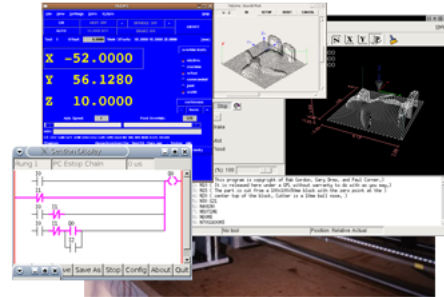




**EMC<sup>2</sup>**

**The Enhanced Machine Controller**



**[www.linuxcnc.org](http://www.linuxcnc.org)**

## V2.0 User Handbook

The EMC Team

June 17, 2006

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](mailto:emc-users@lists.sourceforge.net).

Copyright (c) 2000-6 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 one Back-Cover Text: "This EMC Handbook is the product of several authors writing for linuxCNC.org. As you find it to be of value in your work, we invite you to contribute to its revision and growth." A copy of the license is included in the section entitled "GNU Free Documentation License". If you do not find the license you may order a copy from Free Software Foundation, Inc. 59 Temple Place, Suite 330 Boston, MA 02111-1307

---

# Contents

<b>I</b>	<b>Introduction &amp; installing EMC2</b>	<b>1</b>
<b>1</b>	<b>The Enhanced Machine Control</b>	<b>2</b>
1.1	Introduction . . . . .	2
1.2	The Big CNC Picture . . . . .	2
1.3	Computer Operating Systems . . . . .	3
1.4	History of the Software . . . . .	3
1.5	How the EMC2 Works . . . . .	4
1.5.1	Graphical User Interfaces . . . . .	5
1.5.2	Motion Controller EMCMOT . . . . .	6
1.5.3	Discrete I/O Controller EMCIO . . . . .	7
1.5.4	Task Executor EMCTASK . . . . .	7
1.6	Thinking Like a Machine Operator . . . . .	8
1.6.1	Modes of Operation . . . . .	9
1.6.2	Information Display . . . . .	10
1.7	Thinking Like An Integrator . . . . .	12
1.7.1	Units . . . . .	12
1.7.2	Some things we may not want to change. . . . .	12
1.7.3	Some things we will need to change. . . . .	12
<b>2</b>	<b>Installing the EMC2 software</b>	<b>14</b>
2.1	Introduction . . . . .	14
2.2	EMC Download Page . . . . .	14
2.3	EMC2 install script - the easy way to install . . . . .	15
2.4	Manual installing using apt commands. . . . .	15
<b>II</b>	<b>Configuring EMC2</b>	<b>16</b>
<b>3</b>	<b>Introduction</b>	<b>17</b>
3.1	What is HAL? . . . . .	17
3.1.1	HAL is based on traditional system design techniques . . . . .	17

3.1.1.1	Part Selection . . . . .	17
3.1.1.2	Interconnection Design . . . . .	17
3.1.1.3	Implementation . . . . .	18
3.1.1.4	Testing . . . . .	18
3.1.2	Summary . . . . .	18
3.2	HAL Concepts . . . . .	19
3.3	HAL components . . . . .	20
3.3.1	External Programs with HAL hooks . . . . .	20
3.3.2	Internal Components . . . . .	20
3.3.3	Hardware Drivers . . . . .	21
3.3.4	Tools and Utilities . . . . .	21
3.4	Tinkertoys, Erector Sets, Legos and the HAL . . . . .	21
3.4.1	Tower . . . . .	21
3.4.2	Erector Sets . . . . .	21
3.4.3	Tinkertoys . . . . .	22
3.4.4	A Lego Example . . . . .	23
3.5	Timing Issues In HAL . . . . .	24
3.6	Dynamic Linking and Configuration . . . . .	24
<b>4</b>	<b>HAL Tutorial</b> . . . . .	<b>25</b>
4.1	Before we start . . . . .	25
4.1.1	Notation . . . . .	25
4.1.2	Root Privileges . . . . .	25
4.1.3	The RTAPI environment . . . . .	26
4.2	A Simple Example . . . . .	27
4.2.1	Loading a realtime component . . . . .	27
4.2.2	Examining the HAL . . . . .	27
4.2.3	Making realtime code run . . . . .	28
4.2.4	Changing parameters . . . . .	29
4.2.5	Saving the HAL configuration . . . . .	30
4.2.6	Restoring the HAL configuration . . . . .	31
4.3	Looking at the HAL with halmeter . . . . .	31
4.3.1	Starting halmeter . . . . .	31
4.3.2	Using halmeter . . . . .	33
4.4	A slightly more complex example. . . . .	34
4.4.1	Installing the components . . . . .	34
4.4.2	Connecting pins with signals . . . . .	35
4.4.3	Setting up realtime execution - threads and functions . . . . .	36
4.4.4	Setting parameters . . . . .	38

4.4.5	Run it! . . . . .	38
4.5	Taking a closer look with halscope. . . . .	39
4.5.1	Starting Halscope . . . . .	39
4.5.2	Hooking up the “scope probes” . . . . .	41
4.5.3	Capturing our first waveforms . . . . .	42
4.5.4	Vertical Adjustments . . . . .	43
4.5.5	Triggering . . . . .	43
4.5.6	Horizontal Adjustments . . . . .	45
4.5.7	More Channels . . . . .	46
<b>5</b>	<b>Hardware Drivers</b>	<b>47</b>
5.1	Parport . . . . .	47
5.1.1	Installing . . . . .	47
5.1.2	Removing . . . . .	48
5.1.3	Pins . . . . .	48
5.1.4	Parameters . . . . .	50
5.1.5	Functions . . . . .	50
5.2	AX5214H . . . . .	50
5.2.1	Installing . . . . .	50
5.2.2	Removing . . . . .	51
5.2.3	Pins . . . . .	51
5.2.4	Parameters . . . . .	51
5.2.5	Functions . . . . .	51
5.3	Servo-To-Go . . . . .	52
5.3.1	Installing: . . . . .	52
5.3.2	Removing . . . . .	52
5.3.3	Pins . . . . .	52
5.3.4	Parameters . . . . .	53
5.3.5	Functions . . . . .	53
5.4	Mesa Electronics m5i20 “Anything I/O Card” . . . . .	53
5.4.1	Removing . . . . .	54
5.4.2	Pins . . . . .	54
5.4.3	Parameters . . . . .	55
5.4.4	Functions . . . . .	55
5.4.5	Connector pinout . . . . .	55
5.4.5.1	Connecor P2 . . . . .	56
5.4.5.2	Connector P3 . . . . .	56
5.4.5.3	Connector P4 . . . . .	57
5.4.5.4	LEDs . . . . .	58

5.5	Vital Systems Motenc-100 and Motenc-LITE	58
5.5.1	Removing	59
5.5.2	Pins	59
5.5.3	Parameters	60
5.5.4	Functions	60
5.6	Pico Systems PPMC (Parallel Port Motion Control)	60
5.6.1	Removing	60
5.6.2	Pins	61
5.6.3	Parameters	61
5.6.4	Functions	62
<b>6</b>	<b>Basic configurations for a stepper based system</b>	<b>63</b>
6.1	Introduction	63
6.2	Pinout	63
6.2.1	standard_pinout.hal	64
6.2.2	Overview of the standard_pinout.hal	64
6.2.3	Changing the standard_pinout.hal	65
6.2.4	Adding an enable signal	65
6.2.5	Adding an external ESTOP button	65
<b>7</b>	<b>INI Configuration</b>	<b>66</b>
7.1	Files Used for Configuration	66
7.2	The INI File Layout	66
7.2.1	Comments	67
7.2.2	Sections	67
7.2.3	Variable	68
7.3	INI Variable Definitions	68
7.3.1	[EMC] Section	68
7.3.2	[DISPLAY] Section	68
7.3.3	[EMCMOT] Section	69
7.3.4	[HAL]	69
7.3.5	[TRAJ] Section.	69
7.3.6	[AXIS_#] Section	70
	7.3.6.1 Homing related params	73
7.4	Homing	74
7.4.1	Overview	74
7.4.2	Homing Sequence	74
7.4.3	Configuration	74
	7.4.3.1 HOME_SEARCH_VEL	74

7.4.3.2	HOME_LATCH_VEL	74
7.4.3.3	HOME_IGNORE_LIMITS	74
7.4.3.4	HOME_USE_INDEX	76
7.4.3.5	HOME_OFFSET	76
7.4.3.6	HOME	76
<b>III Using EMC2</b>		<b>77</b>
<b>8</b>	<b>Using the AXIS Graphical Interface</b>	<b>78</b>
8.1	Introduction	78
8.2	Getting Started	78
8.2.1	A typical session with AXIS	78
8.3	Elements of the AXIS window	79
8.3.1	Toolbar buttons	80
8.3.2	Graphical Program Display Area	80
8.3.2.1	Coordinate Display	80
8.3.2.2	Preview Plot	81
8.3.2.3	Program Extents	81
8.3.2.4	Tool Cone	81
8.3.2.5	Backplot	81
8.3.2.6	Interacting with the display	81
8.3.3	Text Program Display Area	82
8.3.4	Manual Control	82
8.3.4.1	The “Axis” group	83
8.3.4.2	The “Spindle” group	83
8.3.4.3	The “Coolant” group	83
8.3.5	Code Entry	83
8.3.5.1	History	83
8.3.5.2	MDI Command	84
8.3.5.3	Active G-Codes	84
8.3.6	Feed Override	84
8.4	Keyboard Controls	84
8.5	emctop: Show EMC Status	85
8.6	mdi: Text-mode MDI interface	85
8.7	Python modules	85
8.8	Advanced configuration of AXIS	86
8.8.1	Program Filters	86
8.8.2	The X Resource Database	87

<b>9</b>	<b>Using The TKEMC Graphical Interface</b>	<b>88</b>
<b>10</b>	<b>Using The MINI Graphical Interface</b>	<b>89</b>
10.1	Introduction . . . . .	89
10.2	Screen layout . . . . .	90
10.3	Menu Bar . . . . .	90
10.4	Control Button Bar . . . . .	92
10.4.1	MANUAL . . . . .	92
10.4.2	AUTO . . . . .	93
10.4.3	MDI . . . . .	94
10.4.4	[FEEDHOLD] – [CONTINUE] . . . . .	94
10.4.5	[ABORT] . . . . .	94
10.4.6	[ESTOP] . . . . .	94
10.5	Left Column . . . . .	95
10.5.1	Axis Position Displays . . . . .	95
10.5.2	Feedrate Override . . . . .	96
10.5.3	Messages . . . . .	96
10.6	Right Column . . . . .	96
10.6.1	Program Editor . . . . .	96
10.6.2	Backplot Display . . . . .	97
10.6.3	Tool Page . . . . .	97
10.6.4	Offset Page . . . . .	98
10.7	Keyboard Bindings . . . . .	98
10.7.1	Common Keys . . . . .	98
10.7.2	Manual Mode . . . . .	99
10.7.3	Auto Mode . . . . .	100
10.8	Misc . . . . .	100
<b>11</b>	<b>Machining Center Overview</b>	<b>101</b>
11.1	Mechanical Components . . . . .	101
11.1.1	Linear Axes . . . . .	101
11.1.2	Rotational axes . . . . .	101
11.1.3	Spindle . . . . .	102
11.1.4	Coolant . . . . .	102
11.1.5	Pallet Shuttle . . . . .	102
11.1.6	Tool Carousel . . . . .	102
11.1.7	Tool Changer . . . . .	102
11.1.8	Message Display . . . . .	102
11.1.9	Feed and Speed Override Switches . . . . .	102



11.1.10	Block Delete Switch . . . . .	102
11.1.11	Optional Program Stop Switch . . . . .	102
11.2	Control and Data Components . . . . .	103
11.2.1	Linear Axes . . . . .	103
11.2.2	Rotational Axes . . . . .	103
11.2.3	Controlled Point . . . . .	103
11.2.4	Coordinate Linear Motion . . . . .	103
11.2.5	Feed Rate . . . . .	103
11.2.6	Coolant . . . . .	104
11.2.7	Dwell . . . . .	104
11.2.8	Units . . . . .	104
11.2.9	Current Position . . . . .	104
11.2.10	Selected Plane . . . . .	104
11.2.11	Tool Carousel . . . . .	105
11.2.12	Tool Change . . . . .	105
11.2.13	Pallet Shuttle . . . . .	105
11.2.14	Feed and Speed Override Switches . . . . .	105
11.2.15	Path Control Mode . . . . .	105
11.3	Interpreter Interaction with Switches . . . . .	105
11.3.1	Feed and Speed Override Switches . . . . .	105
11.3.2	Block Delete Switch . . . . .	105
11.3.3	Optional Program Stop Switch . . . . .	106
11.4	Tool File . . . . .	106
11.5	Parameters . . . . .	107
11.6	Coordinate Systems . . . . .	108
<b>12</b>	<b>Language Overview</b>	<b>109</b>
12.1	Format of a line . . . . .	109
12.2	Line Number . . . . .	110
12.3	Word . . . . .	110
12.3.1	Number . . . . .	111
12.3.2	Parameter Value . . . . .	111
12.3.3	Expressions and Binary Operations . . . . .	111
12.3.4	Unary Operation Value . . . . .	112
12.4	Parameter Setting . . . . .	112
12.5	Comments and Messages . . . . .	112
12.6	Repeated Items . . . . .	112
12.7	Item order . . . . .	113
12.8	Commands and Machine Modes . . . . .	113
12.9	Modal Groups . . . . .	113

<b>13 G Codes</b>	<b>115</b>
13.1 G0: Rapid Linear Motion . . . . .	115
13.2 G1: Linear Motion at Feed Rate . . . . .	115
13.3 G2, G3: Arc at Feed Rate . . . . .	116
13.3.1 Radius format arcs . . . . .	116
13.3.2 Center format arcs . . . . .	117
13.4 G33: Spindle-Synchronized Motion . . . . .	117
13.5 G4: Dwell . . . . .	118
13.6 G10: Set Coordinate System Data . . . . .	118
13.7 G17, G18, G19: Plane Selection . . . . .	118
13.8 G20, G21: Length Units . . . . .	118
13.9 G28, G30: Return to Home . . . . .	119
13.10 G38.2: Straight Probe . . . . .	119
13.11 G40, G41, G42: Cutter Radius Compensation. . . . .	119
13.12 G43, G49: Tool Length Offsets . . . . .	119
13.13 G53: Move in absolute coordinates . . . . .	120
13.14 G54 to G59.3: Select Coordinate System . . . . .	120
13.15 G61, G61.1, G64: Set Path Control Mode . . . . .	120
13.16 G80: Cancel Modal Motion . . . . .	120
13.17 G81 to G89: Canned Cycles . . . . .	120
13.17.1 Preliminary and In-Between Motion . . . . .	122
13.17.2 G81: Drilling Cycle . . . . .	122
13.17.3 G82: Drilling Cycle with Dwell . . . . .	123
13.17.4 G83: Peck Drilling . . . . .	123
13.17.5 G84: Right-Hand Tapping . . . . .	124
13.17.6 G85: Boring, No Dwell, Feed Out . . . . .	124
13.17.7 G86: Boring, Spindle Stop, Rapid Out . . . . .	124
13.17.8 G87: Back Boring . . . . .	124
13.17.9 G88: Boring, Spindle Stop, Manual Out . . . . .	124
13.17.10 G89: Boring, Dwell, Feed Out . . . . .	125
13.17.11 G90, G99: Set Distance Mode . . . . .	125
13.18 G92, G92.1, G92.2, G92.3: Coordinate System Offsets . . . . .	125
13.19 G93, G94: Set Feed Rate Mode . . . . .	126
13.20 G98, G99: Set Canned Cycle Return Level . . . . .	126
<b>14 M Codes</b>	<b>127</b>
14.1 M0, M1, M2, M30, M60: Program Stopping and Ending . . . . .	127
14.2 M3, M4, M5: Spindle Control . . . . .	128
14.3 M6: Tool Change . . . . .	128
14.4 M7, M8, M9: Coolant Control . . . . .	128
14.5 M48, M49: Override Control . . . . .	128
14.6 M100 to M199: User Defined Commands . . . . .	129

<b>15 O Codes</b>	<b>130</b>
15.1 Subroutines: “sub”, “endsub”, “return”, “call” . . . . .	130
15.2 Looping: “do”, “while”, “endwhile”, “break”, “continue” . . . . .	130
15.3 Conditional: “if”, “else”, “endif” . . . . .	131
<b>16 Other Codes</b>	<b>132</b>
16.1 F: Set Feed Rate . . . . .	132
16.2 S: Set Spindle Speed . . . . .	132
16.3 T: Select Tool . . . . .	132
<b>17 Order of Execution</b>	<b>134</b>
<b>18 G-Code Best Practices</b>	<b>135</b>
18.1 Use an appropriate decimal precision . . . . .	135
18.2 Use consistent white space . . . . .	135
18.3 Prefer “Center-format” arcs . . . . .	135
18.4 Put important modal settings at the top of the file . . . . .	135
18.5 Don’t put too many things on one line . . . . .	136
18.6 Don’t use line numbers . . . . .	136
<b>19 Tool File and Compensation</b>	<b>137</b>
19.1 Tool File . . . . .	137
19.2 Tool Compensation . . . . .	138
19.3 Tool Length Offsets . . . . .	138
19.4 Cutter Radius Compensation . . . . .	139
19.4.1 Cutter Radius Compensation Detail . . . . .	139
19.5 Tool Compensation Sources . . . . .	147
<b>20 Coordinate System and G92 Offsets</b>	<b>148</b>
20.1 Introduction . . . . .	148
20.2 The Machine Position Command (G53) . . . . .	148
20.3 Fixture Offsets (G54-G59.3) . . . . .	148
20.3.1 Default coordinate system . . . . .	150
20.3.2 Setting coordinate system values within G-code. . . . .	150
20.4 G92 Offsets . . . . .	150
20.4.1 The G92 commands . . . . .	151
20.4.2 Setting G92 values . . . . .	151
20.4.3 G92 Cautions . . . . .	152
20.5 Sample Program Using Offsets . . . . .	152

<b>21 Canned Cycles</b>	<b>154</b>
21.1 Preliminary Motion	154
21.2 G80	155
21.3 G81 Cycle	156
21.4 G82 Cycle	158
21.5 G83 Cycle	159
21.6 G84 Cycle	159
21.7 G85 Cycle	160
21.8 G86 Cycle	160
21.9 G87 Cycle	160
21.10 G88 Cycle	162
21.11 G89 Cycle	162
21.12 G98 G99	162
21.13 Why use a canned cycle?	163
<b>A Glossary of Common Terms Used in the EMC Documents</b>	<b>165</b>
<b>A Legal Section</b>	<b>168</b>
A.1 GNU Free Documentation License Version 1.1, March 2000	168
A.1.1 GNU Free Documentation License Version 1.1, March 2000	168

## **Part I**

# **Introduction & installing EMC2**

# Chapter 1

## The Enhanced Machine Control

### 1.1 Introduction

This book is intended for people who want to use the Enhanced Machine Controller to run a mill, lathe, router, or to control some other rather standard kind of machine. Computer Numerical Control or CNC is the general term used to name this kind of computer application. In order to get right into the essential task of operating it we have limited the amount of information about installation and setup. We assume that the user will install from one of the standard ways of install (covered in Chapter 2). Machine wiring and setup is limited to what we refer to as a mini or benchtop mill that is powered by stepper motors and amps that use a single parallel port.

If the user is interested in developing their own install using some other distribution of Linux or an other operating system, or applying the EMC2 to a more complex machine, they should study the Integrators Handbook where these topics are covered in greater detail.

### 1.2 The Big CNC Picture

The term CNC has taken on a lot of different meanings over the years. In the early days it replaced the hands of a skilled machinist with motors that followed commands in much the same way that the machinist turned the handwheels. From these early machines, a language of machine tool control has grown. This language is called RS274 and several standard variants of it have been put forward. It has also been expanded by machine tool and control builders in order to meet the needs of specific machines. If a machine changed tools during a program it needed to have tool change commands. If it changed pallets in order to load new castings, it had to have commands that allowed for these kinds of devices as well. Like any language, RS274 has evolved over time. Currently there are several dialects. In general each machine tool maker has been consistent within their product line but different dialects can have commands that cause quite different behavior from one machine to another.

More recently the language of CNC has been hidden behind or side-stepped by several programming schemes that are referred to as “Conversational<sup>1</sup> programming languages.” One common feature of these kinds of programming schemes is the selection of a shape or geometry and the addition of values for the corners, limits, or features of that geometry.

The use of Computer Aided Drafting has also had its affect upon the CNC programming languages. Because CAD drawings are saved as a list or database of geometries and variables associated with each, they are available to be interpreted into G-Code. These interpreters are called CAM (Computer Aided Machining) programs.

---

<sup>1</sup>One machine tool manufacturer, Hurco, claims to have a right to the use of these programming schemes and to the use of the term conversational when used in this context.

Like the CAD converters, the rise of drawing programs, like Corel™ and the whole bunch of paint programs, converters have been written that will take a bitmap or raster or vector image and turn it into G-Code that can be run with a CNC.

You're asking yourself, "Why did I want to know this?" The answer is that the EMC2 as is currently exists does not directly take in CAD or any image and run a machine using it. It. The EMC2 uses a variant of the earlier CNC language named RS 274NGC. (Next Generation Controller). All of the commands given to the EMC2 must be in a form that is recognized and have meaning to the RS274NGC interpreter. This means that if you want to carve parts that were drawn in some graphical or drafting program you will also have to find a converter that will transform the image or geometry list into commands that are acceptable to the EMC2 interpreter. Several commercial CAD/CAM programs are available to do this conversion. At least one converter (Ace) has been written that carries a copyright that makes it available to the public.

There has been recent talk about writing a "conversational" or geometric interface that would allow an operator to enter programs in much the same way that several modern proprietary controls enter programs but it isn't in there yet.

### 1.3 Computer Operating Systems

The EMC2 code can be compiled on almost any GNU-Linux Distribution (assuming it has been patched with a real time extension). In addition to the raw code, some binary distributions are available. The latest packages have been created around the Ubuntu GNU-Linux Distribution. Ubuntu is one of the distributions that is aimed at novice Linux users, and has been found to be very easy to use. Along with that, there are lots of places around the world, that offer support for it. Installing emc2 on it is trivial, as you can see from Chapter 2.

The EMC2 will not run under a normal Microsoft (TM) operating system. The reason for this is that the EMC2 requires a real-time environment for the proper operation of its motion planning and stepper pulse outputs. Along with that, it also benefits from the much-needed stability and performance of the Linux OS.

### 1.4 History of the Software

The EMC2 code was started by the Intelligent Systems Division at the National Institute of Standards and Technology in the United States. The quotation below, taken from the NIST web presence some time back, should lend some understanding of the essential reasons for the existence of this software and of the NIST involvement in it.

As part of our (NIST) collaboration with the OMAC User's Group, we have written software which implements real-time control of equipment such as machine tools, robots, and coordinate measuring machines. The goal of this software development is twofold: first, to provide complete software implementations of all OMAC modules for the purpose of validating application programming interfaces; and second, to provide a vehicle for the transfer of control technology to small- and medium-sized manufacturers via the NIST Manufacturing Extension Partnership. The EMC2 software is based on the NIST Real-time Control System (RCS) Methodology, and is programmed using the NIST RCS Library. The RCS Library eases the porting of controller code to a variety of Unix and Microsoft platforms, providing a neutral application programming interface (API) to operating system resources such as shared memory, semaphores, and timers. The RCS Library also implements a communication model, the Neutral Manufacturing Language, which allows control processes to read and write C++ data structures throughout a single homogeneous environment or a heterogeneous networked environment. The EMC2 software is written in C and C++, and has been ported to the PC Linux, Windows NT, and

Sun Solaris operating systems. When running actual equipment, a real-time version of Linux is used to achieve the deterministic computation rates required (200 microseconds is typical). The software can also be run entirely in simulation, down to simulations of the machine motors. This enables entire factories of EMC2 machines to be set up and run in a computer integrated manufacturing environment.

EMC has been installed on many machines, both with servo motors and stepper motors. Here is a sampling of the earliest applications.

- 3-axis Bridgeport knee mill at Shaver Engineering. The machine uses DC brush servo motors and encoders for motion control, and OPTO-22 compatible I/O interfaced to the PC parallel port for digital I/O to the spindle, coolant, lube, and e-stop systems.
- 3-axis desktop milling machine used for prototype development. The machine uses DC brush servo motors and encoders. Spindle control is accomplished using the 4th motion control axis. The machine cuts wax parts.
- 4-axis Kearney & Trecker horizontal machining center at General Motors Powertrain in Pontiac, MI. This machine ran a precursor to the full-software EMC2 which used a hardware motion control board.

After these early tests, Jon Elson found the Shaver Engineering notes and replaced a refrigerator sized Allen Bradley 7300 control on his Bridgeport with the EMC running on a Red Hat 5.2 distribution of Linux. He was so pleased with the result that he advertised the software on several news groups. He continues to use that installation and has produced several boards that are supported by the software.

From these early applications news of the software spread around the world. It is now used to control many different kinds of machines. More recently the Sherline company <http://www.sherline.com> has released their first CNC mill. It uses a standard release of the EMC.

The source code files that make up the controller are kept in a repository on [cvs.linuxcnc.org](http://cvs.linuxcnc.org) They are available for anyone to download and use as they see fit. All EMC2 files (with some exceptions<sup>2</sup>) carry a GPL or GPLD copyright. These files require that you make your source of them available to your users. GPL copyright also requires that if you modify the code in that file and make a public release of your revisions, you must return your modifications to the developers.

## 1.5 How the EMC2 Works

The Enhanced Machine Controller (EMC2) is a lot more than just another CNC mill program. It can control machine tools, robots, or other automated devices. It can control servo motors, stepper motors, relays, and other devices related to machine tools. In this handbook we focus on only a small part of that awesome capability, the minimill.

Figure 1.1 shows a simple block diagram showing what a typical 3-axis EMC2 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 EMC2 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 EMC2 we will remind the reader of this typical machine.

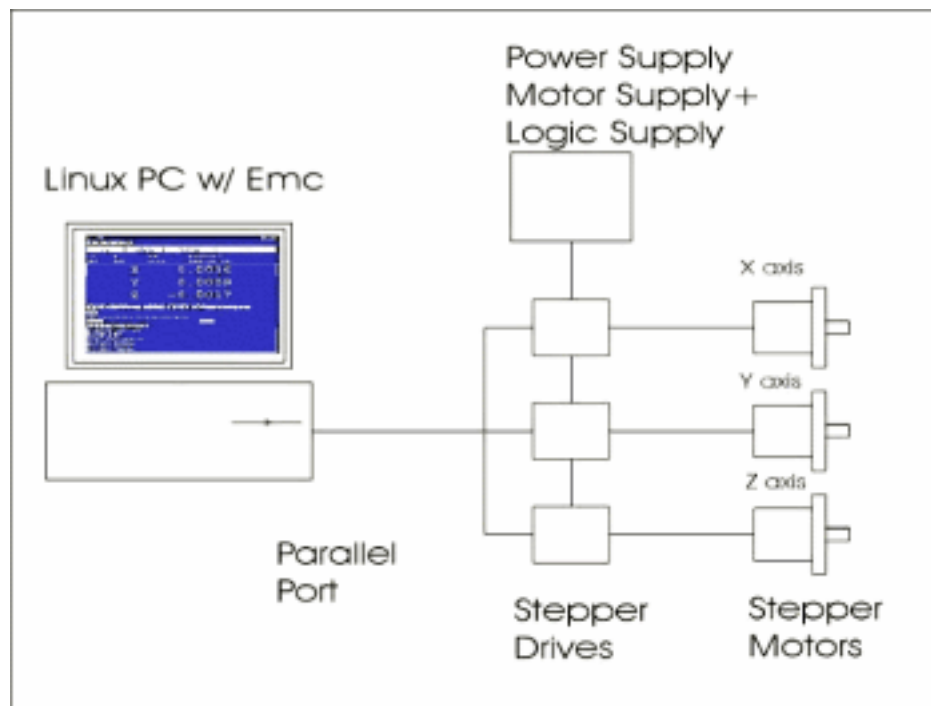
There are four main components to the EMC2 software: a motion controller (EMCMOT), a discrete I/O controller (EMCIO), a task executor which coordinates them (EMCTASK), and a collection of

---

<sup>2</sup>some parts of emc2 are released under LGPL, to allow proprietary software to be linked together with emc2 (GUIs, hardware drivers, etc.)



Figure 1.1: Typical EMC2 Controlled Machine



text-based or graphical user interfaces. An EMC2 capable of running a minimill must start some version of all four of these components in order to completely control it. Each component is briefly described below. In addition there is a layer called HAL (Hardware Abstraction Layer) which allows simple reconfiguration of EMC2 without the need of recompiling.

### 1.5.1 Graphical User Interfaces

A graphical interface is the part of the EMC2 that the machine tool operator interacts with. The EMC2 comes with several types of user interfaces:

- an interactive command-line program named `emcpanel`
- a character-based screen graphics program named `keystick` 1.3
- X Windows programs named `xemc` 1.6 and `yemc`
- a Java-based GUI, `emcgui`
- two Tcl/Tk-based GUIs named `tkemc` 1.5 and `mini` 1.4.
- a modern GL-based GUI written in python called `AXIS` 1.2<sup>3</sup>

`Tkemc` and `Mini` are most commonly used operator interfaces. They will run on Linux, Mac, and Microsoft Windows if the Tcl/Tk programming language has been installed. The Mac and Microsoft Windows version can connect to a real-time EMC2 running on a Linux machine via a network connection, allowing the monitoring of the machine from a remote location. Instructions for installing and configuring the connection between a Mac or Microsoft Machine and a PC running the EMC2 can be found in the integrators handbook.

<sup>3</sup>AXIS has been developed outside the EMC2 project, you can find information about it at <http://axis.unpy.net>

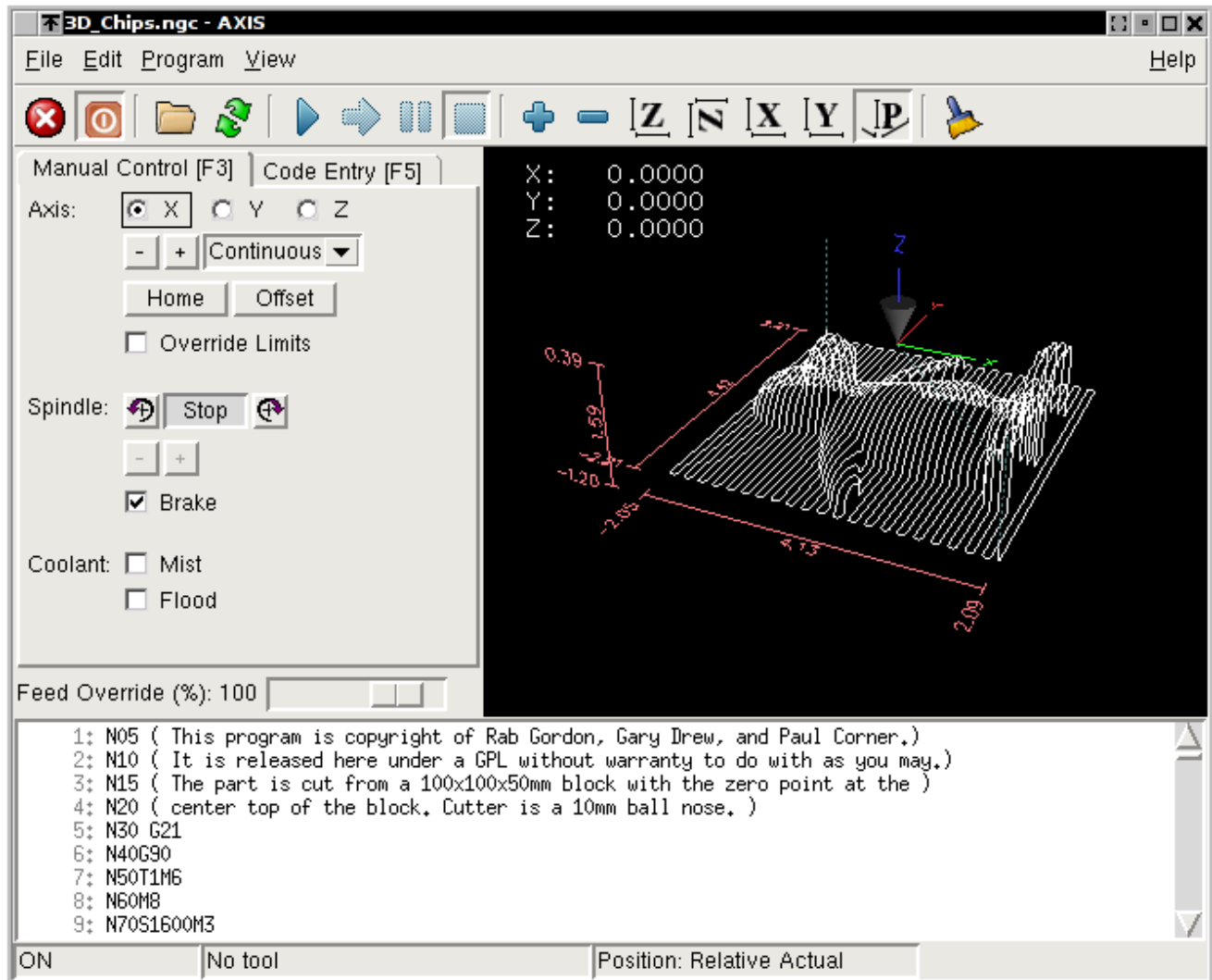


Figure 1.2: The AXIS Graphical Interface

### 1.5.2 Motion Controller EMC2MOT

Motion control includes sampling the position of the axes to be controlled, computing the next point on the trajectory, interpolating between these trajectory points, and computing an output to the motors. For servo systems, the output is based on a PID compensation algorithm. For stepper systems, the calculations run open-loop, and pulses are sent to the steppers based on whether their accumulated position is more than a pulse away from where their commanded position should be. The motion controller includes programmable software limits, interfaces to hardware limit and home switches.

The motion controller is written to be fairly generic. Initialization files (with the same syntax as Microsoft Windows INI files) are used to configure parameters such as number and type of axes (e.g., linear or rotary), scale factors between feedback devices (e.g., encoder counts) and axis units (e.g., millimeters), servo gains, servo and trajectory planning cycle times, and other system parameters. Complex kinematics for robots can be coded in C according to a prescribed function interface and linked in to replace the default 3-axis Cartesian machine kinematics routines.

The motion controllers that you will be using with your stepper motors will most likely be stepgen. The ability and requirements of each of these will be described when we get to hardware and how to configure the EMC2 for your specific hardware.

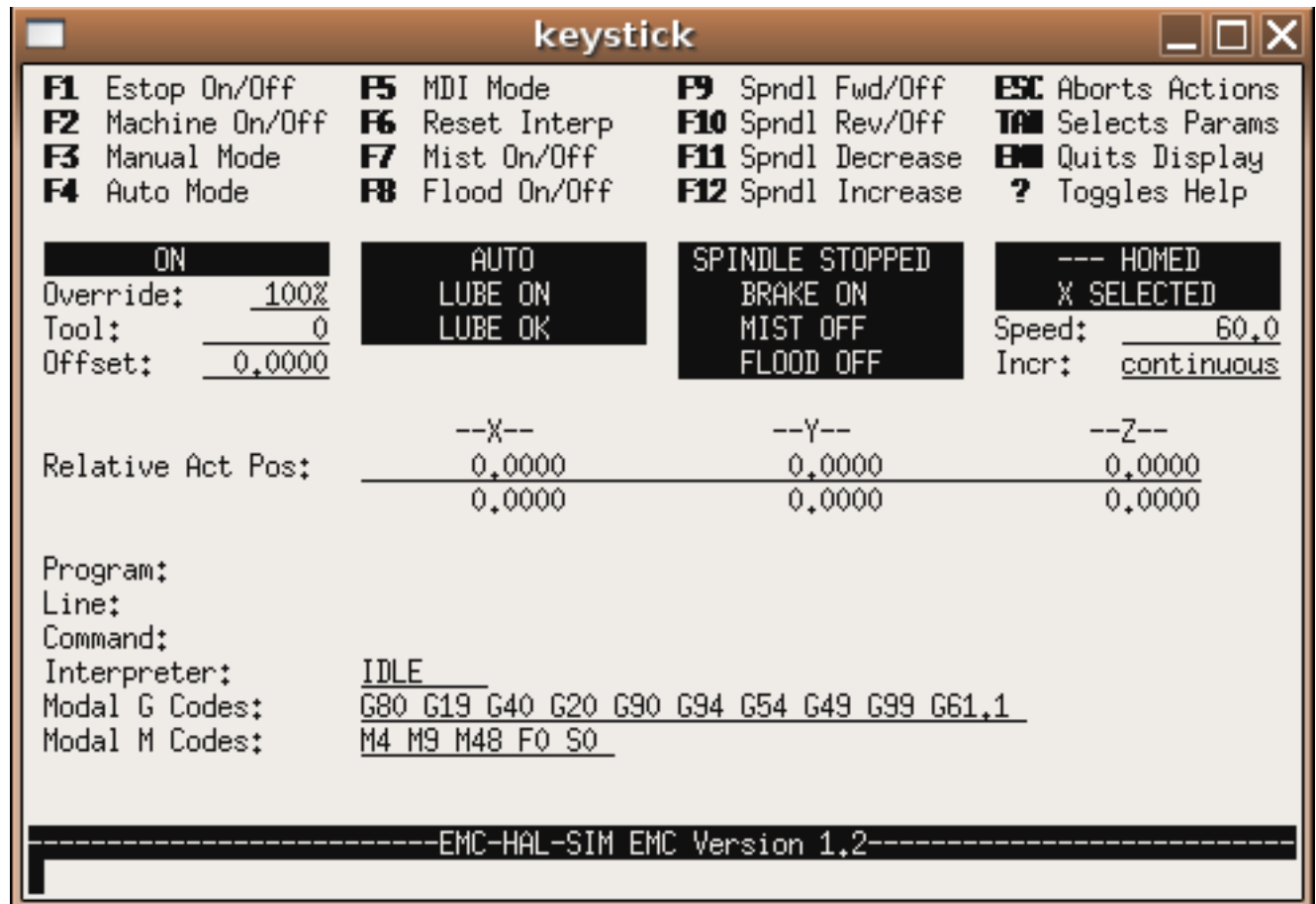


Figure 1.3: The Keystick interface

### 1.5.3 Discrete I/O Controller EMCIO

Discrete I/O controllers are highly machine-specific, and are not customizable in general using the INI file technique used to configure the more generic motion controller. However, since EMC2 uses the HAL, reconfiguration of the IO subsystem has become very powerful and flexible. EMC2 contains even a Programmable Logic Controller module (behaves just like a hardware PLC), which can be used for very complex scenarios (tool changers, etc.).

In EMC2 there is only one big I/O controller, which provides support for all kinds of actions and hardware control. All its outputs and inputs are HAL logic items (more on this later on), so you can use only the subset which interests you (or the one that fits on your hardware output: e.g. if you have only one parallel port, and no additional I/O cards).

### 1.5.4 Task Executor EMCTASK

The Task Executor is responsible for interpreting G and M code programs whose behavior does not vary appreciably between machines. G-code programming is designed to work like a machinist might work. The motion or turns of a handwheel are coded into blocks. If a machinist wanted his mill to move an inch in the +X direction at some feedrate, he might slowly turn the handwheel five turns clockwise in 20 seconds. The same machinist programming that same move for CNC might write the following block of code.

```
G1 F3 X1.000
```

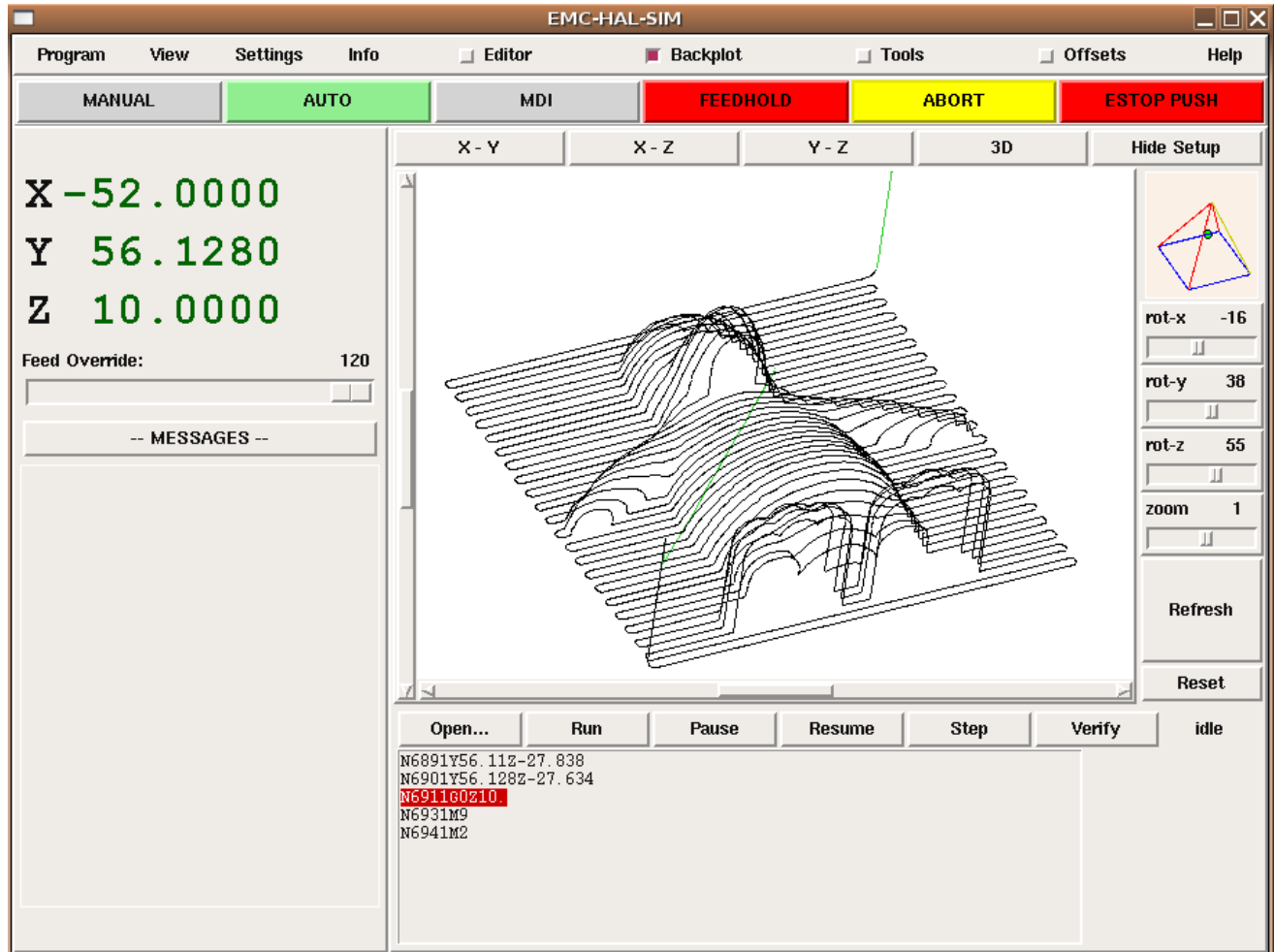


Figure 1.4: The Mini Graphical Interface

G1 means that the machine is supposed to run at a programmed feedrate rather than at the fastest speed that it can. (G0) The F3 means that it should travel at 3 inches a minute or 3 millimeters a minute if it is working in metric mode. The X1.000 assumes that the machine started at zero and is supposed to go one inch in the positive direction. You will read quite a bit more about G-code in the programming chapters.

Figure 1.7 is a block diagram of how a personal computer running the EMC2 is used to control a machine with G-code. The actual G-code can be sent using the MDI (Machine Device Interface) mode or it can be sent as a file when the machine is in Auto mode. These choices are made by the operator and entered using one of the Graphical User Interfaces available with the software.

G-code is sent to the interpreter which compares the new block with what has already been sent to it. The interpreter then figures out what needs to be done for the motion and input or output systems and sends blocks of canonical commands to the task and motion planning programs.

## 1.6 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

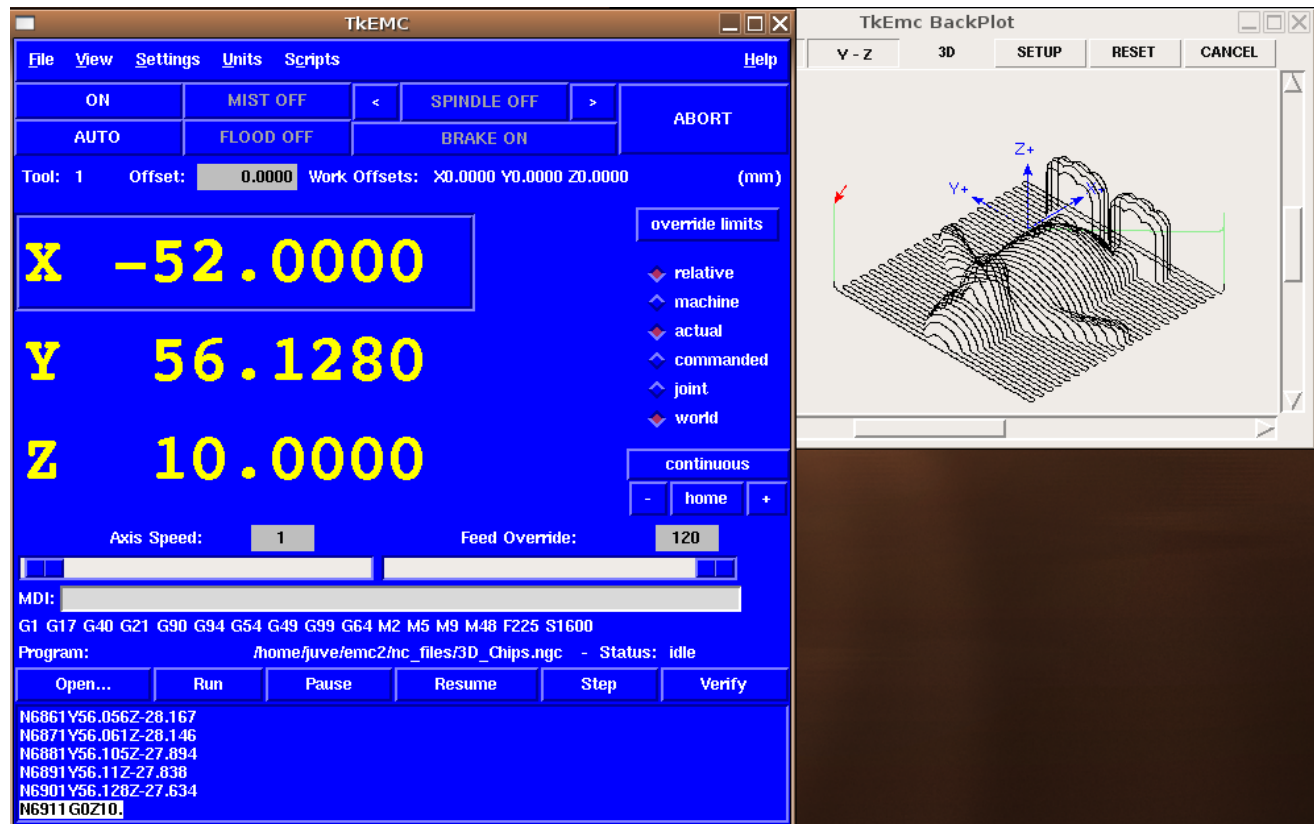


Figure 1.5: The TkEmc Graphical Interface

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 EMC2 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 EMC2 mill. You will need some information right up front here so that the following pages will make sense to you.

### 1.6.1 Modes of Operation

When an EMC2 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 EMC2 behaves. There are specific things that can be done in one mode that can not 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 separate. 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 handwheel 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.

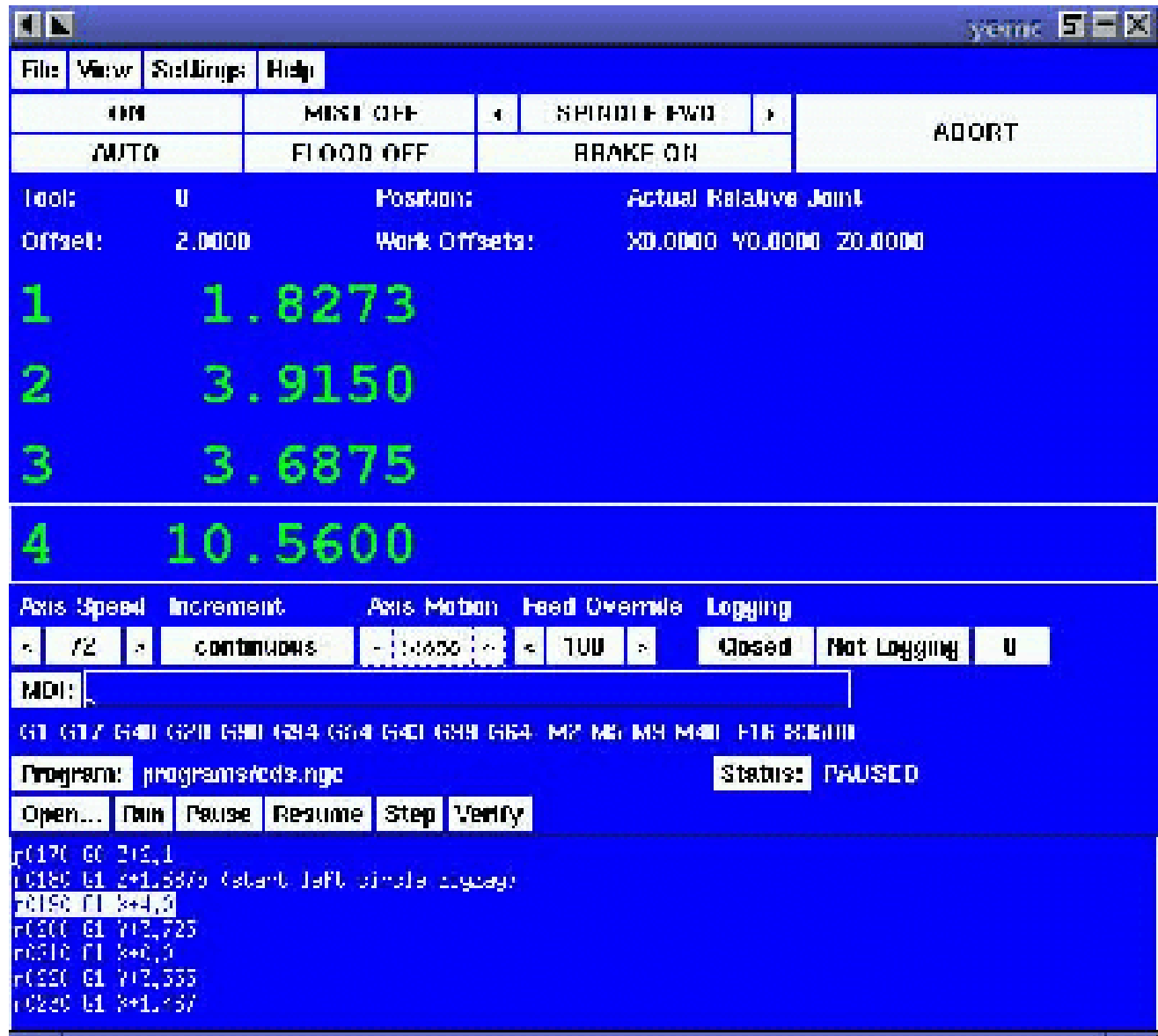


Figure 1.6: The XEMC Graphical Interface

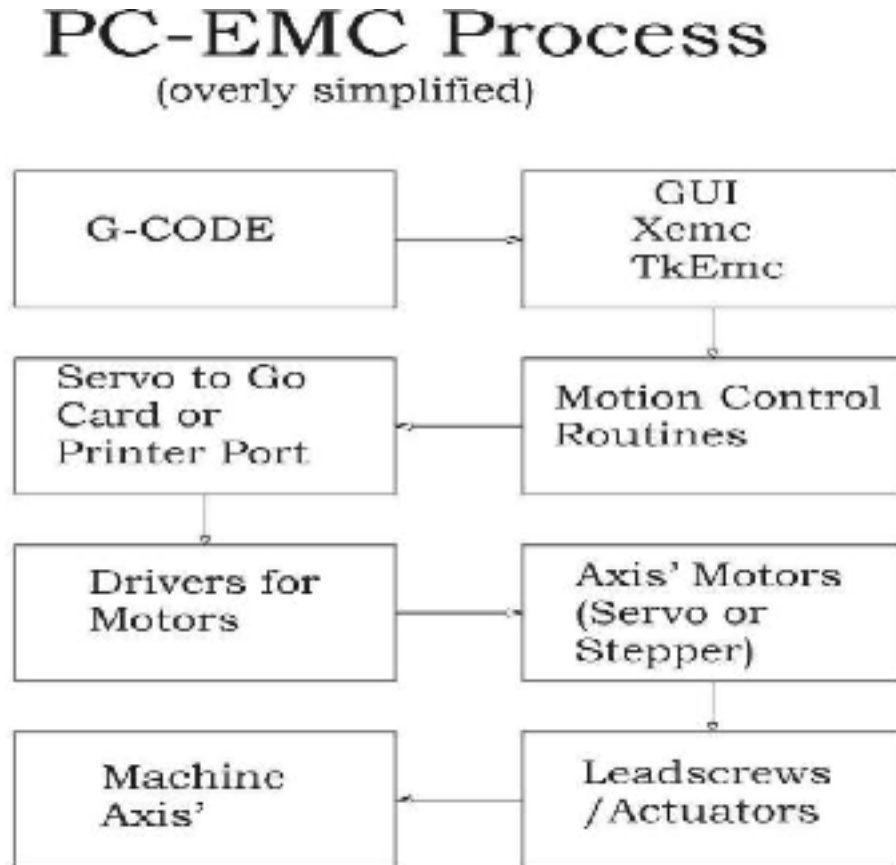
Some motion control commands are available and will cause the same changes in motion in all modes. These include ABORT, ESTOP, and FEEDRATE OVERRIDE. Commands like these should be self explanatory.

### 1.6.2 Information Display

While an EMC2 is running, each of the modules keeps up a dialog with each other and with the graphical display. It is up to the display to select from that stream of information what the operator needs to see and to arrange it on the screen in a way that makes it easy for the operator to understand. Perhaps the most important display is the mode the EMC2 is running in. You will want to keep your eye on the mode display.

Right up there with knowing what mode is active is consistent display of the position of each axis. Most of the interfaces will allow the operator to read position based upon actual or commanded

Figure 1.7: EMC2 Process Diagram



position as well as machine or relative position.

**Machine** This is the position of an axis relative to the place where it started or was homed.

**Relative** This is the position of an axis after work or tool or other offsets have been applied.

**Actual** This is the real position of the axis within the machine or relative system.

**Commanded** This is where the axis is commanded to be.

These may all be exactly the same if no offsets have been applied and there is no deadband set in the INI file. Deadband is a small distance which is assumed to be close enough – perhaps one stepper pulse or one encoder pulse.

It is also important to see any messages or error codes sent by the EMC2. These are used by operators if they need to be reminded to change a tool, to clear up problems in G-code programs, or to find out why the machine stopped running.

As you work your way through this text, you will be learning, bit by bit how to setup and run a machine with your copy of the EMC2 software. While you are learning about setting up and running a minimill here, you will be thinking of other applications and other capabilities. These are the topics of the other linuxcnc.org handbooks.

## 1.7 Thinking Like An Integrator

The biggest task of a machine integrator is figuring out how to connect a PC running the EMC2 to a machine and configuring the software so that it runs the machine correctly. Most of this is not the topic of this book but there are a few things that you will have to understand in order to make our little minimill work for us like we expect it to work.

### 1.7.1 Units

Units can be confusing. A newbie recently asked, “Does it work in inches, feet, centimeters, millimeters, or what?” There are several possible answers to this question but the best one is that it works in the units that you set it to work in.

At a machine level, we set each axis’ units to some value using an INI variable that looks like this.

```
UNITS = 1
```

or

```
UNITS = 0.03937007874016
```

Math folk will get a clue from these numbers because the long decimal number is the distance represented by one millimeter if we convert it into inches. “So,” you say, “the EMC2 uses millimeters internally.” If we use `UNITS = 1` then we have defined our user units as millimeters. If we use `UNITS = 0.03937007874016` then we have defined our user units as inches. Using similar arithmetic we could set our units to most any value we wanted. (Some of the EMC2 people who run vehicles with the EMC2 set units to kilometers or miles.)

After we have decided upon a value for the units for an axis, we tell the EMC2 how many step pulses or encoder pulses it should send or read for each unit of distance to be traveled. Once we have done this, the EMC2 knows how to count units of distance. However it is very important to understand that this counting of distance is different from the commanding of distance. You can command distance in millimeters or inches without even thinking about the units that you defined. There are G-codes that allow you to switch easily between metric and imperial.

### 1.7.2 Some things we may not want to change.

Within the EMC2 code are a few things that are not easily changed. We call these defaults. There are connections that have been made between the running components of the EMC2 that we can not easily change. We’ll see that there are displays and buttons and keyboard keys that are not easily shifted about. We’ll learn about and get used to these in the chapters ahead.

### 1.7.3 Some things we will need to change.

The EMC2 is configured with files that are read at startup and used to override the compiled defaults. No real controller will likely use the compiled defaults, so you will certainly need to edit at least some of these files to reflect the specifics of your machine.

There are five kinds of configuration files: INI, NML, TBL, VAR and HAL files. These are reflected in lower case file extensions to a file name. They may be named `EMC2.tbl` or `generic.tbl` but they do the same thing when they are read by the EMC2 as it starts up. Many users copy these and name them for the specific machine that they are editing them for. A set of these files named `Sherlinemill.ini`, `Sherlinemill.var`, `Sherlinemill.tbl` and `Sherlinemill.nml` are certainly more descriptive than a bunch of files named `generic`.

These files each contain specific information for your CNC.



- `stepper.ini`, contains all the machine parameters such as servo gains, scale factors, cycle times, units, etc. and will certainly need to be edited.
- `emc.nml` contains communication settings for shared memory and network ports you may need to override on your system, although it is likely that you can leave these settings alone.
- `stepper.tbl` contains the tool information such as which pocket contains which tool, and the length and diameter for each tool.
- `rs274ngc.var` contains variables specific to the RS-274-NGC dialect of NC code, notably for setting the persistent numeric variables for the nine work coordinate systems.

We'll get into some of the details of these files as we begin to hook up and operate our little machine.

In addition to these four files, there is a standard startup file. Back in the early days of the EMC it was common to have to start up several different tasks in different terminal windows in order to get the EMC to run a machine. Each of these tasks had to be supplied a bunch of information in the form of arguments in order to be certain that the task started the way that we expected it to. All of this was a tedious thing. All of this has been replaced with the run script file. It is named simply `'emc'`. This executable script file controls the startup of all of the modules needed to run a standard version of the EMC2. When run, it lets the user chose a certain config to run EMC2 from.

## Chapter 2

# Installing the EMC2 software

### 2.1 Introduction

One of the problems users often complained about EMC was installing the software itself. They were forced to get sources, and compile themselves, and try to set up a RT-patched Linux, etc. The developers of EMC2 chose to go with a standard distribution called Ubuntu<sup>1</sup>.

Ubuntu has been chosen, because it fits perfectly into the Open Source views of EMC2:

- Ubuntu will always be free of charge, and there is no extra fee for the "enterprise edition", we make our very best work available to everyone on the same Free terms.
- Ubuntu comes with full professional support on commercial terms from hundreds of companies around the world, if you need those services. Each new version of Ubuntu receives free security updates for 18 months after release, some versions are supported for even longer.
- Ubuntu uses the very best in translations and accessibility infrastructure that the Free Software community has to offer, to make Ubuntu usable for as many people as possible.
- Ubuntu is released regularly and predictably; a new release is made every six months. You can use the current stable release or help improve the current development release.
- The Ubuntu community is entirely committed to the principles of free software development; we encourage people to use open source software, improve it and pass it on.

### 2.2 EMC Download Page

You will find the most recent releases of EMC2 announced on [www.linuxcnc.org](http://www.linuxcnc.org). The releases of EMC2 will be done in two ways (sources and binary package). The sources (described in the Developers Handbook) consist of a tarball (`emc2-version.tar.gz`), which you should download and unpack into your home directory.

This document (oriented towards the end-user) will only try to explain how to install the binary package on the Ubuntu distribution<sup>2</sup>.

---

<sup>1</sup>"Ubuntu" is an ancient African word, meaning "humanity to others". Ubuntu also means "I am what I am because of who we all are". The Ubuntu Linux distribution brings the spirit of Ubuntu to the software world. You can read more about it at <http://www.ubuntu.com>

<sup>2</sup>For information regarding other Linux variants, check the Developers Handbook or ask for help on the emc-developers mailing list [http://sourceforge.net/mail/?group\\_id=6744](http://sourceforge.net/mail/?group_id=6744).

## 2.3 EMC2 install script - the easy way to install

Chris Radek put together a simple script to install emc2 on Ubuntu. It runs the commands explained in 2.4.

To use it you need to :

- Download the script from <http://www.linuxcnc.org/emc2-install.sh>
- Save it on your Desktop. Right-click the icon, select Properties. Go to the Permissions tab and check the box for Owner: Execute. Close the Properties window.
- Now double-click the emc2-install.sh icon, and select "Run in Terminal". A terminal will appear and you will be asked for your password.
- When the installation asks if you are sure you want to install the EMC2 packages, hit Enter to accept. Now just allow the install to finish.
- When it is done, you must reboot (System > Log Out > Restart the Computer), and when you log in again you can run EMC2 by selecting it on the Applications > Other menu.
- If you aren't ready to set up a machine configuration, try the sim-AXIS configuration; it runs a "simulated machine" that requires no attached hardware.
- Now that the initial installation is done, Ubuntu will prompt you when updates of EMC2 or its supporting files are available. When they are, you can update them easily and automatically with the Update Manager.

## 2.4 Manual installing using apt commands.

The following few section will describe how to install EMC2 using a console and apt-commands. If you know a bit about Linux and Debian-flavored distributions this might be trivial. If not, you might consider reading 2.3.

First add the repository to `/etc/apt/sources.list`:

```
$ sudo sh -c 'echo "deb http://dsplabs.cs.upt.ro/emc2/ breezy emc2" >>/etc/apt/sources.list;'
$ sudo sh -c 'echo "deb-src http://dsplabs.cs.upt.ro/emc2/ breezy emc2" >>/etc/apt/sources.list'
```

Then update & get emc2-axis.

```
$ sudo apt-get update
$ sudo apt-get install emc2-axis
```

This command will install the emc2-axis<sup>3</sup> package along with all dependencies<sup>4</sup>.

You might get warnings that the packages are from an untrusted source (this means your computer doesn't recognize the GPG signature on the packages). To correct that issue the following commands:

```
$ gpg --keyserver pgpkeys.mit.edu --recv-key BC92B87F
$ gpg -a --export BC92B87F | sudo apt-key add -
```

<sup>3</sup>The emc2-axis package is the AXIS gui packaged for emc2.

<sup>4</sup>The dependencies are one of the nicest thing in Debian based distributions. They assure you have everything installed that you need. In the case of emc2 it's even a RT-patched kernel, and all needed libraries.

**Part II**

**Configuring EMC2**

# Chapter 3

## Introduction

### 3.1 What is HAL?

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 complicated system. The “Hardware” part is because HAL was originally designed to make it easier to configure EMC 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.

#### 3.1.1 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.

##### 3.1.1.1 Part Selection

The machine builder does not need to worry 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.

##### 3.1.1.2 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 get connected 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.

### 3.1.1.3 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.

### 3.1.1.4 Testing

Very few machines work right the first time. While testing the builder may use a meter to see if 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 equivalent of a voltmeter, oscilloscope, signal generator, and other tools for testing and tuning a system. The same commands used to build the system can be used to make changes as needed.

## 3.1.2 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 EMC.

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.

## 3.2 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 behaviour, 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 8 types<sup>1</sup>, as follows:

- BIT - a single TRUE/FALSE or ON/OFF value
- FLOAT - a 32 bit floating point value, with approximately 24 bits of resolution and over 200 bits of dynamic range.
- U8 - an 8 bit unsigned integer, legal values are 0 to +255
- S8 - an 8 bit signed integer, legal values are -128 to +127
- U16 - a 16 bit unsigned integer, legal values are 0 to +65535
- S16 - a 16 bit signed integer, legal values are -32768 to +32767
- U32 - a 32 bit unsigned integer, legal values are 0 to +4294967295
- S32 - a 32 bit signed integer, legal values are -2147483648 to +2147483647

<sup>1</sup>There has been some discussion about whether we really need all the integer types. Maybe they will be reduced or eliminated later. Most signals and pins will be either floats or bits.

**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.

For now a quick example will help get the concept across. 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.

### 3.3 HAL components

Each HAL component is a piece of software with well defined inputs, outputs, and behaviour, that can be installed and interconnected as needed. This section lists available components and a brief description of what they do. Complete details for each component are available later in this document.

#### 3.3.1 External Programs with HAL hooks

**motion** A realtime module that accepts NML 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 (note: right now experimental), it is intended to work as a full User Interface using external knobs & switches

#### 3.3.2 Internal Components

**stepgen** Software step pulse generator with position loop. See section ??

**freqgen** Software step pulse generator. See section ??

**encoder** Software based encoder counter. See section ??

**pid** Proportional/Integral/Derivative control loops. See section ??

**siggen** A sine/cosine/triangle/square wave generator for testing. See section ??

**supply** a simple source for testing

**blocks** assorted useful components (mux, demux, or, and, integ, ddt, limit, wcomp, etc.)



### 3.3.3 Hardware Drivers

**hal\_ax5214h** A driver for the Axiom Measurement & Control AX5241H digital I/O board

**hal\_m5i20** Mesa Electronics 5i20 board

**hal\_motenc** Vital Systems MOTENC-100 board

**hal\_parport** PC parallel port. See section [5.1](#)

**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

### 3.3.4 Tools and Utilities

**halcmd** Command line tool for configuration and tuning. See section ??

**halgui** GUI tool for configuration and tuning (not implemented yet).

**halmeter** A handy multimeter for HAL signals. See section ??

**halscope** A full featured digital storage oscilloscope for HAL signals. See section ??

Each of these building blocks is described in detail in later chapters.

## 3.4 Tinkertoys, Erector Sets, Legos and the HAL

A first introduction to HAL concepts can be mind boggling. Building anything with blocks can be a challenge but some of the toys that we played with as kids can be an aid to building things with the HAL.

### 3.4.1 Tower

I'm watching as my son and his six year old daughter build a tower from a box full of random sized blocks, rods, jar lids and such. The aim is to see how tall they can make the tower. The narrower the base the more blocks left to stack on top. But the narrower the base, the less stable the tower. I see them studying both the next block and the shelf where they want to place it to see how it will balance out with the rest of the tower.

The notion of stacking cards to see how tall you can make a tower is a very old and honored way of spending spare time. At first read, the integrator may have gotten the impression that building a HAL was a bit like that. It can be but with proper planning an integrator can build a stable system as complex as the machine at hand requires.

### 3.4.2 Erector Sets<sup>2</sup>

What was great about the sets was the building blocks, metal struts and angles and plates, all with regularly spaced holes. You could design things and hold them together with the little screws and nuts.

---

<sup>2</sup>The Erector Set was an invention of AC Gilbert

I got my first erector set for my fourth birthday. I know the box suggested a much older age than I was. Perhaps my father was really giving himself a present. I had a hard time with the little screws and nuts. I really needed four arms, one each for the screwdriver, screw, parts to be bolted together, and nut. Perseverance, along with father's eventual boredom, got me to where I had built every project in the booklet. Soon I was lusting after the bigger sets that were also printed on that paper. Working with those regular sized pieces opened up a world of construction for me and soon I moved well beyond the illustrated projects.

Hal components are not all the same size and shape but they allow for grouping into larger units that will do useful work. In this sense they are like the parts of an Erector set. Some components are long and thin. They essentially connect high level commands to specific physical pins. Other components are more like the rectangular platforms upon which whole machines could be built. An integrator will quickly get beyond the brief examples and begin to bolt together components in ways that are unique to them.

### 3.4.3 Tinkertoys<sup>3</sup>

Wooden Tinker toys had a more humane feel than the cold steel of Erector Sets. The heart of construction with Tinker Toys was a round connector with eight holes equally spaced around the circumference. It also had a hole in the center that was perpendicular to all the holes around the hub.

Hubs were connected with rods of several different lengths. Builders would make large wheels by using these rods as spokes sticking out from the center hub.

My favorite project was a rotating space station. Short spokes radiated from all the holes in the center hub and connected with hubs on the ends of each spoke. These outer hubs were connected to each other with longer spokes. I'd spend hours dreaming of living in such a device, walking from hub to hub around the outside as it slowly rotated producing near gravity in weightless space. Supplies traveled through the spokes in elevators that transferred them to and from rockets docked at the center hub while they transferred their precious cargos.

The idea of one pin or component being the hub for many connections is also an easy concept within the HAL. Examples two and four (see section 4) connect the meter and scope to signals that are intended to go elsewhere. Less easy is the notion of a hub for several incoming signals but that is also possible with proper use of functions within that hub component that handle those signals as they arrive from other components.

Another thought that comes forward from this toy is a mechanical representation of HAL threads. A thread might look a bit like a centipede, caterpillar, or earwig. A backbone of hubs, HAL components, strung together with rods, HAL signals. Each component takes in its own parameters and input pins and passes on output pins and parameters to the next component. Signals travel along the backbone from end to end and are added to or modified by each component in turn.

Threads are all about timing and doing a set of tasks from end to end. A mechanical representation is available with Tinkertoys also when we think of the length of the toy as a measure of the time taken to get from one end to the other. A very different thread or backbone is created by connecting the same set of hubs with different length rods. The total length of the backbone can be changed by the length of rods used to connect the hubs. The order of operations is the same but the time to get from beginning to end is very different.

---

<sup>3</sup>Tinkertoy is now a registered trademark of the Hasbro company.

### 3.4.4 A Lego Example<sup>4</sup>

When Lego blocks first arrived in our stores they were pretty much all the same size and shape. Sure there were half sized one and a few quarter sized as well but that rectangular one did most of the work. Lego blocks interconnected by snapping the holes in the underside of one onto the pins that stuck up on another. By overlapping layers, the joints between could be made very strong, even around corners or tees.

I watched my children and grandchildren build with legos – the same legos. There are a few thousand of them in an old ratty but heavy duty cardboard box that sits in a corner of the recreation room. It stays there in the open because it was too much trouble to put the box away and then get it back out for every visit and it is always used during a visit. There must be Lego parts in there from a couple dozen different sets. The little booklets that came with them are long gone but the magic of building with interlocking pieces all the same size is something to watch.

Notice the following description of building a set of motion components in the HAL and how much like a wall of lego blocks it is.

The motion module exports a pin for each axis in cartesean space, and another pin for each axis in joint space. When it is loaded, it automatically creates a "jumper" signal for each axis, and automatically connects those signals from the joint pin to the cartesean pin. So you automatically have "trivkins" as soon as you load the motion module. (trivkins – trivial kinematics is the case where each motor moves a single axis at 90 degrees to the others)

The motion module is like a pair of legos in a line end to end. Trivkins is just like a single block overlapping the two. The in and out motion pins are plugged into each other by the block resting above. But the parallel goes on.

If you need some other kinematics, you then load a specific kins component. This component "knows" the names of the pins that the motion module uses for each axis, both joint and cartesean. When the module loads, it again automatically creates signals and connects its own pins to the motion module's pins (which will disconnect the "jumpers"). It could also know the thread names used by the motion module, and could automatically add it's own functions to those threads.

Trivkins is removed so that the motion blocks can be spread apart and by using other blocks, a different bridge is built between input and output pins. In Lego terms, trivkins might be a gray block and xxkins might be a yellow block.

So the net result is that 24 HAL signals and two HAL functions are configured, with no action needed by the integrator other than loading the module. (24 signals are from 6 axis \* 2 because we have joint and cartesean \* 2 because we have forward and inverse kinematics. Two functions because we have forward and inverse.) Because these HAL signals exist, they can be metered or scoped or whatever for testing. But because both modules know their names and know how to automatically connect them, the integrator doesn't have to know or care.

This kind of automatic HAL configuration is possible because all kinematics modules "plug in" the same way.

---

<sup>4</sup>The Lego name is a trademark of the Lego company.

### 3.5 Timing Issues In HAL

Threads is going to take a major intellectual push because 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 a HAL-aware realtime version of ClassicLadder would export 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 10mS, 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 EMC we might have a 15uS thread, a 1mS thread, and a 10mS thread. These would be created based on "Period", "ServoPeriod", and "TrajPeriod" respectively - the actual times would depend on the ini. That is one part of the config process, and although it could be done manually, it would normally be automatic.

The next step is to decide what each thread needs to do. Some of those decisions would also be automatic - the motion module would automatically connect its "PlanTrajectory" function to the TrajPeriod thread, and its "ControlMotion" function to the ServoPeriod thread.

Other connections would be made by the integrator (at least the first time). 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 fast thread, along with the parport function(s) to write the steps to the port.

### 3.6 Dynamic Linking and Configuration

It is indeed possible to configure HAL with a form of dynamic linking. But it is different than DLLs as used by Microsoft(tm) or shared libraries as used in Linux. Both DLLs and shared libraries essentially say "Here I am, I have this code you might want to use", where "you" is other modules. Then when those other modules or programs are loaded, they say "I need a function called 'X', is there one?" and if the answer is YES, they link to it.

With HAL, a component still says "Here I am, I have this code you might want to use", but "you" is the system integrator. The integrator gets to decide what functions are used and doesn't have to worry about another module needing "function X" and not finding it.

HAL can follow the normal DLL model as well. Although most components will simply export pins, functions, and parameters, and then wait for the integrator (or a saved file) to interconnect them, we can write modules that (attempt to) make connections when they are installed. One specific place where this would work well is kinematics as illustrated in the Lego section [3.4.4](#) .

# Chapter 4

## HAL Tutorial

### 4.1 Before we start

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. As noted above `halcmd` 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.

#### 4.1.1 Notation

Command line examples are presented in **bold typewriter** font. Responses from the computer will be in `typewriter` font. Text inside square brackets `[like-this]` is optional. Text inside angle brackets `<like-this>` represents a field that can take on different values, and the adjacent paragraph will explain the appropriate values. Text items separated by a vertical bar means that one or the other, but not both, should be present. All command line examples assume that you are in the `emc2/` directory, and paths will be shown accordingly when needed.

#### 4.1.2 Root Privileges

In the beginning days of HAL there was quite often the need to run things with root privileges. Most of those things were related to the fact that HAL uses kernel modules to do much of it's work, and because it also can access hardware directly. Knowing that it is usually safer to avoid doing day-to-day work as root, most of those things were reworked so that very limited root privileges are required. If you are running a version of EMC2 and HAL more recent than early 2006, you can pretty much ignore this section.

To get around the need for root, `emc2` uses a small program called `emc_module_helper`, which during the build process gets `setuid` status (thus has root privileges). This small software module takes care of the `insmod/rmmod` commands that are needed to insert/delete modules from the HAL. You can use this program directly, but it's lots easier to use the **`halcmd loadrt & unloadrt`** commands ??

Here is an example of what happens when you don't have root privileges:

```
emc2$ bin/hal_parport 0278
PARPORT: ERROR: could not get I/O permission
emc2$
```

As an alternative to logging in as root, you can use the `sudo` command or the `su -c` command. The `sudo` command is very convenient to use, and does not require you to know the root password. However, it needs to be configured by someone who does know the root password. The configuration determines who may use `sudo`, and what commands they can use it for. We will not discuss `sudo` configuration here, try `man sudo` and/or talk to your system administrator. If `sudo` is properly configured, here is what happens:

```
emc2$ sudo bin/hal_parport 0278
Password: <enter your password>
PARPORT: installed driver for 1 ports
emc2$
```

As an added convenience, `sudo` remembers your password for a short time, so if you enter another `sudo` command within the time limit (usually 5 minutes) you don't have to type your password again.

The `su -c` command does not require configuration, but does require you to know the root password, and to type it in for every command. You also must put quotes around the command you are trying to run:

```
emc2$ su -c "bin/hal_parport 0278"
Password: <enter root password>
PARPORT: installed driver for 1 ports
emc2$
```

To avoid cluttering up the examples, we will not show `sudo` or `su -c`. Instead, commands that require root privileges will be preceded by `#`, and other commands will be preceded by `$`.

```
emc2$ ls bin
emc2# bin/hal_parport 0278
```

### 4.1.3 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 EMC 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 EMC, you may want to study `emc2/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.

For this tutorial, we are going to assume that you have successfully compiled the `emc2/` source tree. In that case, all you need to do is load the required RTOS and RTAPI modules into memory. Just run the following command (needs root privileges):

```
emc2# scripts/realtime start
```

With the realtime OS and RTAPI loaded, we can move into the first example.

## 4.2 A Simple Example

### 4.2.1 Loading a realtime component

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 section ?? of this document. It is a realtime component, implemented as a Linux kernel module and located in the directory `emc2/rtlib/`. To load `siggen` use the `halcmd loadrt siggen` command:

```
emc2$ bin/halcmd loadrt siggen
emc2$
```

### 4.2.2 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 `halcmd` reference in 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:

```
emc2$ bin/halcmd show comp
Loaded HAL Components:
ID  Type  Name
02  User  halcmd21345
01  RT    siggen
emc2$
```

Since `halcmd` itself is a HAL component, it will always show up in the list<sup>1</sup>. 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.

Next, let’s see what pins `siggen` makes available:

```
emc2$ bin/halcmd show pin
Component Pins:
Owner  Type  Dir  Value      Name
02     float -W    0.00000e+00 siggen.0.cosine
02     float -W    0.00000e+00 siggen.0.sawtooth
02     float -W    0.00000e+00 siggen.0.sine
02     float -W    0.00000e+00 siggen.0.square
02     float -W    0.00000e+00 siggen.0.triangle
emc2$
```

This command displays all of the pins in the HAL - a complex system could have dozens or hundreds of pins. But right now there are only five pins. All five of these pins are floating point, and all five carry data out of the `siggen` component. Since we have not yet executed the code contained within the component, all the pins have a value of zero.

The next step is to look at parameters:

<sup>1</sup>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.

```
emc2$ bin/halcmd show param
Parameters:
Owner  Type  Dir   Value      Name
02     float -W    1.00000e+00 siggen.0.amplitude
02     float -W    1.00000e+00 siggen.0.frequency
02     float -W    0.00000e+00 siggen.0.offset
02     s32   R-    0          siggen.0.update.time
02     s32   RW    0          siggen.0.update.tmax
emc2$
```

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 writeable 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:

```
emc2$ bin/halcmd show funct
Exported Functions:
Owner CodeAddr  Arg   FP  Users  Name
02    C48E31C4 C48D2054 YES  0    siggen.0.update
emc2$
```

The `siggen` component exported a single function. It requires floating point. It is not currently linked to any threads, so "users" is zero<sup>2</sup>.

### 4.2.3 Making realtime code run

To actually run the code contained in the function `siggen.0.update`, we need a realtime thread. Eventually `halcmd` will have a `newthread` command that can be used to create a thread, but that requires some significant internal changes. For now, we have a component called `threads` that is used to create a new thread. Lets create a thread called `test-thread` with a period of 1mS (1000000nS):

```
emc2$ bin/halcmd loadrt threads name1=test-thread period1=1000000
```

Let's see if that worked:

```
emc2$ bin/halcmd show thread
Realtime Threads:
  Period  FP  Name      (Time, Max-Time)
    999849 YES  test-thread ( 0, 0 )
emc2$
```

It did. The period is not exactly 1000000nS 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:

<sup>2</sup>The `codeaddr` and `arg` fields were used in development, and should probably be removed from the `halcmd` listing.



```
emc2$ bin/halcmd addf siggen.0.update test-thread
emc2$
```

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:

```
emc2$ bin/halcmd show thread
Realtime Threads:
  Period  FP  Name      (Time, Max-Time)
  999849 YES test-thread ( 0, 0 )
                1 siggen.0.update
emc2$
```

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:

```
emc2$ bin/halcmd start
emc2$
```

Now the signal generator is running. Let's look at it's output pins:

```
emc2$ bin/halcmd show pin
Component Pins:
Owner  Type  Dir  Value      Name
02     float -W   5.61498e-01 siggen.0.cosine
02     float -W  -6.89775e-01 siggen.0.sawtooth
02     float -W   8.27478e-01 siggen.0.sine
02     float -W  -1.00000e+00 siggen.0.square
02     float -W   3.79549e-01 siggen.0.triangle
emc2$ bin/halcmd show pin
Component Pins:
Owner  Type  Dir  Value      Name
02     float -W   9.23063e-01 siggen.0.cosine
02     float -W  -8.74322e-01 siggen.0.sawtooth
02     float -W   3.84649e-01 siggen.0.sine
02     float -W  -1.00000e+00 siggen.0.square
02     float -W   7.48645e-01 siggen.0.triangle
emc2$
```

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, and it happened to be at -1.0 for both commands.

#### 4.2.4 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:

```
emc2$ bin/halcmd setp siggen.0.amplitude 5
emc2$
```

Check the parameters and pins again:

```
emc2$ bin/halcmd show param
Parameters:
Owner  Type  Dir  Value          Name
02    float -W   5.00000e+00   siggen.0.amplitude
02    float -W   1.00000e+00   siggen.0.frequency
02    float -W   0.00000e+00   siggen.0.offset
emc2$ bin/halcmd show pin
Component Pins:
Owner  Type  Dir  Value          Name
02    float -W  -1.66602e+00   siggen.0.cosine
02    float -W   1.95935e+00   siggen.0.sawtooth
02    float -W  -4.71428e+00   siggen.0.sine
02    float -W   5.00000e+00   siggen.0.square
02    float -W  -1.08130e+00   siggen.0.triangle
emc2$ bin/halcmd show pin
Component Pins:
Owner  Type  Dir  Value          Name
02    float -W  -3.82623e+00   siggen.0.cosine
02    float -W  -1.11309e+00   siggen.0.sawtooth
02    float -W   3.21869e+00   siggen.0.sine
02    float -W  -5.00000e+00   siggen.0.square
02    float -W  -2.77382e+00   siggen.0.triangle
emc2$
```

Note that the value of parameter `siggen.0.amplitude` has changed to 5.000, and that the pins now have larger values. The square wave output now switches from +5.0 to -5.0, and we happened to catch it switching this time.

#### 4.2.5 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:

```
emc2$ bin/halcmd save
# components
loadrt threads name1=test-thread period1=1000000
loadrt siggen
# signals
# links
# parameter values
setp siggen.0.amplitude 5.00000e+00
setp siggen.0.frequency 1.00000e+00
setp siggen.0.offset 0.00000e+00
# realtime thread/function links
addf siggen.0.update test-thread
emc2$
```

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:

```
emc2$ bin/halcmd save >saved.hal
emc2$
```

### 4.2.6 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 `halcmd -f <filename>` which reads commands from a file:

```
emc2$ bin/halcmd -f saved.hal
emc2$
```

## 4.3 Looking at the HAL with 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 `siggen` is already loaded. If not, we can load it just like we did before:

```
emc2$ scripts/realtime start
emc2$ loadrt siggen
emc2$ loadrt threads name1=test-thread period1=1000000
emc2$ bin/halcmd addf siggen.0.update test-thread
emc2$ bin/halcmd start
emc2$ bin/halcmd setp siggen.0.amplitude 5
emc2$
```

### 4.3.1 Starting halmeter

At this point we have the `siggen` component loaded and running. It's time to start `halmeter`. Since `halmeter` is a GUI app, X must be running. We can start `halmeter` in the background by following it's name with a `'&'`:

```
emc2$ bin/halmeter &
[1] 22093
emc2$
```

Since we started `halmeter` in the background, Linux prints its process id `[1] 22093` and immediately returns to the shell prompt. At the same time, a `halcmd` window opens on your screen, looking something like figure 4.1. Note that you don't have to run `halmeter` in the background. If you omit `'&'`, it will start and behave exactly the same, but you won't get your shell prompt back until you exit from `halmeter`.

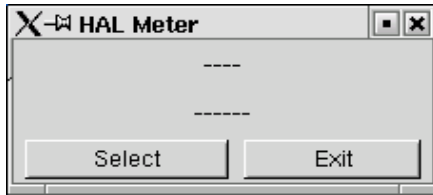


Figure 4.1: Halmeter at startup, nothing selected



Figure 4.2: Halmeter source selection dialog

### 4.3.2 Using halmeter

The meter in figure 4.1 isn't very useful, because it isn't displaying anything. To change that, click on the 'Select' button, which will open the probe selection dialog (figure 4.2).

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.triangle` first, so click on it then click the 'OK' button. The probe selection dialog will close, and the meter looks something like figure 4.3.

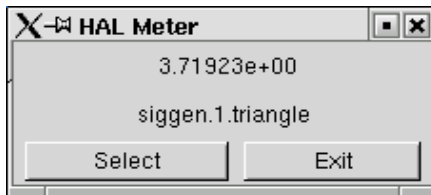


Figure 4.3: Halmeter displaying the value of a pin

You should see the value changing as `siggen` generates its triangle wave. Halmeter refreshes its display about 5 times per second.

If you want to quickly look at a number of pins, you can use the 'Accept' button in the source selection dialog. Click on 'Select' to open the dialog again. This time, click on another pin, like `siggen.0.cosine`, and then click 'Accept'. When you click 'Accept', the meter immediately begins to display the newly selected item, but the dialog does not close. Try displaying a parameter instead of a pin. Click on the 'Parameters' tab, then select a parameter and click 'Accept' again. You can very quickly move the "meter probes" from one item to the next with a couple of clicks.

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. <sup>3</sup>

---

<sup>3</sup>Halmeter is due for a rewrite. The rewrite will do a number of things to make it nicer. Scientific notation will go away - it is a pain to read. Some form of ranging (including autoranging) will be added to allow it to display a wide range of numbers without using scientific notation. An "analog bar graph" display will also be added to give a quick indication of trends. When the rewrite is done, these screenshots and the accompanying text will be revised to match the new version.

## 4.4 A slightly more complex 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:

```
emc2$ bin/halcmd unloadrt all
emc2$ scripts/realtime restart
emc2$
```

### 4.4.1 Installing the components

Now we are going to load the step pulse generator component. For a detailed description of this component refer to section ???. For now, we can skip the details, and just run the following commands:<sup>4</sup>

```
emc2$ bin/halcmd loadrt freqgen step_type=0,0
emc2$ bin/halcmd loadrt siggen
emc2$ bin/halcmd loadrt threads name1=fast fp1=0 period1=50000 \
name2=slow period2=1000000
emc2$
```

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 micro-seconds and a slow one with a period of 1mS. 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:

```
emc2$ bin/halcmd show pin
Component Pins:
Owner  Type  Dir  Value      Name
03     float -W    0.00000e+00 siggen.0.cosine
03     float -W    0.00000e+00 siggen.0.sawtooth
03     float -W    0.00000e+00 siggen.0.sine
03     float -W    0.00000e+00 siggen.0.square
03     float -W    0.00000e+00 siggen.0.triangle
02     s32   -W    0          freqgen.0.counts
02     bit   -W    FALSE     freqgen.0.dir
02     float -W    0.00000e+00 freqgen.0.position
02     bit   -W    FALSE     freqgen.0.step
02     float R-  0.00000e+00 freqgen.0.velocity
02     s32   -W    0          freqgen.1.counts
02     bit   -W    FALSE     freqgen.1.dir
02     float -W    0.00000e+00 freqgen.1.position
02     bit   -W    FALSE     freqgen.1.step
02     float R-  0.00000e+00 freqgen.1.velocity
emc2$ bin/halcmd show param
```

<sup>4</sup>The “\” at the end of a long line indicates line wrapping (needed for formatting this document). When entering the commands at the command line, simply skip the “\” (do not hit enter) and keep typing from the following line.

```

Parameters:
Owner  Type  Dir   Value      Name
03    float -W    1.00000e+00  siggen.0.amplitude
03    float -W    1.00000e+00  siggen.0.frequency
03    float -W    0.00000e+00  siggen.0.offset
02    u8    -W    1 (01)    freqgen.0.dirhold
02    u8    -W    1 (01)    freqgen.0.dirsetup
02    float R-    0.00000e+00  freqgen.0.frequency
02    float -W    0.00000e+00  freqgen.0.maxaccel
02    float -W    1.00000e+15  freqgen.0.maxfreq
02    float -W    1.00000e+00  freqgen.0.position-scale
02    s32   R-    0         freqgen.0.rawcounts
02    u8    -W    1 (01)    freqgen.0.steplen
02    u8    -W    1 (01)    freqgen.0.stepspace
02    float -W    1.00000e+00  freqgen.0.velocity-scale
02    u8    -W    1 (01)    freqgen.1.dirhold
02    u8    -W    1 (01)    freqgen.1.dirsetup
02    float R-    0.00000e+00  freqgen.1.frequency
02    float -W    0.00000e+00  freqgen.1.maxaccel
02    float -W    1.00000e+15  freqgen.1.maxfreq
02    float -W    1.00000e+00  freqgen.1.position-scale
02    s32   R-    0         freqgen.1.rawcounts
02    u8    -W    1 (01)    freqgen.1.steplen
02    u8    -W    1 (01)    freqgen.1.stepspace
02    float -W    1.00000e+00  freqgen.1.velocity-scale
emc2$

```

#### 4.4.2 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`. To create them we use the `newsig` command. We also need to specify the type of data that will flow through these “wires”, in this case it is floating point:

```

emc2$ bin/halcmd newsig X_vel float
emc2$ bin/halcmd newsig Y_vel float
emc2$

```

To make sure that worked, we can look at all the signals:

```

emc2$ bin/halcmd show sig
Signals:
Type      Value      Name
float    0.00000e+00  X_vel
float    0.00000e+00  Y_vel
emc2$

```

The next step is to connect the signals to component pins. 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 `linksp` command.

```
emc2$ bin/halcmd linksp X_vel siggen.0.cosine
emc2$
```

To see the effect of the `linksp` command, we show the signals again:

```
emc2$ bin/halcmd show sig
Signals:
Type      Value      Name
float     0.00000e+00 X_vel
                                     <== siggen.0.cosine
float     0.00000e+00 Y_vel
emc2$
```

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:

```
emc2$ bin/halcmd linksp X_vel freqgen.0.velocity
emc2$
```

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:

```
emc2$ bin/halcmd linksp Y_vel siggen.0.sine
emc2$ bin/halcmd linksp Y_vel freqgen.1.velocity
emc2$
```

Now let’s take a final look at the signals and the pins connected to them:

```
emc2$ bin/halcmd show sig
Signals:
Type      Value      Name
float     0.00000e+00 X_vel
                                     <== siggen.0.cosine
                                     ==> freqgen.0.velocity
float     0.00000e+00 Y_vel
                                     <== siggen.0.sine
                                     ==> freqgen.1.velocity
emc2$
```

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 `freqgen.0.velocity`.

### 4.4.3 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:



```
emc2$ bin/halcmd show funct
Exported Functions:
Owner CodeAddr  Arg    FP  Users  Name
  03  D89051C4  D88F10FC YES    0  siggen.0.update
  02  D8902868  D88F1054 YES    0  freqgen.capture_position
  02  D8902498  D88F1054 NO     0  freqgen.make_pulses
  02  D89026F0  D88F1054 YES    0  freqgen.update_freq
emc2$
```

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, `freqgen.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 `freqgen.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, `freqgen.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 `freqgen.update_freq` to load the new values into the step pulse generator. Finally we need to run `freqgen.make_pulses` as fast as possible for smooth pulses. Because we don't use position feedback, we don't need to run `freqgen.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:

```
emc2$ bin/halcmd show thread
Realtime Threads:
  Period  FP  Name      ( 0, 0 )
    1005720 YES  slow      ( 0, 0 )
     50286 NO   fast      ( 0, 0 )
emc2$
```

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 `freqgen.update_freq`. The second thread is `fast`, which runs every 50 microseconds, and does not support floating point. We will use it for `freqgen.make_pulses`. To connect the functions to the proper thread, we use the `addf` command. We specify the function first, followed by the thread:

```
emc2$ bin/halcmd addf siggen.0.update slow
emc2$ bin/halcmd addf freqgen.update_freq slow
emc2$ bin/halcmd addf freqgen.make_pulses fast
emc2$
```

After we give these commands, we can run the `show thread` command again to see what happened:

```

emc2$ bin/halcmd show thread
Realtime Threads:
  Period  FP  Name      (Time, Max-Time)
  1005720 YES  slow      ( 0, 0 )
           1 siggen.0.update
           2 freqgen.update-freq
  50286   NO  fast      ( 0, 0 )
           1 freqgen.make-pulses
emc2$

```

Now each thread is followed by the names of the functions, in the order in which the functions will run.

#### 4.4.4 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 `freqgen.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:

```

emc2$ bin/halcmd setp freqgen.0.velocity-scale 10000
emc2$ bin/halcmd setp freqgen.1.velocity-scale 10000
emc2$

```

This velocity scaling means that when the pin `freqgen.0.velocity` 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.

#### 4.4.5 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:

```

emc2$ bin/halcmd start
emc2$

```

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.

## 4.5 Taking a closer look with 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.

### 4.5.1 Starting 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. Before starting the GUI you must load the realtime part:

```
emc2$ bin/halcmd loadrt scope_rt
emc2$
```

Once the realtime part is loaded, we can start the GUI. Like halmeter, you can follow it with & so it runs in the background and you get your shell prompt back immediately:

```
emc2$ bin/halscope &
[2] 3678
emc2$
```

The scope GUI window will open, immediately followed by a “Realtime function not linked” dialog that looks like figure 4.4<sup>5</sup>.

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 1.03mS thread “slow” (formerly “siggen.thread”, see footnote), and leave the multiplier at 1. We will also leave the record length at 4047 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 figure 4.5.

---

<sup>5</sup>Several of these screen captures refer to threads named “siggen.thread” and “stepgen.thread” instead of “slow” and “fast”. When the screenshots were captured, the “threads” component didn’t exist, and a different method was used to create threads, giving them different names. Also, the screenshots show pins, etc, as “stepgen.xxx” rather than “freqgen.xxx”. The original name of the freqgen module was stepgen, and I haven’t gotten around to re-doing all the screen shots since it was renamed. The name “stepgen” now refers to a different step pulse generator, one that accepts position instead of velocity commands. Both are described in detail later in this document.

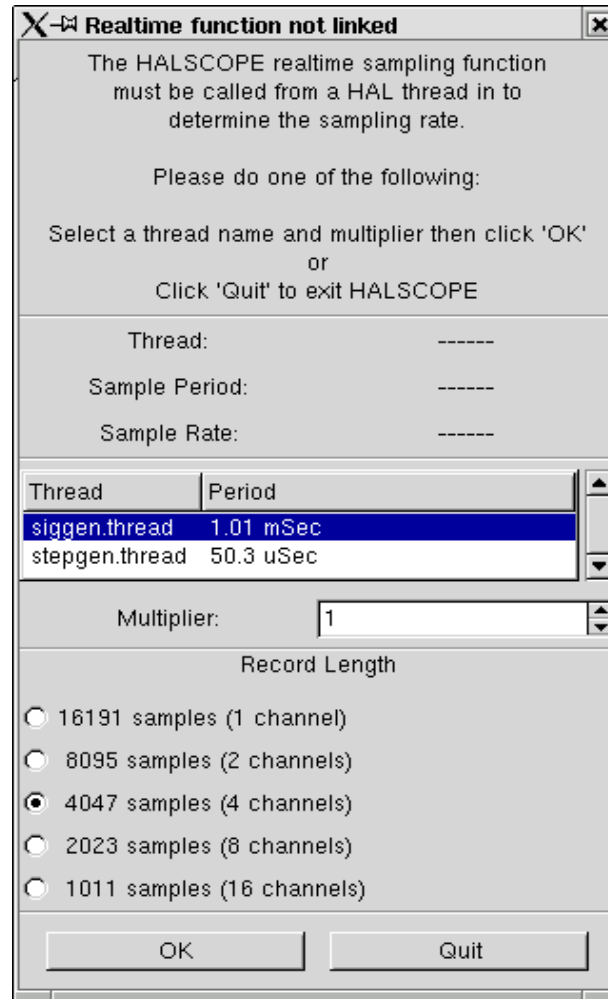


Figure 4.4: "Realtime function not linked" dialog

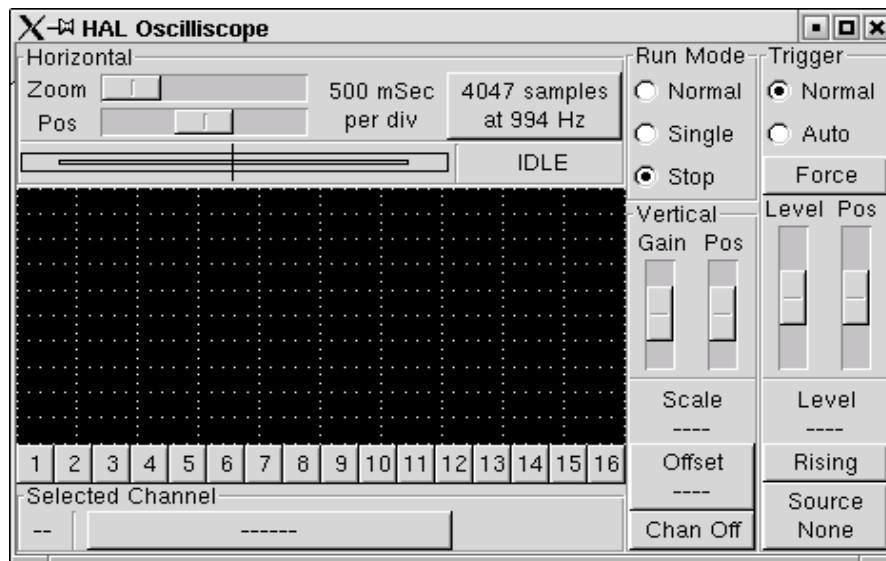


Figure 4.5: Initial scope window

### 4.5.2 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, figure 4.6. 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).

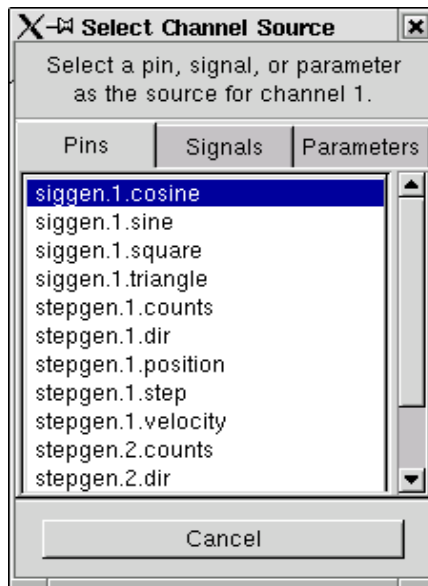


Figure 4.6: Select Channel Source dialog

To choose a signal, just click on it. In this case, we want to use channel 1 to display the signal “X\_vel”. When we click on “X\_vel”, the dialog closes and the channel is now selected. 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. 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”.

### 4.5.3 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 figure 4.7.

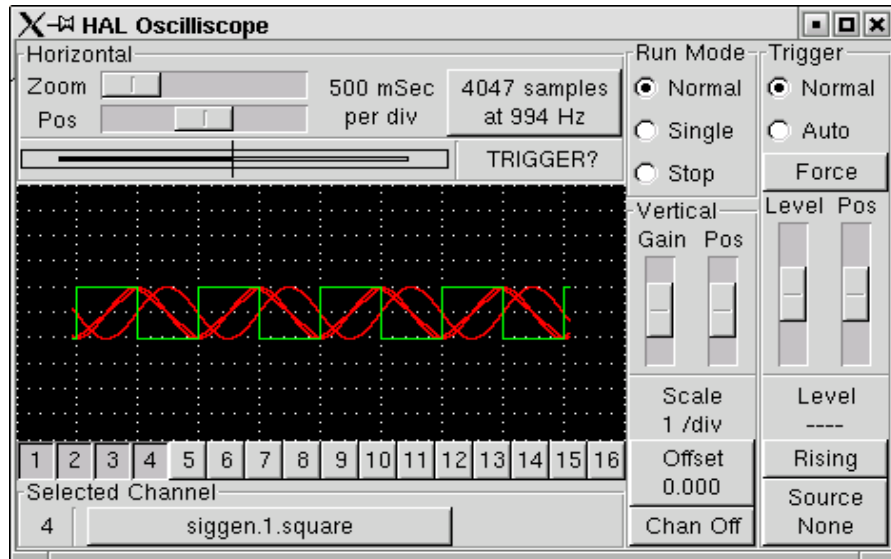


Figure 4.7: Captured Waveforms

The "Selected Channel" box at the bottom tells you that the green trace is the currently selected one, channel 4, which is displaying the value of the pin "siggen.1.square". Try clicking channel buttons 1 through 3 to highlight the other three traces.

#### 4.5.4 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 (see the halscope reference in section ?? for details).

#### 4.5.5 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 (Figure 4.8). Select a probe to use for triggering by clicking on it. For this example we will use channel 3, the triangle wave.

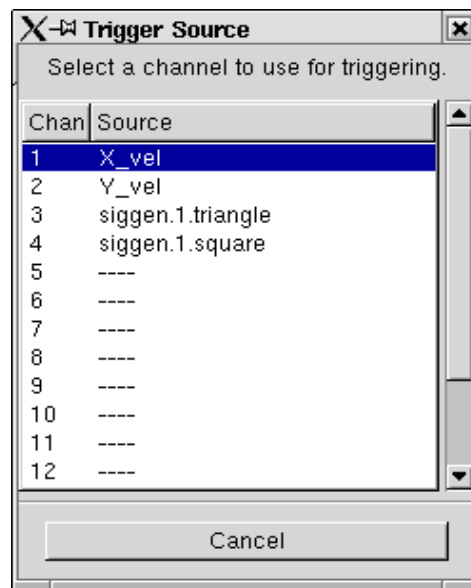


Figure 4.8: Trigger Source Dialog

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. Note that changing the trigger position stops the scope, once the position is adjusted you restart the scope by clicking the “Normal” button in the “Run Mode” box.

Now that we have adjusted the vertical controls and triggering, the scope display looks something like figure 4.9.

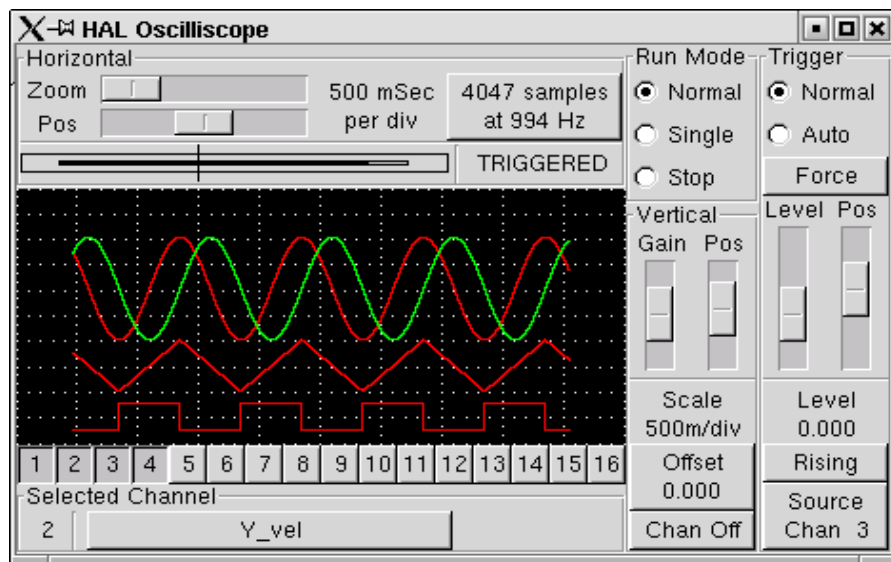


Figure 4.9: Waveforms with Triggering



### 4.5.6 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 50uS long, sampling at 1KHz isn't fast enough. To change the sample rate, click on the button that displays the record length and sample rate to bring up the "Select Sample Rate" dialog, figure . For this example, we will click on the 50uS 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.

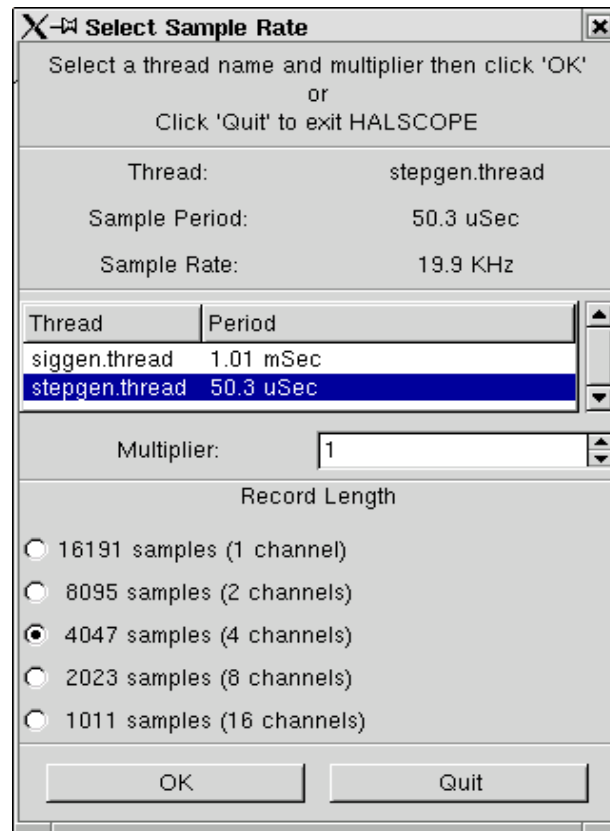


Figure 4.10: Sample Rate Dialog

### 4.5.7 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 "Off" button at the bottom of the "Vertical" box. Then click on channel 3, turn it 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.1.dir", then channel 6, and select "stepgen.1.step". Then click run mode "Normal" to start the scope, and adjust the horizontal zoom to 5mS 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 20m per division to better see the change in the velocity command. The result should look like figure 4.11.

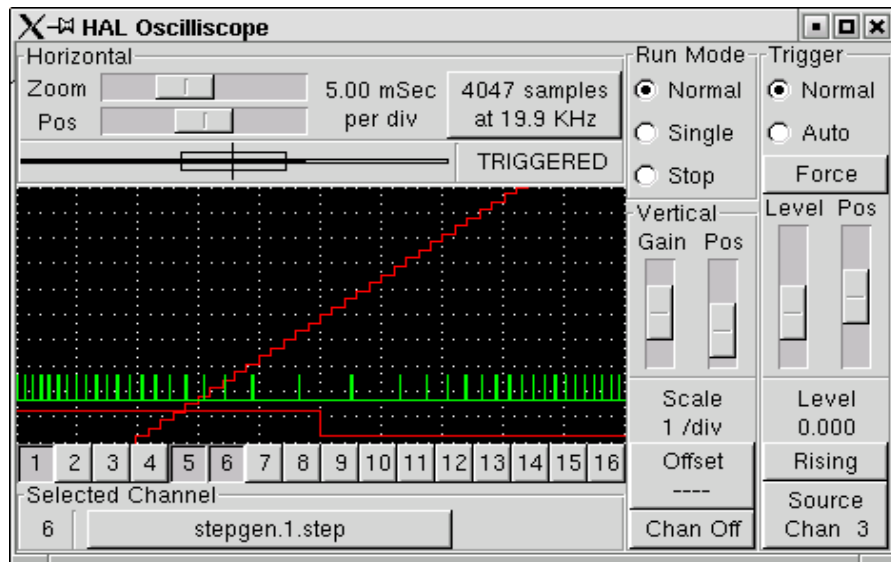


Figure 4.11: Looking at Step Pulses

# Chapter 5

## Hardware Drivers

### 5.1 Parport

Parport 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 output pins, and the status group is 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 output, a port provides a total of 12 outputs and 5 inputs. If configured as input, it provides 4 outputs and 13 inputs. No other combinations are supported, and a port cannot be changed from input to output once the driver is installed. Figure 5.1 shows two block diagrams, one showing the driver when the data group is configured for output, and one showing it configured for input.

There are actually two versions of the parport driver. One is a kernel module, and provides realtime control of the parallel port. The other is a user space process, and is not realtime. The non-realtime version is intended mainly for testing, and is not recommended for most applications. Using both the realtime and non-realtime versions at the same time is a bad idea.

The parport driver can control up to 8 ports (defined by MAX\_PORTS in hal\_parport.c). The ports are numbered starting at zero.

#### 5.1.1 Installing

Realtime version, from command line:

```
emc2$ bin/halcmd loadrt hal_parport 'cfg="<config-string>"'1
```

Realtime version, from a file:

```
loadrt hal_parport cfg="<config-string>"
```

Non-realtime version:

```
emc2# bin/hal_parport <config-string> &
```

---

<sup>1</sup>The single quotes around the entire `cfg=` argument are needed to prevent the shell from misinterpreting the double quotes around the string, and any spaces or special characters in the string. Single quotes should not be used in a file or from the halcmd prompt.

The config string consists of a hex port address, followed by an optional direction, repeated for each port. The direction is either “in” or “out” and determines the direction of the physical pins 2 through 9. If the direction is not specified, the data group defaults to output. For example:

```
emc2# bin/hal_parport 278 378 in 20A0 out
```

This example installs drivers for one port at 0x0278, with pins 2-9 as outputs (by default, since neither “in” nor “out” was specified), one at 0x0378, with pins 2-9 as inputs, and one at 0x20A0, with pins 2-9 explicitly specified as outputs. Note that you must know the base address of the parallel port to properly configure the driver. For ISA bus ports, this is usually not a problem, since the port is almost always at a “well known” address, like 0278 or 0378. However PCI ports may at nearly any address, and finding the address can be tricky<sup>2</sup>. There is no default address - if <config-string> does not contain at least one address, it is an error.

## 5.1.2 Removing

Realtime version:

```
emc2$ bin/halcmd unloadrt hal_parport
```

Non-realtime version:

Remove the non-realtime version by sending SIGINT or SIGTERM.

## 5.1.3 Pins

- (BIT) `parport.<portnum>.pin-<pinnum>-out` - Drives a physical output pin.
- (BIT) `parport.<portnum>.pin-<pinnum>-in` - Tracks a physical input pin.
- (BIT) `parport.<portnum>.pin-<pinnum>-in-not` - Tracks a physical input pin, but inverted.

For each pin, <portnum> is the port number, and <pinnum> 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`. Pins 1, 14, 16, and 17 are always outputs. Pins 2 through 9 are part of the data group and are output pins if the port is defined as an output port. (Output is the default.) These HAL pins control the state of the corresponding physical pins.

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`. Pins 10, 11, 12, 13, and 15 are always input pins. Pins 2 through 9 are input pins only if the port is defined as an input port. 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 - 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.

<sup>2</sup>Perhaps a future version of this driver will attempt to auto-identify PCI port addresses - however, it is very important that the user (or system integrator) makes sure the ports are configured correctly. Sending step and direction pulses to a LaserJet by accident simply wastes paper, but spooling a print job to stepper or servo motors could cause unexpected machine movement and possibly serious or fatal injuries.

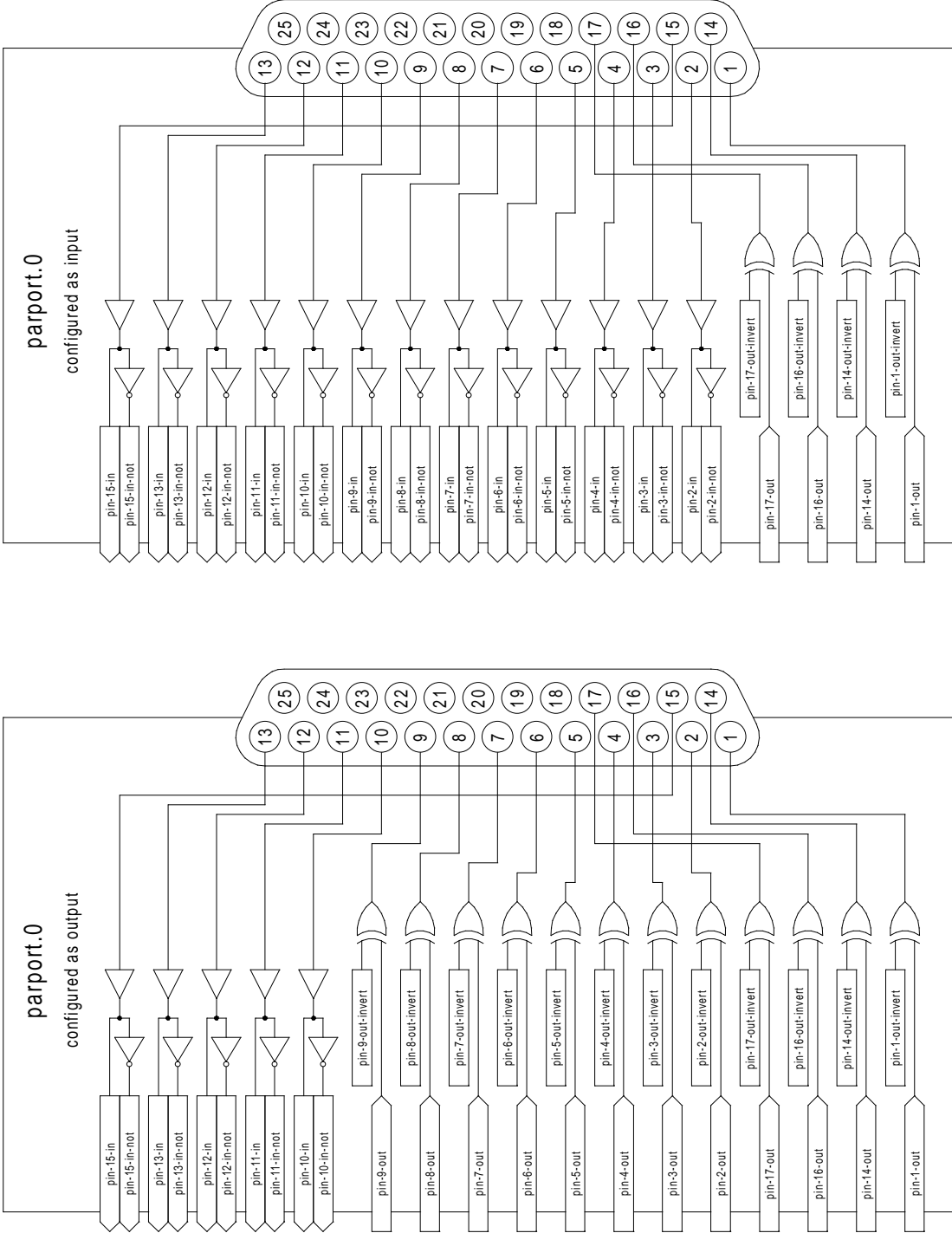


Figure 5.1: Parport Block Diagram

### 5.1.4 Parameters

- (BIT) `parport.<portnum>.pin-<pinnum>-out-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 high, and `FALSE` drives it low. If `-invert` is `TRUE`, then setting the HAL `-out` pin `TRUE` will drive the physical pin low.

### 5.1.5 Functions

- (FUNCT) `parport.<portnum>.read`– Reads physical input pins of port `<portnum>` and updates HAL `-in` and `-in-not` pins.
- (FUNCT) `parport.read-all` – Reads physical input pins of all ports and updates HAL `-in` and `-in-not` pins.
- (FUNCT) `parport.<portnum>.write` – Reads HAL `-out` pins of port `<portnum>` and updates that port's physical output pins.
- (FUNCT) `parport.write-all` – Reads HAL `-out` pins of all ports and updates all physical output pins.

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.

The user space version of the driver cannot export functions, instead it exports parameters with the same names. Then the driver sits in a loop checking the parameters. If they are zero, it does nothing. If any parameter is greater than zero, the corresponding function runs once, then the parameter is reset to zero. If any parameter is less than zero, the corresponding function runs on every pass through the loop. The driver will loop forever, until it receives either `SIGINT` (ctrl-C) or `SIGTERM`, at which point it cleans up and exits.

## 5.2 AX5214H

The Axiom Measurement & Control AX5214H is a 48 channel digital I/O board. It plugs into a PCI bus, and resembles a pair of 8255 chips.<sup>3</sup>

### 5.2.1 Installing

From command line:

```
emc2$ bin/halcmd loadrt hal_ax5214h 'cfg="<config-string>"'4
```

From a file:

```
loadrt hal_ax5214h cfg="<config-string>"
```

<sup>3</sup>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.

<sup>4</sup>The single quotes around the entire `cfg=` argument are needed to prevent the shell from misinterpreting the double quotes around the string, and any spaces or special characters in the string. Single quotes should not be used in a file or from the `halcmd` prompt.

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 IIIIOIIIOO 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).

## 5.2.2 Removing

```
emc2$ bin/halcmd unloadrt hal_ax5214
```

## 5.2.3 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.

## 5.2.4 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.

## 5.2.5 Functions

- (FUNCT) `ax5214.<boardnum>.read` – Reads all digital inputs on one board.
- (FUNCT) `ax5214.<boardnum>.write` – Writes all digital outputs on one board.

## 5.3 Servo-To-Go

The Servo-To-Go is one of the first PC motion control cards<sup>5</sup> supported by EMC. It is an ISA card and it exists in different flavours (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.

### 5.3.1 Installing:

```
emc2$ bin/halcmd loadrt hal_stg [base=<address>] [num_chan=<nr>] \
[dio="<dio-string>"] [model=<model>]
```

The base address field is optional, in case 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<sup>6</sup>. For example:

```
emc2$ bin/halcmd 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, DAC's and ADC's, 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

```
emc2$ bin/halcmd 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.

### 5.3.2 Removing

```
emc2$ bin/halcmd unloadrt hal_stg
```

### 5.3.3 Pins

- (s32) stg.<channel>.counts – Tracks the counted encoder ticks.
- (FLOAT) stg.<channel>.position – Outputs a converted position.
- (FLOAT) stg.<channel>.dac-value – Drives the voltage for the corresponding DAC.
- (FLOAT) stg.<channel>.adc-value – Tracks the measured voltage from the corresponding ADC.

<sup>5</sup>a motion control card usually is a board containing devices to control one or more axes (the control devices are usually DAC's to set an analog voltage, encoder counting chips for feedback, etc.)

<sup>6</sup>hint: after starting up the driver, 'dmesg' can be consulted for messages relevant to the driver (e.g. autodetected version number and base address)



- (BIT) `stg.in-<pinnum>` – Tracks a physical input pin.
- (BIT) `stg.in-<pinnum>-not` – Tracks a physical input pin, but inverted.
- (BIT) `stg.out-<pinnum>` – Drives a physical output pin

For each pin, `<channel>` is the axis number, and `<pinnum>` is the logic pin number of the STG<sup>7</sup>.

The `in-` 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.

### 5.3.4 Parameters

- (FLOAT) `stg.<channel>.position-scale` – The number of counts / user unit (to convert from counts to units).
- (FLOAT) `stg.<channel>.dac-offset` – Sets the offset for the corresponding DAC.
- (FLOAT) `stg.<channel>.dac-gain` – Sets the gain of the corresponding DAC.
- (FLOAT) `stg.<channel>.adc-offset` – Sets the offset of the corresponding ADC.
- (FLOAT) `stg.<channel>.adc-gain` – Sets the gain of the corresponding ADC.
- (BIT) `stg.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 high, and FALSE drives it low. If `-invert` is TRUE, then setting the HAL `out-` pin TRUE will drive the physical pin low.

### 5.3.5 Functions

- (FUNCT) `stg.capture-position` – Reads the encoder counters from the axis `<channel>`.
- (FUNCT) `stg.write-dacs` – Writes the voltages to the DACs.
- (FUNCT) `stg.read-adcs` – Reads the voltages from the ADCs.
- (FUNCT) `stg.di-read` – Reads physical `in-` pins of all ports and updates all HAL `in-` and `in-<pinnum>-not` pins.
- (FUNCT) `stg.do-write` – Reads all HAL `out-` pins and updates all physical output pins.

## 5.4 Mesa Electronics m5i20 “Anything I/O Card”

The Mesa Electronics m5i20 card consists of an FPGA that can be loaded with a wide variety of configurations, and has 72 pins that leave the PC. The assignment of the pins depends on the FPGA configuration. Currently there is a HAL driver for the “4 axis host based motion control” configuration, and this FPGA configurations is also provided with EMC2. It provides 8 encoder counters, 4 PWM outputs (normally used as DACs) and up to 48 digital I/O channels, 32 inputs and 16 outputs.<sup>8</sup>

Installing:

<sup>7</sup>if IIO 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`)

<sup>8</sup>Ideally the encoders, “DACs”, and digital I/O would comply with the canonical interfaces defined earlier, but they don’t. Fixing that is on the things-to-do list.

```
emc2$ bin/halcmd loadrt hal_m5i20 [loadFpga=1|0] [dacRate=<rate>]
```

If `loadFpga` is 1 (the default) the driver will load the FPGA configuration on startup. If it is 0, the driver assumes the configuration is already loaded. `dacRate` sets the carrier frequency for the PWM outputs, in Hz. The default is 32000, for 32KHz PWM.<sup>9</sup> The driver prints some usefull debugging message to the kernel log, which can be viewed with `dmesg`.

### 5.4.1 Removing

```
emc2$ bin/halcmd unloadrt hal_m5i20
```

### 5.4.2 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 uses the PCI board ID, so it may be non-zero even if there is only one board.

- (s32) `m5i20.<board>.enc-<channel>-count` - Encoder position, in counts.
- (s32) `m5i20.<board>.enc-<channel>-cnt-latch` - Position in counts when index pulse arrived.
- (FLOAT) `m5i20.<board>.enc-<channel>-position` - Encoder position, in user units.
- (FLOAT) `m5i20.<board>.enc-<channel>-pos-latch` - Position in user units when index pulse arrived.
- (BIT) `m5i20.<board>.enc-<channel>-index` - Current status of index pulse input?
- (BIT) `m5i20.<board>.enc-<channel>-idx-latch` - Goes true when an index pulse arrives?
- (BIT) `m5i20.<board>.enc-<channel>-latch-index` - Bidirectional - used to control/report index latching?
- (BIT) `m5i20.<board>.enc-<channel>-reset-count` - Bidirectional (why?) - used to reset counter?
- (BIT) `m5i20.<board>.dac-<channel>-enable` - Enables DAC if true. DAC outputs zero volts if false?
- (FLOAT) `m5i20.<board>.dac-<channel>-value` - Analog output value for PWM “DAC” (in user units, see `-scale` and `-offset`)
- (BIT) `m5i20.<board>.in-<channel>` - State of digital input pin, see canonical digital input.
- (BIT) `m5i20.<board>.in-<channel>-not` - Inverted state of digital input pin, see canonical digital input.
- (BIT) `m5i20.<board>.out-<channel>` - Value to be written to digital output, see canonical digital output.
- (BIT) `m5i20.<board>.estop-in` - Dedicated estop input, more details needed.
- (BIT) `m5i20.<board>.estop-in-not` - Inverted state of dedicated estop input.
- (BIT) `m5i20.<board>.watchdog-reset` - Bidirectional, - Set TRUE to reset watchdog once, is automatically cleared.

<sup>9</sup>I don't know what the maximum (and minimum, if any) PWM frequency is, it should be documented here. Also, this is the kind of thing that ideally is controlled by a HAL parameter, rather than being set when the driver is initially loaded. I don't know if that is possible, it depends on the hardware and I don't have the necessary information.

### 5.4.3 Parameters

- (FLOAT) `m5i20.<board>.enc-<channel>-scale` – The number of counts / user unit (to convert from counts to units).
- (FLOAT) `m5i20.<board>.dac-<channel>-offset` – Sets the DAC offset.
- (FLOAT) `m5i20.<board>.dac-<channel>-gain` – Sets the DAC gain (scaling).
- (BIT) `m5i20.<board>.dac-<channel>-interlaced` – Sets the DAC to interlaced mode. Use this mode if you are filtering the PWM to generate an analog voltage.<sup>10</sup>
- (BIT) `m5i20.<board>.out-<channel>-invert` – Inverts a digital output, see canonical digital output.
- (U16) `m5i20.<board>.watchdog-control` – Configures the watchdog. (0x0001 = watchdog enable, 0x0002 = watchdog auto-reset ?)
- (U16) `m5i20.<board>.watchdog-timeout` – Sets watchdog timeout period (in microseconds)
- (U16) `m5i20.<board>.led-view` – Maps some of the I/O to onboard LEDs. See table below.

### 5.4.4 Functions

- (FUNCT) `m5i20.<board>.encoder-read` – Reads all encoder counters.
- (FUNCT) `m5i20.<board>.digital-in-read` – Reads digital inputs.
- (FUNCT) `m5i20.<board>.dac-write` – Writes the voltages (PWM duty cycles) to the “DACs”.
- (FUNCT) `m5i20.<board>.digital-out-write` – Writes digital outputs.
- (FUNCT) `m5i20.<board>.misc-update` – Writes watchdog timer configuration to hardware. Resets watchdog timer. Updates E-stop pin (more info needed). Updates onboard LEDs.

### 5.4.5 Connector pinout

The Hostmot-4 FPGA configuration has the following pinout. There are three 50-pin ribbon cable connectors on the card: P2, P3, and P4. There are also 8 status LEDs.

<sup>10</sup>With normal 10 bit PWM, 50% duty cycle would be 512 cycles on and 512 cycles off = ca 30 kHz with 33 MHz reference counter. With fully interleaved PWM this would be 1 cycle on, 1 cycle off for 1024 cycles (16.66 MHz if the PWM reference counter runs at 33 MHz) = much easier to filter. The 5i20 configuration interlace is somewhat between non and fully interlaced (to make it easy to filter but not have as many transistions as fully interleaved).

**5.4.5.1 Connector P2**

m5i20 card connector P2	Function/HAL-pin
1	enc-01 A input
3	enc-01 B input
5	enc-00 A input
7	enc-00 B input
9	enc-01 index input
11	enc-00 index input
13	dac-01 output
15	dac-00 output
17	DIR output for dac-01
19	DIR output for dac-00
21	dac-01-enable output
23	dac-00-enable output
25	enc-03 B input
27	enc-03 A input
29	enc-02 B input
31	enc-02 A input
33	enc-03 index input
35	enc-02 index input
37	dac-03 output
39	dac-02 output
41	DIR output for dac-03
43	DIR output for dac-02
45	dac-03-enable output
47	dac-02-enable output
49	Power +5 V (or +3.3V ?)
all even pins	Ground

**5.4.5.2 Connector P3**

Encoder counters 4 - 7 work simultaneously with in-00 to in-11.

If you are using in-00 to in-11 as general purpose IO then reading enc-<4-7> will produce some random junk number.

m5i20 card connector P3	Function/HAL-pin	Secondary Function/HAL-pin
1	in-00	enc-04 A input
3	in-01	enc-04 B input
5	in-02	enc-04 index input
7	in-03	enc-05 A input
9	in-04	enc-05 B input
11	in-05	enc-05 index input
13	in-06	enc-06 A input
15	in-07	enc-06 B input
17	in-08	enc-06 index input
19	in-09	enc-07 A input
21	in-10	enc-07 B input
23	in-11	enc-07 index input
25	in-12	
27	in-13	
29	in-14	
31	in-15	
33	out-00	
35	out-01	
37	out-02	
39	out-03	
41	out-04	
43	out-05	
45	out-06	
47	out-07	
49	Power +5 V (or +3.3V ?)	
all even pins	Ground	

*Note!*: This is the intended pinout of P3. Unfortunately, in the current FPGA configuration distributed with EMC2<sup>11</sup>, the secondary encoders, enc-04, enc-05, enc-06, and enc-07 are wrongly configured. The input pins for enc-04 and enc-05 partly overlap, as do the pins for enc-06 and enc-07. Thus it is possible to use enc-04 and enc-06 simultaneously, but using enc-04 and enc-05 is not possible since counts on enc-04 will make the count on enc-05 jump by +/- 1. If you are using pins in-00 to in-11 as general purpose inputs you are not affected by this bug.

### 5.4.5.3 Connector P4

The index mask masks the index input of the encoder so that the encoder index can be combined with a mechanical switch or opto detector to clear or latch the encoder counter only when the mask input bit is in proper state (selected by mask polarity bit) and encoder index occurs. This is useful for homing. The behaviour of these pins is controlled by the Counter Control Register (CCR), however there is currently no function in the driver to change the CCR. See REGMAP4<sup>12</sup> for a description of the CCR.

<sup>11</sup>emc2/src/hal/drivers/m5i20\_HM5-4E.h dated 2005/06/07

<sup>12</sup>emc2/src/hal/drivers/m5i20/REGMAP4E

m5i20 card connector P4	Function/HAL-pin	Secondary Function/HAL-pin
1	in-16	enc-00 index mask
3	in-17	enc-01 index mask
5	in-18	enc-02 index mask
7	in-19	enc-03 index mask
9	in-20	
11	in-21	
13	in-22	
15	in-23	
17	in-24	enc-04 index mask
19	in-25	enc-05 index mask
21	in-26	enc-06 index mask
23	in-27	enc-07 index mask
25	in-28	
27	in-29	
29	in-30	
31	in-31	
33	out-08	
35	out-09	
37	out-10	
39	out-11	
41	out-12	
43	out-13	
45	out-14	
47	out-15	
49	Power +5 V (or +3.3V ?)	
all even pins	Ground	

#### 5.4.5.4 LEDs

The status LEDs will monitor one motion channel set by the `m5i20.<board>.led-view` parameter. A call to `m5i20.<board>.misc-update` is required to update the LEDs.

LED name	Output
LED0	IRQLatch ?
LED1	enc-<channel> A
LED2	enc-<channel> B
LED3	enc-<channel> index
LED4	dac-<channel> DIR
LED5	dac-<channel>
LED6	dac-<channel>-enable
LED7	watchdog timeout ?

## 5.5 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.<sup>13</sup>

Installing:

<sup>13</sup>Ideally the encoders, DACs, ADCs, and digital I/O would comply with the canonical interfaces defined earlier, but they don't. Fixing that is on the things-to-do list.

```
emc2$ bin/halcmd loadrt hal_motenc
```

During loading (or attempted loading) the driver prints some usefull debugging message to the kernel log, which can be viewed with `dmesg`.

### 5.5.1 Removing

```
emc2$ bin/halcmd unloadrt hal_motenc
```

### 5.5.2 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 baord, 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.

### 5.5.3 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.
- (U16) `motenc.<board>.watchdog-control` – Configures the watchdog?
- (U16) `motenc.<board>.led-view` – Maps some of the I/O to onboard LEDs?

### 5.5.4 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.

## 5.6 Pico Systems PPMC (Parallel Port Motion Control)

Pico Systems has a family of boards for doing servo, stepper, and pwm 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 card.

Installing:

```
emc2$ bin/halcmd 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 `<addr3>` 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 usefull debugging message to the kernel log, which can be viewed with `dmesg`.

### 5.6.1 Removing

```
emc2$ bin/halcmd unloadrt hal_ppmc
```



### 5.6.2 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) `ppmc.<port>.encoder.<channel>.count` – Encoder position, in counts.
- (s32) `ppmc.<port>.encoder.<channel>.delta` – Change in counts since last read.
- (FLOAT) `ppmc.<port>.encoder.<channel>.position` – Encoder position, in user units.
- (BIT) `ppmc.<port>.encoder.<channel>.index` – Something to do with index pulse.<sup>14</sup>
- (BIT) `ppmc.<port>.pwm.<channel>.enable` – Enables a PWM generator.
- (FLOAT) `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.
- (BIT) `ppmc.<port>.stepgen.<channel>.enable` – Enables a step pulse generator.
- (FLOAT) `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.
- (BIT) `ppmc.<port>.in-<channel>` – State of digital input pin, see canonical digital input.
- (BIT) `ppmc.<port>.in.<channel>.not` – Inverted state of digital input pin, see canonical digital input.
- (BIT) `ppmc.<port>.out-<channel>` – Value to be written to digital output, see canonical digital output.

### 5.6.3 Parameters

- (FLOAT) `ppmc.<port>.enc.<channel>.scale` – The number of counts / user unit (to convert from counts to units).
- (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 153Hz, maximum is 500KHz.
- (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`).
- (FLOAT) `ppmc.<port>.pwm.<channel>.max-dc` – Maximum duty cycle, from 0.0 to 1.0.
- (FLOAT) `ppmc.<port>.pwm.<channel>.min-dc` – Minimum duty cycle, from 0.0 to 1.0.
- (FLOAT) `ppmc.<port>.pwm.<channel>.duty-cycle` – Actual duty cycle (used mostly for troubleshooting.)
- (BIT) `ppmc.<port>.pwm.<channel>.bootstrap` – If true, the PWM generator will generate a short sequence of pulses of both polarities when it is enabled, to charge the bootstrap capacitors used on some MOSFET gate drivers.

<sup>14</sup>Index handling does `_not_` comply with the canonical encoder interface, and should be changed.

- (U8) `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 PWM generators, as indicated by `<channel-range>`.
- (U8) `ppmc.<port>.stepgen.<channel-range>.pulse-width` – Sets width of step pulses, in units of 100nS. Applies to a group of four consecutive PWM generators, as indicated by `<channel-range>`.
- (U8) `ppmc.<port>.stepgen.<channel-range>.pulse-space-min` – Sets minimum time between pulses, in units of 100nS. The maximum step rate is  $1/(100\text{nS} * (\text{pulse-width} + \text{pulse-space-min}))$ . Applies to a group of four consecutive PWM generators, as indicated by `<channel-range>`.
- (FLOAT) `ppmc.<port>.stepgen.<channel>.scale` – Scaling for step pulse generator. The step frequency in Hz is the absolute value of `velocity * scale`.
- (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.)
- (FLOAT) `ppmc.<port>.stepgen.<channel>.frequency` – Actual step pulse frequency in Hz (used mostly for troubleshooting.)
- (BIT) `ppmc.<port>.out.<channel>.invert` – Inverts a digital output, see canonical digital output.

#### 5.6.4 Functions

- (FUNCT) `ppmc.<port>.read` – Reads all inputs (digital inputs and encoder counters) on one port.
- (FUNCT) `ppmc.<port>.write` – Writes all outputs (digital outputs, stepgens, PWMs) on one port.

## Chapter 6

# Basic configurations for a stepper based system

### 6.1 Introduction

This chapter tries to describe some of the more common settings that users want to change when setting up EMC2. Because of the various possibilities of configuring EMC2, it is very hard to document them all, and keep this document relatively short.

The most common EMC2 usage (as reported by our users) is for stepper based systems. 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 the sample config released along with EMC2. The config is called stepper, and usually it is found in `/etc/emc2/sample-configs/stepper`

### 6.2 Pinout

One of the major flaws in EMC was that you couldn't specify the pinout without recompiling the source code. EMC2 is far more flexible, and now (thanks to the Hardware Abstraction Layer) you can easily specify which signal goes where. (read the [3.1](#) section for more information about the HAL).

As it is described in the HAL Introduction and tutorial, we have signals, pins and parameters inside the HAL.

The ones relevant for our pinout are<sup>1</sup>:

```
signals: Xstep, Xdir & Xen  
pins: parport.0.pin-XX-out & parport.0.pin-XX-in 2
```

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. Furtheron we'll investigate the `standard_pinout.hal`.

---

<sup>1</sup>Note: we are only presenting one axis to keep it short, all others are similar.

<sup>2</sup>Refer to section [5.1](#) for additional information

### 6.2.1 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
linksp Xstep parport.0.pin-03-out
linksp Xdir parport.0.pin-02-out
linksp Ystep parport.0.pin-05-out
linksp Ydir parport.0.pin-04-out
linksp Zstep parport.0.pin-07-out
linksp Zdir parport.0.pin-06-out
# create a signal for the estop loopback
linkpp iocontrol.0.user-enable-out \
      iocontrol.0.emc-enable-in
# create signals for tool loading loopback
linkpp iocontrol.0.tool-prepare \
      iocontrol.0.tool-prepared
linkpp iocontrol.0.tool-change \
      iocontrol.0.tool-changed
# create a signal for "spindle on"
newsig spindle_on bit
# connect it to the iocontroller
linksp spindle_on iocontrol.0.spindle-on
# connect it to a physical pin
linksp spindle_on parport.0.pin-09-out
```

The files starting with '#' are comments, and their only purpose is to guide the reader through the file.

### 6.2.2 Overview of the standard\_pinout.hal

There are a couple of operations that get executed when the standard\_pinout.hal gets executed / interpreted:

1. The Parport driver gets loaded (see [5.1](#) for details)
2. The read & write functions of the parport driver get assigned to the Base thread <sup>3</sup>
3. The step & direction signals for axes X,Y,Z get linked to pins on the parport
4. Further IO signals get connected (estop loopback, toolchanger loopback)
5. A spindle On signal gets defined and linked to a parport pin

---

<sup>3</sup>the fastest thread in the EMC2 setup, usually the code gets executed every few microseconds

### 6.2.3 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:

```
linksp Xstep parport.0.pin-03-out
linksp Xdir  parport.0.pin-02-out
```

can be changed to:

```
linksp Xstep parport.0.pin-02-out
linksp Xdir  parport.0.pin-03-out
```

or basically any other numbers you like.

Hint: make sure you don't have more than one signal connected to the same pin.

### 6.2.4 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:

```
linksp 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 ones will be disabled aswell, so having only one signal / pin is perfectly safe.

### 6.2.5 Adding an external ESTOP button

As you can see in 6.2.1 by default the stepper configuration assumes no external ESTOP button. <sup>4</sup>

To add a simple external button you need to replace the line:

```
linkpp iocontrol.0.user-enable-out \
      iocontrol.0.emc-enable-in
```

with

```
linkpp parport.0.pin-01-in iocontrol.0.emc-enable-in
```

This assumes an ESTOP switch connected to pin 01 on the parport. As long as the switch will stay pushed<sup>5</sup>, EMC2 will be in the ESTOP state. When the external button gets released EMC2 will immediately switch to the ESTOP-RESET state, and all you need to do is switch to Machine On and you'll be able to continue your work with EMC2.

<sup>4</sup>An extensive explanation of hooking up ESTOP circuitry is explained in the wiki.linuxcnc.org and in the Integrator Manual

<sup>5</sup>make sure you use a maintained switch for ESTOP.

# Chapter 7

## INI Configuration

### 7.1 Files Used for Configuration

The EMC is configured with human readable text files. All of these files can be read and edited in any of the common text file editors available with most any Linux distribution.<sup>1</sup> You'll need to be a bit careful when you edit these files. Some mistakes will cause the startup to fail. These files are read whenever the software starts up. Some of them are read repeatedly while the CNC is running.

Configuration files include;

**INI** The ini file overrides defaults that are compiled into the EMC code. It also provides sections that are read directly by the Hardware Abstraction Layer.

**HAL** The hal files start up process modules and provide linkages between EMC signals and specific hardware pins.

**VAR** The var file provide a set of numbered variables for use by the interpreter. These values are saved from one run to another.

**TBL** The tbl file saves tool information.

**NML** The nml file configures the communication channels used by the EMC. It is normally setup to run all of the communication within a single computer but can be modified to communicate between several computers.

**.emcsh** This file saves user specific information and is created to save the name of the directory when the user first selects an EMC configuration.

This chapter describes the EMC2's INI file in just enough detail so that the reader can understand which variable values might need to be edited in order to make a stock configuration conform to a real machine.<sup>2</sup>

### 7.2 The INI File Layout

A typical INI file follows a rather simple layout that includes;

- comments.

---

<sup>1</sup>Don't confuse a text editor with a word processor. A text editor like gedit or kwrite produce files that are plain text. They also produce lines of text that are separated from each other. A word processor like Open Office produce files with paragraphs and word wrapping and lots of embedded codes that control font size and such. A text editor does none of this.

<sup>2</sup>Complete reference to these files are left to the Integrator and Developer Handbooks.

- sections,
- variables.

Each of these elements is separated on single lines. Each end of line or newline character creates a new element.

### 7.2.1 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 some INI element will do.

```
; This is my little mill configuration file.
; I set it up on January 12, 2006
```

Comments can also be used to select between several values of a single variable.

```
# DISPLAY = tkemc
DISPLAY = axis
# DISPLAY = mini
# DISPLAY = keystick
```

In this list, the DISPLAY variable will be set to axis because all of the others are commented out. If someone carelessly edits a list like this and leaves two of the lines uncommented, the first one encountered will be used.

### 7.2.2 Sections

Sections in an INI file work like file folders in a drawer. They separate variables based on what part of the EMC they refer to. A section line looks like [THIS\_SECTION]. The name of the section is enclosed in brackets. Common INI files have several sections including;

- [EMC] general information
- [DISPLAY] selects and sets up some display characteristics
- [TASK] sets up the task planner
- [RS274NGC] location of interpreter specific file
- [EMCMOT] motion module and default characteristics
- [HAL] hardware configuration files and commands
- [TRAJ] information for the motion planner
- [AXIS\_0] ... [AXIS\_n] individual axis variables
- [EMCIO] emc's input and output variables.
- [EMCSERVER] local v remote system setup

Each of these section names are on a line by themselves so you can quickly scan through the file.

### 7.2.3 Variable

A variable line is made up of a variable name, an equals sign(=), and a value. Everything from the first non-whitespace 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. These variables and the values they are assigned are the way that the INI affects the operation of the EMC. You can edit the values for each keyword in any text editor. Changes don't take effect until the next time the controller is run.

The following sections detail each section of the configuration file, using sample values for the configuration lines.

## 7.3 INI Variable Definitions

We'll list these variables under section names in much the same way a real file contains them. Remember that this list describes those that you are most likely to change when you set up a newbie control.

### 7.3.1 [EMC] Section

**VERSION = \$Revision: 1.2 \$**

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.

**RS274NGC\_STARTUP\_CODE = G21 G90**

A string of NC codes that the interpreter is initialised with. These are the codes that an interpreter will be reset to.

### 7.3.2 [DISPLAY] Section

**DISLAY = tkemc**

The name of the user interface to use. Valid options are :

- axis
- keystick
- mini
- tkemc
- xemc



### 7.3.3 [EMCMOT] Section

#### **BASE\_PERIOD = 50000**

Base task period, in nanosecs - this is the fastest thread in the machine. It's units are nanoseconds. This is a fairly conservative value but if you are installing on a very old, slow processor you may have to make this a larger number or the machine may lock up or reboot.

You might want to make this value smaller if you have a fast computer because this value sets the maximum number of stepper pulses you can get from your machine. It has little affect on servo systems so leave it large.

#### **SERVO\_PERIOD = 1000000**

Servo task period is also in nanoseconds. This value will be rounded to an interger multiple of BASE\_PERIOD.

Most systems will not need to change this value. It is the update rate of the low level motion planner. You'll need it even if you only have steppers.

#### **TRAJ\_PERIOD = 10000000**

Trajectory Planner task period in nanoseconds This value will be rounded to an integer multiple of SERVO\_PERIOD.

Folk with fast computers have found that reducing this value by half will give them smother motion blending during contour cutting .

### 7.3.4 [HAL]

The [HAL] section lists files and commands to setup the Hardware Abstraction Layer. If this is a stepper system you might see several files here. The exact set would depend upon the configuration of signals at the parallel port. At a minimum you would see the following. Standard\_pinout matches the traditional EMC way of things while xyloTEX\_pinout matches the setup for those brand of cards.

```
HALFILE = core_stepper.hal
```

```
HALFILE = standard_pinout.hal
```

```
# HALFILE = xyloTEX_pinout.hal
```

You can also add variables (commands really) here.<sup>3</sup>

### 7.3.5 [TRAJ] Section.

The [TRAJ] section contains general parameters for the trajectory planning module in EMCMOT. You will not need to change these if you are applying EMC to a common three axis mill in the United States of America. If you are in an area using metric hardware components you might be working with the stepper\_mm.ini where these numbers are already setup for that system of units.

#### **AXES = 3**

The number of controlled axes in the system. If you have a four axis system put that number here and edit the next two variables as well

#### **COORDINATES = X Y Z**

The names of the axes being controlled. X, Y, Z, A, B, and C are all valid. It is also possible to have X Y Y Z and control ganged slides. For a fourth axis mounted on X you would use X Y Z A

#### **HOME = 0 0 0**

---

<sup>3</sup>See the Integrator Handbook for details.

Coordinates of the homed position of each axis. Again for a fourth axis you will need 0 0 0 0.

**LINEAR\_UNITS = 0.03937007874016**

The number of linear units per millimeter. For systems executing in native English (inch) units, this value is as shown above. For systems executing in native millimeter units, this value is 1. This does not affect the ability to program in English or metric units in NC code. It is used to determine how to interpret the numbers reported in the controller status by external programs.

**ANGULAR\_UNITS = 1.0**

The number of angular units per degree. For systems executing in native degree units, this value is as shown above. For systems executing in radians, this value is 0.01745329252167, or  $\pi/180$ .

**DEFAULT\_VELOCITY = 0.0167**

The initial velocity used for axis or coordinated axis motion, in user units per second. The value shown is one inch per minute.

**DEFAULT\_ACCELERATION = 2.0**

The initial acceleration used for axis or coordinated axis motion, in user units per second per second.

**MAX\_VELOCITY = 5.0**

The maximum velocity for any axis or coordinated move, in user units per second. Think for a moment what this value really means in hardware terms. The formula is  $\text{MAX\_VELOCITY} * 60$ . In this case this is 300 inches per minute.

**MAX\_ACCELERATION = 20.0**

The maximum acceleration for any axis or coordinated axis move, in user units per second per second.

### 7.3.6 [AXIS\_#] 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.

**TYPE = LINEAR**

The type of axes, either LINEAR or ANGULAR. Values for the position of LINEAR axes are in the units (per millimeter) specified in the [AXIS\_#] UNITS entry. Values for the position of ANGULAR axes are in the units (per degree) specified in the same entry.

**UNITS = 0.03937007874016**

Units per millimeter for a LINEAR axis, as defined in the [AXIS\_#] TYPE section, or units per degree for an ANGULAR axis as defined in the same section. The following parameters P, I, D, FF0, FF1, FF2 are used by the servo compensation algorithm to optimize performance while tracking trajectory set-points. See Tuning Servos for information on setting up a servomotor system.

**MAX\_VELOCITY = 1.2**

Per axis maximum velocity while coordinated motion is in effect.

**STEPGEN\_MAXVEL = 1.4<sup>4</sup>**

A value applied to the stepper pulse generator to provide some overhead for following error catch up.

---

<sup>4</sup>NOTE: the step generator module applies its own limits to acceleration and velocity. We have discovered that it needs to have a little "headroom" over the accel by the trajectory planner, otherwise it can fall slightly behind during accel and later overshoot as it catches up. In the long term we hope to come up with a clean fix for this problem. In the meantime, please set STEPGEN\_MAXVEL to a few percent higher than MAX\_VELOCITY, the regular velocity limit and STEPGEN\_MAXACCEL slightly larger than that of MAX\_ACCELERATION.

**MAX\_ACCELERATION = 20.0**

Per axis maximum acceleration while coordinated motion is in effect.

**STEPGEN\_MAXACCEL = 21.0**

Overhead for the stepper pulse generator when it needs to catch up.

**BACKLASH = 0.000**

Backlash compensation value can be used to make up for small deficiencies in the hardware used to drive an axis. Don't expect this to compensate for poor mechanical elements. The value set here is in UNITS.

**INPUT\_SCALE = 40000 0**

This variable has slightly different meaning for stepper and for servo systems. For steppers, the first number it is the number of pulses required to move the axis one UNIT.

For servos these two values are the scale and offset factors for the axis input from the raw feedback device, e.g., an incremental encoder. The second value (offset) is subtracted from raw input (e.g., encoder counts), and divided by the first value (scale factor), before being used as feedback. The units on the scale value are in raw units (e.g., counts) per user units (e.g., inch). The units on the offset value are in raw units (e.g., counts).

Specifically, when reading inputs, the EMC first reads the raw sensor values. The units on these values are the sensor units, typically A/D counts, or encoder ticks. These units, and the location of their 0 value, will not in general correspond to the quasi-SI units used in the EMC. Hence a scaling is done immediately upon sampling:

$$\text{input} = (\text{raw} - \text{offset}) / \text{scale}$$

The value for scale can be obtained by doing a unit analysis, i.e., units are [sensor units]/[desired input SI units]. For example, on a 2000 counts per rev encoder, and 10 revs/inch gearing, and desired units of mm, we have

[scale units] = 2000 [counts/rev] \* 10 [rev/inch] \* 1/25.4 [inch/mm] = 787.4 counts/mm and, as a result,

$$\text{input [mm]} = (\text{encoder [counts]} - \text{offset [counts]}) / 787.4 [\text{counts/mm}]$$

Note that the units of the offset are in sensor units, e.g., counts, and they are pre-subtracted from the sensor readings. The value for this offset is obtained by finding the value of counts for which you want your user units to read 0.0. This is normally accomplished automatically during a homing procedure.

**OUTPUT\_SCALE = 1.000 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 EMC first converts the desired output in quasi-SI units to raw actuator values, e.g., volts for an amplifier DAC. This scaling looks like:

$$\text{raw} = (\text{output} - \text{offset}) / \text{scale}$$

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, we have:

$$[\text{scale units}] = 250 [\text{mm/sec}] / 1 [\text{volts}] = 250 \text{ mm/sec/volt}$$

and, as a result,

$$\text{amplifier [volts]} = (\text{output [mm/sec]} - \text{offset [mm/sec]}) / 250 [\text{mm/sec/volt}]$$

Note that the units of the offset are in user 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 DACs 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:

a. Build a calibration table for the output, driving the DACs with a desired voltage and measuring the result. See table 7.1 for an example of voltage measurements.

Table 7.1: Output Voltage Measurements

Raw	Measured
-10	-9.93
-9	-8.83
0	-0.03
1	0.96
9	9.87
10	10.87

b. Do a least-squares linear fit to get coefficients a, b such that

$$\text{meas} = a * \text{raw} + b$$

c. Note that we want raw output such that our measured result is identical to the commanded output. This means

$$\text{cmd} = a * \text{raw} + b$$

$$\text{raw} = (\text{cmd} - b) / a$$

As a result, the a and b coefficients from the linear fit can be used as the scale and offset for the controller directly.

#### **MIN\_LIMIT = -1000**

The minimum limit (soft limit) for axis motion, in user units. When this limit is exceeded, the controller aborts axis motion.

#### **MAX\_LIMIT = 1000**

The maximum limit (soft limit) for axis motion, in user units. When this limit is exceeded, the controller aborts axis motion.

#### **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.

#### **FERROR = 1.0**

FERROR is the maximum allowable following error, in user 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. The following polarity values determine how inputs are interpreted and how outputs are applied. They can usually be set via trial-and-error since there are only two possibilities. The EMCMOT utility program USRMOT can be used to set these interactively and verify their results so that the proper values can be put in the INI file with a minimum of trouble.

### 7.3.6.1 Homing related params

The next few parameters are Homing related, for a better explanation read Section 7.4

#### **MIN\_FERROR = 0.010**

This is the value 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.

#### **HOME\_OFFSET = 0.0**

A distance to move once the home position has been found. The axis position is set to zero or the [TRAJ] HOME value for that axis after the HOME\_OFFSET distance has been moved.

#### **HOME\_SEARCH\_VEL = 0.0**

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 alone.

#### **HOME\_LATCH\_VEL = 0.0**

This is the final velocity to be used during a home sequence.

#### **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.

#### **HOME\_IGNORE\_LIMITS = NO**

Some machines use a limit switch as a home switch. This variable should be set to yes if you machine does this.

#### **P = 50**

Only used for servo systems and hardware that emulates servo systems. The univstep board from PICO is an example of such a system. The proportional gain for the axis servo. This value multiplies the error between commanded and actual position in user units, resulting in a contribution to the computed voltage for the motor amplifier. The units on the P gain are volts per user unit.

#### **I = 0**

The integral gain for the axis servo. The value multiplies the cumulative error between commanded and actual position in user units, resulting in a contribution to the computed voltage for the motor amplifier. The units on the I gain are volts per user unit-seconds.

#### **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 user unit per second.

#### **FF0 = 0**

The 0-th order feedforward 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 user unit.

#### **FF1 = 0**

The 1st order feedforward 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 user unit per second.

## 7.4 Homing

### 7.4.1 Overview

Homing seems simple enough - just move each joint to a known location, and set EMC's internal variables accordingly. However, different machines have different requirements, and homing is actually quite complicated.

### 7.4.2 Homing Sequence

Figure 7.1 shows four possible homing sequences, along with the associated configuration parameters. For a more detailed description of what each configuration parameter does, see the following section.

### 7.4.3 Configuration

There are six pieces of information that determine exactly how the home sequence behaves. They are defined in the ini, and can be tweaked to obtain the result you are after.

#### 7.4.3.1 HOME\_SEARCH\_VEL

'HOME\_SEARCH\_VEL' is defined in each AXIS\_\* section. The default value is zero. A value of zero causes EMC to assume that there is no home switch. The search and latch stages of homing are skipped, EMC declares the current position to be "HOME\_OFFSET", and does a rapid to "HOME" if "HOME" is not equal to "HOME\_OFFSET".

If 'HOME\_SEARCH\_VEL' is non-zero, then EMC assumes that there is a home switch. It begins searching 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.

#### 7.4.3.2 HOME\_LATCH\_VEL

'HOME\_LATCH\_VEL' is also defined in the ini file, for each AXIS. It specifies the speed and direction that EMC uses when it makes its final accurate determination of the home switch and index pulse location. It will usually be slower than the search velocity to maximise 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, EMC 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 EMC will latch the first pulse after it moves off the switch. If 'HOME\_SEARCH\_VEL' is zero, the latch phase is skipped and this parameter is ignored. 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.

#### 7.4.3.3 HOME\_IGNORE\_LIMITS

'HOME\_IGNORE\_LIMITS' is another settable option in the AXIS\_\* section. It's a boolean flag, and can hold the values YES / NO. This flag determines whether EMC will ignore the limit switch inputs. Some machine configurations do not use a separate home switch, instead they route one of the limit switch signals to the home switch input as well. In this case, EMC needs to ignore that limit during homing. The default value for this parameter is NO.

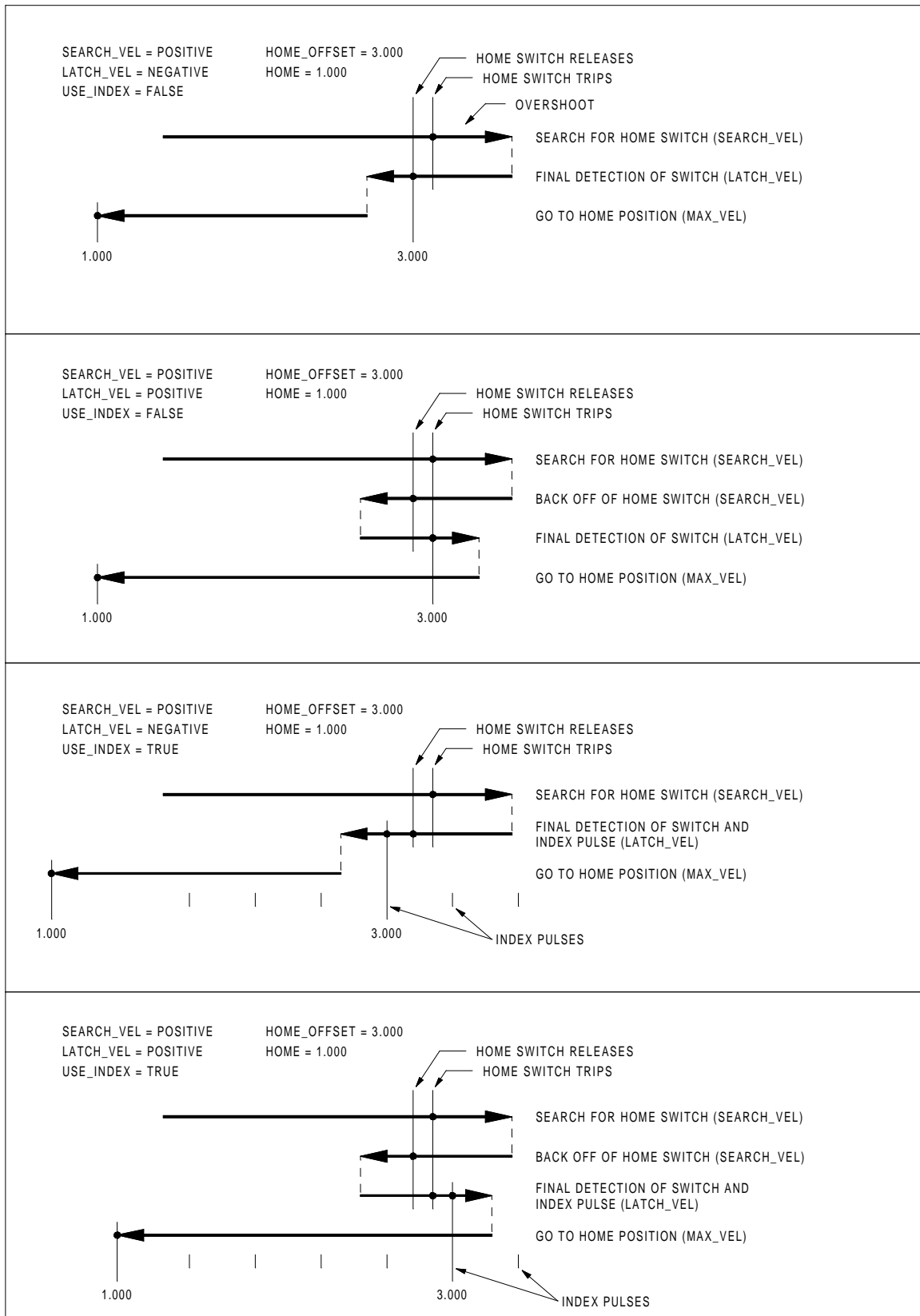


Figure 7.1: Homing Sequences

#### 7.4.3.4 HOME\_USE\_INDEX

'HOME\_USE\_INDEX' is a single bit settable in the AXIS\_\* section. It specifies whether or not there is an index pulse. If the flag is true (HOME\_USE\_INDEX = YES), EMC will latch on the rising edge of the index pulse. If false, EMC will latch on either the rising or falling edge of the home switch (depending on the signs of search\_vel and latch\_vel). If 'HOME\_SEARCH\_VEL' is zero, the latch phase is skipped and this parameter is ignored. The default value is NO.

#### 7.4.3.5 HOME\_OFFSET

'HOME\_OFFSET' is a value settable from the ini in the AXIS\_\* section. It 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, EMC sets the joint coordinate of the current point to "HOME\_OFFSET". The default value is zero.

#### 7.4.3.6 HOME

'HOME' is a value settable from the ini in the AXIS\_\* section. It is the position that the joint will go to upon completion of the homing sequence. After detecting the index pulse, and setting the coordinate of that point to "HOME\_OFFSET", EMC 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 axis 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. Since the axis is now homed, there should be no risk of crashing the machine, and a rapid move is the quickest way to finish the homing sequence. <sup>5</sup>

---

<sup>5</sup>The distinction between 'home' and 'home\_offset' is not as clear as I would like. I intend to make a small drawing and example to help clarify it.



**Part III**

**Using EMC2**

# Chapter 8

## Using the AXIS Graphical Interface

### 8.1 Introduction

AXIS is a graphical front-end for emc2 which features a live preview and backplot. It is written in Python and uses Tk and OpenGL to display its user interface.

### 8.2 Getting Started

To select AXIS as the front-end for emc2, edit the .ini file. In the section [DISPLAY] change the DISPLAY line to read

```
DISPLAY = axis
```

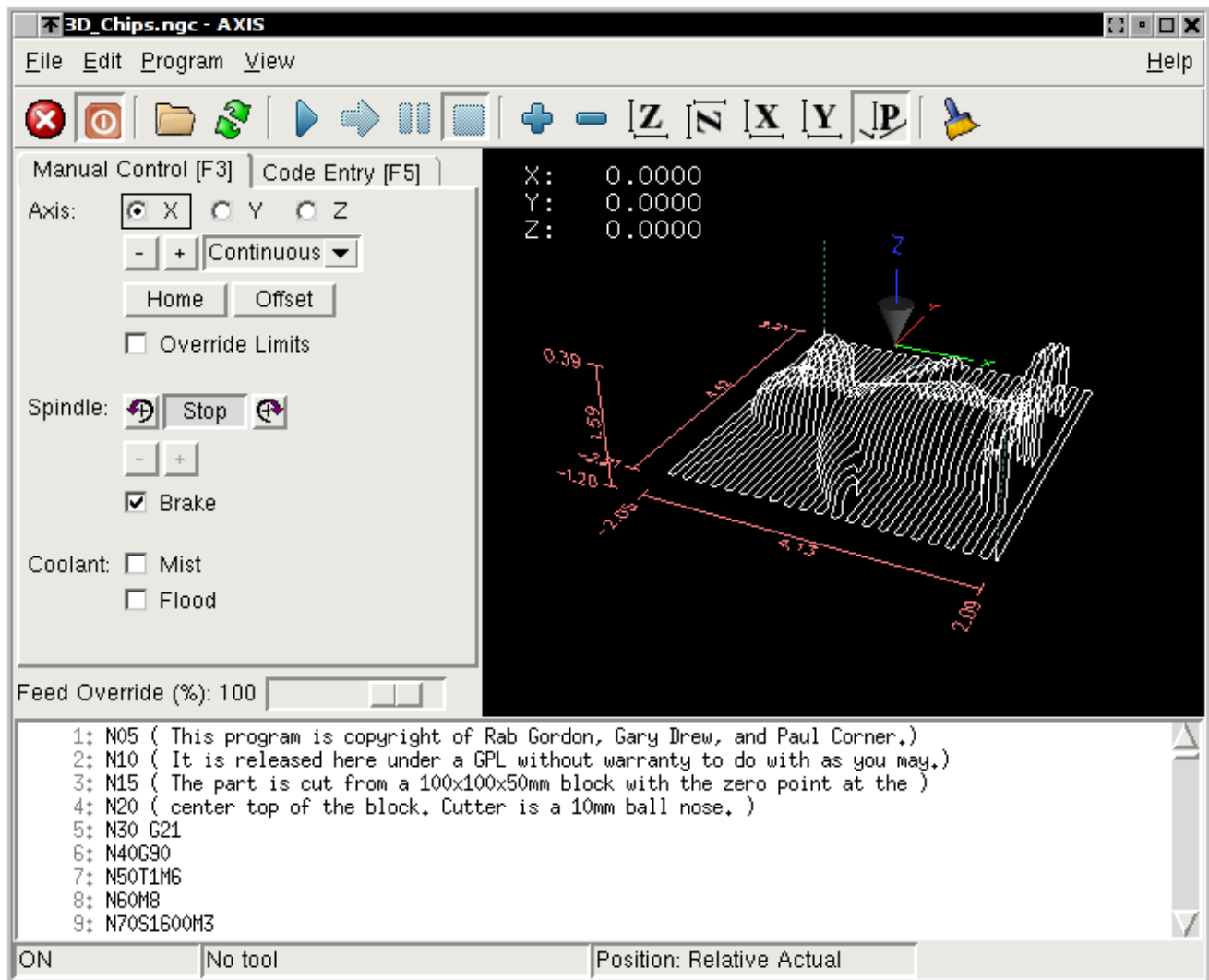
Then, start emc2 and select that ini file. The sample configuration `sim/axis.ini` is already configured to use AXIS as its front-end.

When you start AXIS, a window like the one in [Figure 8.1](#) is shown.

#### 8.2.1 A typical session with AXIS

1. Start emc and select a configuration file.
2. Clear the “ESTOP” condition and turn the machine on.
3. “Home” each axis.
4. Load the file to be milled.
5. Use the preview plot to verify that the program is correct.
6. Put the stock to be milled on the table.
7. Set the proper offsets for each axis by jogging and using the “Offset” button.
8. Run the program.
9. To mill the same file again, return to step 6. To mill a different file, return to step 4. When you're done, exit AXIS.

Figure 8.1: AXIS Window



### 8.3 Elements of the AXIS window

The AXIS window contains the following elements:

- A display area that shows a preview of the loaded file (in this case, "3D\_Chips"), 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 menubar and toolbar that allow you to perform various actions
- "Manual Control", which allows you to make the machine move, turn the spindle on or off, and turn the coolant on or off
- "Code Entry" (also called MDI), where G-code programs can be entered manually, one line at a time.
- "Feed Override", which allows you to increase or decrease the speed at which EMC executes the selected program.

- A text display area that shows the G-code source of the loaded file.
- A status bar which shows the state of the machine. In this screenshot, the machine is turned on, does not have a tool inserted, and the displayed position is “Relative” to the machine offset (as opposed to “Absolute”), and the “Actual” (as opposed to “Commanded” position)

### 8.3.1 Toolbar buttons

From left to right, the toolbar buttons are:

1. Toggle “Emergency Stop” (also called E-Stop)
2. Toggle machine power
3. Open a file
4. Reload the opened file
5. Run the program
6. Run the next line of the program
7. Pause the program
8. Stop the program
9. Zoom In
10. Zoom Out
11. Preset view “Z”
12. Preset view “Rotated Z”
13. Preset view “X”
14. Preset view “Y”
15. Preset view “P”
16. Clear backplot

### 8.3.2 Graphical Program Display Area

#### 8.3.2.1 Coordinate Display

In the upper-left corner of the program display is the coordinate display. It shows the position of the machine. To the left of the axis name, an origin symbol (⊕) is shown if the axis has been properly homed.

To properly interpret these numbers, refer to the “Position:” indicator in the status bar. If the position is “Absolute”, then the displayed number is in the machine coordinate system. If it is “Relative”, then the displayed number is in the offset coordinate system. When the coordinates displayed are relative, the display will include a cyan “machine origin” marker (⊕). If the position is “Commanded”, then it is the ideal position—for instance, the exact coordinate given in a G0 command. If it is “Actual”, then it is the position the machine has actually been moved to. 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).

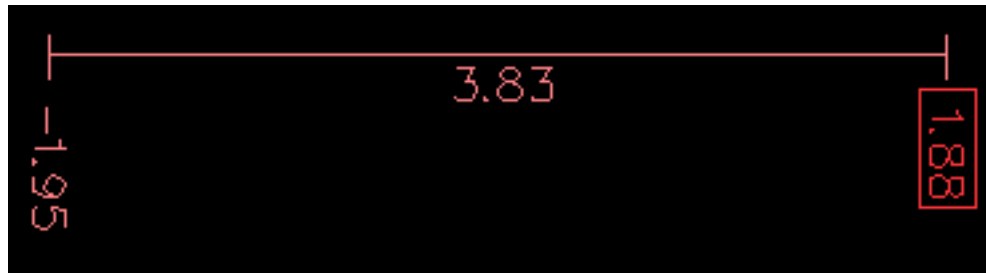
### 8.3.2.2 Preview Plot

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 dotted green 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 “X” marks.

### 8.3.2.3 Program Extents

The “extents” of the program in each axis are shown. At each end, the least or greatest coordinate value is indicated. In the middle, the difference between the coordinates is shown. In Figure 8.1, the X extent of the file is from -2.05 to 2.09 inches, a total of 4.13 inches.

When some coordinates exceed the “soft limits” in the .ini file, the relevant dimension is shown in a different color. Here, the maximum limit is exceeded on the X axis:



### 8.3.2.4 Tool Cone

The location of the tip of the tool is indicated by the “tool cone”. The cone does not indicate anything about the shape, length, or radius of the tool.

### 8.3.2.5 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.

### 8.3.2.6 Interacting with the display

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.

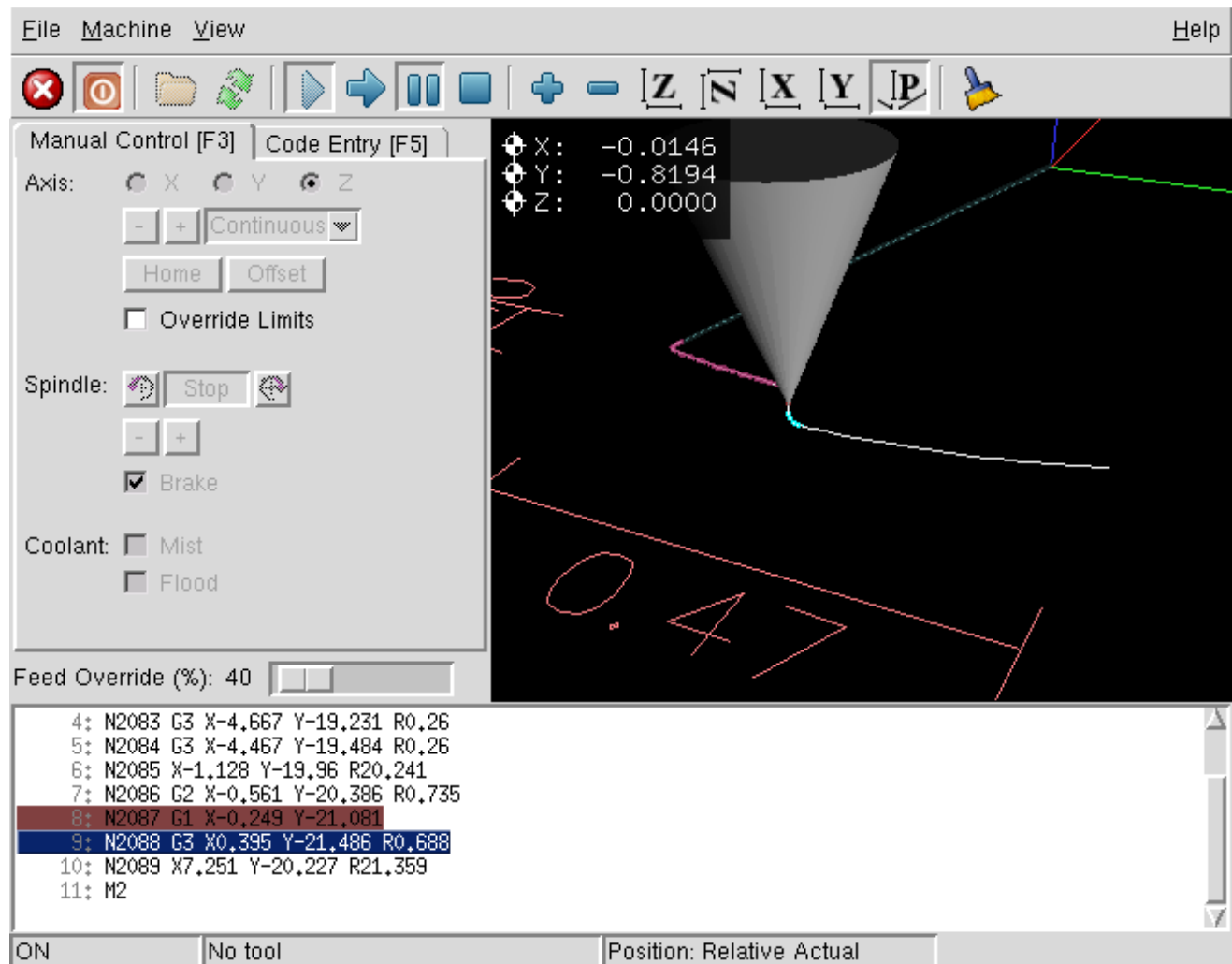
By dragging with the left mouse button pressed, the preview plot will be shifted (panned).

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 file as a whole.

By rotating the mouse wheel, or by dragging with the right 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.

Figure 8.2: Current and Selected Lines



### 8.3.3 Text Program 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.

### 8.3.4 Manual Control

Not all the items shown in this tab are useful on all machines. For instance, many machines have a spindle that can only turn in one direction. However, AXIS will still show both the “Rotate spindle clockwise” and “Rotate spindle counterclockwise” buttons. When the machine is not turned on, or when a program is running, the manual controls are unavailable.

### 8.3.4.1 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.

If “Continuous” is selected, the motion will continue as long as the button is pressed. If another value is selected, the machine will move exactly the displayed distance each time the button is clicked. By default, the available values are:

```
0.1000 0.0100 0.0010 0.0001
```

The .ini file setting `[DISPLAY] INCREMENTS` can be used to override the default. Its value can contain decimal numbers (e.g., 0.1000) or fractional numbers (e.g., 1/16). For machines configured as metric, a good setting might be

```
INCREMENTS = 1 0.1 0.01 0.001
```

For a machine configured as imperial (inches), a good setting might be

```
INCREMENTS = 1/8 .1 1/16 1/32 .01 0.001 0.0001
```

By pressing “Home”, the selected axis will be homed. Depending on your configuration, this may just set the current 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”.

By pressing “Offset”, the “G54 offset” for the current axis is changed so that the current axis value will be the relative position 0.0.

By pressing “Override Limits”, the machine will temporarily be permitted to jog outside the limits defined in the .ini file.

### 8.3.4.2 The “Spindle” group

The buttons on the first row select the direction for the spindle to rotate: Counterclockwise, Stopped, Clockwise. 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

### 8.3.4.3 The “Coolant” group

The two buttons allow the “Mist” and “Flood” coolants to be turned on and off.

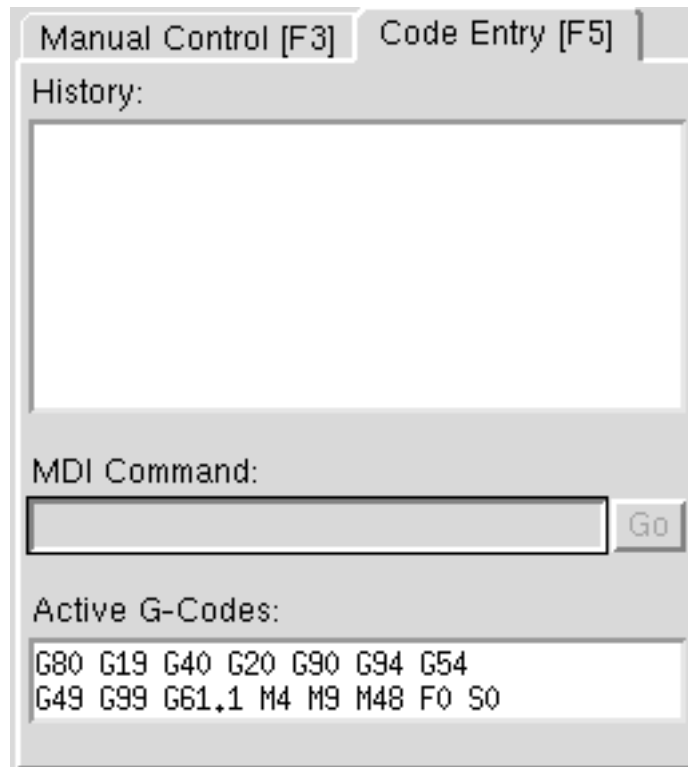
## 8.3.5 Code Entry

Code Entry” (also called MDI), allows G-code programs can be entered manually, one line at a time. When the machine is not turned on, or when a program is running, the code entry controls are unavailable.

### 8.3.5.1 History

This shows MDI commands that have been typed earlier in this session.

Figure 8.3: The Code Entry tab



### 8.3.5.2 MDI Command

This allows you to enter a g-code command to be executed. Execute the command by pressing Enter or by clicking “Go”.

### 8.3.5.3 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.

### 8.3.6 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.

## 8.4 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 Code Entry mode.

The most frequently used keyboard shortcuts are shown in Table 8.1.



Table 8.1: Most Common Keyboard Shortcuts

Keystroke	Action Taken
F1	Toggle Emergency Stop
F2	Turn machine on/off
X, `	Activate first axis
Y, 1	Activate second axis
Z, 2	Activate third axis
A, 3	Activate fourth axis
I	Select jog increment
C	Continuous jog
Home	Send active axis Home
Shift-Home	Set G54 offset for active axis
Left, Right	Jog first axis
Up, Down	Jog second axis
Pg Up, Pg Dn	Jog third axis
[, ]	Jog fourth axis
O	Open File
Control-R	Reload File
R	Run file
P	Pause execution
S	Resume Execution
ESC	Stop execution
Control-K	Clear backplot
V	Cycle among preset views

## 8.5 emctop: Show EMC Status

AXIS includes a program called “emctop” which shows some of the details of emc’s state. You can run this program by invoking `MACHINE > SHOW EMC STATUS`

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.

## 8.6 mdi: Text-mode MDI interface

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

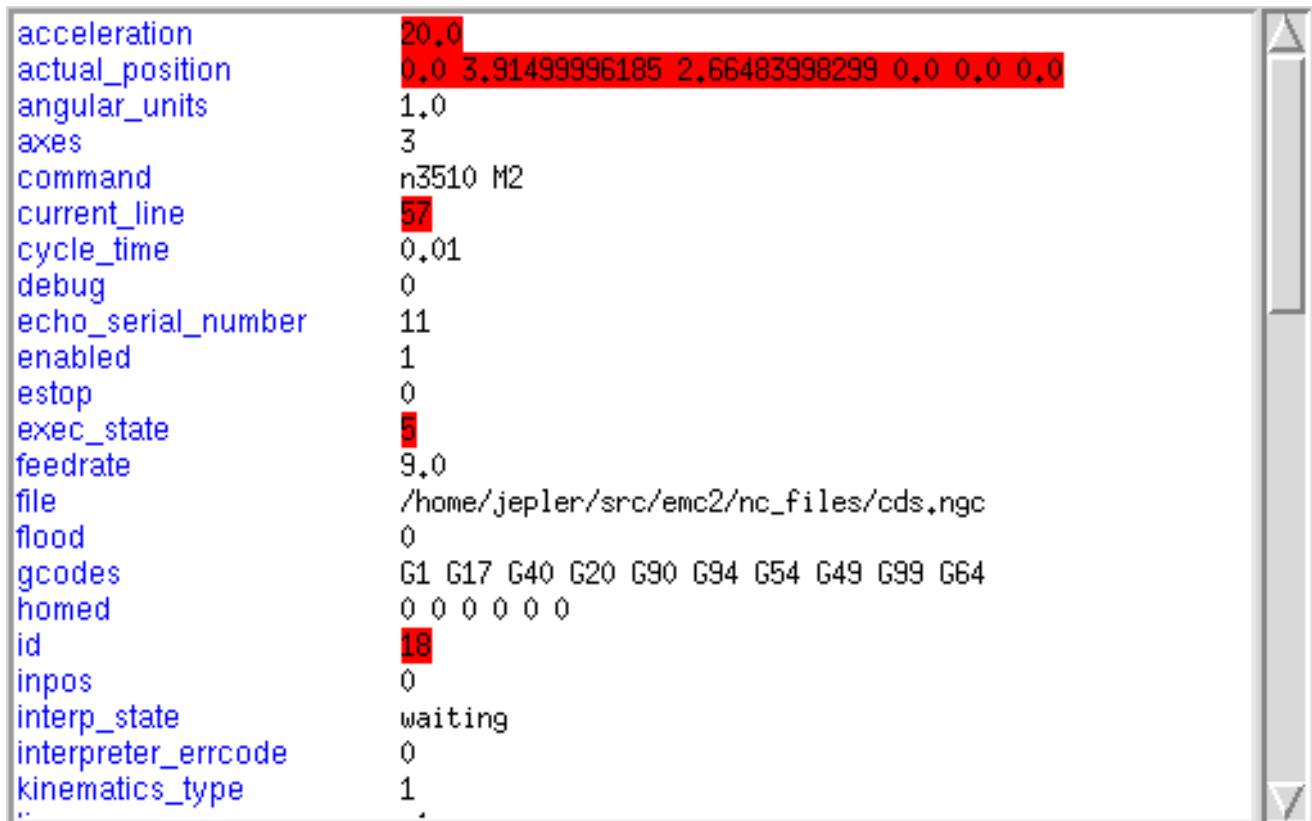
```
mdi /path/to/emc.nml
```

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 emc to be executed. A sample session of mdi is shown in Figure 8.5.

## 8.7 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.

Figure 8.4: EMC Status Window



acceleration	20.0
actual_position	0.0 3.91499996185 2.66483998299 0.0 0.0 0.0
angular_units	1.0
axes	3
command	n3510 M2
current_line	57
cycle_time	0.01
debug	0
echo_serial_number	11
enabled	1
estop	0
exec_state	5
feedrate	9.0
file	/home/jepler/src/emc2/nc_files/cds.ngc
flood	0
gcodes	G1 G17 G40 G20 G90 G94 G54 G49 G99 G64
homed	0 0 0 0 0 0
id	18
inpos	0
interp_state	waiting
interpreter_errcode	0
kinematics_type	1
..	.

Figure 8.5: MDI Session

```

$ mdi ~/emc2/configs/sim/emc.nml
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)

```

- `emc` provides access to the emc command, status, and error channels.
- `gcode` provides access to the rs274ngc interpreter
- `_togl` provides an OpenGL widget that can be used in Tkinter applications
- `minigl` provides access to the subset of OpenGL used by AXIS

## 8.8 Advanced configuration of AXIS

### 8.8.1 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 an M2 command, 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 program is specified in the .ini file as `[EMC]PROGRAM_FILTER` and must take the input as the first argument, or from standard input if no argument is given.

The filter program 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 emc when “Run”.

The list of permitted file extensions can be extended by adding one or more `[EMC]PROGRAM_EXTENSION` lines. For instance, if the filter 'myfilter' can convert png depth images into g-code, then it would be appropriate to write

```
[EMC]
PROGRAM_FILTER = myfilter
PROGRAM_EXTENSION = *.png Depth Image
```

When 'myfilter' is given rs274ngc on its input, it should simply copy it to the output.

In a future version, AXIS will come with a standard filter which can recognize files based on extension or magic numbers and invoke the filter appropriate to that type of file.

### 8.8.2 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 serve as a reference for the configurable items in the display area.

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

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.

## **Chapter 9**

# **Using The TKEMC Graphical Interface**

# Chapter 10

## Using The MINI Graphical Interface

### 10.1 Introduction<sup>1</sup>

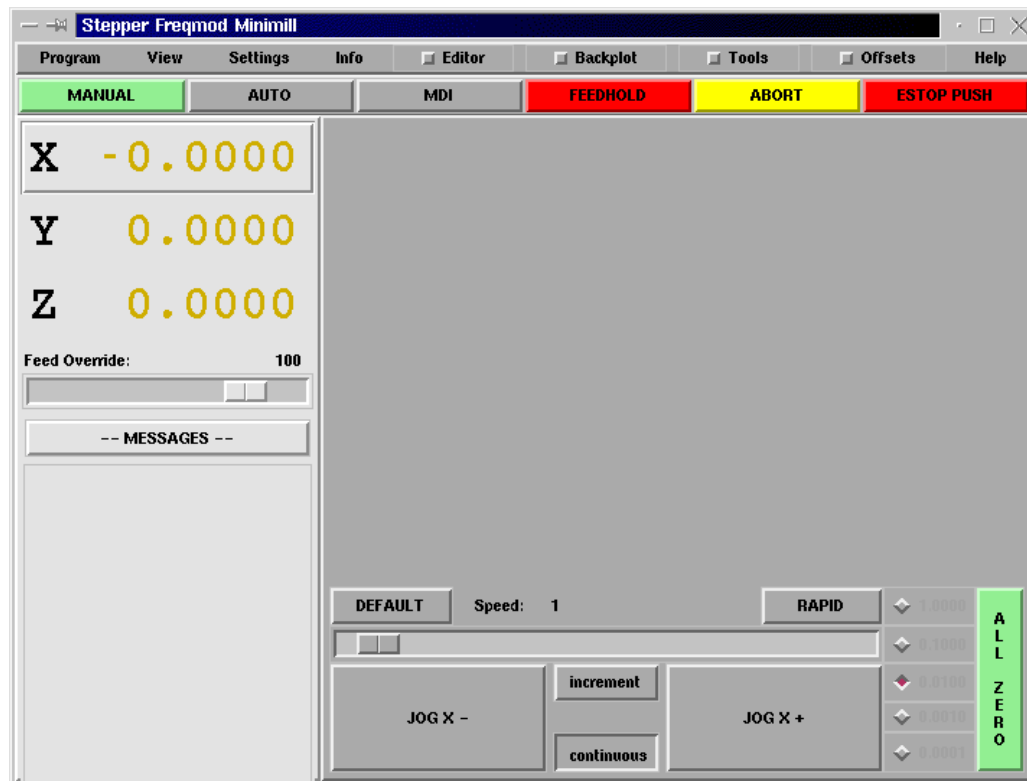


Figure 10.1: The Mini Graphical Interface

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. Mini was written largely for the Sherline CNC mill. Parts of this chapter are copied from the instructions that were written for that mill by Joe Martin and Ray Henry.

<sup>1</sup>Much of this chapter quotes from a chapter of the Sherline CNC operators manual.

## 10.2 Screen layout

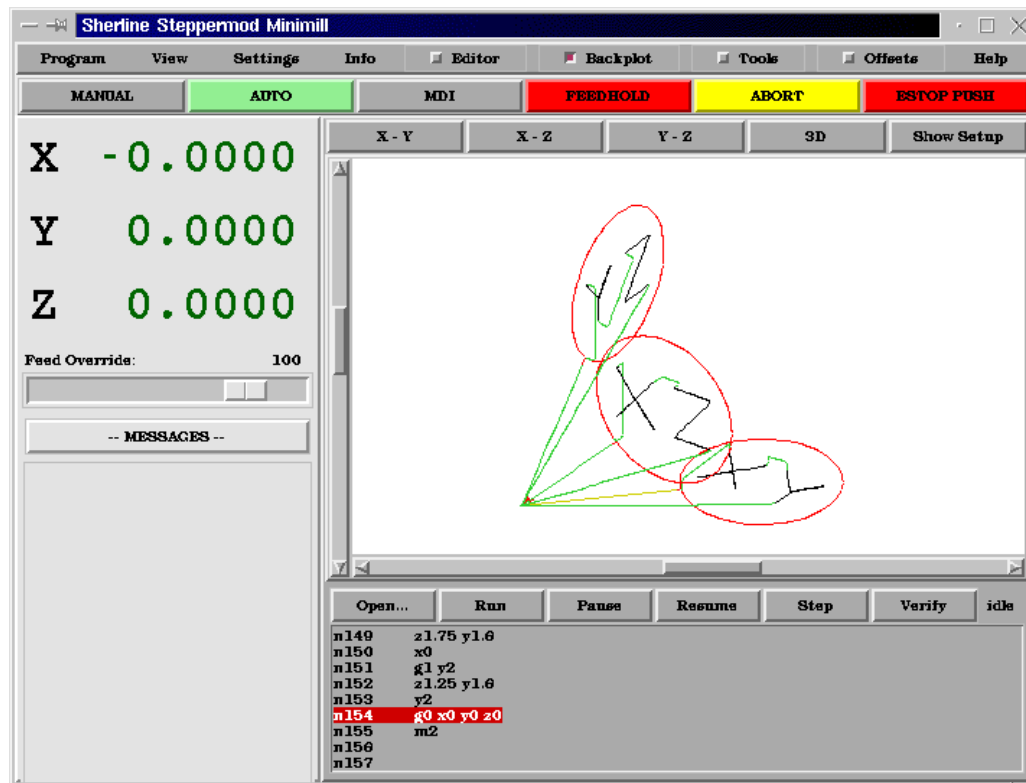


Figure 10.2: Mini Display for a Running EMC

The Mini screen is laid out in several sections. (See Figure 10.1 ) 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 figure 10.1 with figure 10.2 you will see many differences. In the second figure

- each axis has been homed – the display numbers are dark green
- the EMC 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 EMC and allows you to make changes to it.

## 10.3 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 EMC 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 shown 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 chapter on coordinate systems.

**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 introductory part programming chapter.

**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.

## 10.4 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.

### 10.4.1 MANUAL

This button or pressing <F3> sets the EMC 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.

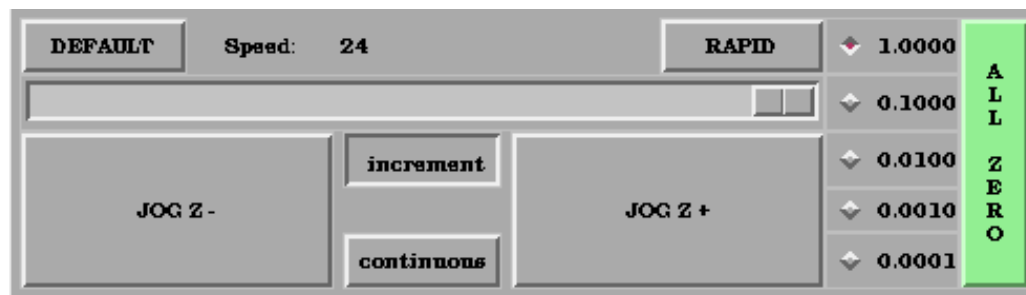


Figure 10.3: Manual Mode Buttons

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 figure 10.1) 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.

The EMC 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.

## 10.4.2 AUTO

When the Auto button is pressed, or <F4> on the keyboard, the EMC is changed into 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 query keys are bound to feedrate override. The 0 sets 100%, 9 sets 90% and such. Other keys work much the same as they do with the tkemc graphical interface.

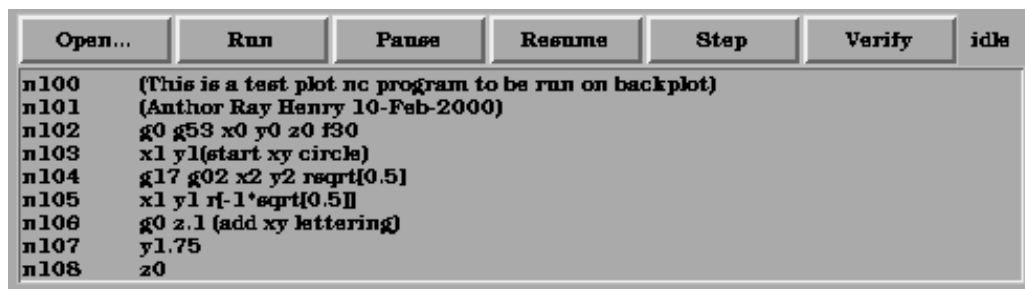


Figure 10.4: Auto Mode

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 displays to the right of the text that allows the operator to shift the restart line forward or backwards. 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 feedrate, and a finger poised over the pause button is advised during any program restart.

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 the EMC. 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. Feedrate 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 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.

### 10.4.3 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.

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.

### 10.4.4 [FEEDHOLD] – [CONTINUE]

Feedhold is a toggle. When the EMC 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 feedrate 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.

### 10.4.5 [ABORT]

The abort button stops any motion when it is pressed. It also removes the motion command from the EMC. 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.

### 10.4.6 [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 EMC 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.

Most of the time, when we abort or EStop 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, the EMC 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 the EMC 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.

## 10.5 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.

### 10.5.1 Axis Position Displays

The axis position displays work exactly like they do with tkemc. 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.

## 10.5.2 Feedrate Override

Immediately below the axis position displays is the feedrate 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 feedrate.

## 10.5.3 Messages

The message display located under the axis positions is a sort of scratch pad for the EMC. 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 the EMC 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.

## 10.6 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.

### 10.6.1 Program Editor

```

file  edit  settings  scripts  Help
-----
n100 (This is a test plot.nc program to be run on backplot)
n101 (Author Ray Henry 10-Feb-2000)
n102 G0 G90 X0 Y0 Z0 F60
n103 X1 Y1 (start xy circle)
n104 G17 G02 X2 Y2 R=90(0.5)
n105 X1 Y1 R=1 (start 0.5)
n106 G0 Z1 (add xy lettering)
n107 Y1.75
n108 Z0
n109 G1 Y1.25 X1.4
n110 Y1.5 X1.2
n111 Y1.25 X1
n112 Y1.75 X1.4
n113 G0 Z1
n114 Y1.75 X1.6
n115 Z0
n116 G1 Y1.5 X1.8
n117 Y1.75 X2
n118 Y1.5 X1.8
n119 Y1.25
n120 G0 X0 Y0 Z0
n121 X1 Z1 (start xz circle)
n122 G18 G02 X2 Z2 R=90(0.5)
n123 X1 Z1 R=1 (start 0.5)
n124 G0 Y1 (add xz lettering)
n125 Z1.75

```

Figure 10.5: 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 EMC interpreter with the same menu click. You can work on a file in here for a while and then save and load if the EMC 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.

### 10.6.2 Backplot Display

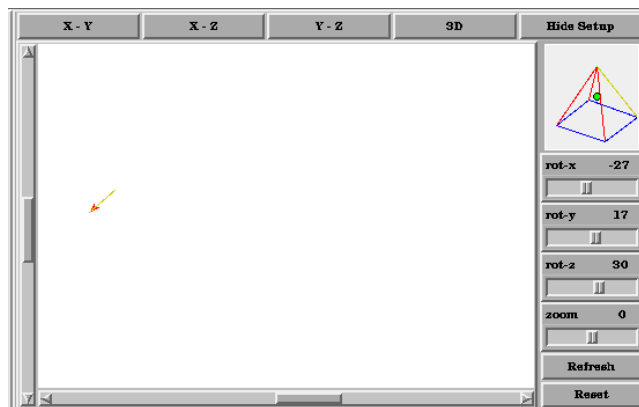


Figure 10.6: Mini's Backplotter

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 feedrate 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.

### 10.6.3 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.

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 decending 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 EMC configuration files but you will run out of display space in Mini long before you get there. (Hint You can use menu -> view -> show popin full to see more tools if you need.)

**TOOL SETUP**  
Click or tab to edit. Press enter to return to keyboard machine control.

TOOL NUMBER	LENGTH	DIAMETER	COMMENT
1	1.456	0.250	Drill
2	1.000	0.4968	End Mill
3	0.0	0.0	empty
4	0.0	0.0	empty
5	0.0	0.0	empty
6	0.0	0.0	empty

Figure 10.7: Mini Tool Display

### 10.6.4 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. You can also teach using an edfinder by adding the

**COORDINATE SYSTEM SETUP**  
Click value to edit with keyboard. Press enter to return to keyboard control of machine.

	Axis	Value	
G54 G55 G56 G57 G58 G59 G59.1 G59.2 G59.3	X	0.000000	Teach
	Y	0.000000	Teach
	Z	0.000000	Teach
	Offset By Radius	0.000000	
	Offset By Length	0.000000	
	Subtract	Add	
		Zero All G55	Write And Load File

Figure 10.8: Mini Offset Display

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.

## 10.7 Keyboard Bindings

A number of the bindings used with tkemc 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 regardless of the mode. Others change with the mode that EMC is operating in.

### 10.7.1 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

### 10.7.2 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

### 10.7.3 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

### 10.8 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.



# Chapter 11

## Machining Center Overview

This section gives a brief description of how a machining center is viewed from the input and output ends of the Interpreter. It is assumed the reader is already familiar with machining centers.

Both the RS274/NGC input language and the output canonical machining functions have a view of (1) mechanical components of a machining center being controlled and (2) what activities of the machining center may be controlled, and what data is used in control.

The view here includes some items that a given machining center may not have, such as a pallet shuttle. The RS274/NGC language and canonical machining functions may be used with such a machine provided that no NC program used with the controller includes commands intended to activate physical capabilities the machine does not have. For such a machine, it would be useful to modify the Interpreter so it will reject input commands and will not produce output canonical function calls addressed to non-existent equipment.

### 11.1 Mechanical Components

A machining center 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.

#### 11.1.1 Linear Axes

A machining center has independent mechanisms<sup>1</sup> for producing relative linear motion of the tool and workpiece in three mutually orthogonal directions. These are the X, Y and Z axes.

#### 11.1.2 Rotational axes

Three additional independent mechanisms produce relative rotation of the workpiece and the tool around an axis. These mechanisms (often a rotary table on which the workpiece is mounted or a drum on which the spindle is mounted) are called rotational axes and labelled A, B, and C. The A-axis is parallel to the X-axis. B is parallel to the Y-axis, and C parallel to the Z-axis<sup>2</sup>. Each rotational mechanism may or may not have a mechanical limit on how far it can rotate.

---

<sup>1</sup>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 as would be produced by independent axes.

<sup>2</sup>The requirement of parallelism is not used by either language, so both languages are usable if any rotational axis is not parallel to any linear axis. Rotational axis commands flow through both languages to lower levels of control without significant change in nature.

### **11.1.3 Spindle**

A machining center has a spindle which holds one cutting tool, probe, or other item. The spindle can rotate in either direction, and it can be made to rotate at a constant rate, which may be changed. Except on machines where the spindle may be moved by moving a rotational axis, the axis of the spindle is kept parallel to the Z-axis and is coincident with the Z-axis when X and Y are zero. The spindle can be stopped in a fixed orientation or stopped without specifying orientation.

### **11.1.4 Coolant**

A machining center has components to provide mist coolant and/or flood coolant.

### **11.1.5 Pallet Shuttle**

A machining center has a pallet shuttle system. The system has two movable pallets on which workpieces can be fixtured. Only one pallet at a time is in position for machining.

### **11.1.6 Tool Carousel**

A machining center has a tool carousel with slots for tools fixed in tool holders.

### **11.1.7 Tool Changer**

A machining center has a mechanism for changing tools (fixed in tool holders) between the spindle and the tool carousel.

### **11.1.8 Message Display**

A machining center has a device that can display messages.

### **11.1.9 Feed and Speed Override Switches**

A machining center has separate feed and speed override switches, which let the operator specify that the actual feed rate or spindle speed used in machining should be some percentage of the programmed rate. See Section [11.3.1](#).

### **11.1.10 Block Delete Switch**

A machining center has a block delete switch. See Section [11.3.2](#).

### **11.1.11 Optional Program Stop Switch**

A machining center has an optional program stop switch. See Section [11.3.3](#).

## 11.2 Control and Data Components

### 11.2.1 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.

### 11.2.2 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 decreases 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<sup>3</sup>

### 11.2.3 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.

### 11.2.4 Coordinate 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. If physical limits on axis speed make the desired rate unobtainable, all axes are slowed to maintain the desired path.

### 11.2.5 Feed Rate

The rate at which the controlled point or the axes move is nominally a steady rate which may be set by the user. In the Interpreter, the interpretation of the feed rate is as follows unless inverse time feed rate mode is being used in the RS274/NGC view (see Section 13.19).

<sup>3</sup>If the parallelism requirement is violated, the system builder will have to say how to distinguish clockwise from counterclockwise.

1. For motion involving one or more of the X, Y, and Z axes (with or without simultaneous rotational axis motion), the feed rate means length units per minute along the programmed XYZ path, as if the rotational axes were not moving.
2. For motion of one rotational axis with X, Y, and Z axes not moving, the feed rate means degrees per minute rotation of the rotational axis.
3. For motion of two or three rotational axes with X, Y, and Z axes not moving, the rate is applied as follows. Let  $dA$ ,  $dB$ , and  $dC$  be the angles in degrees through which the A, B, and C axes, respectively, must move. Let  $D = \sqrt{(dA)^2 + (dB)^2 + (dC)^2}$ . Conceptually, D is a measure of total angular motion, using the usual Euclidean metric. Let T be the amount of time required to move through D degrees at the current feed rate in degrees per minute. The rotational axes should be moved in coordinated linear motion so that the elapsed time from the start to the end of the motion is T plus any time required for acceleration or deceleration.

### 11.2.6 Coolant

Flood coolant and mist coolant may each be turned on independently. The RS274/NGC language turns them off together (see Section 14.4).

### 11.2.7 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 11.2.15) the machine will stop exactly at the end of the previous programmed move, as though it was in exact path mode.

### 11.2.8 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 in degrees per minute, as described in Section 11.2.5.

### 11.2.9 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.

### 11.2.10 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.

### **11.2.11 Tool Carousel**

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

### **11.2.12 Tool Change**

A machining center may be commanded to change tools.

### **11.2.13 Pallet Shuttle**

The two pallets may be exchanged by command.

### **11.2.14 Feed and Speed Override Switches**

The feed and speed override switches may be enabled (so they work as expected) or disabled (so they have no effect on the feed rate or spindle speed). The RS274/NGC language has one command that enables both switches and one command that disables both (see Section 14.4). See Section 11.3.1 for further details.

### **11.2.15 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 Section 13.15.

## **11.3 Interpreter Interaction with Switches**

The Interpreter interacts with three 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.

### **11.3.1 Feed and Speed Override Switches**

The Interpreter will interpret RS274/NGC commands which enable (M48) or disable (M49) the feed and speed override switches. It is useful to be able to override these switches for some machining operations. The idea is that optimal settings have been included in the program, and the operator should not change them.

EMC2 reacts to the setting of the speed or feed override switches on the control panel, when these switches are enabled.

### **11.3.2 Block Delete Switch**

If the block delete switch is on, lines of RS274/NGC code which start with a slash (the block delete character) are not interpreted. If the switch is off, such lines are interpreted.

The Interpreter runs in two stages (read and execute). The driver tells the Interpreter when to perform each stage. When the Interpreter reads a line starting with a slash, it informs the driver,

“I just read a line starting with a slash.” The driver checks the setting of the block delete switch. If the switch is off, it tells the Interpreter, “Execute that line.” If the switch is on, the driver does not tell the Interpreter to execute the line. Instead, it tells the Interpreter to read another line, with the result that the line starting with the slash is not executed.

### 11.3.3 Optional Program Stop Switch

The optional program stop switch works as follows. If this switch is on and an input RS274/NGC code line contains an M1 code, program execution is supposed to stop until the cycle start button is pushed.

EMC2 checks the optional stop switch when the `OPTIONAL_PROGRAM_STOP` canonical function call is executed and either stops (if the switch is on) or not (if the switch is off).

## 11.4 Tool File

A tool file is required to use the Interpreter. The file tells which tools are in which carousel slots and what the length and diameter of each tool are.

The format of a tool file is exemplified in Table 11.1.

Table 11.1: Sample Tool File

Pocket	FMS	TLO	Diameter	Comment
1	1	2.0	1.0	
2	2	1.0	0.2	
5	5	1.5	0.25	endmill
10	10	2.4	-0.3	for testing
21	21	173.740	0	1/2" spot drill
32	32	247.615	0	8.5mm drill
41	41	228.360	0	10mm tap
60	60	0	0	large chuck

The file consists of any number of header lines, followed by one blank line, followed by any number of lines of data. The header lines are ignored. It is important that there be exactly one blank line (with no spaces or tabs, even) before the data. The header line shown in Table 11.1 describes the data columns, so it is suggested (but not required) that such a line always be included in the header.

Each data line of the file contains the data for one tool. Each line has five entries. The first four entries are required. The last entry (a comment) 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 first three entries on a line and a space, tab, or newline at the end of the fourth entry. The meanings of the columns and the type of data to be put in each are as follows.

The “Pocket” column contains an unsigned integer which represents the pocket number (slot number) of the tool carousel slot in which the tool is placed. The entries in this column must all be different.

The “FMS” column contains an 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.

The “TLO” column contains a real number which represents the tool length offset. This number will be used if tool length offsets are being used and this pocket is selected. This is normally a positive real number, but it may be zero or any other number if it is never to be used.

The “Diameter” column contains a real number. This number is used only if tool radius compensation is turned on using this pocket. 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, negative, or zero) representing the difference between the measured diameter of the tool and the nominal diameter. If cutter radius 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 units used for the length and diameter of the tool may be in either millimeters or inches, but if the data is used by an NC program, the user must be sure the units used for a tool in the file are the same as the units in effect when NC code that uses the tool data is interpreted. The table shows a mixture of types of units.

The lines do not have to be in any particular order. Switching the order of lines has no effect unless the same slot number is used on two or more lines, which should not normally be done, in which case the data for only the last such line will be used.

## 11.5 Parameters

In the RS274/NGC language view, a machining center maintains an array of 5400 numerical parameters. Many of them have specific uses. The parameter array persists over time, even if the machining center is powered down. EMC2 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.

Table 11.2: Parameters Used by the RS274NGC Interpreter

Parameter Number(s)	Meaning
5161-5166	“G28” Home
5181-5186	“G30” Home
5211-5216	“G92” offset
5220	Coordinate System Number
5221-5226	Coordinate System 1
5241-5246	Coordinate System 2
5261-5266	Coordinate System 3
5281-5286	Coordinate System 4
5301-5306	Coordinate System 5
5321-5326	Coordinate System 6
5341-5346	Coordinate System 7
5361-5366	Coordinate System 8
5381-5386	Coordinate System 9

The format of a parameter file is shown in Table 11.3. 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 Table 11.3 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 Table 11.3 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 signalled 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 signalled 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 11.3: Parameter File Format

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

## 11.6 Coordinate Systems

In the RS274/NGC language view, a machining center has an absolute coordinate system and nine program coordinate systems.

You can set the offsets of the nine program coordinate systems using `G10 L2 Pn` (`n` is the number of the coordinate system) with values for the axes in terms of the absolute coordinate system. See Section [13.6](#).

You can select one of the nine systems by using `G54`, `G55`, `G56`, `G57`, `G58`, `G59`, `G59.1`, `G59.2`, or `G59.3` (see Section [13.14](#)). It is not possible to select the absolute coordinate system directly.

You can offset the current coordinate system using `G92` or `G92.3`. This offset will then apply to all nine program coordinate systems. This offset may be cancelled with `G92.1` or `G92.2`. See Section [13.18](#).

You can make straight moves in the absolute machine coordinate system by using `G53` with either `G0` or `G1`. See Section [13.13](#).

Data for coordinate systems is stored in parameters.

During initialization, the coordinate system is selected that is specified by parameter 5220. A value of 1 means the first coordinate system (the one `G54` activates), a value of 2 means the second coordinate system (the one `G55` activates), and so on. It is an error for the value of parameter 5220 to be anything but a whole number between one and nine.



# Chapter 12

## Language Overview

The RS274/NGC language is based on lines of code. Each line (also called a “block”) may include commands to a machining center 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”, and “X3” provides an argument value (the value of X should be 3 at the end of the move). Most RS274/NGC commands start with either G or M (for General and Miscellaneous). The words for these commands are called “G codes” and “M codes.”

The RS274/NGC 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 signalled 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 RS274/NGC 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.

### 12.1 Format of a line

A permissible line of input RS274/NGC 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.

## 12.2 Line Number

A line number is the letter N followed by an integer (with no sign) between 0 and 99999 written with no more than five digits (000009 is not OK, for example). Line numbers 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.

## 12.3 Word

A word is a letter other than N followed by a real value.

Words may begin with any of the letters shown in Table 12.1. 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.

Table 12.1: 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	Feedrate
G	General function (See table 5)
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.
M	Miscellaneous function (See table 7)
N	Line number
P	Dwell time in canned cycles and with G4. Key used with G10
Q	Feed increment in G83 canned cycle
R	Arc radius or canned cycle plane
S	Spindle speed
T	Tool selection
X	X axis of machine
Y	Y axis of machine
Z	Z axis of machine

### 12.3.1 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 to be close to an integer is considered close enough if it is within 0.0001 of an integer.

### 12.3.2 Parameter Value

A parameter value is the pound character # followed by a real value. The real value must evaluate to an integer between 1 and 5399. The integer is a parameter number, and the value of the parameter value is whatever number is stored in the numbered parameter.

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.

### 12.3.3 Expressions and Binary Operations

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 may be 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]] ].

Binary operations appear only inside expressions. Nine binary operations are defined. 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 binary operations are divided into three groups. The first group is: power. The second group is: multiplication, division, and modulus. The third group is: addition, subtraction, logical non-exclusive or, logical exclusive or, and logical and. If operations are strung together (for example in the expression [2.0 / 3 \* 1.5 - 5.5 / 11.0]), operations in the first group are to be performed before operations in the second group and operations in the second group before operations in the

third 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 simplifies 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.

### 12.3.4 Unary Operation Value

A unary operation value is either “ATAN” followed by one expression divided by another expression (for example “ATAN[2]/[1+3]”) or any other unary operation name followed by an expression (for example “SIN[90]”). The unary operations are: ABS (absolute value), ACOS (arc cosine), ASIN (arc sine), ATAN (arc tangent), COS (cosine), EXP (e raised to the given power), FIX (round down), FUP (round up), LN (natural logarithm), ROUND (round to the nearest whole number), SIN (sine), SQRT (square root), and TAN (tangent). 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.

The FIX operation rounds towards the left (less positive or more negative) on a number line, so that  $FIX[2.8] = 2$  and  $FIX[-2.8] = -3$ , for example. The FUP operation rounds towards the right (more positive or less negative) on a number line;  $FUP[2.8] = 3$  and  $FUP[-2.8] = -2$ , for example.

## 12.4 Parameter Setting

A parameter setting is the following four items one after the other: (1) a pound character #, (2) a real value which evaluates to an integer between 1 and 5399, (3) an equal sign =, and (4) a real value. For example “#3 = 15” is a parameter setting meaning “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.

## 12.5 Comments and Messages

Printable characters and white space inside parentheses is a comment. A left parenthesis always starts a comment. The comment ends at the first right parenthesis found thereafter. Once a left parenthesis is placed on a line, a matching right parenthesis must appear before the end of the line. Comments may not be nested; it is an error if a left parenthesis is found after the start of a comment and before the end of the comment. Here is an example of a line containing a comment: “G80 M5 (stop motion)”. Comments do not cause a machining center to do anything.

A comment contains a 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. Comments not containing messages need not be displayed there.

## 12.6 Repeated Items

A line may have any number of G words, but two G words from the same modal group (see Section 12.9) may not appear on the same line.

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.

## 12.7 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.

## 12.8 Commands and Machine Modes

In RS274/NGC, many commands cause a machining center 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 cancelling motion.

“Non-modal” codes have effect only on the lines on which they occur. For example, G4 (dwell) is non-modal.

## 12.9 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 Table 12.2.

Table 12.2: Modal Groups

Modal Group Meaning	Member Words
Motion ("Group 1")	G0 G1 G2 G3 G33 G38.2 G80 G81 G82 G83 G84 G85 G86 G87 G88 G89
Plane selection	G17 G18 G19
Distance Mode	G90 G91
Feed Rate Mode	G93, G94
Units	G20, G21
Cutter Radius Compensation	G40, G41, G42
Tool Length Offset	G43, G49
Return Mode in Canned Cycles	G98, G99
Coordinate System Selection	G54, G55, G56, G57, G58, G59, G59.1, G59.2, G59.3
Stopping	M0, M1, M2, M30, M60
Tool Change	M6
Spindle Turning	M3, M4, M5
Coolant	M7, M8, M9. Special case: M7 and M8 may be active at the same time
Override Switches	M48, M49
Flow Control	O-
Non-modal codes ("Group 0")	G4, G10, G28, G30, G53 G92, G92.1, G92.2, G92.3 M100 to 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.

# Chapter 13

## G Codes

G codes of the RS274/NGC language are shown in Table 5 and described following that.

In the command prototypes, the hyphen (-) stands for a real value. As described earlier, a real value may be (1) an explicit number, 4, for example, (2) an expression, [2+2], for example, (3) a parameter value, #88, for example, or (4) a unary function value, `acos[0]`, for example.

In most cases, if axis words (any or all of X-, Y-, Z-, A-, B-, C-) are given, 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 have their current value. Any items in the command prototypes not explicitly described as optional are required. It is an error if a required item is omitted.

In the prototypes, 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. Using real values which are not explicit numbers as just shown in the examples is rarely useful.

If L- is written in a prototype the “-” will often be referred to as the “L number”. Similarly the “-” in H- may be called the “H number”, and so on for any other letter.

### 13.1 G0: Rapid Linear Motion

For rapid linear motion, program `G0 X- Y- Z- A- B- C-`, where all the axis words are optional, except that at least one must be used. The `G0` is optional if the current motion mode is `G0`. This will produce coordinated linear motion to the destination point at the current traverse rate (or slower if the machine will not go that fast). It is expected that cutting will not take place when a `G0` command is executing.

It is an error if:

- all axis words are omitted.

If cutter radius compensation is active, the motion will differ from the above; see Chapter ???. If `G53` is programmed on the same line, the motion will also differ; see Section 13.13.

### 13.2 G1: Linear Motion at Feed Rate

For linear motion at feed rate (for cutting or not), program `G1 X- Y- Z- A- B- C-`, where all the axis words are optional, except that at least one must be used. The `G1` is optional if the current

motion mode is G1. This will produce coordinated linear motion to the destination point at the current feed rate (or slower if the machine will not go that fast).

It is an error if:

- all axis words are omitted.

If cutter radius compensation is active, the motion will differ from the above; see Chapter ???. If G53 is programmed on the same line, the motion will also differ; see Section 13.13.

### 13.3 G2, G3: Arc at Feed Rate

A circular or helical arc is specified using either G2 (clockwise arc) or G3 (counterclockwise arc). 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). If the arc is circular, it lies in a plane parallel to the selected plane.

If a line of RS274/NGC code makes an arc and includes rotational axis motion, the rotational axes turn at a constant rate so that the rotational motion starts and finishes when the XYZ motion starts and finishes. Lines of this sort are hardly ever programmed.

If cutter radius compensation is active, the motion will differ from what is described here. See Chapter ??.

Two formats are allowed for specifying an arc. We will call these the center format and the radius format. In both formats the G2 or G3 is optional if it is the current motion mode.

#### 13.3.1 Radius format arcs

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 X- Y- Z- A- B- C- 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 180 degrees or less, while a negative radius indicates a turn of 180 degrees to 359.999 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.

It is not good practice to program radius format arcs that are nearly full circles or are semicircles (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. Nearly full circles are outrageously bad, semicircles (and nearly so) are only very bad. Other size arcs (in the range tiny to 165 degrees or 195 to 345 degrees) are OK.

Here is an example of a radius format command to mill an arc: G17 G2 x 10 y 15 r 20 z 5.

That means to make 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.



### 13.3.2 Center format arcs

In the center format, the coordinates of the end point of the arc in the selected plane are specified along with the offsets of the center of the arc from the current location. In this format, it is OK if the end point of the arc is the same as the current point. It is an error if:  $\hat{A}$  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 0.0002 inch (if inches are being used) or 0.002 millimeter (if millimeters are being used).

When the XY-plane is selected, program G2 X- Y- Z- A- B- C- I- J- (or use G3 instead of G2). The axis words are all optional except that at least one of X and Y must be used. I and J are the offsets from the current location (in the X and Y directions, respectively) of the center of the circle. I and J are optional except that at least one of the two must be used. It is an error if:

- X and Y are both omitted
- or I and J are both omitted.

When the XZ-plane is selected, program G2 X- Y- Z- A- B- C- I- K- (or use G3 instead of G2). The axis words are all optional except that at least one of X and Z must be used. I and K are the offsets from the current location (in the X and Z directions, respectively) of the center of the circle. I and K are optional except that at least one of the two must be used. It is an error if:

- X and Z are both omitted,
- or I and K are both omitted.

When the YZ-plane is selected, program G2 X- Y- Z- A- B- C- J- K- (or use G3 instead of G2). The axis words are all optional except that at least one of Y and Z must be used. J and K are the offsets from the current location (in the Y and Z directions, respectively) of the center of the circle. J and K are optional except that at least one of the two must be used. It is an error if:

- Y and Z are both omitted
- or J and K are both omitted.

Here is an example of a center format command to mill an arc: G17 G2 x10 y16 i3 j4 z9.

That means to make 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=16, and Z=9, with its center offset in the X direction by 3 units from the current X location and offset in the Y direction by 4 units from the current Y location. If the current location has X=7, Y=7 at the outset, the center will be at X=10, Y=11. If the starting value of Z is 9, this is a circular arc; otherwise it is a helical arc. The radius of this arc would be 5.

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.

## 13.4 G33: Spindle-Synchronized Motion

For spindle-synchronized motion, code G33 X- Y- Z- K- where K gives the distance moved in XYZ for each revolution of the spindle. This syntax is subject to change (In particular, to use F- instead of K-). For instance, G33 Z1 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.

All the axis words are optional, except that at least one must be used. This will produce coordinated linear motion to the destination point at a rate dependant on the speed of the spindle.

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

### 13.5 G4: Dwell

For a dwell, program G4 P- . This will keep the axes unmoving for the period of time in seconds specified by the P number. It is an error if:

- the P number is negative.

### 13.6 G10: Set Coordinate System Data

The RS274/NGC language view of coordinate systems is described in Section 11.6.

To set the coordinate values for the origin of a coordinate system, program G10 L2 P - X- Y- Z- A- B- C-, where the P number must evaluate to an integer in the range 1 to 9 (corresponding to G54 to G59.3) and all axis words are optional. The coordinates of the origin of the coordinate system specified by the P number are reset to the coordinate values given (in terms of the absolute coordinate system). Only those coordinates for which an axis word is included on the line will be reset.

It is an error if:

- the P number does not evaluate to an integer in the range 1 to 9.

If origin offsets (made by G92 or G92.3) were in effect before G10 is used, they 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.

Example: G10 L2 P1 x 3.5 y 17.2 sets the origin of the first coordinate system (the one selected by G54) to a point where X is 3.5 and Y is 17.2 (in absolute coordinates). The Z coordinate of the origin (and the coordinates for any rotational axes) are whatever those coordinates of the origin were before the line was executed.

### 13.7 G17, G18, G19: Plane Selection

Program G17 to select the XY-plane, G18 to select the XZ-plane, or G19 to select the YZ-plane. The effects of having a plane selected are discussed in Section 13.3 and Section 13.17

### 13.8 G20, G21: Length Units

Program G20 to use inches for length units. Program G21 to use millimeters.

It is usually a good idea to program either G20 or G21 near the beginning of a program before any motion occurs, and not to use either one anywhere else in the program. It is the responsibility of the user to be sure all numbers are appropriate for use with the current length units.

## 13.9 G28, G30: Return to Home

Two home positions are defined (by parameters 5161-5166 for G28 and parameters 5181-5186 for G30). The parameter values are in terms of the absolute coordinate system, but are in unspecified length units.

To return to home position by way of the programmed position, program G28 X- Y- Z- A- B- C- (or use G30). All axis words are optional. The path is made by a traverse move from the current position to the programmed position, followed by a traverse move to the home position. If no axis words are programmed, the intermediate point is the current point, so only one move is made.

## 13.10 G38.2: Straight Probe

This code is currently unimplemented in EMC2. It is accepted, but the behavior is undefined.

## 13.11 G40, G41, G42: Cutter Radius Compensation.

To turn cutter radius compensation off, program G40. It is OK to turn compensation off when it is already off.

Cutter radius compensation may be performed only if the XY-plane is active.

To turn cutter radius compensation on left (i.e., the cutter stays to the left of the programmed path when the tool radius is positive), program G41 D- . To turn cutter radius compensation on right (i.e., the cutter stays to the right of the programmed path when the tool radius is positive), program G42 D- . The D word is optional; if there is no D word, the radius of the tool currently in the spindle will be used. If used, the D number should normally be the slot number of the tool in the spindle, although this is not required. It is OK for the D number to be zero; a radius value of zero will be used.

It is an error if:

- the D number is not an integer, is negative or is larger than the number of carousel slots,
- the XY-plane is not active,
- or cutter radius compensation is commanded to turn on when it is already on.

The behavior of the machining center when cutter radius compensation is on is described in Chapter ??

## 13.12 G43, G49: Tool Length Offsets

To use a tool length offset, program G43 H-, where the H number is the desired index in the tool table. It is expected that all entries in this table will be positive. The H number should be, but does not have to be, the same as the slot number of the tool currently in the spindle. It is OK for the H number to be zero; an offset value of zero will be used.

It is an error if:

- the H number is not an integer, is negative, or is larger than the number of carousel slots.

To use no tool length offset, program G49.

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.

### 13.13 G53: Move in absolute coordinates

For linear motion to a point expressed in absolute coordinates, program G1 G53 X- Y- Z- A- B- C- (or use G0 instead of G1), where all the axis words are optional, except that at least one must be used. The G0 or G1 is optional if it is the current motion mode. G53 is not modal and must be programmed on each line on which it is intended to be active. This will produce coordinated linear motion to the programmed point. If G1 is active, the speed of motion is the current feed rate (or slower if the machine will not go that fast). If G0 is active, the speed of motion is the current traverse rate (or slower if the machine will not go that fast).

It is an error if:

- G53 is used without G0 or G1 being active,
- or G53 is used while cutter radius compensation is on.

See Section 11.6 for an overview of coordinate systems.

### 13.14 G54 to G59.3: Select Coordinate System

To select coordinate system 1, program G54, and similarly for other coordinate systems. The system-number-G-code pairs are: (1-G54), (2-G55), (3-G56), (4-G57), (5-G58), (6-G59), (7-G59.1), (8-G59.2), and (9-G59.3).

It is an error if:

- one of these G-codes is used while cutter radius compensation is on.

See Section 11.6 for an overview of coordinate systems.

### 13.15 G61, G61.1, G64: Set Path Control Mode

Program G61 to put the machining center into exact path mode, G61.1 for exact stop mode, or G64 P- for continuous mode with optional tolerance. It is OK to program for the mode that is already active. See Section 11.2.15 for a discussion of these modes.

### 13.16 G80: Cancel Modal Motion

Program G80 to ensure no axis motion will occur. It is an error if:

- Axis words are programmed when G80 is active, unless a modal group 0 G code is programmed which uses axis words.

### 13.17 G81 to G89: Canned Cycles

The canned cycles G81 through G89 have been implemented as described in this section. Two examples are given with the description of G81 below.

All canned cycles are performed with respect to the currently selected plane. Any of the three planes (XY, YZ, ZX) may be selected. Throughout this section, most of the descriptions assume the XY-plane has been selected. The behavior is always analogous if the YZ or XZ-plane is selected.

Rotational axis words are allowed in canned cycles, but it is better to omit them. If rotational axis words are used, the numbers must be the same as the current position numbers so that the rotational axes do not move.

All canned cycles use X, Y, R, and Z numbers in the NC code. These numbers are used to determine X, Y, R, and Z positions. The R (usually meaning retract) position is along the axis perpendicular to the currently selected plane (Z-axis for XY-plane, X-axis for YZ-plane, Y-axis for XZ-plane). Some canned cycles use additional arguments.

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: when the XY-plane is selected, X, Y, and R numbers are treated as increments to the current position and Z as an increment from the Z-axis position before the move involving Z takes place; when the YZ or XZ-plane is selected, treatment of the axis words is analogous. In absolute distance mode, the X, Y, R, and Z numbers are absolute positions in the current coordinate system.

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. 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  $L = 1$ . The L number is not sticky.

When  $L > 1$  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). The R and Z positions do not change during the repeats.

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 Section 13.20

It is an error if:

- X, Y, and Z words are all missing during a canned cycle,
- 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,
- rotational axis motion is used during a canned cycle,
- inverse time feed rate is active during a canned cycle,
- or cutter radius compensation is active during a canned cycle.

When 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.

When the XZ plane is active, the Y number is sticky, and it is an error if:

- the Y number is missing and the same canned cycle was not already active,

- or the R number is less than the Y number.

When the YZ plane is active, the X number is sticky, and it is an error if:

- the X number is missing and the same canned cycle was not already active,
- or the R number is less than the X number.

### 13.17.1 Preliminary and In-Between Motion

At the very beginning of the execution of any of the canned cycles, with the XY-plane selected, if the current Z position is below the R position, the Z-axis is traversed 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 straight traverse parallel to the XY-plane to the given XY-position,
2. a straight traverse of the Z-axis only to the R position, if it is not already at the R position.

If the XZ or YZ plane is active, the preliminary and in-between motions are analogous.

### 13.17.2 G81: Drilling Cycle

The G81 cycle is intended for drilling. Program G81 X- Y- Z- A- B- C- R- L-

1. Preliminary motion, as described above.
2. Move the Z-axis only at the current feed rate to the Z position.
3. Retract the Z-axis at traverse rate to clear Z.

**Example 1.** Suppose the current position is (1, 2, 3) and the XY-plane has been selected, and the following line of NC code is interpreted.

```
G90 G81 G98 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 number and X position are 4. The Y number and Y position are 5. The Z number and Z position are 1.5. The R number and clear Z are 2.8. Old Z is 3. The following moves take place.

1. a traverse parallel to the XY-plane to (4,5,3)
2. a traverse parallel to the Z-axis to (4,5,2.8)
3. a feed parallel to the Z-axis to (4,5,1.5)
4. a traverse parallel to the Z-axis to (4,5,3)

**Example 2.** Suppose the current position is (1, 2, 3) and the XY-plane has been selected, and the following line of NC code is interpreted.

```
G91 G81 G98 X4 Y5 Z-0.6 R1.8 L3
```

This calls for incremental distance mode (G91) and OLD\_Z retract mode (G98) and calls for the G81 drilling cycle to be repeated three times. The X number is 4, the Y number is 5, the Z number is -0.6 and the R number 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 move is a traverse along the Z-axis to (1,2,4.8), since old Z < clear Z.

The first repeat consists of 3 moves.

1. a traverse parallel to the XY-plane to (5,7,4.8)
2. a feed parallel to the Z-axis to (5,7, 4.2)
3. a traverse parallel to the Z-axis to (5,7,4.8)

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 traverse parallel to the XY-plane to (9,12,4.8)
2. a feed parallel to the Z-axis to (9,12, 4.2)
3. a traverse parallel to the Z-axis to (9,12,4.8)

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 traverse parallel to the XY-plane to (13,17,4.8)
2. a feed parallel to the Z-axis to (13,17, 4.2)
3. a traverse parallel to the Z-axis to (13,17,4.8)

### 13.17.3 G82: Drilling Cycle with Dwell

The G82 cycle is intended for drilling. Program G82 X- Y- Z- A- B- C- R- L- P-

1. Preliminary motion, as described above.
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 traverse rate to clear Z.

### 13.17.4 G83: Peck Drilling

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. Program G83 X- Y- Z- A- B- C- R- L- Q-

1. Preliminary motion, as described above.
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 back out to the clear\_z.

4. Rapid back down to the current hole bottom, backed off a bit.
5. Repeat steps 1, 2, and 3 until the Z position is reached at step 1.
6. Retract the Z-axis at traverse rate to clear Z.

It is an error if:

- the Q number is negative or zero.

### **13.17.5 G84: Right-Hand Tapping**

This code is currently unimplemented in EMC2. It is accepted, but the behavior is undefined.

### **13.17.6 G85: Boring, No Dwell, Feed Out**

The G85 cycle is intended for boring or reaming, but could be used for drilling or milling. Program  
G85 X- Y- Z- A- B- C- R- L-

1. Preliminary motion, as described above.
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 clear Z.

### **13.17.7 G86: Boring, Spindle Stop, Rapid Out**

The G86 cycle is intended for boring. This cycle uses a P number for the number of seconds to dwell. Program G86 X- Y- Z- A- B- C- R- L- P-

1. Preliminary motion, as described above.
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. Retract the Z-axis at traverse rate to clear Z.
6. Restart the spindle in the direction it was going.

The spindle must be turning before this cycle is used. It is an error if:

- the spindle is not turning before this cycle is executed.

### **13.17.8 G87: Back Boring**

This code is currently unimplemented in EMC2. It is accepted, but the behavior is undefined.

### **13.17.9 G88: Boring, Spindle Stop, Manual Out**

This code is currently unimplemented in EMC2. It is accepted, but the behavior is undefined.



### 13.17.10 G89: Boring, Dwell, Feed Out

The G89 cycle is intended for boring. This cycle uses a P number, where P specifies the number of seconds to dwell. program G89 X- Y- Z- A- B- C- R- L- P-

1. Preliminary motion, as described above.
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.

### 13.17.11 G90, G99: Set Distance Mode

Interpretation of RS274/NGC code can be in one of two distance modes: absolute or incremental.

To go into absolute distance mode, program G90. In absolute distance mode, axis numbers (X, Y, Z, A, B, C) usually represent positions in terms of the currently active coordinate system. Any exceptions to that rule are described explicitly in this Section 13.17.

To go into incremental distance mode, program G91. In incremental distance mode, axis numbers (X, Y, Z, A, B, C) usually represent increments from the current values of the numbers.

I and J numbers always represent increments, regardless of the distance mode setting. K numbers represent increments in all but one usage (see Section 13.17.8), where the meaning changes with distance mode.

## 13.18 G92, G92.1, G92.2, G92.3: Coordinate System Offsets

See Section 11.6 for an overview of coordinate systems.

To make the current point have the coordinates you want (without motion), program G92 X- Y- Z- A- B- C- , 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. It is an error if:

1. all axis words are omitted.

When G92 is executed, the origin of the currently active coordinate system moves. To do this, origin offsets are calculated so that the coordinates of the current point with respect to the moved origin are as specified on the line containing the G92. In addition, parameters 5211 to 5216 are set to the X, Y, Z, A, B, and C-axis offsets. The offset for an axis is the amount the origin must be moved so that the coordinate of the controlled point on the axis has the specified value.

Here is an example. Suppose the current point is at X=4 in the currently specified coordinate system and the current X-axis offset is zero, then G92 x7 sets the X-axis offset to -3, sets parameter 5211 to -3, and causes the X-coordinate of the current point to be 7.

The axis offsets are always used when motion is specified in absolute distance mode using any of the nine coordinate systems (those designated by G54 - G59.3). Thus all nine coordinate systems are affected by G92.

Being in incremental distance mode has no effect on the action of G92.

Non-zero offsets may be already be in effect when the G92 is called. If this is the case, the new value of each offset is A+B, where A is what the offset would be if the old offset were zero, and B is the old

offset. For example, after the previous example, the X-value of the current point is 7. If G92 X9 is then programmed, the new X-axis offset is -5, which is calculated by  $[[7-9] + -3]$ .

To reset axis offsets to zero, program G92.1 or G92.2. G92.1 sets parameters 5211 to 5216 to zero, whereas G92.2 leaves their current values alone.

To set the axis offset values to the values given in parameters 5211 to 5216, program G92.3.

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 5216. 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. If other programs are to run between the the program that sets the offsets and the one that restores them, make a copy of the parameter file written by the first program and use it as the parameter file for the second program.

### 13.19 G93, G94: Set Feed Rate Mode

Two feed rate modes are recognized: units per minute and inverse time. Program G94 to start the units per minute mode. Program G93 to start the inverse time mode.

In units per minute feed rate 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.

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 traverse) motions. It is an error if:

- inverse time feed rate mode is active and a line with G1, G2, or G3 (explicitly or implicitly) does not have an F word.

### 13.20 G98, G99: Set Canned Cycle Return Level

When the spindle retracts during canned cycles, there is a choice of how far it retracts: (1) retract perpendicular to the selected plane to the position indicated by the R word, or (2) retract perpendicular to the selected plane to the position that axis was in just before the canned cycle started (unless that position is lower than the position indicated by the R word, in which case use the R word position).

To use option (1), program G99. To use option (2), program G98. Remember that the R word has different meanings in absolute distance mode and incremental distance mode.

# Chapter 14

## M Codes

### 14.1 M0, M1, M2, M30, M60: Program Stopping and Ending

To stop a running program temporarily (regardless of the setting of the optional stop switch), program M0.

To stop a running program temporarily (but only if the optional stop switch is on), program M1.

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.

To exchange pallet shuttles and then stop a running program temporarily (regardless of the setting of the optional stop switch), program M60.

If a program is stopped by an M0, M1, or M60, pressing the cycle start button will restart the program at the following line.

To end a program, program M2. To exchange pallet shuttles and then end a program, program M30. Both of these commands have the following effects.

1. Axis offsets are set to zero (like G92.2) and origin offsets are set to the default (like G54).
2. Selected plane is set to CANON\_PLANE\_XY (like G17).
3. Distance mode is set to MODE\_ABSOLUTE (like G90).
4. Feed rate mode is set to UNITS\_PER\_MINUTE (like G94).
5. Feed and speed overrides are set to ON (like M48).
6. Cutter compensation is turned off (like G40).
7. The spindle is stopped (like M5).
8. The current motion mode is set to G\_1 (like G1).
9. Coolant is turned off (like M9).

No more lines of code in an RS274/NGC file will be executed after the M2 or M30 command is executed. Pressing cycle start will start the program back at the beginning of the file.

## 14.2 M3, M4, M5: Spindle Control

To start the spindle turning clockwise at the currently programmed speed, program M3.

To start the spindle turning counterclockwise at the currently programmed speed, program M4.

To stop the spindle from turning, program M5.

It is OK to use M3 or M4 if the 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.

## 14.3 M6: Tool Change

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 16.3), 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. The T number is an integer giving the changer slot of the tool (not its id).
- 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 in its changer slot.
- The coordinate axes will be stopped in the same absolute position they were in before the tool change (but the spindle may be re-oriented).
- 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.

The tool change may include axis motion while it is in progress. 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.

## 14.4 M7, M8, M9: Coolant Control

To turn mist coolant on, program M7.

To turn flood coolant on, program M8.

To turn all coolant off, program M9.

It is always OK to use any of these commands, regardless of what coolant is on or off.

## 14.5 M48, M49: Override Control

To enable the speed and feed override switches, program M48. To disable both switches, program M49. See Section 11.3.1 for more details. It is OK to enable or disable the switches when they are already enabled or disabled.

## 14.6 M100 to M199: User Defined Commands

To invoke a user-defined command, program `M- P- Q-` where `P-` and `Q-` are both optional. The external program “`Mnnn`” in the directory `[DISPLAY]PROGRAM_PREFIX` is executed with the `P` and `Q` values as its two arguments. Execution of the `RS274NGC` file pauses until the invoked program exits.

It is an error if

- The specified User Defined Command does not exist

# Chapter 15

## 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.

### 15.1 Subroutines: “sub”, “endsub”, “return”, “call”

Subroutines extend from a O- sub to an O- endsub. The lines inside the subroutine (the “body”) are not executed in order; instead, they are executed each time the subroutine is called with O-call.

```
O100 sub (subroutine to move to machine home)
G0 X0 Y0 Z0
O100 endsub
(many intervening lines)
O100 call
```

Inside a subroutine, O- return can be executed. This immediately returns to the calling code, just as though O- endsub was encountered.

O- call takes optional arguments, which are passed to the subroutine as #1, #2, etc up to #30. On return from the subroutine the original values of those parameters will be restored.

Subroutine calls 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.

### 15.2 Looping: “do”, “while”, “endwhile”, “break”, “continue”

The “while loop” has two structures: while/endwhile, and do/while. In each case, the loop is exited when the “while” condition is false.

```
(draw a sawtooth shape)
F100
#1 = 0
O101 while [#1 lt 10]
G1 X0
G1 Y[#1/10] X1
#1 = [#1+1]
O101 endwhile
```

Inside a while loop, `break` immediately exits the loop, and `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.

### 15.3 Conditional: “if”, “else”, “endif”

The “if” conditional executes one group of statements if a condition is true and another if it is false.

```
(Set feed rate depending on a variable)
O102 if [#2 > 5]
F100
O102 else
F200
O102 endif
```

# Chapter 16

## Other Codes

### 16.1 F: Set Feed Rate

To set the feed rate, program `F-`. The application of the feed rate is as described in Section [11.2.5](#), unless inverse time feed rate mode is in effect, in which case the feed rate is as described in Section [13.19](#).

### 16.2 S: Set Spindle Speed

To set the speed in revolutions per minute (rpm) of the spindle, program `S-`. The spindle will turn at that speed when it has been programmed to start turning. 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.

As described in Section [13.17.5](#), if a `G84` (tapping) canned cycle is active and the feed and speed override switches are enabled, the one set at the lower setting will take effect. The speed and feed rates will still be synchronized. In this case, the speed may differ from what is programmed, even if the speed override switch is set at 100%.

### 16.3 T: Select Tool

To select a tool, program `T-`, where the T number is the carousel slot for the tool. The tool is not changed until an `M6` is programmed (see Section [14.3](#)). The T word may appear on the same line as the `M6` or on a previous line. It is OK, but not normally useful, if T words appear on two or more lines with no tool change. The carousel may move a lot, but only the most recent T word will take effect at the next tool change. It is OK to program `T0`; no tool will be selected. This is useful if you want the spindle to be empty after a tool change. It is an error if:

- a negative T number is used,
- or a T number larger than the number of slots in the carousel is used.



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.

# Chapter 17

## Order of Execution

The order of execution of items on a line is critical to safe and effective machine operation. Items are executed in the order shown below if they occur on the same line.

1. Comment (including message)
2. set feed rate mode (G93, G94).
3. set feed rate (F).
4. set spindle speed (S).
5. select tool (T).
6. change tool (M6).
7. spindle on or off (M3, M4, M5).
8. coolant on or off (M7, M8, M9).
9. . enable or disable overrides (M48, M49).
10. dwell (G4).
11. set active plane (G17, G18, G19).
12. set length units (G20, G21).
13. cutter radius compensation on or off (G40, G41, G42)
14. cutter length compensation on or off (G43, G49)
15. coordinate system selection (G54, G55, G56, G57, G58, G59, G59.1, G59.2, G59.3).
16. set path control mode (G61, G61.1, G64)
17. set distance mode (G90, G91).
18. set retract mode (G98, G99).
19. home (G28, G30) or change coordinate system data (G10) or set axis offsets (G92, G92.1, G92.2, G94).
20. perform motion (G0 to G3, G33, G80 to G89), as modified (possibly) by G53.
21. stop (M0, M1, M2, M30, M60).

# Chapter 18

## G-Code Best Practices

### 18.1 Use an appropriate decimal precision

Use 3 digits after the decimal when milling in millimeters, and 4 digits after the decimal when milling in inches. In particular, arc tolerance checks are made to .001 and .0001 depending on the active units.

### 18.2 Use consistent white space

G-code is most legible when at least one space appears before words. While it is permitted to insert whitespace in the middle of numbers, there is no reason to do so.

### 18.3 Prefer “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 degrees.

### 18.4 Put important modal settings at the top of the file

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 command line.

As a good preventative measure, put the following line at the top of all your programs:

```
G17 G20 G40 G49 G54 G80 G90 G94
```

(XY plane select, inch mode, cancel diameter comp, cancel length offset, coordinate system 1, cancel motion, non-incremental motion, feed/minute mode)

Perhaps the most critical modal setting is the distance value–If you do not include G20 or G21, (inch or mm,) then different machines will mill the program at different scales. Other settings, such as the return mode in canned cycles may also be important.

## 18.5 Don't put too many things on one line

Ignore everything in Section 17, and instead write no line of code that is the slightest bit ambiguous. Similarly, don't use and set a parameter on the same line, even though the semantics are well defined. (Exception: Updating a variable to a new value, such as `#1=#1+#2`)

## 18.6 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.

# Chapter 19

## Tool File and Compensation

### 19.1 Tool File

The EMC uses a tool file that is read in when a machine control is started. In a standard release this file is named *emc.var*, *generic.var*, or *sim.var* and is used by the similarly named *run* file. The specific name of the file that will be used is set by the *ini file* that is read at startup.

A tool file is required. It tells which tools are in which carousel slots and what the length and diameter of each tool are. The Interpreter does not deal directly with tool files. A tool file is read by the EMC system and the Interpreter gets the tool information by making calls to canonical functions that obtain it.

The header line shown in Table 19.1 is essential for some of the graphical interfaces, so it is suggested (but not required) that such a line always be included as the first line in the file.

Each data line of the file contains the data for one tool. Each line has five entries. The first four entries are required. The last entry (a comment) 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 first three entries on a line and a space, tab, or newline at the end of the fourth entry. The meanings of the columns and the type of data to be put in each are as follows.

Figure 19.1: Typical Tool File

POC	FMS	LEN	DIAM	COMMENT
1	1	1.565	0.250	Drill
2	2	1.000	0.247	Reground End Mill
3	3	1.125	2.000	Carbide Insert Face Mill
4	4	-	-	-
5	5	-	-	-
32	32	-	-	-

The "POC" column contains an unsigned integer which represents the pocket number (slot number) of the tool carousel slot in which the tool is placed. The entries in this column must all be different.

The "FMS" column contains an 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.

The "LEN" column contains a real number which represents the tool length offset. This number will be used if tool length offsets are being used and this pocket is selected. This is normally a positive real number, but it may be zero.

The "DIAM" column contains a real number. This number is used if tool radius compensation is turned on using this pocket number. 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, negative, or zero) representing the difference

between the measured diameter of the tool and the nominal diameter used when the G-code for the part was written.

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 units used for the length and diameter of the tool may be in either millimeters or inches, but if the data is used by an NC program, the user must be sure the units used for a tool in the file are the same as the units in effect when NC code that uses the tool data is interpreted.

The lines do not have to be in any particular order. Switching the order of lines has no effect. If the same pocket number is used on two or more lines, which should not normally be done, the data for only the last such line will persist and be used.

## 19.2 Tool Compensation

Tool compensation can cause problems for the best of nc code programmers. But it can be a powerful aid when used to help an operator get a part to size. By setting and resetting length and diameter of tools in a single tool table, offsets can be made during a production run that allow for variation in tool size, or for minor deviation from the programmed distances and size. And these changes can be made without the operator having to search through and change numbers in a program file.

Throughout this unit you will find occasional references to canonical functions where these are necessary for the reader to understand how a tool offset works in a specific situation. These references are intended to give the reader a sense of sequence rather than requiring the reader to understand the way that canonical functions themselves work within the EMC.

## 19.3 Tool Length Offsets

Tool length offsets are given as positive numbers in the tool table. A tool length offset is programmed using G43 Hn, where n is the desired table index. It is expected that all entries in the tool table will be positive. The H number is checked for being a non-negative integer when it is read. The interpreter behaves as follows.

1. If G43 Hn is programmed, A `USE_TOOL_LENGTH_OFFSET(length)` function call is made (where length is the value of the tool length offset entry in the tool table whose index is n), `tool_length_offset` is reset in the machine settings model, and the value of `current_z` in the model is adjusted. Note that n does not have to be the same as the slot number of the tool currently in the spindle.
2. If G49 is programmed, `USE_TOOL_LENGTH_OFFSET(0.0)` is called, `tool_length_offset` is reset to 0.0 in the machine settings model, and the value of `current_z` in the model is adjusted. The effect of tool length compensation is illustrated in the screen shot below. Notice that the length of the tool is subtracted from the z setting so that the tool tip appears at the programmed setting. You should note that the effect of tool length compensation is immediate when you view the z position as a relative coordinate but it does affect actual machine position until you program a z move.

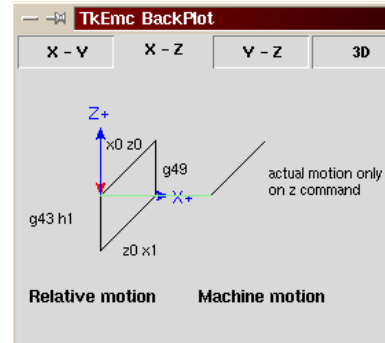
Test tool length program.

Tool #1 is 1 inch long.

```

N01 G1 F15 X0 Y0 Z0
N02 G43 H1 Z0 X1
N03 G49 X0 Z0
N04 G0 X2
N05 G1 G43 H1 G4 P10 Z0 X3
N06 G49 X2 Z0
N07 G0 X0

```



The effect of this is that in most cases the machine will pick up the offset as a ramp during the next xyz move after the g43 word.

## 19.4 Cutter Radius Compensation

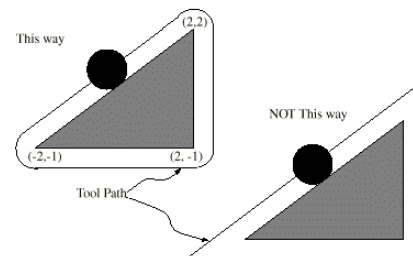
Cutter Diameter Compensation (also called Cutter Radius Compensation) is something that was obviously added onto the RS-274D specification at the demand of users, as it is VERY useful, but the implementation was poorly thought out. The purpose of this feature is to allow the programmer of the tool path program to 'virtualize' the tool path, so that the control can, at run time, determine the correct offset from the surface to be cut, based on the tools available. If you resharpen the side cutting edges of end mills, then they will end up smaller than the standard diameters.

The problem is to describe to the control whether the tool is going to be cutting on the outside of an imaginary path, or on the inside. Since these paths are not necessarily closed paths (although they can be), it is essentially impossible for the control to know which side of the line it is supposed to offset to. It was decided that there would only be two choices, tool 'left' of path, and tool 'right' of path. This is to be interpreted as left or right 'when facing the direction of cutter motion'. The interpretation is as if you were standing on the part, walking behind the tool as it progresses across the part.

### 19.4.1 Cutter Radius Compensation Detail

The cutter radius compensation capabilities of the interpreter enable the programmer to specify that a cutter should travel to the right or left of an open or closed contour in the XY-plane composed of arcs of circles and straight line segments. The contour may be the outline of material not to be machined away, or it may be a tool path to be followed by an exactly sized tool. This figure shows two examples of the path of a tool cutting using cutter radius compensation so that it leaves a triangle of material remaining.

In both examples, the shaded triangle represents material which should remain after cutting, and the line outside the shaded triangle represents the path of the tip of a cutting tool. Both paths will leave the shaded triangle uncut. The one on the left (with rounded corners) is the path the interpreter will generate. In the method on the right (the one not used), the tool does not stay in contact with the shaded triangle at sharp corners.



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, following the contour to the next

point at which machining should be done, and re-extending the Z-axis. These skip motions may be performed at feed rate (G1) or at traverse rate (G0). Inverse time feed rate (G93) or units per minute feed rate (G94) may be used with cutter radius compensation. Under G94, the feed rate will apply to the actual path of the cutter tip, not to the programmed contour.

### **Programming Instructions**

- To start cutter radius compensation, program either G41 (for keeping the tool to the left of the contour) or G42 (for keeping the tool to the right of the contour). In Figure 7, for example, if G41 were programmed, the tool would stay left and move clockwise around the triangle, and if G42 were programmed, the tool would stay right and move counterclockwise around the triangle.
- To stop cutter radius compensation, program G40.
- If G40, G41, or G42 is programmed in the same block as tool motion, cutter compensation will be turned on or off before the motion is made. To make the motion come first, the motion must be programmed in a separate, previous block.

### **D Number**

The current interpreter requires a D number on each line that has the G41 or G42 word. The value specified with D must be a non-negative integer. It represents the slot number of the tool whose radius (half the diameter given in the tool table) will be used, or it may be zero (which is not a slot number). If it is zero, the value of the radius will also be zero. Any slot in the tool table may be selected this way. The D number does not have to be the same as the slot number of the tool in the spindle.

### **Tool Table**

Cutter radius compensation uses data from the machining center's tool table. For each slot in the tool carousel, the tool table contains the diameter of the tool in that slot (or the difference between the actual diameter of the tool in the slot and its nominal value). The tool table is indexed by slot number. How to put data into the table when using the stand-alone interpreter is discussed in the tool table page.

### **Two Kinds of Contour**

The interpreter handles compensation for two types of contour:

- The contour given in the NC code is the edge of material that is not to be machined away. We will call this type a "material edge contour".
- The contour given in the NC code is the tool path that would be followed by a tool of exactly the correct radius. We will call this type a "tool path contour".

The interpreter does not have any setting that determines which type of contour is used, but the description of the contour will differ (for the same part geometry) between the two types and the values for diameters in the tool table will be different for the two types.



### Material Edge Contour

When the contour is the edge of the material, the outline of the edge is described in the NC program. For a material edge contour, the value for the diameter in the tool table is the actual value of the diameter of the tool. The value in the table must be positive. The NC code for a material edge contour is the same regardless of the (actual or intended) diameter of the tool.

Example 1 :

Here is an NC program which cuts material away from the outside of the triangle in figure above. In this example, the cutter compensation radius is the actual radius of the tool in use, which is 0.5. The value for the diameter in the tool table is twice the radius, which is 1.0.

```
N0010 G41 G1 X2 Y2 (turn compensation on and make entry move)
N0020 Y-1 (follow right side of triangle)
N0030 X-2 (follow bottom side of triangle)
N0040 X2 Y2 (follow hypotenuse of triangle)
N0050 G40 (turn compensation off)
```

This will result in the tool following a path consisting of an entry move and the path shown on the left going clockwise around the triangle. Notice that the coordinates of the triangle of material appear in the NC code. Notice also that the tool path includes three arcs which are not explicitly programmed; they are generated automatically.

### Tool Path Contour

When the contour is a tool path contour, the path is described in the NC program. It is expected that (except for during the entry moves) the path is intended to create some part geometry. The path may be generated manually or by a post-processor, considering the part geometry which is intended to be made. For the interpreter to work, the tool path must be such that the tool stays in contact with the edge of the part geometry, as shown on the left side of Figure 7. If a path of the sort shown on the right of Figure 7 is used, in which the tool does not stay in contact with the part geometry all the time, the interpreter will not be able to compensate properly when undersized tools are used.

For a tool path contour, the value for the cutter diameter in the tool table will be a small positive number if the selected tool is slightly oversized and will be a small negative number if the tool is slightly undersized. As implemented, if a cutter diameter value is negative, the interpreter compensates on the other side of the contour from the one programmed and uses the absolute value of the given diameter. If the actual tool is the correct size, the value in the table should be zero.

Tool Path Contour example

Suppose the diameter of the cutter currently in the spindle is 0.97, and the diameter assumed in generating the tool path was 1.0. Then the value in the tool table for the diameter for this tool should be -0.03. Here is an NC program which cuts material away from the outside of the triangle in the figure.

```
N0010 G1 X1 Y4.5 (make alignment move)
N0020 G41 G1 Y3.5 (turn compensation on and make first entry move)
N0030 G3 X2 Y2.5 I1 (make second entry move)
N0040 G2 X2.5 Y2 J-0.5 (cut along arc at top of tool path)
N0050 G1 Y-1 (cut along right side of tool path)
N0060 G2 X2 Y-1.5 I-0.5 (cut along arc at bottom right of tool path)
N0070 G1 X-2 (cut along bottom side of tool path)
N0080 G2 X-2.3 Y-0.6 J0.5 (cut along arc at bottom left of tool path)
N0090 G1 X1.7 Y2.4 (cut along hypotenuse of tool path)
N0100 G2 X2 Y2.5 I0.3 J-0.4 (cut along arc at top of tool path)
N0110 G40 (turn compensation off)
```

This will result in the tool making an alignment move and two entry moves, and then following a path slightly inside the path shown on the left in Figure 7 going clockwise around the triangle. This path is to the right of the programmed path even though G41 was programmed, because the diameter value is negative.

### Programming Errors and Limitations

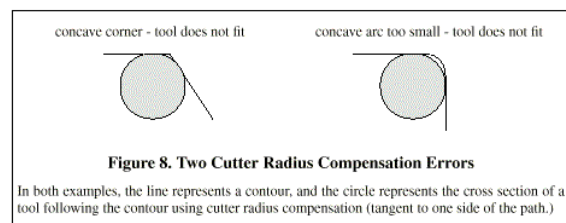
The interpreter will issue the following messages involving cutter radius compensation.

1. Cannot change axis offsets with cutter radius comp
2. Cannot change units with cutter radius comp
3. Cannot turn cutter radius comp on out of XY-plane
4. Cannot turn cutter radius comp on when already on
5. Cannot use G28 or G30 with cutter radius comp
6. Cannot use G53 with cutter radius comp
7. Cannot use XZ plane with cutter radius comp
8. Cannot use YZ plane with cutter radius comp
9. Concave corner with cutter radius comp
10. Cutter gouging with cutter radius comp
11. D word on line with no cutter comp on (G41 or G42) command
12. Tool radius index too big
13. Tool radius not less than arc radius with cutter radius comp
14. Two G codes used from same modal group.

For some of these messages additional explanation is given below.

Changing a tool while cutter radius compensation is on is not treated as an error, although it is unlikely this would be done intentionally. The radius used when cutter radius compensation was first turned on will continue to be used until compensation is turned off, even though a new tool is actually being used.

When cutter radius compensation is on, it must be physically possible for a circle whose radius is the half the diameter given in the tool table to be tangent to the contour at all points of the contour.



In particular, the interpreter treats concave corners and concave arcs into which the circle will not fit as errors, since the circle cannot be kept tangent to the contour in these situations. This error detection does not limit the shapes which can be cut, but it does require that the programmer specify the actual shape to be cut (or path to be followed), not an approximation. In this respect, the interpreter differs from interpreters used with many other controllers, which often allow these errors silently and either gouge the part or round the corner. If cutter radius compensation has already been turned on, it cannot be turned on again. It must be turned off first; then it can be turned on again. It is not necessary to move the cutter between turning compensation off and back on, but the move after turning it back on will be treated as a first move, as described below.

It is not possible to change from one cutter radius index to another while compensation is on because of the combined effect of rules 4 and 11. It is also not possible to switch compensation from one side to another while compensation is on. If the tool is already covering up the next XY destination point when cutter radius compensation is turned on, the gouging message is given when the line of NC code which gives the point is reached. In this situation, the tool is already cutting into material it should not cut.

If a D word is programmed that is larger than the number of tool carousel slots, an error message is given. In the current implementation, the number of slots is 68.

The error message, "two G Codes Used from Same Modal Group," is a generic message used for many sets of G codes. As applied to cutter radius compensation, it means that more than one of G40, G41, and G42 appears on a line of NC code. This is not allowed.

## First Move

The algorithm used for the first move when the first move is a straight line is to draw a straight line from the destination point which is tangent to a circle whose center is at the current point and whose radius is the radius of the tool. The destination point of the tool tip is then found as the center of a circle of the same radius tangent to the tangent line at the destination point. This is shown in Figure 9. If the programmed point is inside the initial cross section of the tool (the circle on the left), an error is signalled.

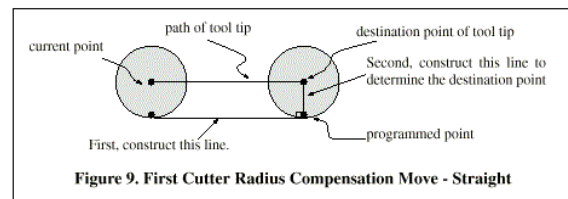


Figure 9. First Cutter Radius Compensation Move - Straight

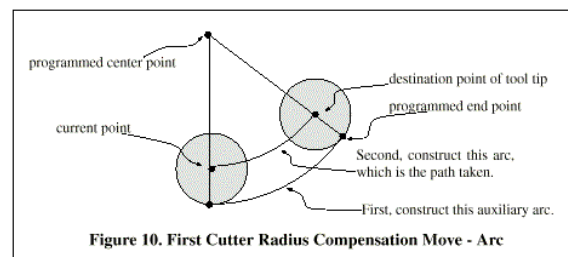


Figure 10. First Cutter Radius Compensation Move - Arc

If the first move after cutter radius compensation has been turned on is an arc, the arc which is generated is derived from an auxiliary arc which has its center at the programmed center point, passes through the programmed end point, and is tangent to the cutter at its current location. If the auxiliary arc cannot be constructed, an error is signalled. The generated arc moves the tool so that it stays tangent to the auxiliary arc throughout the move. This is shown in Figure 10.

Regardless of whether the first move is a straight line or an arc, the Z axis may also move at the same time. It will move linearly, as it does when cutter radius compensation is not being used. Rotary axis motions (A, B, and C axes) are allowed with cutter radius compensation, but using them would be very unusual.

After the entry moves of cutter radius compensation, the interpreter keeps the tool tangent to the programmed path on the appropriate side. If a convex corner is on the path, an arc is inserted to go around the corner. The radius of the arc is half the diameter given in the tool table.

When cutter radius compensation is turned off, no special exit move takes place. The next move is what it would have been if cutter radius compensation had never been turned on and the previous move had placed the tool at its current position.

## Programming Entry Moves

In general, an alignment move and two entry moves are needed to begin compensation correctly. However, where the programmed contour is a material edge contour and there is a convex corner on the contour, only one entry move (plus, possibly, a pre-entry move) is needed. The general method, which will work in all situations, is described first. We assume here that the programmer knows what the contour is already and has the job of adding entry moves.

### General Method

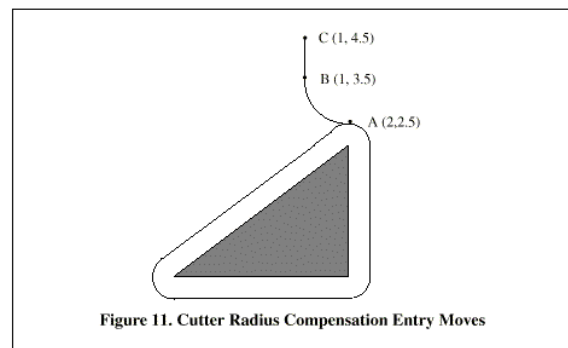
The general method includes programming an alignment move and two entry moves. The entry moves given above will be used as an example. Here is the relevant code again:

```
N0010 G1 X1 Y4.5 (make alignment move to point C)
N0020 G41 G1 Y3.5 (turn compensation on and make first entry move to point B)
N0030 G3 X2 Y2.5 I1 (make second entry move to point A)
```

See Figure 11. The figure shows the two entry moves but not the alignment move.

First, pick a point A on the contour where it is convenient to attach an entry arc. Specify an arc outside the contour which begins at a point B and ends at A tangent to the contour (and going in the same direction as it is planned to go around the contour). The radius of the arc should be larger than half the diameter given in the tool table. Then extend a line tangent to the arc from B to some point C, located so that the line BC is more than one radius long.

After the construction is finished, the code is written in the reverse order from the construction. Cutter radius compensation is turned on after the alignment move and before the first entry move. In the code above, line N0010 is the alignment move, line N0020 turns compensation on and makes the first entry move, and line N0030 makes the second entry move.



In this example, the arc AB and the line BC are fairly large, but they need not be. For a tool path contour, the radius of arc AB need only be slightly larger than the maximum possible deviation of the radius of the tool from the exact size. Also for a tool path contour, the side chosen for compensation should be the one to use if the tool is oversized. As mentioned earlier, if the tool is undersized, the interpreter will switch sides.

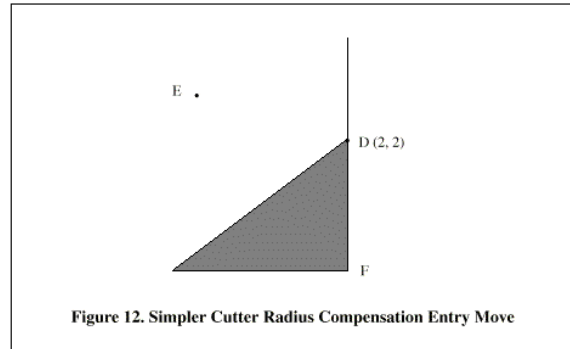
### Simple Method

If the contour is a material edge contour and there is a convex corner somewhere on the contour, a simpler method of making an entry is available. See Figure 12.

First, pick a convex corner, D. Decide which way you want to go along the contour from D. In our example we are keeping the tool to the left of the contour and going next towards F. Extend the line FD (if the next part of the contour is an arc, extend the tangent to arc FD from D) to divide the area outside the contour near D into two regions. Make sure the center of the tool is currently in the region on the same side of the extended line as the material inside the contour near D. If not, move the tool into that region. In the example, point E represents the current location of the center of the tool. Since it is on the same side of line DF as the shaded triangle, no additional move is needed. Now write a line of NC code that turns compensation on and moves to point D

```
N0010 G41 G1 X2 Y2 (turn
compensation on and make entry
move)
```

This method will also work at a concave corner on a tool path contour, if the actual tool is oversized, but it will fail with a tool path contour if the tool is undersized.



### Other Items Where Cutter Radius Compensation is Performed.

The complete set of canonical functions includes functions which turn cutter radius on and off, so that cutter radius compensation can be performed in the controller executing the canonical functions. In the interpreter, however, these commands are not used. Compensation is done by the interpreter and reflected in the output commands, which continue to direct the motion of the center of the cutter tip. This simplifies the job of the motion controller while making the job of the interpreter a little harder.

### Algorithms for Cutter Radius Compensation

The interpreter allows the entry and exit moves to be arcs. The behavior for the intermediate moves is the same, except that some situations treated as errors in the interpreter are not treated as errors in other machine controls.

#### Data for Cutter Radius Compensation

The interpreter machine model keeps three data items for cutter radius compensation: the setting itself (right, left, or off), program\_x, and program\_y. The last two represent the X and Y positions which are given in the NC code while compensation is on. When compensation is off, these both are set to a very small number (10<sup>-20</sup>) whose symbolic value (in a #define) is "unknown". The interpreter machine model uses the data items current\_x and current\_y to represent the position of the center of the tool tip (in the currently active coordinate system) at all times.

### Jon Elson's Example

All further system-specific information refers to NIST's EMC program, but much of it applies to most modern CNC controls. My method of checking these programs is to first select tool zero, which always has a diameter of zero, so offset commands are essentially ignored. Then, I tape a sheet of paper to a piece of material that sits level in my vise, as a sort of platen. I install a spring-loaded pen in the spindle. This is a standard ballpoint pen refill cartridge made of metal, in a 1/2" diameter steel housing. It has a spring that loads the pen against the front, and a 'collet' at the front that allows the pen to retract against the spring, but keeps it centered within a few thousandths of an inch. I run the program with tool zero selected, and it draws a line at the actual part's outline. (see figure below) Then, I select a tool with the diameter of the tool I intend to use, and run the program again. (Note that Z coordinates in the program may need to be changed to prevent plunging the pen through the platen.) Now, I get to see whether the G41 or G42 compensation that I specified will cut on the desired side of the part. If it doesn't, I then edit the opposite compensation command into the program, and try again. Now, with the tool on the correct side of the work, you get to see if there are any places where the tool is 'too fat' to fit in a concave part of the surface. My old Allen-Bradley 7320 was pretty forgiving on this, but EMC is a complete stickler. If you have ANY concavity where

two lines meet at less than 180 degrees on the side that a tool of finite size cuts on, EMC will stop with an error message there. Even if the gouge will be .0001" deep. So, I always make the approach on the lead-in and lead-out moves such that they just nip the corner of the part a tiny bit, providing an angle just over 180 degrees, so that EMC won't squawk. This requires some careful adjustment of the starting and ending points, which are not compensated by cutter radius, but must be chosen with an approximate radius in mind.

The operative commands are :

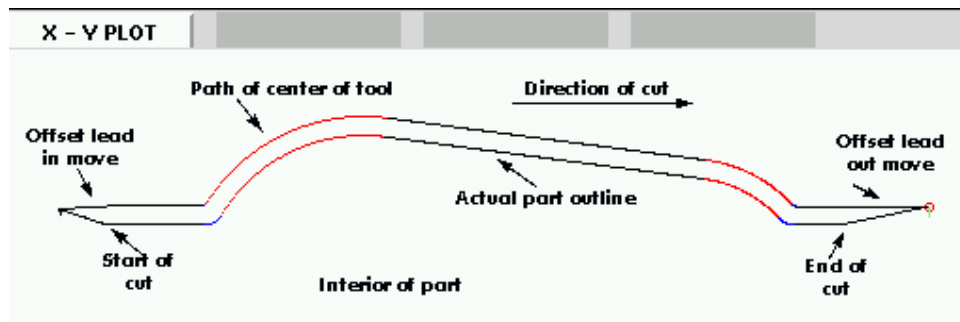
G40 - Cancel Cutter compensation  
 G41 - Cutter Compensation, Tool Left of Path  
 G42 - Cutter Compensation, Tool Right of Path

Here is a short file that cuts one side of a part with multiple convex and concave arcs, and several straight cuts, too. It is to clamp a high speed drilling spindle to the side of the main Bridgeport spindle. Most of these commands are straight from Bobcad/CAM, but lines N15 and N110 were added by me, and some of the coordinates around those lines had to be fudged a bit by me.

```
N10 G01 G40 X-1.3531 Y3.4
N15 F10 G17 G41 D4 X-0.7 Y3.1875 (COMP LEAD IN)
N20 X0. Y3.1875
N40 X0.5667 F10
N50 G03 X0.8225 Y3.3307 R0.3
N60 G02 X2.9728 Y4.3563 R2.1875
N70 G01 X7.212 Y3.7986
N80 G02 X8.1985 Y3.2849 R1.625
N90 G03 X8.4197 Y3.1875 R0.3
N100 G01 X9.
N110 G40 X10.1972 Y3.432 (COMP LEAD OUT)
N220 M02
```

Line 15 contains G41 D4, which means that the diameter of the tool described as tool #4 in the tool table will be used to offset the spindle by 1/2 the diameter, which is, of course, the tool's radius. Note that the line with the G41 command contains the endpoint of the move where the radius compensation is interpolated in. What this means is that at the beginning of this move, there is no compensation in effect, and at the end, the tool is offset by 100% of the selected tool radius. Immediately after the G41 is D4, meaning that the offset is by the radius of tool number 4 in the tool table. Note that tool DIAMETERS are entered in the tool table. (Jon's tool diameter is about 0.4890)

But, note that in line 110, where the G40 'cancel cutter compensation' command is, that cutter compensation will be interpolated out in this move. The way I have these set up, the moves in lines 15 and 110 are almost exactly parallel to the X axis, and the difference in Y coordinates is to line the tool up outside the portion of the program where cutter compensation is in force.



Some other things to note are that the program starts with a G40, to turn off any compensation that was in effect. This saves a lot of hassle when the program stops due to a concavity error, but

leaves the compensation turned on. Also note in line 15 that G17 is used to select the XY plane for circular interpolation. I have used the radius form of arc center specification rather than the I,J form. EMC is very picky about the radius it computes from I,J coordinates, and they must match at the beginning and end of the move to within  $10^{-11}$  internal units, so you will have lots of problems with arbitrary arcs. Usually, if you do an arc of 90 degrees, centered at (1.0,1.0) with a radius of 1", everything will go fine, but if it has a radius that can not be expressed exactly in just a few significant digits, or the arc is a strange number of degrees, then there will be trouble with EMC. The R word clears up all that mess, and is a lot easier to work with, anyway. If the arc is more than 180 degrees, R should be negative.

## 19.5 Tool Compensation Sources

This unit borrows heavily from the published works of Tom Kramer and Fred Proctor at NIST and the cutter compensation web page of Jon Elson.

Papers by Tom Kramer and Fred Proctor

<http://www.isd.mel.nist.gov/personnel/kramer/publications.html>

[http://www.isd.mel.nist.gov/personnel/kramer/pubs/RS274NGC\\_22.pdf](http://www.isd.mel.nist.gov/personnel/kramer/pubs/RS274NGC_22.pdf)

[http://www.isd.mel.nist.gov/personnel/kramer/pubs/RS274VGER\\_11.pdf](http://www.isd.mel.nist.gov/personnel/kramer/pubs/RS274VGER_11.pdf)

Pages by Jon Elson

<http://artsci.wustl.edu/~jmelson/>

<http://206.19.206.56/diacomp.htm>

<http://206.19.206.56/lencomp.htm>

# Chapter 20

## Coordinate System and G92 Offsets

### 20.1 Introduction

You have seen how handy a tool length offset can be. Having this allows the programmer to ignore the actual tool length when writing a part program. In the same way, it is really nice to be able to find a prominent part of a casting or block of material and work a program from that point rather than having to take account of the location at which the casting or block will be held during the machining.

This chapter introduces you to offsets as they are used by the EMC. These include;

- machine coordinates (G53)
- nine offsets (G54-G59.3)
- a set of global offsets (G92).

### 20.2 The Machine Position Command (G53)

Regardless of any offsets that may be in effect, putting a G53 in a block of code tells the interpreter to go to the real or absolute axis positions commanded in the block. For example

```
g53 g0 x0 y0 z0
```

will get you to the actual position where these three axes are zero. You might use a command like this if you have a favorite position for tool changes or if your machine has an auto tool changer. You might also use this command to get the tool out of the way so that you can rotate or change a part in a vice.

G53 is not a modal command. It must be used on each line where motion based upon absolute machine position is desired.

### 20.3 Fixture Offsets (G54-G59.3)

Work or fixture offset are used to make a part home that is different from the absolute, machine coordinate system. This allows the part programmer to set up home positions for multiple parts. A typical operation that uses fixture offsets would be to mill multiple copies of parts on "islands" in a piece, similar to figure [20.1](#)

The values for offsets are stored in the VAR file that is requested by the INI file during the startup of an EMC. In our example below we'll use G55. The values for each axis for G55 are stored as variable numbers.



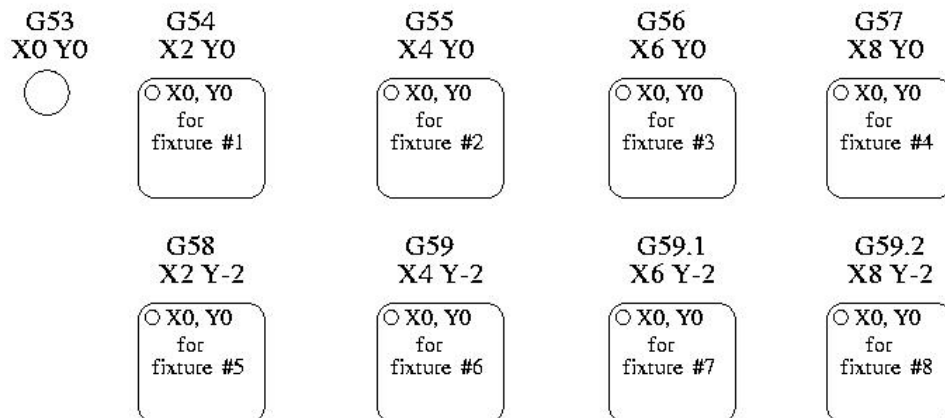


Figure 20.1: Work Offsets

```

5241  0.000000
5242  0.000000
5243  0.000000
5244  0.000000
5245  0.000000
5246  0.000000

```

In the VAR file scheme, the first variable number stores the X offset, the second the Y offset and so on for all six axes. There are numbered sets like this for each of the fixture 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 issuing a [reset] so that the EMC reads the new values. For our example let's directly edit the file so that G55 takes on the following values.

```

5241  2.000000
5242  1.000000
5243 -2.000000
5244  0.000000
5245  0.000000
5246  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 figure 20.1 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 G0 x0 y0 z0
g1 f2 z-0.2000
x1

```

```
y1  
x0  
y0  
g0 z0  
g54 x0 y0 z0  
m2
```

“But,” you say, “why is there a G54 in there near the end.” Many programmers leave the G54 coordinate system with all zero values 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.

```
G54   use preset work coordinate system 1  
G55   use preset work coordinate system 2  
G56   use preset work coordinate system 3  
G57   use preset work coordinate system 4  
G58   use preset work coordinate system 5  
G59   use preset work coordinate system 6  
G59.1 use preset work coordinate system 7  
G59.2 use preset work coordinate system 8  
G59.3 use preset work coordinate system 9
```

### 20.3.1 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 it's value is set to 1.00000. This means that when the EMC 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 startup and reset. Any value other than an interger (decimal really) between 1 and 9 will cause the EMC to fault on startup.

### 20.3.2 Setting coordinate system values within G-code.

In the general programming chapter we listed a G10 command word. This command can be used to change the values of the offsets in a coordinate system. (add here)

## 20.4 G92 Offsets

G92 is the most misunderstood and maligned part of EMC programming. The way that it works has changed just a bit from the early days to the current releases. This change has confused many users. It should be thought of as a temporary offset that is applied to all other offsets.

In response to criticism of it, Ray Henry studied it by comparing the way the interpreter authors expected it to work and the way that it worked on his Grizzly minimill. The following quoted paragraphs are extracted from his paper which is available in several text formats in the dropbox at <http://www.linuxcnc.org>.

### 20.4.1 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.

A user must understand the correct ways that the g92 values work. They are set based upon the location of each axis when the g92 command is invoked. The NIST document is clear that, "To make the current point 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 likely that the absence of home switches and proper home procedures will result in very large errors in the application of g92 values if they exist in the var file. Many EMC users do not have home switches in place on their machines. For them home should be found by moving each axis to a location and issuing the home command. When each axis is in a known location, the home command will recalculate how the g92 values are applied and will produce consistent results. Without a home sequence, the values are applied to the position of the machine when the EMC begins to run.

### 20.4.2 Setting G92 values

There are at least two ways to set G92 values.

- right mouse click on position displays of tkemc will popup a window into which you can type a value.
- the g92 command

Both of these work from the current location of the axis to which the offset is to be applied.

Issuing g92 x y z a b c does in fact set values to the g92 variables such that each axis takes on the value associated with it's name. These values are assigned to the current position of the machine axis. These results satisfy paragraphs one and two of the NIST document.

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.

### 20.4.3 G92 Cautions

Sometimes the values of a G92 offset get stuck in the VAR file. When this happens reset or a startup will cause them to become active again. The variables are named

```
5211  0.000000
5212  0.000000
5213  0.000000
5214  0.000000
5215  0.000000
5216  0.000000
```

where 5211 is the X axis offset and so on. If you are seeing unexpected positions as the result of a commanded move, or even unexpected numbers in the position displays when you start up, look at these variables in the VAR file and see if they contain values. If they do, set them to zeros and the problems should go away.

With these tests we can see that reset returns g92 to the condition that it had when the interpreter started up. The reader should note that we have established ... that no write of these values occurs during a normal run so if no g92 was set at the startup, none will be read in during a reset.

It may be that this is the heart of the problem that some have experienced with differences between the old and the new interpreter. It may well be, but I leave it to others to test, that the old interpreter and task programs immediately wrote values to the var file and then found those values during a reset.

On the other hand, if G92 values existed in the VAR file when the EMC started up

... starting the EMC with g92 values in the var file is that it will apply the values to 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 moving each axis to a known home position and issuing an axis home command, g92 commands and values work as advertised.

These tests did not study the effect of re-reading the var file while they contain numbers. This could cause problems if g92 offsets had been removed with g92.1 but the var file still contained the previous numbers.

It is this complexity that causes us to say that G92 values must be treated as temporary. They should be used to set global short term offsets. The G54-59.3 coordinate systems should be used whenever long lasting and predictable offsets are needed.

## 20.5 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-.1 y0 z0
g1 f1 z-.25
g3 x-.1 y0 i.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-.1 y0 z0 (center circle)
g1 f1 z-.25
g3 x-.1 y0 i.1 j0
g0 z0
g55 g0 x-.1 y0 z0 (first offset circle)
g1 f1 z-.25
g3 x-.1 y0 i.1 j0
g0 z0
g56 g0 x-.1 y0 z0 (second offset circle)
g1 f1 z-.25
g3 x-.1 y0 i.1 j0
g0 z0
g57 g0 x-.1 y0 z0 (third offset circle)
g1 f1 z-.25
g3 x-.1 y0 i.1 j0
g0 z0
g58 g0 x-.1 y0 z0 (fourth offset circle)
g1 f1 z-.25
g3 x-.1 y0 i.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 down 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 EMC and they will be recalled when you start up again.

# Chapter 21

## Canned Cycles

Canned Cycles G81 through G89 have been implemented for milling. This section describes how each cycle has been implemented. In addition G80 and G98/G99 are considered here because their primary use is related to canned cycles.

All canned cycles are performed with respect to the XY plane. With the current 3 axis interpreter, no A, B, C-axis motion is allowed during canned cycles. Inverse time feed rate is not allowed. Cutter radius compensation is not allowed. Each of the canned cycles defines a new machine motion mode. As a motion mode, they will stay in effect until replaced by another motion G word or by G80 as described below.

All canned cycles use X, Y, R, and Z values in the NC code. These values are used to determine X, Y, R, and Z positions. The R (usually meaning retract) position is along the Z-axis. Some canned cycles use additional arguments that are listed with the specific cycle.

In absolute distance mode, the X, Y, R, and Z values are absolute positions in the current coordinate system. In incremental distance mode, X, Y, and R values are treated as increments to the current position and Z as an increment from the Z-axis position before the move involving Z takes place.

A repeat feature has been implemented. The L word represents the number of repeats. 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. EMC allows L > 1 in absolute distance mode to mean "do the same cycle in the same place several times." Omitting the L value is equivalent to specifying L=1.

When L>1 in incremental mode, the X and Y positions are determined by adding the given X and Y values 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 second and successive go-arounds). The R and Z positions do not change during the repeats.

The number of repeats of a canned cycle only works for in the block containing L word. If you want to repeat a canned cycle using the repeat feature by placing a new L word on each line for which you want repeats.

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 G98/G99 below)

### 21.1 Preliminary Motion

Preliminary motion may be confusing on first read. It should come clear as you work through the examples in G80 and G81 below. Preliminary motion is a set of motions that is common to all

of the milling canned cycles. These motions are computed at the time the canned cycle block is encountered by the interpreter. They move the tool into the proper location for the execution of the canned cycle itself.

These motions will be different depending on whether the canned cycle is to be executed using absolute distances or incremental distances. These motions will also be affected by the initial position of the z axis when the canned cycle block is encountered in a program.

If the current Z position is below the R position, the Z axis is traversed to the R position. This happens only once, regardless of the value of L.

In addition, for each repeat as specified by L, one or two moves are made before the rest of the cycle:  
 1. a straight traverse parallel to the XY-plane to the given XY-position  
 2. a straight traverse of the Z-axis only to the R position, if it is not already at the R position.

## 21.2 G80

G80 turns off all motion. You should think of it as the off position on a rotary switch where the other positions are the different possible motion modes. In the EMC interpreter, G80 is one of the modal codes so any other code will replace it. The result of the following lines of code is the same.

```
N1000 G90 G81 X1 Y1 Z1.5 R2.8 (absolute distance canned cycle)
N1001 G80 (turn off canned cycle motion)
N1002 G0 X0 Y0 Z0 (turn on rapid traverse and move to coordinate home)
```

produces the same final position and machine state as

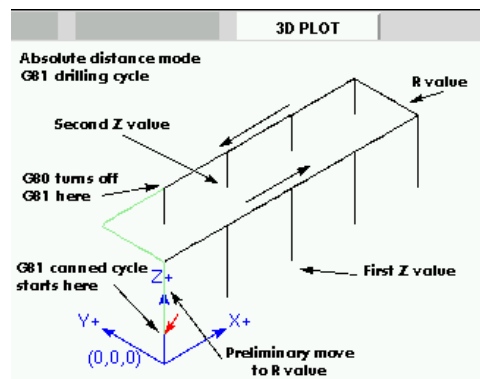
```
N1000 G90 G81 X1 Y1 Z1.5 R2.8 (absolute distance canned cycle)
N1001 G0 X0 Y0 Z0 (turn on rapid traverse and move to coordinate home)
```

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.

Example 1 - Use of a canned cycle as a modal motion code

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. (note the z position change after the first four holes.)

```
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 home moves)
N220 Y0
N220 Z0
N220 M2 (program end)
```



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 example 1 shows, will provide for an easily readable canned cycle. Without it, it is not so obvious that all of the blocks between N120 and N200 belong to the canned cycle.

If you use G80 and do not set another modal motion code soon after, you may get one of the following error messages.

Cannot use axis commands with G80  
Coordinate setting given with G80

These should serve as a reminder that you need to write in a new motion word.

## 21.3 G81 Cycle

The G81 cycle is intended for drilling.

0. Preliminary motion, as described above.

1. Move the Z-axis only at the current feed rate to the Z position.

2. Retract the Z-axis at traverse rate to clear Z. This cycle was used in the description of G80 above but is explained in detail here.

Example 2 - Absolute Position G81

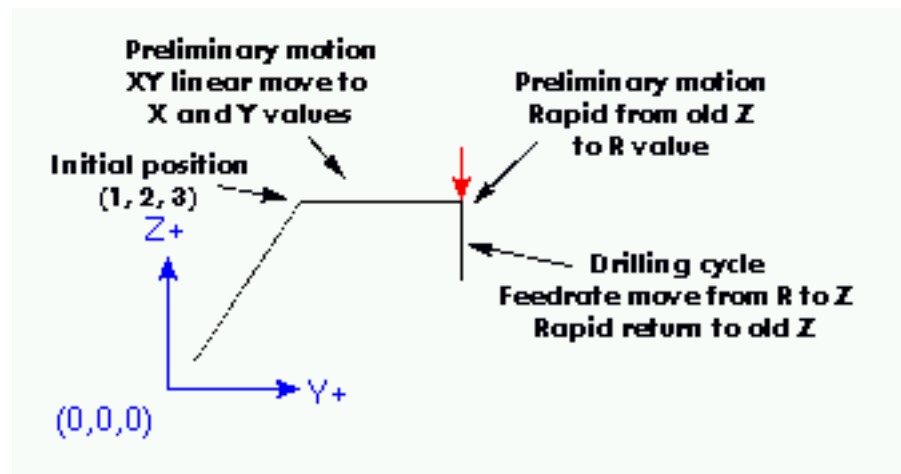
Suppose the current position is (1, 2, 3) and the following line of NC code is interpreted.

G90 G81 G98 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 traverse parallel to the XY plane to (4,5,3)
2. a traverse parallel to the Z-axis to (4,5,2.8).
3. a feed parallel to the Z-axis to (4,5,1.5)
4. a traverse parallel to the Z-axis to (4,5,3)



Example 2 - Absolute Position G81

Suppose the current position is (1, 2, 3) and the following line of NC code is interpreted.

G91 G81 G98 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

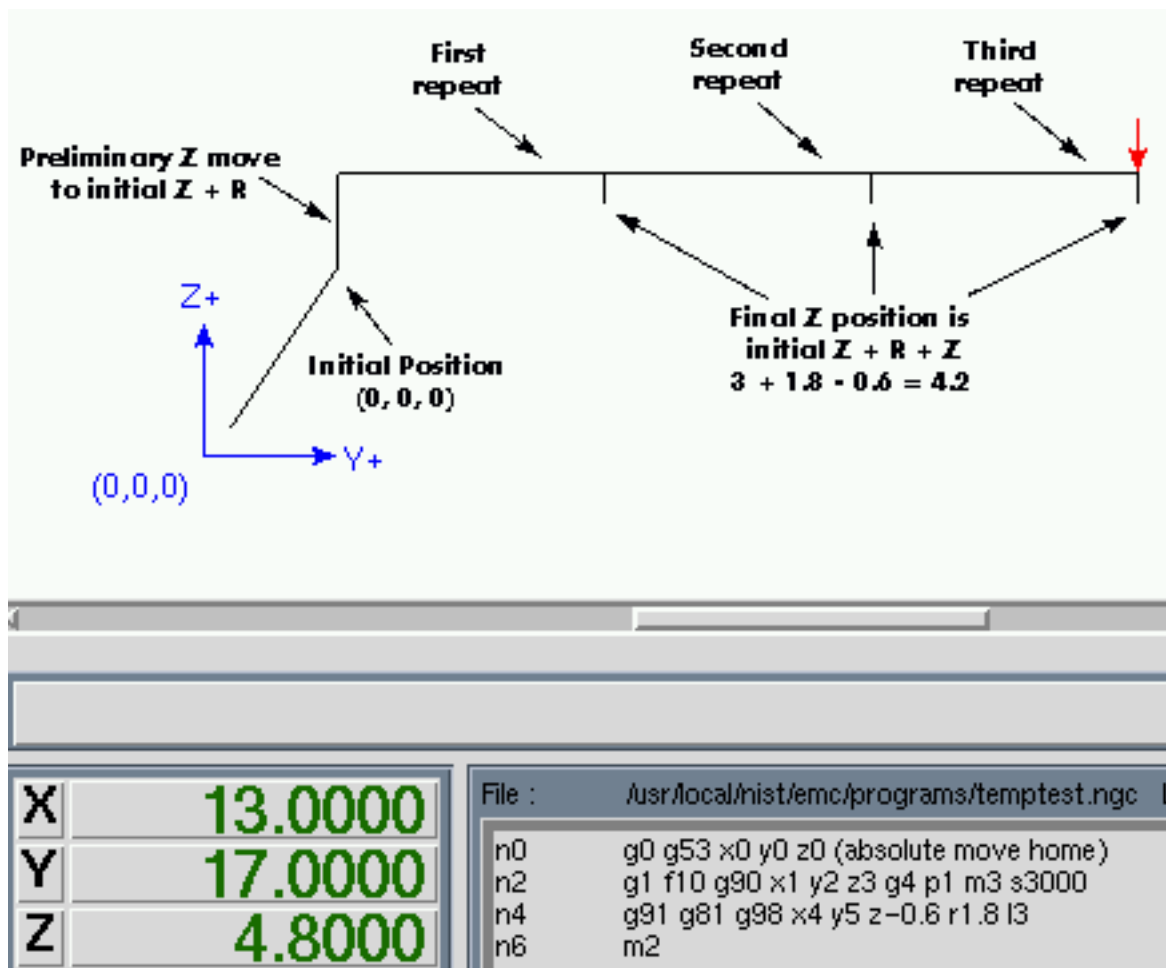


and the R value is 1.8. The initial X position is 5 ( $=1+4$ ), the initial Y position is 7 ( $=2+5$ ), the clear Z position is 4.8 ( $=1.8+3$ ), and the Z position is 4.2 ( $=4.8-0.6$ ). OLD\_Z is 3.

The first preliminary move is a traverse along the Z axis to (1,2,4.8), since  $OLD\_Z < clear\ Z$ .

The first repeat consists of 3 moves.

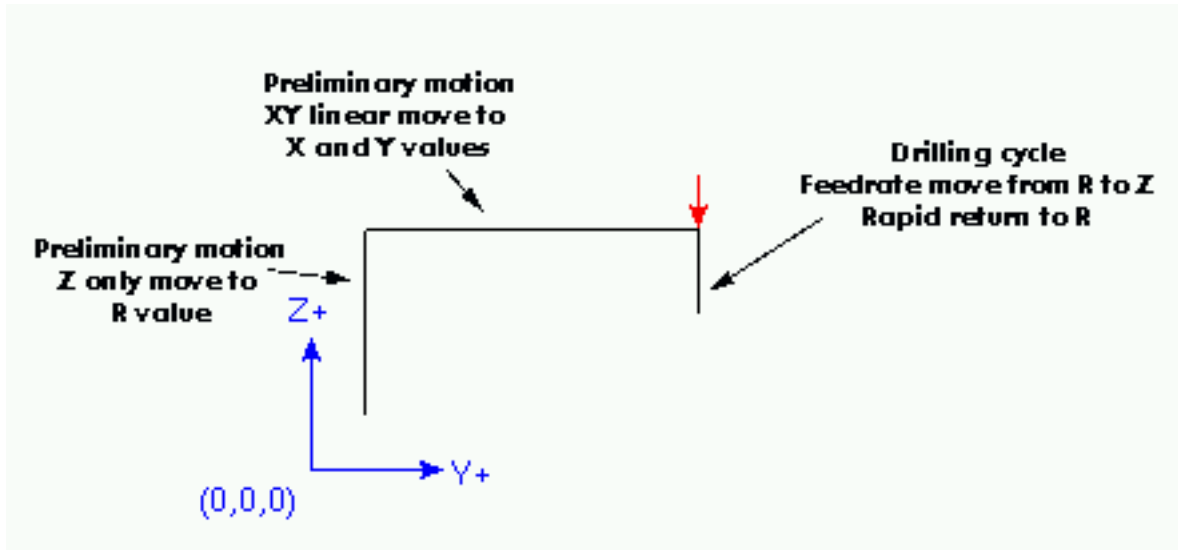
1. a traverse parallel to the XY-plane to (5,7,4.8)
2. a feed parallel to the Z-axis to (5,7, 4.2)
3. a traverse parallel to the Z-axis to (5,7,4.8) The second repeat consists of 3 moves. The X position is reset to 9 ( $=5+4$ ) and the Y position to 12 ( $=7+5$ ).
1. a traverse parallel to the XY-plane to (9,12,4.8)
2. a feed parallel to the Z-axis to (9,12, 4.2)
3. a traverse parallel to the Z-axis to (9,12,4.8) 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 traverse parallel to the XY-plane to (13,17,4.8)
2. a feed parallel to the Z-axis to (13,17, 4.2)
3. a traverse parallel to the Z-axis to (13,17,4.8)



Example 3 - Relative Position G81

Now suppose that you execute the first g81 block of code but from (0, 0, 0) rather than from (1, 2, 3).

G90 G81 G98 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.

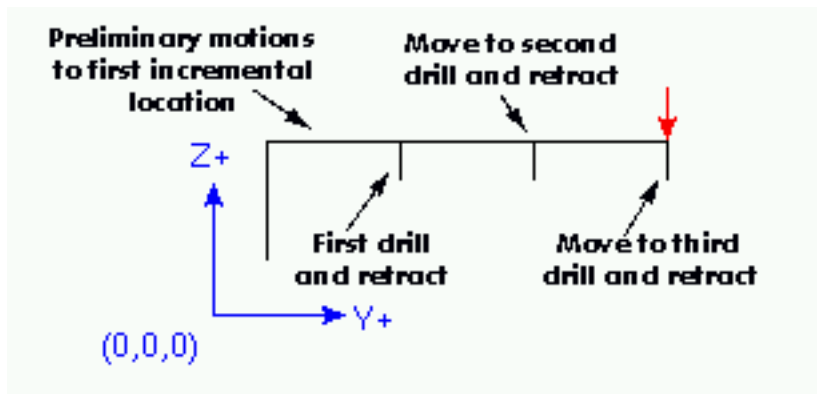


Example 4 - Absolute G81 R > Z

This is a plot of the path of motion for the second g81 block of code.

```
G91 G81 G98 X4 Y5 Z-0.6 R1.8 L3
```

Since this plot starts with (0, 0, 0), the interpreter adds the initial Z 0 and R 1.8 and rapids 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

## 21.4 G82 Cycle

The G82 cycle is intended for drilling.

0. Preliminary motion, as described above.
1. Move the Z-axis only at the current feed rate to the Z position.
2. Dwell for the given number of seconds.
3. Retract the Z-axis at traverse rate 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.

G90 G82 G98 X4 Y5 Z1.5 R2.8 P2

Would be equivalent to example 2 above with a dwell added at the bottom of the hole.

## 21.5 G83 Cycle

The G83 cycle is intended for deep drilling or milling with chip breaking. The dwell in this cycle causes any long stringers (which are common when drilling in aluminum) to be cut off. This cycle takes a Q value which represents a "delta" increment along the Z-axis. Machinists often refer to this as peck drilling.

0. Preliminary motion, as described above.
1. Move the Z-axis only at the current feed rate downward by delta or to the Z position, whichever is less deep.
2. Dwell for 0.25 second.
3. Retract at traverse rate to clear Z
4. Repeat steps 1 - 3 until the Z position is reached.
5. Retract the Z-axis at traverse rate to clear Z.

NIST lists the elements of the command as G83 X- Y- Z- A- B- C- R- L- Q-

I find this command very handy for many of my deep drilling projects. I have not tried to use the L for a repeat so can't say much about that feature. A typical g83 line that I would write might look like G83 X0.285 Y0.00 Z-0.500 R0.2 L1 Q0.05. EMC moves to position X0.285 Y0.00 at the z height before the block. It then pecks its way down to Z-0.500. Each peck pulls the drill tip up to R0.2 after moving Q0.05.

## 21.6 G84 Cycle

The G84 cycle is intended for right-hand tapping.

0. Preliminary motion, as described above.
1. Start speed-feed synchronization.
2. Move the Z-axis only at the current feed rate to the Z position.
3. Stop the spindle.
4. Start the spindle counterclockwise.
5. Retract the Z-axis at the current feed rate to clear Z.
6. If speed-feed synch was not on before the cycle started, stop it.
7. Stop the spindle.
8. Start the spindle clockwise.

## 21.7 G85 Cycle

The G85 cycle is intended for boring or reaming.

0. Preliminary motion, as described above.
1. Move the Z-axis only at the current feed rate to the Z position.
2. Retract the Z-axis at the current feed rate to clear Z. This motion is very similar to g81 except that the tool is retracted from the hole at feedrate rather than rapid.

## 21.8 G86 Cycle

The G86 cycle is intended for boring.

0. Preliminary motion, as described above.
1. Move the Z-axis only at the current feed rate to the Z position.
2. Dwell for the given number of seconds.
3. Stop the spindle turning.
4. Retract the Z-axis at traverse rate to clear Z.
5. Restart the spindle in the direction it was going. This cycle is very similar to g82 except that it stops the spindle before it retracts the tool and restarts the spindle when it reaches the clearance value R.

## 21.9 G87 Cycle

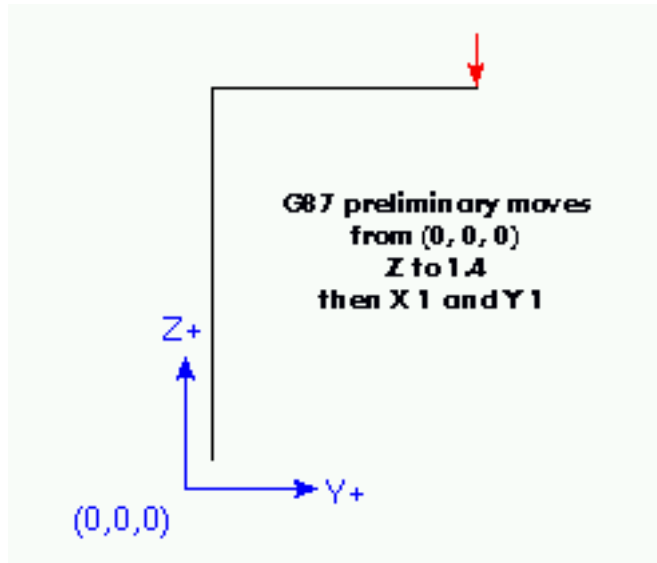
The G87 cycle is intended for back boring.

The situation is that you have a through hole and you want to counter bore the bottom of hole. To do this you put an L-shaped tool in the spindle with a cutting surface on the UPPER side of its base. You stick it carefully through the hole when it is not spinning and is oriented so it fits through the hole, then you move it so the stem of the L is on the axis of the hole, start the spindle, and feed the tool upward to make the counter bore. Then you stop the tool, get it out of the hole, and restart it.

This cycle uses I and J values to indicate the position for inserting and removing the tool. I and J will always be increments from the X position and the Y position, regardless of the distance mode setting. This cycle also uses a K value to specify the position along the Z-axis of the top of counterbore. The K value is an absolute Z-value in absolute distance mode, and an increment (from the Z position) in incremental distance mode.

0. Preliminary motion, as described above.
1. Move at traverse rate parallel to the XY-plane to the point indicated by I and J.
2. Stop the spindle in a specific orientation.
3. Move the Z-axis only at traverse rate downward to the Z position.
4. Move at traverse rate parallel to the XY-plane to the X,Y location.
5. Start the spindle in the direction it was going before.
6. Move the Z-axis only at the given feed rate upward to the position indicated by K.
7. Move the Z-axis only at the given feed rate back down to the Z position.
8. Stop the spindle in the same orientation as before.

9. Move at traverse rate parallel to the XY-plane to the point indicated by I and J.
10. Move the Z-axis only at traverse rate to the clear Z.
11. Move at traverse rate parallel to the XY-plane to the specified X,Y location.
12. Restart the spindle in the direction it was going before.



Example 6 - Backbore

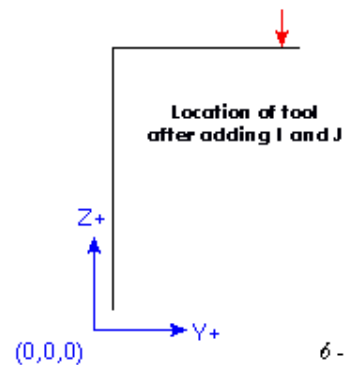
Example six uses a incremental distances from (0, 0, 0) so the preliminary moves look much like those in example five but they are done using the G87 backbore canned cycle.

```
G91 G87 M3 S1000 X1 Y1 Z-0.4 R1.4 I-0.1 J-0.1 K-0.1
```

You will notice that the preliminary moves shift the tool to directly above the center axis of the existing bore.

Next it increments that location by the I and J values. I offsets X with a plus value being added to the current X. J does the same for the Y axis.

For our example block both I and J are negative so they move back from the hole axis along the path just made by the tool. The amount of offset required should be just enough that the tool tip will slide down through the bore.

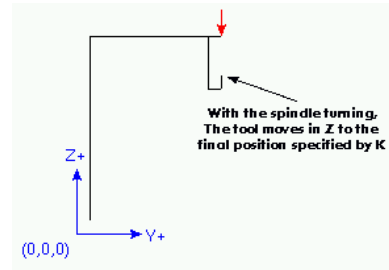


Next the car  
bottom loca  
it moves the

6-1

Now the g87 canned cycle turns the spindle on and moves back up into the bore at the programmed feedrate. This is the real cutting action of this canned cycle. With the proper tool in a boring bar this cycle will produce a chamfer on the bottom side of the bore. G87 can also be used to produce a larger diameter bore on the bottom side of the bore.

When the tool has reached the K position it is returned to the bottom location, the spindle is stopped and oriented and follows the earlier path back out of the bore to the initial position above.



This canned cycle assumes spindle orientation which has not been implemented in the EMC to date. The proper alignment of the tool tip to the oriented spindle is critical to the successful insertion of the tool through the hole to be backbored.

## 21.10 G88 Cycle

The G88 cycle is intended for boring. This cycle uses a P value, where P specifies the number of seconds to dwell.

0. Preliminary motion, as described above.
1. Move the Z-axis only at the current feed rate to the Z position.
2. Dwell for the given number of seconds.
3. Stop the spindle turning.
4. Stop the program so the operator can retract the spindle manually.
5. Restart the spindle in the direction it was going. It is unclear how the operator is to manually move the tool because a change to manual mode resets the program to the top. We will attempt to clarify that step in this procedure.

## 21.11 G89 Cycle

The G89 cycle is intended for boring. This cycle uses a P value, where P specifies the number of seconds to dwell.

0. Preliminary motion, as described above.
1. Move the Z-axis only at the current feed rate to the Z position.
2. Dwell for the given number of seconds.
3. Retract the Z-axis at the current feed rate to clear Z. This cycle is like G82 except that the tool is drawn back at feedrate rather than rapid.

## 21.12 G98 G99

G98 - initial level return in canned cycles

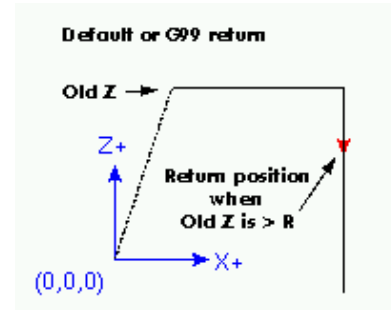
G99 - R value return in canned cycles

These codes are treated together because they behave very much alike. You will recall that when Z is above R the preparatory move is from the current location to the X, Y values. If G98 is not specified, then the canned cycle will return to the R value rather than the Z value that was used on the approach.

N01 G0 X1 Y2 Z3

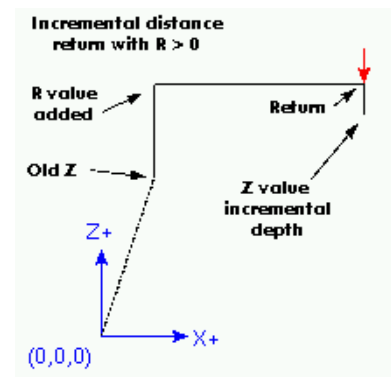
N02 G90 G81 X4 Y5 Z-0.6 R1.8

Adding G98 to the second line above means that the return move will be to the value of OLD\_Z since it is higher than the R value specified.



Neither code will have any affect when incremental moves with a positive R value are specified because the R value is added to OLD\_Z and that result is used as the initial level for a G98. The same value is the computed R value so G99 will also return to the same place.

When the value of R is less than OLD\_Z and incremental distance mode is turned on, G98 will return the tool to the value of OLD\_Z. Under those conditions G99 will retract the tool to OLD\_Z plus the negative R value. The return will be below OLD\_Z.



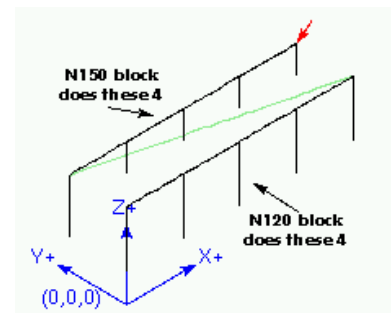
### 21.13 Why use a canned cycle?

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.

Example 1 above demonstrated how a canned cycle could be used to produce 8 holes with ten lines of nc 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.

#### Example 7 - Eight Holes Revisited

```
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-.5 r1 l4(canned drill cycle)
n160 g80 (turn off canned cycle)
n170 m2 (program end)
```



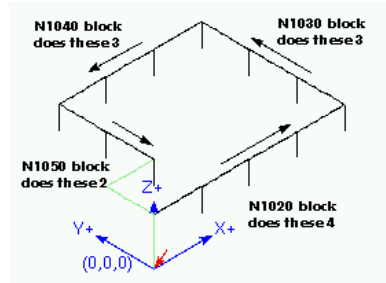
### Example 8 - Twelve holes in a square

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 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.



## Appendix A

# Glossary of Common Terms Used in the EMC Documents

GPLD Copyright 2003, LinuxCNC.org

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 [A](#) that uses an acme thread form. Acme threads have somewhat lower friction and wear than simple triangular threads, but ball-screws [A](#) are lower yet. Most manual machine tools use acme lead-screws.

**Axis** One of the computer control 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. Additional linear axes parallel to X, Y, and Z are called U, V, and W respectively. Angular axes like rotary tables are referred to as A, B, and C.

**Backlash** The amount of "play" or lost motion that occurs when direction is reversed in a lead screw [A](#), 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 [A](#) and screw to reduce friction. Ball-screws have very low friction and backlash [A](#), but are usually quite expensive.

**Ball Nut** A special nut designed for use with a ball-screw. It contains an internal passage to recirculate the balls from one end of the screw to the other.

**bridgeportio** An I/O task [A](#) designed to work with a Bridgeport milling machine, having a variable speed spindle [A](#), coolant, and lube pumps, and some other stuff.

**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 [A](#).

**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.

**DRO** A Digital Read Out is a device attached to the slides of a machine tool or other device which has parts that move in a precise manner to indicate the current location of the tool with respect to some reference position. Nearly all DRO's use linear quadrature encoders to pick up position information from the machine.

**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 corners with a radius only slightly larger than the radius on the corner of the convex EDM electrode.

**EMC** The Enhanced Machine Controller. Initially a NIST [A](#) project. EMC is able to run a wide range of motion devices.

**EMCIO** The module within EMC [A](#) that handles general purpose I/O, unrelated to the actual motion of the axes. A couple examples are bridgeportio [A](#) and minimillio [A](#).

**EMCMOT** The module within EMC [A](#) that handles the actual motion of the cutting tool. It runs as a real-time program and directly controls the motors.

### **Encoder**

**Feed** Relatively slow, controlled motion of the tool used when making a cut.

**Feedrate** The speed at which a motion occurs. In manual mode, jog speed can be set from the graphical interface. In auto or mdi mode feedrate is commanded using a (f) word. F10 would mean ten units per minute.

### **Feedback**

**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".

**G-Code** The generic term used to refer to the most common part programming language. There are several dialects of G-code, EMC uses RS274/NGC [A](#).

**GUI** Graphical User Interface.

**General** A type of interface that allows communications between a computer and human (in most cases) via the manipulation of icons and other elements (widgets) on a computer screen.

**EMC** An application that presents a graphical screen to the machine operator allowing manipulation of machine and the corresponding controlling program.

**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 EMC [A](#) for a particular machine

**Joint Coordinates:** These specify the angles between the individual joints of the machine. 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.

### kernel-space

**Kinematics** The position relationship between world coordinates **A** and joint coordinates **A** 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 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 **A** or acme screws **A**, although conventional triangular threaded screws may be used where accuracy and long life are not as important as low cost.

**MDI** Manual Data Input. This is a mode of operation where the controller executes single lines of G-code **A** as they are typed by the operator.

**minimillio** An I/O task **A** designed to work with small table-top mills.

**NIST** National Institute of Standards and Technology. An agency of the Department of Commerce in the United States.

### Offsets

**Part Program** A description of a part, in a language that the controller can understand. For EMC, that language is RS-274/NGC, commonly known as G-code **A**.

**Rapid** Fast, possibly less precise motion of the tool, commonly used to move between cuts. If the tool meets the material during a rapid, it is probably a bad thing!

**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 RTAI **A** or RTLINUX **A** and build the software to run in those special environments. For this reason real-time software runs in kernel-space.

**RTAI** Real Time Application Interface, see <http://www.aero.polimi.it/~rtai/>, one of two real-time extensions for Linux that EMC can use to achieve real-time **A** performance.

**RTLINUX** See <http://www.rtlinux.org> <http://www.rtlinux.org>, one of two real-time extensions for Linux that EMC can use to achieve real-time **A** performance.

**RS-274/NGC** The formal name for the language used by EMC **A** part programs **A**.

### Servo Motor

#### Servo Loop

**Spindle** On a mill or drill, the spindle holds the cutting tool. On a lathe, the spindle holds the workpiece.

**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 EMC **A** that coordinates the overall execution and interprets the part program.

**Tcl/Tk** A scripting language and graphical widget toolkit with which EMC's most popular GUI's **A** were written.

**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.

# Appendix A

## Legal Section

### Handbook Copyright Terms

Copyright (c) 2000 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 one Back-Cover Text: "This EMC Handbook is the product of several authors writing for linuxCNC.org. As you find it to be of value in your work, we invite you to contribute to its revision and growth." A copy of the license is included in the section entitled "GNU Free Documentation License". If you do not find the license you may order a copy from Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307

### A.1 GNU Free Documentation License Version 1.1, March 2000

Copyright (C) 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.

#### A.1.1 GNU Free Documentation License Version 1.1, March 2000

Copyright (C) 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. Secondly, 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.



# Index

encoder, [71](#)  
ESTOP, [65](#)

G-code, [8](#)  
G53, [148](#)  
G54, [150](#)  
G55, [148](#), [150](#)  
G56, [150](#)  
G57, [150](#)  
G58, [150](#)  
G59, [150](#)  
G80, [155](#)  
G81, [156](#)  
G82, [158](#), [159](#)  
G84, [159](#)  
G87, [160](#)  
G88, [162](#)  
G89, [162](#)  
G98, [162](#)  
G99, [162](#)

HAL, [63](#)

MachineOn, [65](#)