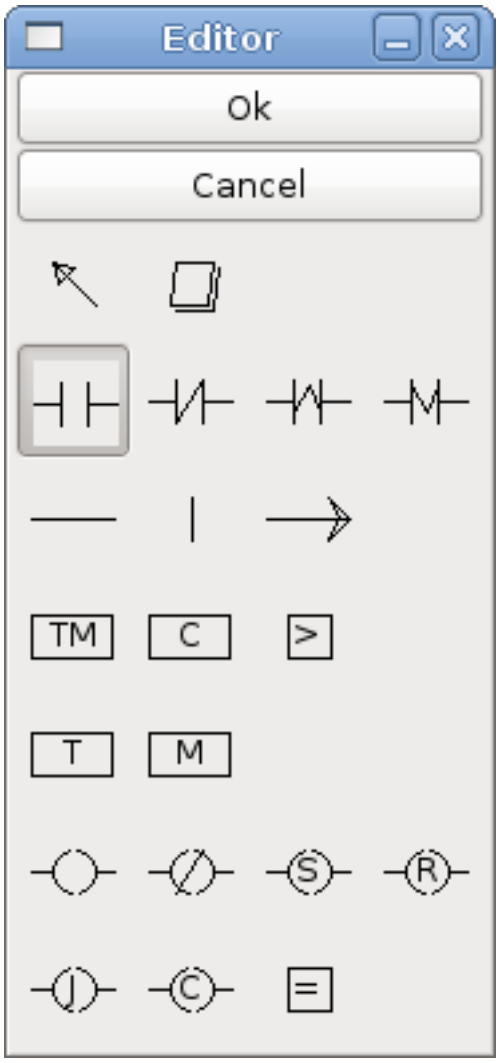



Figure 11.13: Editor Window

Now click in the upper left grid to place the N.O. Input into the ladder.

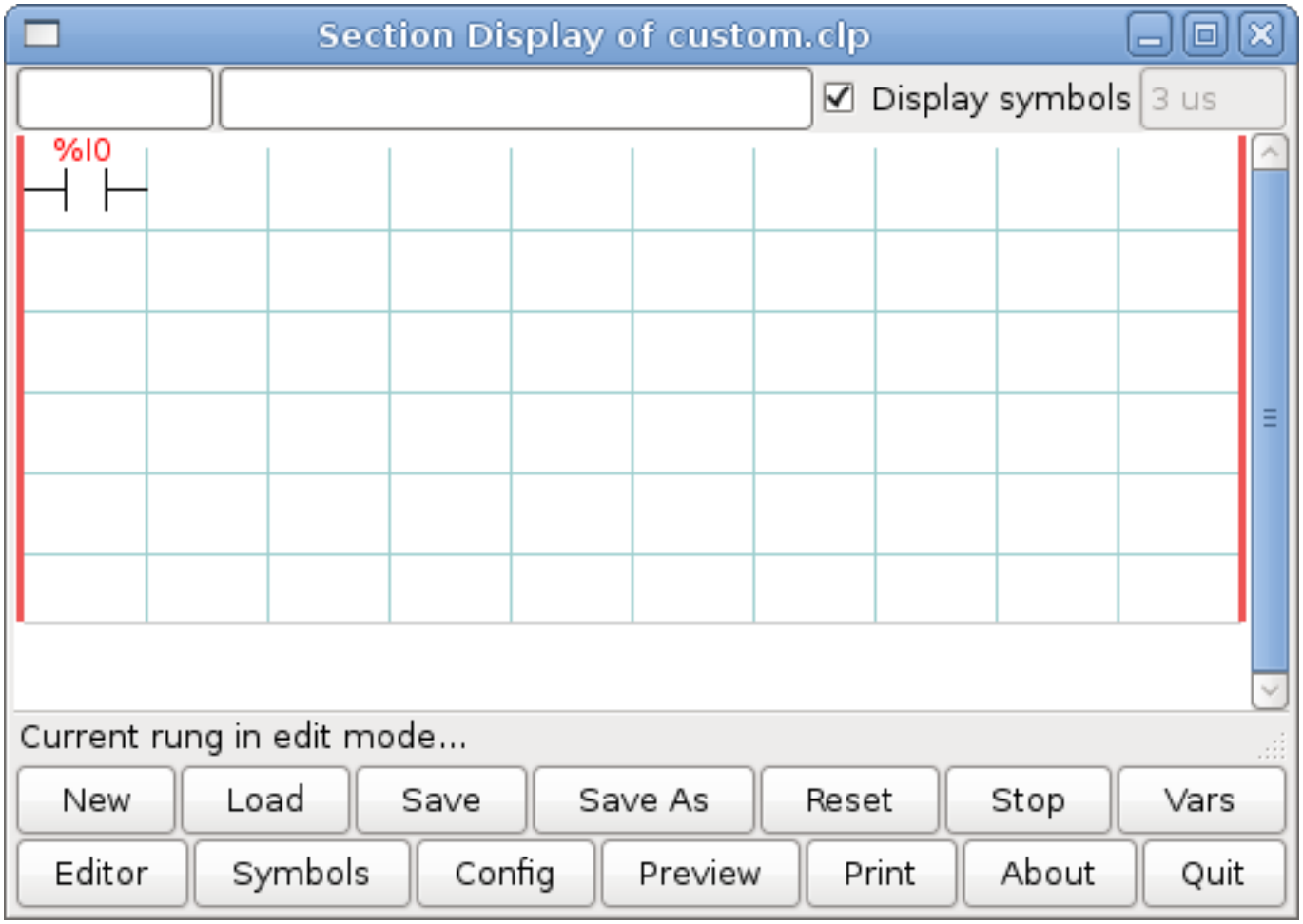



Figure 11.14: Section Display with Input

Repeat the above steps to add a N.O. Output to the upper right grid and use the Horizontal Connection to connect the two. It should look like the following. If not, use the Eraser to remove unwanted sections.

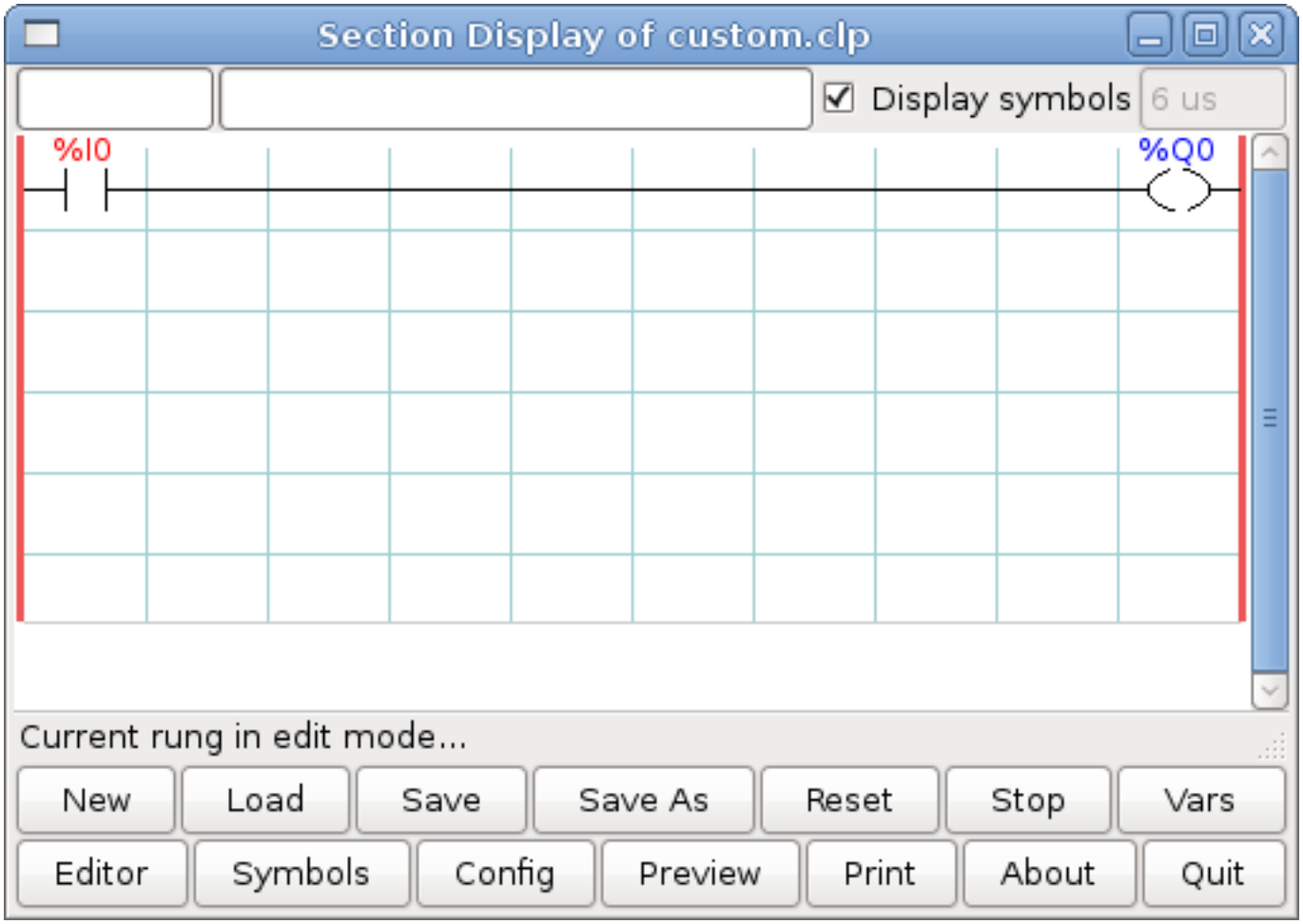



Figure 11.15: Section Display with Rung

Now click on the OK button in the Editor window. Now your Section Display should look like this.

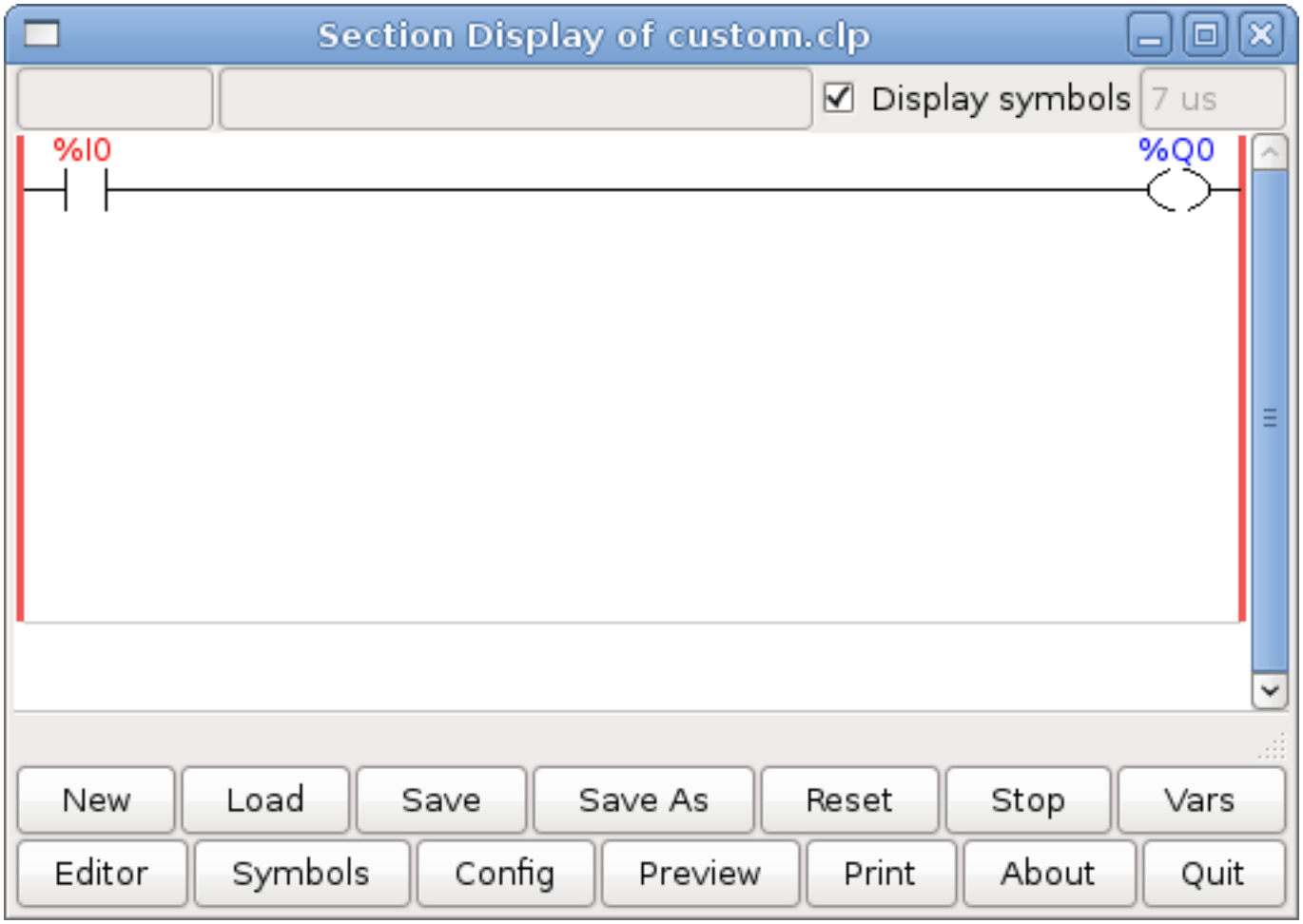



Figure 11.16: Section Display Finished

To save the new file select Save As and give it a name. The .clp extension will be added automatically. It should default to the running config directory as the place to save it.

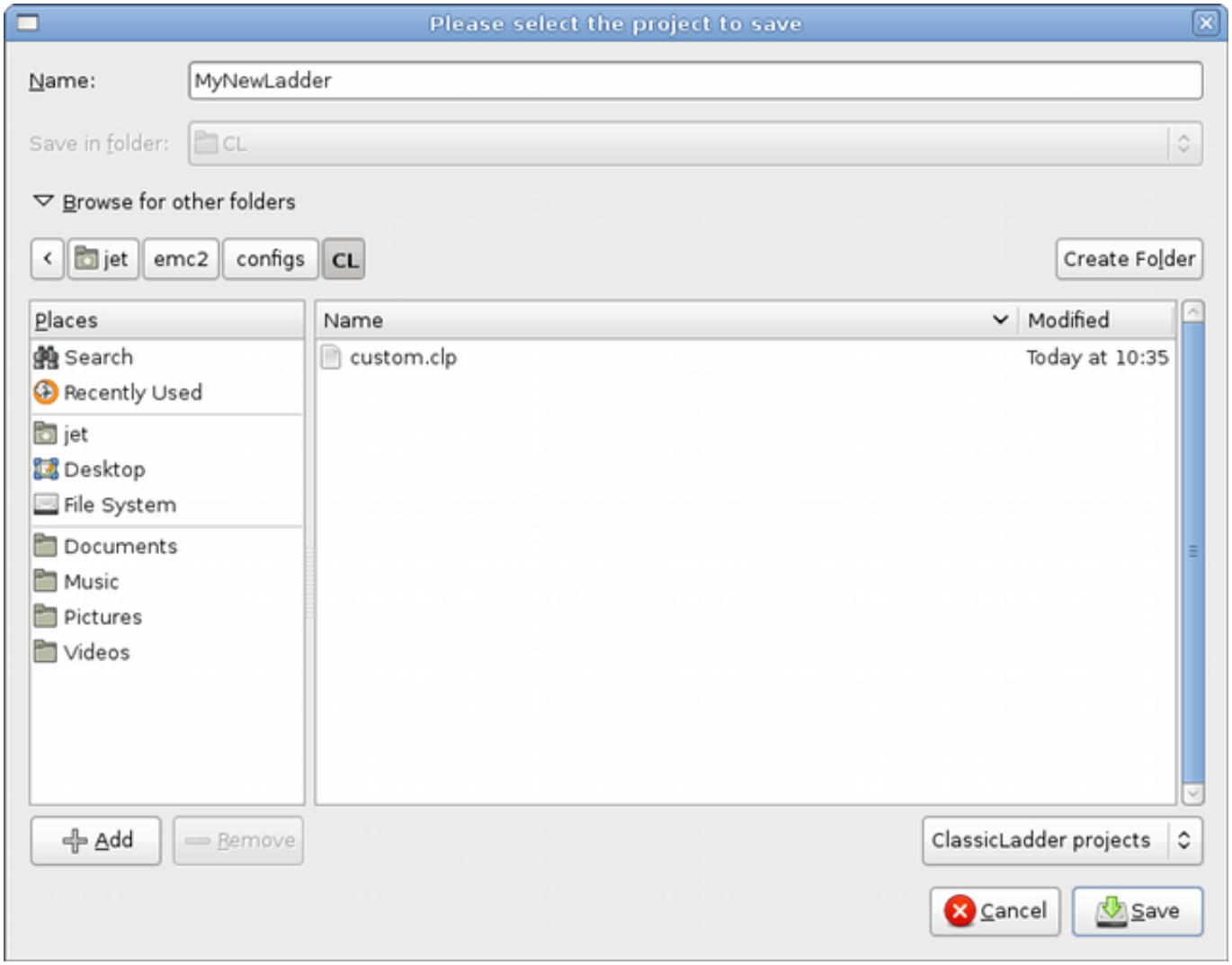



Figure 11.17: Save As Dialog

Again if you used the Stepconf Wizard to add Classic Ladder you can skip this step.

To manually add a ladder you need to add add a line to your custom.hal file that will load your ladder file. Close your LinuxCNC session and add this line to your custom.hal file.

```
loadusr -w classicladder --nogui MyLadder.clp
```

Now if you start up your LinuxCNC config your ladder program will be running as well. If you select "File/Ladder Editor", the program you created will show up in the Section Display window.

## 11.3 Classicladder Examples

### 11.3.1 Wrapping Counter

To have a counter that *wraps around* you have to use the preset pin and the reset pin. When you create the counter set the preset at the number you wish to reach before wrapping around to 0. The logic is if the counter value is over the preset then reset the counter and if the underflow is on then set the counter value to the preset value. As you can see in the example when the counter

value is greater than the counter preset the counter reset is triggered and the value is now 0. The underflow output %Q2 will set the counter value at the preset when counting backwards.

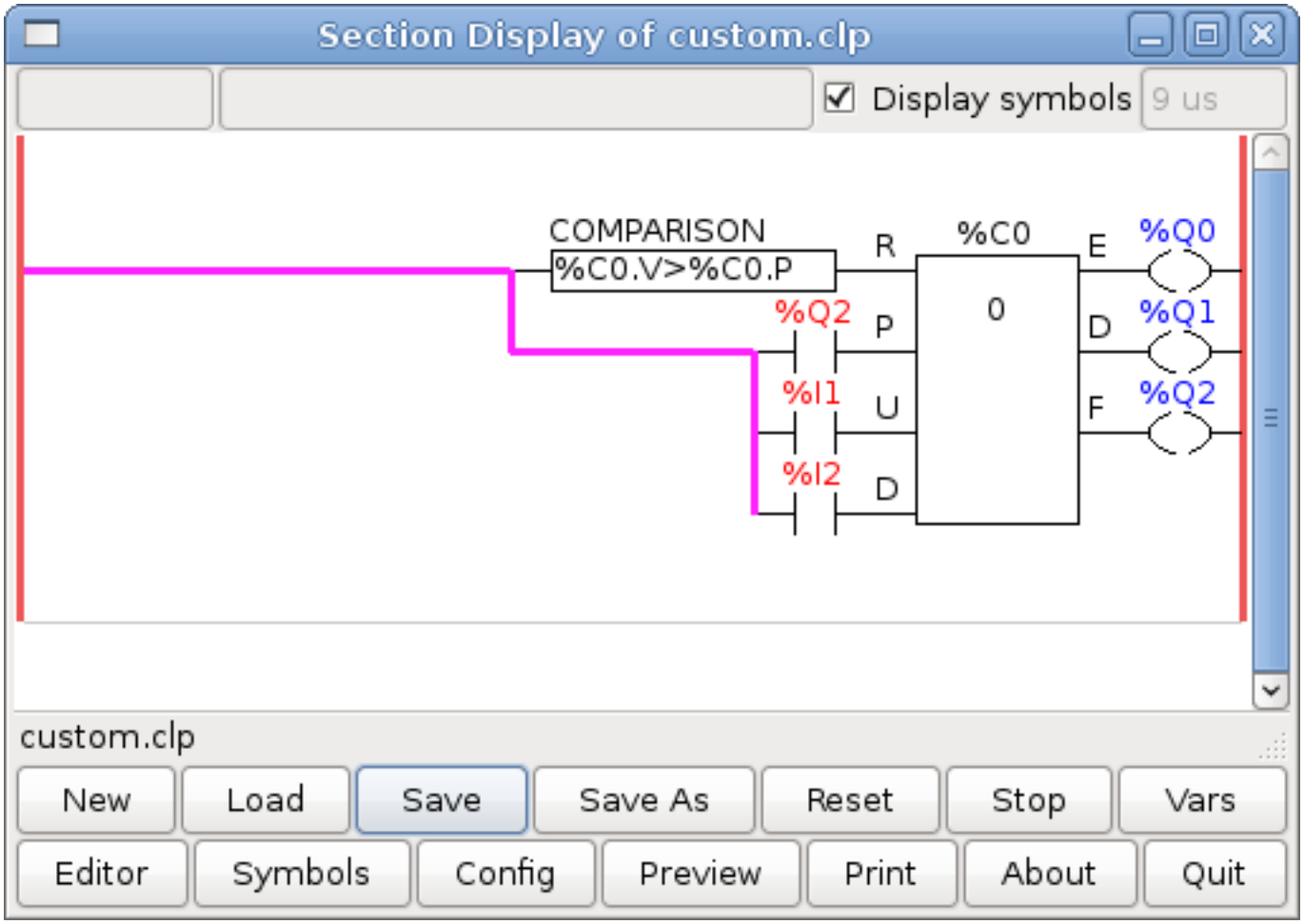



Figure 11.18: Wrapping Counter

### 11.3.2 Reject Extra Pulses

This example shows you how to reject extra pulses from an input. Suppose the input pulse %I0 has an annoying habit of giving an extra pulse that spoils our logic. The TOF (Timer Off Delay) prevents the extra pulse from reaching our cleaned up output %Q0. How this works is when the timer gets an input the output of the timer is on for the duration of the time setting. Using a normally closed contact %TM0.Q the output of the timer blocks any further inputs from reaching our output until it times out.

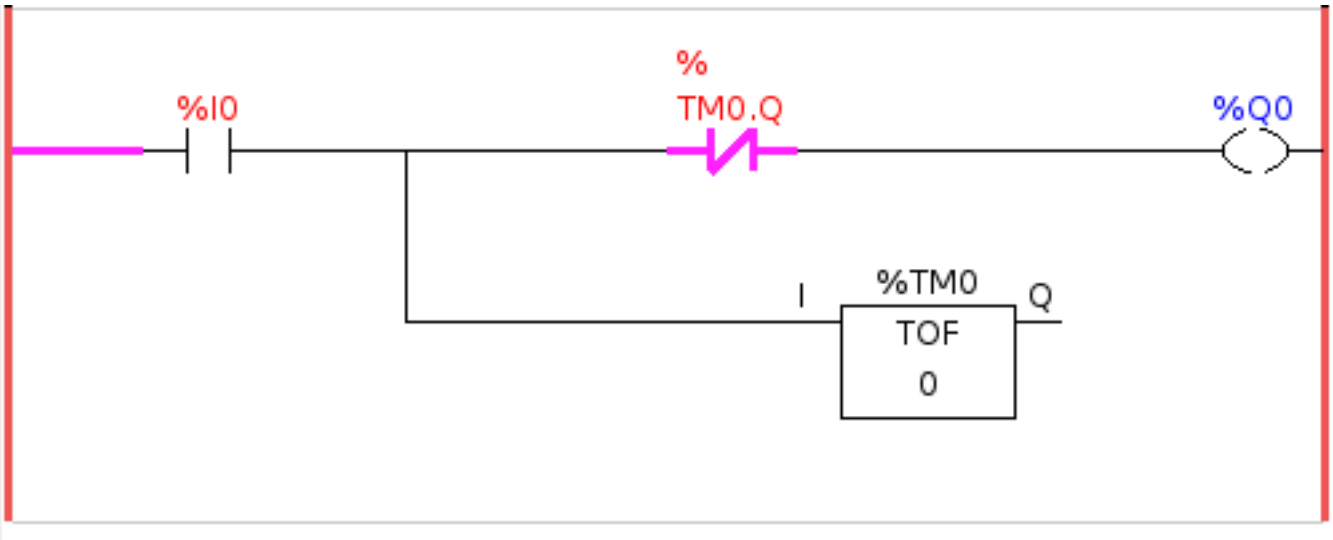



Figure 11.19: Reject Extra Pulse

### 11.3.3 External E-Stop

The External E-Stop example is in the /config/classicladder/cl-estop folder. It uses a pyVCP panel to simulate the external components.

To interface an external E-Stop to LinuxCNC and have the external E-Stop work together with the internal E-Stop requires a couple of connections through Classic Ladder.

First we have to open the E-Stop loop in the main HAL file by commenting out by adding the pound sign as shown or removing the following lines.

```
# net estop-out <= iocontrol.0.user-enable-out
# net estop-out => iocontrol.0.emc-enable-in
```

Next we add Classic Ladder to our custom.hal file by adding these two lines:

```
loadrt classicladder_rt
addf classicladder.0.refresh servo-thread
```

Next we run our config and build the ladder as shown here.

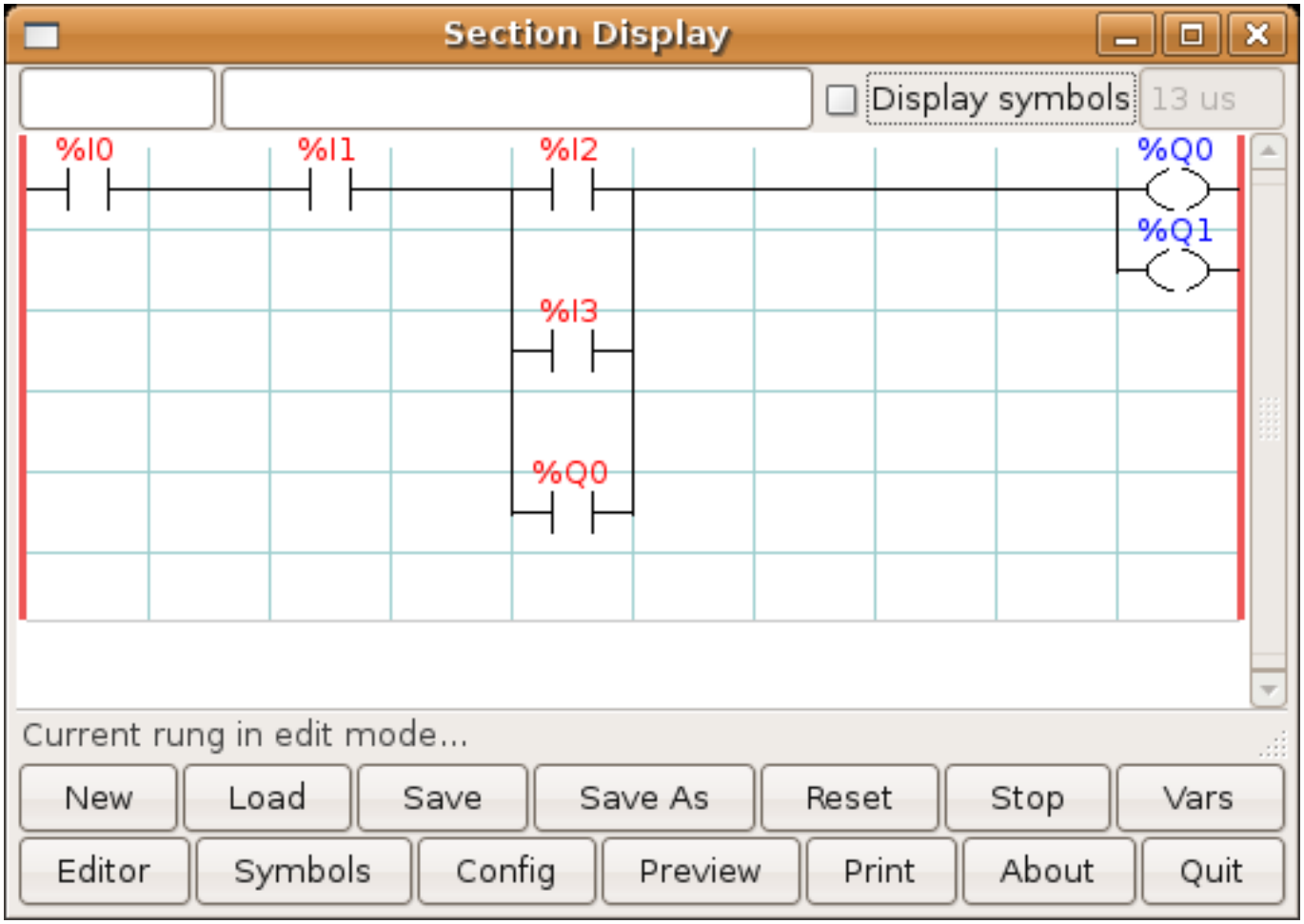



Figure 11.20: E-Stop Section Display

After building the ladder select Save As and save the ladder as estop.clp

Now add the following line to your custom.hal file.

```
# Load the ladder
loadusr classicladder --nogui estop.clp
```

I/O assignments

- %I0 = Input from the pyVCP panel simulated E-Stop (the checkbox)
- %I1 = Input from LinuxCNC's E-Stop
- %I2 = Input from LinuxCNC's E-Stop Reset Pulse
- %I3 = Input from the pyVCP panel reset button
- %Q0 = Ouput to LinuxCNC to enable
- %Q1 = Output to external driver board enable pin (use a N/C output if your board had a disable pin)

Next we add the following lines to the custom_postgui.hal file

```
# E-Stop example using pyVCP buttons to simulate external components

# The pyVCP checkbutton simulates a normally closed external E-Stop
net ext-estop classicladder.0.in-00 <= pyvcp.py-estop

# Request E-Stop Enable from LinuxCNC
net estop-all-ok iocontrol.0.emc-enable-in <= classicladder.0.out-00

# Request E-Stop Enable from pyVCP or external source
net ext-estop-reset classicladder.0.in-03 <= pyvcp.py-reset

# This line resets the E-Stop from LinuxCNC
 net emc-reset-estop iocontrol.0.user-request-enable =>
classicladder.0.in-02

# This line enables LinuxCNC to unlatch the E-Stop in Classic Ladder
net emc-estop iocontrol.0.user-enable-out => classicladder.0.in-01

# This line turns on the green indicator when out of E-Stop
net estop-all-ok => pyvcp.py-es-status
```

Next we add the following lines to the panel.xml file. Note you have to open it with the text editor not the default html viewer.

```
<pyvcp>
<vbox>
<label><text>"E-Stop Demo"</text></label>
<led>
<halpin>"py-es-status"</halpin>
<size>50</size>
<on_color>"green"</on_color>
<off_color>"red"</off_color>
</led>
<checkbutton>
<halpin>"py-estop"</halpin>
<text>"E-Stop"</text>
</checkbutton>
</vbox>
<button>
<halpin>"py-reset"</halpin>
<text>"Reset"</text>
</button>
</pyvcp>
```

Now start up your config and it should look like this.

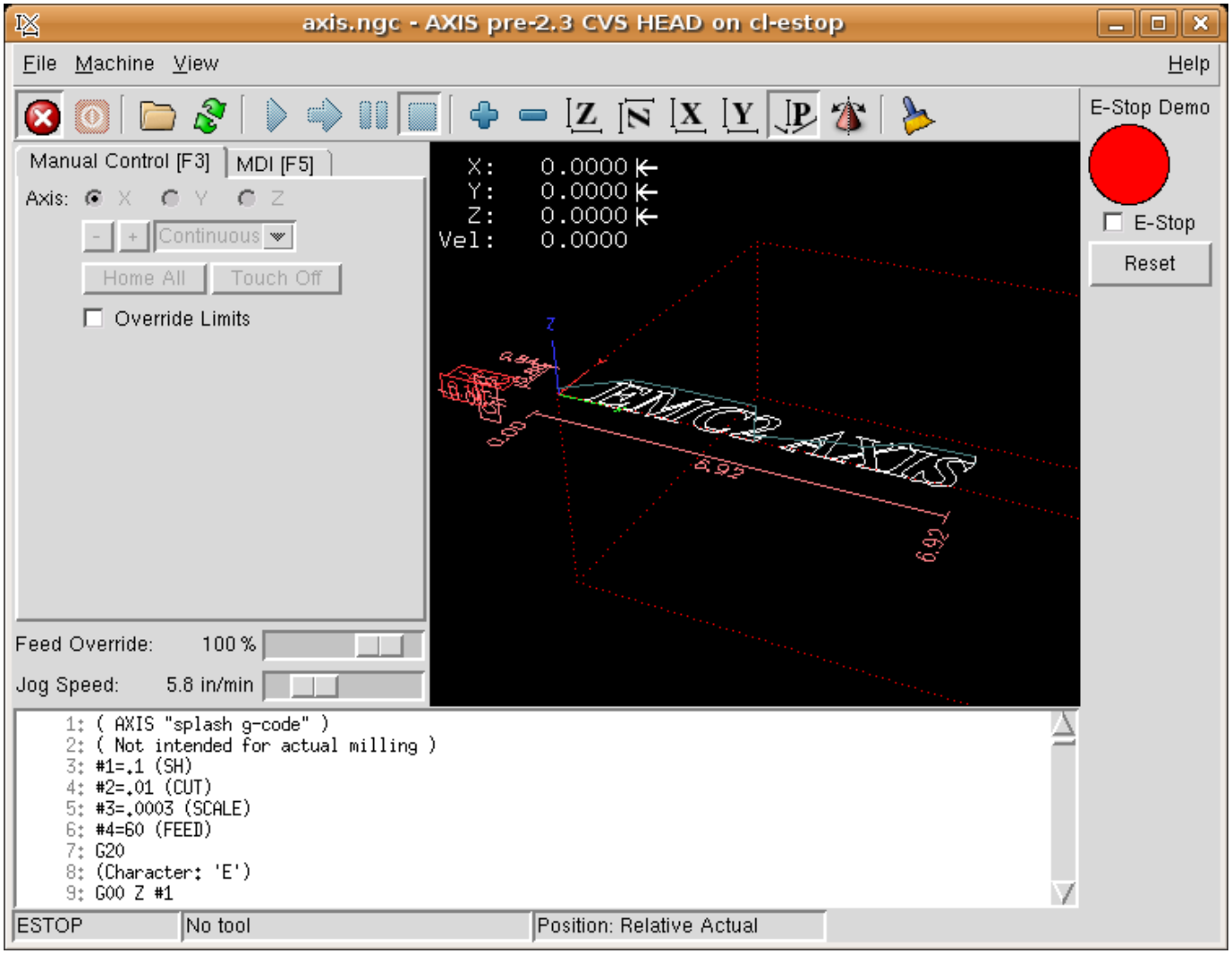



Figure 11.21: AXIS E-Stop

Note that in this example like in real life you must clear the remote E-Stop (simulated by the checkbox) before the AXIS E-Stop or the external Reset will put you in OFF mode. If the E-Stop in the AXIS screen was pressed, you must press it again to clear it. You cannot reset from the external after you do an E-Stop in AXIS.

### 11.3.4 Timer/Operate Example

In this example we are using the Operate block to assign a value to the timer preset based on if an input is on or off.

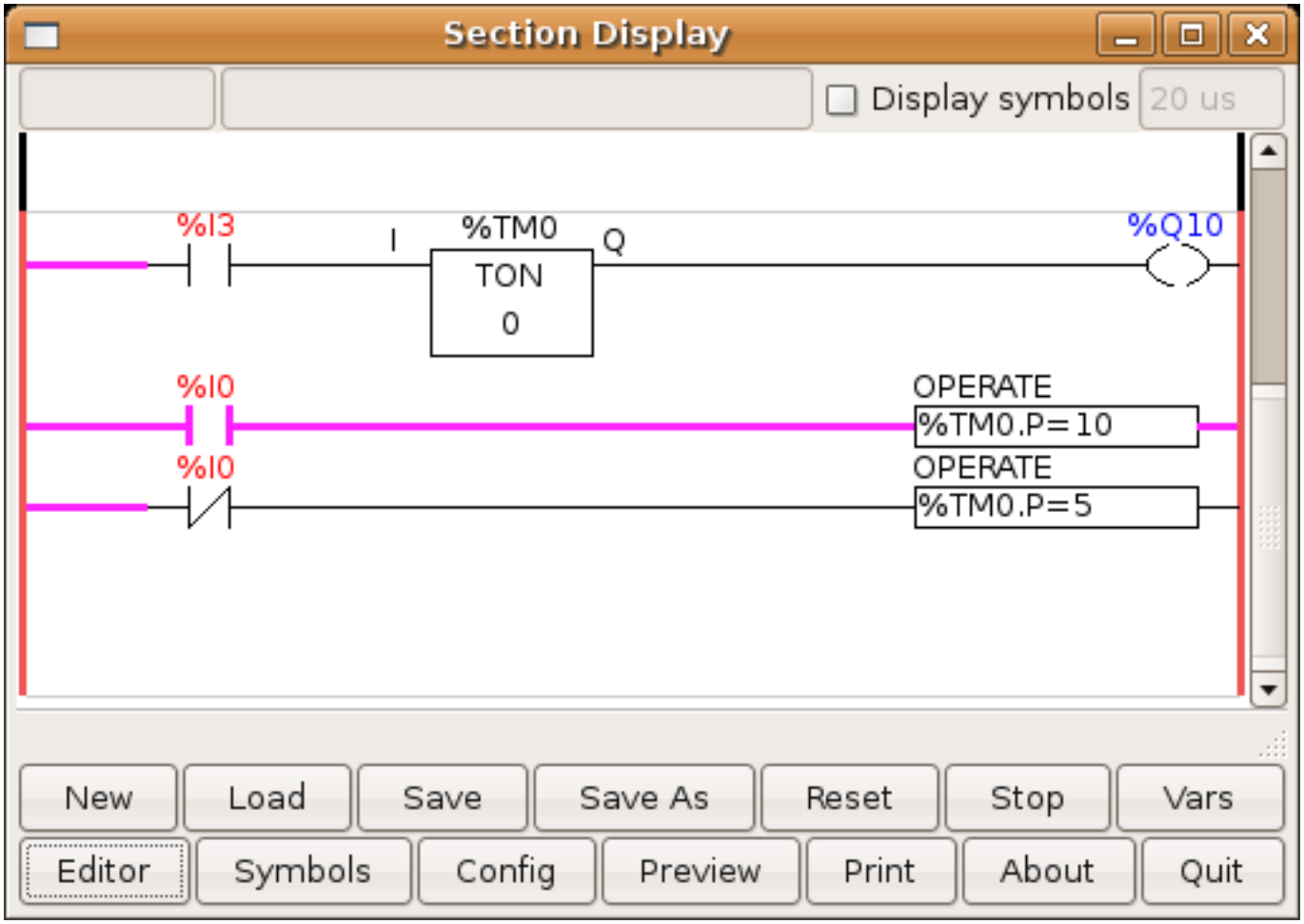



Figure 11.22: Timer/Operate Example

In this case %I0 is true so the timer preset value is 10. If %I0 was false the timer preset would be 5.

# Chapter 12

# HAL

## 12.1 HAL Introduction

HAL stands for Hardware Abstraction Layer. At the highest level, it is simply a way to allow a number of *building blocks* to be loaded and interconnected to assemble a complex system. The *Hardware* part is because HAL was originally designed to make it easier to configure LinuxCNC for a wide variety of hardware devices. Many of the building blocks are drivers for hardware devices. However, HAL can do more than just configure hardware drivers.

### 12.1.1 HAL is based on traditional system design techniques

HAL is based on the same principles that are used to design hardware circuits and systems, so it is useful to examine those principles first.

Any system (including a CNC machine), consists of interconnected components. For the CNC machine, those components might be the main controller, servo amps or stepper drives, motors, encoders, limit switches, pushbutton pendants, perhaps a VFD for the spindle drive, a PLC to run a toolchanger, etc. The machine builder must select, mount and wire these pieces together to make a complete system.

#### 12.1.1.1 Part Selection

The machine builder does not need to worry about how each individual part works. He treats them as black boxes. During the design stage, he decides which parts he is going to use - steppers or servos, which brand of servo amp, what kind of limit switches and how many, etc. The integrator's decisions about which specific components to use is based on what that component does and the specifications supplied by the manufacturer of the device. The size of a motor and the load it must drive will affect the choice of amplifier needed to run it. The choice of amplifier may affect the kinds of feedback needed by the amp and the velocity or position signals that must be sent to the amp from a control.

In the HAL world, the integrator must decide what HAL components are needed. Usually every interface card will require a driver. Additional components may be needed for software generation of step pulses, PLC functionality, and a wide variety of other tasks.

#### 12.1.1.2 Interconnection Design

The designer of a hardware system not only selects the parts, he also decides how those parts will be interconnected. Each black box has terminals, perhaps only two for a simple switch, or dozens for a servo drive or PLC. They need to be wired together. The motors connect to the servo amps, the limit switches connect to the controller, and so on. As the machine builder works on the design, he creates a large wiring diagram that shows how all the parts should be interconnected.

When using HAL, components are interconnected by signals. The designer must decide which signals are needed, and what they should connect.

#### 12.1.1.3 Implementation

Once the wiring diagram is complete it is time to build the machine. The pieces need to be acquired and mounted, and then they are interconnected according to the wiring diagram. In a physical system, each interconnection is a piece of wire that needs to be cut and connected to the appropriate terminals.

HAL provides a number of tools to help *build* a HAL system. Some of the tools allow you to *connect* (or disconnect) a single *wire*. Other tools allow you to save a complete list of all the parts, wires, and other information about the system, so that it can be *rebuilt* with a single command.

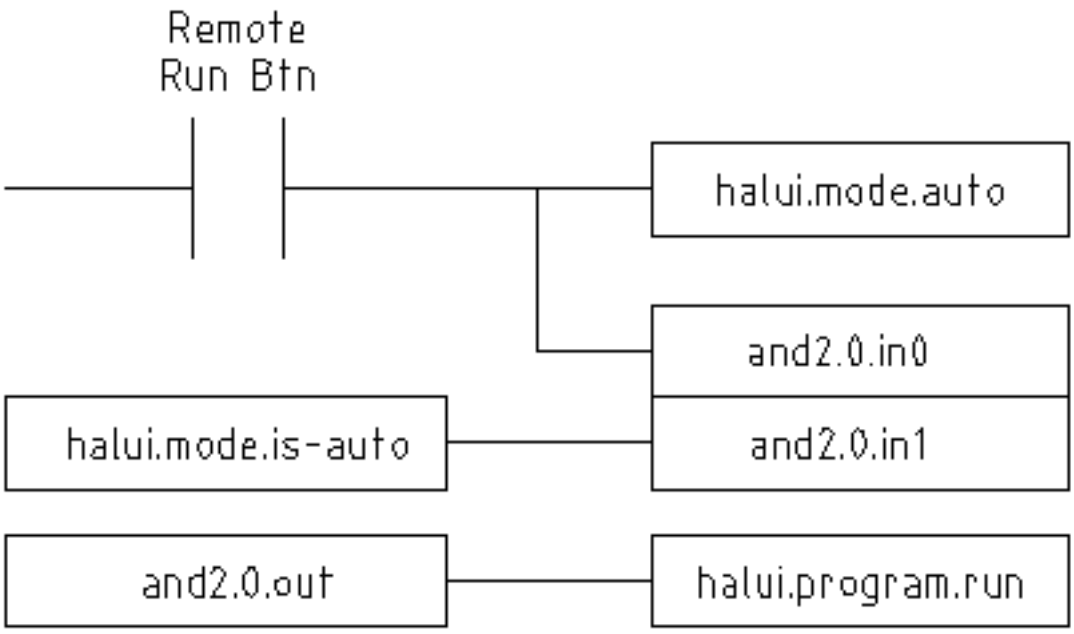
#### 12.1.1.4 Testing

Very few machines work right the first time. While testing, the builder may use a meter to see whether a limit switch is working or to measure the DC voltage going to a servo motor. He may hook up an oscilloscope to check the tuning of a drive, or to look for electrical noise. He may find a problem that requires the wiring diagram to be changed; perhaps a part needs to be connected differently or replaced with something completely different.

HAL provides the software equivalents of a voltmeter, oscilloscope, signal generator, and other tools needed for testing and tuning a system. The same commands used to build the system can be used to make changes as needed.

#### 12.1.1.5 Summary

This document is aimed at people who already know how to do this kind of hardware system integration, but who do not know how to connect the hardware to LinuxCNC. See the Remote Start Example section in the HAL UI Examples documentation.




The traditional hardware design as described above ends at the edge of the main control. Outside the control are a bunch of relatively simple boxes, connected together to do whatever is needed. Inside, the control is a big mystery — one huge black box that we hope works.

HAL extends this traditional hardware design method to the inside of the big black box. It makes device drivers and even some internal parts of the controller into smaller black boxes that can be interconnected and even replaced just like the external hardware. It allows the *system wiring diagram* to show part of the internal controller, rather than just a big black box. And most importantly, it allows the integrator to test and modify the controller using the same methods he would use on the rest of the hardware.

Terms like motors, amps, and encoders are familiar to most machine integrators. When we talk about using extra flexible eight conductor shielded cable to connect an encoder to the servo input board in the computer, the reader immediately understands

what it is and is led to the question, *what kinds of connectors will I need to make up each end.* The same sort of thinking is essential for the HAL but the specific train of thought may take a bit to get on track. Using HAL words may seem a bit strange at first, but the concept of working from one connection to the next is the same.

This idea of extending the wiring diagram to the inside of the controller is what HAL is all about. If you are comfortable with the idea of interconnecting hardware black boxes, you will probably have little trouble using HAL to interconnect software black boxes.

### 12.1.2 HAL Concepts

This section is a glossary that defines key HAL terms but it is a bit different than a traditional glossary because these terms are not arranged in alphabetical order. They are arranged by their relationship or flow in the HAL way of things.

**Component**
When we talked about hardware design, we referred to the individual pieces as *parts*, *building blocks*, *black boxes*, etc. The HAL equivalent is a *component* or *HAL component*. (This document uses *HAL component* when there is likely to be confusion with other kinds of components, but normally just uses *component*.) A HAL component is a piece of software with well-defined inputs, outputs, and behavior, that can be installed and interconnected as needed.

**Parameter**
Many hardware components have adjustments that are not connected to any other components but still need to be accessed. For example, servo amps often have trim pots to allow for tuning adjustments, and test points where a meter or scope can be attached to view the tuning results. HAL components also can have such items, which are referred to as *parameters*. There are two types of parameters: Input parameters are equivalent to trim pots - they are values that can be adjusted by the user, and remain fixed once they are set. Output parameters cannot be adjusted by the user - they are equivalent to test points that allow internal signals to be monitored.

**Pin**
Hardware components have terminals which are used to interconnect them. The HAL equivalent is a *pin* or *HAL pin*. (*HAL pin* is used when needed to avoid confusion.) All HAL pins are named, and the pin names are used when interconnecting them. HAL pins are software entities that exist only inside the computer.

**Physical_Pin**
Many I/O devices have real physical pins or terminals that connect to external hardware, for example the pins of a parallel port connector. To avoid confusion, these are referred to as *physical pins*. These are the things that *stick out* into the real world.

**Signal**
In a physical machine, the terminals of real hardware components are interconnected by wires. The HAL equivalent of a wire is a *signal* or *HAL signal*. HAL signals connect HAL pins together as required by the machine builder. HAL signals can be disconnected and reconnected at will (even while the machine is running).

**Type**
When using real hardware, you would not connect a 24 volt relay output to the +/-10V analog input of a servo amp. HAL pins have the same restrictions, which are based upon their type. Both pins and signals have types, and signals can only be connected to pins of the same type. Currently there are 4 types, as follows:

- bit - a single TRUE/FALSE or ON/OFF value
- float - a 64 bit floating point value, with approximately 53 bits of resolution and over 1000 bits of dynamic range.
- u32 - a 32 bit unsigned integer, legal values are 0 to 4,294,967,295
- s32 - a 32 bit signed integer, legal values are -2,147,483,647 to +2,147,483,647

**Function**
Real hardware components tend to act immediately on their inputs. For example, if the input voltage to a servo amp changes, the output also changes automatically. However software components cannot act *automatically*. Each component has specific code that must be executed to do whatever that component is supposed to do. In some cases, that code simply runs as part of the component. However in most cases, especially in realtime components, the code must run in a specific sequence and at specific intervals. For example, inputs should be read before calculations are performed on the input data,

and outputs should not be written until the calculations are done. In these cases, the code is made available to the system in the form of one or more *functions*. Each function is a block of code that performs a specific action. The system integrator can use *threads* to schedule a series of functions to be executed in a particular order and at specific time intervals.

**Thread**
A *thread* is a list of functions that runs at specific intervals as part of a realtime task. When a thread is first created, it has a specific time interval (period), but no functions. Functions can be added to the thread, and will be executed in order every time the thread runs.

As an example, suppose we have a parport component named hal_parport. That component defines one or more HAL pins for each physical pin. The pins are described in that component's doc section: their names, how each pin relates to the physical pin, are they inverted, can you change polarity, etc. But that alone doesn't get the data from the HAL pins to the physical pins. It takes code to do that, and that is where functions come into the picture. The parport component needs at least two functions: one to read the physical input pins and update the HAL pins, the other to take data from the HAL pins and write it to the physical output pins. Both of these functions are part of the parport driver.

### 12.1.3 HAL components

Each HAL component is a piece of software with well-defined inputs, outputs, and behavior, that can be installed and interconnected as needed. This section lists some of the available components and a brief description of what each does. Complete details for each component are available later in this document.

#### 12.1.3.1 External Programs with HAL hooks

**motion**
A realtime module that accepts NML [1] motion commands and interacts with HAL

**iocontrol**
A user space module that accepts NML I/O commands and interacts with HAL

**classicladder**
A PLC using HAL for all I/O

**halui**
A user space program that interacts with HAL and sends NML commands, it is intended to work as a full User Interface using external knobs & switches

#### 12.1.3.2 Internal Components

**stepgen**
Software step pulse generator with position loop. See section [sec:stepgen]

**encoder**
Software based encoder counter. See section [sec:encoder]

**pid**
Proportional/Integral/Derivative control loops. See section [sec:pid]

**siggen**
A sine/cosine/triangle/square wave generator for testing. See section [sec:siggen]

**supply**
a simple source for testing

[1] Neutral Message Language provides a mechanism for handling multiple types of messages in the same buffer as well as simplifying the interface for encoding and decoding buffers in neutral format and the configuration mechanism.

#### 12.1.3.3 Hardware Drivers

**hal_ax5214h**
A driver for the Axiom Measurement & Control AX5241H digital I/O board

**hal_gm**
General Mechatronics GM6-PCI board

**hal_m5i20**
Mesa Electronics 5i20 board

**hal_motenc**
Vital Systems MOTENC-100 board

**hal_parport**
PC parallel port.

**hal_ppmc**
Pico Systems family of controllers (PPMC, USC and UPC)

**hal_stg**
Servo To Go card (version 1 & 2)

**hal_vti**
Vigilant Technologies PCI ENCDAC-4 controller

#### 12.1.3.4 Tools and Utilities

**halcmd**
Command line tool for configuration and tuning. See section Section 12.7.1

**halgui**
GUI tool for configuration and tuning (not implemented yet).

**halmeter**
A handy multimeter for HAL signals. See section [sec:halmeter].

**halscope**
A full featured digital storage oscilloscope for HAL signals. See the Halscope section.

Each of these building blocks is described in detail in later chapters.

### 12.1.4 Timing Issues In HAL

Unlike the physical wiring models between black boxes that we have said that HAL is based upon, simply connecting two pins with a hal-signal falls far short of the action of the physical case.

True relay logic consists of relays connected together, and when a contact opens or closes, current flows (or stops) immediately. Other coils may change state, etc, and it all just *happens*. But in PLC style ladder logic, it doesn't work that way. Usually in a single pass through the ladder, each rung is evaluated in the order in which it appears, and only once per pass. A perfect example is a single rung ladder, with a NC contact in series with a coil. The contact and coil belong to the same relay.

If this were a conventional relay, as soon as the coil is energized, the contacts begin to open and de-energize it. That means the contacts close again, etc, etc. The relay becomes a buzzer.

With a PLC, if the coil is OFF and the contact is closed when the PLC begins to evaluate the rung, then when it finishes that pass, the coil is ON. The fact that turning on the coil opens the contact feeding it is ignored until the next pass. On the next pass, the PLC sees that the contact is open, and de-energizes the coil. So the relay still switches rapidly between on and off, but at a rate determined by how often the PLC evaluates the rung.

In HAL, the function is the code that evaluates the rung(s). In fact, the HAL-aware realtime version of ClassicLadder exports a function to do exactly that. Meanwhile, a thread is the thing that runs the function at specific time intervals. Just like you can choose to have a PLC evaluate all its rungs every 10 ms, or every second, you can define HAL threads with different periods.

What distinguishes one thread from another is *not* what the thread does - that is determined by which functions are connected to it. The real distinction is simply how often a thread runs.

In LinuxCNC you might have a 50 us thread and a 1 ms thread. These would be created based on BASE_PERIOD and SERVO_PERIOD, the actual times depend on the values in your ini file.

The next step is to decide what each thread needs to do. Some of those decisions are the same in (nearly) any LinuxCNC system—For instance, motion-command-handler is always added to servo-thread.

Other connections would be made by the integrator. These might include hooking the STG driver's encoder read and DAC write functions to the servo thread, or hooking stepgen's function to the base-thread, along with the parport function(s) to write the steps to the port.

## 12.2 Basic HAL Reference

This document provides a reference to the basics of HAL.

### 12.2.1 HAL Commands

More detailed information can be found in the man page for halcmd: run *man halcmd* in a terminal window.

To see the HAL configuration and check the status of pins and parameters use the HAL Configuration window on the Machine menu in AXIS. To watch a pin status open the Watch tab and click on each pin you wish to watch and it will be added to the watch window.

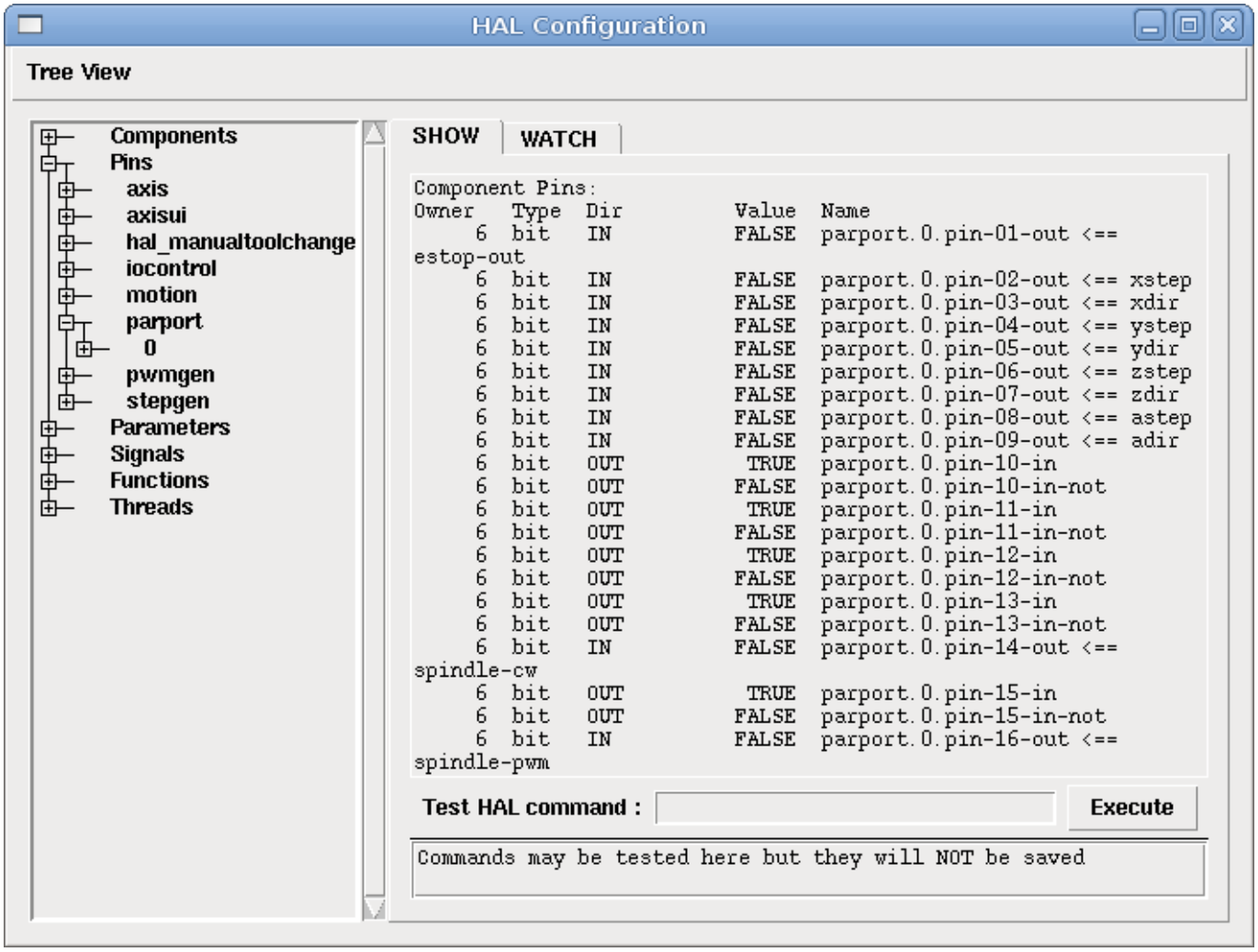



Figure 12.1: HAL Configuration Window

#### 12.2.1.1 loadrt

The command *loadrt* loads a real time HAL component. Real time component functions need to be added to a thread to be updated at the rate of the thread. You cannot load a user space component into the real time space.

The syntax and an example:

```
loadrt <component> <options>

loadrt mux4 count=1
```

#### 12.2.1.2 addf

Adds function *functname* to thread *threadname*. Default is to add the function in the order they are in the file. If *position* is specified, adds the function to that spot in the thread. Negative *position* means position with respect to the end of the thread. For example *1* is start of thread, *-1* is the end of the thread, *-3* is third from the end.

Some functions it is important to load them in a certain order like the parport read and write functions. The function name is usually the the component name plus a number. In the following example the component *or2* is loaded and *show function* shows the name of the or2 function

```
$ halrun
halcmd: loadrt or2
halcmd: show function
Exported Functions:
Owner   CodeAddr  Arg       FP   Users  Name
 00004  f8bc5000  f8f950c8  NO       0   or2.0
```

You have to add a function from a HAL real time component to a thread to get the function to update at the rate of the thread.

Usually there are two threads as shown in this example. Some components use floating point math and must be added to a thread that supports floating point math. The *FP* indicates if floating point math is supported in that thread.

```
$ halrun
halcmd: loadrt motmod base_period_nsec=55555 servo_period_nsec=1000000 num_joints=3
halcmd: show thread
Realtime Threads:
     Period  FP     Name               (     Time, Max-Time )
     995976  YES          servo-thread (        0,        0 )
      55332  NO            base-thread (        0,        0 )
```

- base-thread (the high-speed thread): this thread handles items that need a fast response, like making step pulses, and reading and writing the parallel port. Does not support floating point math.
- servo-thread (the slow-speed thread): this thread handles items that can tolerate a slower response, like the motion controller, ClassicLadder, and the motion command handler and supports floating point math.

The syntax and an example:

```
addf <function> <thread>

addf mux4.0 servo-thread
```

#### 12.2.1.3 loadusr

The command *loadusr* loads a user space HAL component. User space programs are their own separate processes, which optionally talk to other HAL components via pins and parameters. You cannot load real time components into user space.

Flags may be one or more of the following:

| | |
|---|---|
| -W | to wait for the component to become ready. The component is assumed to have the same name as the first argument of the command. |
| -Wn <name> | to wait for the component, which will have the given <name>. This only applies if the component has a name option. |
| -w | to wait for the program to exit |
| -i | to ignore the program return value (with -w) |
| -n | name a component when it is a valid option for that component. |

The syntax and examples:

```
loadusr <component> <options>

loadusr halui
```

```
loadusr -Wn spindle gs2_vfd -n spindle
```

In English it means *loadusr wait for name spindle component gs2_vfd name spindle*.

#### 12.2.1.4 net

The command *net* creates a *connection* between a signal and one or more pins. If the signal does not exist net creates the new signal. This replaces the need to use the command newsig. The optional direction arrows <=, => and <=> make it easier to follow the logic when reading a *net* command line and are not used by the net command. The direction arrows must be separated by a space from the pin names.

**Syntax and Example:**

```
net signal-name pin-name <optional arrow> <optional second pin-name>

net home-x axis.0.home-sw-in <= parport.0.pin-11-in
```

In the above example *home-x* is the signal name, *axis.0.home-sw-in* is a *Direction IN* pin, <= is the optional direction arrow, and *parport.0.pin-11-in* is a *Direction OUT* pin. This may seem confusing but the in and out labels for a parallel port pin indicates the physical way the pin works not how it is handled in HAL.

A pin can be connected to a signal if it obeys the following rules:

- An IN pin can always be connected to a signal
- An IO pin can be connected unless there's an OUT pin on the signal
- An OUT pin can be connected only if there are no other OUT or IO pins on the signal

The same *signal-name* can be used in multiple net commands to connect additional pins, as long as the rules above are obeyed.

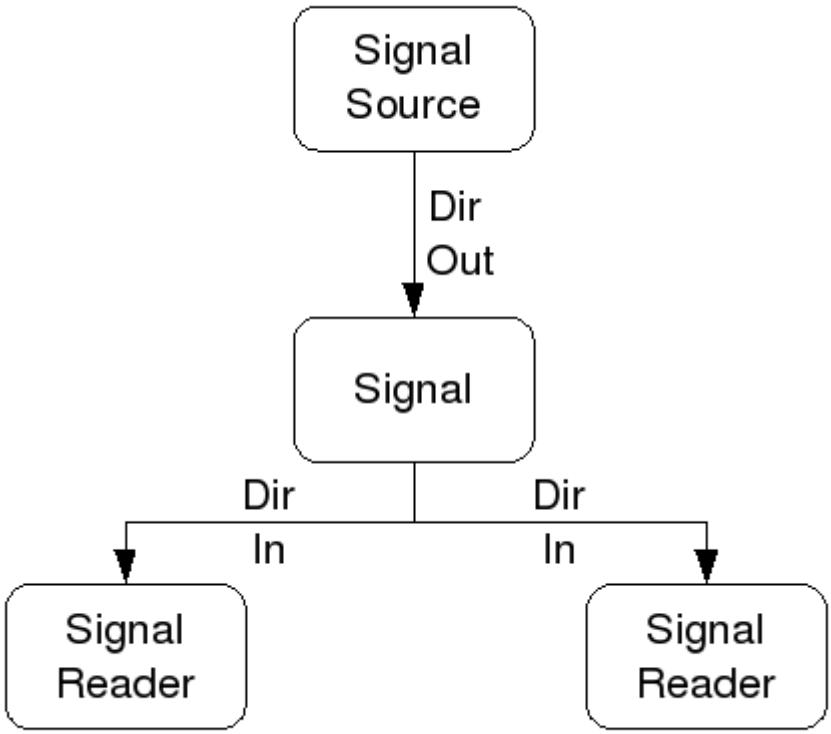



Figure 12.2: Signal Direction

This example shows the signal xStep with the source being stepgen.0.out and with two readers, parport.0.pin-02-out and parport.0.pin-08-out. Basically the value of stepgen.0.out is sent to the signal xStep and that value is then sent to parport.0.pin-02-out and parport.0.pin-08-out.

```
#   signal    source            destination          destination
net xStep stepgen.0.out => parport.0.pin-02-out parport.0.pin-08-out
```

Since the signal xStep contains the value of stepgen.0.out (the source) you can use the same signal again to send the value to another reader. To do this just use the signal with the readers on another line.

```
net xStep => parport.0.pin-02-out
```

**I/O pins** An I/O pin like encoder.N.index-enable can be read or set as allowed by the component.

#### 12.2.1.5 setp

The command *setp* sets the value of a pin or parameter. The valid values will depend on the type of the pin or parameter. It is an error if the data types do not match.

Some components have parameters that need to be set before use. Parameters can be set before use or while running as needed. You cannot use setp on a pin that is connected to a signal.

The syntax and an example:

```
setp <pin/parameter-name> <value>

setp parport.0.pin-08-out TRUE
```

#### 12.2.1.6 sets

The command *sets* sets the value of a signal.

The syntax and an example:

```
sets <signal-name> <value>

net mysignal and2.0.in0 pyvcp.my-led

sets mysignal 1
```

It is an error if:

- The signal-name does not exist
- If the signal already has a writer
- If value is not the correct type for the signal

#### 12.2.1.7 unlinkp

The command *unlinkp* unlinks a pin from the connected signal. If no signal was connected to the pin prior running the command, nothing happens. The *unlinkp* command is useful for trouble shooting.

The syntax and an example:

```
unlinkp <pin-name>

unlinkp parport.0.pin-02-out
```

#### 12.2.1.8 Obsolete Commands

The following commands are depreciated and may be removed from future versions. Any new configuration should use the net command. These commands are included so older configurations will still work.

**linksp** The command *linksp* creates a *connection* between a signal and one pin.

The syntax and an example:

```
linksp <signal-name> <pin-name>
linksp X-step parport.0.pin-02-out
```

The *linksp* command has been superseded by the *net* command.

**linkps** The command *linkps* creates a *connection* between one pin and one signal. It is the same as linksp but the arguments are reversed.

The syntax and an example:

```
linkps <pin-name> <signal-name>

linkps parport.0.pin-02-out X-Step
```

The *linkps* command has been superseded by the *net* command.

**newsig** the command *newsig* creates a new HAL signal by the name <signame> and the data type of <type>. Type must be *bit*, *s32*, *u32* or *float*. Error if <signame> all ready exists.

The syntax and an example:

```
newsig <signame> <type>

newsig Xstep bit
```

More information can be found in the HAL manual or the man pages for halrun.

### 12.2.2 HAL Data

#### 12.2.2.1 Bit

A bit value is an on or off.

- bit values = true or 1 and false or 0 (True, TRUE, true are all valid)

#### 12.2.2.2 Float

A *float* is a floating point number. In other words the decimal point can move as needed.

- float values = a 64 bit floating point value, with approximately 53 bits of resolution and over 1000 bits of dynamic range.

For more information on floating point numbers see:

http://en.wikipedia.org/wiki/Floating_point

#### 12.2.2.3 s32

An *s32* number is a whole number that can have a negative or positive value.

- s32 values = integer numbers -2147483648 to 2147483647

#### 12.2.2.4 u32

A *u32* number is a whole number that is positive only.

- u32 values = integer numbers 0 to 4294967295

### 12.2.3 HAL Files

If you used the Stepper Config Wizard to generate your config you will have up to three HAL files in your config directory.

- my-mill.hal (if your config is named *my-mill*) This file is loaded first and should not be changed if you used the Stepper Config Wizard.
- custom.hal This file is loaded next and before the GUI loads. This is where you put your custom HAL commands that you want loaded before the GUI is loaded.
- custom_postgui.hal This file is loaded after the GUI loads. This is where you put your custom HAL commands that you want loaded after the GUI is loaded. Any HAL commands that use pyVCP widgets need to be placed here.

### 12.2.4 HAL Components

Two parameters are automatically added to each HAL component when it is created. These parameters allow you to scope the execution time of a component.

`.time`

`.tmax`

Time is the number of CPU cycles it took to execute the function.

Tmax is the maximum number of CPU cycles it took to execute the function. Tmax is a read/write parameter so the user can set it to 0 to get rid of the first time initialization on the function's execution time.

### 12.2.5 Logic Components

HAL contains several real time logic components. Logic components follow a *Truth Table* that states what the output is for any given input. Typically these are bit manipulators and follow electrical logic gate truth tables.

#### 12.2.5.1 and2

The *and2* component is a two input *and* gate. The truth table below shows the output based on each combination of input.

Syntax

```
and2 [count=N] | [names=name1[,name2...]]
```

Functions

and2.n

Pins

```
and2.N.in0 (bit, in)
and2.N.in1 (bit, in)
and2.N.out (bit, out)
```

Truth Table

| in0 | in1 | out |
|---|---|---|
| False | False | False |
| True | False | False |
| False | True | False |
| True | True | True |

#### 12.2.5.2 not

The *not* component is a bit inverter.

Syntax

```
not [count=n] | [names=name1[,name2...]]
```

Functions

```
not.all
not.n
```

Pins

```
not.n.in (bit, in)
not.n.out (bit, out)
```

Truth Table

| in | out |
|---|---|
| True | False |
| False | True |

#### 12.2.5.3 or2

The *or2* component is a two input OR gate.

Syntax

```
or2[count=n] | [names=name1[,name2...]]
```

Functions

`or2.n`

Pins

```
or2.n.in0 (bit, in)
or2.n.in1 (bit, in)
or2.n.out (bit, out)
```

Truth Table

| in0 | in1 | out |
|---|---|---|
| True | False | True |
| True | True | True |
| False | True | True |
| False | False | False |

#### 12.2.5.4 xor2

The *xor2* component is a two input XOR (exclusive OR)gate.

Syntax

```
xor2[count=n] | [names=name1[,name2...]]
```

Functions

`xor2.n`

Pins

```
xor2.n.in0 (bit, in)
xor2.n.in1 (bit, in)
xor2.n.out (bit, out)
```

Truth Table

| in0 | in1 | out |
|---|---|---|
| True | False | True |
| True | True | False |
| False | True | True |
| False | False | False |

#### 12.2.5.5 Logic Examples

An *and2* example connecting two inputs to one output.

```
loadrt and2 count=1
```

```
addf and2.0 servo-thread

net my-sigin1 and2.0.in0 <= parport.0.pin-11-in

net my-sigin2 and2.0.in1 <= parport.0.pin-12-in

net both-on parport.0.pin-14-out <= and2.0.out
```

In the above example one copy of and2 is loaded into real time space and added to the servo thread. Next pin 11 of the parallel port is connected to the in0 bit of the and gate. Next pin 12 is connected to the in1 bit of the and gate. Last we connect the and2 out bit to the parallel port pin 14. So following the truth table for and2 if pin 11 and pin 12 are on then the output pin 14 will be on.

### 12.2.6 Conversion Components

#### 12.2.6.1 weighted_sum

The weighted_sum converts a group of bits to an integer. The conversion is the sum of the *weights* of the bits that are on plus any offset. The weight of the m-th bit is 2^m. This is similar to a binary coded decimal but with more options. The *hold* bit stops processing the input changes so the *sum* will not change.

The following syntax is used to load the weighted_sum component.

```
loadrt weighted_sum wsum_sizes=size[,size,...]
```

Creates weighted sum groups each with the given number of input bits (size).

To update the weighted_sum you need to attach process_wsums to a thread.

```
addf process_wsums servo-thread
```

This updates the weighted_sum component.

In the following example clipped from the HAL Configuration window in Axis the bits *0* and *2* are true and there is no offset. The *weight* of 0 is 1 and the *weight* of 2 is 4 so the sum is 5.

**weighted_sum**

```
Component Pins:
Owner   Type  Dir         Value  Name
    10  bit   In           TRUE  wsum.0.bit.0.in
    10  s32   I/O             1  wsum.0.bit.0.weight
    10  bit   In          FALSE  wsum.0.bit.1.in
    10  s32   I/O             2  wsum.0.bit.1.weight
    10  bit   In           TRUE  wsum.0.bit.2.in
    10  s32   I/O             4  wsum.0.bit.2.weight
    10  bit   In          FALSE  wsum.0.bit.3.in
    10  s32   I/O             8  wsum.0.bit.3.weight
    10  bit   In          FALSE  wsum.0.hold
    10  s32   I/O             0  wsum.0.offset
    10  s32   Out             5  wsum.0.sum
```

## 12.3 HAL Tutorial

### 12.3.1 Introduction

Configuration moves from theory to device — HAL device that is. For those who have had just a bit of computer programming, this section is the *Hello World* of the HAL. Halrun can be used to create a working system. It is a command line or text file tool for configuration and tuning. The following examples illustrate its setup and operation.

#### 12.3.1.1 Notation

Terminal commands are shown without the system prompt unless you are running *HAL*. The terminal window is in *Applications/Accessories* from the main Ubuntu menu bar.

**Terminal Command Example**

```
me@computer:~linuxcnc$ halrun
(will be shown like the following line)
halrun

(the halcmd: prompt will be shown when running HAL)
halcmd: loadrt debounce
halcmd: show pin
```

#### 12.3.1.2 Tab-completion

Your version of halcmd may include tab-completion. Instead of completing file names as a shell does, it completes commands with HAL identifiers. You will have to type enough letters for a unique match. Try pressing tab after starting a HAL command:

**Tab Completion**

```
halcmd: loa<TAB>
halcmd: load
halcmd: loadrt
halcmd: loadrt deb<TAB>
halcmd: loadrt debounce
```

#### 12.3.1.3 The RTAPI environment

RTAPI stands for Real Time Application Programming Interface. Many HAL components work in realtime, and all HAL components store data in shared memory so realtime components can access it. Normal Linux does not support realtime programming or the type of shared memory that HAL needs. Fortunately there are realtime operating systems (RTOS's) that provide the necessary extensions to Linux. Unfortunately, each RTOS does things a little differently.

To address these differences, the LinuxCNC team came up with RTAPI, which provides a consistent way for programs to talk to the RTOS. If you are a programmer who wants to work on the internals of LinuxCNC, you may want to study *linuxcnc/src/rtapi/rtapi.h* to understand the API. But if you are a normal person all you need to know about RTAPI is that it (and the RTOS) needs to be loaded into the memory of your computer before you do anything with HAL.

### 12.3.2 A Simple Example

#### 12.3.2.1 Loading a component

For this tutorial, we are going to assume that you have successfully installed the Live CD and, if using a RIP [2], invoked the *rip-environment* script to prepare your shell. In that case, all you need to do is load the required RTOS and RTAPI modules into memory. Just run the following command from a terminal window:

**Loading HAL**

```
cd linuxcnc
halrun
halcmd:
```

[2] Run In Place, when the source files have been downloaded to a user directory.

With the realtime OS and RTAPI loaded, we can move into the first example. Notice that the prompt is now shown as *halcmd:*. This is because subsequent commands will be interpreted as HAL commands, not shell commands.

For the first example, we will use a HAL component called *siggen*, which is a simple signal generator. A complete description of the *siggen* component can be found in the Siggen section of this Manual. It is a realtime component, implemented as a Linux kernel module. To load *siggen* use the HAL command *loadrt*.

**Loading siggen**

```
halcmd: loadrt siggen
```

#### 12.3.2.2 Examining the HAL

Now that the module is loaded, it is time to introduce *halcmd* , the command line tool used to configure the HAL. This tutorial will introduce some halcmd features, for a more complete description try *man halcmd*, or see the reference in Hal Commands section of this document. The first halcmd feature is the *show* command. This command displays information about the current state of the HAL. To show all installed components:

**Show Components**

```
halcmd: show comp

Loaded HAL Components:
ID      Type  Name                                PID   State
    3   RT    siggen                                    ready
    2   User  halcmd2177                          2177  ready
```

Since *halcmd* itself is a HAL component, it will always show up in the list. The number after halcmd in the component list is the process ID. It is possible to run more than one copy of halcmd at the same time (in different windows for example), so the PID is added to the end of the name to make it unique. The list also shows the *siggen* component that we installed in the previous step. The *RT* under *Type* indicates that *siggen* is a realtime component. The *User* under *Type* indicates it is a user space component.

Next, let's see what pins *siggen* makes available:

**Show Pins**

```
halcmd: show pin

Component Pins:
Owner   Type   Dir         Value  Name
     3  float  IN              1  siggen.0.amplitude
     3  bit    OUT         FALSE  siggen.0.clock
     3  float  OUT             0  siggen.0.cosine
     3  float  IN              1  siggen.0.frequency
     3  float  IN              0  siggen.0.offset
     3  float  OUT             0  siggen.0.sawtooth
     3  float  OUT             0  siggen.0.sine
     3  float  OUT             0  siggen.0.square
     3  float  OUT             0  siggen.0.triangle
```

This command displays all of the pins in the current HAL. A complex system could have dozens or hundreds of pins. But right now there are only nine pins. Of these pins eight are floating point and one is bit (boolean). Six carry data out of the *siggen* component and three are used to transfer settings into the component. Since we have not yet executed the code contained within the component, some the pins have a value of zero.

The next step is to look at parameters:

**Show Parameters**

```
halcmd: show param

Parameters:
```

```
Owner   Type  Dir        Value   Name
     3  s32   RO             0   siggen.0.update.time
     3  s32   RW             0   siggen.0.update.tmax
```

The *show param* command shows all the parameters in the HAL. Right now each parameter has the default value it was given when the component was loaded. Note the column labeled *Dir*. The parameters labeled *-W* are writable ones that are never changed by the component itself, instead they are meant to be changed by the user to control the component. We will see how to do this later. Parameters labeled *R-* are read only parameters. They can be changed only by the component. Finally, parameter labeled *RW* are read-write parameters. That means that they are changed by the component, but can also be changed by the user. Note: the parameters *siggen.0.update.time* and *siggen.0.update.tmax* are for debugging purposes, and won't be covered in this section.

Most realtime components export one or more functions to actually run the realtime code they contain. Let's see what function(s) *siggen* exported:

**Show Functions**

```
halcmd: show funct

Exported Functions:
Owner   CodeAddr  Arg       FP   Users  Name
 00003  f801b000  fae820b8  YES      0   siggen.0.update
```

The siggen component exported a single function. It requires floating point. It is not currently linked to any threads, so *users* is zero.

#### 12.3.2.3 Making realtime code run

To actually run the code contained in the function *siggen.0.update*, we need a realtime thread. The component called *threads* that is used to create a new thread. Lets create a thread called *test-thread* with a period of 1 ms (1,000 us or 1,000,000 ns):

```
halcmd: loadrt threads name1=test-thread period1=1000000
```

Let's see if that worked:

**Show Threads**

```
halcmd: show thread

Realtime Threads:
     Period  FP     Name               (     Time, Max-Time )
     999855  YES           test-thread (        0,        0 )
```

It did. The period is not exactly 1,000,000 ns because of hardware limitations, but we have a thread that runs at approximately the correct rate, and which can handle floating point functions. The next step is to connect the function to the thread:

**Add Function**

```
halcmd: addf siggen.0.update test-thread
```

Up till now, we've been using *halcmd* only to look at the HAL. However, this time we used the *addf* (add function) command to actually change something in the HAL. We told *halcmd* to add the function *siggen.0.update* to the thread *test-thread*, and if we look at the thread list again, we see that it succeeded:

```
halcmd: show thread

Realtime Threads:
     Period  FP     Name               (     Time, Max-Time )
     999855  YES          test-thread   (        0,        0 )
                  1 siggen.0.update
```

There is one more step needed before the *siggen* component starts generating signals. When the HAL is first started, the thread(s) are not actually running. This is to allow you to completely configure the system before the realtime code starts. Once you are happy with the configuration, you can start the realtime code like this:

```
halcmd: start
```

Now the signal generator is running. Let's look at its output pins:

```
halcmd: show pin

Component Pins:
Owner   Type  Dir         Value  Name
     3  float IN              1  siggen.0.amplitude
     3  bit   OUT         FALSE  siggen.0.clock
     3  float OUT    -0.1640929  siggen.0.cosine
     3  float IN              1  siggen.0.frequency
     3  float IN              0  siggen.0.offset
     3  float OUT    -0.4475303  siggen.0.sawtooth
     3  float OUT     0.9864449  siggen.0.sine
     3  float OUT            -1  siggen.0.square
     3  float OUT    -0.1049393  siggen.0.triangle
```

And let's look again:

```
halcmd: show pin

Component Pins:
Owner   Type  Dir         Value  Name
     3  float IN              1  siggen.0.amplitude
     3  bit   OUT         FALSE  siggen.0.clock
     3  float OUT     0.0507619  siggen.0.cosine
     3  float IN              1  siggen.0.frequency
     3  float IN              0  siggen.0.offset
     3  float OUT     -0.516165  siggen.0.sawtooth
     3  float OUT     0.9987108  siggen.0.sine
     3  float OUT            -1  siggen.0.square
     3  float OUT    0.03232994  siggen.0.triangle
```

We did two *show pin* commands in quick succession, and you can see that the outputs are no longer zero. The sine, cosine, sawtooth, and triangle outputs are changing constantly. The square output is also working, however it simply switches from +1.0 to -1.0 every cycle.

#### 12.3.2.4 Changing Parameters

The real power of HAL is that you can change things. For example, we can use the *setp* command to set the value of a parameter. Let's change the amplitude of the signal generator from 1.0 to 5.0:

**Set Pin**

```
halcmd: setp siggen.0.amplitude 5
```

**Check the parameters and pins again**

```
halcmd: show param

Parameters:
Owner   Type  Dir         Value  Name
     3  s32   RO           1754  siggen.0.update.time
     3  s32   RW          16997  siggen.0.update.tmax

halcmd: show pin
```

```
Component Pins:
Owner   Type  Dir         Value  Name
     3  float IN              5  siggen.0.amplitude
     3  bit   OUT         FALSE  siggen.0.clock
     3  float OUT     0.8515425  siggen.0.cosine
     3  float IN              1  siggen.0.frequency
     3  float IN              0  siggen.0.offset
     3  float OUT      2.772382  siggen.0.sawtooth
     3  float OUT     -4.926954  siggen.0.sine
     3  float OUT             5  siggen.0.square
     3  float OUT      0.544764  siggen.0.triangle
```

Note that the value of parameter *siggen.0.amplitude* has changed to 5, and that the pins now have larger values.

#### 12.3.2.5 Saving the HAL configuration

Most of what we have done with *halcmd* so far has simply been viewing things with the *show* command. However two of the commands actually changed things. As we design more complex systems with HAL, we will use many commands to configure things just the way we want them. HAL has the memory of an elephant, and will retain that configuration until we shut it down. But what about next time? We don't want to manually enter a bunch of commands every time we want to use the system. We can save the configuration of the entire HAL with a single command:

**Save**

```
halcmd: save
# components
loadrt threads name1=test-thread period1=1000000
loadrt siggen
# pin aliases
# signals
# nets
# parameter values
setp siggen.0.update.tmax 14687
# realtime thread/function links
addf siggen.0.update test-thread
```

The output of the *save* command is a sequence of HAL commands. If you start with an *empty* HAL and run all these commands, you will get the configuration that existed when the *save* command was issued. To save these commands for later use, we simply redirect the output to a file:

**Save to a file**

```
halcmd: save all saved.hal
```

#### 12.3.2.6 Exiting halrun

When you're finished with your HAL session type *exit* at the *halcmd:* prompt. This will return you to the system prompt and close down the HAL session. Do not simply close the terminal window without shutting down the HAL session.

**Exit HAL**

```
halcmd: exit
```

#### 12.3.2.7 Restoring the HAL configuration

To restore the HAL configuration stored in *saved.hal*, we need to execute all of those HAL commands. To do that, we use *-f <file name>* which reads commands from a file, and *-I* (upper case i) which shows the halcmd prompt after executing the commands:

**Run a Saved File**

```
halrun -I -f saved.hal
```

Notice that there is not a *start* command in saved.hal. It's necessary to issue it again (or edit saved.hal to add it there).

#### 12.3.2.8 Removing HAL from memory

If an unexpected shut down of a HAL session occurs you might have to unload HAL before another session can begin. To do this type the following command in a terminal window.

**Removing HAL**

```
halrun -U
```

### 12.3.3 Halmeter

You can build very complex HAL systems without ever using a graphical interface. However there is something satisfying about seeing the result of your work. The first and simplest GUI tool for the HAL is halmeter. It is a very simple program that is the HAL equivalent of the handy Fluke multimeter (or Simpson analog meter for the old timers).

We will use the siggen component again to check out halmeter. If you just finished the previous example, then you can load siggen using the saved file. If not, we can load it just like we did before:

```
halrun
halcmd: loadrt siggen
halcmd: loadrt threads name1=test-thread period1=1000000
halcmd: addf siggen.0.update test-thread
halcmd: start
halcmd: setp siggen.0.amplitude 5
```

At this point we have the siggen component loaded and running. It's time to start halmeter.

**Starting Halmeter**

```
halcmd: loadusr halmeter
```

The first window you will see is the *Select Item to Probe* window.

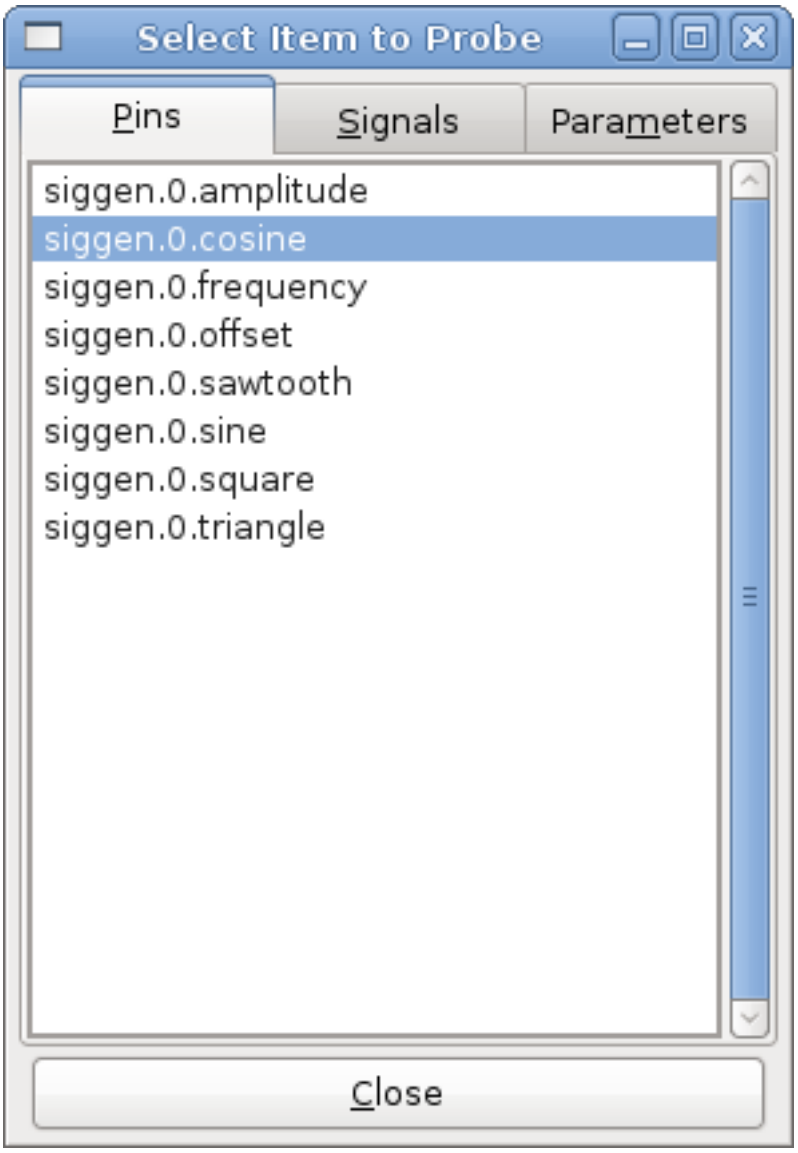



Figure 12.3: Halmeter Select Window

This dialog has three tabs. The first tab displays all of the HAL pins in the system. The second one displays all the signals, and the third displays all the parameters. We would like to look at the pin *siggen.0.cosine* first, so click on it then click the *Close* button. The probe selection dialog will close, and the meter looks something like the following figure.

Figure 12.4: Halmeter

To change what the meter displays press the *Select* button which brings back the *Select Item to Probe* window.

You should see the value changing as siggen generates its cosine wave. Halmeter refreshes its display about 5 times per second.

To shut down halmeter, just click the exit button.

If you want to look at more than one pin, signal, or parameter at a time, you can just start more halmeters. The halmeter window was intentionally made very small so you could have a lot of them on the screen at once.

### 12.3.4 Stepgen Example

Up till now we have only loaded one HAL component. But the whole idea behind the HAL is to allow you to load and connect a number of simple components to make up a complex system. The next example will use two components.

Before we can begin building this new example, we want to start with a clean slate. If you just finished one of the previous examples, we need to remove the all components and reload the RTAPI and HAL libraries.

```
halcmd: exit
```

#### 12.3.4.1 Installing the components

Now we are going to load the step pulse generator component. For a detailed description of this component refer to the stepgen section of the Integrator Manual. In this example we will use the *velocity* control type of stepgen. For now, we can skip the details, and just run the following commands.

```
halrun
halcmd: loadrt stepgen step_type=0,0 ctrl_type=v,v
halcmd: loadrt siggen
halcmd: loadrt threads name1=fast fp1=0 period1=50000 name2=slow period2=1000000
```

The first command loads two step generators, both configured to generate stepping type 0. The second command loads our old friend siggen, and the third one creates two threads, a fast one with a period of 50 microseconds and a slow one with a period of 1 millisecond. The fast thread doesn't support floating point functions.

As before, we can use *halcmd show* to take a look at the HAL. This time we have a lot more pins and parameters than before:

```
halcmd: show pin

Component Pins:
Owner   Type  Dir         Value  Name
     4  float IN              1  siggen.0.amplitude
     4  bit   OUT         FALSE  siggen.0.clock
     4  float OUT             0  siggen.0.cosine
     4  float IN              1  siggen.0.frequency
     4  float IN              0  siggen.0.offset
     4  float OUT             0  siggen.0.sawtooth
     4  float OUT             0  siggen.0.sine
     4  float OUT             0  siggen.0.square
     4  float OUT             0  siggen.0.triangle
     3  s32   OUT             0  stepgen.0.counts
     3  bit   OUT         FALSE  stepgen.0.dir
     3  bit   IN          FALSE  stepgen.0.enable
     3  float OUT             0  stepgen.0.position-fb
     3  bit   OUT         FALSE  stepgen.0.step
     3  float IN              0  stepgen.0.velocity-cmd
     3  s32   OUT             0  stepgen.1.counts
     3  bit   OUT         FALSE  stepgen.1.dir
     3  bit   IN          FALSE  stepgen.1.enable
     3  float OUT             0  stepgen.1.position-fb
     3  bit   OUT         FALSE  stepgen.1.step
     3  float IN              0  stepgen.1.velocity-cmd
```

```
halcmd: show param

Parameters:
Owner   Type  Dir         Value  Name
     4  s32   RO              0  siggen.0.update.time
     4  s32   RW              0  siggen.0.update.tmax
     3  u32   RW     0x00000001  stepgen.0.dirhold
     3  u32   RW     0x00000001  stepgen.0.dirsetup
     3  float RO              0  stepgen.0.frequency
     3  float RW              0  stepgen.0.maxaccel
     3  float RW              0  stepgen.0.maxvel
     3  float RW              1  stepgen.0.position-scale
     3  s32   RO              0  stepgen.0.rawcounts
     3  u32   RW     0x00000001  stepgen.0.steplen
     3  u32   RW     0x00000001  stepgen.0.stepspace
     3  u32   RW     0x00000001  stepgen.1.dirhold
     3  u32   RW     0x00000001  stepgen.1.dirsetup
     3  float RO              0  stepgen.1.frequency
     3  float RW              0  stepgen.1.maxaccel
     3  float RW              0  stepgen.1.maxvel
     3  float RW              1  stepgen.1.position-scale
     3  s32   RO              0  stepgen.1.rawcounts
     3  u32   RW     0x00000001  stepgen.1.steplen
     3  u32   RW     0x00000001  stepgen.1.stepspace
     3  s32   RO              0  stepgen.capture-position.time
     3  s32   RW              0  stepgen.capture-position.tmax
     3  s32   RO              0  stepgen.make-pulses.time
     3  s32   RW              0  stepgen.make-pulses.tmax
     3  s32   RO              0  stepgen.update-freq.time
     3  s32   RW              0  stepgen.update-freq.tmax
```

#### 12.3.4.2 Connecting pins with signals

What we have is two step pulse generators, and a signal generator. Now it is time to create some HAL signals to connect the two components. We are going to pretend that the two step pulse generators are driving the X and Y axis of a machine. We want to move the table in circles. To do this, we will send a cosine signal to the X axis, and a sine signal to the Y axis. The siggen module creates the sine and cosine, but we need *wires* to connect the modules together. In the HAL, *wires* are called signals. We need to create two of them. We can call them anything we want, for this example they will be *X-vel* and *Y-vel*. The signal *X-vel* is intended to run from the cosine output of the signal generator to the velocity input of the first step pulse generator. The first step is to connect the signal to the signal generator output. To connect a signal to a pin we use the net command.

**net command**

```
halcmd: net X-vel <= siggen.0.cosine
```

To see the effect of the *net* command, we show the signals again.

```
halcmd: show sig

Signals:
Type          Value  Name     (linked to)
float             0  X-vel <== siggen.0.cosine
```

When a signal is connected to one or more pins, the show command lists the pins immediately following the signal name. The *arrow* shows the direction of data flow - in this case, data flows from pin *siggen.0.cosine* to signal *X-vel*. Now let's connect the *X-vel* to the velocity input of a step pulse generator.

```
halcmd: net X-vel => stepgen.0.velocity-cmd
```

We can also connect up the Y axis signal *Y-vel*. It is intended to run from the sine output of the signal generator to the input of the second step pulse generator. The following command accomplishes in one line what two *net* commands accomplished for *X-vel*.

```
halcmd: net Y-vel siggen.0.sine => stepgen.1.velocity-cmd
```

Now let's take a final look at the signals and the pins connected to them.

```
halcmd: show sig

Signals:
Type          Value  Name     (linked to)
float             0  X-vel <== siggen.0.cosine
                            ==> stepgen.0.velocity-cmd
float             0  Y-vel <== siggen.0.sine
                            ==> stepgen.1.velocity-cmd
```

The *show sig* command makes it clear exactly how data flows through the HAL. For example, the *X-vel* signal comes from pin *siggen.0.cosine*, and goes to pin *stepgen.0.velocity-cmd*.

#### 12.3.4.3 Setting up realtime execution - threads and functions

Thinking about data flowing through *wires* makes pins and signals fairly easy to understand. Threads and functions are a little more difficult. Functions contain the computer instructions that actually get things done. Thread are the method used to make those instructions run when they are needed. First let's look at the functions available to us.

```
halcmd: show funct

Exported Functions:
Owner   CodeAddr  Arg       FP   Users  Name
 00004  f9992000  fc731278  YES      0   siggen.0.update
 00003  f998b20f  fc7310b8  YES      0   stepgen.capture-position
 00003  f998b000  fc7310b8  NO       0   stepgen.make-pulses
 00003  f998b307  fc7310b8  YES      0   stepgen.update-freq
```

In general, you will have to refer to the documentation for each component to see what its functions do. In this case, the function *siggen.0.update* is used to update the outputs of the signal generator. Every time it is executed, it calculates the values of the sine, cosine, triangle, and square outputs. To make smooth signals, it needs to run at specific intervals.

The other three functions are related to the step pulse generators.

The first one, *stepgen.capture_position*, is used for position feedback. It captures the value of an internal counter that counts the step pulses as they are generated. Assuming no missed steps, this counter indicates the position of the motor.

The main function for the step pulse generator is *stepgen.make_pulses*. Every time *make_pulses* runs it decides if it is time to take a step, and if so sets the outputs accordingly. For smooth step pulses, it should run as frequently as possible. Because it needs to run so fast, *make_pulses* is highly optimized and performs only a few calculations. Unlike the others, it does not need floating point math.

The last function, *stepgen.update-freq*, is responsible for doing scaling and some other calculations that need to be performed only when the frequency command changes.

What this means for our example is that we want to run *siggen.0.update* at a moderate rate to calculate the sine and cosine values. Immediately after we run *siggen.0.update*, we want to run *stepgen.update_freq* to load the new values into the step pulse generator. Finally we need to run *stepgen.make_pulses* as fast as possible for smooth pulses. Because we don't use position feedback, we don't need to run *stepgen.capture_position* at all.

We run functions by adding them to threads. Each thread runs at a specific rate. Let's see what threads we have available.

```
halcmd: show thread

Realtime Threads:
     Period  FP     Name               (     Time, Max-Time )
     996980  YES                  slow (        0,        0 )
      49849  NO                   fast (        0,        0 )
```

The two threads were created when we loaded *threads*. The first one, *slow*, runs every millisecond, and is capable of running floating point functions. We will use it for *siggen.0.update* and *stepgen.update_freq*. The second thread is *fast*, which runs every 50 microseconds, and does not support floating point. We will use it for *stepgen.make_pulses*. To connect the functions to the proper thread, we use the *addf* command. We specify the function first, followed by the thread.

```
halcmd: addf siggen.0.update slow
halcmd: addf stepgen.update-freq slow
halcmd: addf stepgen.make-pulses fast
```

After we give these commands, we can run the *show thread* command again to see what happened.

```
halcmd: show thread

Realtime Threads:
     Period  FP     Name               (     Time, Max-Time )
     996980  YES                  slow (        0,        0 )
                  1 siggen.0.update
                  2 stepgen.update-freq
      49849  NO                   fast (        0,        0 )
                  1 stepgen.make-pulses
```

Now each thread is followed by the names of the functions, in the order in which the functions will run.

#### 12.3.4.4 Setting parameters

We are almost ready to start our HAL system. However we still need to adjust a few parameters. By default, the siggen component generates signals that swing from +1 to -1. For our example that is fine, we want the table speed to vary from +1 to -1 inches per second. However the scaling of the step pulse generator isn't quite right. By default, it generates an output frequency of 1 step per second with an input of 1.000. It is unlikely that one step per second will give us one inch per second of table movement. Let's assume instead that we have a 5 turn per inch leadscrew, connected to a 200 step per rev stepper with 10x microstepping. So it takes 2000 steps for one revolution of the screw, and 5 revolutions to travel one inch. that means the overall scaling is 10000 steps per inch. We need to multiply the velocity input to the step pulse generator by 10000 to get the proper output. That is exactly what the parameter *stepgen.n.velocity-scale* is for. In this case, both the X and Y axis have the same scaling, so we set the scaling parameters for both to 10000.

```
halcmd: setp stepgen.0.position-scale 10000
halcmd: setp stepgen.1.position-scale 10000
halcmd: setp stepgen.0.enable 1
halcmd: setp stepgen.1.enable 1
```

This velocity scaling means that when the pin *stepgen.0.velocity-cmd* is 1.000, the step generator will generate 10000 pulses per second (10KHz). With the motor and leadscrew described above, that will result in the axis moving at exactly 1.000 inches per second. This illustrates a key HAL concept - things like scaling are done at the lowest possible level, in this case in the step pulse generator. The internal signal *X-vel* is the velocity of the table in inches per second, and other components such as *siggen* don't know (or care) about the scaling at all. If we changed the leadscrew, or motor, we would change only the scaling parameter of the step pulse generator.

#### 12.3.4.5 Run it!

We now have everything configured and are ready to start it up. Just like in the first example, we use the *start* command.

```
halcmd: start
```

Although nothing appears to happen, inside the computer the step pulse generator is cranking out step pulses, varying from 10KHz forward to 10KHz reverse and back again every second. Later in this tutorial we'll see how to bring those internal signals out to run motors in the real world, but first we want to look at them and see what is happening.

### 12.3.5 Halscope

The previous example generates some very interesting signals. But much of what happens is far too fast to see with halmeter. To take a closer look at what is going on inside the HAL, we want an oscilloscope. Fortunately HAL has one, called halscope.

Halscope has two parts - a realtime part that is loaded as a kernel module, and a user part that supplies the GUI and display. However, you don't need to worry about this, because the userspace portion will automatically request that the realtime part be loaded. Also notice the first time you run halscope in a directory it gives a benign message that the file *autosave.halscope* could not be opened.

**Starting Halscope**

```
halcmd: loadusr halscope
halcmd: halscope: config file 'autosave.halscope' could not be opened
```

The scope GUI window will open, immediately followed by a *Realtime function not linked* dialog that looks like the following figure.

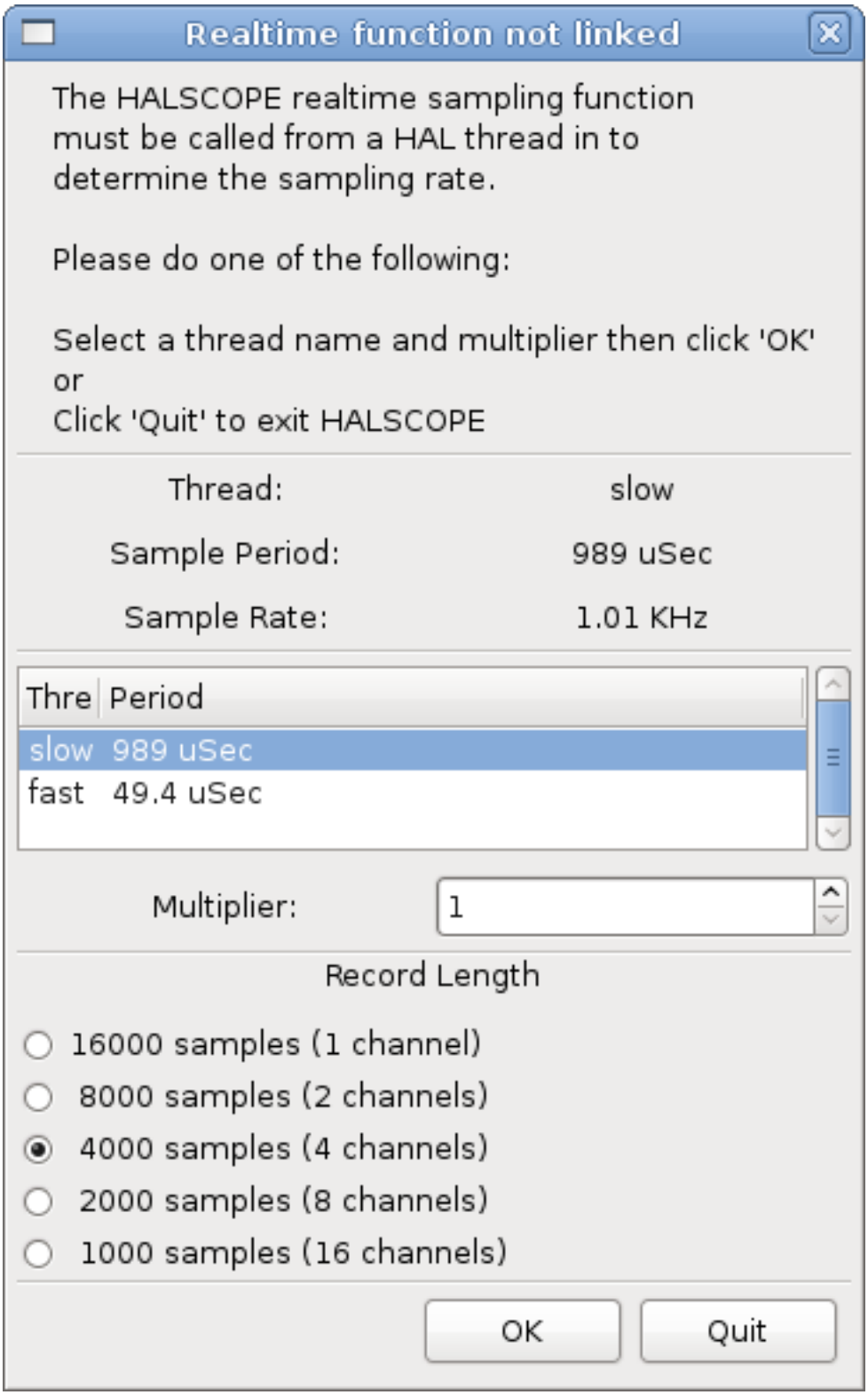



Figure 12.5: Realtime function not linked dialog

This dialog is where you set the sampling rate for the oscilloscope. For now we want to sample once per millisecond, so click on the 989 us thread *slow* and leave the multiplier at 1. We will also leave the record length at 4000 samples, so that we can use up to four channels at one time. When you select a thread and then click *OK*, the dialog disappears, and the scope window looks

something like the following figure.

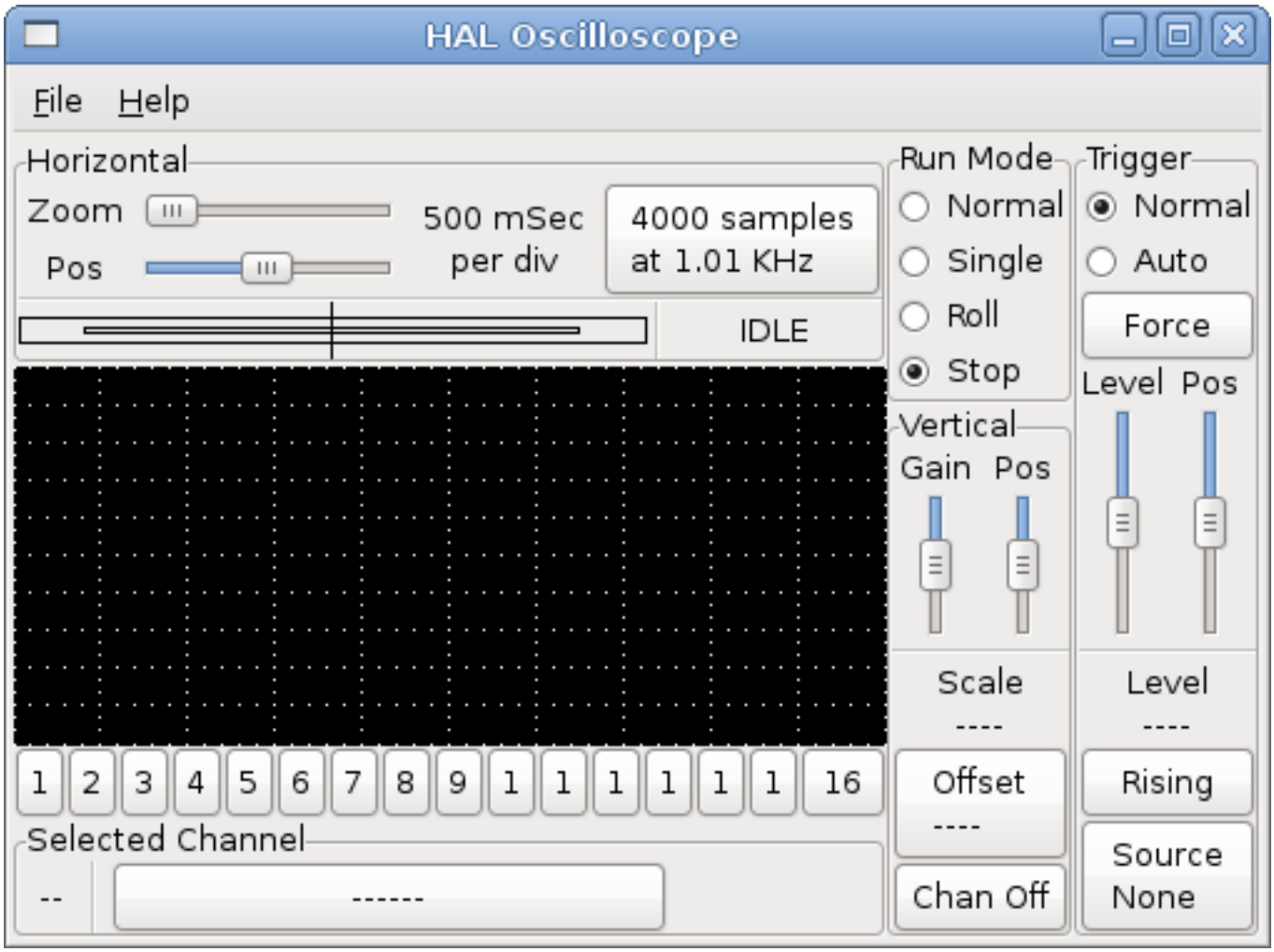



Figure 12.6: Initial scope window

#### 12.3.5.1 Hooking up the scope probes

At this point, Halscope is ready to use. We have already selected a sample rate and record length, so the next step is to decide what to look at. This is equivalent to hooking *virtual scope probes* to the HAL. Halscope has 16 channels, but the number you can use at any one time depends on the record length - more channels means shorter records, since the memory available for the record is fixed at approximately 16,000 samples.

The channel buttons run across the bottom of the halscope screen. Click button *1*, and you will see the *Select Channel Source* dialog as shown in the following figure. This dialog is very similar to the one used by Halmeter. We would like to look at the signals we defined earlier, so we click on the *Signals* tab, and the dialog displays all of the signals in the HAL (only two for this example).

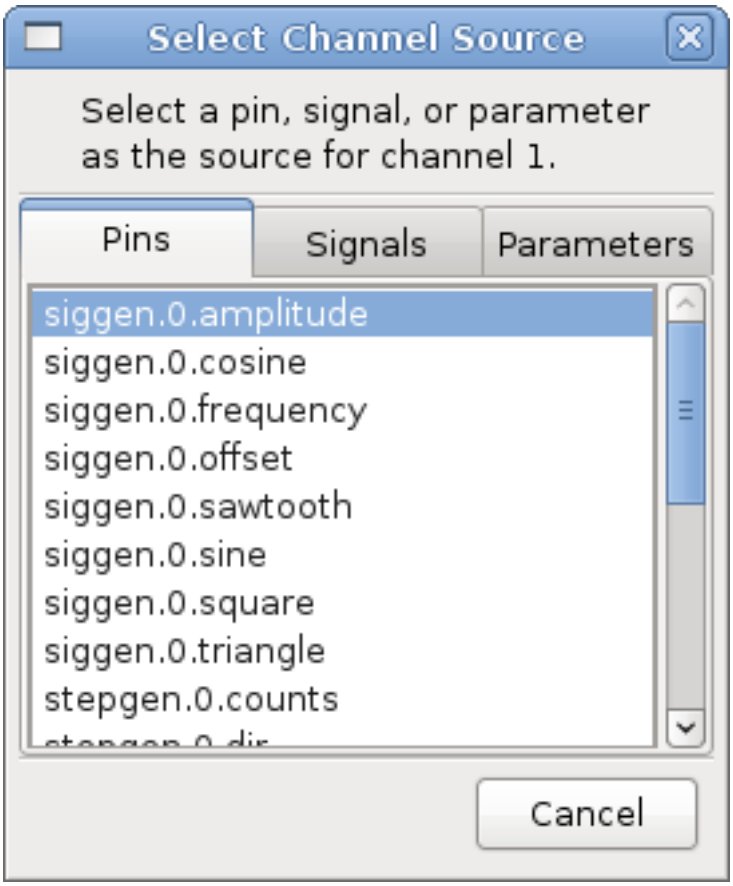



Figure 12.7: Select Channel Source

To choose a signal, just click on it. In this case, we want channel 1 to display the signal *X-vel*. Click on the Signals tab then click on *X-vel* and the dialog closes and the channel is now selected.

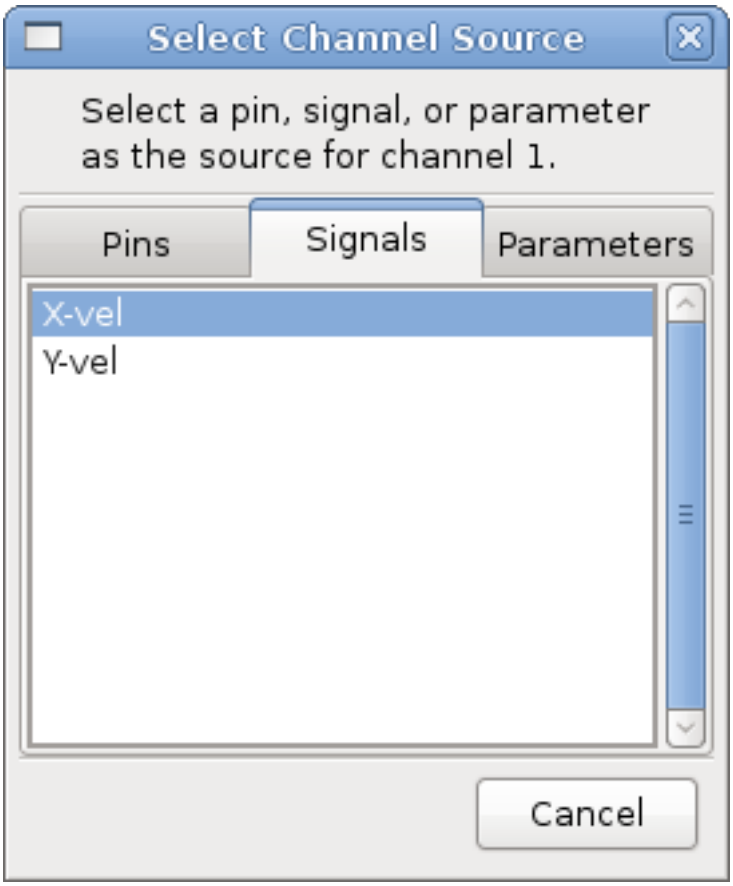



Figure 12.8: Select Signal

The channel 1 button is pressed in, and channel number 1 and the name *X-vel* appear below the row of buttons. That display always indicates the selected channel - you can have many channels on the screen, but the selected one is highlighted, and the various controls like vertical position and scale always work on the selected one.

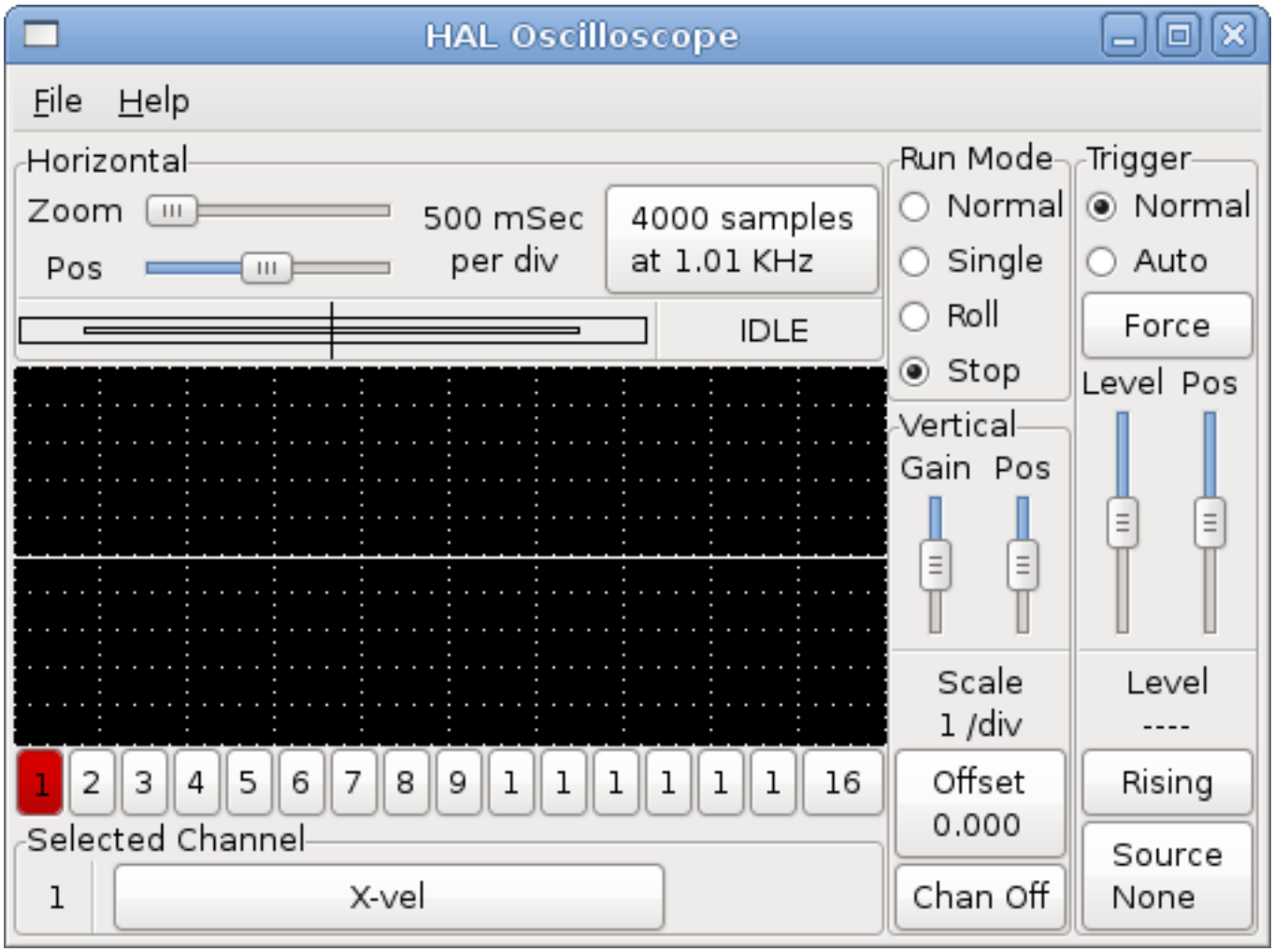



Figure 12.9: Halscope

To add a signal to channel 2, click the *2* button. When the dialog pops up, click the *Signals* tab, then click on *Y-vel*. We also want to look at the square and triangle wave outputs. There are no signals connected to those pins, so we use the *Pins* tab instead. For channel 3, select *siggen.0.triangle* and for channel 4, select *siggen.0.square*.

#### 12.3.5.2 Capturing our first waveforms

Now that we have several probes hooked to the HAL, it's time to capture some waveforms. To start the scope, click the *Normal* button in the *Run Mode* section of the screen (upper right). Since we have a 4000 sample record length, and are acquiring 1000 samples per second, it will take halscope about 2 seconds to fill half of its buffer. During that time a progress bar just above the main screen will show the buffer filling. Once the buffer is half full, the scope waits for a trigger. Since we haven't configured one yet, it will wait forever. To manually trigger it, click the *Force* button in the *Trigger* section at the top right. You should see the remainder of the buffer fill, then the screen will display the captured waveforms. The result will look something like the following figure.

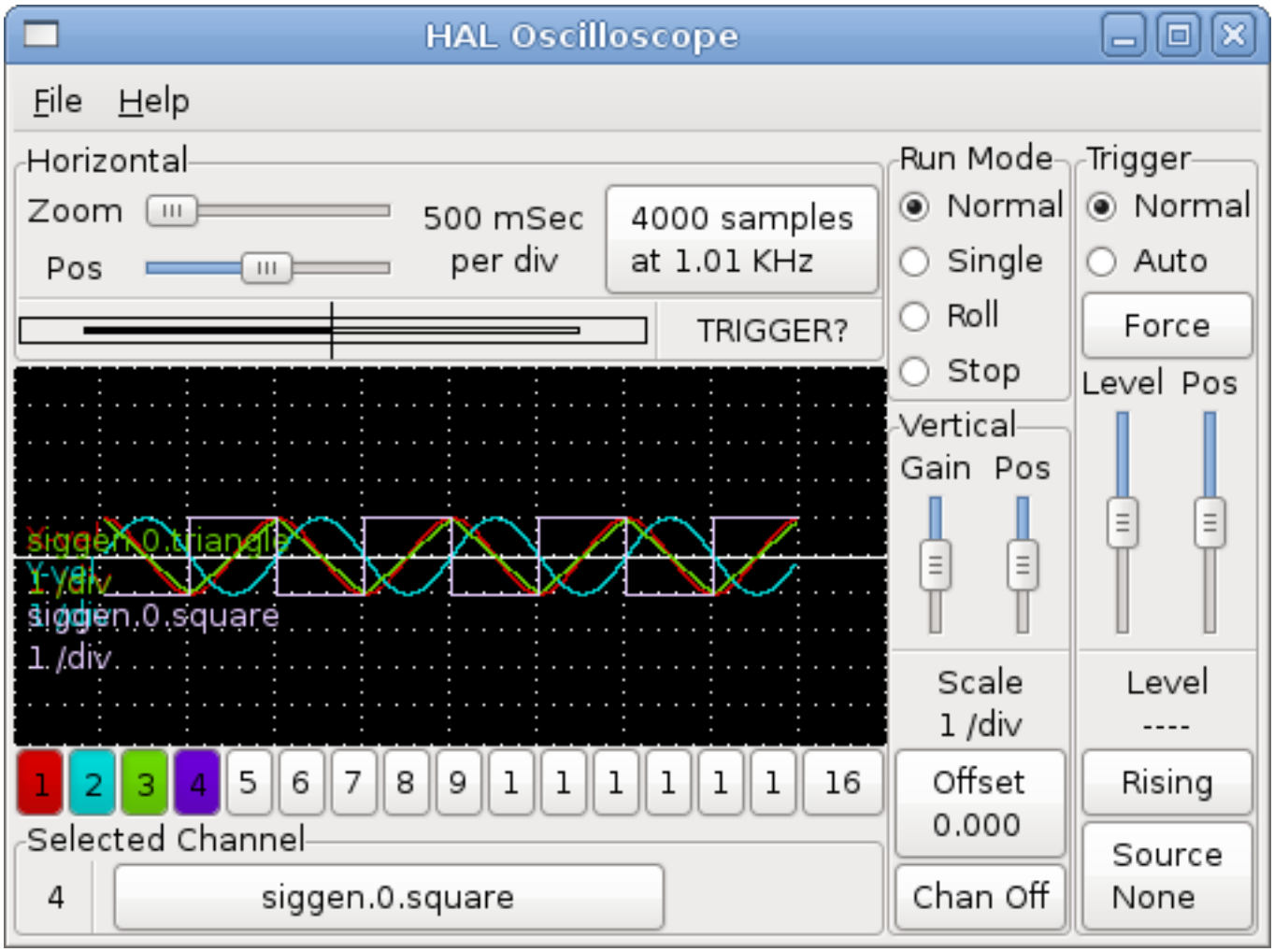



Figure 12.10: Captured Waveforms

The *Selected Channel* box at the bottom tells you that the purple trace is the currently selected one, channel 4, which is displaying the value of the pin *siggen.0.square*. Try clicking channel buttons 1 through 3 to highlight the other three traces.

#### 12.3.5.3 Vertical Adjustments

The traces are rather hard to distinguish since all four are on top of each other. To fix this, we use the *Vertical* controls in the box to the right of the screen. These controls act on the currently selected channel. When adjusting the gain, notice that it covers a huge range - unlike a real scope, this one can display signals ranging from very tiny (pico-units) to very large (Tera-units). The position control moves the displayed trace up and down over the height of the screen only. For larger adjustments the offset button should be used.

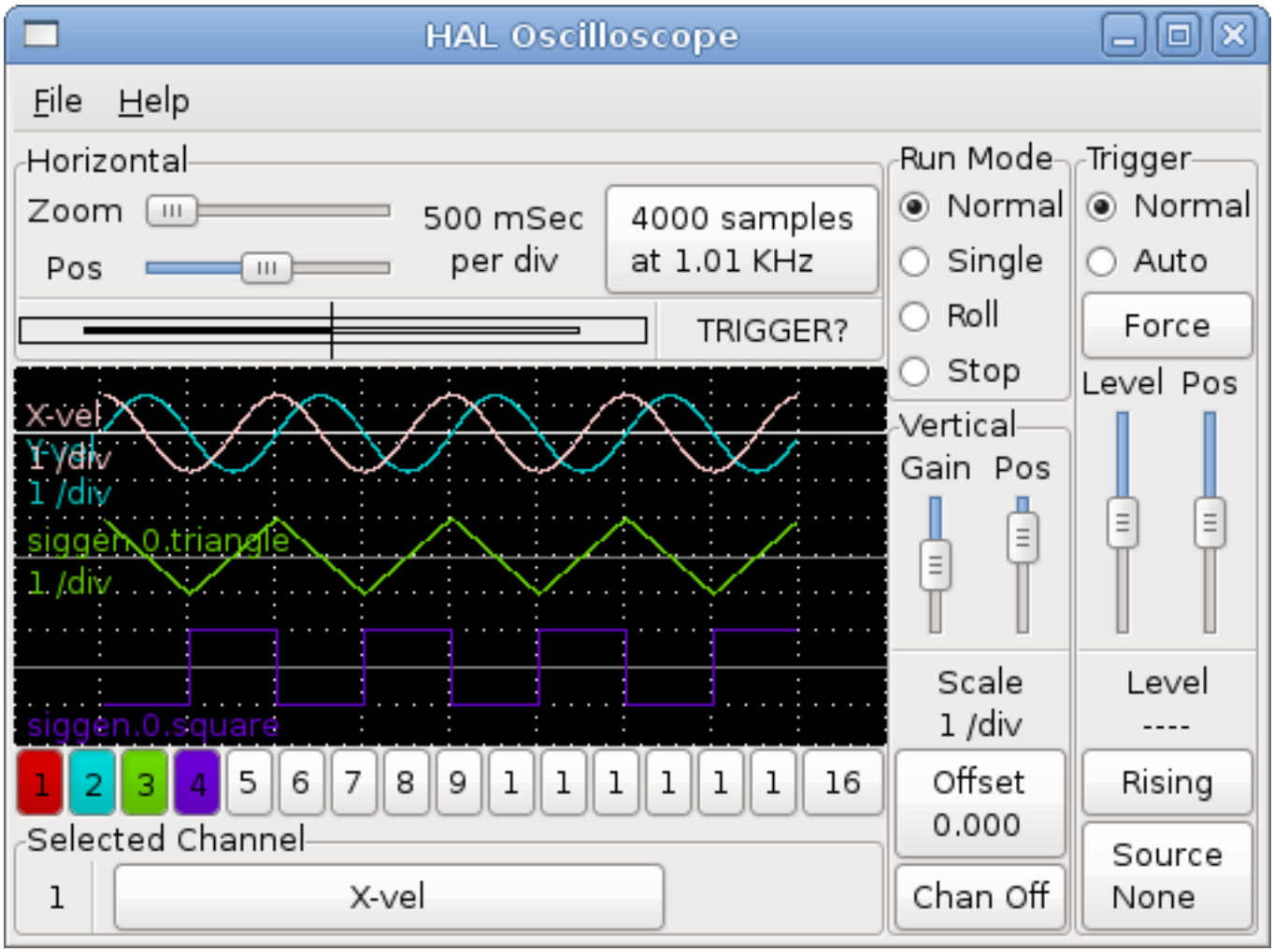



Figure 12.11: Vertical Adjustment

#### 12.3.5.4 Triggering

Using the *Force* button is a rather unsatisfying way to trigger the scope. To set up real triggering, click on the *Source* button at the bottom right. It will pop up the *Trigger Source* dialog, which is simply a list of all the probes that are currently connected. Select a probe to use for triggering by clicking on it. For this example we will use channel 3, the triangle wave as shown in the following figure.

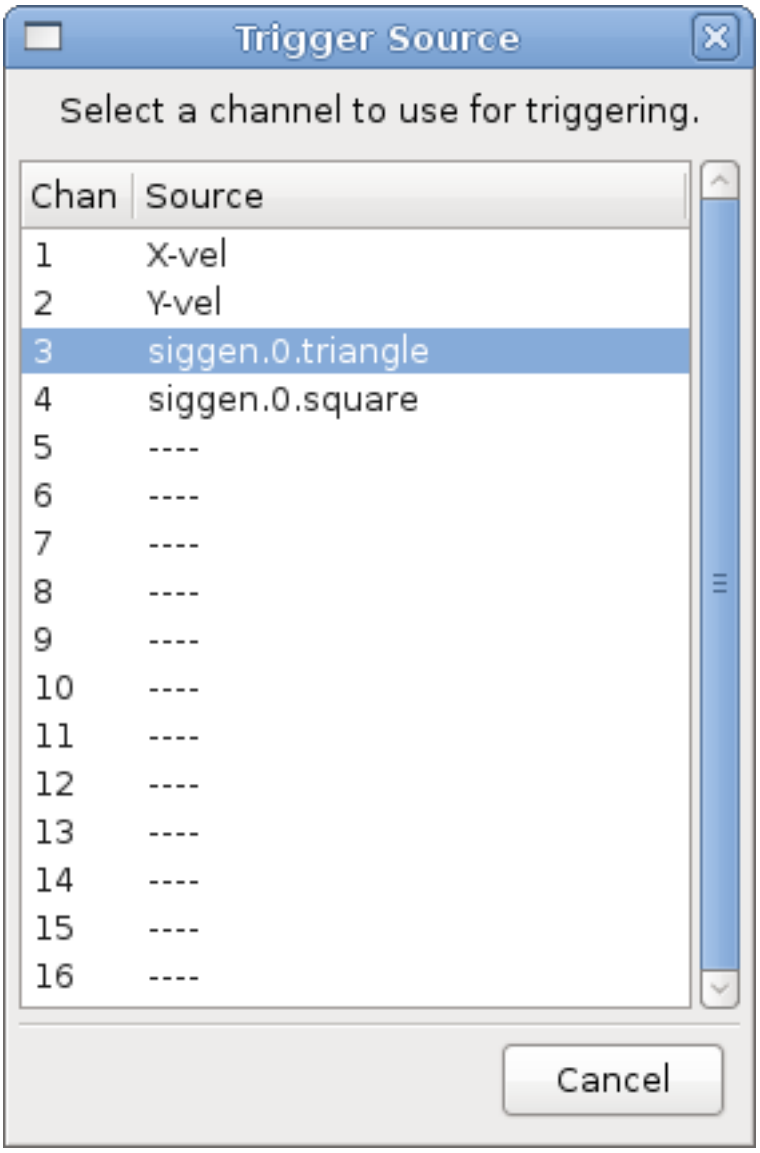



Figure 12.12: Trigger Source Dialog

After setting the trigger source, you can adjust the trigger level and trigger position using the sliders in the *Trigger* box along the right edge. The level can be adjusted from the top to the bottom of the screen, and is displayed below the sliders. The position is the location of the trigger point within the overall record. With the slider all the way down, the trigger point is at the end of the record, and halscope displays what happened before the trigger point. When the slider is all the way up, the trigger point is at the beginning of the record, displaying what happened after it was triggered. The trigger point is visible as a vertical line in the progress box above the screen. The trigger polarity can be changed by clicking the button just below the trigger level display.

Now that we have adjusted the vertical controls and triggering, the scope display looks something like the following figure.

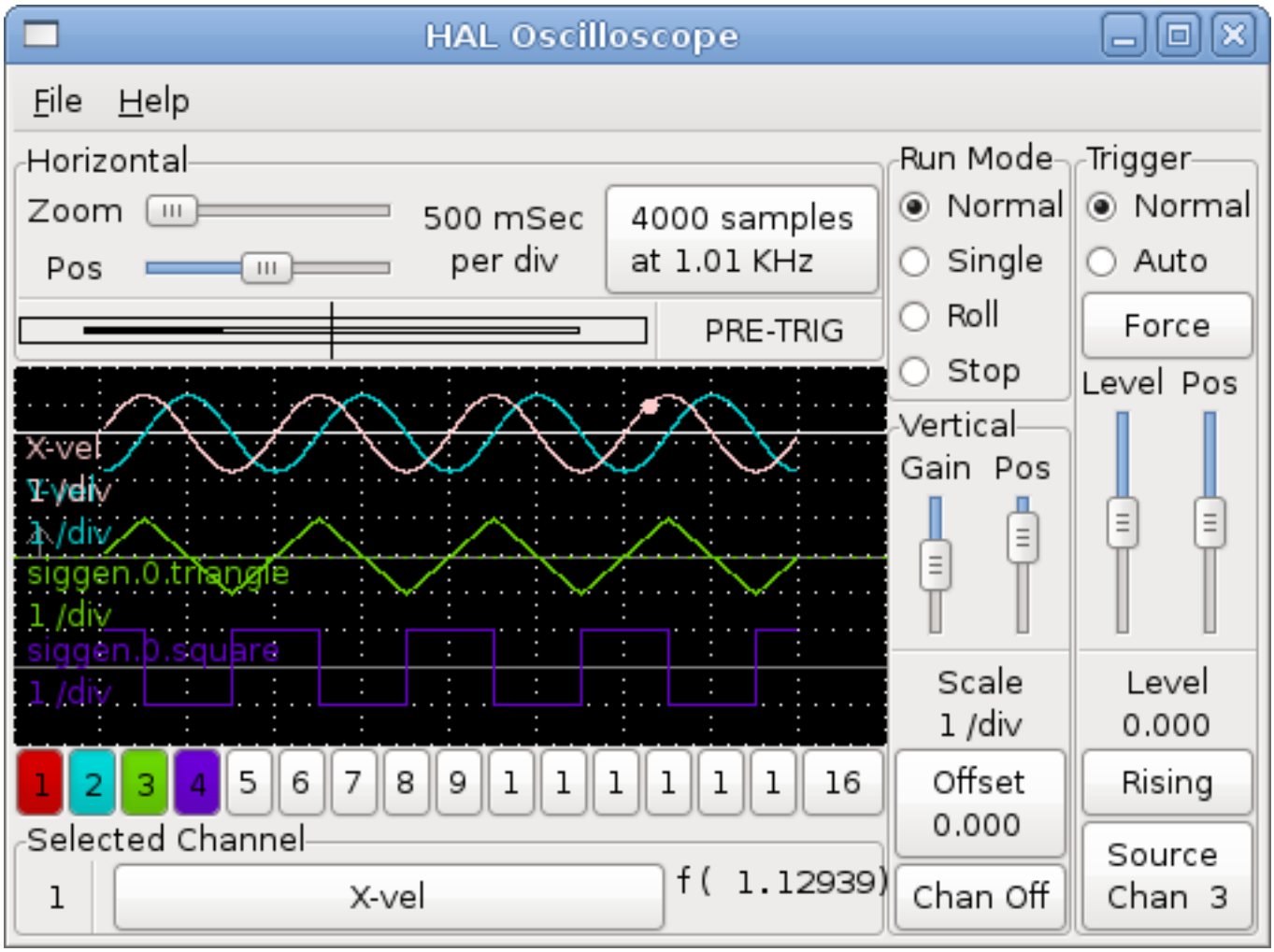



Figure 12.13: Waveforms with Triggering

#### 12.3.5.5 Horizontal Adjustments

To look closely at part of a waveform, you can use the zoom slider at the top of the screen to expand the waveforms horizontally, and the position slider to determine which part of the zoomed waveform is visible. However, sometimes simply expanding the waveforms isn't enough and you need to increase the sampling rate. For example, we would like to look at the actual step pulses that are being generated in our example. Since the step pulses may be only 50 us long, sampling at 1KHz isn't fast enough. To change the sample rate, click on the button that displays the number of samples and sample rate to bring up the *Select Sample Rate* dialog, figure . For this example, we will click on the 50 us thread, *fast*, which gives us a sample rate of about 20KHz. Now instead of displaying about 4 seconds worth of data, one record is 4000 samples at 20KHz, or about 0.20 seconds.

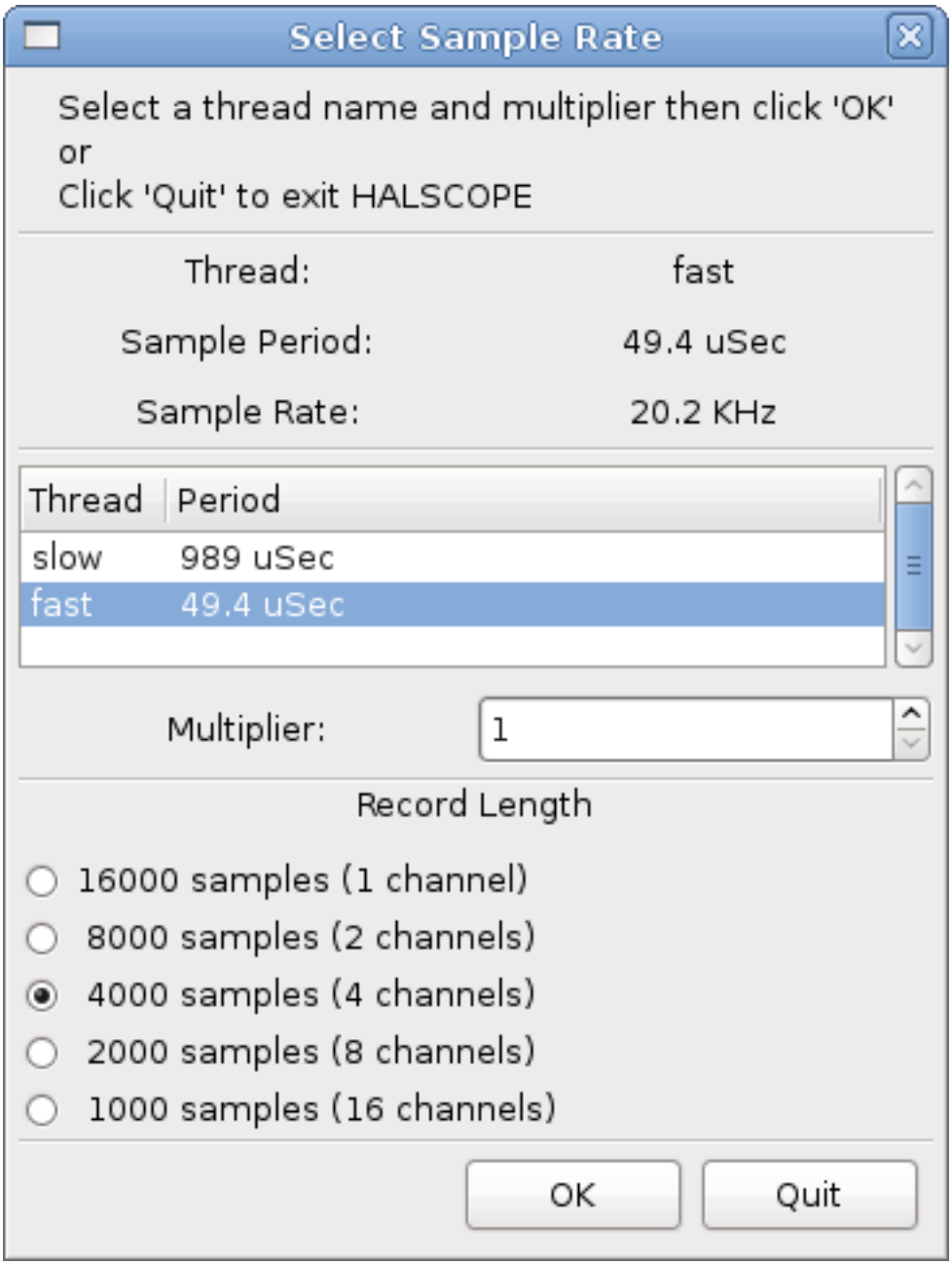



Figure 12.14: Sample Rate Dialog

#### 12.3.5.6 More Channels

Now let's look at the step pulses. Halscope has 16 channels, but for this example we are using only 4 at a time. Before we select any more channels, we need to turn off a couple. Click on the channel 2 button, then click the *Chan Off* button at the bottom of the *Vertical* box. Then click on channel 3, turn if off, and do the same for channel 4. Even though the channels are turned off, they still remember what they are connected to, and in fact we will continue to use channel 3 as the trigger source. To add new channels, select channel 5, and choose pin *stepgen.0.dir*, then channel 6, and select *stepgen.0.step*. Then click run mode *Normal* to start the scope, and adjust the horizontal zoom to 5 ms per division. You should see the step pulses slow down as the velocity command (channel 1) approaches zero, then the direction pin changes state and the step pulses speed up again. You might want to increase the gain on channel 1 to about 20 milli per division to better see the change in the velocity command. The result should look like the following figure.

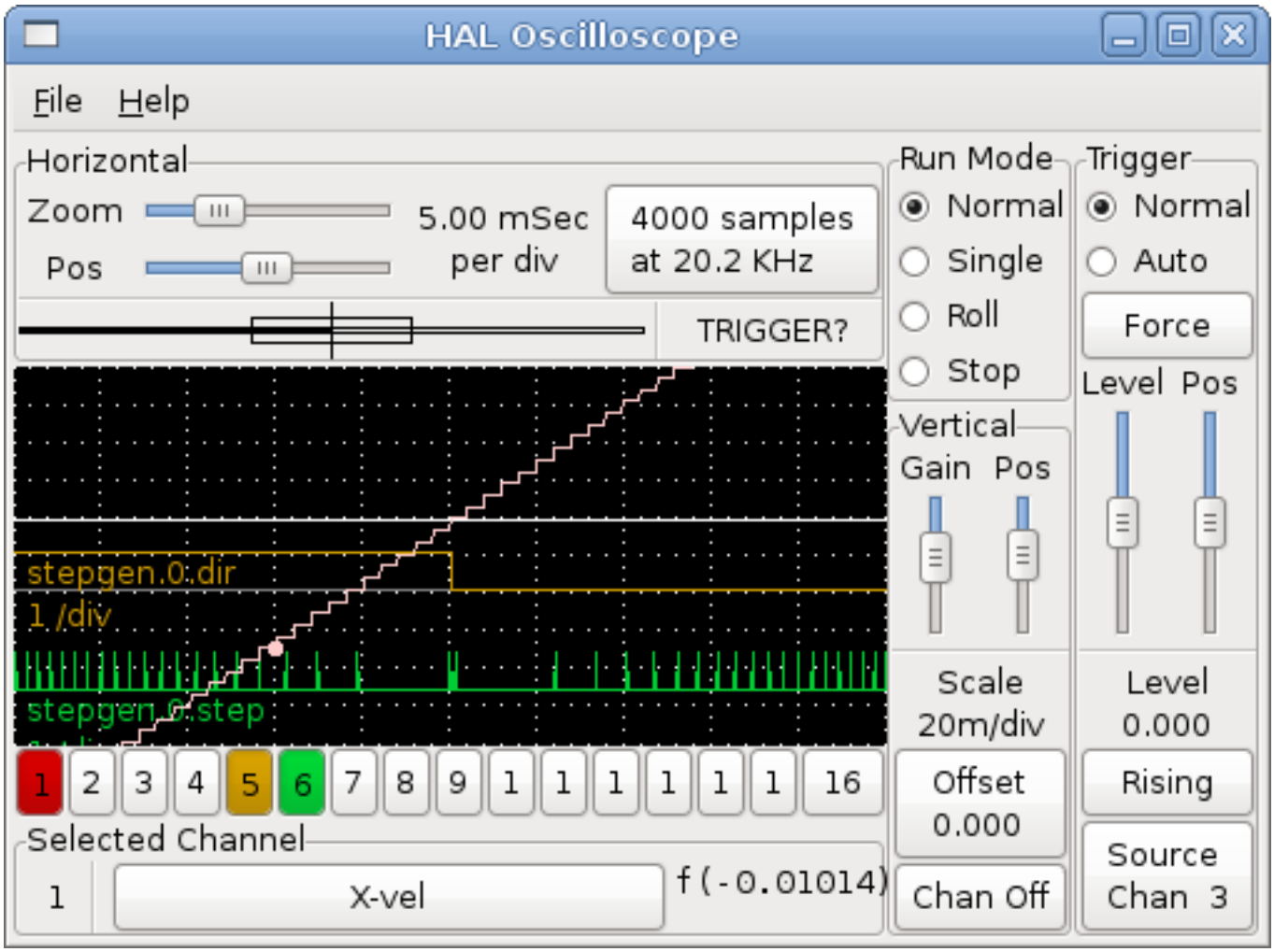



Figure 12.15: Step Pulses

#### 12.3.5.7 More samples

If you want to record more samples at once, restart realtime and load halscope with a numeric argument which indicates the number of samples you want to capture.

```
halcmd: loadusr halscope 80000
```

If the *scope_rt* component was not already loaded, halscope will load it and request 80000 total samples, so that when sampling 4 channels at a time there will be 20000 samples per channel. (If *scope_rt* was already loaded, the numeric argument to halscope will have no effect).

## 12.4 General Reference

### 12.4.1 General Naming Conventions

Consistent naming conventions would make HAL much easier to use. For example, if every encoder driver provided the same set of pins and named them the same way it would be easy to change from one type of encoder driver to another. Unfortunately, like many open-source projects, HAL is a combination of things that were designed, and things that simply evolved. As a result,

there are many inconsistencies. This section attempts to address that problem by defining some conventions, but it will probably be a while before all the modules are converted to follow them.

Halcmd and other low-level HAL utilities treat HAL names as single entities, with no internal structure. However, most modules do have some implicit structure. For example, a board provides several functional blocks, each block might have several channels, and each channel has one or more pins. This results in a structure that resembles a directory tree. Even though halcmd doesn't recognize the tree structure, proper choice of naming conventions will let it group related items together (since it sorts the names). In addition, higher level tools can be designed to recognize such structure, if the names provide the necessary information. To do that, all HAL components should follow these rules:

- Dots (".") separate levels of the hirearchy. This is analogous to the slash ("/") in a filename.
- Hyphens ("-") separate words or fields in the same level of the hirearchy.
- HAL components should not use underscores or "MixedCase".
- Use only lowercase letters and numbers in names.

### 12.4.2 Hardware Driver Naming Conventions

#### 12.4.2.1 Pin/Parameter names

Hardware drivers should use five fields (on three levels) to make up a pin or parameter name, as follows:

**`<device-name>.<device-num>.<io-type>.<chan-num>.<specific-name>`**

The individual fields are:

**<device-name>**
: The device that the driver is intended to work with. This is most often an interface board of some type, but there are other possibilities.

**<device-num>**
: It is possible to install more than one servo board, parallel port, or other hardware device in a computer. The device number identifies a specific device. Device numbers start at 0 and increment.

**<io-type>**
: Most devices provide more than one type of I/O. Even the simple parallel port has both digital inputs and digital outputs. More complex boards can have digital inputs and outputs, encoder counters, pwm or step pulse generators, analog-to-digital converters, digital-to-analog converters, or other unique capabilities. The I/O type is used to identify the kind of I/O that a pin or parameter is associated with. Ideally, drivers that implement the same I/O type, even if for very different devices, should provide a consistent set of pins and parameters and identical behavior. For example, all digital inputs should behave the same when seen from inside the HAL, regardless of the device.

**<chan-num>**
: Virtually every I/O device has multiple channels, and the channel number identifies one of them. Like device numbers, channel numbers start at zero and increment.[3] If more than one device is installed, the channel numbers on additional devices start over at zero. If it is possible to have a channel number greater than 9, then channel numbers should be two digits, with a leading zero on numbers less than 10 to preserve sort ordering. Some modules have pins and/or parameters that affect more than one channel. For example a PWM generator might have four channels with four independent "duty-cycle" inputs, but one "frequency" parameter that controls all four channels (due to hardware limitations). The frequency parameter should use "0-3" as the channel number.

**<specific-name>**
: An individual I/O channel might have just a single HAL pin associated with it, but most have more than one. For example, a digital input has two pins, one is the state of the physical pin, the other is the same thing inverted. That allows the configurator to choose between active high and active low inputs. For most io-types, there is a standard set of pins and parameters, (referred to as the "canonical interface") that the driver should implement. The canonical interfaces are described in the Canonical Device Interfaces chapter.

[3] One exception to the "channel numbers start at zero" rule is the parallel port. Its HAL pins are numbered with the corresponding pin number on the DB-25 connector. This is convenient for wiring, but inconsistent with other drivers. There is some debate over whether this is a bug or a feature.

EXAMPLES

**motenc.0.encoder.2.position**
— the position output of the third encoder channel on the first Motenc board.

**stg.0.din.03.in**
— the state of the fourth digital input on the first Servo-to-Go board.

**ppmc.0.pwm.00-03.frequency**
— the carrier frequency used for PWM channels 0 through 3 on the first Pico Systems ppmc board.

#### 12.4.2.2 Function Names

Hardware drivers usually only have two kinds of HAL functions, ones that read the hardware and update HAL pins, and ones that write to the hardware using data from HAL pins. They should be named as follows:

```
<device-name>-<device-num>.<io-type>-<chan-num-range>.read|write
```

**<device-name>**
The same as used for pins and parameters.

**<device-num>**
The specific device that the function will access.

**<io-type>**
Optional. A function may access all of the I/O on a board, or it may access only a certain type. For example, there may be independent functions for reading encoder counters and reading digital I/O. If such independent functions exist, the <io-type> field identifies the type of I/O they access. If a single function reads all I/O provided by the board, <io-type> is not used. [4]

**<chan-num-range>**
Optional. Used only if the <io-type> I/O is broken into groups and accessed by different functions.

**read|write**
Indicates whether the function reads the hardware or writes to it.

EXAMPLES

**motenc.0.encoder.read**
— reads all encoders on the first motenc board.

**generic8255.0.din.09-15.read**
— reads the second 8 bit port on the first generic 8255 based digital I/O board.

**ppmc.0.write**
— writes all outputs (step generators, pwm, DACs, and digital) on the first Pico Systems ppmc board.

## 12.5 Core Components

See also the man pages *motion(9)*.

[4] Note to driver programmers: do NOT implement separate functions for different I/O types unless they are interruptible and can work in independent threads. If interrupting an encoder read, reading digital inputs, and then resuming the encoder read will cause problems, then implement a single function that does everything.

### 12.5.1 Motion

These pins and parameters are created by the realtime *motmod* module. This module provides a HAL interface for LinuxCNC's motion planner. Basically motmod takes in a list of waypoints and generates a nice blended and constraint-limited stream of joint positions to be fed to the motor drives.

Optionally the number of Digital I/O is set with num_dio. The number of Analog I/O is set with num_aio. The default is 4 each.

Pin names starting with *axis* are actually joint values, but the pins and parameters are still called *axis.N*. They are read and updated by the motion-controller function.

Motion is loaded with the motmod command. A kins should be loaded before motion.

```
loadrt motmod [base_period_nsec=period] [servo_period_nsec=period]
[traj_period_nsec=period] [num_joints=[0-9] ([num_dio=1-64] num_aio=1-16])]
```

- *base_period_nsec = 50000* - the *Base* task period in nanoseconds. This is the fastest thread in the machine.

  **Note**
  On servo-based systems, there is generally no reason for *base_period_nsec* to be smaller than *servo_period_nsec*. On machines with software step generation, the *base_period_nsec* determines the maximum number of steps per second. In the absence of long step length and step space requirements, the absolute maximum step rate is one step per *base_period_nsec*. Thus, the *base_period_nsec* shown above gives an absolute maximum step rate of 20,000 steps per second. 50,000 ns (50 us) is a fairly conservative value. The smallest usable value is related to the Latency Test result, the necessary step length, and the processor speed. Choosing a *base_period_nsec* that is too low can lead to the "Unexpected real time delay" message, lockups, or spontaneous reboots.

- *servo_period_nsec = 1000000* - This is the *Servo* task period in nanoseconds. This value will be rounded to an integer multiple of *base_period_nsec*. This period is used even on systems based on stepper motors.

  This is the rate at which new motor positions are computed, following error is checked, PID output values are updated, and so on. Most systems will not need to change this value. It is the update rate of the low level motion planner.

- *traj_period_nsec = 100000* - This is the *Trajectory Planner* task period in nanoseconds. This value will be rounded to an integer multiple of *servo_period_nsec*. Except for machines with unusual kinematics (e.g., hexapods) there is no reason to make this value larger than *servo_period_nsec*.

#### 12.5.1.1 Options

If the number of digital I/O needed is more than the default of 4 you can add up to 64 digital I/O by using the num_dio option when loading motmod.

If the number of analog I/O needed is more than the default of 4 you can add up to 16 analog I/O by using the num_aio option when loading motmod.

#### 12.5.1.2 Pins

These pins, parameters, and functions are created by the realtime *motmod* module.

- *motion.adaptive-feed* - (float, in) When adaptive feed is enabled with *M52 P1* , the commanded velocity is multiplied by this value. This effect is multiplicative with the NML-level feed override value and *motion.feed-hold*.
- *motion.analog-in-00* - (float, in) These pins (00, 01, 02, 03 or more if configured) are controlled by M66.
- *motion.analog-out-00* - (float, out) These pins (00, 01, 02, 03 or more if configured) are controlled by M67 or M68.
- *motion.coord-error* - (bit, out) TRUE when motion has encountered an error, such as exceeding a soft limit

- *motion.coord-mode* - (bit, out) TRUE when motion is in *coordinated mode*, as opposed to *teleop mode*
- *motion.current-vel* - (float, out) The current tool velocity in user units per second.
- *motion.digital-in-00* - (bit, in) These pins (00, 01, 02, 03 or more if configured) are controlled by M62-65.
- *motion.digital-out-00* - (bit, out) These pins (00, 01, 02, 03 or more if configured) are controlled by the *M62-65*.
- *motion.distance-to-go* - (float,out) The distance remaining in the current move.
- *motion.enable* - (bit, in) If this bit is driven FALSE, motion stops, the machine is placed in the *machine off* state, and a message is displayed for the operator. For normal motion, drive this bit TRUE.
- *motion.feed-hold* - (bit, in) When Feed Stop Control is enabled with *M53 P1*, and this bit is TRUE, the feed rate is set to 0.
- *motion.feed-inhibit* - (bit, in) When this bit is TRUE, the feed rate is set to 0. This will be delayed during spindle synch moves till the end of the move.
- *motion.in-position* - (bit, out) TRUE if the machine is in position.
- *motion.motion-enabled* - (bit, out) TRUE when in *machine on* state.
- *motion.on-soft-limit* - (bit, out) TRUE when the machine is on a soft limit.
- *motion.probe-input* - (bit, in) *G38.x* uses the value on this pin to determine when the probe has made contact. TRUE for probe contact closed (touching), FALSE for probe contact open.
- *motion.program-line* - (s32, out) The current program line while executing. Zero if not running or between lines while single stepping.
- *motion.requested-vel* - (float, out) The current requested velocity in user units per second from the F=n setting in the G Code file. No feed overrides or any other adjustments are applied to this pin.
- *motion.spindle-at-speed* - (bit, in) Motion will pause until this pin is TRUE, under the following conditions: before the first feed move after each spindle start or speed change; before the start of every chain of spindle-synchronized moves; and if in CSS mode, at every rapid to feed transition. This input can be used to ensure that the spindle is up to speed before starting a cut, or that a lathe spindle in CSS mode has slowed down after a large to small facing pass before starting the next pass at the large diameter. Many VFDs have an *at speed* output. Otherwise, it is easy to generate this signal with the *HAL near* component, by comparing requested and actual spindle speeds.
- *motion.spindle-brake* - (bit, out) TRUE when the spindle brake should be applied.
- *motion.spindle-forward* - (bit, out) TRUE when the spindle should rotate forward.
- *motion.spindle-index-enable* - (bit, I/O) For correct operation of spindle synchronized moves, this pin must be hooked to the index-enable pin of the spindle encoder.
- *motion.spindle-inhibit* - (bit, in) When this bit is TRUE, the spindle speed is set to 0.
- *motion.spindle-on* - (bit, out) TRUE when spindle should rotate.
- *motion.spindle-reverse* - (bit, out) TRUE when the spindle should rotate backward
- *motion.spindle-revs* - (float, in) For correct operation of spindle synchronized moves, this signal must be hooked to the position pin of the spindle encoder. The spindle encoder position should be scaled such that spindle-revs increases by 1.0 for each rotation of the spindle in the clockwise (*M3*) direction.
- *motion.spindle-speed-in* - (float, in) Feedback of actual spindle speed in rotations per second. This is used by feed-per-revolution motion (*G95*). If your spindle encoder driver does not have a velocity output, you can generate a suitable one by sending the spindle position through a *ddt* component. If you do not have a spindle encoder, you can loop back *motion.spindle-speed-out-rps*.
- *motion.spindle-speed-out* - (float, out) Commanded spindle speed in rotations per minute. Positive for spindle forward (*M3*), negative for spindle reverse (*M4*).

- *motion.spindle-speed-out-abs* - (float, out) Commanded spindle speed in rotations per minute. This will always be a positive number.
- *motion.spindle-speed-out-rps* - (float, out) Commanded spindle speed in rotations per second. Positive for spindle forward (*M3*), negative for spindle reverse (*M4*).
- *motion.spindle-speed-out-rps-abs* - (float, out) Commanded spindle speed in rotations per second. This will always be a positive number.
- *motion.teleop-mode* - (bit, out) TRUE when motion is in *teleop mode*, as opposed to *coordinated mode*
- *motion.tooloffset.x* ... *motion.tooloffset.w* - (float, out, one per axis) shows the tool offset in effect; it could come from the tool table (*G43* active), or it could come from the gcode (*G43.1* active)
- `motion.spindle-orient-angle` - (float,out) Desired spindle orientation for M19. Value of the M19 R word parameter plus the value of the [RS274NGC]ORIENT_OFFSET ini parameter.
- `motion.spindle-orient-mode` - (s32,out) Desired spindle rotation mode M19. Default 0.
- `motion.spindle-orient` - (out,bit) Indicates start of spindle orient cycle. Set by M19. Cleared by any of M3,M4,M5. If spindle-orient-fault is not zero during spindle-orient true, the M19 command fails with an error message.
- `motion.spindle-is-oriented` - (in, bit) Acknowledge pin for spindle-orient. Completes orient cycle. If spindle-orient was true when spindle-is-oriented was asserted, the spindle-orient pin is cleared and the spindle-locked pin is asserted. Also, the spindle-brake pin is asserted.
- `motion.spindle-orient-fault` - (s32, in) Fault code input for orient cycle. Any value other than zero will cause the orient cycle to abort.
- `motion.spindle-lock` - (bit, out) Spindle orient complete pin. Cleared by any of M3,M4,M5.

**HAL pin usage for M19 orient spindle** Conceptually the spindle is in one of the following modes:

- rotation mode (the default)
- searching for desired orientation mode
- orienation complete mode.

When an M19 is executed, the spindle changes to *searching for desired orientation* , and the `spindle-orient` HAL pin is asserted. The desired target position is specified by the `spindle-orient-angle` and `spindle-orient-fwd` pins and driven by the M19 R and P parameters.

The HAL support logic is expected to react to `spindle-orient` by moving the spindle to the desired position. When this is complete, the HAL logic is expected to acknowledge this by asserting the `spindle-is-oriented` pin.

Motion then acknowledges this by deasserting the `spindle-orient` pin and asserts the `spindle-locked` pin to indicate *orientation complete* mode. It also raises the `spindle-brake` pin. The spindle now is in *orientation complete* mode.

If, during `spindle-orient` being true, and `spindle-is-oriented` not yet asserted the `spindle-orient-fault` pin has a value other than zero, the M19 command is aborted, a message including the fault code is displayed, and the motion queue is flushed. The spindle reverts to rotation mode.

Also, any of the M3,M4 or M5 commands cancel either *searching for desired orientation* or *orientation complete* mode. This is indicated by deasserting both the `spindle-orient` and `spindle-locked` pins.

The `spindle-orient-mode` pin reflects the M19 P word and shall be interpreted as follows:

- 0: rotate clockwise or counterclockwise for smallest angular movement
- 1: always rotate clockwise
- 2: always rotate counterclockwise

It can be used with the `orient` HAL component which provides a PID command value based on spindle encoder positon, `spindle-orient-angle` and `spindle-orient-mode`.

#### 12.5.1.3 Parameters

Many of these parameters serve as debugging aids, and are subject to change or removal at any time.

- *motion-command-handler.time* - (s32, RO)
- *motion-command-handler.tmax* - (s32, RW)
- *motion-controller.time* - (s32, RO)
- *motion-controller.tmax* - (s32, RW)
- *motion.debug-bit-0* - (bit, RO) This is used for debugging purposes.
- *motion.debug-bit-1* - (bit, RO) This is used for debugging purposes.
- *motion.debug-float-0* - (float, RO) This is used for debugging purposes.
- *motion.debug-float-1* - (float, RO) This is used for debugging purposes.
- *motion.debug-float-2* - (float, RO) This is used for debugging purposes.
- *motion.debug-float-3* - (float, RO) This is used for debugging purposes.
- *motion.debug-s32-0* - (s32, RO) This is used for debugging purposes.
- *motion.debug-s32-1* - (s32, RO) This is used for debugging purposes.
- *motion.servo.last-period* - (u32, RO) The number of CPU cycles between invocations of the servo thread. Typically, this number divided by the CPU speed gives the time in seconds, and can be used to determine whether the realtime motion controller is meeting its timing constraints
- *motion.servo.last-period-ns* - (float, RO)

#### 12.5.1.4 Functions

Generally, these functions are both added to the servo-thread in the order shown.

- *motion-command-handler* - Processes motion commands coming from user space
- *motion-controller* - Runs the LinuxCNC motion controller

### 12.5.2 Axis (Joints)

These pins and parameters are created by the realtime *motmod* module. These are actually joint values, but the pins and parameters are still called *axis.N*.[5] They are read and updated by the *motion-controller* function.

#### 12.5.2.1 Pins

- *axis.N.active* - (bit, out)
- *axis.N.amp-enable-out* - (bit, out) TRUE if the amplifier for this joint should be enabled
- *axis.N.amp-fault-in* - (bit, in) Should be driven TRUE if an external fault is detected with the amplifier for this joint
- *axis.N.backlash-corr* - (float, out)
- *axis.N.backlash-filt* - (float, out)
- *axis.N.backlash-vel* - (float, out)

[5] In *trivial kinematics* machines, there is a one-to-one correspondence between joints and axes.

- *axis.N.coarse-pos-cmd* - (float, out)
- *axis.N.error* - (bit, out)
- *axis.N.f-error* - (float, out)
- *axis.N.f-error-lim* - (float, out)
- *axis.N.f-errored* - (bit, out)
- *axis.N.faulted* - (bit, out)
- *axis.N.free-pos-cmd* - (float, out)
- *axis.N.free-tp-enable* - (bit, out)
- *axis.N.free-vel-lim* - (float, out)
- *axis.N.home-sw-in* - (bit, in) Should be driven TRUE if the home switch for this joint is closed.
- *axis.N.homed* - (bit, out)
- *axis.N.homing* - (bit, out) TRUE if the joint is currently homing
- *axis.N.in-position* - (bit, out)
- *axis.N.index-enable* - (bit, I/O)
- *axis.N.jog-counts* - (s32, in) Connect to the *counts* pin of an external encoder to use a physical jog wheel.
- *axis.N.jog-enable* - (bit, in) When TRUE (and in manual mode), any change in *jog-counts* will result in motion. When false, *jog-counts* is ignored.
- *axis.N.jog-scale* - (float, in) Sets the distance moved for each count on *jog-counts*, in machine units.
- *axis.N.jog-vel-mode* - (bit, in) When FALSE (the default), the jogwheel operates in position mode. The axis will move exactly jog-scale units for each count, regardless of how long that might take. When TRUE, the wheel operates in velocity mode - motion stops when the wheel stops, even if that means the commanded motion is not completed.
- *axis.N.joint-pos-cmd* - (float, out) The joint (as opposed to motor) commanded position. There may be an offset between the joint and motor positions—for example, the homing process sets this offset.
- *axis.N.joint-pos-fb* - (float, out) The joint (as opposed to motor) feedback position.
- *axis.N.joint-vel-cmd* - (float, out)
- *axis.N.kb-jog-active* - (bit, out)
- *axis.N.motor-pos-cmd* - (float, out) The commanded position for this joint.
- *axis.N.motor-pos-fb* - (float, in) The actual position for this joint.
- *axis.N.neg-hard-limit* - (bit, out)
- *axis.N.pos-lim-sw-in* - (bit, in) Should be driven TRUE if the positive limit switch for this joint is closed.
- *axis.N.pos-hard-limit* - (bit, out)
- *axis.N.neg-lim-sw-in* - (bit, in) Should be driven TRUE if the negative limit switch for this joint is closed.
- *axis.N.wheel-jog-active* - (bit, out)

#### 12.5.2.2 Parameters

- *axis.N.home-state* - Reflects the step of homing currently taking place.

### 12.5.3 iocontrol

iocontrol — accepts NML I/O commands, interacts with HAL in userspace.

The signals are turned on and off in userspace - if you have strict timing requirements or simply need more i/o, consider using the realtime synchronized i/o provided by motion instead.

#### 12.5.3.1 Pins

- *iocontrol.0.coolant-flood* - (bit, out) TRUE when flood coolant is requested.
- *iocontrol.0.coolant-mist* - (bit, out) TRUE when mist coolant is requested.
- *iocontrol.0.emc-enable-in* - (bit, in) Should be driven FALSE when an external E-Stop condition exists.
- *iocontrol.0.lube* - (bit, out) TRUE when lube is commanded.
- *iocontrol.0.lube_level* - (bit, in) Should be driven TRUE when lube level is high enough.
- *iocontrol.0.tool-change* - (bit, out) TRUE when a tool change is requested.
- *iocontrol.0.tool-changed* - (bit, in) Should be driven TRUE when a tool change is completed.
- *iocontrol.0.tool-number* - (s32, out) The current tool number.
- *iocontrol.0.tool-prep-number* - (s32, out) The number of the next tool, from the RS274NGC T-word.
- *iocontrol.0.tool-prepare* - (bit, out) TRUE when a tool prepare is requested.
- *iocontrol.0.tool-prepared* - (bit, in) Should be driven TRUE when a tool prepare is completed.
- *iocontrol.0.user-enable-out* - (bit, out) FALSE when an internal E-Stop condition exists.
- *iocontrol.0.user-request-enable* - (bit, out) TRUE when the user has requested that E-Stop be cleared.

### 12.5.4 ini settings

A number of ini settings are made available as hal input pins.

#### 12.5.4.1 Pins

- *ini.n.min_limit* - (float, in) [AXIS_n]MIN_LIMIT
- *ini.n.max_limit* - (float, in) [AXIS_n]MAX_LIMIT
- *ini.n.ferror* - (float, in) [AXIS_n]FERROR
- *ini.n.min_ferror* - (float, in) [AXIS_n]MIN_FERROR
- *ini.n.max_velocity* - (float, in) [AXIS_n]MAX_VELOCITY
- *ini.n.max_acceleration* - (float, in) [AXIS_n]MAX_ACCELERATION
- *ini.n.backlash* - (float, in) [AXIS_n]BACKLASH

**Note**
The per-axis min_limit and max_limit pins are honored continuously after homing. The per-axis ferror and min_ferror pins are honored when the machine is on and not in position. The per-axis max_velocity and max_acceleration pins are sampled when the machine is on and the motion_state is free (homing or jogging) but are not sampled when in a program is running (auto mode) or in mdi mode. Consequently, changing the pin values when a program is running will not have effect until the program is stopped and the motion_state is again free.

- *ini.traj_arc_blend_enable* - (bit, in) [TRAJ]ARC_BLEND_ENABLE
- *ini.traj_arc_blend_fallback_enable* - (bit, in) [TRAJ]ARC_BLEND_FALLBACK_ENABLE
- *ini.traj_arc_blend_gap_cycles* - (float, in) [TRAJ]ARC_BLEND_GAP_CYCLES
- *ini.traj_arc_blend_optimization_depth* - (float, in) [TRAJ]ARC_BLEND_OPTIMIZATION_DEPTH
- *ini.traj_arc_blend_ramp_freq* - (float, in) [TRAJ]ARC_BLEND_RAMP_FREQ

**Note**
The traj_arc_blend pins are sampled continuously but changing pin values while a program is running may not have immediate effect due to queueing of commands.

- *ini.traj_default_acceleration* - (float, in) [TRAJ]DEFAULT_ACCELERATION
- *ini.traj_default_velocity* - (float, in) [TRAJ]DEFAULT_VELOCITY
- *ini.traj_max_acceleration* - (float, in) [TRAJ]MAX_ACCELERATION

## 12.6 Canonical Device Interfaces

### 12.6.1 Introduction

The following sections show the pins, parameters, and functions that are supplied by "canonical devices". All HAL device drivers should supply the same pins and parameters, and implement the same behavior.

Note that the only the `<io-type>` and `<specific-name>` fields are defined for a canonical device. The `<device-name>`, `<device-num>`, and `<chan-num>` fields are set based on the characteristics of the real device.

### 12.6.2 Digital Input

The canonical digital input (I/O type field: `digin`) is quite simple.

#### 12.6.2.1 Pins

- (bit) **in** — State of the hardware input.
- (bit) **in-not** — Inverted state of the input.

#### 12.6.2.2 Parameters

- None

#### 12.6.2.3 Functions

- (funct) **read** — Read hardware and set `in` and `in-not` HAL pins.

### 12.6.3 Digital Output

The canonical digital output (I/O type field: `digout`) is also very simple.

#### 12.6.3.1 Pins

- (bit) **out** — Value to be written (possibly inverted) to the hardware output.

#### 12.6.3.2 Parameters

- (bit) **invert** — If TRUE, **out** is inverted before writing to the hardware.

#### 12.6.3.3 Functions

- (funct) **write** — Read **out** and **invert**, and set hardware output accordingly.

### 12.6.4 Analog Input

The canonical analog input (I/O type: `adcin` ). This is expected to be used for analog to digital converters, which convert e.g. voltage to a continuous range of values.

#### 12.6.4.1 Pins

- (float) **value** — The hardware reading, scaled according to the **scale** and **offset** parameters. **Value** = ((input reading, in hardware-dependent units) * **scale**) - **offset**

#### 12.6.4.2 Parameters

- (float) **scale** — The input voltage (or current) will be multiplied by **scale** before being output to **value**.
- (float) **offset** — This will be subtracted from the hardware input voltage (or current) after the scale multiplier has been applied.
- (float) **bit_weight** — The value of one least significant bit (LSB). This is effectively the granularity of the input reading.
- (float) **hw_offset** — The value present on the input when 0 volts is applied to the input pin(s).

#### 12.6.4.3 Functions

- (funct) **read** — Read the values of this analog input channel. This may be used for individual channel reads, or it may cause all channels to be read

### 12.6.5 Analog Output

The canonical analog output (I/O Type: **adcout**). This is intended for any kind of hardware that can output a more-or-less continuous range of values. Examples are digital to analog converters or PWM generators.

#### 12.6.5.1 Pins

- (float) **value** — The value to be written. The actual value output to the hardware will depend on the scale and offset parameters.
- (bit) **enable** — If false, then output 0 to the hardware, regardless of the **value** pin.

#### 12.6.5.2 Parameters

- (float) **offset** — This will be added to the **value** before the hardware is updated
- (float) **scale** — This should be set so that an input of 1 on the **value** pin will cause the analog output pin to read 1 volt.
- (float) **high_limit** (optional) — When calculating the value to output to the hardware, if **value + offset** is greater than **high_limit**, then **high_limit** will be used instead.
- (float) **low_limit** (optional) — When calculating the value to output to the hardware, if **value + offset** is less than **low_limit**, then **low_limit** will be used instead.
- (float) **bit_weight** (optional) — The value of one least significant bit (LSB), in volts (or mA, for current outputs)
- (float) **hw_offset** (optional) — The actual voltage (or current) that will be output if 0 is written to the hardware.

#### 12.6.5.3 Functions

(funct) **write** — This causes the calculated value to be output to the hardware. If enable is false, then the output will be 0, regardless of **value**, **scale**, and **offset**. The meaning of "0" is dependent on the hardware. For example, a bipolar 12-bit A/D may need to write 0x1FF (mid scale) to the D/A get 0 volts from the hardware pin. If enable is true, read scale, offset and value and output to the adc (**scale** * **value**) + **offset**. If enable is false, then output 0.

## 12.7 HAL Tools

### 12.7.1 Halcmd

Halcmd is a command line tool for manipulating the HAL. There is a rather complete man page for halcmd, which will be installed if you have installed LinuxCNC from either source or a package. The manpage provides usage info:

```
man halcmd
```

If you have compiled LinuxCNC for "run-in-place", you must source the rip-environment script to make the man page available:

```
cd toplevel_directory_for_rip_build
. scripts/rip-environment
man halcmd
```

The HAL Tutorial has a number of examples of halcmd usage, and is a good tutorial for halcmd.

### 12.7.2 Halmeter

Halmeter is a *voltmeter* for the HAL. It lets you look at a pin, signal, or parameter, and displays the current value of that item. It is pretty simple to use. Start it by typing **halmeter** in an X windows shell. Halmeter is a GUI application. It will pop up a small window, with two buttons labeled *Select* and *Exit*. Exit is easy - it shuts down the program. Select pops up a larger window, with three tabs. One tab lists all the pins currently defined in the HAL. The next lists all the signals, and the last tab lists all the parameters. Click on a tab, then click on a pin/signal/parameter. Then click on *OK*. The lists will disappear, and the small window will display the name and value of the selected item. The display is updated approximately 10 times per second. If you click *Accept* instead of *OK*, the small window will display the name and value of the selected item, but the large window will remain on the screen. This is convenient if you want to look at a number of different items quickly.

You can have many halmeters running at the same time, if you want to monitor several items. If you want to launch a halmeter without tying up a shell window, type *halmeter &* to run it in the background. You can also make halmeter start displaying a specific item immediately, by adding *pin|sig|par[am] <name>* to the command line. It will display the pin, signal, or parameter <name> as soon as it starts. (If there is no such item, it will simply start normally.) And finally, if you specify an item to display, you can add *-s* before the pin|sig|param to tell halmeter to use a small window. The item name will be displayed in the title bar instead of under the value, and there will be no buttons. Useful when you want a lot of meters in a small amount of screen space.

Refer to Halmeter Tutorial section for more information.

Halmeter can be loaded from a terminal or from Axis. Halmeter is faster than Halshow at displaying values. Halmeter has two windows, one to pick the pin, signal, or parameter to monitor and one that displays the value. Multiple Halmeters can be open at the same time. If you use a script to open multiple Halmeters you can set the position of each one with -g X Y relative to the upper left corner of your screen. For example:

```
loadusr halmeter pin hm2.0.stepgen.00.velocity-fb -g 0 500
```

See the man page for more options. See section Halmeter.

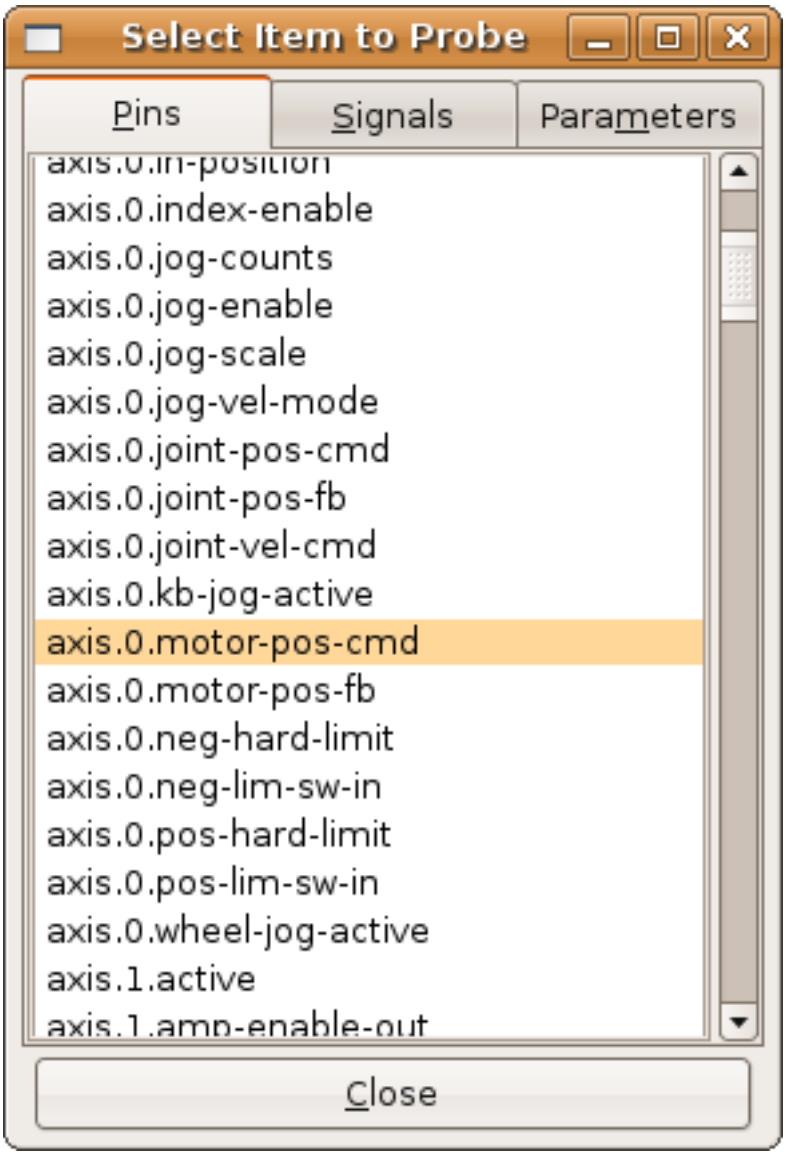
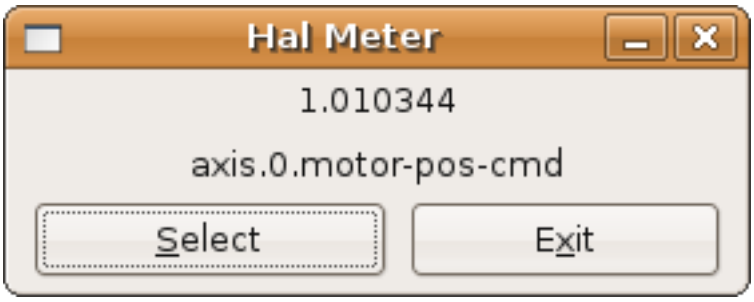



Figure 12.16: Halmeter




### 12.7.3 Halshow

Halshow (see separate section for complete usage description) can be started from the command line to show details for selected components, pins, parameters, signals, functions, and threads of a running HAL. The WATCH tab provides a continuous display of selected pin, parameters, and signal items. The File menu provides buttons to 1) save the watch items to a watch list and to load and existing watch list. The watch list items can also be loaded automatically on startup. For command line usage:

```
halshow --help
Usage:
  halshow [Options] [watchfile]
  Options:
           --help  (this help)

Notes:
       Create watchfile in halshow using: 'File/Save Watch List'
       linuxcnc must be running for standalone usage
```

### 12.7.4 Halscope

Halscope is an *oscilloscope* for the HAL. It lets you capture the value of pins, signals, and parameters as a function of time. Complete operating instructions should be located here eventually. For now, refer to section [sec:tutorial-halscope] in the tutorial chapter, which explains the basics.

The halscope File menu selector provides buttons to save a configuration or open a previously saved configuration. When halscope is terminated, the last configuration is saved in a file named autosave.halscope.

Configuration files may also be specified when starting halscope from the commandline. Commandline help (-h) usage:

```
halscope -h
Usage:
  halscope [-h] [-i infile] [-o outfile] [num_samples]
```

### 12.7.5 Sim Pin

sim_pin is a command line utility to display and update any number of writable pins, parameters or signals. Usage:

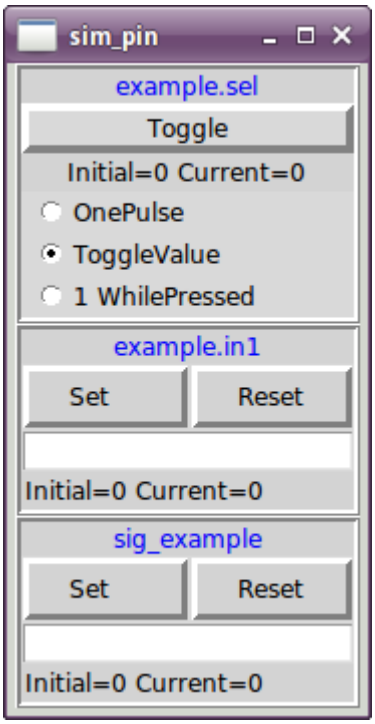
```
sim_pin
  Usage:
         sim_pin name1 [name2 ...] &

  Note:  linuxcnc must be running
         A named item can specify a pin, param, or signal
         The item must be writable, e.g.:
            pin:    IN or I/O (and not connected to a signal with a writer)
            param:  RW
            signal: connected to a writable pin
```

For complete information, see the man page:

```
man sim_pin
```

Example (with LinuxCNC running):

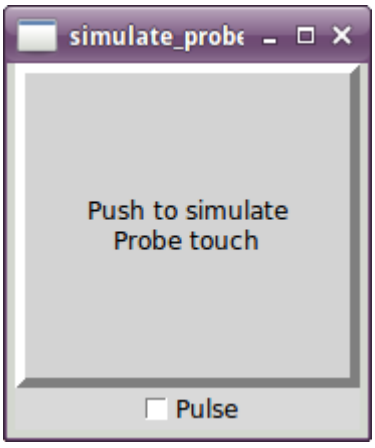
```
halcmd loadrt mux2 names=example; halcmd net sig_example example.in0
sim_pin example.sel example.in1 sig_example &
```




### 12.7.6 Simulate Probe

simulate_probe is a simple gui to simulate activation of the pin motion.probe-input. Usage:

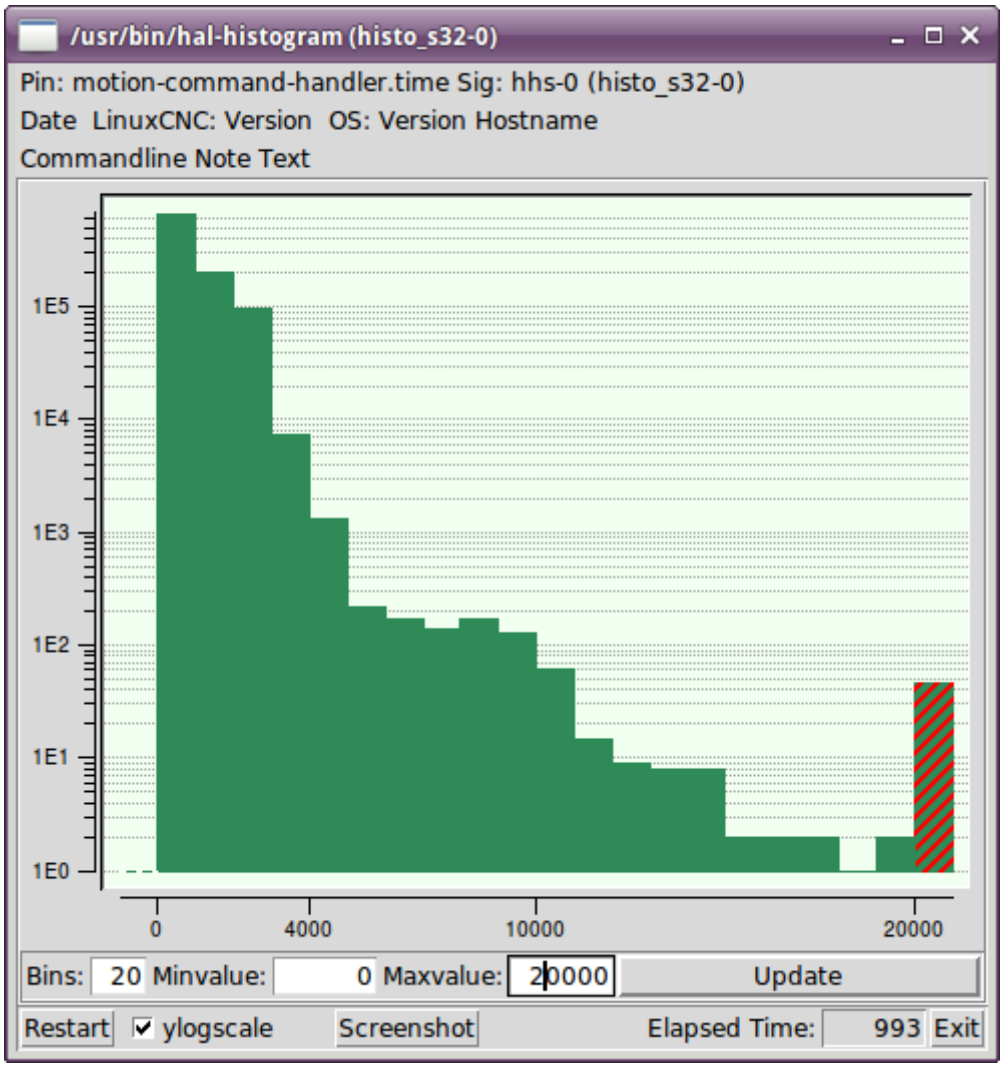
```
simulate_probe &
```



### 12.7.7 Hal Histogram

hal-histogram is a command line utility to display histograms for hal pins.

```
Usage:
   hal-histogram --help | -?
or
   hal-histogram [Options] [pinname]

Options:
  --minvalue  minvalue (minimum bin, default: 0)
  --binsize   binsize  (binsize, default: 100)
  --nbins     nbins    (number of bins, default: 50)

  --logscale  0|1      (y axis log scale, default: 1)
  --text      note     (text display, default: "" )
  --show               (show count of undisplayed nbins, default off)
  --verbose            (progress and debug, default off)

Notes:
  1) LinuxCNC (or another Hal application) must be running
  2) If no pinname is specified, default is: motion-command-handler.time
  3) This app may be opened for 5 pins
  4) pintypes float, s32, u32, bit are supported
  5) The pin must be associated with a thread supporting floating point
     For a base thread, this may require using:
     loadrt motmod ... base_thread_fp=1
```




## 12.8 Halshow

The script halshow can help you find your way around a running HAL. This is a very specialized system and it must connect to a working HAL. It cannot run standalone because it relies on the ability of HAL to report what it knows of itself through the halcmd interface library. It is discovery based. Each time halshow runs with a different LinuxCNC configuration it will be different.

As we will soon see, this ability of HAL to document itself is one key to making an effective CNC system.

### 12.8.1 Starting Halshow

Halshow is in the AXIS menu under Machine/Show HAL Configuration.

Halshow is in the TkLinuxCNC menu under Scripts/HAL Show.

### 12.8.2 HAL Tree Area

At the left of its display as shown in figure is a tree view, somewhat like you might see with some file browsers. At the right is a tabbed notebook with tabs for show and watch.

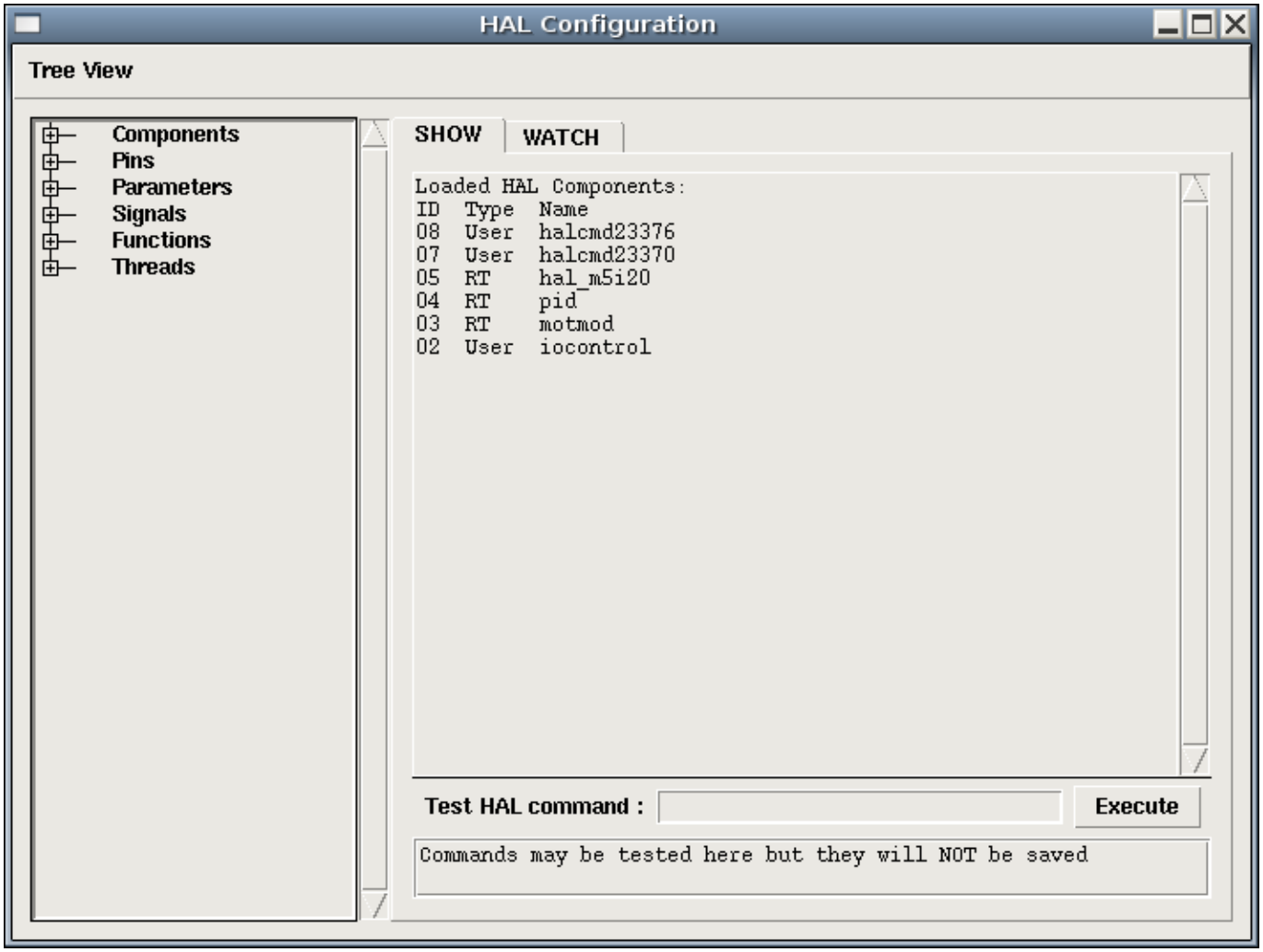



Figure 12.17: Halshow Layout

The tree shows all of the major parts of a HAL. In front of each is a small plus (+) or minus (-) sign in a box. Clicking the plus will expand that tree node to display what is under it. If that box shows a minus sign, clicking it will collapse that section of the tree.

You can also expand or collapse the tree display using the Tree View menu at the upper left edge of the display. Under Tree View you will find: Expand Tree, Collapse Tree; Expand Pins, Expand Parameters, Expand Signals; and Erase Watch. (Note that Erase Watch erases *all* previously set watches, you cannot erase just one watch.)

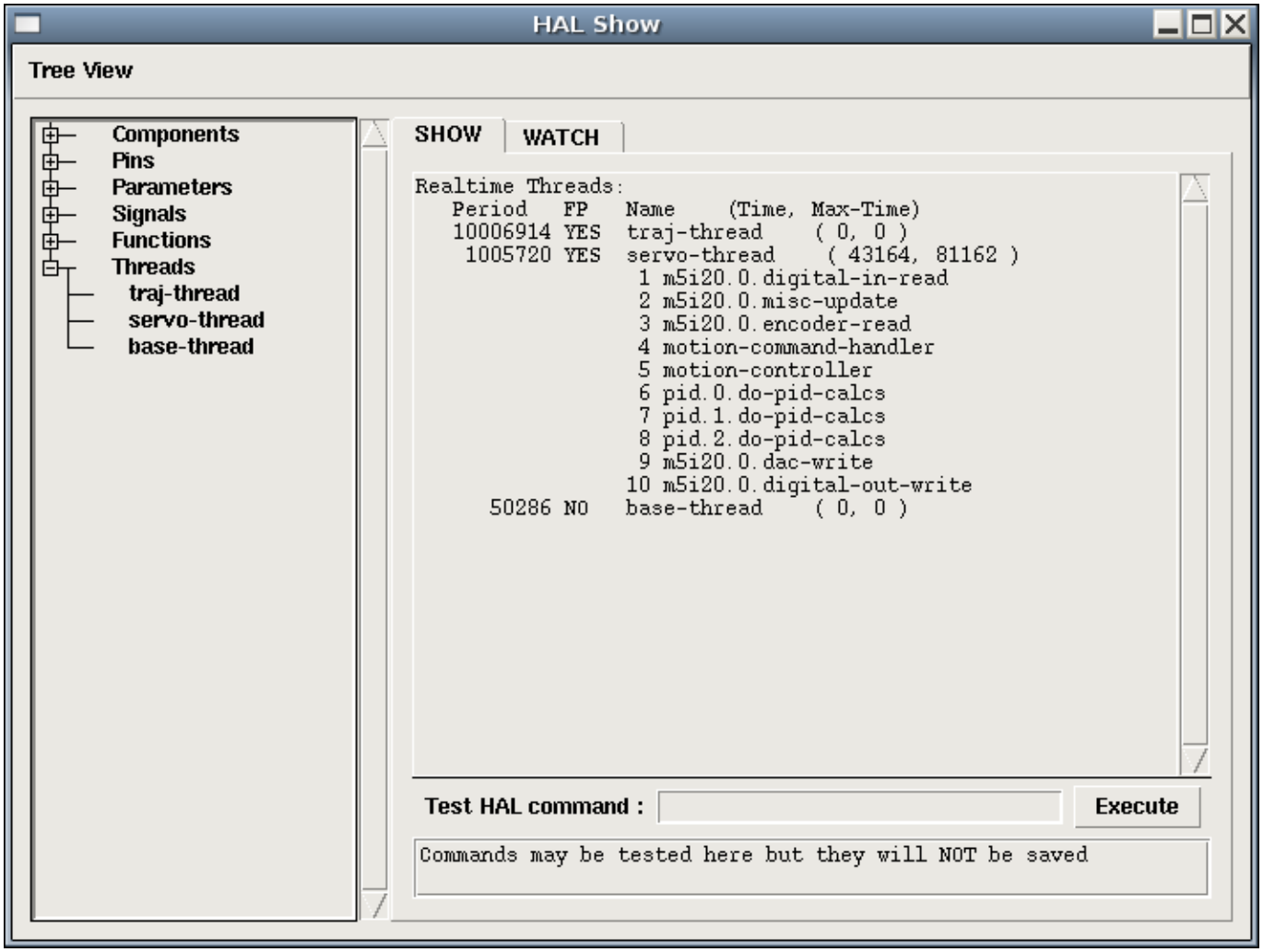



Figure 12.18: Show Menu

### 12.8.3 HAL Show Area

Clicking on the node name, the word "Components" for example, will show you (under the "Show" tab) all that HAL knows about the contents of that node. Figure Figure 12.17 shows a list exactly like you will see if you click the "Components" name while you are running a standard m5i20 servo card. The information display is exactly like those shown in traditional text based HAL analysis tools. The advantage here is that we have mouse click access, access that can be as broad or as focused as you need.

If we take a closer look at the tree display we can see that the six major parts of a HAL can all be expanded at least one level. As these levels are expanded you can get more focused with the reply when you click on the rightmost tree node. You will find that there are some HAL pins and parameters that show more than one reply. This is due to the nature of the search routines in halcmd itself. If you search one pin you may get two, like this:

```
Component Pins:
Owner  Type  Dir  Value  Name
06     bit    -W   TRUE  parport.0.pin-10-in
06     bit    -W  FALSE  parport.0.pin-10-in-not
```

The second pin's name contains the complete name of the first.

Below the show area on the right is a set of widgets that will allow you to play with the running HAL. The commands you enter

here and the effect that they have on the running HAL are not saved. They will persist as long as LinuxCNC remains up but are gone as soon as LinuxCNC is.

The entry box labeled "Test HAL Command:" will accept any of the commands listed for halcmd. These include:

- loadrt, unloadrt (load/unload real-time module)
- loadusr, unloadusr (load/unload user-space component)
- addf, delf (add/delete a function to/from a real-time thread)
- net (create a connection between two or more items)
- setp (set parameter (or pin) to a value)

This little editor will enter a command any time you press <enter> or push the execute button. An error message from halcmd will show below this entry widget when these commands are not properly formed. If you are not certain how to set up a proper command you'll need to read again the documentation on halcmd and the specific modules that you are working with.

Let's use this editor to add a differential module to a HAL and connect it to axis position so that we could see the rate of change in position, i.e., acceleration. We first need to load a HAL component named ddt, add it to the servo thread, then connect it to the position pin of an axis. Once that is done we can find the output of the differentiator in halscope. So let's go. (Yes, I looked this one up.)

```
loadrt ddt
```

Now look at the components node and you should see ddt in there someplace.

```
Loaded HAL Components:
ID Type        Name
10 User halcmd29800
09 User halcmd29374
08   RT         ddt
06   RT hal_parport
05   RT    scope_rt
04   RT     stepgen
03   RT      motmod
02 User   iocontrol
```

Sure enough there it is. Notice that its ID is 08. Next we need to find out what functions are available with it so we look at functions:

```
Exported Functions:
Owner  CodeAddr      Arg  FP Users Name
  08   E0B97630 E0DC7674 YES     0 ddt.0
  03   E0DEF83C 00000000 YES     1 motion-command-handler
  03   E0DF0BF3 00000000 YES     1 motion-controller
  06   E0B541FE E0DC75B8  NO     1 parport.0.read
  06   E0B54270 E0DC75B8  NO     1 parport.0.write
  06   E0B54309 E0DC75B8  NO     0 parport.read-all
  06   E0B5433A E0DC75B8  NO     0 parport.write-all
  05   E0AD712D 00000000  NO     0 scope.sample
  04   E0B618C1 E0DC7448 YES     1 stepgen.capture-position
  04   E0B612F5 E0DC7448  NO     1 stepgen.make-pulses
  04   E0B614AD E0DC7448 YES     1 stepgen.update-freq
```

Here we look for owner #08 and see a function named ddt.0. We should be able to add ddt.0 to the servo thread and it will do its math each time the servo thread is updated. Once again we look up the addf command and find that it uses three arguments like this:

```
addf <functname> <threadname> [<position>]
```

We already know the functname=ddt.0 so let's get the thread name right by expanding the thread node in the tree. Here we see two threads, servo-thread and base-thread. The position of ddt.0 in the thread is not critical. So we add the function ddt.0 to the servo-thread:

```
addf ddt.0 servo-thread
```

This is just for viewing, so we leave position blank and get the last position in the thread. The following figure shows the state of halshow after this command has been issued.

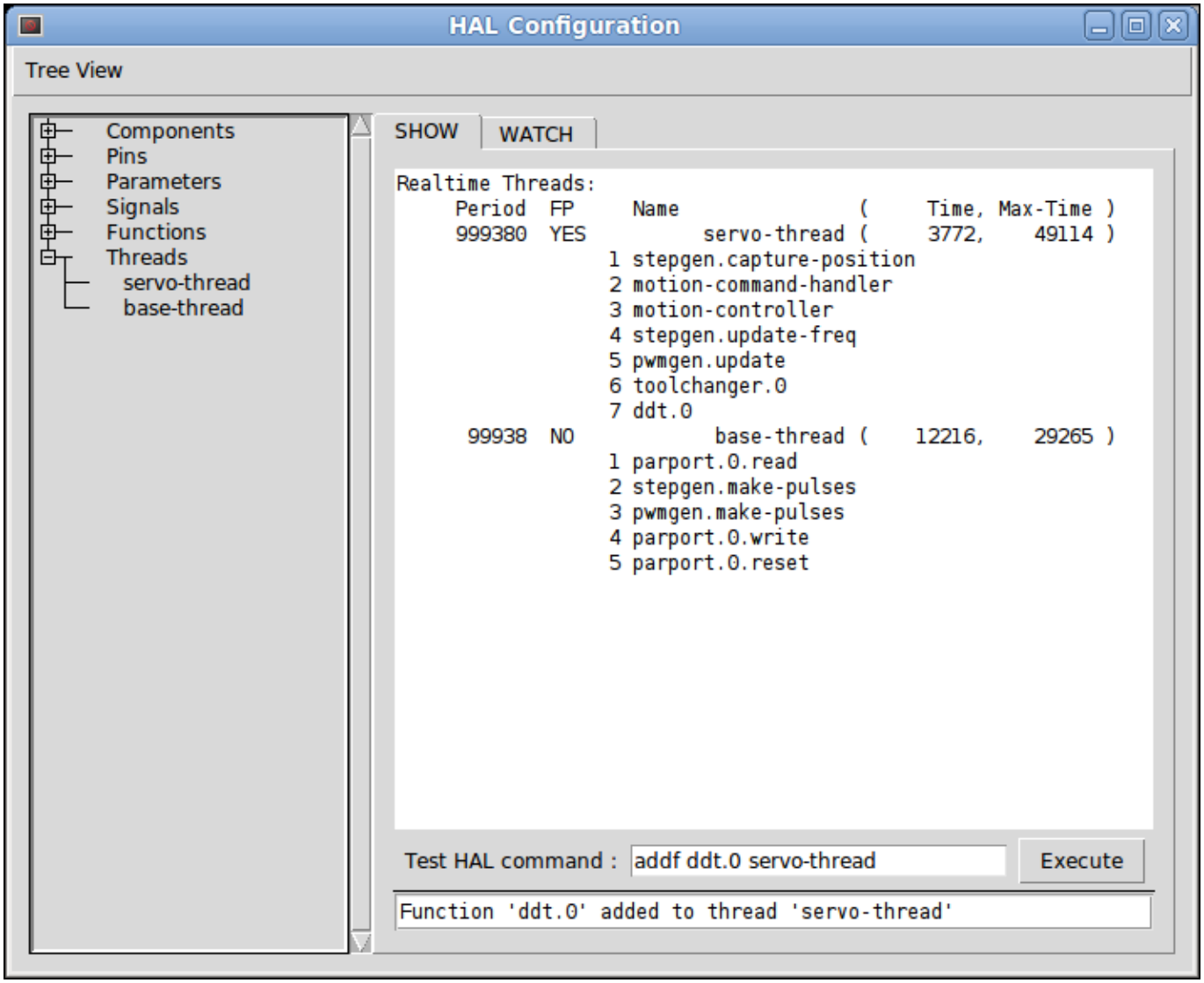



Figure 12.19: Addf Command

Next we need to connect ddt to something. But how do we know what pins are available? The answer is to look under pins. There we find ddt and see this:

```
Component Pins:
Owner Type  Dir Value       Name
08    float R-  0.00000e+00 ddt.0.in
08    float -W  0.00000e+00 ddt.0.out
```

That looks easy enough to understand, but what signal or pin do we want to connect to it? It could be an axis pin, a stepgen pin, or a signal. We see this when we look at axis.0:

```
Component Pins:
Owner Type  Dir Value       Name
03    float -W  0.00000e+00 axis.0.motor-pos-cmd ==> Xpos-cmd
```

So it looks like Xpos-cmd should be a good signal to use. Back to the editor where we enter the following command:

```
linksp Xpos-cmd ddt.0.in
```

Now if we look at the Xpos-cmd signal using the tree node we'll see what we've done:

```
Signals:
Type Value Name
float 0.00000e+00 Xpos-cmd
<== axis.0.motor-pos-cmd
==> ddt.0.in
==> stepgen.0.position-cmd
```

We see that this signal comes from axis.o.motor-pos-cmd and goes to both ddt.0.in and stepgen.0.position-cmd. By connecting our block to the signal we have avoided any complications with the normal flow of this motion command.

The HAL Show Area uses halcmd to discover what is happening in a running HAL. It gives you complete information about what it has discovered. It also updates as you issue commands from the little editor panel to modify that HAL. There are times when you want a different set of things displayed without all of the information available in this area. That is where the HAL Watch Area is of value.

### 12.8.4 Watch Tab

Clicking the watch tab produces a blank canvas. You can add signals and pins to this canvas and watch their values.[6] You can add signals or pins when the watch tab is displayed by clicking on the name of it. The following figure shows this canvas with several "bit" type signals. These signals include enable-out for the first three axes and two of the three iocontrol "estop" signals. Notice that the axes are not enabled even though the estop signals say that LinuxCNC is not in estop. A quick look at TkLinuxCNC shows that the condition of LinuxCNC is ESTOP RESET. The amp enables do not turn true until the machine has been turned on.

[6] The refresh rate of the watch display is much lower than Halmeter or Halscope. If you need good resolution of the timing of signals those tools are much more effective.

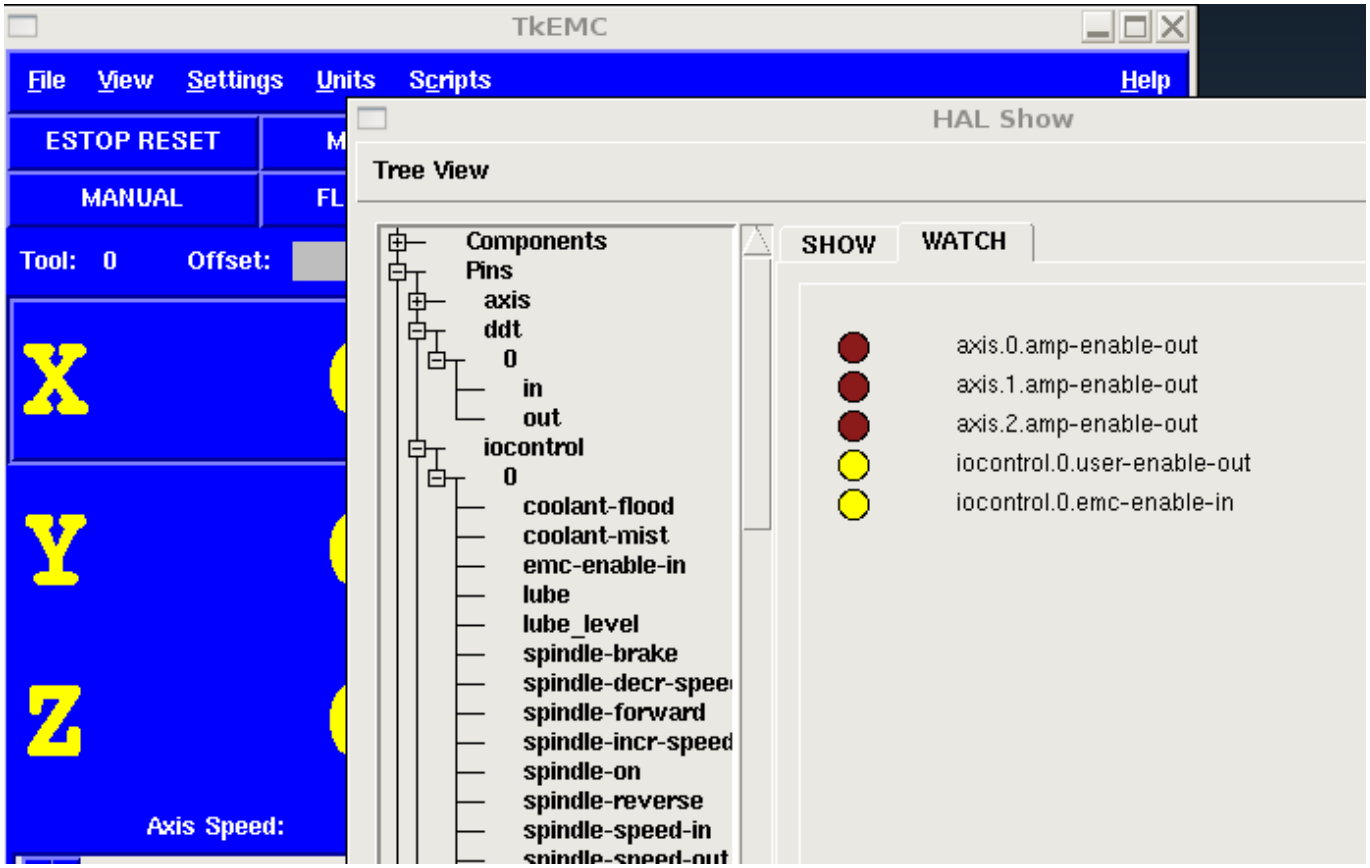



Figure 12.20: Watch Tab

Watch displays bit type (binary) values using colored circles representing LEDs. They show as dark brown when a bit signal or pin is false, and as light yellow whenever that signal is true. If you select a pin or signal that is not a bit type (binary) signal, watch will show it as a numerical value.

Watch will quickly allow you to test switches or see the effect of changes that you make to LinuxCNC while using the graphical interface. Watch's refresh rate is a bit slow to see stepper pulses, but you can use it for these if you move an axis very slowly or in very small increments of distance. If you've used IO_Show in LinuxCNC, the watch page in halshow can be set up to watch a parport much as IO_Show did.

## 12.9 HAL Components

### 12.9.1 Commands and Userspace Components

All of the commands in the following list have man pages. Some will have expanded descriptions, some will have limited descriptions. Also, all of the components listed below have man pages. From these two lists you know what components exist, and you can use *man n name* to get additional information. To view the information in the man page, in a terminal window type:

```
man axis (or perhaps 'man 1 axis' if your system requires it.)
```

**axis**
: AXIS LinuxCNC (The Enhanced Machine Controller) Graphical User Interface.

**axis-remote**
: AXIS Remote Interface.

**comp**
Build, compile and install LinuxCNC HAL components.

**gladevcp**
Virtual Control Panel for LinuxCNC based on Glade, Gtk and HAL widgets.

**gs2**
HAL userspace component for Automation Direct GS2 VFD's.

**halcmd**
Manipulate the Enhanced Machine Controller HAL from the command line.

**hal_input**
Control HAL pins with any Linux input device, including USB HID devices.

**halmeter**
Observe HAL pins, signals, and parameters.

**halrun**
Manipulate the Enhanced Machine Controller HAL from the command line.

**halsampler**
Sample data from HAL in realtime.

**halstreamer**
Stream file data into HAL in real time.

**halui**
Observe HAL pins and command LinuxCNC through NML.

**io**
Accepts NML I/O commands, interacts with HAL in userspace.

**iocontrol**
Accepts NML I/O commands, interacts with HAL in userspace.

**linuxcnc**
LinuxCNC (The Enhanced Machine Controller).

**pyvcp**
Virtual Control Panel for LinuxCNC.

**shuttlexpress**
control HAL pins with the ShuttleXpress device made by Contour Design.

### 12.9.2 Realtime Components List

Some of these will have expanded descriptions from the man pages. Some will have limited descriptions. All of the components have man pages. From this list you know what components exist and can use *man n name* to get additional information in a terminal window.

**Note**
If the component requires a floating point thread that is usually the slower servo-thread.

#### 12.9.2.1 Core LinuxCNC components

**motion**
Accepts NML motion commands, interacts with HAL in realtime.

**axis**
Accepts NML motion commands, interacts with HAL in realtime.

**classicladder**
Realtime software PLC based on ladder logic. See ClassicLadder chapter for more information.

**gladevcp**
Displays Virtual Control Panels built with GTK/Glade.

**threads**
Creates hard realtime HAL threads.

#### 12.9.2.2 Logic and Bitwise components

**and2**
Two-input AND gate. For out to be true both inputs must be true.

**not**
Inverter.

**or2**
Two-input OR gate.

**xor2**
Two-input XOR (exclusive OR) gate.

**debounce**
Filter noisy digital inputs. Description

**edge**
Edge detector.

**flipflop**
D type flip-flop.

**oneshot**
One-shot pulse generator.

**logic**
General logic function component.

**lut5**
A 5-input logic function based on a look-up table. Description

**match8**
8-bit binary match detector.

**select8**
8-bit binary match detector.

#### 12.9.2.3 Arithmetic and float-components

**abs**
Compute the absolute value and sign of the input signal.

**blend**
Perform linear interpolation between two values.

**comp**
Two input comparator with hysteresis.

**constant**
Use a parameter to set the value of a pin.

**sum2**
Sum of two inputs (each with a gain) and an offset.

**counter**
Counts input pulses (deprecated). Use the encoder component.

**updown**
Counts up or down, with optional limits and wraparound behavior.

**ddt**
Compute the derivative of the input function.

**deadzone**
Return the center if within the threshold.

**hypot**
Three-input hypotenuse (Euclidean distance) calculator.

**mult2**
Product of two inputs.

**mux16**
Select from one of sixteen input values.

**mux2**
Select from one of two input values.

**mux4**
Select from one of four input values.

**mux8**
Select from one of eight input values.

**near**
Determine whether two values are roughly equal.

**offset**
Adds an offset to an input, and subtracts it from the feedback value.

**integ**
Integrator.

**invert**
Compute the inverse of the input signal.

**wcomp**
Window comparator.

**weighted_sum**
Convert a group of bits to an integer.

**biquad**
Biquad IIR filter

**lowpass**
Low-pass filter

**limit1**
Limit the output signal to fall between min and max. [7]

**limit2**
Limit the output signal to fall between min and max. Limit its slew rate to less than maxv per second. [8]

**limit3**
Limit the output signal to fall between min and max. Limit its slew rate to less than maxv per second. Limit its second derivative to less than MaxA per second squared. [9]

**maj3**
Compute the majority of 3 inputs.

**scale**
Applies a scale and offset to its input.

#### 12.9.2.4 Type conversion

**conv_bit_s32**
Convert a value from bit to s32.

**conv_bit_u32**
Convert a value from bit to u32.

**conv_float_s32**
Convert a value from float to s32.

**conv_float_u32**
Convert a value from float to u32.

**conv_s32_bit**
Convert a value from s32 to bit.

**conv_s32_float**
Convert a value from s32 to float.

**conv_s32_u32**
Convert a value from s32 to u32.

**conv_u32_bit**
Convert a value from u32 to bit.

**conv_u32_float**
Convert a value from u32 to float.

**conv_u32_s32**
Convert a value from u32 to s32.

[7] When the input is a position, this means that the *position* is limited.
[8] When the input is a position, this means that *position* and *velocity* are limited.
[9] When the input is a position, this means that the *position*, *velocity*, and *acceleration* are limited.

#### 12.9.2.5 Hardware Drivers

**hm2_7i43**
HAL driver for the Mesa Electronics 7i43 EPP Anything IO board with HostMot2.

**hm2_pci**
HAL driver for the Mesa Electronics 5i20, 5i22, 5i23, 4i65, and 4i68 Anything I/O boards, with HostMot2 firmware.

**hostmot2**
HAL driver for the Mesa Electronics HostMot2 firmware.

**mesa_7i65**
Support for the Mesa 7i65 eight-axis servo card.

**pluto_servo**
Hardware driver and firmware for the Pluto-P parallel-port FPGA, for use with servos.

**pluto_step**
Hardware driver and firmware for the Pluto-P parallel-port FPGA, for use with steppers.

**thc**
Torch Height Control using a Mesa THC card.

**serport**
Hardware driver for the digital I/O bits of the 8250 and 16550 serial port.

#### 12.9.2.6 Kinematics

**kins**
kinematics definitions for LinuxCNC.

**gantrykins**
A kinematics module that maps one axis to multiple joints.

**genhexkins**
Gives six degrees of freedom in position and orientation (XYZABC). The location of the motors is defined at compile time.

**genserkins**
Kinematics that can model a general serial-link manipulator with up to 6 angular joints.

**maxkins**
Kinematics for a tabletop 5 axis mill named *max* with tilting head (B axis) and horizontal rotary mounted to the table (C axis). Provides UVW motion in the rotated coordinate system. The source file, maxkins.c, may be a useful starting point for other 5-axis systems.

**tripodkins**
The joints represent the distance of the controlled point from three predefined locations (the motors), giving three degrees of freedom in position (XYZ).

**trivkins**
There is a 1:1 correspondence between joints and axes. Most standard milling machines and lathes use the trivial kinematics module.

**pumakins**
Kinematics for PUMA-style robots.

**rotatekins**
The X and Y axes are rotated 45 degrees compared to the joints 0 and 1.

**scarakins**
Kinematics for SCARA-type robots.

#### 12.9.2.7 Motor control

**at_pid**
Proportional/integral/derivative controller with auto tuning.

**pid**
Proportional/integral/derivative controller. Description

**pwmgen**
Software PWM/PDM generation. Description

**encoder**
Software counting of quadrature encoder signals. Description.

**stepgen**
Software step pulse generation. Description.

#### 12.9.2.8 BLDC and 3-phase motor control

**bldc_hall3**
3-wire Bipolar trapezoidal commutation BLDC motor driver using Hall sensors.

**clarke2**
Two input version of Clarke transform.

**clarke3**
Clarke (3 phase to cartesian) transform.

**clarkeinv**
Inverse Clarke transform.

#### 12.9.2.9 Other

**charge_pump**
Creates a square-wave for the *charge pump* input of some controller boards. The *Charge Pump* should be added to the base thread function. When enabled the output is on for one period and off for one period. To calculate the frequency of the output 1/(period time in seconds x 2) = hz. For example if you have a base period of 100,000ns that is 0.0001 seconds and the formula would be 1/(0.0001 x 2) = 5,000 hz or 5 Khz.

**encoder_ratio**
An electronic gear to synchronize two axes.

**estop_latch**
ESTOP latch.

**feedcomp**
Multiply the input by the ratio of current velocity to the feed rate.

**gearchange**
Select from one of two speed ranges.

**ilowpass**
While it may find other applications, this component was written to create smoother motion while jogging with an MPG.

In a machine with high acceleration, a short jog can behave almost like a step function. By putting the ilowpass component between the MPG encoder counts output and the axis jog-counts input, this can be smoothed.

Choose scale conservatively so that during a single session there will never be more than about 2e9/scale pulses seen on the MPG. Choose gain according to the smoothing level desired. Divide the axis.N.jog-scale values by scale.

**joyhandle**
Sets nonlinear joypad movements, deadbands and scales.

**knob2float**
Convert counts (probably from an encoder) to a float value.

**minmax**
Track the minimum and maximum values of the input to the outputs.

**sample_hold**
Sample and Hold.

**sampler**
Sample data from HAL in real time.

**siggen**
Signal generator. Description.

**sim_encoder**
Simulated quadrature encoder. Description.

**sphereprobe**
Probe a pretend hemisphere.

**steptest**
Used by Stepconf to allow testing of acceleration and velocity values for an axis.

**streamer**
Stream file data into HAL in real time.

**supply**
Set output pins with values from parameters (deprecated).

**threadtest**
Component for testing thread behavior.

**time**
Accumulated run-time timer counts HH:MM:SS of *active* input.

**timedelay**
The equivalent of a time-delay relay.

**timedelta**
Component that measures thread scheduling timing behavior.

**toggle2nist**
Toggle button to nist logic.

**toggle**
Push-on, push-off from momentary pushbuttons.

**tristate_bit**
Place a signal on an I/O pin only when enabled, similar to a tristate buffer in electronics.

**tristate_float**
Place a signal on an I/O pin only when enabled, similar to a tristate buffer in electronics.

**watchdog**
Monitor one to thirty-two inputs for a *heartbeat*.

### 12.9.3 HAL API calls

hal_add_funct_to_thread.3hal
hal_bit_t.3hal
hal_create_thread.3hal
hal_del_funct_from_thread.3hal
hal_exit.3hal
hal_export_funct.3hal
hal_float_t.3hal
hal_get_lock.3hal
hal_init.3hal
hal_link.3hal
hal_malloc.3hal
hal_param_bit_new.3hal
hal_param_bit_newf.3hal
hal_param_float_new.3hal
hal_param_float_newf.3hal
hal_param_new.3hal
hal_param_s32_new.3hal
hal_param_s32_newf.3hal
hal_param_u32_new.3hal
hal_param_u32_newf.3hal
hal_parport.3hal
hal_pin_bit_new.3hal
hal_pin_bit_newf.3hal
hal_pin_float_new.3hal
hal_pin_float_newf.3hal
hal_pin_new.3hal
hal_pin_s32_new.3hal
hal_pin_s32_newf.3hal
hal_pin_u32_new.3hal
hal_pin_u32_newf.3hal
hal_ready.3hal
hal_s32_t.3hal
hal_set_constructor.3hal
hal_set_lock.3hal
hal_signal_delete.3hal
hal_signal_new.3hal
hal_start_threads.3hal
hal_type_t.3hal
hal_u32_t.3hal
hal_unlink.3hal
intro.3hal
undocumented.3hal

### 12.9.4 RTAPI calls

EXPORT_FUNCTION.3rtapi
MODULE_AUTHOR.3rtapi
MODULE_DESCRIPTION.3rtapi
MODULE_LICENSE.3rtapi
RTAPI_MP_ARRAY_INT.3rtapi
RTAPI_MP_ARRAY_LONG.3rtapi
RTAPI_MP_ARRAY_STRING.3rtapi
RTAPI_MP_INT.3rtapi
RTAPI_MP_LONG.3rtapi

```
RTAPI_MP_STRING.3rtapi
intro.3rtapi
rtapi_app_exit.3rtapi
rtapi_app_main.3rtapi
rtapi_clock_set_period.3rtapi
rtapi_delay.3rtapi
rtapi_delay_max.3rtapi
rtapi_exit.3rtapi
rtapi_get_clocks.3rtapi
rtapi_get_msg_level.3rtapi
rtapi_get_time.3rtapi
rtapi_inb.3rtapi
rtapi_init.3rtapi
rtapi_module_param.3rtapi
RTAPI_MP_ARRAY_INT.3rtapi
RTAPI_MP_ARRAY_LONG.3rtapi
RTAPI_MP_ARRAY_STRING.3rtapi
RTAPI_MP_INT.3rtapi
RTAPI_MP_LONG.3rtapi
RTAPI_MP_STRING.3rtapi
rtapi_mutex.3rtapi
rtapi_outb.3rtapi
rtapi_print.3rtap
rtapi_prio.3rtapi
rtapi_prio_highest.3rtapi
rtapi_prio_lowest.3rtapi
rtapi_prio_next_higher.3rtapi
rtapi_prio_next_lower.3rtapi
rtapi_region.3rtapi
rtapi_release_region.3rtapi
rtapi_request_region.3rtapi
rtapi_set_msg_level.3rtapi
rtapi_shmem.3rtapi
rtapi_shmem_delete.3rtapi
rtapi_shmem_getptr.3rtapi
rtapi_shmem_new.3rtapi
rtapi_snprintf.3rtapi
rtapi_task_delete.3rtpi
rtapi_task_new.3rtapi
rtapi_task_pause.3rtapi
rtapi_task_resume.3rtapi
rtapi_task_start.3rtapi
rtapi_task_wait.3rtapi
```

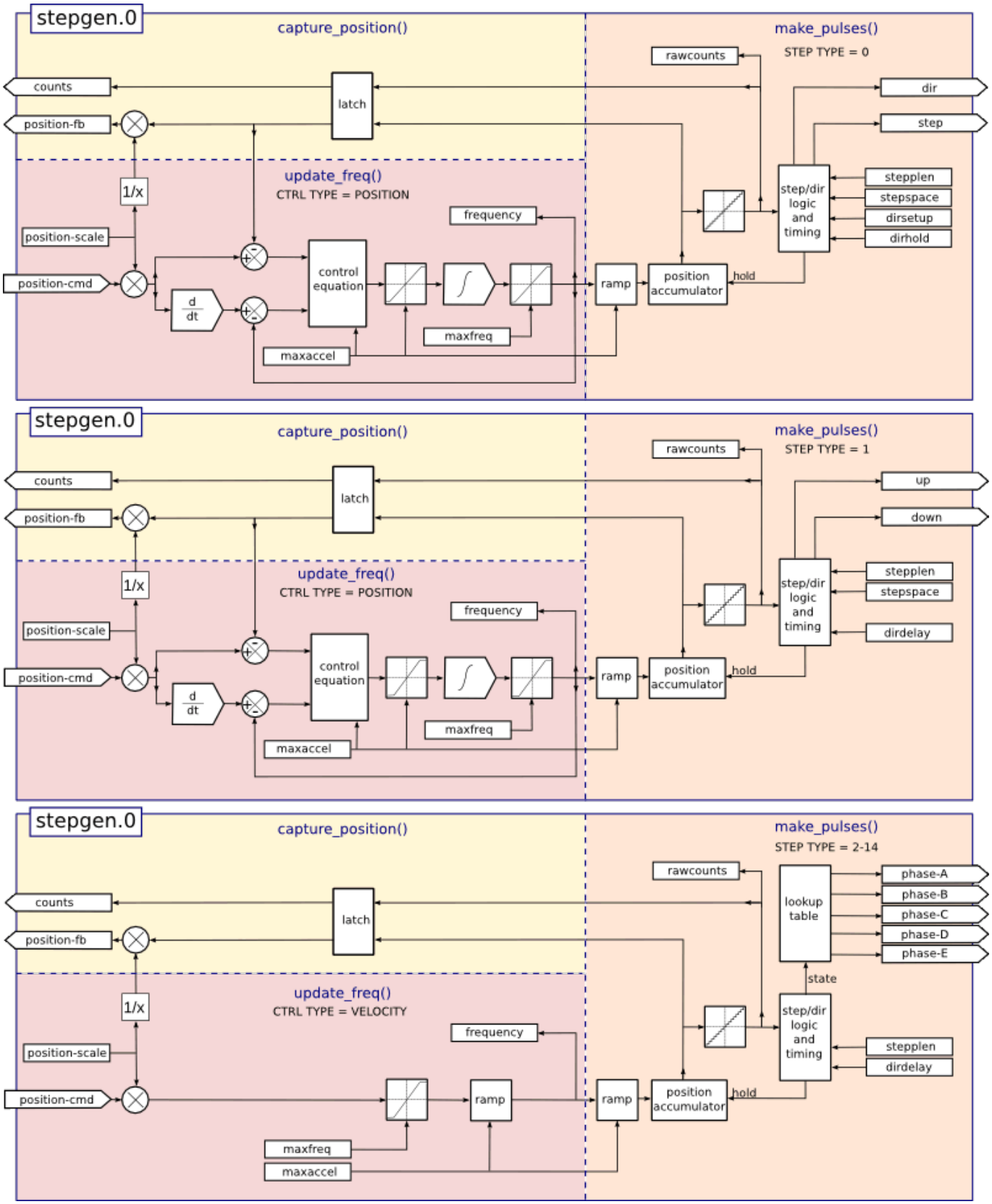
## 12.10 HAL Component Descriptions

### 12.10.1 Stepgen

This component provides software based generation of step pulses in response to position or velocity commands. In position mode, it has a built in pre-tuned position loop, so PID tuning is not required. In velocity mode, it drives a motor at the commanded speed, while obeying velocity and acceleration limits. It is a realtime component only, and depending on CPU speed, etc, is capable of maximum step rates of 10kHz to perhaps 50kHz. The step pulse generator block diagram shows three block diagrams, each is a single step pulse generator. The first diagram is for step type *0*, (step and direction). The second is for step type *1* (up/down, or pseudo-PWM), and the third is for step types 2 through 14 (various stepping patterns). The first two diagrams

show position mode control, and the third one shows velocity mode. Control mode and step type are set independently, and any combination can be selected.

.Step Pulse Generator Block Diagram position mode



### Installing

```
halcmd: loadrt stepgen step_type=<type-array> [ctrl_type=<ctrl_array>]
```

*<type-array>* is a series of comma separated decimal integers. Each number causes a single step pulse generator to be loaded, the value of the number determines the stepping type. *<ctrl_array>* is a comma separated series of *p* or *v* characters, to specify position or velocity mode. *ctrl_type* is optional, if ommitted, all of the step generators will be position mode.

For example:

```
halcmd: loadrt stepgen step_type=0,0,2 ctrl_type=p,p,v
```

will install three step generators. The first two use step type *0* (step and direction) and run in position mode. The last one uses step type *2* (quadrature) and runs in velocity mode. The default value for *<config-array>* is *0,0,0* which will install three type *0* (step/dir) generators. The maximum number of step generators is 8 (as defined by MAX_CHAN in stepgen.c). Each generator is independent, but all are updated by the same function(s) at the same time. In the following descriptions, *<chan>* is the number of a specific generator. The first generator is number 0. .Removing

```
halcmd: unloadrt stepgen
```

**Pins** Each step pulse generator will have only some of these pins, depending on the step type and control type selected.

- *(float) stepgen.<chan>.position-cmd* - Desired motor position, in position units (position mode only).
- *(float) stepgen.<chan>.velocity-cmd* - Desired motor velocity, in position units per second (velocity mode only).
- *(s32) stepgen.<chan>.counts* - Feedback position in counts, updated by *capture_position()*.
- *(float) stepgen.<chan>.position-fb* - Feedback position in position units, updated by *capture_position()*.
- *(bit) stepgen.<chan>.enable* - Enables output steps - when false, no steps are generated.
- *(bit) stepgen.<chan>.step* - Step pulse output (step type 0 only).
- *(bit) stepgen.<chan>.dir* - Direction output (step type 0 only).
- *(bit) stepgen.<chan>.up* - UP pseudo-PWM output (step type 1 only).
- *(bit) stepgen.<chan>.down* - DOWN pseudo-PWM output (step type 1 only).
- *(bit) stepgen.<chan>.phase-A* - Phase A output (step types 2-14 only).
- *(bit) stepgen.<chan>.phase-B* - Phase B output (step types 2-14 only).
- *(bit) stepgen.<chan>.phase-C* - Phase C output (step types 3-14 only).
- *(bit) stepgen.<chan>.phase-D* - Phase D output (step types 5-14 only).
- *(bit) stepgen.<chan>.phase-E* - Phase E output (step types 11-14 only).

PARAMETERS

- *(float) stepgen.<chan>.position-scale* - Steps per position unit. This parameter is used for both output and feedback.
- *(float) stepgen.<chan>.maxvel* - Maximum velocity, in position units per second. If 0.0, has no effect.
- *(float) stepgen.<chan>.maxaccel* - Maximum accel/decel rate, in positions units per second squared. If 0.0, has no effect.
- *(float) stepgen.<chan>.frequency* - The current step rate, in steps per second.
- *(float) stepgen.<chan>.steplen* - Length of a step pulse (step type 0 and 1) or minimum time in a given state (step types 2-14), in nano-seconds.

- *(float) stepgen.<chan>.stepspace* - Minimum spacing between two step pulses (step types 0 and 1 only), in nano-seconds. Set to 0 to enable the stepgen *doublefreq* function. To use *doublefreq* the parport reset function must be enabled.
- *(float) stepgen.<chan>.dirsetup* - Minimum time from a direction change to the beginning of the next step pulse (step type 0 only), in nanoseconds.
- *(float) stepgen.<chan>.dirhold* - Minmum time from the end of a step pulse to a direction change (step type 0 only), in nanoseconds.
- *(float) stepgen.<chan>.dirdelay* - Minmum time any step to a step in the opposite direction (step types 1-14 only), in nano-seconds.
- *(s32) stepgen.<chan>.rawcounts* - The raw feedback count, updated by *make_pulses()*.

In position mode, the values of maxvel and maxaccel are used by the internal position loop to avoid generating step pulse trains that the motor cannot follow. When set to values that are appropriate for the motor, even a large instantaneous change in commanded position will result in a smooth trapezoidal move to the new location. The algorithm works by measuring both position error and velocity error, and calculating an acceleration that attempts to reduce both to zero at the same time. For more details, including the contents of the *control equation* box, consult the code.

In velocity mode, maxvel is a simple limit that is applied to the commanded velocity, and maxaccel is used to ramp the actual frequency if the commanded velocity changes abruptly. As in position mode, proper values for these parameters ensure that the motor can follow the generated pulse train.

**Step Type 0** Step type 0 is the standard step and direction type. When configured for step type 0, there are four extra parameters that determine the exact timing of the step and direction signals. In the following figure the meaning of these parameters is shown. The parameters are in nanoseconds, but will be rounded up to an integer multiple of the thread period for the threaed that calls *make_pulses()*. For example, if *make_pulses()* is called every 16 us, and steplen is 20000, then the step pulses will be 2 x 16 = 32 us long. The default value for all four of the parameters is 1ns, but the automatic rounding takes effect the first time the code runs. Since one step requires *steplen* ns high and *stepspace* ns low, the maximum frequency is 1,000,000,000 divided by *(steplen+stepspace)*. If *maxfreq* is set higher than that limit, it will be lowered automatically. If maxfreq is zero, it will remain zero, but the output frequency will still be limited.

When using the parallel port driver the step frequency can be doubled using the parport reset function together with stepgen's *doublefreq* setting.

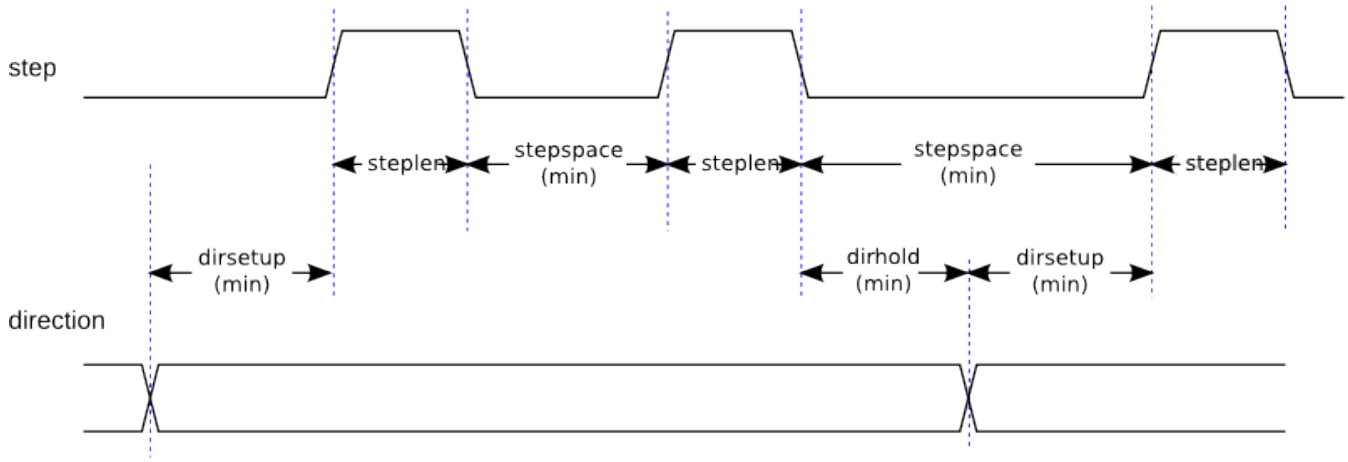


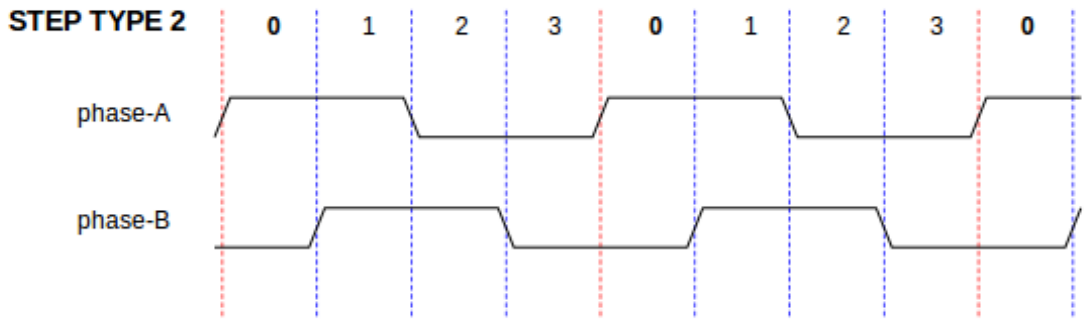

Figure 12.21: Step and Direction Timing

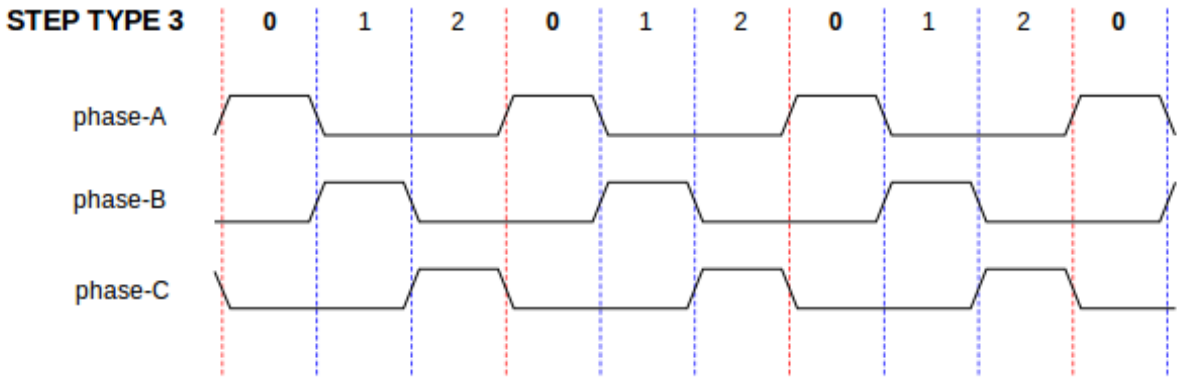
**Step Type 1** Step type 1 has two outputs, up and down. Pulses appear on one or the other, depending on the direction of travel. Each pulse is *steplen* ns long, and the pulses are separated by at least *stepspace* ns. The maximum frequency is the same as for step type 0. If *maxfreq* is set higher than the limit it will be lowered. If *maxfreq* is zero, it will remain zero but the output frequency will still be limited.

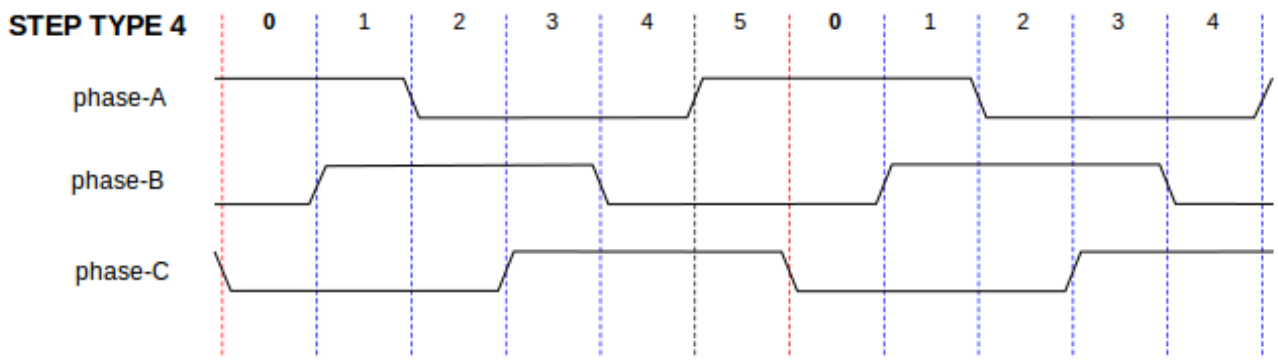
**Warning**
Do not use the parport reset function with step types 2 - 14. Unexpected results can happen.

**Step Type 2 - 14** Step types 2 through 14 are state based, and have from two to five outputs. On each step, a state counter is incremented or decremented. The Two-and-Three-Phase, Four-Phase, and Five-Phase show the output patterns as a function of the state counter. The maximum frequency is 1,000,000,000 divided by *steplen*, and as in the other modes, *maxfreq* will be lowered if it is above the limit.

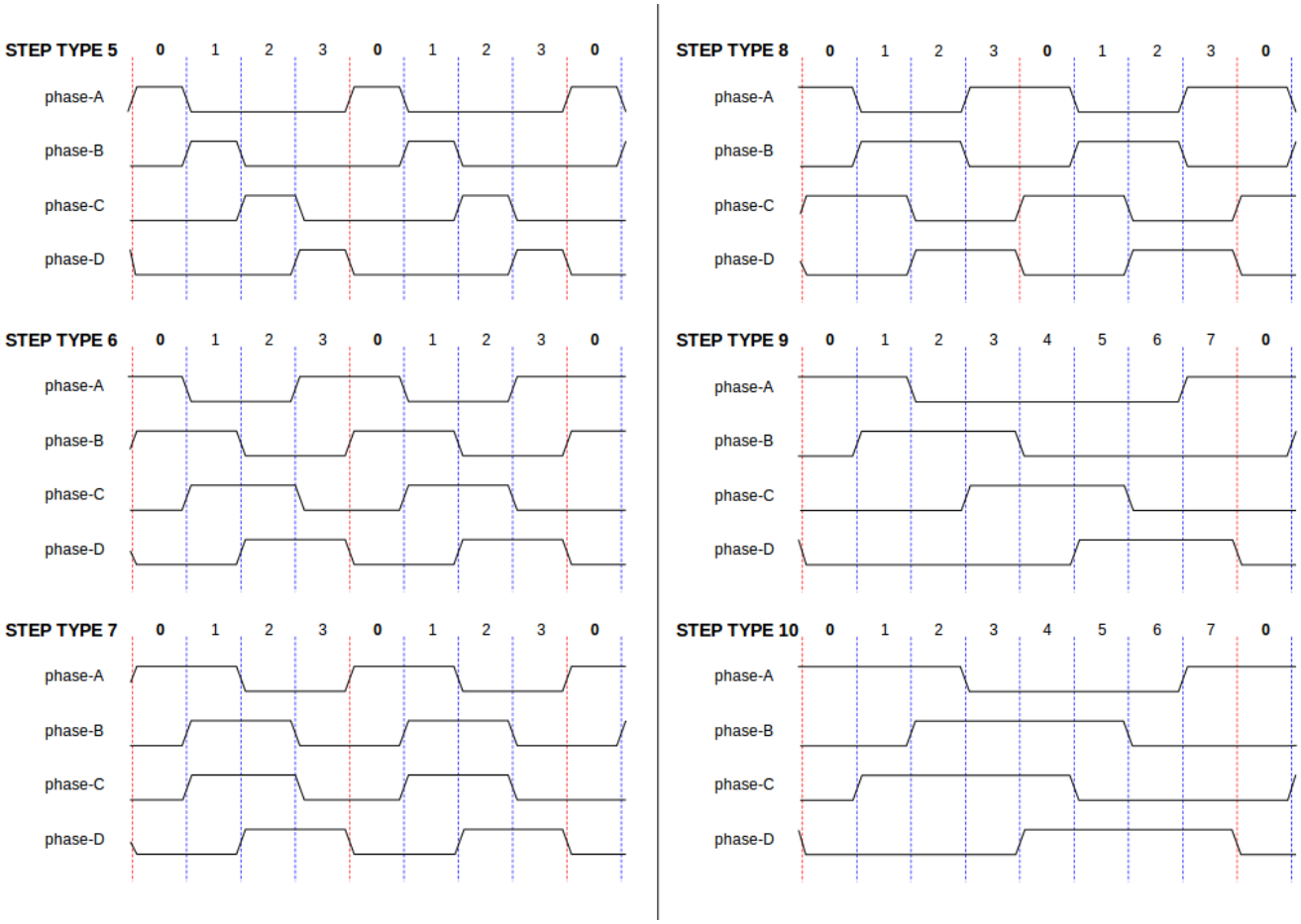
**Two-and-Three-Phase Step Types**



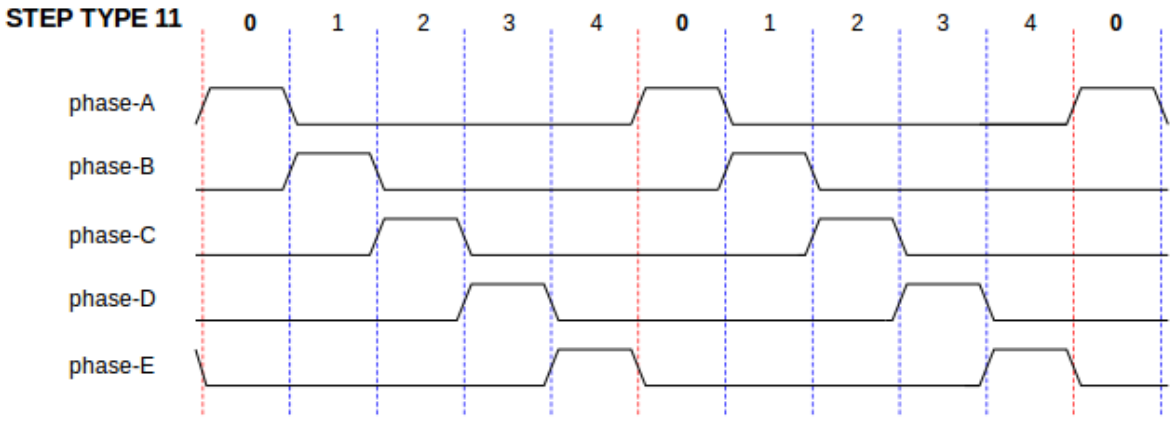



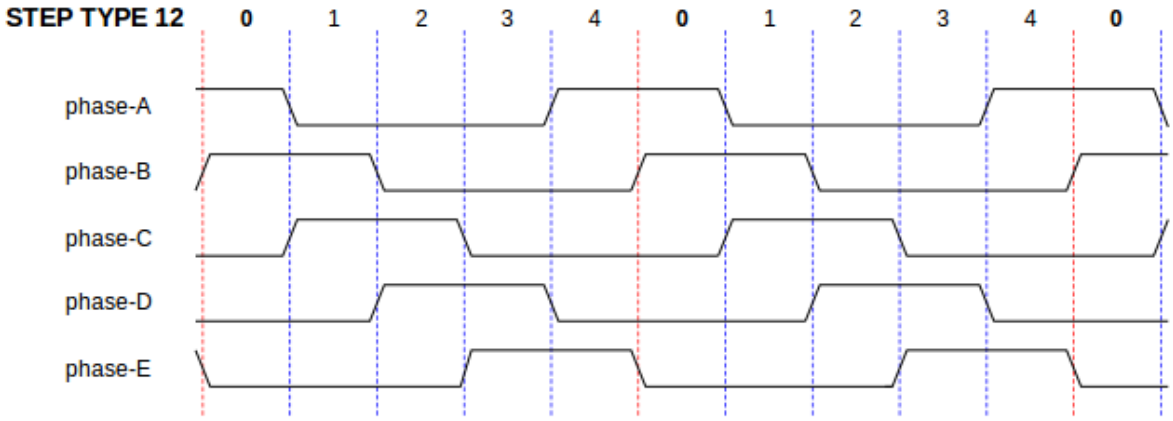



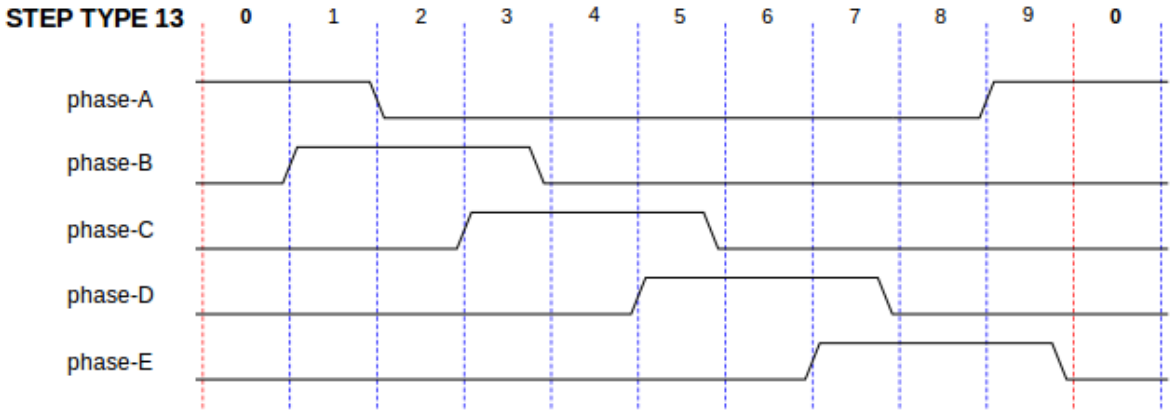

**Four-Phase Step Types**

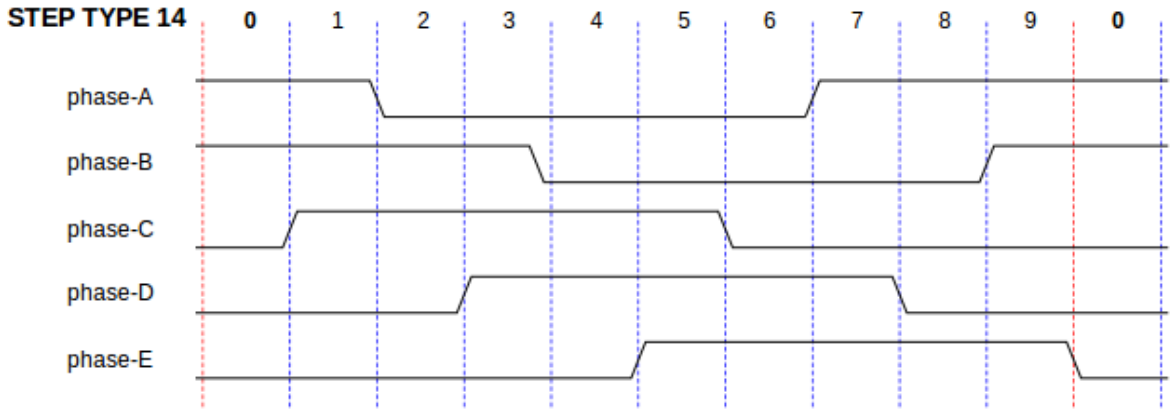



**Five-Phase Step Types**


STEP TYPE 11
0 1 2 3 4 0 1 2 3 4 0
phase-A
phase-B
phase-C
phase-D
phase-E


STEP TYPE 12
0 1 2 3 4 0 1 2 3 4 0
phase-A
phase-B
phase-C
phase-D
phase-E


STEP TYPE 13
0 1 2 3 4 5 6 7 8 9 0
phase-A
phase-B
phase-C
phase-D
phase-E


STEP TYPE 14
0 1 2 3 4 5 6 7 8 9 0
phase-A
phase-B
phase-C
phase-D
phase-E

**Functions** The component exports three functions. Each function acts on all of the step pulse generators - running different generators in different threads is not supported.

- *(funct) stepgen.make-pulses* - High speed function to generate and count pulses (no floating point).
- *(funct) stepgen.update-freq* - Low speed function does position to velocity conversion, scaling and limiting.
- *(funct) stepgen.capture-position* - Low speed function for feedback, updates latches and scales position.

The high speed function *stepgen.make-pulses* should be run in a very fast thread, from 10 to 50 us depending on the capabilities of the computer. That thread's period determines the maximum step frequency, since *steplen*, *stepspace*, *dirsetup*, *dirhold*, and *dirdelay* are all rounded up to a integer multiple of the thread periond in nanoseconds. The other two functions can be called at a much lower rate.

### 12.10.2 PWMgen

This component provides software based generation of PWM (Pulse Width Modulation) and PDM (Pulse Density Modulation) waveforms. It is a realtime component only, and depending on CPU speed, etc, is capable of PWM frequencies from a few hundred Hertz at pretty good resolution, to perhaps 10KHz with limited resolution.

**Installing**

```
loadrt pwmgen output_type=<config-array>
```

The <*config-array*> is a series of comma separated decimal integers. Each number causes a single PWM generator to be loaded, the value of the number determines the output type. The following example will install three PWM generators. There is no default value, if <*config-array*> is not specified, no PWM generators will be installed. The maximum number of frequency generators is 8 (as defined by MAX_CHAN in pwmgen.c). Each generator is independent, but all are updated by the same function(s) at the same time. In the following descriptions, <*chan*> is the number of a specific generator. The first generator is number 0.

**Example**

```
loadrt pwmgen output_type=0,1,2
```

**Removing**

```
unloadrt pwmgen
```

**Output Types** The PWM generator supports three different *output types*.

- *Output type 0* - PWM output pin only. Only positive commands are accepted, negative values are treated as zero (and will be affected by the parameter *min-dc* if it is non-zero).
- *Output type 1* - PWM/PDM and direction pins. Positive and negative inputs will be output as positive and negative PWM. The direction pin is false for positive commands, and true for negative commands. If your control needs positive PWM for both CW and CCW use the abs component to convert your PWM signal to positive value when a negative input is input.
- *Output type 2* - UP and DOWN pins. For positive commands, the PWM signal appears on the up output, and the down output remains false. For negative commands, the PWM signal appears on the down output, and the up output remains false. Output type 2 is suitable for driving most H-bridges.

**Pins** Each PWM generator will have the following pins:

- *(float) pwmgen.<chan>.value* - Command value, in arbitrary units. Will be scaled by the *scale* parameter (see below).
- *(bit) pwmgen.<chan>.enable* - Enables or disables the PWM generator outputs.

Each PWM generator will also have some of these pins, depending on the output type selected:

- *(bit) pwmgen.<chan>.pwm* - PWM (or PDM) output, (output types 0 and 1 only).
- *(bit) pwmgen.<chan>.dir* - Direction output (output type 1 only).
- *(bit) pwmgen.<chan>.up* - PWM/PDM output for positive input value (output type 2 only).
- *(bit) pwmgen.<chan>.down* - PWM/PDM output for negative input value (output type 2 only).

PARAMETERS

- *(float) pwmgen.<chan>.scale* - Scaling factor to convert *value* from arbitrary units to duty cycle. For example if scale is set to 4000 and the input value passed to the pwmgen.<chan>.value is 4000 then it will be 100% duty-cycle (always on). If the value is 2000 then it will be a 50% 25Hz square wave.
- *(float) pwmgen.<chan>.pwm-freq* - Desired PWM frequency, in Hz. If 0.0, generates PDM instead of PWM. If set higher than internal limits, next call of *update_freq()* will set it to the internal limit. If non-zero, and *dither* is false, next call of *update_freq()* will set it to the nearest integer multiple of the *make_pulses()* function period.
- *(bit) pwmgen.<chan>.dither-pwm* - If true, enables dithering to achieve average PWM frequencies or duty cycles that are unobtainable with pure PWM. If false, both the PWM frequency and the duty cycle will be rounded to values that can be achieved exactly.
- *(float) pwmgen.<chan>.min-dc* - Minimum duty cycle, between 0.0 and 1.0 (duty cycle will go to zero when disabled, regardless of this setting).
- *(float) pwmgen.<chan>.max-dc* - Maximum duty cycle, between 0.0 and 1.0.
- *(float) pwmgen.<chan>.curr-dc* - Current duty cycle - after all limiting and rounding (read only).

**Functions** The component exports two functions. Each function acts on all of the PWM generators - running different generators in different threads is not supported.

- *(funct) pwmgen.make-pulses* - High speed function to generate PWM waveforms (no floating point). The high speed function *pwmgen.make-pulses* should be run in the base (fastest) thread, from 10 to 50 us depending on the capabilities of the computer. That thread's period determines the maximum PWM carrier frequency, as well as the resolution of the PWM or PDM signals. If the base thread is 50,000nS then every 50uS the module decides if it is time to change the state of the output. At 50% duty cycle and 25Hz PWM frequency this means that the output changes state every (1 / 25) seconds / 50uS * 50% = 400 iterations. This also means that you have a 800 possible duty cycle values (without dithering)
- *(funct) pwmgen.update* - Low speed function to scale and limit value and handle other parameters. This is the function of the module that does the more complicated mathematics to work out how many base-periods the output should be high for, and how many it should be low for.

### 12.10.3 Encoder

This component provides software based counting of signals from quadrature encoders. It is a realtime component only, and depending on CPU speed, latency, etc, is capable of maximum count rates of 10kHz to perhaps up to 50kHz.

The base thread should be 1/2 count speed to allow for noise and timing variation. For example if you have a 100 pulse per revolution encoder on the spindle and your maximnum RPM is 3000 the maximum base thread should be 25 us. A 100 pulse per revolution encoder will have 400 counts. The spindle speed of 3000 RPM = 50 RPS (revolutions per second). 400 * 50 = 20,000 counts per second or 50 us between counts.

The Encoder Counter Block Diagram is a block diagram of one channel of an encoder counter.

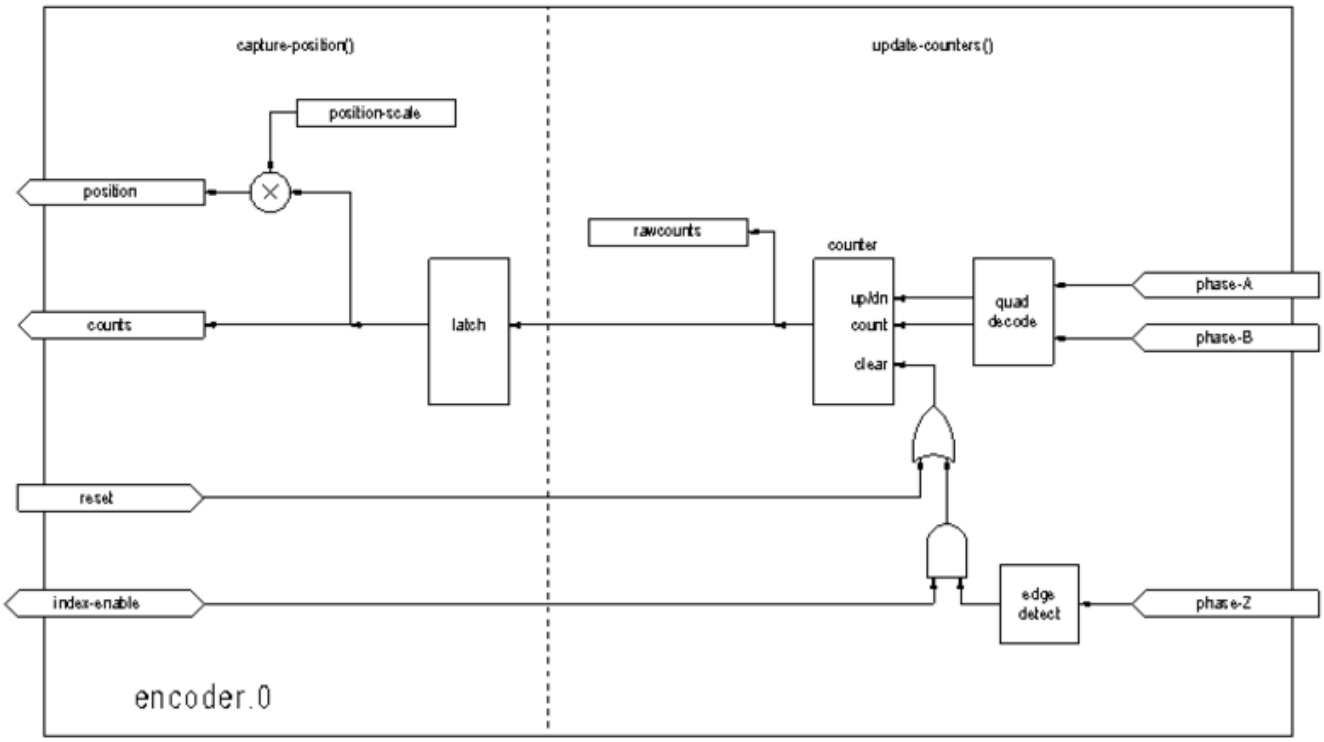



Figure 12.22: Encoder Counter Block Diagram

### Installing

```
halcmd: loadrt encoder [num_chan=<counters>]
```

*<counters>* is the number of encoder counters that you want to install. If *numchan* is not specified, three counters will be installed. The maximum number of counters is 8 (as defined by MAX_CHAN in encoder.c). Each counter is independent, but all are updated by the same function(s) at the same time. In the following descriptions, *<chan>* is the number of a specific counter. The first counter is number 0.

### Removing

```
halcmd: unloadrt encoder
```

PINS

- *encoder.<chan>.counter-mode* (bit, I/O) (default: FALSE) - Enables counter mode. When true, the counter counts each rising edge of the phase-A input, ignoring the value on phase-B. This is useful for counting the output of a single channel (non-quadrature) sensor. When false, it counts in quadrature mode.
- *encoder.<chan>.counts* (s32, Out) - Position in encoder counts.
- *encoder.<chan>.counts-latched* (s32, Out) - Not used at this time.

- *encoder.<chan>.index-enable* (bit, I/O) - When True, *counts* and *position are* reset to zero on next rising edge of Phase Z. At the same time, *index-enable* is reset to zero to indicate that the rising edge has occoured. The *index-enable* pin is bi-directional. If *index-enable* is False, the Phase Z channel of the encoder will be ignored, and the counter will count normally. The encoder driver will never set *index-enable* True. However, some other component may do so.
- *encoder.<chan>.latch-falling* (bit, In) (default: TRUE) - Not used at this time.
- *encoder.<chan>.latch-input* (bit, In) (default: TRUE) - Not used at this time.
- *encoder.<chan>.latch-rising* (bit, In) - Not used at this time.
- *encoder.<chan>.min-speed-estimate* (float, in) - Determine the minimum true velocity magnitude at which velocity will be estimated as nonzero and postition-interpolated will be interpolated. The units of *min-speed-estimate* are the same as the units of *velocity* . Scale factor, in counts per length unit. Setting this parameter too low will cause it to take a long time for velocity to go to 0 after encoder pulses have stopped arriving.
- *encoder.<chan>.phase-A* (bit, In) - Phase A of the quadrature encoder signal.
- *encoder.<chan>.phase-B* (bit, In) - Phase B of the quadrature encoder signal.
- *encoder.<chan>.phase-Z* (bit, In) - Phase Z (index pulse) of the quadrature encoder signal.
- *encoder.<chan>.position* (float, Out) - Position in scaled units (see *position-scale*).
- *encoder.<chan>.position-interpolated* (float, Out) - Position in scaled units, interpolated between encoder counts. The *position-interpolated* attempts to interpolate between encoder counts, based on the most recently measured velocity. Only valid when velocity is approximately constant and above *min-speed-estimate*. Do not use for position control, since its value is incorrect at low speeds, during direction reversals, and during speed changes. However, it allows a low ppr encoder (including a one pulse per revolution *encoder*) to be used for lathe threading, and may have other uses as well.
- *encoder.<chan>.position-latched (float, Out)* - Not used at this time.
- *encoder.<chan>.position-scale (float, I/O)* - Scale factor, in counts per length unit. For example, if position-scale is 500, then 1000 counts of the encoder will be reported as a position of 2.0 units.
- *encoder.<chan>.rawcounts (s32, In)* - The raw count, as determined by update-counters. This value is updated more frequently than counts and position. It is also unaffected by reset or the index pulse.
- *encoder.<chan>.reset* (bit, In) - When True, force *counts* and *position* to zero immediately.
- *encoder.<chan>.velocity* (float, Out) - Velocity in scaled units per second. *encoder* uses an algorithm that greatly reduces quantization noise as compared to simply differentiating the *position* output. When the magnitude of the true velocity is below min-speed-estimate, the velocity output is 0.
- *encoder.<chan>.x4-mode (bit, I/O) (default: TRUE)* - Enables times-4 mode. When true, the counter counts each edge of the quadrature waveform (four counts per full cycle). When false, it only counts once per full cycle. In counter-mode, this parameter is ignored. The 1x mode is useful for some jogwheels.

PARAMETERS

- *encoder.<chan>.capture-position.time (s32, RO)*
- *encoder.<chan>.capture-position.tmax (s32, RW)*
- *encoder.<chan>.update-counters.time (s32, RO)*
- *encoder.<chan>.update-counter.tmax (s32, RW)*

**Functions** The component exports two functions. Each function acts on all of the encoder counters - running different counters in different threads is not supported.

- *(funct) encoder.update-counters* - High speed function to count pulses (no floating point).
- *(funct) encoder.capture-position* - Low speed function to update latches and scale position.

### 12.10.4 PID

This component provides Proportional/Integral/Derivative control loops. It is a realtime component only. For simplicity, this discussion assumes that we are talking about position loops, however this component can be used to implement other feedback loops such as speed, torch height, temperature, etc. The PID Loop Block Diagram is a block diagram of a single PID loop.

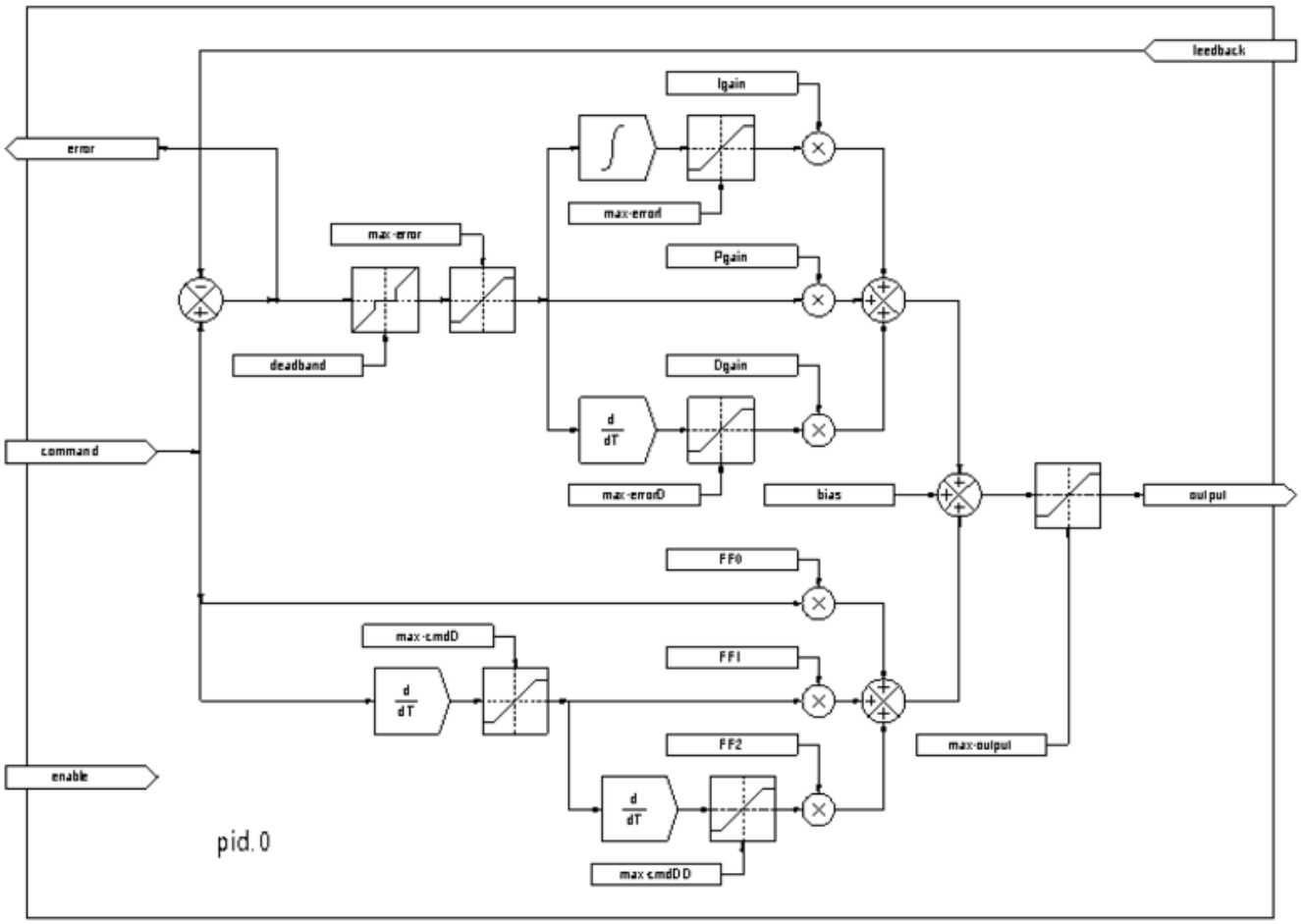



Figure 12.23: PID Loop Block Diagram

**Installing**

```
halcmd: loadrt pid [num_chan=<loops>] [debug=1]
```

*<loops>* is the number of PID loops that you want to install. If *numchan* is not specified, one loop will be installed. The maximum number of loops is 16 (as defined by MAX_CHAN in pid.c). Each loop is completely independent. In the following descriptions, *<loopnum>* is the loop number of a specific loop. The first loop is number 0.

If *debug=1* is specified, the component will export a few extra pins that may be useful during debugging and tuning. By default, the extra pins are not exported, to save shared memory space and avoid cluttering the pin list.

**Removing**

```
halcmd: unloadrt pid
```

**Pins** The three most important pins are

- *(float) pid.<loopnum>.command* - The desired position, as commanded by another system component.
- *(float) pid.<loopnum>.feedback* - The present position, as measured by a feedback device such as an encoder.
- *(float) pid.<loopnum>.output* - A velocity command that attempts to move from the present position to the desired position.

For a position loop, *command* and *feedback* are in position units. For a linear axis, this could be inches, mm, meters, or whatever is relevant. Likewise, for an angular axis, it could be degrees, radians, etc. The units of the *output* pin represent the change needed to make the feedback match the command. As such, for a position loop *Output* is a velocity, in inches/sec, mm/sec, degrees/sec, etc. Time units are always seconds, and the velocity units match the position units. If command and feedback are in meters, then output is in meters per second.

Each loop has two pins which are used to monitor or control the general operation of the component.

- *(float) pid.<loopnum>.error* - Equals *.command* minus *.feedback*.
- *(bit) pid.<loopnum>.enable* - A bit that enables the loop. If *.enable* is false, all integrators are reset, and the output is forced to zero. If *.enable* is true, the loop operates normally.

Pins used to report saturation. Saturation occurs when the output of the PID block is at its maximum or minimum limit.

- *(bit) pid.<loopnum>.saturated* - True when output is saturated.
- *(float) pid.<loopnum>.saturated_s* - The time the output has been saturated.
- *(s32) pid.<loopnum>.saturated_count* - The time the output has been saturated.

The PID gains, limits, and other *tunable* features of the loop are available as pins so that they can be adjusted dynamically for more advanced tuning possibilities.

- *(float) pid.<loopnum>.Pgain* - Proportional gain
- *(float) pid.<loopnum>.Igain* - Integral gain
- *(float) pid.<loopnum>.Dgain* - Derivative gain
- *(float) pid.<loopnum>.bias* - Constant offset on output
- *(float) pid.<loopnum>.FF0* - Zeroth order feedforward - output proportional to command (position).
- *(float) pid.<loopnum>.FF1* - First order feedforward - output proportional to derivative of command (velocity).
- *(float) pid.<loopnum>.FF2* - Second order feedforward - output proportional to 2nd derivative of command (acceleration).
- *(float) pid.<loopnum>.deadband* - Amount of error that will be ignored
- *(float) pid.<loopnum>.maxerror* - Limit on error
- *(float) pid.<loopnum>.maxerrorI* - Limit on error integrator
- *(float) pid.<loopnum>.maxerrorD* - Limit on error derivative
- *(float) pid.<loopnum>.maxcmdD* - Limit on command derivative
- *(float) pid.<loopnum>.maxcmdDD* - Limit on command 2nd derivative
- *(float) pid.<loopnum>.maxoutput* - Limit on output value

If *debug=1* was specified when the component was installed, four additional pins will be exported:

- *(float) pid.<loopnum>.errorI* - Integral of error.
- *(float) pid.<loopnum>.errorD* - Derivative of error.

- *(float) pid.<loopnum>.commandD* - Derivative of the command.
- *(float) pid.<loopnum>.commandDD* - 2nd derivative of the command.

**Functions** The component exports one function for each PID loop. This function performs all the calculations needed for the loop. Since each loop has its own function, individual loops can be included in different threads and execute at different rates.

- *(funct) pid.<loopnum>.do_pid_calcs* - Performs all calculations for a single PID loop.

If you want to understand the exact algorithm used to compute the output of the PID loop, refer to figure PID Loop Block Diagram, the comments at the beginning of *emc2/src/hal/components/pid.c* , and of course to the code itself. The loop calculations are in the C function *calc_pid()*.

### 12.10.5 Simulated Encoder

The simulated encoder is exactly that. It produces quadrature pulses with an index pulse, at a speed controlled by a HAL pin. Mostly useful for testing.

**Installing**

```
halcmd: loadrt sim-encoder num_chan=<number>
```

*<number>* is the number of encoders that you want to simulate. If not specified, one encoder will be installed. The maximum number is 8 (as defined by MAX_CHAN in sim_encoder.c).

**Removing**

```
halcmd: unloadrt sim-encoder
```

PINS

- *(float) sim-encoder.<chan-num>.speed* - The speed command for the simulated shaft.
- *(bit) sim-encoder.<chan-num>.phase-A* - Quadrature output.
- *(bit) sim-encoder.<chan-num>.phase-B* - Quadrature output.
- *(bit) sim-encoder.<chan-num>.phase-Z* - Index pulse output.

When *.speed* is positive, *.phase-A* leads *.phase-B*.

PARAMETERS

- *(u32) sim-encoder.<chan-num>.ppr* - Pulses Per Revolution.
- *(float) sim-encoder.<chan-num>.scale* - Scale Factor for *speed*. The default is 1.0, which means that *speed* is in revolutions per second. Change to 60 for RPM, to 360 for degrees per second, 6.283185 for radians per seconed, etc.

Note that pulses per revolution is not the same as counts per revolution. A pulse is a complete quadrature cycle. Most encoder counters will count four times during one complete cycle.

**Functions** The component exports two functions. Each function affects all simulated encoders.

- *(funct) sim-encoder.make-pulses* - High speed function to generate quadrature pulses (no floating point).
- *(funct) sim-encoder.update-speed* - Low speed function to read *speed*, do scaling, and set up *make-pulses*.

### 12.10.6 Debounce

Debounce is a realtime component that can filter the glitches created by mechanical switch contacts. It may also be useful in other applications where short pulses are to be rejected.

**Installing**

```
halcmd: loadrt debounce cfg=<config-string>
```

*<config-string>* is a series of comma separated decimal integers. Each number installs a group of identical debounce filters, the number determines how many filters are in the group.

For example:

```
halcmd: loadrt debounce cfg=1,4,2
```

will install three groups of filters. Group 0 contains one filter, group 1 contains four, and group 2 contains two filters. The default value for *<config-string>* is *"1"* which will install a single group containing a single filter. The maximum number of groups 8 (as defined by MAX_GROUPS in debounce.c). The maximum number of filters in a group is limited only by shared memory space. Each group is completely independent. All filters in a single group are identical, and they are all updated by the same function at the same time. In the following descriptions, *<G>* is the group number and *<F>* is the filter number within the group. The first filter is group 0, filter 0.

**Removing**

```
halcmd: unloadrt debounce
```

**Pins** Each individual filter has two pins.

- *(bit) debounce.<G>.<F>.in* - Input of filter *<F>* in group *<G>*.
- *(bit) debounce.<G>.<F>.out* - Output of filter *<F>* in group *<G>*.

**Parameters** Each group of filters has one parameter[10].

- *(s32) debounce.<G>.delay* - Filter delay for all filters in group *<G>*.

The filter delay is in units of thread periods. The minimum delay is zero. The output of a zero delay filter exactly follows its input - it doesn't filter anything. As *delay* increases, longer and longer glitches are rejected. If *delay* is 4, all glitches less than or equal to four thread periods will be rejected.

**Functions** Each group of filters has one function, which updates all the filters in that group *simultaneously*. Different groups of filters can be updated from different threads at different periods.

- *(funct) debounce.<G>* - Updates all filters in group *<G>*.

### 12.10.7 Siggen

Siggen is a realtime component that generates square, triangle, and sine waves. It is primarily used for testing.

**Installing**

```
halcmd: loadrt siggen [num_chan=<chans>]
```

[10] Each individual filter also has an internal state variable. There is a compile time switch that can export that variable as a parameter. This is intended for testing, and simply wastes shared memory under normal circumstances.

*<chans>* is the number of signal generators that you want to install. If *numchan* is not specified, one signal generator will be installed. The maximum number of generators is 16 (as defined by MAX_CHAN in siggen.c). Each generator is completely independent. In the following descriptions, *<chan>* is the number of a specific signal generator (the numbers start at 0).

**Removing**

```
halcmd: unloadrt siggen
```

**Pins** Each generator has five output pins.

- *(float) siggen.<chan>.sine* - Sine wave output.
- *(float) siggen.<chan>.cosine* - Cosine output.
- *(float) siggen.<chan>.sawtooth* - Sawtooth output.
- *(float) siggen.<chan>.triangle* - Triangle wave output.
- *(float) siggen.<chan>.square* - Square wave output.

All five outputs have the same frequency, amplitude, and offset.

In addition to the output pins, there are three control pins:

- *(float) siggen.<chan>.frequency* - Sets the frequency in Hertz, default value is 1 Hz.
- *(float) siggen.<chan>.amplitude* - Sets the peak amplitude of the output waveforms, default is 1.
- *(float) siggen.<chan>.offset* - Sets DC offset of the output waveforms, default is 0.

For example, if *siggen.0.amplitude* is 1.0 and *siggen.0.offset* is 0.0, the outputs will swing from -1.0 to +1.0. If *siggen.0.amplitude* is 2.5 and *siggen.0.offset* is 10.0, then the outputs will swing from 7.5 to 12.5.

**Parameters** None. [11]

FUNCTIONS

- *(funct) siggen.<chan>.update* - Calculates new values for all five outputs.

### 12.10.8 lut5

The lut5 component is a 5 input logic component based on a look up table.

- *lut5* does not require a floating point thread.

**Installing**

```
loadrt lut5 [count=N|names=name1[,name2...]]
addf lut5.N servo-thread | base-thread
setp lut5.N.function 0xN
```

**Computing Function** To compute the hexadecimal number for the function starting from the top put a 1 or 0 to indicate if that row would be true or false. Next write down every number in the output column starting from the top and writing them from right to left. This will be the binary number. Using a calculator with a program view like the one in Ubuntu enter the binary number and then convert it to hexadecimal and that will be the value for function.

[11] Prior to version 2.1, frequency, amplitude, and offset were parameters. They were changed to pins to allow control by other components.

Table 12.1: Look Up Table

| Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Output |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | |
| 0 | 0 | 0 | 0 | 1 | |
| 0 | 0 | 0 | 1 | 0 | |
| 0 | 0 | 0 | 1 | 1 | |
| 0 | 0 | 1 | 0 | 0 | |
| 0 | 0 | 1 | 0 | 1 | |
| 0 | 0 | 1 | 1 | 0 | |
| 0 | 0 | 1 | 1 | 1 | |
| 0 | 1 | 0 | 0 | 0 | |
| 0 | 1 | 0 | 0 | 1 | |
| 0 | 1 | 0 | 1 | 0 | |
| 0 | 1 | 0 | 1 | 1 | |
| 0 | 1 | 1 | 0 | 0 | |
| 0 | 1 | 1 | 0 | 1 | |
| 0 | 1 | 1 | 1 | 0 | |
| 0 | 1 | 1 | 1 | 1 | |
| 1 | 0 | 0 | 0 | 0 | |
| 1 | 0 | 0 | 0 | 1 | |
| 1 | 0 | 0 | 1 | 0 | |
| 1 | 0 | 0 | 1 | 1 | |
| 1 | 0 | 1 | 0 | 0 | |
| 1 | 0 | 1 | 0 | 1 | |
| 1 | 0 | 1 | 1 | 0 | |
| 1 | 0 | 1 | 1 | 1 | |
| 1 | 1 | 0 | 0 | 0 | |
| 1 | 1 | 0 | 0 | 1 | |
| 1 | 1 | 0 | 1 | 0 | |
| 1 | 1 | 0 | 1 | 1 | |
| 1 | 1 | 1 | 0 | 0 | |
| 1 | 1 | 1 | 0 | 1 | |
| 1 | 1 | 1 | 1 | 0 | |
| 1 | 1 | 1 | 1 | 1 | |

**Two Input Example** In the following table we have selected the output state for each line that we wish to be true.

Table 12.2: Look Up Table

| Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Output |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 |

Looking at the output column of our example we want the output to be on when Bit 0 or Bit 0 and Bit1 is on and nothing else. The binary number is *b1010* (rotate the output 90 degrees CW). Enter this number into the calculator then change the display to hexadecimal and the number needed for function is *0xa*. The hexadecimal prefix is *0x*.

## 12.11 HAL Examples

All of these examples assume you are starting with a stepconf based configuration and have two threads *base-thread* and *servo-thread*. The stepconf wizard will create an empty custom.hal and a custom_postgui.hal file. The custom.hal file will be loaded after the configuration HAL file and the custom_postgui.hal file is loaded after the GUI has been loaded.

### 12.11.1 Connecting Two Outputs

To connect two outputs to an input you can use the or2 component. The or2 works like this, if either input to or2 is on then the or2 output is on. If neither input to or2 is on the or2 output is off.

For example to have two pyvcp buttons both connected to one led.

**The .xml file**

```
<pyvcp>
  <button>
    <halpin>"button-1"</halpin>
    <text>"Button 1"</text>
  </button>

  <button>
    <halpin>"button-2"</halpin>
    <text>"Button 2"</text>
  </button>

  <led>
    <halpin>"led-1"</halpin>
    <size>50</size>
    <on_color>"green"</on_color>
    <off_color>"red"</off_color>
  </led>
</pyvcp>
```

**The postgui.hal file**

```
loadrt or2
addf or2.0 servo-thread
net button-1 or2.0.in0 <= pyvcp.button-1
net button-2 or2.0.in1 <= pyvcp.button-2
net led-1 pyvcp.led-1 <= or2.0.out
```

When you run this example in an Axis simulator created with the Stepconf Wizard you can open a terminal and see the pins created with *loadrt or2* by typing in *halcmd show pin or2* in the terminal

```
halcmd show pin or2
Component Pins:
Owner   Type  Dir         Value  Name
    22  bit   IN          FALSE  or2.0.in0 <== button-1
    22  bit   IN          FALSE  or2.0.in1 <== button-2
    22  bit   OUT         FALSE  or2.0.out ==> led-1
```

You can see from the hal command *show pin or2* that the *button-1* pin is connected to the *or2.0.in0* pin and from the direction arrow you can see that the button is and output and the *or2.0.in0* is an input. The output from or2 goes to the input of the led.

### 12.11.2 Manual Toolchange

In this example it is assumed that you're *rolling your own* configuration and wish to add the HAL Manual Toolchange window. The HAL Manual Toolchange is primarily useful if you have presettable tools and you store the offsets in the tool table. If you

need to touch off for each tool change then it is best just to split up your g code. To use the HAL Manual Toolchange window you basically have to load the hal_manualtoolchange component then send the iocontrol *tool change* to the hal_manualtoolchange *change* and send the hal_manualtoolchange *changed* back to the iocontrol *tool changed*. A pin is provided for an external input to indicate the tool change is complete.

This is an example of manual toolchange *with* the HAL Manual Toolchange component:

```
loadusr -W hal_manualtoolchange
net tool-change iocontrol.0.tool-change => hal_manualtoolchange.change
net tool-changed iocontrol.0.tool-changed <= hal_manualtoolchange.changed
net external-tool-changed hal_manualtoolchange.change_button <= parport.0.pin-12-in
net tool-number iocontrol.0.tool-prep-number => hal_manualtoolchange.number
net tool-prepare-loopback iocontrol.0.tool-prepare => iocontrol.0.tool-prepared
```

This is an example of manual toolchange *without* the HAL Manual Toolchange component:

```
net tool-number <= iocontrol.0.tool-prep-number
net tool-change-loopback iocontrol.0.tool.-change => iocontrol.0.tool-changed
net tool-prepare-loopback iocontrol.0.tool-prepare => iocontrol.0.tool-prepared
```

### 12.11.3 Compute Velocity

This example uses *ddt*, *mult2* and *abs* to compute the velocity of a single axis. For more information on the real time components see the man pages or the Realtime Components section (Section 12.9.2).

The first thing is to check your configuration to make sure you are not using any of the real time components all ready. You can do this by opening up the HAL Configuration window and look for the components in the pin section. If you are then find the .hal file that they are being loaded in and increase the counts and adjust the instance to the correct value. Add the following to your custom.hal file.

Load the realtime components.

```
loadrt ddt count=1
loadrt mult2 count=1
loadrt abs count=1
```

Add the functions to a thread so it will get updated.

```
addf ddt.0 servo-thread
addf mult2.0 servo-thread
addf abs.0 servo-thread
```

Make the connections.

```
setp mult2.in1 60
net xpos-cmd ddt.0.in
net X-IPS mult2.0.in0 <= ddt.0.out
net X-ABS abs.0.in <= mult2.0.out
net X-IPM abs.0.out
```

In this last section we are setting the mult2.0.in1 to 60 to convert the inch per second to inch per minute that we get from the ddt.0.out.

The xpos-cmd sends the commanded position to the ddt.0.in. The ddt computes the derivative of the change of the input.

The ddt2.0.out is multiplied by 60 to give IPM.

The mult2.0.out is sent to the abs to get the absolute value.

The following figure shows the result when the X axis is moving at 15 IPM in the minus direction. Notice that we can get the absolute value from either the abs.0.out pin or the X-IPM signal.

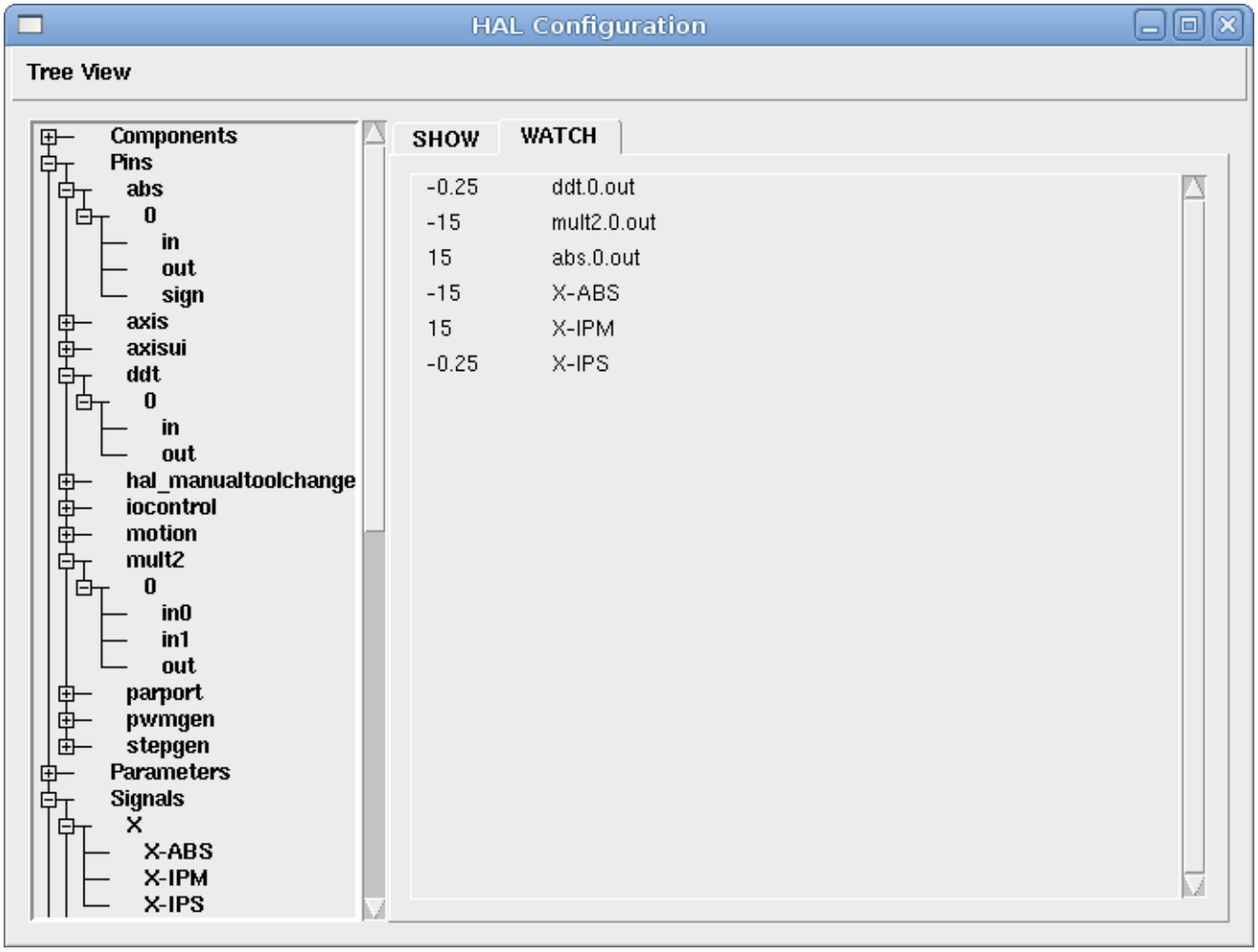



Figure 12.24: Velocity Example

### 12.11.4 Soft Start

This example shows how the HAL components *lowpass*, *limit2* or *limit3* can be used to limit how fast a signal changes.

In this example we have a servo motor driving a lathe spindle. If we just used the commanded spindle speeds on the servo it will try to go from present speed to commanded speed as fast as it can. This could cause a problem or damage the drive. To slow the rate of change we can send the motion.spindle-speed-out through a limiter before the PID, so that the PID command value changes to new settings more slowly.

Three built-in components that limit a signal are:

- *limit2* limits the range and first derivative of a signal.
- *limit3* limits the range, first and second derivatives of a signal.
- *lowpass* uses an exponentially-weighted moving average to track an input signal.

To find more information on these HAL components check the man pages.

Place the following in a text file called softstart.hal. If you're not familiar with Linux place the file in your home directory.

```
loadrt threads period1=1000000 name1=thread
loadrt siggen
loadrt lowpass
loadrt limit2
loadrt limit3
net square siggen.0.square => lowpass.0.in limit2.0.in limit3.0.in
net lowpass <= lowpass.0.out
net limit2 <= limit2.0.out
net limit3 <= limit3.0.out
setp siggen.0.frequency .1
setp lowpass.0.gain .01
setp limit2.0.maxv 2
setp limit3.0.maxv 2
setp limit3.0.maxa 10
addf siggen.0.update thread
addf lowpass.0 thread
addf limit2.0 thread
addf limit3.0 thread
start
loadusr halscope
```

Open a terminal window and run the file with the following command.

```
halrun -I softstart.hal
```

When the HAL Oscilloscope first starts up click *OK* to accept the default thread.

Next you have to add the signals to the channels. Click on channel 1 then select *square* from the Signals tab. Repeat for channels 2-4 and add lowpass, limit2, and limit3.

Next to set up a trigger signal click on the Source None button and select square. The button will change to Source Chan 1.

Next click on Single in the Run Mode radio buttons box. This will start a run and when it finishes you will see your traces.

To separate the signals so you can see them better click on a channel then use the Pos slider in the Vertical box to set the positions.

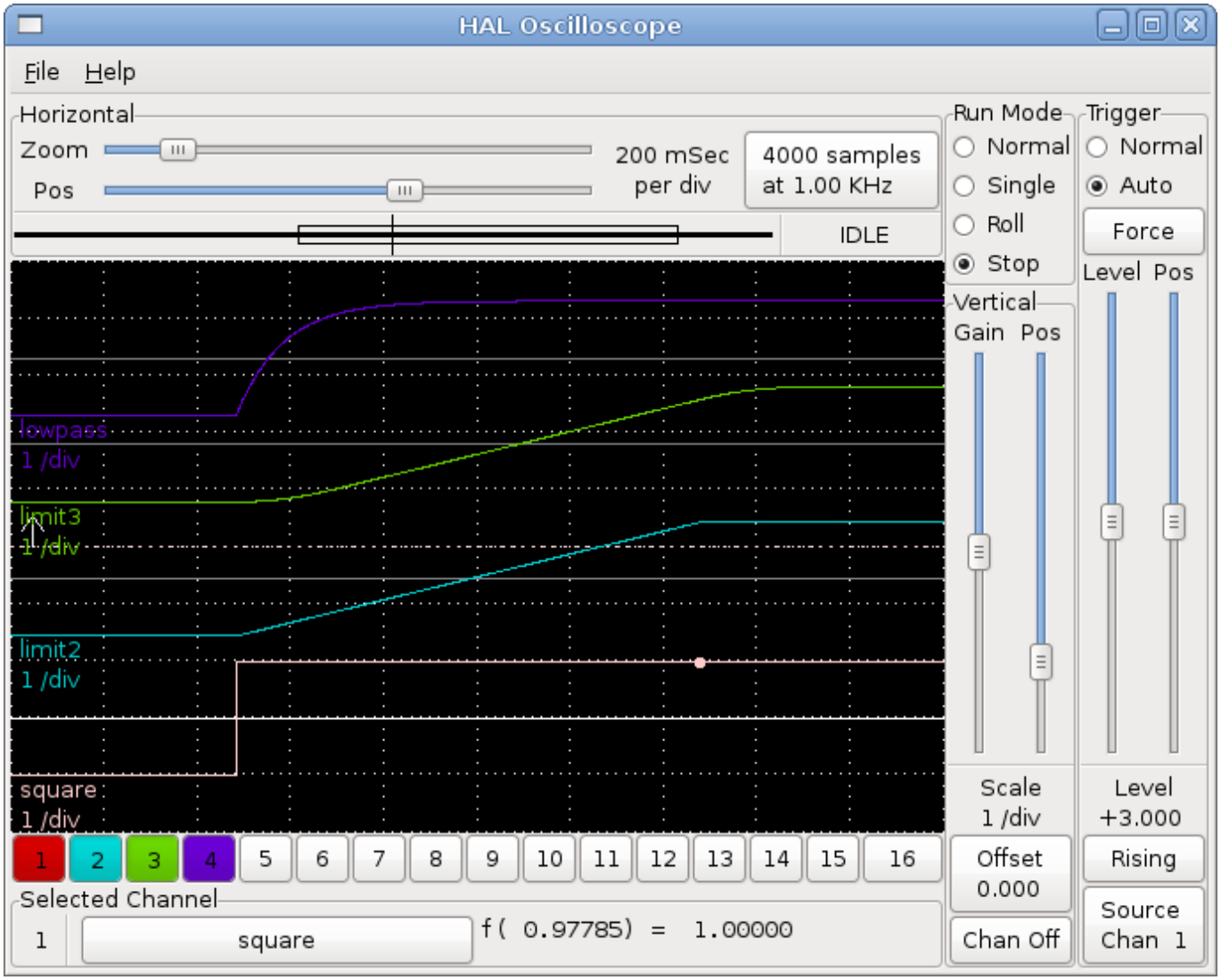



Figure 12.25: Softstart

To see the effect of changing the set point values of any of the components you can change them in the terminal window. To see what different gain settings do for lowpass just type the following in the terminal window and try different settings.

```
setp lowpass.0.gain *.01
```

After changing a setting run the oscilloscope again to see the change.

When you're finished type *exit* in the terminal window to shut down halrun and close the halscope. Don't close the terminal window with halrun running as it might leave some things in memory that could prevent EMC from loading.

For more information on Halscope see the HAL manual.

### 12.11.5 Stand Alone HAL

In some cases you might want to run a GladeVCP screen with just HAL. For example say you had a stepper driven device that all you need is to run a stepper motor. A simple *Start/Stop* interface is all you need for your application so no need to load up and configure a full blown CNC application.

In the following example we have created a simple GladeVCP panel with one

**Basic Syntax**

```
# load the winder.glade GUI and name it winder
loadusr -Wn winder gladevcp -c winder -u handler.py winder.glade

# load realtime components
loadrt threads name1=fast period1=50000 fp1=0 name2=slow period2=1000000
loadrt stepgen step_type=0 ctrl_type=v
loadrt hal_parport cfg="0x378 out"

# add functions to threads
addf stepgen.make-pulses fast
addf stepgen.update-freq slow
addf stepgen.capture-position slow
addf parport.0.read fast
addf parport.0.write fast

# make hal connections
net winder-step parport.0.pin-02-out <= stepgen.0.step
net winder-dir parport.0.pin-03-out <= stepgen.0.dir
net run-stepgen stepgen.0.enable <= winder.start_button



# start the threads
start

# comment out the following lines while testing and use the interactive
# option halrun -I -f start.hal to be able to show pins etc.

# wait until the gladevcp GUI named winder terminates
waitusr winder

# stop HAL threads
stop

# unload HAL all components before exiting
unloadrt all
```

## 12.12 The HAL Component Generator

### 12.12.1 Introduction

Writing a HAL component can be a tedious process, most of it in setup calls to *rtapi_* and *hal_* functions and associated error checking. *halcompile* will write all this code for you, automatically.

Compiling a HAL component is also much easier when using *halcompile*, whether the component is part of the LinuxCNC source tree, or outside it.

For instance, when coded in C, a simple component such as "ddt" is around 80 lines of code. The equivalent component is very short when written using the *halcompile* preprocessor:

**Simple Component Example**

```
component ddt "Compute the derivative of the input function";
pin in float in;
pin out float out;
variable double old;
function _;
license "GPL"; // indicates GPL v2 or later
;;
```

```
float tmp = in;
out = (tmp - old) / fperiod;
old = tmp;
```

### 12.12.2 Installing

If you're working with an installed version of LinuxCNC you will need to install the development packages.

One method is to say following line in a terminal.

**Installing Dev**

```
sudo apt-get install linuxcnc-dev
```

Another method is to use the Synaptic Package Manager from the main Ubuntu menu to install linuxcnc-dev.

### 12.12.3 Definitions

- *component* - A component is a single real-time module, which is loaded with *halcmd loadrt*. One *.comp* file specifies one component. The component name and file name must match.
- *instance* - A component can have zero or more instances. Each instance of a component is created equal (they all have the same pins, parameters, functions, and data) but behave independently when their pins, parameters, and data have different values.
- *singleton* - It is possible for a component to be a "singleton", in which case exactly one instance is created. It seldom makes sense to write a *singleton* component, unless there can literally only be a single object of that kind in the system (for instance, a component whose purpose is to provide a pin with the current UNIX time, or a hardware driver for the internal PC speaker)

### 12.12.4 Instance creation

For a singleton, the one instance is created when the component is loaded.

For a non-singleton, the *count* module parameter determines how many numbered instances are created. If *count* is not specified, the *names* module parameter determines how many named instances are created. If neither *count* nor *names* is specified, a single numbered instance is created.

### 12.12.5 Implicit Parameters

Functions are implicitly passed the *period* parameter which is the time in nanoseconds of the last period to execute the component. Functions which use floating-point can also refer to *fperiod* which is the floating-point time in seconds, or (period*1e-9). This can be useful in components that need the timing information.

### 12.12.6 Syntax

A *.comp* file consists of a number of declarations, followed by *;;* on a line of its own, followed by C code implementing the module's functions.

Declarations include:

- *component HALNAME (DOC);*
- *pin PINDIRECTION TYPE HALNAME ([SIZE]\[MAXSIZE: CONDSIZE]) (if CONDITION) (= STARTVALUE) (DOC) ;*
- *param PARAMDIRECTION TYPE HALNAME ([SIZE]\[MAXSIZE: CONDSIZE]) (if CONDITION) (= STARTVALUE) (DOC) ;*

- *function HALNAME (fp | nofp) (DOC);*
- *option OPT (VALUE);*
- *variable CTYPE STARREDNAME ([SIZE]);*
- *description DOC;*
- *notes DOC;*
- *see_also DOC;*
- *license LICENSE;*
- *author AUTHOR;*
- *include HEADERFILE;*

Parentheses indicate optional items. A vertical bar indicates alternatives. Words in *CAPITALS* indicate variable text, as follows:

- *NAME* - A standard C identifier
- *STARREDNAME* - A C identifier with zero or more * before it. This syntax can be used to declare instance variables that are pointers. Note that because of the grammar, there may not be whitespace between the * and the variable name.
- *HALNAME* - An extended identifier. When used to create a HAL identifier, any underscores are replaced with dashes, and any trailing dash or period is removed, so that "this_name_" will be turned into "this-name", and if the name is "_", then a trailing period is removed as well, so that "function _" gives a HAL function name like "component.<num>" instead of "component.<num>."

  If present, the prefix *hal_* is removed from the beginning of the component name when creating pins, parameters and functions.

  In the HAL identifier for a pin or parameter, # denotes an array item, and must be used in conjunction with a *[SIZE]* declaration. The hash marks are replaced with a 0-padded number with the same length as the number of # characters.

  When used to create a C identifier, the following changes are applied to the HALNAME:

  1. Any "#" characters, and any ".", "_" or "-" characters immediately before them, are removed.
  2. Any remaining "." and "-" characters are replaced with "_".
  3. Repeated "_" characters are changed to a single "_" character.

  A trailing "_" is retained, so that HAL identifiers which would otherwise collide with reserved names or keywords (e.g., *min*) can be used.

| HALNAME | C Identifier | HAL Identifier |
|---|---|---|
| x_y_z | x_y_z | x-y-z |
| x-y.z | x_y_z | x-y.z |
| x_y_z_ | x_y_z_ | x-y-z |
| x.##.y | x_y(MM) | x.MM.z |
| x.## | x(MM) | x.MM |

- *if CONDITION* - An expression involving the variable *personality* which is nonzero when the pin or parameter should be created
- *SIZE* - A number that gives the size of an array. The array items are numbered from 0 to *SIZE*-1.
- *MAXSIZE : CONDSIZE* - A number that gives the maximum size of the array followed by an expression involving the variable *personality* and which always evaluates to less than *MAXSIZE*. When the array is created its size will be *CONDSIZE*.
- *DOC* - A string that documents the item. String can be a C-style "double quoted" string, like:

```
"Selects the desired edge: TRUE means falling, FALSE means rising"
```

or a Python-style "triple quoted" string, which may include embedded newlines and quote characters, such as:

```
"""The effect of this parameter, also known as "the orb of zot",
will require at least two paragraphs to explain.

Hopefully these paragraphs have allowed you to understand "zot"
better."""
```

The documentation string is in "groff -man" format. For more information on this markup format, see *groff_man(7)*. Remember that *halcompile* interprets backslash escapes in strings, so for instance to set the italic font for the word *example*, write:

```
"\\fIexample\\fB"
```

- *TYPE* - One of the HAL types: *bit*, *signed*, *unsigned*, or *float*. The old names *s32* and *u32* may also be used, but *signed* and *unsigned* are preferred.
- *PINDIRECTION* - One of the following: *in*, *out*, or *io*. A component sets a value for an *out* pin, it reads a value from an *in* pin, and it may read or set the value of an *io* pin.
- *PARAMDIRECTION* - One of the following: *r* or *rw*. A component sets a value for a *r* parameter, and it may read or set the value of a *rw* parameter.
- *STARTVALUE* - Specifies the initial value of a pin or parameter. If it is not specified, then the default is *0* or *FALSE*, depending on the type of the item.
- *HEADERFILE* - The name of a header file, either in double-quotes (`include "myfile.h";`) or in angle brackets (`include <systemfile.h>;`). The header file will be included (using C's #include) at the top of the file, before pin and parameter declarations.

#### 12.12.6.1 HAL functions

- *fp* - Indicates that the function performs floating-point calculations.
- *nofp* - Indicates that it only performs integer calculations. If neither is specified, *fp* is assumed. Neither *halcompile* nor gcc can detect the use of floating-point calculations in functions that are tagged *nofp*, but use of such operations results in undefined behavior.

#### 12.12.6.2 Options

The currently defined options are:

- *option singleton yes* - (default: no) Do not create a *count* module parameter, and always create a single instance. With *singleton*, items are named *component-name.item-name* and without *singleton*, items for numbered instances are named *component-name.<num>.item-name*.
- *option default_count number* - (default: 1) Normally, the module parameter *count* defaults to 1. If specified, the *count* will default to this value instead.
- *option count_function yes* - (default: no) Normally, the number of instances to create is specified in the module parameter *count*; if *count_function* is specified, the value returned by the function *int get_count(void)* is used instead, and the *count* module parameter is not defined.
- *option rtapi_app no* - (default: yes) Normally, the functions `rtapi_app_main()` and `rtapi_app_exit()` are automatically defined. With *option rtapi_app no*, they are not, and must be provided in the C code. Use the following prototypes:
  `int rtapi_app_main(void);`
  `void rtapi_app_exit(void);`
  When implementing your own `rtapi_app_main()`, call the function `int export(char *prefix, long extra_arg)` to register the pins, parameters, and functions for `prefix`.

- *option data TYPE* - (default: none) **deprecated** If specified, each instance of the component will have an associated data block of type *TYPE* (which can be a simple type like *float* or the name of a type created with *typedef*). In new components, *variable* should be used instead.
- *option extra_setup yes* - (default: no) If specified, call the function defined by *EXTRA_SETUP* for each instance. If using the automatically defined *rtapi_app_main*, *extra_arg* is the number of this instance.
- *option extra_cleanup yes* - (default: no) If specified, call the function defined by *EXTRA_CLEANUP* from the automatically defined *rtapi_app_exit*, or if an error is detected in the automatically defined *rtapi_app_main*.
- *option userspace yes* - (default: no) If specified, this file describes a userspace (ie, non-realtime) component, rather than a regular (ie, realtime) one. A userspace component may not have functions defined by the *function* directive. Instead, after all the instances are constructed, the C function `void user_mainloop(void);` is called. When this function returns, the component exits. Typically, *user_mainloop()* will use *FOR_ALL_INSTS()* to perform the update action for each instance, then sleep for a short time. Another common action in *user_mainloop()* may be to call the event handler loop of a GUI toolkit.
- *option userinit yes* - (default: no) This option is ignored if the option *userspace* (see above) is set to *no*. If *userinit* is specified, the function *userinit(argc,argv)* is called before *rtapi_app_main()* (and thus before the call to *hal_init()* ). This function may process the commandline arguments or take other actions. Its return type is *void*; it may call *exit()* if it wishes to terminate rather than create a HAL component (for instance, because the commandline arguments were invalid).
- *option extra_link_args "..."* - (default: "") This option is ignored if the option *userspace* (see above) is set to *no*. When linking a userspace component, the arguments given are inserted in the link line. Note that because compilation takes place in a temporary directory, "-L." refers to the temporary directory and not the directory where the .comp source file resides.

If an option's VALUE is not specified, then it is equivalent to specifying *option ... yes*. The result of assigning an inappropriate value to an option is undefined. The result of using any other option is undefined.

#### 12.12.6.3 License and Authorship

- *LICENSE* - Specify the license of the module for the documentation and for the MODULE_LICENSE() module declaration. For example, to specify that the module's license is GPL v2 or later,

  ```
  license "GPL"; // indicates GPL v2 or later
  ```

  For additional information on the meaning of MODULE_LICENSE() and additional license identifiers, see <*linux/module.h*>. or the manual page *rtapi_module_param(3)*

  This declaration is required.
- *AUTHOR* - Specify the author of the module for the documentation.

#### 12.12.6.4 Per-instance data storage

- *variable CTYPE STARREDNAME;*
- *variable CTYPE STARREDNAME[SIZE];*
- *variable CTYPE STARREDNAME = DEFAULT;*
- *variable CTYPE STARREDNAME[SIZE] = DEFAULT;*

  Declare a per-instance variable *STARREDNAME* of type *CTYPE*, optionally as an array of *SIZE* items, and optionally with a default value *DEFAULT*. Items with no *DEFAULT* are initialized to all-bits-zero. *CTYPE* is a simple one-word C type, such as *float*, *u32*, *s32*, int, etc. Access to array variables uses square brackets.

  If a variable is to be of a pointer type, there may not be any space between the "*" and the variable name. Therefore, the following is acceptable:

  ```
  variable int *example;
  ```

but the following are not:

```
variable int* badexample;
variable int * badexample;
```

#### 12.12.6.5 Comments

C++-style one-line comments (//... ) and

C-style multi-line comments (/* ... */) are both supported in the declaration section.

### 12.12.7 Restrictions

Though HAL permits a pin, a parameter, and a function to have the same name, *halcompile* does not.

Variable and function names that can not be used or are likely to cause problems include:

- Anything beginning with _comp.
- *comp_id*
- *fperiod*
- *rtapi_app_main*
- *rtapi_app_exit*
- *extra_setup*
- *extra_cleanup*

### 12.12.8 Convenience Macros

Based on the items in the declaration section, *halcompile* creates a C structure called `struct __comp_state`. However, instead of referring to the members of this structure (e.g., `*(inst->name)`), they will generally be referred to using the macros below. The details of `struct __comp_state` and these macros may change from one version of *halcompile* to the next.

- *FUNCTION(name)* - Use this macro to begin the definition of a realtime function which was previously declared with *function NAME*. The function includes a parameter *period* which is the integer number of nanoseconds between calls to the function.
- *EXTRA_SETUP()* - Use this macro to begin the definition of the function called to perform extra setup of this instance. Return a negative Unix *errno* value to indicate failure (e.g., *return -EBUSY* on failure to reserve an I/O port), or 0 to indicate success.
- *EXTRA_CLEANUP()* - Use this macro to begin the definition of the function called to perform extra cleanup of the component. Note that this function must clean up all instances of the component, not just one. The "pin_name", "parameter_name", and "data" macros may not be used here.
- *pin_name* or *parameter_name* - For each pin *pin_name* or param *parameter_name* there is a macro which allows the name to be used on its own to refer to the pin or parameter. When *pin_name* or *parameter_name* is an array, the macro is of the form *pin_name(idx)* or *param_name(idx)* where *idx* is the index into the pin array. When the array is a variable-sized array, it is only legal to refer to items up to its *condsize*.

  When the item is a conditional item, it is only legal to refer to it when its *condition* evaluated to a nonzero value.
- *variable_name* - For each variable *variable_name* there is a macro which allows the name to be used on its own to refer to the variable. When *variable_name* is an array, the normal C-style subscript is used: *variable_name[idx]*
- *data* - If "option data" is specified, this macro allows access to the instance data.
- *fperiod* - The floating-point number of seconds between calls to this realtime function.
- *FOR_ALL_INSTS() {...}* - For userspace components. This macro iterates over all the defined instances. Inside the body of the loop, the *pin_name*, *parameter_name*, and *data* macros work as they do in realtime functions.

### 12.12.9 Components with one function

If a component has only one function and the string "FUNCTION" does not appear anywhere after *;;*, then the portion after *;;* is all taken to be the body of the component's single function. See the Simple Comp for and example of this.

### 12.12.10 Component Personality

If a component has any pins or parameters with an "if condition" or "[maxsize : condsize]", it is called a component with *personality*. The *personality* of each instance is specified when the module is loaded. *Personality* can be used to create pins only when needed. For instance, personality is used in the *logic* component, to allow for a variable number of input pins to each logic gate and to allow for a selection of any of the basic boolean logic functions *and*, *or*, and *xor*.

### 12.12.11 Compiling

Place the *.comp* file in the source directory *linuxcnc/src/hal/components* and re-run *make*. *Comp* files are automatically detected by the build system.

If a *.comp* file is a driver for hardware, it may be placed in *linuxcnc/src/hal/drivers* and will be built unless LinuxCNC is configured as a userspace simulator.

### 12.12.12 Compiling realtime components outside the source tree

*halcompile* can process, compile, and install a realtime component in a single step, placing *rtexample.ko* in the LinuxCNC realtime module directory:

```
halcompile --install rtexample.comp
```

Or, it can process and compile in one step, leaving *example.ko* (or *example.so* for the simulator) in the current directory:

```
halcompile --compile rtexample.comp
```

Or it can simply process, leaving *example.c* in the current directory:

```
halcompile rtexample.comp
```

*halcompile* can also compile and install a component written in C, using the *--install* and *--compile* options shown above:

```
halcompile --install rtexample2.c
```

man-format documentation can also be created from the information in the declaration section:

```
halcompile --document rtexample.comp
```

The resulting manpage, *example.9* can be viewed with

```
man ./example.9
```

or copied to a standard location for manual pages.

### 12.12.13 Compiling userspace components outside the source tree

*halcompile* can process, compile, install, and document userspace components:

```
halcompile usrexample.comp
halcompile --compile usrexample.comp
halcompile --install usrexample.comp
halcompile --document usrexample.comp
```

This only works for *.comp* files, not for *.c* files.

### 12.12.14 Examples

#### 12.12.14.1 constant

Note that the declaration "function _" creates functions named "constant.0" , etc. The file name must match the component name.

```
component constant;
pin out float out;
param r float value = 1.0;
function _;
license "GPL"; // indicates GPL v2 or later
;;
FUNCTION(_) { out = value; }
```

#### 12.12.14.2 sincos

This component computes the sine and cosine of an input angle in radians. It has different capabilities than the "sine" and "cosine" outputs of siggen, because the input is an angle, rather than running freely based on a "frequency" parameter.

The pins are declared with the names *sin_* and *cos_* in the source code so that they do not interfere with the functions *sin()* and *cos()*. The HAL pins are still called *sincos.<num>.sin*.

```
component sincos;
pin out float sin_;
pin out float cos_;
pin in float theta;
function _;
license "GPL"; // indicates GPL v2 or later
;;
#include <rtapi_math.h>
FUNCTION(_) { sin_ = sin(theta); cos_ = cos(theta); }
```

#### 12.12.14.3 out8

This component is a driver for a *fictional* card called "out8", which has 8 pins of digital output which are treated as a single 8-bit value. There can be a varying number of such cards in the system, and they can be at various addresses. The pin is called *out_* because *out* is an identifier used in *<asm/io.h>*. It illustrates the use of *EXTRA_SETUP* and *EXTRA_CLEANUP* to request an I/O region and then free it in case of error or when the module is unloaded.

```
component out8;
pin out unsigned out_ "Output value; only low 8 bits are used";
param r unsigned ioaddr;

function _;

option count_function;
option extra_setup;
option extra_cleanup;
option constructable no;

license "GPL"; // indicates GPL v2 or later
;;
#include <asm/io.h>

#define MAX 8
int io[MAX] = {0,};
RTAPI_MP_ARRAY_INT(io, MAX, "I/O addresses of out8 boards");

int get_count(void) {
```

```
    int i = 0;
    for(i=0; i<MAX && io[i]; i++) { /* Nothing */ }
    return i;
}

EXTRA_SETUP() {
    if(!rtapi_request_region(io[extra_arg], 1, "out8")) {
        // set this I/O port to 0 so that EXTRA_CLEANUP does not release the IO
        // ports that were never requested.
        io[extra_arg] = 0;
        return -EBUSY;
    }
    ioaddr = io[extra_arg];
    return 0; }

EXTRA_CLEANUP() {
    int i;
    for(i=0; i < MAX && io[i]; i++) {
        rtapi_release_region(io[i], 1);
    }
}

FUNCTION(_) { outb(out_, ioaddr); }
```

#### 12.12.14.4 hal_loop

```
component hal_loop;
pin out float example;
```

This fragment of a component illustrates the use of the *hal_* prefix in a component name. *loop* is the name of a standard Linux kernel module, so a *loop* component might not successfully load if the Linux *loop* module was also present on the system.

When loaded, *halcmd show comp* will show a component called *hal_loop*. However, the pin shown by *halcmd show pin* will be *loop.0.example*, not *hal-loop.0.example*.

#### 12.12.14.5 arraydemo

This realtime component illustrates use of fixed-size arrays:

```
component arraydemo "4-bit Shift register";
pin in bit in;
pin out bit out-# [4];
function _ nofp;
license "GPL"; // indicates GPL v2 or later
;;
int i;
for(i=3; i>0; i--) out(i) = out(i-1);
out(0) = in;
```

#### 12.12.14.6 rand

This userspace component changes the value on its output pin to a new random value in the range (0,1) about once every 1ms.

```
component rand;
option userspace;

pin out float out;
license "GPL"; // indicates GPL v2 or later
```

```
;;
#include <unistd.h>

void user_mainloop(void) {
    while(1) {
        usleep(1000);
        FOR_ALL_INSTS() out = drand48();
    }
}
```

#### 12.12.14.7 logic

This realtime component shows how to use "personality" to create variable-size arrays and optional pins.

```
component logic "LinuxCNC HAL component providing experimental logic functions";
pin in bit in-##[16 : personality & 0xff];
pin out bit and if personality & 0x100;
pin out bit or if personality & 0x200;
pin out bit xor if personality & 0x400;
function _ nofp;
description """
Experimental general 'logic function' component.  Can perform 'and', 'or'
and 'xor' of up to 16 inputs.  Determine the proper value for 'personality'
by adding:
.IP \\(bu 4
The number of input pins, usually from 2 to 16
.IP \\(bu
256 (0x100)  if the 'and' output is desired
.IP \\(bu
512 (0x200)  if the 'or' output is desired
.IP \\(bu
1024 (0x400)  if the 'xor' (exclusive or) output is desired""";
license "GPL"; // indicates GPL v2 or later
;;
FUNCTION(_) {
    int i, a=1, o=0, x=0;
    for(i=0; i < (personality & 0xff); i++) {
        if(in(i)) { o = 1; x = !x; }
        else { a = 0; }
    }
    if(personality & 0x100) and = a;
    if(personality & 0x200) or = o;
    if(personality & 0x400) xor = x;
}
```

A typical load line for this component might be

```
loadrt logic count=3 personality=0x102,0x305,0x503
```

which creates the following pins:

- A 2-input AND gate: logic.0.and, logic.0.in-00, logic.0.in-01
- 5-input AND and OR gates: logic.1.and, logic.1.or, logic.1.in-00, logic.1.in-01, logic.1.in-02, logic.1.in-03, logic.1.in-04,
- 3-input AND and XOR gates: logic.2.and, logic.2.xor, logic.2.in-00, logic.2.in-01, logic.2.in-02

## 12.13 Creating Userspace Python Components

### 12.13.1 Basic usage

A userspace component begins by creating its pins and parameters, then enters a loop which will periodically drive all the outputs from the inputs. The following component copies the value seen on its input pin (*passthrough.in*) to its output pin (*passthrough.out*) approximately once per second.

```
#!/usr/bin/python
import hal, time
h = hal.component("passthrough")
h.newpin("in", hal.HAL_FLOAT, hal.HAL_IN)
h.newpin("out", hal.HAL_FLOAT, hal.HAL_OUT)
h.ready()
try:
    while 1:
        time.sleep(1)
        h['out'] = h['in']
except KeyboardInterrupt:
    raise SystemExit
```

Copy the above listing into a file named "passthrough", make it executable (*chmod +x*), and place it on your *$PATH*. Then try it out:

```
halrun

halcmd: loadusr passthrough

halcmd: show pin

    Component Pins:
    Owner Type  Dir     Value  Name
     03   float IN          0  passthrough.in
     03   float OUT         0  passthrough.out

halcmd: setp passthrough.in 3.14

halcmd: show pin

    Component Pins:
    Owner Type  Dir     Value  Name
     03   float IN       3.14  passthrough.in
     03   float OUT      3.14  passthrough.out
```

### 12.13.2 Userspace components and delays

If you typed “show pin” quickly, you may see that *passthrough.out* still had its old value of 0. This is because of the call to *time.sleep(1)*, which makes the assignment to the output pin occur at most once per second. Because this is a userspace component, the actual delay between assignments can be much longer if the memory used by the passthrough component is swapped to disk, the assignment could be delayed until that memory is swapped back in.

Thus, userspace components are suitable for user-interactive elements such as control panels (delays in the range of milliseconds are not noticed, and longer delays are acceptable), but not for sending step pulses to a stepper driver board (delays must always be in the range of microseconds, no matter what).

### 12.13.3 Create pins and parameters

```
h = hal.component("passthrough")
```

The component itself is created by a call to the constructor *hal.component*. The arguments are the HAL component name and (optionally) the prefix used for pin and parameter names. If the prefix is not specified, the component name is used.

```
h.newpin("in", hal.HAL_FLOAT, hal.HAL_IN)
```

Then pins are created by calls to methods on the component object. The arguments are: pin name suffix, pin type, and pin direction. For parameters, the arguments are: parameter name suffix, parameter type, and parameter direction.

Table 12.3: HAL Option Names

| | | | | |
|---|---|---|---|---|
| **Pin and Parameter Types:** | HAL_BIT | HAL_FLOAT | HAL_S32 | HAL_U32 |
| **Pin Directions:** | HAL_IN | HAL_OUT | HAL_IO | |
| **Parameter Directions:** | HAL_RO | HAL_RW | | |

The full pin or parameter name is formed by joining the prefix and the suffix with a ".", so in the example the pin created is called *passthrough.in*.

```
h.ready()
```

Once all the pins and parameters have been created, call the *.ready()* method.

#### 12.13.3.1 Changing the prefix

The prefix can be changed by calling the *.setprefix()* method. The current prefix can be retrieved by calling the *.getprefix()* method.

### 12.13.4 Reading and writing pins and parameters

For pins and parameters which are also proper Python identifiers, the value may be accessed or set using the attribute syntax:

```
h.out = h.in
```

For all pins, whether or not they are also proper Python identifiers, the value may be accessed or set using the subscript syntax:

```
h['out'] = h['in']
```

#### 12.13.4.1 Driving output (HAL_OUT) pins

Periodically, usually in response to a timer, all HAL_OUT pins should be "driven" by assigning them a new value. This should be done whether or not the value is different than the last one assigned. When a pin is connected to a signal, its old output value is not copied into the signal, so the proper value will only appear on the signal once the component assigns a new value.

#### 12.13.4.2 Driving bidirectional (HAL_IO) pins

The above rule does not apply to bidirectional pins. Instead, a bidirectional pin should only be driven by the component when the component wishes to change the value. For instance, in the canonical encoder interface, the encoder component only sets the *index-enable* pin to **FALSE** (when an index pulse is seen and the old value is **TRUE**), but never sets it to **TRUE**. Repeatedly driving the pin **FALSE** might cause the other connected component to act as though another index pulse had been seen.

### 12.13.5 Exiting

A *halcmd unload* request for the component is delivered as a *KeyboardInterrupt* exception. When an unload request arrives, the process should either exit in a short time, or call the *.exit()* method on the component if substantial work (such as reading or writing files) must be done to complete the shutdown process.

### 12.13.6 Helpful Functions

#### 12.13.6.1 component_exists

Does the specified component exist at this time.
Example:
hal.component_exists("testpanel")

#### 12.13.6.2 component_is_ready

Is the specified component ready at this time.
Example:
hal.component_is_ready("testpanel")

#### 12.13.6.3 connect

Connect a pin to a signal.
example:
hal.connect("pinname","signal_name")

#### 12.13.6.4 new_signal

Create a New signal of the type specified.
example"
hal.new_sig("signalname",hal.HAL_BIT)

#### 12.13.6.5 pin_has_writer

Does the specified pin have a driving pin connected.
Returns True or False.

#### 12.13.6.6 set_p

Set a pin value.
example:
hal.set_p("pinname","10")

### 12.13.7 Constants

Use These To specify details rather then the value they hold.

- HAL_BIT
- HAL_FLOAT
- HAL_S32
- HAL_U32
- HAL_IN
- HAL_OUT
- HAL_RO
- HAL_RW
- MSG_NONE
- MSG_ALL
- MSG_DBG
- MSG_ERR
- MSG_INFO
- MSG_WARN

### 12.13.8 System Information

Read these to aquire information about the realtime system.

- is_kernelspace
- is_rt
- is_sim
- is_userspace

### 12.13.9 Project ideas

- Create an external control panel with buttons, switches, and indicators. Connect everything to a microcontroller, and connect the microcontroller to the PC using a serial interface. Python has a very capable serial interface module called pyserial (Ubuntu package name "python-serial", in the universe repository)
- Attach a LCDProc-compatible LCD module and use it to display a digital readout with information of your choice (Ubuntu package name "lcdproc", in the universe repository)
- Create a virtual control panel using any GUI library supported by Python (gtk, qt, wxwindows, etc)

# Chapter 13

# Advanced Topics

## 13.1 Kinematics

### 13.1.1 Introduction

When we talk about CNC machines, we usually think about machines that are commanded to move to certain locations and perform various tasks. In order to have an unified view of the machine space, and to make it fit the human point of view over 3D space, most of the machines (if not all) use a common coordinate system called the Cartesian Coordinate System.

The Cartesian Coordinate system is composed of three axes (X, Y, Z) each perpendicular to the other two. [1]

When we talk about a G-code program (RS274/NGC) we talk about a number of commands (G0, G1, etc.) which have positions as parameters (X- Y- Z-). These positions refer exactly to Cartesian positions. Part of the LinuxCNC motion controller is responsible for translating those positions into positions which correspond to the machine kinematics. [2]

#### 13.1.1.1 Joints vs. Axes

A joint of a CNC machine is a one of the physical degrees of freedom of the machine. This might be linear (leadscrews) or rotary (rotary tables, robot arm joints). There can be any number of joints on a given machine. For example, one popular robot has 6 joints, and a typical simple milling machine has only 3.

There are certain machines where the joints are laid out to match kinematics axes (joint 0 along axis X, joint 1 along axis Y, joint 2 along axis Z), and these machines are called Cartesian machines (or machines with Trivial Kinematics). These are the most common machines used in milling, but are not very common in other domains of machine control (e.g. welding: puma-typed robots).

### 13.1.2 Trivial Kinematics

The simplest machines are those in which which each joint is placed along one of the Cartesian axes. On these machines the mapping from Cartesian space (the G-code program) to the joint space (the actual actuators of the machine) is trivial. It is a simple 1:1 mapping:

```
pos->tran.x = joints[0];
pos->tran.y = joints[1];
pos->tran.z = joints[2];
pos->a = joints[3];
pos->b = joints[4];
pos->c = joints[5];
```

[1] The word “axes” is also commonly (and wrongly) used when talking about CNC machines, and referring to the moving directions of the machine.
[2] Kinematics: a two way function to transform from Cartesian space to joint space

In the above code snippet one can see how the mapping is done: the X position is identical with the joint 0, the Y posittion with with joint 1, etc. The above refers to the direct kinematics (one direction of the transformation). The next code snippet refers to the inverse kinematics (or the inverse direction of the transformation):

```
joints[0] = pos->tran.x;
joints[1] = pos->tran.y;
joints[2] = pos->tran.z;
joints[3] = pos->a;
joints[4] = pos->b;
joints[5] = pos->c;
```

As one can see, it's pretty straightforward to do the transformation for a trivial "kins" (kinematics) or Cartesian machine. It gets a bit more complicated if the machine is missing one of the axes.[3] [4]

### 13.1.3 Non-trivial kinematics

There can be quite a few types of machine setups (robots: puma, scara; hexapods etc.). Each of them is set up using linear and rotary joints. These joints don't usually match with the Cartesian coordinates, therefore we need a kinematics function which does the conversion (actually 2 functions: forward and inverse kinematics function).

To illustrate the above, we will analyze a simple kinematics called bipod (a simplified version of the tripod, which is a simplified version of the hexapod).

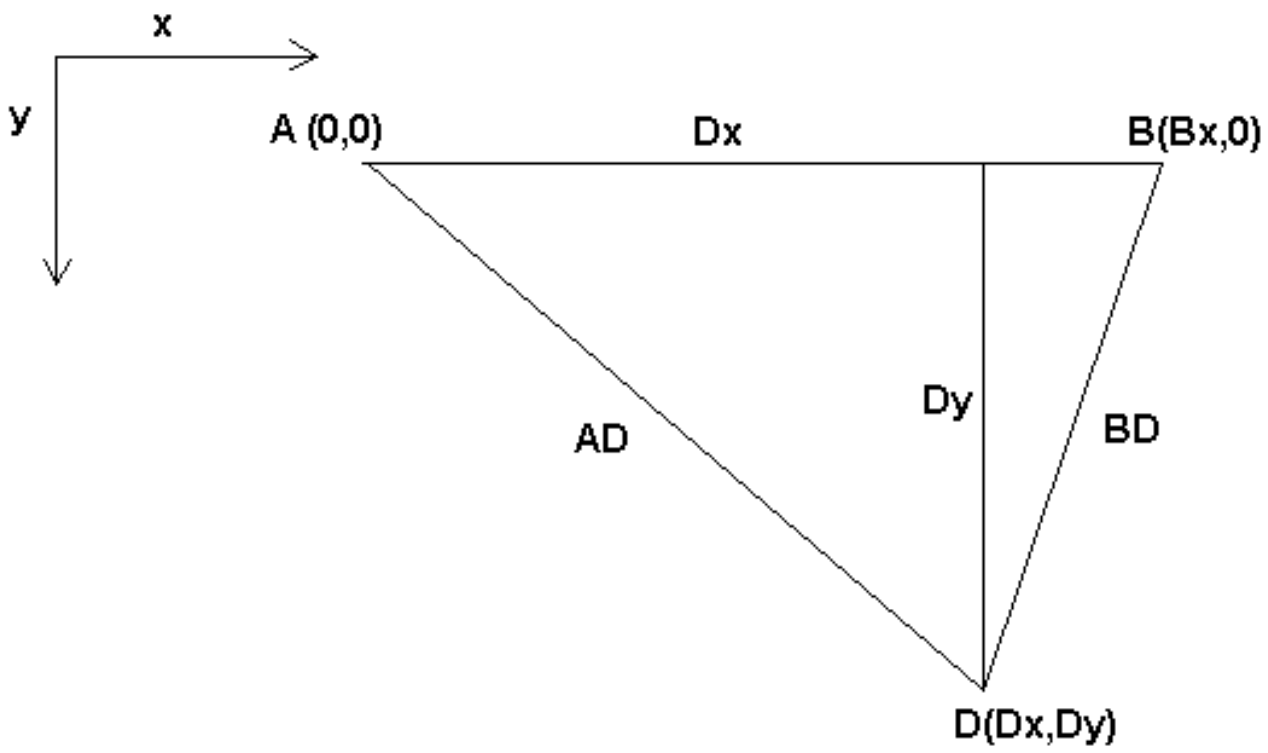



Figure 13.1: Bipod setup

[3] If a machine (e.g. a lathe) is set up with only the axes X,Z & A, and the LinuxCNC inifile holds only these 3 joints defined, then the above matching will be faulty. That is because we actually have (joint0=x, joint1=Z, joint2=A) whereas the above assumes joint1=Y. To make it easily work in LinuxCNC one needs to define all axes (XYZA), then use a simple loopback in HAL for the unused Y axis.

[4] One other way of making it work, is by changing the matching code and recompiling the software.

The Bipod we are talking about is a device that consists of 2 motors placed on a wall, from which a device is hung using some wire. The joints in this case are the distances from the motors to the device (named AD and BD in the figure).

The position of the motors is fixed by convention. Motor A is in (0,0), which means that its X coordinate is 0, and its Y coordinate is also 0. Motor B is placed in (Bx, 0), which means that its X coordinate is Bx.

Our tooltip will be in point D which gets defined by the distances AD and BD, and by the Cartesian coordinates Dx, Dy.

The job of the kinematics is to transform from joint lengths (AD, BD) to Cartesian coordinates (Dx, Dy) and vice-versa.

#### 13.1.3.1 Forward transformation

To transform from joint space into Cartesian space we will use some trigonometry rules (the right triangles determined by the points (0,0), (Dx,0), (Dx,Dy) and the triangle (Dx,0), (Bx,0) and (Dx,Dy).

We can easily see that $AD^2 = x^2 + y^2 BD^2 = (Bx - x)^2 + y^2$, likewise $BD^2 = (Bx - x)^2 + y^2$

If we subtract one from the other we will get:

$$AD^2 - BD^2 = x^2 + y^2 - x^2 + 2 * x * Bx - Bx^2 - y^2$$

and therefore:

$$x = \frac{AD^2 - BD^2 + Bx^2}{2 * Bx}$$

From there we calculate:

$$y = \sqrt{AD2 - x2}$$

Note that the calculation for y involves the square root of a difference, which may not result in a real number. If there is no single Cartesian coordinate for this joint position, then the position is said to be a singularity. In this case, the forward kinematics return -1.

Translated to actual code:

```
double AD2 = joints[0] * joints[0];
double BD2 = joints[1] * joints[1];
double x = (AD2 - BD2 + Bx * Bx) / (2 * Bx);
double y2 = AD2 - x * x;
if(y2 < 0) return -1;
pos->tran.x = x;
pos->tran.y = sqrt(y2);
return 0;
```

#### 13.1.3.2 Inverse transformation

The inverse kinematics is lots easier in our example, as we can write it directly:

$$AD = \sqrt{x^2 + y^2}$$

$$BD = \sqrt{(Bx - x)^2 + y^2}$$

or translated to actual code:

```
double x2 = pos->tran.x * pos->tran.x;
double y2 = pos->tran.y * pos->tran.y;
joints[0] = sqrt(x2 + y2);
joints[1] = sqrt((Bx - pos->tran.x)*(Bx - pos->tran.x) + y2);
return 0;
```

### 13.1.4 Implementation details

A kinematics module is implemented as a HAL component, and is permitted to export pins and parameters. It consists of several "C" functions (as opposed to HAL functions):

```
int kinematicsForward(const double *joint, EmcPose *world,
const KINEMATICS_FORWARD_FLAGS *fflags,
KINEMATICS_INVERSE_FLAGS *iflags)
```

Implements the forward kinematics function.

```
int kinematicsInverse(const EmcPose * world, double *joints,
const KINEMATICS_INVERSE_FLAGS *iflags,
KINEMATICS_FORWARD_FLAGS *fflags)
```

Implements the inverse kinematics function.

```
KINEMATICS_TYPE kinematicsType(void)
```

Returns the kinematics type identifier, typically *KINEMATICS_BOTH*.

```
int kinematicsHome(EmcPose *world, double *joint,
KINEMATICS_FORWARD_FLAGS *fflags,
KINEMATICS_INVERSE_FLAGS *iflags)
```

The home kinematics function sets all its arguments to their proper values at the known home position. When called, these should be set, when known, to initial values, e.g., from an INI file. If the home kinematics can accept arbitrary starting points, these initial values should be used.

```
int rtapi_app_main(void)
void rtapi_app_exit(void)
```

These are the standard setup and tear-down functions of RTAPI modules.

When they are contained in a single source file, kinematics modules may be compiled and installed by *comp*. See the *comp(1)* manpage or the HAL manual for more information.

## 13.2 PID Tuning

### 13.2.1 PID Controller

A proportional-integral-derivative controller (PID controller) is a common feedback loop component in industrial control systems.[5]

The Controller compares a measured value from a process (typically an industrial process) with a reference set point value. The difference (or *error* signal) is then used to calculate a new value for a manipulable input to the process that brings the process measured value back to its desired set point.

Unlike simpler control algorithms, the PID controller can adjust process outputs based on the history and rate of change of the error signal, which gives more accurate and stable control. (It can be shown mathematically that a PID loop will produce accurate, stable control in cases where a simple proportional control would either have a steady-state error or would cause the process to oscillate).

[5] This Subsection is taken from an much more extensive article found at http://en.wikipedia.org/wiki/PID_controller

#### 13.2.1.1 Control loop basics

Intuitively, the PID loop tries to automate what an intelligent operator with a gauge and a control knob would do. The operator would read a gauge showing the output measurement of a process, and use the knob to adjust the input of the process (the *action*) until the process's output measurement stabilizes at the desired value on the gauge.

In older control literature this adjustment process is called a *reset* action. The position of the needle on the gauge is a *measurement*, *process value* or *process variable*. The desired value on the gauge is called a *set point* (also called *set value*). The difference between the gauge's needle and the set point is the *error*.

A control loop consists of three parts:

1. Measurement by a sensor connected to the process (e.g. encoder),
2. Decision in a controller element,
3. Action through an output device such as an motor.

As the controller reads a sensor, it subtracts this measurement from the *set point* to determine the *error*. It then uses the error to calculate a correction to the process's input variable (the *action*) so that this correction will remove the error from the process's output measurement.

In a PID loop, correction is calculated from the error in three ways: cancel out the current error directly (Proportional), the amount of time the error has continued uncorrected (Integral), and anticipate the future error from the rate of change of the error over time (Derivative).

A PID controller can be used to control any measurable variable which can be affected by manipulating some other process variable. For example, it can be used to control temperature, pressure, flow rate, chemical composition, speed, or other variables. Automobile cruise control is an example of a process outside of industry which utilizes crude PID control.

Some control systems arrange PID controllers in cascades or networks. That is, a *master* control produces signals used by *slave* controllers. One common situation is motor controls: one often wants the motor to have a controlled speed, with the *slave* controller (often built into a variable frequency drive) directly managing the speed based on a proportional input. This *slave* input is fed by the *master* controller's output, which is controlling based upon a related variable.

#### 13.2.1.2 Theory

*PID* is named after its three correcting calculations, which all add to and adjust the controlled quantity. These additions are actually *subtractions* of error, because the proportions are usually negative:

**Proportional** To handle the present, the error is multiplied by a (negative) constant P (for *proportional*), and added to (subtracting error from) the controlled quantity. P is only valid in the band over which a controller's output is proportional to the error of the system. Note that when the error is zero, a proportional controller's output is zero.

**Integral** To learn from the past, the error is integrated (added up) over a period of time, and then multiplied by a (negative) constant I (making an average), and added to (subtracting error from) the controlled quantity. I averages the measured error to find the process output's average error from the set point. A simple proportional system either oscillates, moving back and forth around the set point because there's nothing to remove the error when it overshoots, or oscillates and/or stabilizes at a too low or too high value. By adding a negative proportion of (i.e. subtracting part of) the average error from the process input, the average difference between the process output and the set point is always being reduced. Therefore, eventually, a well-tuned PID loop's process output will settle down at the set point.

**Derivative** To handle the future, the first derivative (the slope of the error) over time is calculated, and multiplied by another (negative) constant D, and also added to (subtracting error from) the controlled quantity. The derivative term controls the response to a change in the system. The larger the derivative term, the more rapidly the controller responds to changes in the process's output.

More technically, a PID loop can be characterized as a filter applied to a complex frequency-domain system. This is useful in order to calculate whether it will actually reach a stable value. If the values are chosen incorrectly, the controlled process input can oscillate, and the process output may never stay at the set point.

#### 13.2.1.3 Loop Tuning

*Tuning* a control loop is the adjustment of its control parameters (gain/proportional band, integral gain/reset, derivative gain/rate) to the optimum values for the desired control response. The optimum behavior on a process change or set point change varies depending on the application. Some processes must not allow an overshoot of the process variable from the set point. Other processes must minimize the energy expended in reaching a new set point. Generally stability of response is required and the process must not oscillate for any combination of process conditions and set points.

Tuning of loops is made more complicated by the response time of the process; it may take minutes or hours for a set point change to produce a stable effect. Some processes have a degree of non-linearity and so parameters that work well at full-load conditions don't work when the process is starting up from no-load. This section describes some traditional manual methods for loop tuning.

There are several methods for tuning a PID loop. The choice of method will depend largely on whether or not the loop can be taken *offline* for tuning, and the response speed of the system. If the system can be taken offline, the best tuning method often involves subjecting the system to a step change in input, measuring the output as a function of time, and using this response to determine the control parameters.

**Simple method** If the system must remain on line, one tuning method is to first set the I and D values to zero. Increase the P until the output of the loop oscillates. Then increase I until oscillation stops. Finally, increase D until the loop is acceptably quick to reach its reference. A fast PID loop tuning usually overshoots slightly to reach the set point more quickly; however, some systems cannot accept overshoot.

| Parameter | Rise Time | Overshoot | Settling Time | Steady State Error |
|---|---|---|---|---|
| P | Decrease | Increase | Small Change | Decrease |
| I | Decrease | Increase | Increase | Eliminate |
| D | Small Change | Decrease | Decrease | Small Change |

Effects of increasing parameters

**Ziegler-Nichols method** Another tuning method is formally known as the *Ziegler-Nichols method*, introduced by John G. Ziegler and Nathaniel B. Nichols. It starts in the same way as the method described before: first set the I and D gains to zero and then increase the P gain and expose the loop to external interference for example knocking the motor axis to cause it to move out of equilibrium in order to determine critical gain and period of oscillation until the output of the loop starts to oscillate. Write down the critical gain ($K_c$) and the oscillation period of the output ($P_c$). Then adjust the P, I and D controls as the table shows:

| Control type | P | I | D |
|---|---|---|---|
| P | $.5K_c$ | | |
| PI | $.45K_c$ | $P_c/1.2$ | |
| PID | $.6K_c$ | $P_c/2$ | $P_c/8$ |

**Final Steps** After tuning the axis check the following error with Halscope to make sure it is within your machine requirements. More information on Halscope is in the HAL User manual.

# Part IV

# Glossary

A listing of terms and what they mean. Some terms have a general meaning and several additional meanings for users, installers, and developers.

**Acme Screw**
A type of lead-screw that uses an Acme thread form. Acme threads have somewhat lower friction and wear than simple triangular threads, but ball-screws are lower yet. Most manual machine tools use acme lead-screws.

**Axis**
One of the computer controlled movable parts of the machine. For a typical vertical mill, the table is the X axis, the saddle is the Y axis, and the quill or knee is the Z axis. Angular axes like rotary tables are referred to as A, B, and C. Additional linear axes relative to the tool are called U, V, and W respectively.

**Axis(GUI)**
One of the Graphical User Interfaces available to users of LinuxCNC. It features the modern use of menus and mouse buttons while automating and hiding some of the more traditional LinuxCNC controls. It is the only open-source interface that displays the entire tool path as soon as a file is opened.

**Gmoccapy (GUI)**
A Graphical User Interfaces available to users of LinuxCNC. It features the use and feel of an industrial comtrol and can be used with touch screen, mouse and keyboard. It support embedded tabs and hal driven user messages, it offers a lot of hal beens to be controled with hardware. Gmoccapy is highly cusomizable.

**Backlash**
The amount of "play" or lost motion that occurs when direction is reversed in a lead screw. or other mechanical motion driving system. It can result from nuts that are loose on leadscrews, slippage in belts, cable slack, "wind-up" in rotary couplings, and other places where the mechanical system is not "tight". Backlash will result in inaccurate motion, or in the case of motion caused by external forces (think cutting tool pulling on the work piece) the result can be broken cutting tools. This can happen because of the sudden increase in chip load on the cutter as the work piece is pulled across the backlash distance by the cutting tool.

**Backlash Compensation**
Any technique that attempts to reduce the effect of backlash without actually removing it from the mechanical system. This is typically done in software in the controller. This can correct the final resting place of the part in motion but fails to solve problems related to direction changes while in motion (think circular interpolation) and motion that is caused when external forces (think cutting tool pulling on the work piece) are the source of the motion.

**Ball Screw**
A type of lead-screw that uses small hardened steel balls between the nut and screw to reduce friction. Ball-screws have very low friction and backlash, but are usually quite expensive.

**Ball Nut**
A special nut designed for use with a ball-screw. It contains an internal passage to re-circulate the balls from one end of the screw to the other.

**CNC**
Computer Numerical Control. The general term used to refer to computer control of machinery. Instead of a human operator turning cranks to move a cutting tool, CNC uses a computer and motors to move the tool, based on a part program.

**Comp**
A tool used to build, compile and install LinuxCNC HAL components.

**Configuration(n)**
A directory containing a set of configuration files. Custom configurations are normally saved in the users home/linuxcnc/-configs directory. These files include LinuxCNC's traditional INI file and HAL files. A configuration may also contain several general files that describe tools, parameters, and NML connections.

**Configuration(v)**
The task of setting up LinuxCNC so that it matches the hardware on a machine tool.

**Coordinate Measuring Machine**
A Coordinate Measuring Machine is used to make many accurate measurements on parts. These machines can be used to create CAD data for parts where no drawings can be found, when a hand-made prototype needs to be digitized for moldmaking, or to check the accuracy of machined or molded parts.

**Display units**
The linear and angular units used for onscreen display.

**DRO**
A Digital Read Out is a system of position-measuring devices attached to the slides of a machine tool, which are connected to a numeric display showing the current location of the tool with respect to some reference position. DROs are very popular on hand-operated machine tools because they measure the true tool position without backlash, even if the machine has very loose Acme screws. Some DROs use linear quadrature encoders to pick up position information from the machine, and some use methods similar to a resolver which keeps rolling over.

**EDM**
EDM is a method of removing metal in hard or difficult to machine or tough metals, or where rotating tools would not be able to produce the desired shape in a cost-effective manner. An excellent example is rectangular punch dies, where sharp internal corners are desired. Milling operations can not give sharp internal corners with finite diameter tools. A *wire* EDM machine can make internal corners with a radius only slightly larger than the wire's radius. A *sinker* EDM can make internal corners with a radius only slightly larger than the radius on the corner of the sinking electrode.

**EMC**
The Enhanced Machine Controller. Initially a NIST project. Renamed to LinuxCNC in 2012.

**EMCIO**
The module within LinuxCNC that handles general purpose I/O, unrelated to the actual motion of the axes.

**EMCMOT**
The module within LinuxCNC that handles the actual motion of the cutting tool. It runs as a real-time program and directly controls the motors.

**Encoder**
A device to measure position. Usually a mechanical-optical device, which outputs a quadrature signal. The signal can be counted by special hardware, or directly by the parport with LinuxCNC.

**Feed**
Relatively slow, controlled motion of the tool used when making a cut.

**Feed rate**
The speed at which a cutting motion occurs. In auto or mdi mode, feed rate is commanded using an F word. F10 would mean ten machine units per minute.

**Feedback**
A method (e.g., quadrature encoder signals) by which LinuxCNC receives information about the position of motors

**Feedrate Override**
A manual, operator controlled change in the rate at which the tool moves while cutting. Often used to allow the operator to adjust for tools that are a little dull, or anything else that requires the feed rate to be "tweaked".

**Floating Point Number**
A number that has a decimal point. (12.300) In HAL it is known as float.

**G-Code**
The generic term used to refer to the most common part programming language. There are several dialects of G-code, LinuxCNC uses RS274/NGC.

**GUI**
Graphical User Interface.

**General**
A type of interface that allows communications between a computer and a human (in most cases) via the manipulation of icons and other elements (widgets) on a computer screen.

**LinuxCNC**
An application that presents a graphical screen to the machine operator allowing manipulation of the machine and the corresponding controlling program.

**HAL**
Hardware Abstraction Layer. At the highest level, it is simply a way to allow a number of building blocks to be loaded and interconnected to assemble a complex system. Many of the building blocks are drivers for hardware devices. However, HAL can do more than just configure hardware drivers.

**Home**
A specific location in the machine's work envelope that is used to make sure the computer and the actual machine both agree on the tool position.

**ini file**
A text file that contains most of the information that configures LinuxCNC for a particular machine.

**Instance**
One can have an instance of a class or a particular object. The instance is the actual object created at runtime. In programmer jargon, the Lassie object is an instance of the Dog class.

**Joint Coordinates**
These specify the angles between the individual joints of the machine. See also Kinematics

**Jog**
Manually moving an axis of a machine. Jogging either moves the axis a fixed amount for each key-press, or moves the axis at a constant speed as long as you hold down the key. In manual mode, jog speed can be set from the graphical interface.

**kernel-space**
See real-time.

**Kinematics**
The position relationship between world coordinates and joint coordinates of a machine. There are two types of kinematics. Forward kinematics is used to calculate world coordinates from joint coordinates. Inverse kinematics is used for exactly the opposite purpose. Note that kinematics does not take into account, the forces, moments etc. on the machine. It is for positioning only.

**Lead-screw**
An screw that is rotated by a motor to move a table or other part of a machine. Lead-screws are usually either ball-screws or acme screws, although conventional triangular threaded screws may be used where accuracy and long life are not as important as low cost.

**Machine units**
The linear and angular units used for machine configuration. These units are specified and used in the ini file. HAL pins and parameters are also generally in machine units.

**MDI**
Manual Data Input. This is a mode of operation where the controller executes single lines of G-code as they are typed by the operator.

**NIST**
National Institute of Standards and Technology. An agency of the Department of Commerce in the United States.

**NML**
Neutral Message Language provides a mechanism for handling multiple types of messages in the same buffer as well as simplifying the interface for encoding and decoding buffers in neutral format and the configuration mechanism.

**Offsets**
An arbitrary amount, added to the value of something to make it equal to some desired value. For example, gcode programs are often written around some convenient point, such as X0, Y0. Fixture offsets can be used to shift the actual execution point of that gcode program to properly fit the true location of the vise and jaws. Tool offsets can be used to shift the "uncorrected" length of a tool to equal that tool's actual length.

**Part Program**
A description of a part, in a language that the controller can understand. For LinuxCNC, that language is RS-274/NGC, commonly known as G-code.

**Program Units**
The linear and angular units used in a part program. The linear program units do not have to be the same as the linear machine units. See G20 and G21 for more information. The angular program units are always measured in degrees.

**Python**
General-purpose, very high-level programming language. Used in LinuxCNC for the Axis GUI, the Stepconf configuration tool, and several G-code programming scripts.

**Rapid**
Fast, possibly less precise motion of the tool, commonly used to move between cuts. If the tool meets the workpiece or the fixturing during a rapid, it is probably a bad thing!

**Rapid rate**
The speed at which a rapid motion occurs. In auto or mdi mode, rapid rate is usually the maximum speed of the machine. It is often desirable to limit the rapid rate when testing a g-code program for the first time.

**Real-time**
Software that is intended to meet very strict timing deadlines. Under Linux, in order to meet these requirements it is necessary to install a realtime kernel such as RTAI and build the software to run in the special real-time environment. For this reason real-time software runs in kernel-space.

**RTAI**
Real Time Application Interface, see https://www.rtai.org/, the real-time extensions for Linux that LinuxCNC can use to achieve real-time performance.

**RTLINUX**
See https://en.wikipedia.org/wiki/RTLinux, an older real-time extension for Linux that LinuxCNC used to use to achieve real-time performance. Obsolete, replaced by RTAI.

**RTAPI**
A portable interface to real-time operating systems including RTAI and POSIX pthreads with realtime extensions.

**RS-274/NGC**
The formal name for the language used by LinuxCNC part programs.

**Servo Motor**
Generally, any motor that is used with error-sensing feedback to correct the position of an actuator. Also, a motor which is specially-designed to provide improved performance in such applications.

**Servo Loop**
A control loop used to control position or velocity of an motor equipped with a feedback device.

**Signed Integer**
A whole number that can have a positive or negative sign. In HAL it is known as s32. (A signed 32-bit integer has a usable range of -2,147,483,647 to +2,147,483,647.)

**Spindle**
The part of a machine tool that spins to do the cutting. On a mill or drill, the spindle holds the cutting tool. On a lathe, the spindle holds the workpiece.

**Spindle Speed Override**
A manual, operator controlled change in the rate at which the tool rotates while cutting. Often used to allow the operator to adjust for chatter caused by the cutter's teeth. Spindle Speed Override assumes that the LinuxCNC software has been configured to control spindle speed.

**Stepconf**
An LinuxCNC configuration wizard. It is able to handle many step-and-direction motion command based machines. It writes a full configuration after the user answers a few questions about the computer and machine that LinuxCNC is to run on.

**Stepper Motor**
A type of motor that turns in fixed steps. By counting steps, it is possible to determine how far the motor has turned. If the load exceeds the torque capability of the motor, it will skip one or more steps, causing position errors.

**TASK**
The module within LinuxCNC that coordinates the overall execution and interprets the part program.

**Tcl/Tk**
A scripting language and graphical widget toolkit with which several of LinuxCNCs GUIs and selection wizards were written.

**Traverse Move**
A move in a straight line from the start point to the end point.

**Units**
See "Machine Units", "Display Units", or "Program Units".

**Unsigned Integer**
A whole number that has no sign. In HAL it is known as u32. (An unsigned 32-bit integer has a usable range of zero to 4,294,967,296.)

**World Coordinates**
This is the absolute frame of reference. It gives coordinates in terms of a fixed reference frame that is attached to some point (generally the base) of the machine tool.

# Part V

# Legal Section

# Chapter 14

# Copyright Terms

Copyright (c) 2000-2015 LinuxCNC.org

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

# Chapter 15

# GNU Free Documentation License

GNU Free Documentation License Version 1.1, March 2000

Copyright © 2000 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

**0. PREAMBLE**

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

**1. APPLICABILITY AND DEFINITIONS**

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

**2. VERBATIM COPYING**

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

**3. COPYING IN QUANTITY**

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

**4. MODIFICATIONS**

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five). C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. H. Include an unaltered copy of this License. I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating

the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version. N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

**5. COMBINING DOCUMENTS**

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

**6. COLLECTIONS OF DOCUMENTS**

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

**7. AGGREGATION WITH INDEPENDENT WORKS**

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

**8. TRANSLATION**

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

**9. TERMINATION**

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

**10. FUTURE REVISIONS OF THIS LICENSE**

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

**ADDENDUM**: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

# Chapter 16

# Index

_
0-10v Spindle Speed Example, 525
7i65, 633

**A**
abs, 631
acme screw, 680
and2, 630
ANGULAR UNITS, 297
APPLICATIONS Section, 294
Arc Distance Mode, 236
at_pid, 634
Auto, 168, 175
Axis
    Manual Tool Change, 97
    Tool Touch Off, 88
axis, 630, 680
axis (HAL pins), 612
AXIS Section, 297
axis-iterface, 504

**B**
Backlash, 298
backlash, 680
backlash compensation, 680
ball nut, 680
ball screw, 680
biquad, 632
Bit, 580
bldc_hall3, 634
blend, 631
Block Delete, 184

**C**
Calling Files, 252
Cartesian machines, 673
charge_pump, 634
clarke2, 634
clarke3, 634
clarkeinv, 634
ClassicLadder, 572
classicladder, 630
CNC, 569, 680
Comment Parameters, 198
Comments, 286
comp, 631, 680
Compensation, 298
Conditional Loops, 251
Configuration Launcher, 25
connecting-rs485, 515
constant, 631
conv_bit_s32, 632
conv_bit_u32, 632
conv_float_s32, 632
conv_float_u32, 632
conv_s32_bit, 632
conv_s32_float, 632
conv_s32_u32, 632
conv_u32_bit, 632
conv_u32_float, 632
conv_u32_s32, 632
coolant, 21, 22
coordinate measuring machine, 681
counter, 631

**D**
ddt, 631
deadzone, 631
debounce, 630, 650
Debug Messages, 197
Determining Spindle Calibration, 39
DISPLAY Section, 289
display units, 681
DRO, 681
Dwell
    Feed Out, 235
dwell, 22

**E**
edge, 630
EDM, 681
EMC, 681
EMC Section, 289
EMCIO, 681
EMCIO Section, 302
EMCMOT, 681

EMCMOT Section, 293
Enabling optional features, 345
encoder, 572, 634, 645, 681
Encoder Block Diagram, 645
encoder_ratio, 634
estop_latch, 634

**F**
F: Set Feed Rate, 253
feed, 681
Feed Out, 234, 235
feed override, 170
Feed Rate, 22
feed rate, 681
feedback, 681
feedcomp, 634
feedrate override, 681
FERROR, 298
FILTER Section, 291
Finding Maximum Acceleration, 37
Five Phase, 642
flipflop, 630
Float, 580
Four Phase, 641

**G**
G Code Best Practices, 199
G Code Order of Execution, 198
G Code Table, 200
G-Code, 681
G0 Rapid Move, 201
G1 Linear Move, 202
G10 L1 Tool Table, 211
G10 L10 Set Tool Table, 213
G10 L11 Set Tool Table, 213
G10 L2 Coordinate System, 211
G10 L20 Set Coordinate System, 214
G17 - G19.1 Plane Select, 214
G2
    G3 Arc Move, 203
G20 Units, 214
G28 Go/Set Predefined Position, 215
G3 Arc Move, 203
G30 Go/Set Predefined Position, 215
G33 Spindle Synchronized Motion, 216
G33.1 Rigid Tapping, 216
G38.x Probe, 217
G4 Dwell, 207
G40 Cutter Compensation Off, 218
G41 G42 Cutter Compensation, 219
G41.1 G42.1 Dynamic Compensation, 219
G43 Tool Length Offset, 220
G43.1 Dynamic Tool Length Offset, 220
G43.2 Apply additional Tool Length Offset, 221
G49 Cancel Tool Length Offset, 221
G5 Cubic spline, 208
G5.1 Quadratic spline, 208
G5.2 G5.3 NURBS Block, 209
G53 Machine Coordinates, 221
G54-G59.3 Select Coordinate System, 222
G61 G61.1 G64 Path Mode, 222
G64 Path Blending, 222
G7 Lathe Diameter Mode, 210
G73 Drilling Cycle Chip Break, 223
G76 Threading Cycle, 224
G8 Lathe Radius Mode, 210
G80 Cancel Modal Motion, 229
G80-G89 Canned Cycles, 226
G81 Drilling Cycle, 230
G82 Drilling Cycle Dwell, 233
G83 Peck Drilling, 234
G85 Boring
    Feed Out, 234
G86 Boring
    Spindle Stop
        Rapid Move Out, 235
G89 Boring
    Dwell
        Feed Out, 235
G90
    G91 Distance Mode, 235
G91 Distance Mode, 235
G92 Coordinate System Offset, 236
G92.3 Restore G92 Offsets, 237
G93
    G94
        G95: Feed Rate Mode, 237
G94
    G95: Feed Rate Mode, 237
G95: Feed Rate Mode, 237
G96
    G97 Spindle Control Mode, 237
G97 Spindle Control Mode, 237
G98
    G99 Canned Cycle Return, 238
G99 Canned Cycle Return, 238
gantrykins, 633
gearchange, 634
genhexkins, 633
genserkins, 633
gladevcp, 630
GUI, 680, 681

**H**
HAL, 569, 682
HAL Component, 571
HAL Parameter, 571
HAL Physical-Pin, 571
HAL Pin, 571
HAL Section, 293
HAL Signal, 571
HAL Type, 571
hal-ax5214h, 573
hal-gm, 573

hal-m5i20, 573
hal-motenc, 573
hal-parport, 573
hal-ppmc, 573
hal-stg, 573
hal-vti, 573
halcmd, 573
Halmeter, 617
    Tutorial Halmeter, 589
halmeter, 573
halscope, 573
halui, 572
HALUI Section, 294
hm2_7i43, 633
hm2_pci, 633
HOME, 306
home, 682
HOME IGNORE LIMITS, 305
HOME IS SHARED, 306
HOME LATCH VEL, 305
HOME OFFSET, 306
HOME SEARCH VEL, 299, 305
HOME SEQUENCE, 306
HOME USE INDEX, 306
hostmot2, 633
hypot, 631

**I**
ilowpass, 634
Immediate Homing, 307
Indirection, 252
INI, 682
INI File
    APPLICATIONS Section, 294
    AXIS Section, 297
    Comments, 286
    DISPLAY Section, 289
    EMC Section, 289
    EMCIO Section, 302
    EMCMOT Section, 293
    FILTER Section, 291
    HAL Section, 293
    HALUI Section, 294
    RS274NGC Section, 292
    TASK Section, 293
    TRAJ Section, 295
ini settings (HAL pins), 614
Instance, 682
integ, 631
invert, 631
iocontrol, 572
iocontrol (HAL pins), 614

**J**
jog, 682
joint coordinates, 682
joyhandle, 635

**K**
kinematics, 673, 682
kins, 633
knob2float, 635

**L**
Latency Test, 31
lead screw, 682
limit1, 632
limit2, 632
limit3, 632
Line Number, 184
LINEAR UNITS, 297
Linux, 6
LOCKING INDEXER, 307
Logging, 197
logic, 630
loop, 683
Looping, 250
lowpass, 632
lut5, 630, 652

**M**
M0
    M1 Program Pause, 238
M1 Program Pause, 238
M100 - M199 User Defined Commands, 247
M19 Orient Spindle, 241
M2
    M30 Program End, 239
M3
    M4
        M5 Spindle Control, 240
M30 Program End, 239
M4
    M5 Spindle Control, 240
M48
    M49 Speed and Feed Override Control, 241
M49 Speed and Feed Override Control, 241
M5 Spindle Control, 240
M50 Feed Override Control, 241
M51 Spindle Speed Override, 242
M52 Adaptive Feed Control, 242
M53 Feed Stop Control, 242
M6-Tool-Change, 240
M60 Pallet Change Pause, 239
M61 Set Current Tool, 242
M62 - M65 Digital Output Control, 242
M66 Wait on Input, 243
M67 Analog Output
    Synchronized, 243
M68 Analog Output, 244
M7
    M8
        M9 Coolant Control, 240
M70 Save Modal State, 244
M71 Invalidate Stored Modal State, 245

M72 Restore Modal State, 245
M73 Save and Autorestore Modal State, 246
M8
    M9 Coolant Control, 240
M9 Coolant Control, 240
machine units, 682
maj3, 632
Manual, 167, 175
Manual Tool Change, 97
match8, 630
MAX ACCELERATION, 297
MAX LIMIT, 298
MAX VELOCITY, 297
maxkins, 633
MDI, 682
MIN FERROR, 298
MIN LIMIT, 298
minmax, 635
Modal Groups, 195
motion, 572, 630
motion (HAL pins), 609
mult2, 631
mux16, 631
mux2, 631
mux4, 631
mux8, 631

**N**
near, 631
net, 577
NIST, 682
NML, 682
not, 630

**O**
offset, 631
offsets, 682
oneshot, 630
Operating without Home Switches, 41
or2, 630
ORIENT OFFSET, 292

**P**
PARAMETER FILE, 292
Parameters, 185
part Program, 683
Path Control Mode, 23
pci-card connectors, 501
PID, 647
PID Block Diagram, 648
pin-numbering-axis, 503
pin-numbering-endsw, 512
pin-numbering-gpio, 501
pluto-servo pinout, 494
pluto-step, 495
pluto-step pinout, 496
pluto-step timings, 497
pluto_servo, 633
pluto_step, 633
Polar Coordinates, 193
Print Messages, 197
Probe Logging, 197
program units, 683
Programming the Planner, 15
pumakins, 633
PWM Spindle Speed Example, 525
PWMgen, 644
pwmgen, 634

**R**
rapid, 683
Rapid Move Out, 235
rapid rate, 683
real-time, 683
refsig-timing-diagram, 508
Repeat Loop, 251
Return Values, 252
rotatekins, 633
RS274NGC, 683
RS274NGC Section, 292
RS274NGC STARTUP CODE, 292
RTAI, 683
RTAPI, 683
RTLINUX, 683

**S**
s32, 580
S: Set Spindle Speed, 253
sample_hold, 635
sampler, 635
scale, 632
scarakins, 633
select8, 630
serport, 633
servo motor, 683
Sherline, 164, 284
Siggen, 651
Signed Integer, 683
sim_encoder, 635
Simulated Encoder, 650
sphereprobe, 635
spindle, 21, 683
Spindle At Speed Example, 527
Spindle Direction Example, 525
Spindle Enable Example, 525
Spindle Soft Start Example, 525
Spindle Stop
    Rapid Move Out, 235
Spindle Synchronized Motion Example, 526
stepgen, 572, 591, 634, 637
Stepgen Block Diagram, 638, 639
stepper motor, 684
steptest, 635
streamer, 635

SUBROUTINE PATH, 292
Subroutines, 249
    Conditional Loops, 251
    Looping, 250
    Repeat Loop, 251
sum2, 631
supply, 572, 635
Synchronized, 243

**T**

T: Select Tool, 253
TASK, 684
TASK Section, 293
threads, 630
threadtest, 635
time, 580, 635
timedelay, 635
timedelta, 635
Tk, 684
tmax, 580
toggle, 635
toggle2nist, 635
Tool Touch Off, 88
Tool-Table-Format, 262
torch height control, 633
Touch Off, 261
TRAJ Section, 295
Trajectory Control, 15
Traverse Move, 684
tripodkins, 633
tristate_bit, 635
tristate_float, 635
Trivial Kinematics, 673
trivkins, 633
Tutorial Halmeter, 589
Tutorial Halscope, 594
Two and Three Phase, 641

**U**

u32, 580
UNITS, 298
units, 22, 684
Unsigned Integer, 684
updown, 631
USER M PATH, 292

**V**

VOLATILE HOME, 306

**W**

watchdog, 635
wcomp, 631
weighted_sum, 631
world coordinates, 684

**X**

xor2, 630
Xylotex, 284