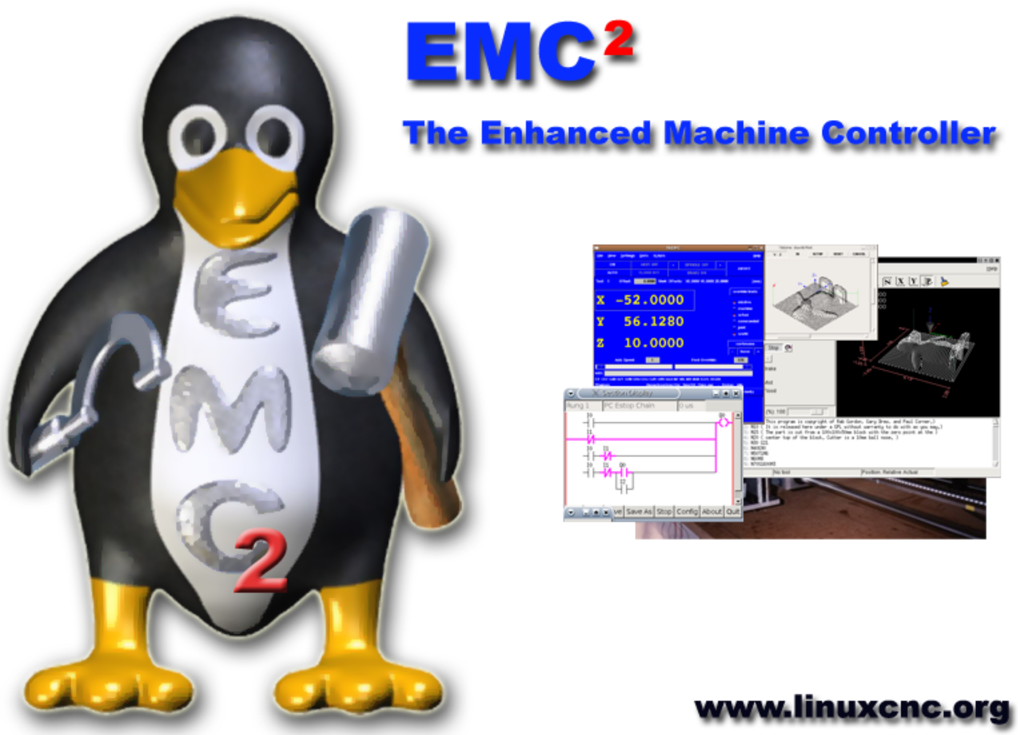


V2.2 Integrators Handbook

April 13, 2008

The EMC Team

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

Copyright (c) 2000-7 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

Part I

Introduction

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 one of the standard ways (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 another 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 CNC 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[1] 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 an effect on 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.

[1] 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 it currently exists does not directly take in CAD or any image and run a machine using it. The EMC2 uses a variant of the earlier CNC language named RS274NGC. (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 is 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, see section 2.3

The EMC2 will not run under a 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 EMC 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 EMC 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 EMC 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 EMC 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 EMC 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 newsgroups. 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 http://cvs.linuxcnc.org. They are available for anyone to inspect or download. The EMC2 source code (with a few exceptions[2]) is released under the GNU General Public License (GPL). The GPL controls the terms under which EMC2 can be changed and distributed. This is done in order to protect the rights of people like you to use, study, adapt, improve, and redistribute it freely, now and in the future. To read about your rights as a user of EMC2, and the terms under which you are allowed to distribute any modifications you may make, see the full GPL at http://www.gnu.org/copyleft/gpl.html.

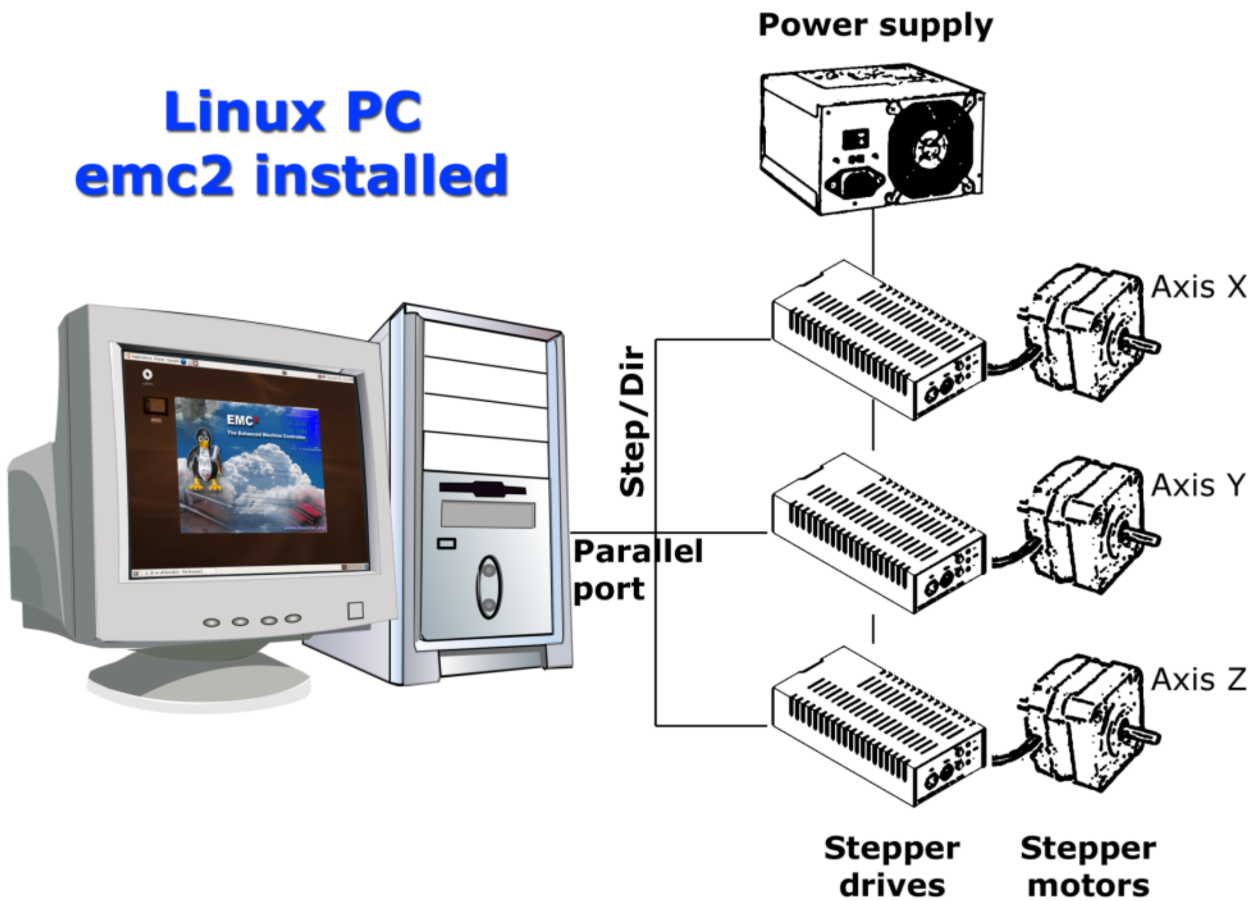
1.5 How EMC2 Works

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 system we will remind the reader of this typical machine.

[2] some parts of EMC2 are released under the "Lesser" GPL (LPGL), which allows them to be used with proprietary software as long as certain restrictions are observed.

Figure 1.1: Simple EMC2 Controlled 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 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:

- a character-based screen graphics program named keystick 1.3
- an X Windows programs named xemc 1.6
- two Tcl/Tk-based GUIs named tkemc 1.5 and mini 1.4.
- an OpenGL-based GUI, with an interactive G-Code previewer, called AXIS 1.2

Tkemc and Mini 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

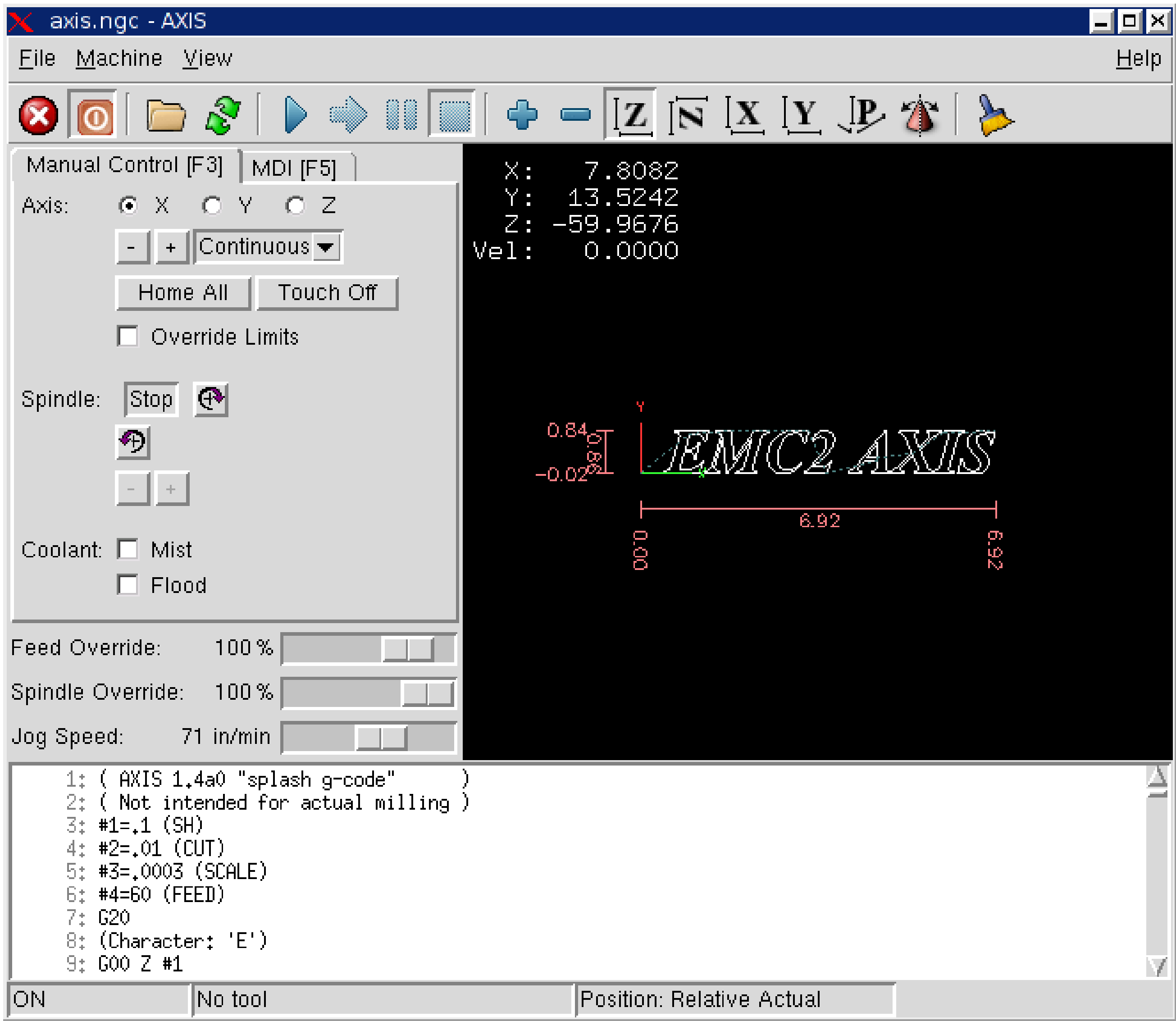


Figure 1.2: The AXIS Graphical Interface

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.

1.5.2 Motion Controller EMCMOT

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 their commanded position. The motion controller includes programmable software limits, and 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.

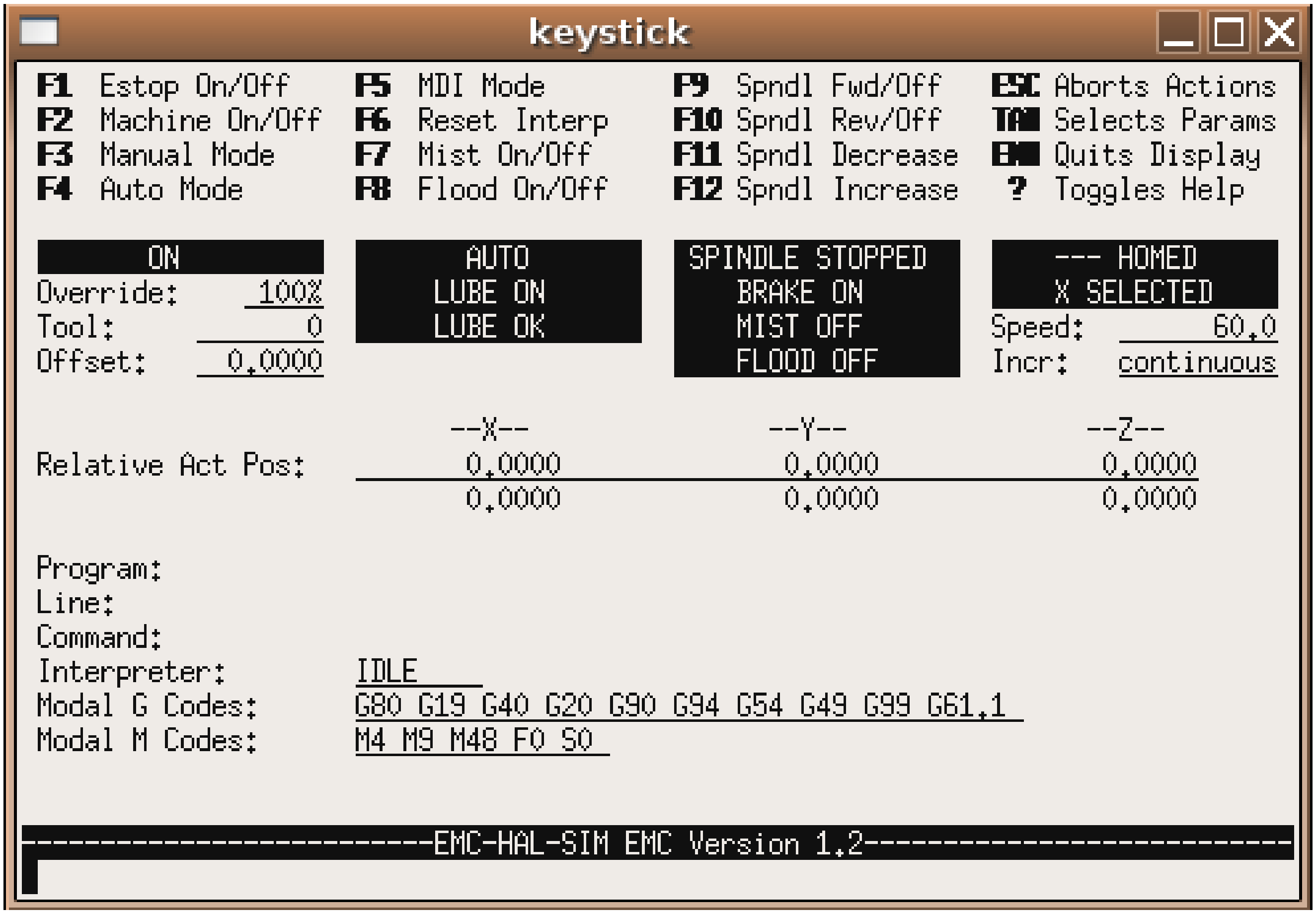


Figure 1.3: The Keystick interface

Complex kinematics for robots can be coded in C according to a prescribed interface to replace the default 3-axis Cartesian machine kinematics routines.

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 I/O subsystem has become very powerful and flexible. EMC2 contains a Programmable Logic Controller module (behaves just like a hardware PLC) that 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 pins (more on this later on), so you can use only the subset that fits your hardware and is necessary for your application.

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 feed rate, 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.

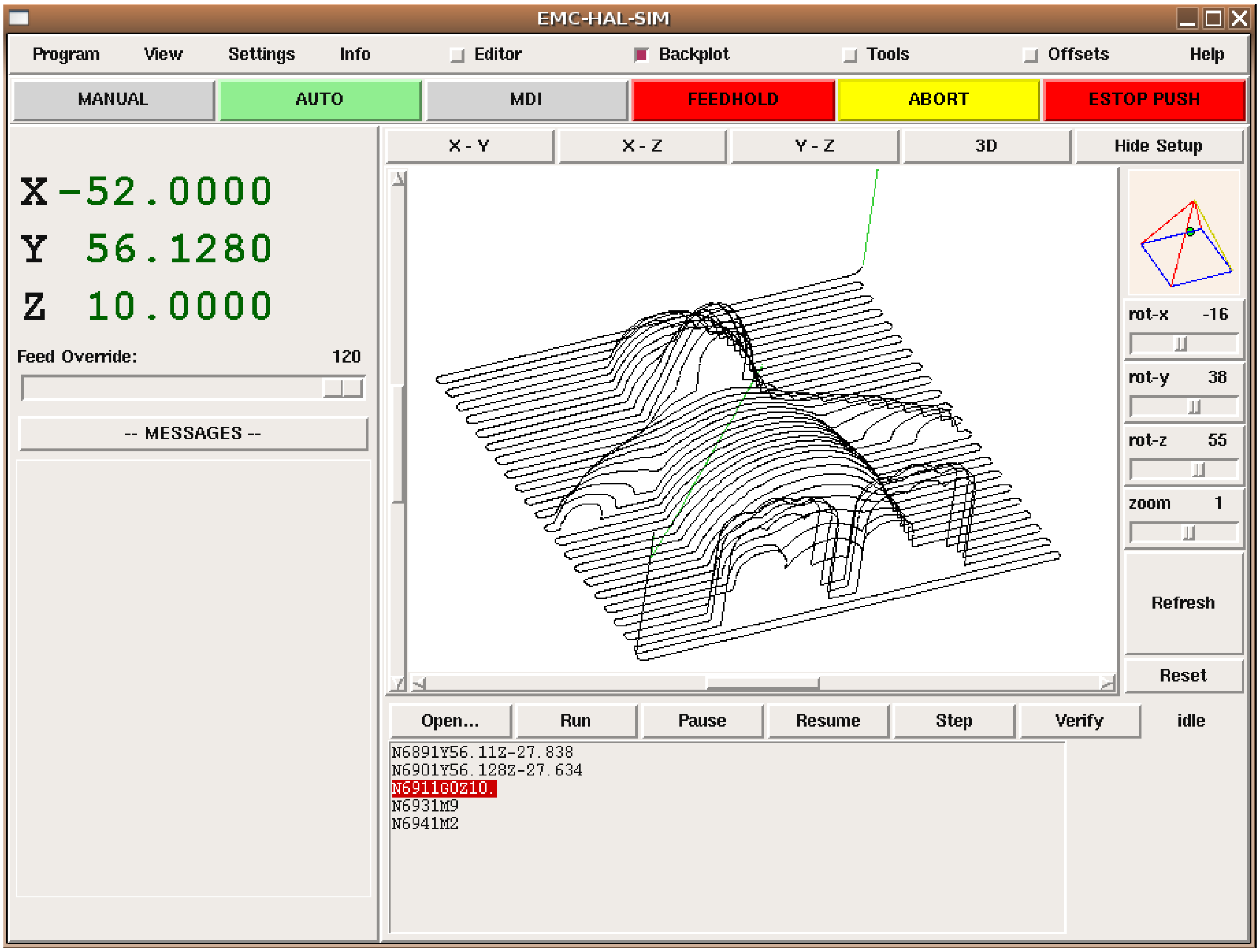


Figure 1.4: The Mini Graphical Interface

```
G1 F3 X1.000
```

G1 means that the machine is supposed to run at a programmed feed rate rather than at the fastest speed that it can (G0 is the way to command a rapid move like you would make above the work when not cutting). 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 (assuming that the X axis started at zero) means the machine should move one inch in the positive X 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,

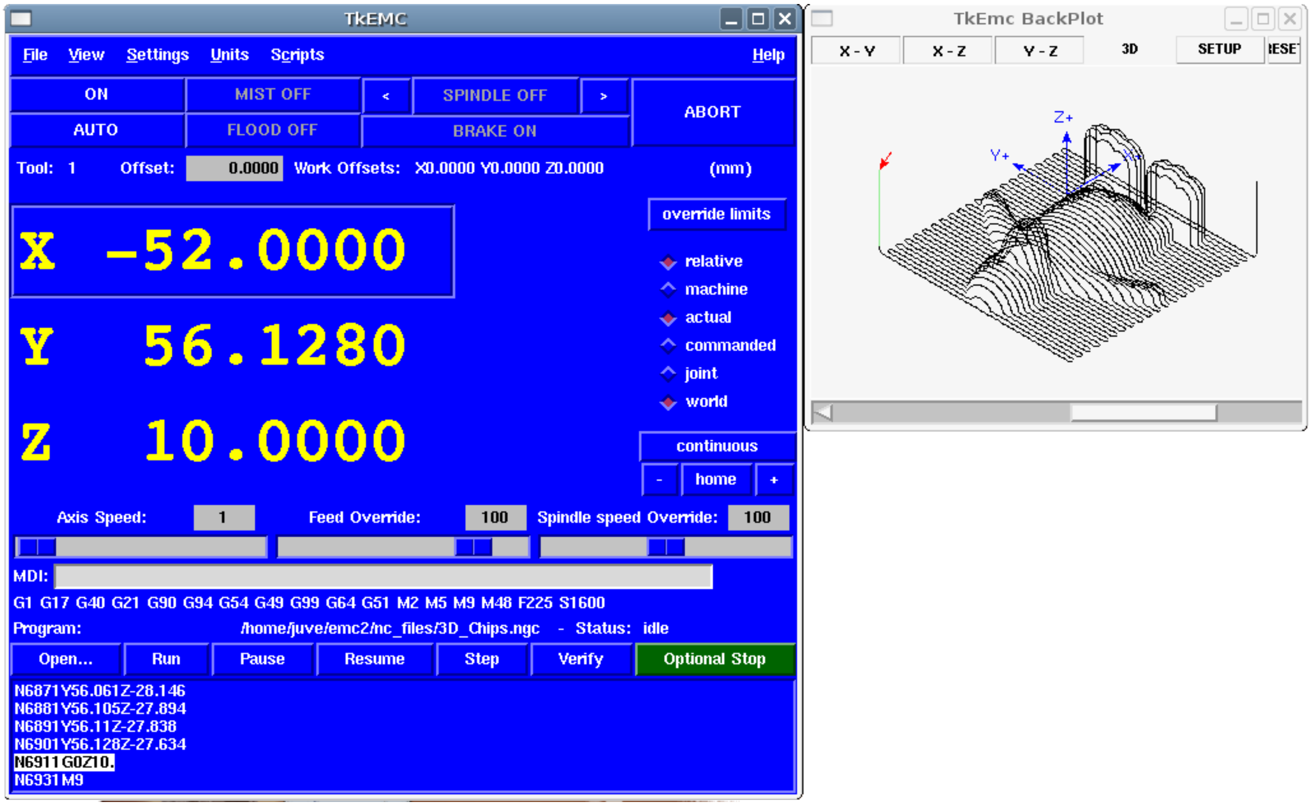


Figure 1.5: The TkEmc Graphical Interface

gouged vices, and scars are the evidence of lessons taught. Good part finish, close tolerances, and careful work are the evidence of lessons learned. No machine, no computer program, can take the place of human experience.

As you begin to work with the 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 separately. In human terms a manual command might be “turn on coolant” or “jog X at 25 inches per minute.” These are roughly equivalent to flipping a switch or turning the 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.

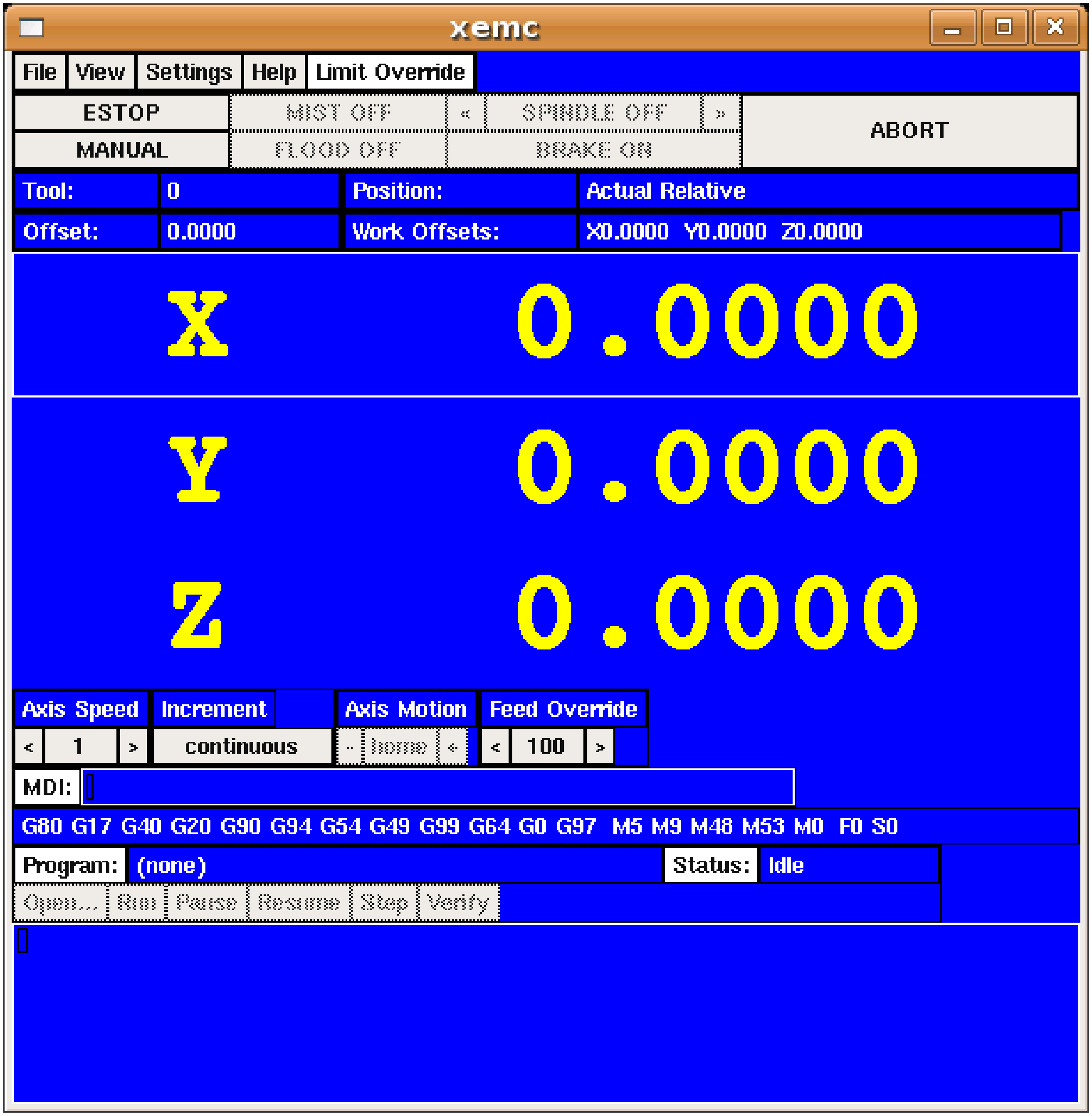
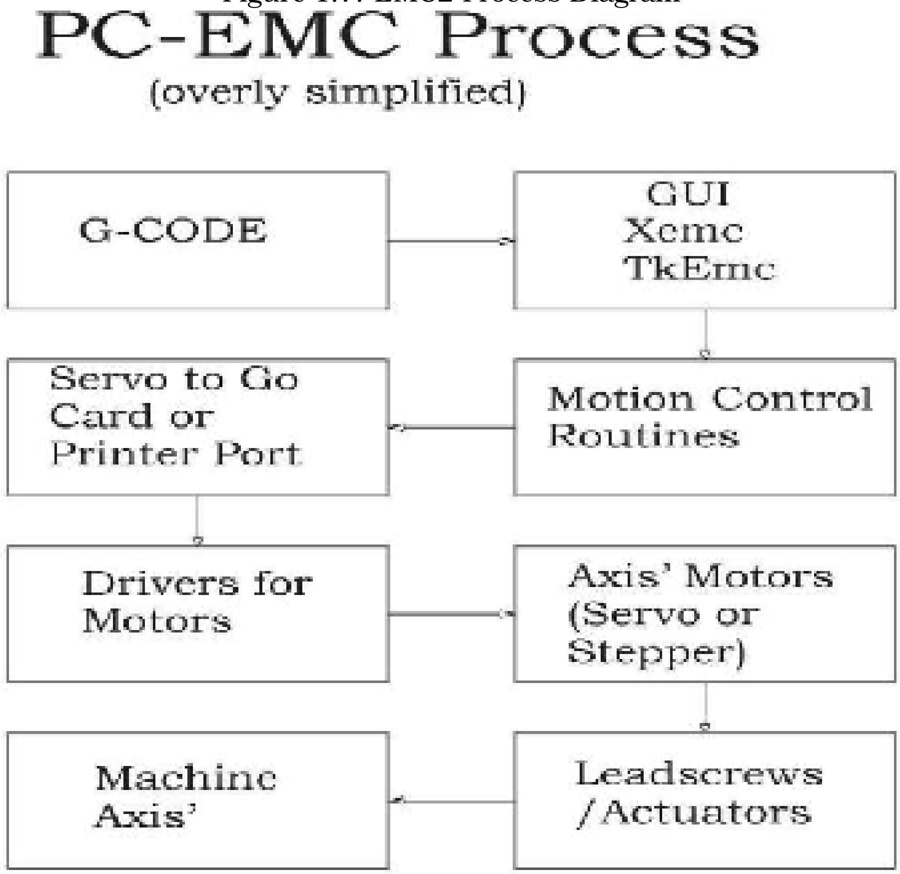


Figure 1.6: The XEMC Graphical Interface

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

The AXIS user interface removes some of the distinctions between Auto and the other modes by making Auto-commands available at most times. It also blurs the distinction between Manual and MDI because some Manual commands like Touch Off are actually implemented by sending MDI commands.

Figure 1.7: EMC2 Process Diagram



1.6.2 Information Display

While an EMC2 is running, each of the modules keeps up a conversation with the others 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 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 count.

It is also important to see any messages or error codes sent by the EMC2. These are used to request the operator change a tool, to describe problems in G-code programs, or to tell why the machine stopped running.

As you work your way through this text, you will be learning, bit by bit, how to set up 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. You might ask, "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's units to some value using an INI variable that looks like this.

```
UNITS = inch
```

or

```
UNITS = mm
```

After we have decided upon a value for the units for an axis, we tell the EMC2 how may 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.

Part II

Installing

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[1].

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. 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[2].

[1] "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

[2] 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.

2.3 EMC2 Live CD

The EMC2 team now has a custom Live-CD based on Ubuntu 6.06 that will let you try out EMC2 before installing, and it's also the easiest way to install Ubuntu and EMC2 together.

Just download the ISO http://linuxcnc.org/iso/emc2-ubuntu6.06-desktop-i386.iso (EU Mirror http://dsplabs.utt.ro/~juve/emc/) and burn it to a CD. (The MD5SUM of the CD is to be determined)

When you boot the CD on your machine, you can see and experiment with the exact environment and EMC2 software that you will have if you choose to install it.

If you like what you see, just click the Install icon on the desktop, answer a few questions (your name, timezone, password) and the install completes in a few minutes.

This install gives you all the benefits of the community-supported Ubuntu distribution as well as being automatically configured for EMC2. As new Ubuntu updates or EMC2 releases are made, the Update manager will let you know and allow you to easily upgrade.

2.4 EMC2 install script

We also provide a simple script to install emc2 on Ubuntu for users with an existing installation of Ubuntu. It runs the commands explained in 2.5.

To use it you need to :

- Download the script from http://linuxcnc.org/dapper/emc2-install.sh (For Ubuntu 6.06)
- 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 > CNC 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.

2.5 Manual installing using apt commands.

The following few section will describe how to install EMC2 on Ubuntu 6.06 “Dapper Dreake” 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 2.4.

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

```
$ sudo sh -c 'echo "deb http://www.linuxcnc.org/emc2/ dapper emc2.2" >>/etc/apt/sources.list;'
$ sudo sh -c 'echo "deb-src http://www.linuxcnc.org/emc2/ dapper emc2.2" >>/etc/apt/sources.list'
```

Then update & get emc2.

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

This command will install the emc2 package along with all dependencies[3].

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 -
```

[3]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.

Chapter 3

Compiling EMC2 from source

3.1 Introduction

The third hurdle that you face when you begin to set up the EMC2 is getting and installing the EMC2 software itself. All of EMC2 has been placed on cvs.linuxcnc.org in a concurrent versioning (CVS) repository. EMC2 is also available as a precompiled package (for various platforms) for download from that site.

Installation can be a daunting task to people new to Linux. The hardest part is getting the Real Time Linux patch up and running. After that, installing EMC is pretty easy. With that said, we recently provided a completely new experience for users, they only need to install Ubuntu (a very friendly linux distribution), then run a single install script, and they already should have the Real Time part and EMC2 working. Information how to access this can be found on the www.linuxcnc.org page under Download.

3.2 EMC Download Page

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

3.3 EMC2 Release Description

EMC2 will be using a release model similar to (but simpler than) the one used by Debian. At any one time there will be three versions of EMC2. Debian uses "stable", "testing", and "unstable". We will be using "Released", "Testing", and "Head". For the latest information, click on the version you are interested in.

Released is exactly that, a released version of EMC2 with a version number. It is tested by both developers and beta users before being released, and is suitable for the average user. Most developers and IRC/mailing list regulars are able to help support people running a released version. **"Released"** is available in several forms, including .debs for Ubuntu and source tarballs for local compilation. There will be a debian repository which will always have the latest released version (and thus allows for easy upgrades from one stable release to the next).

Testing is a version of EMC2 that is ready for "beta testing" but not for general release. Before a version is labeled **testing** it will be known to compile and run on several different platforms, but there will probably be various limitations and known problems. The **Testing** wiki page will attempt to list known problems and workarounds, but there will probably also be undiscovered bugs. Since **Testing** is "beta" software, it should not be used for anything critical. Users of **Testing** need to understand that it is beta software, and must be willing to give detailed bug reports if things go wrong. **Testing** is available primarily as a tag in CVS, although for convenience of testers, a "testing" debian repository and/or tarballs may also be available. The EMC Board of Directors will decide when "Testing" is worthy of becoming "Released". This is a formal decision, made by motion and voting on the board mailing list or board IRC channel.

TRUNK is a CVS term for where all the primary development takes place. **TRUNK** can be broken at any time. When **TRUNK** reaches a state that is deemed worthy of testing by a larger number of people, the **"Testing"** tag will be moved. This is an informal decision, made by concensus of lead developers, usually on IRC. Development will immediately continue, and **TRUNK** will once again diverge from **Testing**. **TRUNK** has no "version number", and on a busy weekend it can literally change every 10 minutes.

3.4 Download and source preparation.

The following few section will describe how to get EMC2, and compile it.

To download, simply go to www.linuxcnc.org to the Download page, and get the latest release or testing tarball.

Once you have it, extract it to your home folder:

```
$ cd ~/
$ tar xzvf emc2-version.tar.gz
```

Next you'll need to decide what kind of install you want. There are two ways to try EMC2 out:

Installed Like most other software on Linux, the files are placed in system directories, and is automatically available to all users of that computer.[1]

Run-in-place All the files for EMC2 are kept inside the `emc2` directory. This is useful for trying out EMC2, especially when there is another version of EMC2 already installed on the system.

3.4.1 Downloading the CVS version

If you wish to use the TRUNK version of emc2, please follow the instructions on our wiki to obtain the source code: http://wiki.linuxcnc.org/cgi-bin/emcinfo.pl?CVS

3.5 Installed

EMC2 followes the standard way of compiling linux software. To compile it simply go to the sources folder:

```
$ cd ~/emc2/src
```

and issue these commands:

```
$ ./configure
$ make && sudo make install
```

To run it simply type 'emc'.

[1] The pre-built packages for Ubuntu Linux use the “installed” method

3.6 Run-in-place

If you want only to test the software before installing it, or if you're worried about overwriting an existing installation, there is a Run-In-Place (RIP) mode which you can try out. In this mode, there is no installation step, and no files are placed outside the top directory , `~/emc2` in this example.

```
$ cd ~/emc2/src
```

and issue these commands:

```
$ ./configure --enable-run-in-place
$ make && sudo make setuid
```

In a shell session where you want to use the run-in-place version of emc[2], execute

```
$ . ~/emc2/scripts/emc-environment
```

Until you close that terminal, it will be set up so that the programs and manual pages from the Run-In-Place directory are available without referring to the path each time. After that you can run EMC2 by issuing:

```
$ emc
```

3.7 Simulator

To install EMC2 on a system without a realtime kernel, add `--enable-simulator` to the `configure` commandline. In this mode, EMC2 runs as a purely userspace program. No hardware can be controlled and realtime scheduling is not guaranteed, but the other features of HAL, EMC and its various user interfaces are available. When using `--enable-run-in-place`, the `sudo make setuid` step is unneeded.

3.8 Editing and Recompiling

You may need to recompile the EMC2 code for a number of reasons. You may have modified the source code, or you may have downloaded just a few new files. To recompile, do the following:

```
$ cd ~/emc2/src
$ make && sudo make install # for run-installed
$ make && sudo make setuid  # for run-in-place
$ make                      # for run-in-place, simulator
```

The build process is smart enough to only rebuild things that are affected by your changes.

[2]By putting this command in a shell start-up script, such as `~/.bash_profile`, you do not need to manually run it in each terminal window.

Part III

EMC Configuration

Chapter 4

INI Configuration

4.1 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.[1] 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 ("parameters") for use by the interpreter. These values are saved from one run to another. See section **??**.

TBL The tbl file saves tool information. See section **??**.

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.

.emcrc This file saves user specific information and is created to save the name of the directory when the user first selects an EMC configuration.[2]

Items marked **(HAL)** are used only by the sample HAL files and are suggested as a good convention. Other items are used by EMC directly, and must always have the section and item names given.

4.2 The INI File Layout

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

- comments.

[1] 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.

[2] Usually this file is in the users home directory (e.g. /home/user/)

- sections,
- variables.

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

4.2.1 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.

Note that inside a variable, the “#” and “;” characters do not denote comments:

```
INCORRECT = value     # and a comment
```

4.2.2 Sections

Related parts of an ini file are separated into sections. A section line looks like [THIS_SECTION]. The name of the section is enclosed in brackets. The order of sections is unimportant. The following sections are used by emc:

- [EMC] general information (4.3.1)
- [DISPLAY] settings related to the graphical user interface (4.3.2)
- [RS274NGC] settings used by the g-code interpreter
- [EMCMOT] settings used by the realtime motion controller (4.3.3)
- [HAL] specifies .hal files (4.3.5)
- [TASK] settings used by the task controller (4.3.4)
- [TRAJ] additional settings used by the realtime motion controller (4.3.6)
- [AXIS_0] ... [AXIS_n] individual axis variables (4.3.7)
- [EMCIO] settings used by the I/O Controller (4.3.8)

4.2.3 Variables

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.

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

Some of the variables are used by EMC, and must always use the section names and variable names shown. Other variables are used only by HAL, and the section names and variable names shown are those used in the sample configuration files.

4.2.4 Definitions

Machine Units The units (of length or angle) specified in the inifile for a particular axis

4.3 INI Variable Definitions

4.3.1 [EMC] Section

VERSION = $Revision: 1.3 $ The version number for the INI file. The value shown here looks odd because it is automatically updated when using the Revision Control System. It's a good idea to change this number each time you revise your file. If you want to edit this manually just change the number and leave the other tags alone.

MACHINE = My Controller This is the name of the controller, which is printed out at the top of most graphical interfaces. You can put whatever you want here as long as you make it a single line long.

RS274NGC_STARTUP_CODE = G21 G90 A string of NC codes that the interpreter is initialized with. This is not a substitute for specifying modal g-codes at the top of each ngc file, beacause the modal codes of machines differ, and may be changed by g-code interpreted earlier in the session.

4.3.2 [DISPLAY] Section

Different user interface programs use different options, and not every option is supported by every user interface.

DISPLAY = tkemc The name of the user interface to use. Valid options may include:

- axis
- keystick
- mini
- tkemc
- xemc

POSITION_OFFSET = RELATIVE The coordinate system (RELATIVE or MACHINE) to show when the user interface starts. The RELATIVE coordinate system reflects the G92 and G5x coordinate offsets currently in effect.

POSITION_FEEDBACK = ACTUAL The coordinate value (COMMANDED or ACTUAL) to show when the user interface starts. The COMMANDED position is the ideal position requested by emc. The ACTUAL position is the feedback position of the motors.

MAX_FEED_OVERRIDE = 1.2 The maximum feed override the user may select. 1.2 means 120% of the programmed feed rate

MIN_SPINDLE_OVERRIDE = 0.5 The minimum spindle override the user may select. 0.5 means 50% of the programmed spindle speed. (This is useful as it's dangerous to run a program with a too low spindle speed).

MAX_SPINDLE_OVERRIDE = 1.0 The maximum spindle override the user may select. 1.0 means 100% of the programmed spindle speed

DEFAULT_LINEAR_VELOCITY = .25 The default velocity for linear jogs, in machine units per second. Only in the AXIS user interface.

MAX_LINEAR_VELOCITY = 1.0 The maximum velocity for linear jogs, in machine units per second. Only in the AXIS user interface.

DEFAULT_ANGULAR_VELOCITY = .25 The default velocity for angular jogs, in machine units per second. Only in the AXIS user interface.

MAX_ANGULAR_VELOCITY = 1.0 The maximum velocity for angular jogs, in machine units per second. Only in the AXIS user interface.

PROGRAM_PREFIX = ~/emc2/nc_files The default location for g-code files and the location for user-defined M-codes

INCREMENTS = 1 mm, .5 mm, . . . Defines the increments available for incremental jogs. See section **??** for more information. Only in the AXIS user interface

INTRO_GRAPHIC = emc2.gif The image shown on the splash screen

INTRO_TIME = 5 The maximum time to show the splash screen

OPEN_FILE = /full/path/to/file.ngc The file to show in the preview plot when AXIS starts

4.3.3 [EMCMOT] Section

BASE_PERIOD = 50000 (HAL) "Base" task period, in nanoseconds - this is the fastest thread in the machine.
On servo-based systems, there is generally no reason for **BASE_PERIOD** to be smaller than **SERVO_PERIOD**.
On machines with software step generation, the **BASE_PERIOD** determines the maximum number of steps per second. In the absence of long step length and step space requirements, the absolute maximum step rate is one step per **BASE_PERIOD**. Thus, the **BASE_PERIOD** shown above gives an absolute maximum step rate of 20000 steps per second. 50000ns is a fairly conservative value. The smallest usable value is related to the Latency Test result (**??**), the necessary step length, and the processor speed.
Choosing a BASE_PERIOD that is too low can lead to the "Unexpected realtime delay" message, lockups, or spontaneous reboots.

SERVO_PERIOD = 1000000 (HAL) "Servo" task period is also in nanoseconds. This value will be rounded to an integer multiple of **BASE_PERIOD**. This value is used even on systems based on stepper motors.
This is the rate at which new motor positions are computed, following error is checked, PID output values are updated, and so on.
Most systems will not need to change this value. It is the update rate of the low level motion planner.

TRAJ_PERIOD = 1000000 (HAL) Trajectory Planner task period in nanoseconds This value will be rounded to an integer multiple of **SERVO_PERIOD**.
Except for machines with unusual kinematics (e.g., hexapods) there is no reason to make this value larger than **SERVO_PERIOD**.

4.3.4 [TASK] Section

CYCLE_TIME = 0.001 The period, in seconds, at which EMCTASK will run. This parameter affects the polling interval when waiting for motion to complete, when executing a pause instruction, and when accepting a command from a user interface. There is usually no need to change this number.

4.3.5 [HAL] section

HALFILE = example.hal Execute the file 'example.hal' at startup. If **HALFILE** is specified multiple times, the files are executed in the order they appear in the inifile. Almost all configurations will have at least one **HALFILE**, and stepper systems typically have two such files, one which specifies the generic stepper configuration (`core_stepper.hal`) and one which specifies the machine pinout (`xxx_pinout.hal`)

HAL = command Execute 'command' as a single hal command. If **HAL** is specified multiple times, the commands are executed in the order they appear in the inifile. **HAL** lines are executed after all **HALFILE** lines.

SHUTDOWN = shutdown.hal Execute the file 'shutdown.hal' when emc is exiting. Depending on the hardware drivers used, this may make it possible to set outputs to defined values when emc is exited normally. However, because there is no guarantee this file will be executed (for instance, in the case of a computer crash) it is not a replacement for a proper physical estop chain or other protections against software failure.

POSTGUI_HALFILE = example2.hal *(Only with the AXIS GUI)* Execute 'example2.hal' after the GUI has created its HAL pins. See section **??** for more information.

4.3.6 [TRAJ] Section

The [TRAJ] section contains general parameters for the trajectory planning module in EMCMOT.

COORDINATES = X Y Z The names of the axes being controlled. X, Y, Z, A, B, C, U, V, and W are all valid. Only axis named in **COORDINATES** are accepted in g-code. This has no effect on the mapping from G-code axis names (X- Y- Z-) to joint numbers–for "trivial kinematics", X is always joint 0, A is always joint 4, and U is always joint 7, and so on. It is permitted to write an axis name twice (e.g., X Y Y Z for a gantry machine) but this has no effect.

AXES = 3 One more than the number of the highest joint number in the system. For an XYZ machine, the joints are numbered 0, 1 and 2; in this case AXES should be 3. For an XYUV machine using "trivial kinematics", the V joint is numbered 7 and therefore AXES should be 8. For a machine with nontrivial kinematics (e.g., scarakins) this will generally be the number of controlled joints.

HOME = 0 0 0 Coordinates of the homed position of each axis. Again for a fourth axis you will need 0 0 0 0. This value is only used for machines with nontrivial kinematics. On machines with trivial kinematics this value is ignored.

LINEAR_UNITS = <units> Specifies the machine units for linear axes. Possible choices are (in, inch, imperial, metric, mm).
This does not affect the linear units in NC code (the G20 and G21 words do this).

ANGULAR_UNITS = <units> Specifies the machine units for rotational axes. Possible choices are 'deg', 'degree' (360 per circle), 'rad', 'radian' (2pi per circle), 'grad', or 'gon' (400 per circle). This does not affect the angular units of NC code. In RS274NGC, A-, B- and C- words are always expressed in degrees.

DEFAULT_VELOCITY = 0.0167 The initial rate for jogs of linear axes, in machine units per second. The value shown equals one unit per minute.

DEFAULT_ACCELERATION = 2.0 In machines with nontrivial kinematics, the acceleration used for "teleop" (cartesian space) jogs, in machine units per second per second.

MAX_VELOCITY = 5.0 The maximum velocity for any axis or coordinated move, in machine units per second. The value shown equals 300 units per minute.

MAX_ACCELERATION = 20.0 The maximum acceleration for any axis or coordinated axis move, in machine units per second per second.

POSITION_FILE = position.txt If set to a non-empty value, the joint positions are stored between runs in this file. This allows the machine to start with the same coordinates it had on shutdown.[3] If unset, joint positions are not stored and will begin at 0 each time emc is started.

4.3.7 [AXIS_<num>] Section

The [AXIS_0], [AXIS_1], etc. sections contains general parameters for the individual components in the axis control module. The axis section names begin numbering at 0, and run through the number of axes specified in the [TRAJ] AXES entry minus 1.

TYPE = LINEAR The type of axes, either LINEAR or ANGULAR.

UNITS = inch If specified, this setting overrides the related [TRAJ] UNITS setting. (e.g., [TRAJ]LINEAR_UNITS if the TYPE of this axis is LINEAR, [TRAJ]ANGULAR_UNITS if the TYPE of this axis is ANGULAR)

MAX_VELOCITY = 1.2 Maximum velocity for this axis in machine units per second.

MAX_ACCELERATION = 20.0 Maximum acceleration for this axis in machine units per second squared.

BACKLASH = 0.000 Backlash in machine units. Backlash compensation value can be used to make up for small deficiencies in the hardware used to drive an axis.

COMP_FILE = file.extension A file holding a compensation structure for the specific axis. The values inside are triplets of nominal, forward and reverse positions which correspond to the nominal position (where it should be), forward (where the axis is while travelling forward) and reverse (where the axis is while travelling back). One set of triplets per line. Currently the limit inside EMC2 is for 256 triplets / axis. If COMP_FILE is specified, BACKLASH is ignored. COMP_FILE values are in machine units.

COMP_FILE_TYPE = 1 Specifying a non-zero value changes the expected format of the COMP_FILE. For COMP_FILE_TYPE of zero, the values are triplets for nominal, forward & reverse. Otherwise, the values in the COMP_FILE are nominal, forward_trim and reverse_trim. These correspond to the nominal, nominal-forward and nominal-reverse defined above.

MIN_LIMIT = -1000 The minimum limit (soft limit) for axis motion, in machine units. When this limit is exceeded, the controller aborts axis motion.

MAX_LIMIT = 1000 The maximum limit (soft limit) for axis motion, in machine units. When this limit is exceeded, the controller aborts axis motion.

[3] This assumes there was no movement of the machine while powered off. It helps on smaller machines without home switches.

MIN_FERROR = 0.010 This is the value in machine units by which the axis is permitted to deviate from commanded position at very low speeds. If MIN_FERROR is smaller than FERROR, the two produce a ramp of error trip points. You could think of this as a graph where one dimension is speed and the other is permitted following error. As speed increases the amount of following error also increases toward the FERROR value.

FERROR = 1.0 FERROR is the maximum allowable following error, in machine units. If the difference between commanded and sensed position exceeds this amount, the controller disables servo calculations, sets all the outputs to 0.0, and disables the amplifiers. If MIN_FERROR is present in the .ini file, velocity-proportional following errors are used. Here, the maximum allowable following error is proportional to the speed, with FERROR applying to the rapid rate set by [TRAJ]MAX_VELOCITY, and proportionally smaller following errors for slower speeds. The maximum allowable following error will always be greater than MIN_FERROR. This prevents small following errors for stationary axes from inadvertently aborting motion. Small following errors will always be present due to vibration, etc. 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.

4.3.7.1 Homing-related items

The next few parameters are Homing related, for a better explanation read Section 4.4

HOME_OFFSET = 0.0 The axis position of the home switch or index pulse, in machine units.

HOME_SEARCH_VEL = 0.0 Initial homing velocity in machine units per escond. 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 Final homing velocity in machine units per second.

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.

4.3.7.2 Servo-related items

The following items are for servo-based systems and servo-like systems including the univstep board from Pico Systems.[4] This description assumes that the units of output from the PID component are volts.

P = 50 (HAL) The proportional gain for the axis servo. This value multiplies the error between commanded and actual position in machine units, resulting in a contribution to the computed voltage for the motor amplifier. The units on the P gain are volts per machine unit, e.g., $\frac{volt}{mm}$ if machine units are millimeters.

I = 0 (HAL) The integral gain for the axis servo. The value multiplies the cumulative error between commanded and actual position in machine units, resulting in a contribution to the computed voltage for the motor amplifier. The units on the I gain are volts per machine unit per second, e.g., $\frac{volt}{mm\,s}$ if machine units are millimeters.

[4] Refer to the the EMC2_Integrator_Manual for further information about servo systems and PID control.

D = 0 (HAL) The derivative gain for the axis servo. The value multiplies the difference between the current and previous errors, resulting in a contribution to the computed voltage for the motor amplifier. The units on the D gain are volts per machine unit per second, e.g., $\frac{volt}{mm/s}$ if machine units are millimeters.

FF0 = 0 (HAL) 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 machine unit, e.g., $\frac{volt}{mm}$ if machine units are millimeters.

FF1 = 0 (HAL) 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 machine unit per second, e.g., $\frac{volt}{mm\,s}$ if machine units are millimeters.

FF2 = 0 (HAL) The 2nd order feedforward gain. This number is multiplied by the change in commanded position per second per second, resulting in a contribution to the computed voltage for the motor amplifier. The units on the FF2 gain are volts per machine unit per second per second, e.g., $\frac{volt}{mm\,s^2}$ if machine units are millimeters.

OUTPUT_SCALE = 1.000

OUTPUT_OFFSET = 0.000 (HAL) 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:

$$raw = \frac{output - offset}{scale}$$

The value for scale can be obtained analytically by doing a unit analysis, i.e., units are [output SI units]/[actuator units]. For example, on a machine with a velocity mode amplifier such that 1 volt results in 250 mm/sec velocity,

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

Note that the units of the offset are in machine units, e.g., mm/sec, and they are pre-subtracted from the sensor readings. The value for this offset is obtained by finding the value of your output which yields 0.0 for the actuator output. If the DAC is linearized, this offset is normally 0.0.
The scale and offset can be used to linearize the 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:

1. Build a calibration table for the output, driving the DACs with a desired voltage and measuring the result. See table 4.3.7.2 for an example of voltage measurements.
2. Do a least-squares linear fit to get coefficients a, b such that

$$meas = a * raw + b$$

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

   (a)

   $$cmd = a * raw + b$$

   (b)

   $$raw = (cmd - b)/a$$

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

MAX_OUTPUT = 10 (HAL) 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.

MIN_OUTPUT = -10 (HAL) The minimum 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.

Output Voltage Measurements

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

INPUT_SCALE = 20000 (HAL) Specifies the number of pulses that corresponds to a move of one machine unit. A second number, if specified, is ignored.
For example, on a 2000 counts per rev encoder, and 10 revs/inch gearing, and desired units of mm, we have

$$\begin{aligned} input\_scale &= 2000\frac{counts}{rev} * 10\frac{rev}{inch} \\ &= 20000\frac{counts}{inch} \end{aligned}$$

4.3.7.3 Stepper-related items

SCALE = 4000 (HAL) Specifies the number of pulses that corresponds to a move of one machine unit. For stepper systems, this is the number of step pulses issued per machine unit. For servo systems, this is the number of feedback pulses per machine unit. A second number, if specified, is ignored.
For example, on a 1.8 degree stepper motor with half-stepping, and 10 revs/inch gearing, and desired units of mm, we have

$$\begin{aligned} input\_scale &= \frac{2\,steps}{1.8\,degree} * 360\frac{degree}{rev} * 10\frac{rev}{inch} \\ &= 4000\frac{steps}{inch} \end{aligned}$$

Older stepper configuration .ini and .hal used INPUT_SCALE for this value.

STEPGEN_MAXACCEL = 21.0 (HAL) Acceleration limit for the step generator. This should be 1% to 10% larger than the axis MAX_ACCELERATION. This value improves the tuning of stepgen's "position loop".

STEPGEN_MAXVEL = 1.4 (HAL) Older configuration files have a velocity limit for the step generator as well. If specified, it should also be 1% to 10% larger than the axis MAX_VELOCITY. Subsequent testing has shown that use of STEPGEN_MAXVEL does not improve the tuning of stepgen's position loop.

4.3.8 [EMCIO] Section

CYCLE_TIME = 0.100 The period, in seconds, at which EMCIO will run. Making it 0.0 or a negative number will tell EMCIO not to sleep at all. There is usually no need to change this number.

TOOL_TABLE = tool.tbl The file which contains tool information, described in **??**

TOOL_CHANGE_POSITION = 0 0 2 Specifies the XYZ location to move to when performing a tool change.

4.4 Homing

4.4.1 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.

4.4.2 Homing Sequence

Figure 4.1 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.

4.4.3 Configuration

There are six pieces of information that determine exactly how the home sequence behaves. They are defined in an [AXIS] section of the inifile.

4.4.3.1 HOME_SEARCH_VEL

The default value is zero. A value of zero causes EMC to assume that there is no home switch; the search stage of homing is skipped.

If 'HOME_SEARCH_VEL' is non-zero, then EMC assumes that there is a home switch. It begins by checking whether the home switch is already tripped. If so, it backs off the switch at HOME_SEARCH_VEL (the direction of the back-off is opposite the sign of HOME_SEARCH_VEL). Then it searches for the home switch by moving in the direction specified by the sign of 'HOME_SEARCH_VEL', at a speed determined by its absolute value. When the home switch is detected, the joint will stop as fast as possible, but there will always be some overshoot. The amount of overshoot depends on the speed. If it is too high, the joint might overshoot enough to hit a limit switch or crash into the end of travel. On the other hand, if 'HOME_SEARCH_VEL' is too low, homing can take a long time.

4.4.3.2 HOME_LATCH_VEL

Specifies the speed and direction that EMC uses when it makes its final accurate determination of the home switch (if present) and index pulse location (if present). It will usually be slower than the search velocity to 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 (meaning there is no home switch), and this parameter is nonzero, EMC goes ahead to the index pulse search. If 'HOME_SEARCH_VEL' is non-zero and this parameter is zero, it is an error and the homing operation will fail. The default value is zero.

4.4.3.3 HOME_IGNORE_LIMITS

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.

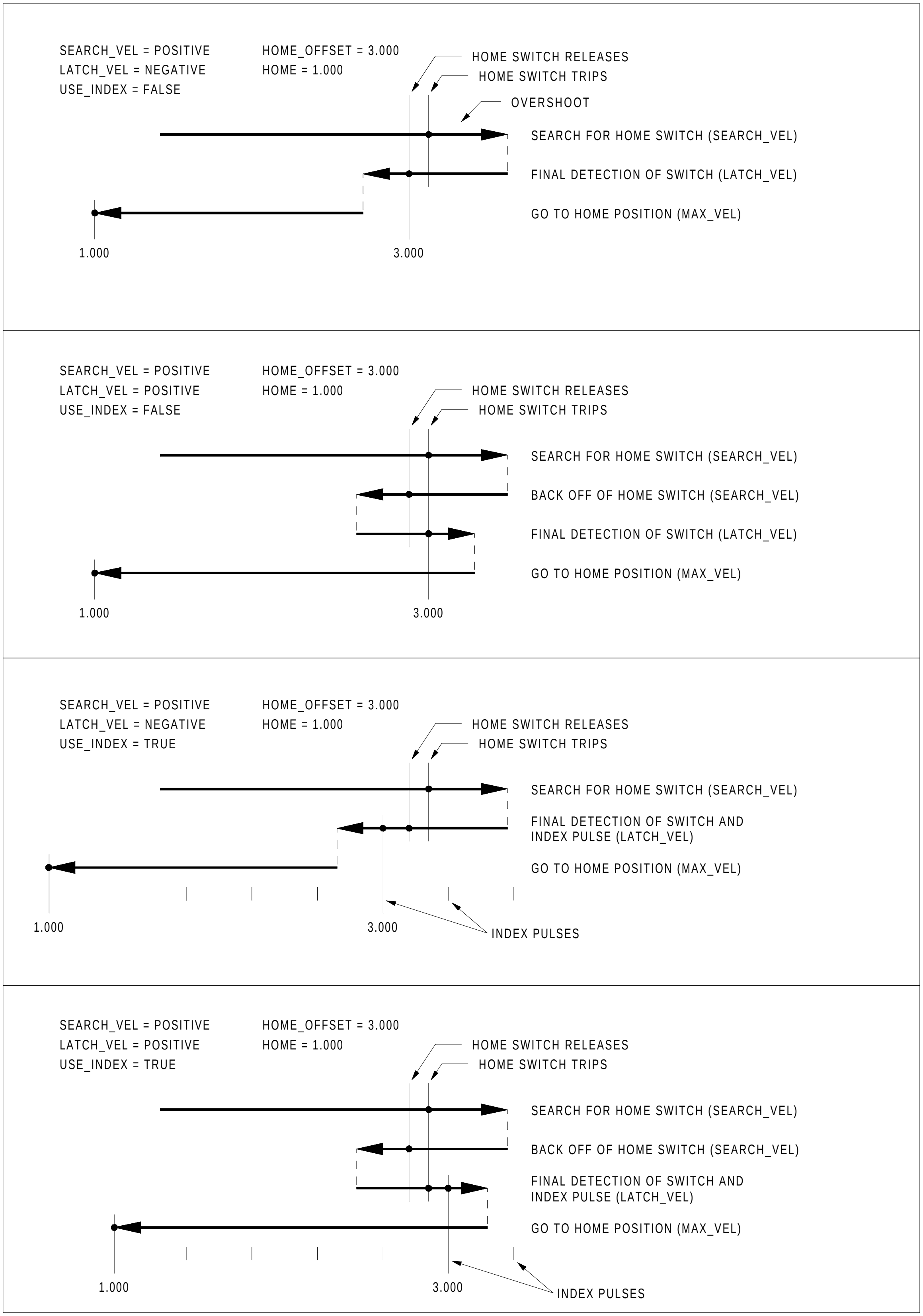


Figure 4.1: Homing Sequences

4.4.3.4 HOME_USE_INDEX

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 'HOME_SEARCH_VEL' and 'HOME_LATCH_VEL'). The default value is NO.

4.4.3.5 HOME_OFFSET

Contains the location of the home switch or index pulse, in joint coordinates. It can also be treated as the distance between the point where the switch or index pulse is latched and the zero point of the joint. After detecting the index pulse, EMC sets the joint coordinate of the current point to "HOME_OFFSET". The default value is zero.

4.4.3.6 HOME

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. [5]

4.4.3.7 HOME_IS_SHARED

If there is not a separate home switch input for this axis, but a number of momentary switches wired to the same pin, set this value to 1 to prevent homing from starting if one of the shared switches is already closed. Set this value to 0 to permit homing even if the switch is already closed.

4.4.3.8 HOME_SEQUENCE

Used to define a multi-axis homing sequence ("HOME ALL") and enforce homing order (e.g., Z may not be homed if X is not yet homed). An axis may be homed after all axes with a lower HOME_SEQUENCE have already been homed and are at the HOME_OFFSET. If two axes have the same HOME_SEQUENCE, they may be homed at the same time. If HOME_SEQUENCE is -1 or not specified then this joint will not be homed by the HOME ALL sequence. HOME_SEQUENCE numbers start with 0 and there may be no unused numbers.

[5] 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.

Chapter 5

EMC2 and HAL

See also the manual pages **motion(9)** and **iocontrol(1)**.

5.1 motion (realtime)

These pins, parameters, and functions are created by the realtime `motmod` module.

5.1.1 Pins

motion.adaptive-feed IN float When adaptive feed is enabled with `M52 P1` (See section **??**), the commanded velocity is multiplied by this value. This effect is multiplicative with the NML-level feed override value and **motion.feed-hold**.

motion.digital-out-NN OUT bit These pins are controlled by the `M62` through `M65` words.

motion.enable IN bit If this bit is driven FALSE, motion stops, the machine is placed in the "machine off" state, and a message is displayed for the operator. For normal motion, drive this bit TRUE.

motion.feed-hold IN bit When Feed Stop Control is enabled with `M53 P1` (See section **??**), and this bit is TRUE, the feed rate is set to 0.

motion.motion-inpos OUT bit TRUE if the machine is in position.

motion.probe-input IN bit `G38.2` uses the value on this pin to determine when the probe has made contact. TRUE for probe contact closed (touching), FALSE for probe contact open.

motion.spindle-brake OUT bit TRUE when the spindle brake should be applied

motion.spindle-forward OUT bit TRUE when the spindle should rotate forward

motion.spindle-reverse OUT bit TRUE when the spindle should rotate backward

motion.spindle-on OUT bit TRUE when spindle should rotate

motion.spindle-speed-out OUT float Desired spindle speed in rotations per minute

motion.spindle-index-enable I/O bit For correct operation of spindle synchronized moves, this signal must be hooked to the index-enable pin of the spindle encoder.

motion.spindle-revs IN float For correct operation of spindle synchronized moves, this signal must be hooked to the position pin of the spindle encoder.

5.1.2 Parameters

Many of these parameters serve as debugging aids, and are subject to change or removal at any time.

motion.coord-error TRUE when motion has encountered an error, such as exceeding a soft limit

motion.coord-mode TRUE when motion is in "coordinated mode", as opposed to "teleop mode"

motion.in-position Same as the pin *motion.motion-inpos*

motion.motion-enabled TRUE when motion is enabled

motion.servo.last-period The number of CPU cycles between invocations of the servo thread. Typically, this number divided by the CPU speed gives the time in seconds, and can be used to determine whether the realtime motion controller is meeting its timing constraints

motion.servo.overruns By noting large differences between successive values of *motion.servo.last-period*, the motion controller can determine that there has probably been a failure to meet its timing constraints. Each time such a failure is detected, this value is incremented.

motion.debug-bit-0

motion.debug-bit-1

motion.debug-float-0

motion.debug-float-1 These values are used for debugging purposes.

5.1.3 Functions

Generally, these functions are both added to the servo-thread in the order shown.

motion-command-handler Processes motion commands coming from user space

motion-controller Runs the emc motion controller

5.2 axis.N (realtime)

These pins and parameters are created by the realtime `motmod` module. These are actually joint values, but the pins and parameters are still called "axis.N".[1] They are read and updated by the *motion-controller* function.

5.2.1 Pins

axis.N.amp-enable-out OUT bit TRUE if the amplifier for this joint should be enabled

axis.N.amp-fault-in IN bit Should be driven TRUE if an external fault is detected with the amplifier for this joint

axis.N.home-sw-in IN bit Should be driven TRUE if the home switch for this joint is closed

axis.N.homing OUT bit TRUE if the joint is currently homing

axis.N.pos-lim-sw-in IN bit Should be driven TRUE if the positive limit switch for this joint is closed

[1] In "trivial kinematics" machines, there is a one-to-one correspondence between joints and axes.

axis.N.neg-lim-sw-in IN bit Should be driven TRUE if the negative limit switch for this joint is closed

axis.N.index-enable IO BIT Should be attached to the index-enable pin of the joint's encoder to enable homing to index pulse

axis.N.jog-counts IN s32 Connect to the "counts" pin of an external encoder to use a physical jog wheel.

axis.N.jog-enable IN bit When TRUE (and in manual mode), any change in "jog-counts" will result in motion. When false, "jog-counts" is ignored.

axis.N.jog-scale IN float Sets the distance moved for each count on "jog-counts", in machine units.

axis.N.jog-vel-mode IN bit When FALSE (the default), the jogwheel operates in position mode. The axis will move exactly jog-scale units for each count, regardless of how long that might take. When TRUE, the wheel operates in velocity mode - motion stops when the wheel stops, even if that means the commanded motion is not completed.

axis.N.motor-pos-cmd OUT float The commanded position for this joint.

axis.N.motor-pos-fb IN float The actual position for this joint.

axis.N.joint-pos-cmd The joint (as opposed to motor) commanded position. There may be an offset between the joint and motor positions–for example, the homing process sets this offset.

axis.N.joint-pos-fb The joint (as opposed to motor) feedback position.

5.2.2 Parameters

Many of these parameters serve as debugging aids, and are subject to change or removal at any time.

axis.N.active TRUE when this joint is active

axis.N.backlash-corr Backlash or screw compensation raw value

axis.N.backlash-filt Backlash or screw compensation filtered value (respecting motion limits)

axis.N.backlash-vel Backlash or screw compensation velocity

axis.N.coarse-pos-cmd

axis.N.error TRUE when this joint has encountered an error, such as a limit switch closing

axis.N.f-error The actual following error

axis.N.f-error-lim The following error limit

axis.N.f-errored TRUE when this joint has exceeded the following error limit

axis.N.free-pos-cmd The "free planner" commanded position for this joint.

axis.N.free-tp-enable TRUE when the "free planner" is enabled for this joint

axis.N.free-vel-lim The velocity limit for the free planner

axis.N.home-state Reflects the step of homing currently taking place

axis.N.homed TRUE if the joint has been homed

axis.N.in-position TRUE if the joint is using the "free planner" and has come to a stop

axis.N.joint-vel-cmd The joint's commanded velocity

axis.N.neg-hard-limit The negative hard limit for the joint

axis.N.neg-soft-limit The negative soft limit for the joint

axis.N.pos-hard-limit The positive hard limit for the joint

axis.N.pos-soft-limit The positive soft limit for the joint

5.3 iocontrol (userspace)

These pins are created by the userspace IO controller, usually called `io`.

5.3.1 Pins

iocontrol.0.coolant-flood TRUE when flood coolant is requested

iocontrol.0.coolant-mist TRUE when mist coolant is requested

iocontrol.0.emc-enable-in Should be driven FALSE when an external estop condition exists

iocontrol.0.lube

iocontrol.0.lube_level Should be driven TRUE when

iocontrol.0.tool-change TRUE when a tool change is requested

iocontrol.0.tool-changed Should be driven TRUE when a tool change is completed

iocontrol.0.tool-prep-number The number of the next tool, from the RS274NGC T-word

iocontrol.0.tool-prepare TRUE when a tool prepare is requested

iocontrol.0.tool-prepared Should be driven TRUE when a tool prepare is completed

iocontrol.0.user-enable-out FALSE when an internal estop condition exists

iocontrol.0.user-request-enable TRUE when the user has requested that estop be cleared

Part IV

HAL Specifics

Chapter 6

Introduction

6.1 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 complex 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.

6.1.1 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.

6.1.1.1 Part Selection

The machine builder does not need to worry about how each individual part works. He treats them as black boxes. During the design stage, he decides which parts he is going to use - steppers or servos, which brand of servo amp, what kind of limit switches and how many, etc. The integrator’s decisions about which specific components to use is based on what that component does and the specifications supplied by the manufacturer of the device. The size of a motor and the load it must drive will affect the choice of amplifier needed to run it. The choice of amplifier may affect the kinds of feedback needed by the amp and the velocity or position signals that must be sent to the amp from a control.

In the HAL world, the integrator must decide what HAL components are needed. Usually every interface card will require a driver. Additional components may be needed for software generation of step pulses, PLC functionality, and a wide variety of other tasks.

6.1.1.2 Interconnection Design

The designer of a hardware system not only selects the parts, he also decides how those parts will be interconnected. Each black box has terminals, perhaps only two for a simple switch, or dozens for a servo drive or PLC. They need to be wired together. The motors connect to the servo amps, the limit switches connect to the controller, and so on. As the machine builder works on the design, he creates a large wiring diagram that shows how all the parts should be interconnected.

When using HAL, components are interconnected by signals. The designer must decide which signals are needed, and what they should connect.

6.1.1.3 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.

6.1.1.4 Testing

Very few machines work right the first time. While testing, the builder may use a meter to see whether a limit switch is working or to measure the DC voltage going to a servo motor. He may hook up an oscilloscope to check the tuning of a drive, or to look for electrical noise. He may find a problem that requires the wiring diagram to be changed; perhaps a part needs to be connected differently or replaced with something completely different.

HAL provides the software equivalents of a voltmeter, oscilloscope, signal generator, and other tools needed for testing and tuning a system. The same commands used to build the system can be used to make changes as needed.

6.1.2 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.

6.2 HAL Concepts

This section is a glossary that defines key HAL terms but it is a bit different than a traditional glossary because these terms are not arranged in alphabetical order. They are arranged by their relationship or flow in the HAL way of things.

Component: When we talked about hardware design, we referred to the individual pieces as "parts", "building blocks", "black boxes", etc. The HAL equivalent is a "component" or "HAL component". (This document uses "HAL component" when there is likely to be confusion with other kinds of components, but normally just uses "component".) A HAL component is a piece of software with well-defined inputs, outputs, and behavior, that can be installed and interconnected as needed.

Parameter: Many hardware components have adjustments that are not connected to any other components but still need to be accessed. For example, servo amps often have trim pots to allow for tuning adjustments, and test points where a meter or scope can be attached to view the tuning results. HAL components also can have such items, which are referred to as "parameters". There are two types of parameters: Input parameters are equivalent to trim pots - they are values that can be adjusted by the user, and remain fixed once they are set. Output parameters cannot be adjusted by the user - they are equivalent to test points that allow internal signals to be monitored.

Pin: Hardware components have terminals which are used to interconnect them. The HAL equivalent is a "pin" or "HAL pin". ("HAL pin" is used when needed to avoid confusion.) All HAL pins are named, and the pin names are used when interconnecting them. HAL pins are software entities that exist only inside the computer.

Physical_Pin: Many I/O devices have real physical pins or terminals that connect to external hardware, for example the pins of a parallel port connector. To avoid confusion, these are referred to as "physical pins". These are the things that “stick out” into the real world.

Signal: In a physical machine, the terminals of real hardware components are interconnected by wires. The HAL equivalent of a wire is a "signal" or "HAL signal". HAL signals connect HAL pins together as required by the machine builder. HAL signals can be disconnected and reconnected at will (even while the machine is running).

Type: When using real hardware, you would not connect a 24 volt relay output to the +/-10V analog input of a servo amp. HAL pins have the same restrictions, which are based upon their type. Both pins and signals have types, and signals can only be connected to pins of the same type. Currently there are 4 types, as follows:

- BIT - a single TRUE/FALSE or ON/OFF value
- FLOAT - a 32 bit floating point value, with approximately 24 bits of resolution and over 200 bits of dynamic range.
- U32 - a 32 bit unsigned integer, legal values are 0 to +4294967295
- S32 - a 32 bit signed integer, legal values are -2147483648 to +2147483647

Function: Real hardware components tend to act immediately on their inputs. For example, if the input voltage to a servo amp changes, the output also changes automatically. However software components cannot act "automatically". Each component has specific code that must be executed to do whatever that component is supposed to do. In some cases, that code simply runs as part of the component. However in most cases, especially in realtime components, the code must run in a specific sequence and at specific intervals. For example, inputs should be read before calculations are performed on the input data, and outputs should not be written until the calculations are done. In these cases, the code is made available to the system in the form of one or more "functions". Each function is a block of code that performs a specific action. The system integrator can use "threads" to schedule a series of functions to be executed in a particular order and at specific time intervals.

Thread: A "thread" is a list of functions that runs at specific intervals as part of a realtime task. When a thread is first created, it has a specific time interval (period), but no functions. Functions can be added to the thread, and will be executed in order every time the thread runs.

As an example, suppose we have a parport component named hal_parport. That component defines one or more HAL pins for each physical pin. The pins are described in that component's doc section: their names, how each pin relates to the physical pin, are they inverted, can you change polarity, etc. But that alone doesn't get the data from the HAL pins to the physical pins. It takes code to do that, and that is where functions come into the picture. The parport component needs at least two functions: one to read the physical input pins and update the HAL pins, the other to take data from the HAL pins and write it to the physical output pins. Both of these functions are part of the parport driver.

6.3 HAL components

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

6.3.1 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, it is intended to work as a full User Interface using external knobs & switches

6.3.2 Internal Components

stepgen Software step pulse generator with position loop. See section 14.1

encoder Software based encoder counter. See section 14.3

pid Proportional/Integral/Derivative control loops. See section 14.4

siggen A sine/cosine/triangle/square wave generator for testing. See section 14.7

supply a simple source for testing

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

6.3.3 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 15.1

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

6.3.4 Tools and Utilities

halcmd Command line tool for configuration and tuning. See section 10.1

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

halmeter A handy multimeter for HAL signals. See section 10.2

halscope A full featured digital storage oscilloscope for HAL signals. See section 10.3

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

6.4 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.

6.4.1 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.

6.4.2 Erector Sets[1]

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.

[1]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. Perseverence, 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.

6.4.3 Tinkertoys[2]

> Wooden Tinker toys had a more humane feel that 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 transfered them to an from rockets docked at the center hub while they transfered 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 7) 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 it 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 diferent.

[2] Tinkertoy is now a registered trademark of the Hasbro company.

6.4.4 A Lego Example[3]

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.

6.5 Timing Issues In HAL

Unlike the physical wiring models between black boxes that we have said that HAL is based upon, simply connecting two pins with a hal-signal falls far short of the action of the physical case.

True relay logic consists of relays connected together, and when a contact opens or closes, current flows (or stops) immediately. Other coils may change state, etc, and it all just "happens". But in PLC style ladder logic, it doesn't work that way. Usually in a single pass through the ladder, each rung is evaluated in the order in which it appears, and only once per pass. A perfect example is a single rung ladder, with a NC contact in series with a coil. The contact and coil belong to the same relay.

If this were a conventional relay, as soon as the coil is energized, the contacts begin to open and de-energize it. That means the contacts close again, etc, etc. The relay becomes a buzzer.

With a PLC, if the coil is OFF and the contact is closed when the PLC begins to evaluate the rung, then when it finishes that pass, the coil is ON. The fact that turning on the coil opens the contact feeding it is ignored until the next pass. On the next pass, the PLC sees that the contact is open, and de-energizes the coil. So the relay still switches rapidly between on and off, but at a rate determined by how often the PLC evaluates the rung.

In HAL, the function is the code that evaluates the rung(s). In fact, the HAL-aware realtime version of ClassicLadder exports a function to do exactly that. Meanwhile, a thread is the thing that runs the function at specific time intervals. Just like you can choose to have a PLC evaluate all its rungs every 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 you might have a 50μs thread and a 1ms thread. These would be created baseds on BASE_PERIOD and SERVO_PERIOD–the actual times depend on the ini.

The next step is to decide what each thread needs to do. Some of those decisions are the same in (nearly) any emc system–For instance, motion-command-handler is always added to servo-thread.

Other connections would be made by the integrator. These might include hooking the STG driver's encoder read and DAC write functions to the servo thread, or hooking stepgen's function to the base-thread, along with the parport function(s) to write the steps to the port.

[3] The Lego name is a trademark of the Lego company.

Chapter 7

HAL Tutorial

7.1 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 halrun can be used to create a working system. It is a command line or text file tool for configuration and tuning. The following examples illustrate its setup and operation.

7.1.1 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.

7.1.2 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 neccessary 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 and, if necessary, invoked the `emc-environment` script to prepare your shell. In that case, all you need to do is load the required RTOS and RTAPI modules into memory. Just run the following command:

```
emc2$ halrun
halcmd:
```

With the realtime OS and RTAPI loaded, we can move into the first example. Notice that the prompt has changed from the shell's "$" to "halcmd". This is because subsequent commands will be interpreted as HAL commands, not shell commands. `halrun` is a simple shell script, and it is more or less equivalent to running

```
emc2$ realtime start
emc2$ halcmd -kf
```

When `halcmd` exits, halrun stops the realtime system, just like

```
emc2$ realtime stop
```

You can also supply arguments to halrun that are passed on to halcmd, or give the name of a .hal file. Because `halrun` stops the realtime system when it exits, the hal file run in this way will typically end with a command that waits for completion, like `loadrt -w halscope`.

7.2 Tab-completion

Your version of halcmd may include tab-completion. Instead of completing filenames as a shell does, it completes commands with HAL identifiers. Try pressing tab after starting a HAL command:

```
halcmd: lo<TAB>
loadrt    loadusr   lock
halcmd: loadrt d<TAB>
ddt       debounce
```

7.3 A Simple Example

7.3.1 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 14.7 of this document. It is a realtime component, implemented as a Linux kernel module. To load `siggen` use the `halcmd loadrt` command:

```
halcmd: loadrt siggen
```

7.3.2 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 10.1 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:

```
halcmd: show comp
Loaded HAL Components:
ID      Type  Name                                            PID   State
32769   RT    siggen                                                ready
9775    User  halcmd9775                                       9775 initializing
```

Since halcmd itself is a HAL component, it will always show up in the list[1]. 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:

```
halcmd: show pin
Component Pins:
Owner  Type  Dir    Value      Name 02    float -W   0.00000e+00  siggen.0.cosine
32769  float OUT  0.00000e+00  siggen.0.sawtooth
32769  float OUT  0.00000e+00  siggen.0.sine
32769  float OUT  0.00000e+00  siggen.0.square
32769  float OUT  0.00000e+00  siggen.0.triangle
```

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:

```
halcmd: show param
Parameters:
Owner  Type  Dir    Value      Name
 32769    float  RW   1.00000e+00  siggen.0.amplitude
 32769    float  RW   1.00000e+00  siggen.0.frequency
 32769    float  RW   0.00000e+00  siggen.0.offset
 32769    s32    RO             0   siggen.0.update.time
 32769    s32    RW             0   siggen.0.update.tmax
```

The show param command shows all the parameters in the HAL. Right now each parameter has the default value it was given when the component was loaded. Note the column labeled Dir. The parameters labeled -W are 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 thay 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:

```
halcmd: show funct
Exported Functions:
Owner CodeAddr   Arg     FP  Users  Name
 32769   b7f74ac5 b7d0c0b4 YES    0   siggen.0.update
```

The siggen component exported a single function. It requires floating point. It is not currently linked to any threads, so “users” is zero[2].

[1]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.

[2]The codeaddr and arg fields were used in development, and should probably be removed from the halcmd listing.

7.3.3 Making realtime code run

To actually run the code contained in the function `siggen.0.update`, we need a realtime thread. The component called `threads` that is used to create a new thread. Lets create a thread called `test-thread` with a period of 1mS (1000000nS):

```
halcmd: loadrt threads name1=test-thread period1=1000000
```

Let's see if that worked:

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

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:

```
halcmd: addf siggen.0.update test-thread
```

Up till now, we've been using `halcmd` only to look at the HAL. However, this time we used the `addf` (add function) command to actually change something in the HAL. We told `halcmd` to add the function `siggen.0.update` to the thread `test-thread`, and if we look at the thread list again, we see that it succeeded:

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

There is one more step needed before the `siggen` component starts generating signals. When the HAL is first started, the thread(s) are not actually running. This is to allow you to completely configure the system before the realtime code starts. Once you are happy with the configuration, you can start the realtime code like this:

```
halcmd: start
```

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

```
halcmd: show pin
Component Pins:
Owner  Type  Dir     Value      Name
 32769    float OUT   2.12177e-01  siggen.0.cosine
 32769    float OUT  -5.64055e-01  siggen.0.sawtooth
 32769    float OUT   9.79820e-01  siggen.0.sine
 32769    float OUT  -1.00000e+00  siggen.0.square
 32769    float OUT   1.28110e-01  siggen.0.triangle
halcmd: show pin
Component Pins:
Owner  Type  Dir     Value      Name
 32769    float OUT   5.19530e-01  siggen.0.cosine
 32769    float OUT   6.73893e-01  siggen.0.sawtooth
 32769    float OUT  -8.54452e-01  siggen.0.sine
 32769    float OUT   1.00000e+00  siggen.0.square
 32769    float OUT   3.47785e-01  siggen.0.triangle
```

We did two `show pin` commands in quick succession, and you can see that the outputs are no longer zero. The sine, cosine, sawtooth, and triangle outputs are changing constantly. The square output is also working, however it simply switches from +1.0 to -1.0 every cycle.

7.3.4 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:

```
halcmd: setp siggen.0.amplitude 5
emc2$
```

Check the parameters and pins again:

```
halcmd: setp siggen.0.amplitude 5
halcmd: show param
Parameters:
Owner  Type  Dir    Value      Name
 32769    float  RW   5.00000e+00  siggen.0.amplitude
 32769    float  RW   1.00000e+00  siggen.0.frequency
 32769    float  RW   0.00000e+00  siggen.0.offset
 32769    s32    RO          397   siggen.0.update.time
 32769    s32    RW       109100   siggen.0.update.tmax
halcmd: show pin
Component Pins:
Owner  Type  Dir     Value      Name
 32769    float OUT   4.78453e+00  siggen.0.cosine
 32769    float OUT  -4.53106e+00  siggen.0.sawtooth
 32769    float OUT   1.45198e+00  siggen.0.sine
 32769    float OUT  -5.00000e+00  siggen.0.square
 32769    float OUT   4.02213e+00  siggen.0.triangle
```

Note that the value of parameter `siggen.0.amplitude` has changed to 5.000, and that the pins now have larger values.

7.3.5 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:

```
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
```

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:

```
halcmd: save all saved.hal
```

7.3.6 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 `-f <filename>` which reads commands from a file, and `-I` which shows the halcmd prompt after execiting the commands:

```
emc2$ halrun -I -f saved.hal
```

Notice that there is not a 'start' command in saved.hal. It's necessary to issue it again (or edit saved.hal to add it there):

```
halcmd: start
```

7.4 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$ halrun
halcmd: loadrt siggen
halcmd: loadrt threads name1=test-thread period1=1000000
halcmd: addf siggen.0.update test-thread
halcmd: start
halcmd: setp siggen.0.amplitude 5
```

7.4.1 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.

```
halcmd: loadusr halmeter
```

At the same time, a halmeter window opens on your screen, looking something like figure 7.1.

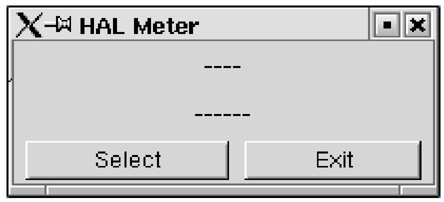


Figure 7.1: Halmeter at startup, nothing selected

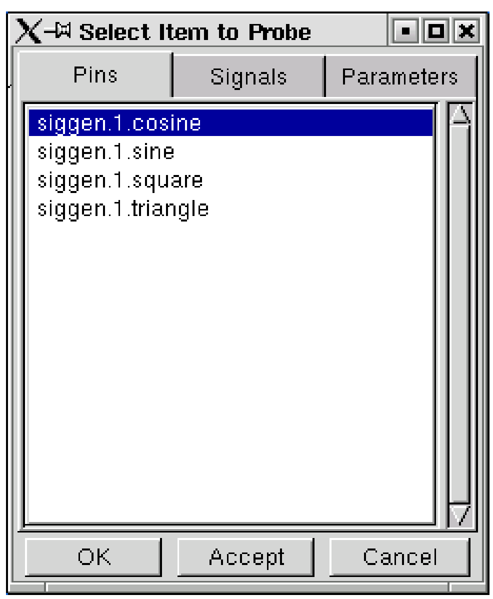


Figure 7.2: Halmeter source selection dialog

7.4.2 Using halmeter

The meter in figure 7.1 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 7.2).

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 7.3.

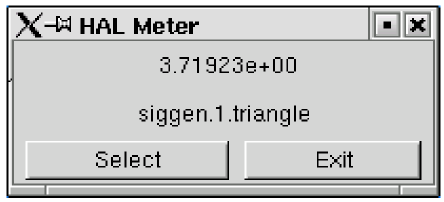


Figure 7.3: Halmeter displaying the value of a pin

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. [3]

[3] 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.

7.5 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:

```
halcmd: exit
emc2$ halrun
```

7.5.1 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:[4]

```
halrun: loadrt freqgen step_type=0,0
halcmd: loadrt siggen
halcmd: loadrt threads name1=fast fp1=0 period1=50000 name2=slow period2=1000000
```

The first command loads two step generators, both configured to generate stepping type 0. The second command loads our old friend siggen, and the third one creates two threads, a fast one with a period of 50 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:

```
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
halcmd: show param
Parameters:
Owner  Type  Dir    Value      Name
 03    float  -W   1.00000e+00  siggen.0.amplitude
```

[4]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.

```
03    float  -W   1.00000e+00  siggen.0.frequency
03    float  -W   0.00000e+00  siggen.0.offset
02    u32    -W     000000001  freqgen.0.dirhold
02    u32    -W     000000001  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    u32    -W     000000001  freqgen.0.steplen
02    u32    -W     000000001  freqgen.0.stepspace
02    float  -W   1.00000e+00  freqgen.0.velocity-scale
02    u32    -W     000000001  freqgen.1.dirhold
02    u32    -W     000000001  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    u32    -W     000000001  freqgen.1.steplen
02    u32    -W     000000001  freqgen.1.stepspace
02    float  -W   1.00000e+00  freqgen.1.velocity-scale
```

7.5.2 Connecting pins with signals

What we have is two step pulse generators, and a signal generator. Now it is time to create some HAL signals to connect the two components. We are going to pretend that the two step pulse generators are driving the X and Y axis of a machine. We want to move the table in circles. To do this, we will send a cosine signal to the X axis, and a sine signal to the Y axis. The siggen module creates the sine and cosine, but we need "wires" to connect the modules together. In the HAL, "wires" are called signals. We need to create two of them. We can call them anything we want, for this example they will be `X_vel` and `Y_vel`. The signal `X_vel` is intended to run from the cosine output of the signal generator to the velocity input of the first step pulse generator. The first step is to connect the signal to the signal generator output. To connect a signal to a pin we use the net command.

```
halcmd: net X_vel <= siggen.0.cosine
```

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

```
halcmd: show sig
ignals:
Type      Value       Name
float   0.00000e+00  X_vel
                         <== siggen.0.cosine
```

When a signal is connected to one or more pins, the show command lists the pins immediately following the signal name. The "arrow" shows the direction of data flow - in this case, data flows from pin `siggen.0.cosine` to signal `X_vel`. Now let's connect the `X_vel` to the velocity input of a step pulse generator:

```
halcmd: net X_vel => freqgen.0.velocity
```

We can also connect up the Y axis signal `Y_vel`. It is intended to run from the sine output of the signal generator to the input of the second step pulse generator. The following command accomplishes in one line what two `net` commands accomplished for `X_vel`:

```
halcmd: net Y_vel siggen.0.sine => freqgen.1.velocity
```

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

```
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
```

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`.

7.5.3 Setting up realtime execution - threads and functions

Thinking about data flowing through "wires" makes pins and signals fairly easy to understand. Threads and functions are a little more difficult. Functions contain the computer instructions that actually get things done. Thread are the method used to make those instructions run when they are needed. First let's look at the functions available to us:

```
halcmd: show funct
Exported Functions:
Owner CodeAddr   Arg    FP  Users  Name
 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
```

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:

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

The two threads were created when we loaded `threads`. The first one, `slow`, runs every millisecond, and is capable of running floating point functions. We will use it for `siggen.0.update` and `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:

```
halcmd: addf siggen.0.update slow
halcmd: addf freqgen.update_freq slow
halcmd: addf freqgen.make_pulses fast
```

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

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

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

7.5.4 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:

```
halcmd: setp freqgen.0.velocity-scale 10000
halcmd: setp freqgen.1.velocity-scale 10000
```

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.

7.5.5 Run it!

We now have everything configured and are ready to start it up. Just like in the first example, we use the `start` command:

```
halcmd: start
```

Although nothing appears to happen, inside the computer the step pulse generator is cranking out step pulses, varying from 10KHz forward to 10KHz reverse and back again every second. Later in this tutorial we'll see how to bring those internal signals out to run motors in the real world, but first we want to look at them and see what is happening.

7.6 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.

7.6.1 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. However, you don't need to worry about this, because the userspace portion will automatically request that the realtime part be loaded.

```
halcmd: loadusr halscope
```

The scope GUI window will open, immediately followed by a "Realtime function not linked" dialog that looks like figure 7.4[5].

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 7.5.

[5]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.

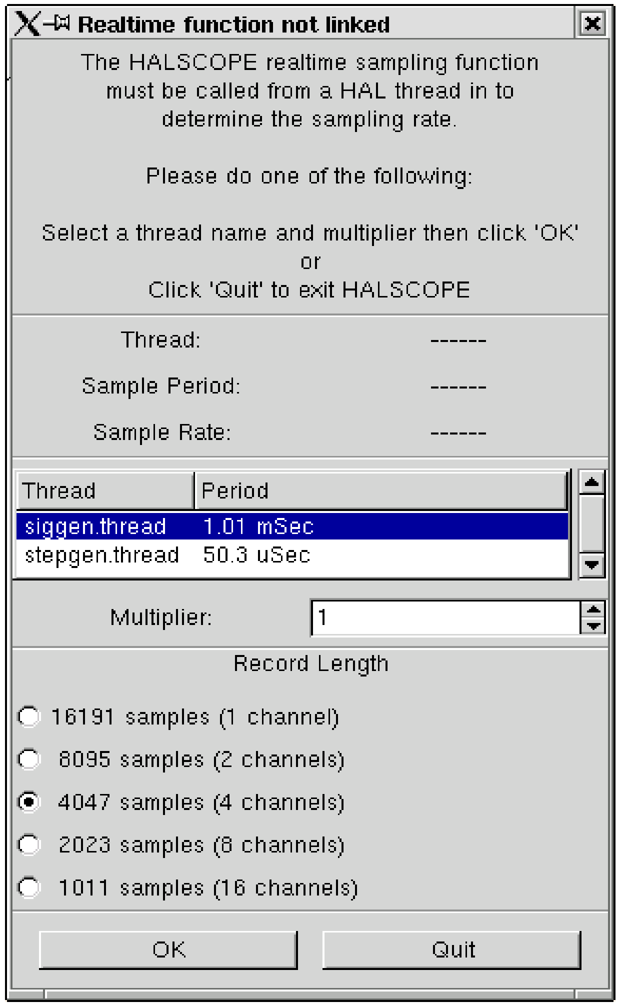


Figure 7.4: “Realtime function not linked” dialog

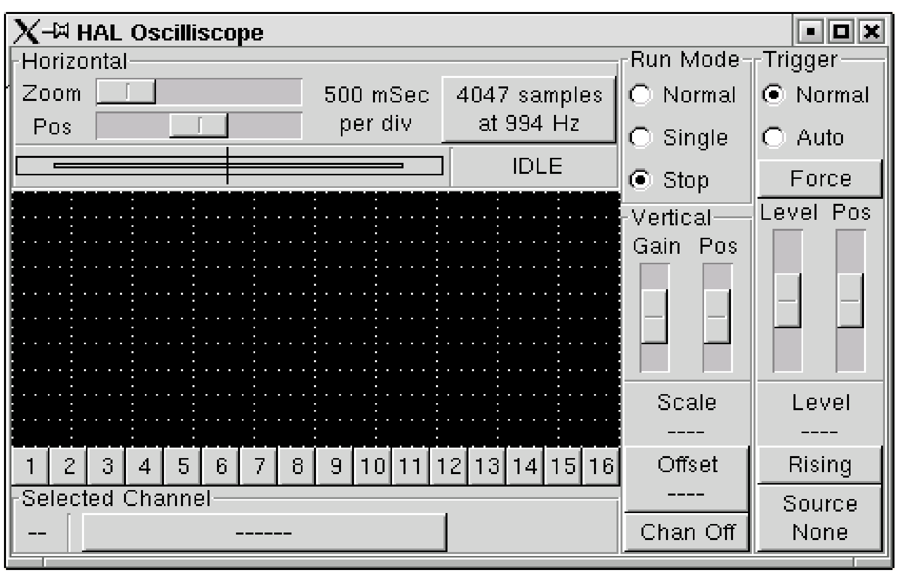


Figure 7.5: Initial scope window

7.6.2 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 7.6. 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).

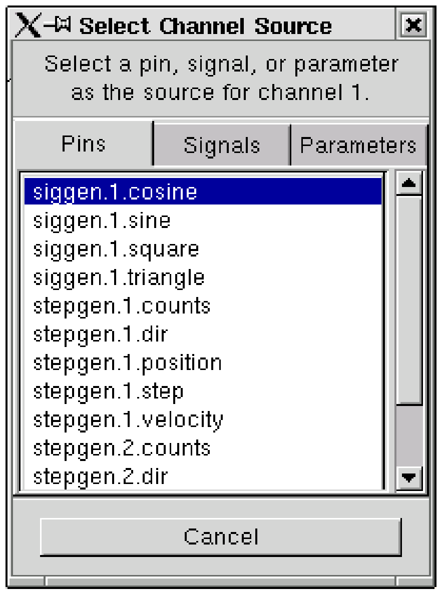


Figure 7.6: Select Channel Source dialog

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".

7.6.3 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 7.7.

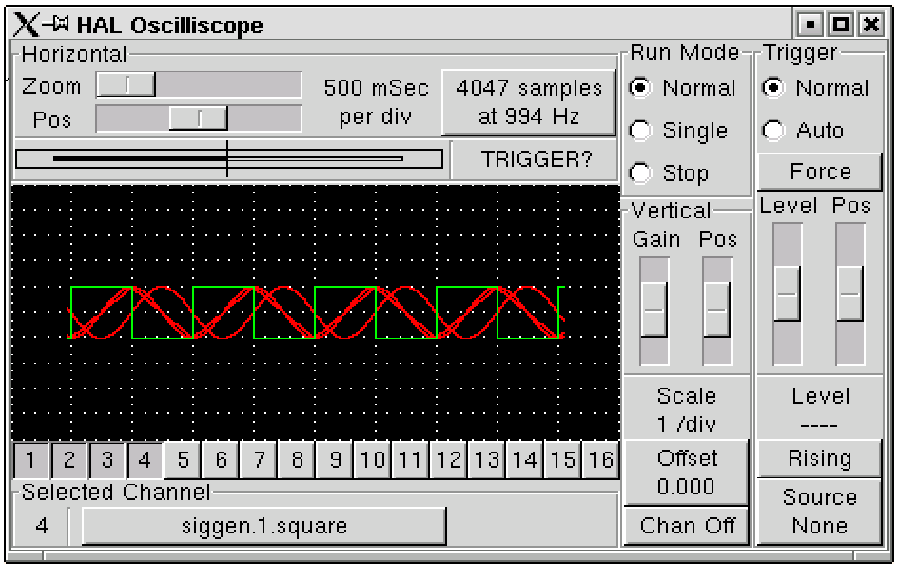


Figure 7.7: Captured Waveforms

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.

7.6.4 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 10.3 for details).

7.6.5 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 7.8). Select a probe to use for triggering by clicking on it. For this example we will use channel 3, the triangle wave.

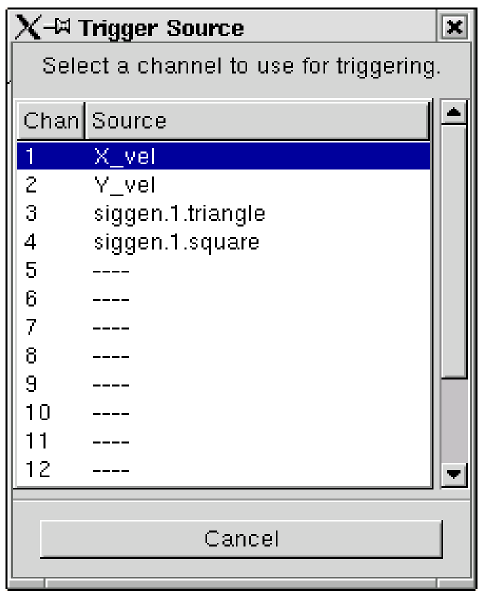


Figure 7.8: Trigger Source Dialog

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 7.9.

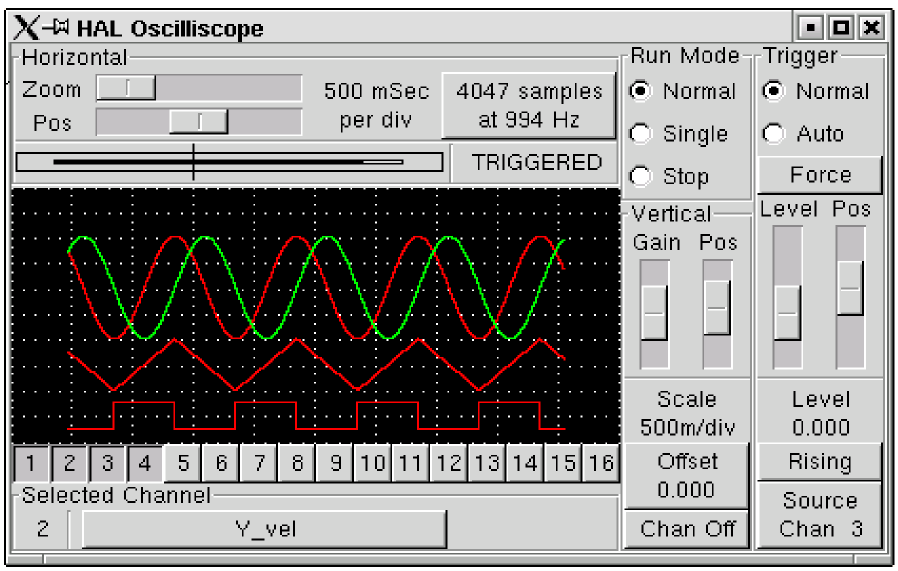


Figure 7.9: Waveforms with Triggering

7.6.6 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.

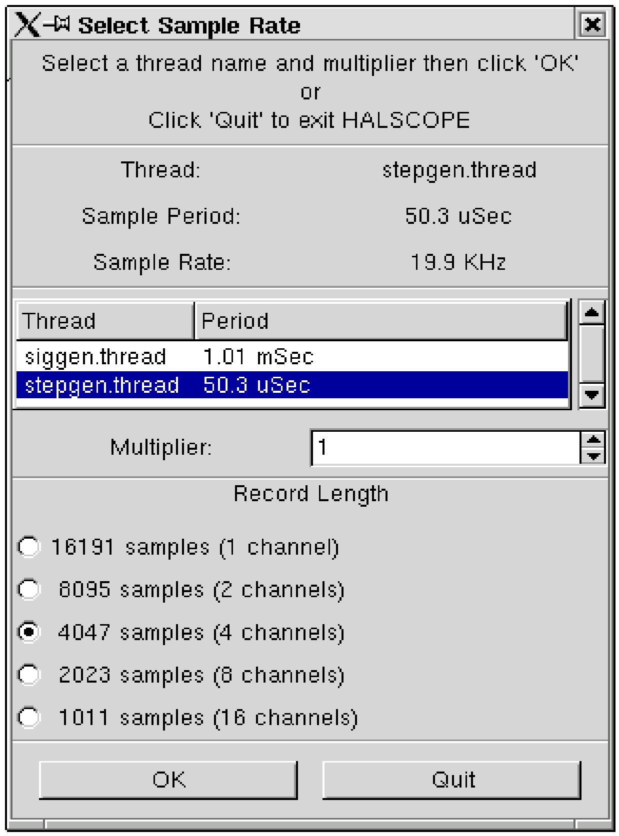


Figure 7.10: Sample Rate Dialog

7.6.7 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 if off, and do the same for channel 4. Even though the channels are turned off, they still remember what they are connected to, and in fact we will continue to use channel 3 as the trigger source. To add new channels, select channel 5, and choose pin "stepgen.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 7.11.

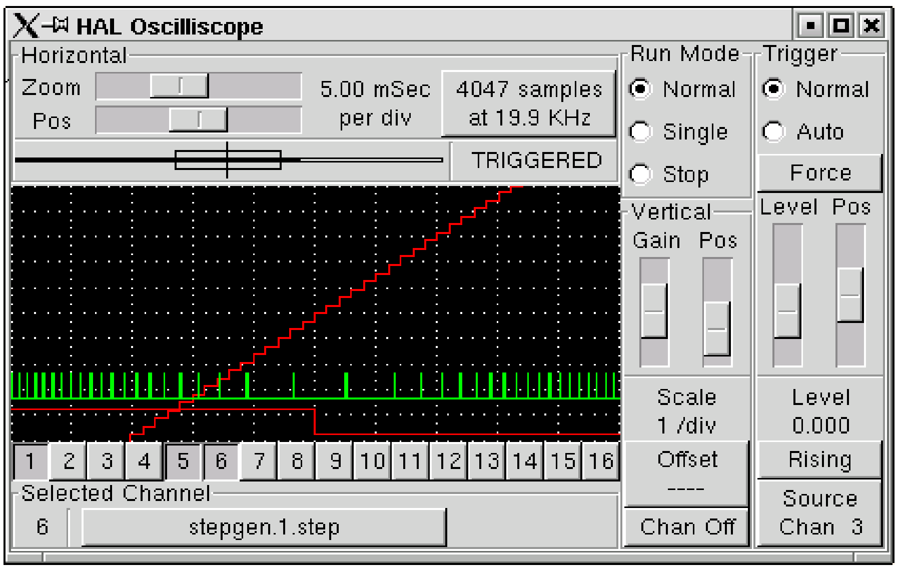


Figure 7.11: Looking at Step Pulses

7.6.8 More samples

If you want to record more samples at once, restart realtime and load halscope with a numeric argument which indicates the number of samples you want to capture, such as

```
halcmd: loadusr halscope 80000
```

if the `scope_rt` component was not already loaded, halscope will load it and request 80000 total samples, so that when sampling 4 channels at a time there will be 20000 samples per channel. (If `scope_rt` was already loaded, the numeric argument to halscope will have no effect)

Chapter 8

General Reference Information

8.1 Notation

8.1.1 Typographical Conventions

Command line examples are presented in **bold typewriter** font. Responses from the computer will be in typewriter font. As of early 2006, there are no longer commands that require root privileges, so all examples will be preceded by the normal user prompt, $. 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 you configured/compiled emc2 for the run-in-place scenario. Paths will be shown accordingly when needed.

8.1.2 Names

All HAL entities are accessed and manipulated by their names, so documenting the names of pins, signals, parameters, etc, is very important. HAL names are a maximum of 41 characters long (as defined by HAL_NAME_LEN in hal.h). Many names will be presented in a general form, with text inside angle brackets <like-this> representing fields that can take on different values.

When pins, signals, or parameters are described for the first time, their names will be preceeded by their type in (SMALL CAPS) and followed by a brief description. A typical pin definition will look something like these examples:

- (BIT) parport.<portnum>.pin-<pinnum>-in – The HAL pin associated with the physical input pin <pinnum> on the 25 pin D-shell connector.
- (FLOAT) pid.<loopnum>.output – The output of the PID loop.

At times, a shortened version of a name may be used - for example the second pin above might be referred to simply as .output when it can be done without causing confusion.

8.2 General Naming Conventions

Consistent naming conventions would make HAL much easier to use. For example, if every encoder driver provided the same set of pins and named them the same way it would be easy to change from one type of encoder driver to another. Unfortunately, like many open-source projects, HAL is a combination of things that were designed, and things that simply evolved. As a result, there are many inconsistencies. This section attempts to address that problem by defining some conventions, but it will probably be a while before all the modules are converted to follow them.

Halcmd and other low-level HAL utilities treat HAL names as single entities, with no internal structure. However, most modules do have some implicit structure. For example, a board provides several functional blocks, each block might have several channels, and each channel has one or more pins. This results in a structure that resembles a directory tree. Even though halcmd doesn't recognize the tree structure, proper choice of naming conventions will let it group related items together (since it sorts the names). In addition, higher level tools can be designed to recognize such structure, if the names provide the neccessary information. To do that, all HAL modules should follow these rules:

- Dots (".") separate levels of the heirarchy. This is analogous to the slash ("/") in a filename.
- Hypens ("-") separate words or fields in the same level of the heirarchy.
- HAL modules should not use underscores or "MixedCase". [1]
- Use only lowercase letters and numbers in names.

8.3 Hardware Driver Naming Conventions[2]

8.3.1 Pin/Parameter names

Hardware drivers should use five fields (on three levels) to make up a pin or parameter name, as follows:

```
<device-name>.<device-num>.<io-type>.<chan-num>.<specific-name>
```

The individual fields are:

<device-name> The device that the driver is intended to work with. This is most often an interface board of some type, but there are other possibilities.

<device-num> It is possible to install more than one servo board, parallel port, or other hardware device in a computer. The device number identifies a specific device. Device numbers start at 0 and increment.[3]

<io-type> Most devices provide more than one type of I/O. Even the simple parallel port has both digital inputs and digital outputs. More complex boards can have digital inputs and outputs, encoder counters, pwm or step pulse generators, analog-to-digital converters, digital-to-analog converters, or other unique capabilities. The I/O type is used to identify the kind of I/O that a pin or parameter is associated with. Ideally, drivers that implement the same I/O type, even if for very different devices, should provide a consistent set of pins and parameters and identical behavior. For example, all digital inputs should behave the same when seen from inside the HAL, regardless of the device.

[1] Underscores have all been removed, but there are still a few instances of mixed case, for example "pid.0.Pgain" instead of "pid.0.p-gain".

[2] Most drivers do not follow these conventions as of version 2.0. This chapter is really a guide for future development.

[3] Some devices use jumpers or other hardware to attach a specific ID to each board. Ideally, the driver provides a way for the user to specifically say "device-num 0 is the board with ID XXX", and the device numbers always start at 0. However at present some drivers use the board ID directly as the device number. That means it is possible to have a device number 2, without a device 0. This is a bug and will be fixed in version 2.1.

<chan-num> Virtually every I/O device has multiple channels, and the channel number identifies one of them. Like device numbers, channel numbers start at zero and increment.[4] If more than one device is installed, the channel numbers on additional devices start over at zero. If it is possible to have a channel number greater than 9, then channel numbers should be two digits, with a leading zero on numbers less than 10 to preserve sort ordering. Some modules have pins and/or parameters that affect more than one channel. For example a PWM generator might have four channels with four independent "duty-cycle" inputs, but one "frequency" parameter that controls all four channels (due to hardware limitations). The frequency parameter should use "0-3" as the channel number.

<specific-name> An individual I/O channel might have just a single HAL pin associated with it, but most have more than one. For example, a digital input has two pins, one is the state of the physical pin, the other is the same thing inverted. That allows the configurator to choose between active high and active low inputs. For most io-types, there is a standard set of pins and parameters, (referred to as the "canonical interface") that the driver should implement. The canonical interfaces are described in chapter 9.

8.3.1.1 Examples

motenc.0.encoder.2.position – the position output of the third encoder channel on the first Motenc board.

stg.0.din.03.in – the state of the fourth digital input on the first Servo-to-Go board.

ppmc.0.pwm.00-03.frequency – the carrier frequency used for PWM channels 0 through 3.

8.3.2 Function Names

Hardware drivers usually only have two kinds of HAL functions, ones that read the hardware and update HAL pins, and ones that write to the hardware using data from HAL pins. They should be named as follows:

```
<device-name>-<device-num>[.<io-type>[-<chan-num-range>]].read|write
```

<device-name> The same as used for pins and parameters.

<device-num> The specific device that the function will access.

<io-type> Optional. A function may access all of the I/O on a board, or it may access only a certain type. For example, there may be independent functions for reading encoder counters and reading digital I/O. If such independent functions exist, the <io-type> field identifies the type of I/O they access. If a single function reads all I/O provided by the board, <io-type> is not used.[5]

<chan-num-range> Optional. Used only if the <io-type> I/O is broken into groups and accessed by different functions.

read|write Indicates whether the function reads the hardware or writes to it.

[4] One glaring exception to the "channel numbers start at zero" rule is the parallel port. Its HAL pins are numbered with the corresponding pin number on the DB-25 connector. This is convenient for wiring, but inconsistent with other drivers. There is some debate over whether this is a bug or a feature.

[5] Note to driver programmers: do NOT implement separate functions for different I/O types unless they are interruptable and can work in independent threads. If interrupting an encoder read, reading digital inputs, and then resuming the encoder read will cause problems, then implement a single function that does everything.

8.3.2.1 Examples

`motenc.0.encoder.read` – reads all encoders on the first motenc board

`generic8255.0.din.09-15.read` – reads the second 8 bit port on the first generic 8255 based digital I/O board

`ppmc.0.write` – writes all outputs (step generators, pwm, DACs, and digital) on the first ppmc board

Chapter 9

Canonical Device Interfaces[1]

The following sections show the pins, parameters, and functions that are supplied by "canonical devices". All HAL device drivers should supply the same pins and parameters, and implement the same behavior.

Note that the only the `<io-type>` and `<specific-name>` fields are defined for a canonical device. The `<device-name>`, `<device-num>`, and `<chan-num>` fields are set based on the characteristics of the real device.

9.1 Digital Input

The canonical digital input (I/O type field: **`digin`**) is quite simple.

9.1.1 Pins

- (BIT) **`in`** – State of the hardware input.
- (BIT) **`in-not`** – Inverted state of the input.

9.1.2 Parameters

- None

9.1.3 Functions

- (FUNCT) **`read`** – Read hardware and set **`in`** and **`in-not`** HAL pins.

9.2 Digital Output

The canonical digital output (I/O type field: **`digout`**) is also very simple.

9.2.1 Pins

- (BIT) **`out`** – Value to be written (possibly inverted) to the hardware output.

[1]As of version 2.0, most of the HAL drivers don't quite match up to the canonical interfaces defined here. In version 2.1, the drivers will be changed to match these specs.

9.2.2 Parameters

- (BIT) **invert** – If TRUE, **out** is inverted before writing to the hardware.

9.2.3 Functions

- (FUNCT) **write** – Read **out** and **invert**, and set hardware output accordingly.

9.3 Analog Input

The canonical analog input (I/O type: **adcin**). This is expected to be used for analog to digital converters, which convert e.g. voltage to a continuous range of values.

9.3.1 Pins

- (FLOAT) **value** – The hardware reading, scaled according to the **scale** and **offset** parameters. **Value** = ((input reading, in hardware-dependent units) * **scale**) - **offset**

9.3.2 Parameters

- (FLOAT) **scale** – The input voltage (or current) will be multiplied by **scale** before being output to **value**.
- (FLOAT) **offset** – This will be subtracted from the hardware input voltage (or current) after the scale multiplier has been applied.
- (FLOAT) **bit_weight** – The value of one least significant bit (LSB). This is effectively the granularity of the input reading.
- (FLOAT) **hw_offset** – The value present on the input when 0 volts is applied to the input pin(s).

9.3.3 Functions

- (FUNCT) **read** – Read the values of this analog input channel. This may be used for individual channel reads, or it may cause all channels to be read

9.4 Analog Output

The canonical analog output (I/O Type: **adcout**). This is intended for any kind of hardware that can output a more-or-less continuous range of values. Examples are digital to analog converters or PWM generators.

Pins

- (FLOAT) **value** – The value to be written. The actual value output to the hardware will depend on the scale and offset parameters.
- (BIT) **enable** – If false, then output 0 to the hardware, regardless of the **value** pin.

9.4.1 Parameters

- (FLOAT) **offset** – This will be added to the **value** before the hardware is updated
- (FLOAT) **scale** – This should be set so that an input of 1 on the **value** pin will cause 1V
- (FLOAT) **high_limit** (optional) – When calculating the value to output to the hardware, if **value** + **offset** is greater than **high_limit**, then **high_limit** will be used instead.
- (FLOAT) **low_limit** (optional) – When calculating the value to output to the hardware, if **value** + **offset** is less than **low_limit**, then **low_limit** will be used instead.
- (FLOAT) **bit_weight** (optional) – The value of one least significant bit (LSB), in volts (or mA, for current outputs)
- (FLOAT) **hw_offset** (optional) – The actual voltage (or current) that will be output if 0 is written to the hardware.

9.4.2 Functions

(FUNCT) **write** – This causes the calculated value to be output to the hardware. If enable is false, then the output will be 0, regardles of **value**, **scale**, and **offset**. The meaning of "0" is dependent on the hardware. For example, a bipolar 12-bit A/D may need to write 0x1FF (mid scale) to the D/A get 0 volts from the hardware pin. If enable is true, read scale, offset and value and output to the adc (**scale** * **value**) + **offset**. If enable is false, then output 0.

9.5 Encoder

The canonical encoder interface (I/O type field: `encoder`) provides the functionality needed for homing to an index pulse and doing spindle synchronization, as well as basic position and/or velocity control. This interface should be implementable regardless of the actual underlying hardware, although some hardware will provide "better" results. (For example, capture the index position to +/- 1 count while moving faster, or have less jitter on the velocity pin.)

9.5.1 Pins

- (S32) `count` – Encoder value in counts.
- (FLOAT) `position` – Encoder value in position units (see parameter "scale").
- (FLOAT) `velocity` – Velocity in position units per second.
- (BIT) `reset` – When True, force counter to zero.
- (BIT) `index-enable` – (bidirectional) When True, reset to zero on next index pulse, and set pin False.

The "index-enable" pin is bi-directional, and might require a little more explanation. If "index-enable" is False, the index channel of the encoder will be ignored, and the counter will count normally. The encoder driver will never set "index-enable" True. However, some other component may do so. If "index-enable" is True, then when the next index pulse arrives, the encoder counter will be reset to zero, and the driver will set "index-enable" False. That will let the other component know that an index pulse arrived. This is a form of handshaking - the other component sets "index-enable" True to request a index pulse reset, and the driver sets it False when the request has been satisfied.

9.5.2 Parameters

- (FLOAT) **`scale`** – The scale factor used to convert counts to position units. It is in "counts per position unit". For example, if you have a 512 count per turn encoder on a 5 turn per inch screw, the scale should be 512*5 = 2560 counts per inch, which will result in "position" in inches and "velocity" in inches per second.
- (FLOAT) **`max-index-vel`** – (optional) The maximum velocity (in position units per second) at which the encoder can reset on an index pulse with `+/-` 1 count accuracy. This is an output from the encoder driver, and is intended to tell the user something about the hardware capabilities. Some hardware can reset the counter at the exact moment the index pulse arrives. Other hardware can only tell that an index pulse arrived sometime since the last time the read function was called. For the latter, `+/-` 1 count accuracy can only be achieved if the encoder advances by 1 count or less between calls to the read function.
- (FLOAT) **`velocity-resolution`** – (optional) The resolution of the velocity output, in position units per second. This is an output from the encoder driver, and is intended to tell the user something about the hardware capabilities. The simplest implementation of the velocity output is the change in postion from one call of the read function to the next, divided by the time between calls. This yields a rather coarse velocity signal that jitters back and forth between widely spaced possible values (quantization error). However, some hardware captures both the counts and the exact time when a count occurres (possibly with a very high resolution clock). That data allows the driver to calculate velocity with finer resolution and less jitter.

9.5.3 Functions

There is only one function, to read the encoder(s).

- (FUNCT) **`read`** – Capture counts, update position and velocity.

Chapter 10

Tools and Utilities

10.1 Halcmd

Halcmd is a command line tool for manipulating the HAL. There is a rather complete man page for halcmd, which will be installed if you have installed EMC2 from either source or a package. If you have compiled EMC2 for "run-in-place", the man page is not installed, but it is accessible. From the main EMC2 directory, do:

```
$ man -M docs/man halcmd
```

Chapter 7 has a number of examples of halcmd usage, and is a good tutorial for halcmd.

10.2 Halmeter

Halmeter is a "voltmeter" for the HAL. It lets you look at a pin, signal, or parameter, and displays the current value of that item. It is pretty simple to use. Start it by typing "`halmeter`" in a X windows shell. Halmeter is a GUI application. It will pop up a small window, with two buttons labeled "Select" and "Exit". Exit is easy - it shuts down the program. Select pops up a larger window, with three tabs. One tab lists all the pins currently defined in the HAL. The next lists all the signals, and the last tab lists all the parameters. Click on a tab, then click on a pin/signal/parameter. Then click on "OK". The lists will disappear, and the small window will display the name and value of the selected item. The display is updated approximately 10 times per second. If you click "Accept" instead of "OK", the small window will display the name and value of the selected item, but the large window will remain on the screen. This is convenient if you want to look at a number of different items quickly.

You can have many halmeters running at the same time, if you want to monitor several items. If you want to launch a halmeter without tying up a shell window, type "`halmeter &`" to run it in the background. You can also make halmeter start displaying a specific item immediately, by adding "`pin|sig|par[am] <name>`" to the command line. It will display the pin, signal, or parameter <name> as soon as it starts. (If there is no such item, it will simply start normally.) And finally, if you specify an item to display, you can add "-s" before the pin|sig|param to tell halmeter to use a small window. The item name will be displayed in the title bar instead of under the value, and there will be no buttons. Usefull when you want a lot of meters in a small amount of screen space.

10.3 Halscope

Halscope is an "oscilloscope" for the HAL. It lets you capture the value of pins, signals, and parameters as a function of time. Complete operating instructions should be located here eventually. For now, refer to section 7.6 in the tutorial chapter, which explains the basics.

10.4 Halshow

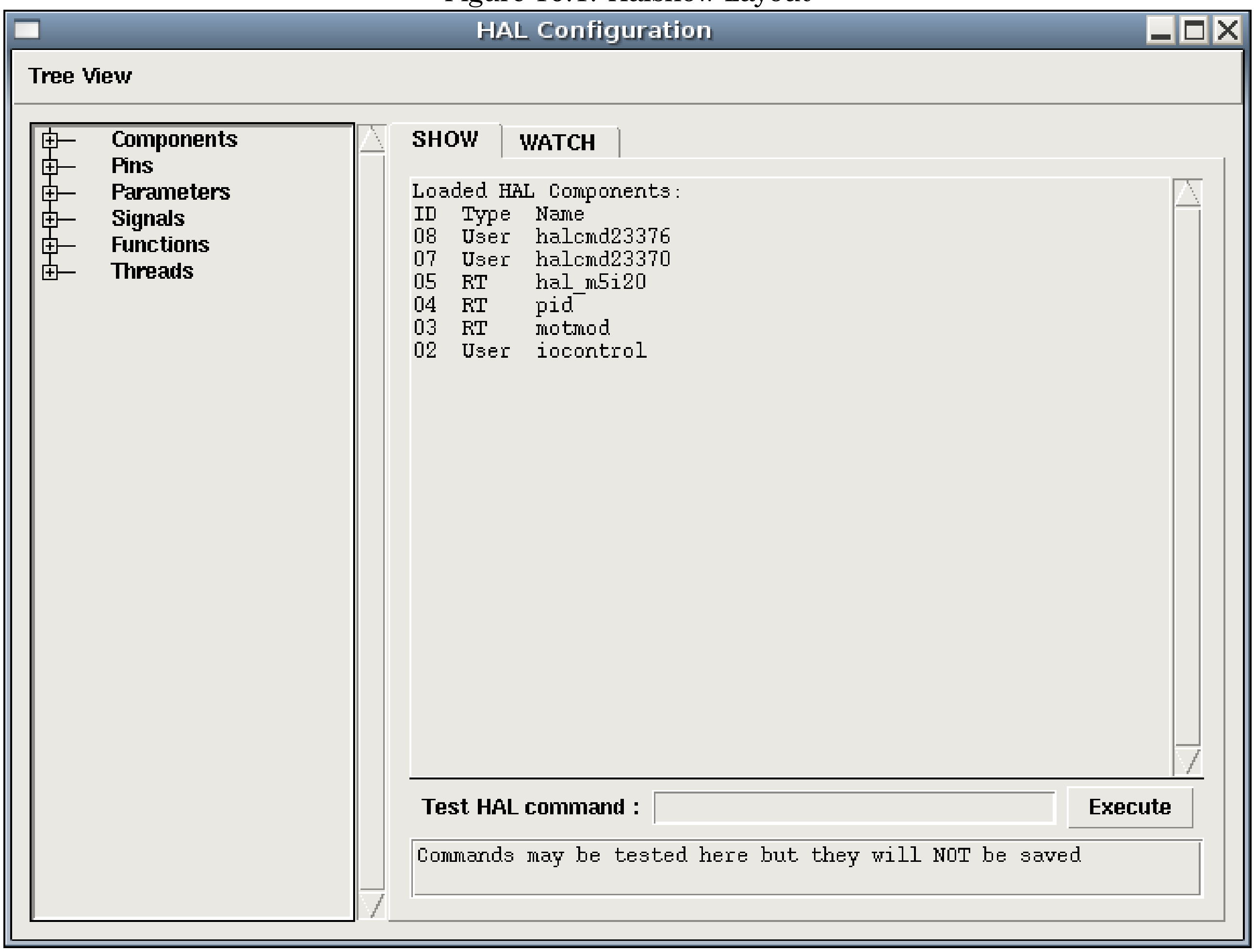
The script halshow can help you find your away around a running HAL. This is a very specialized system and it must connect to a working HAL. It can not run stand alone because it relies on the ability of HAL to report what it knows of itself through the halcmd interface library. It is discovery based. Each time halshow runs with a different EMC configuration it will be different.

As we will soon see, this ability of HAL to document itself is one key to making an effective CNC system.

10.4.1 Hal Tree Area

At the left of its display as shown in figure 10.1 is a tree view, somewhat like you might see with some file browsers. At the right is a tabbed notebook with tabs for show and watch.

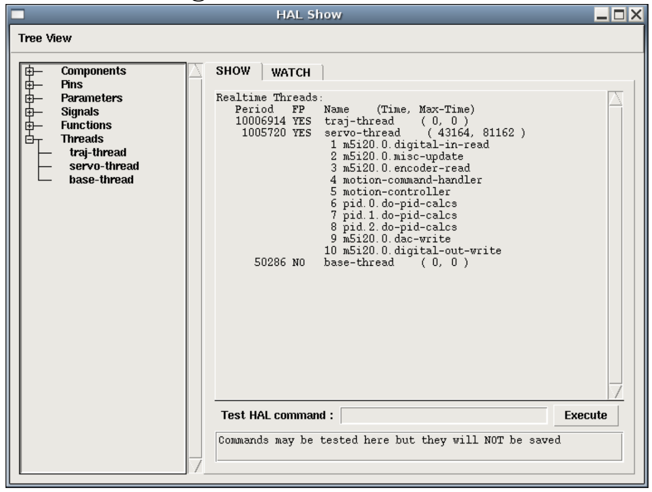
Figure 10.1: Halshow Layout



The tree shows all of the major parts of a HAL. In front of each is a small plus (+) or minus (-) sign in a box. Clicking the plus will expand that tree node to display what is under it. If that box shows a minus sign clicking it will close that section of the tree.

You can also expand or collapse the tree display using the Tree View menu at the upper left edge of the display. This menu is shown in figure xx

Figure 10.2: Show Menu



10.4.2 Hal Show Area

Clicking on the node name rather than its plus or minus sign, the word "Components" for example, will show you all that hal knows about the contents of it. Figure 10.1 shows a list exactly like you will see if you click the "Components" name while you are running a standard m5i20 servo card. The information display is exactly like those shown in traditional text based HAL analysis tools. The advantage here is that we have mouse click access. Access that can be as broad or as focused as you need.

If we take a closer look at the tree display we can see that the six major parts of a HAL can all be expanded at least one level. As these levels are expanded you can get more focused with the reply when you click on the rightmost tree node. You will find that there are some hal pins and parameters that show more than one reply. This is do to the nature of the search routines in halcmd itself. If you search one pin you may get two like this.

```
Component Pins:
Owner Type Dir Value Name
06 bit -W TRUE parport.0.pin-10-in
06 bit -W FALSE parport.0.pin-10-in-not
```

The second pins name contains the complete name of the first.

Below the show area on the right is a set of widgets that will allow you to play with the running HAL. The commands you enter here and the effect that they have on the running HAL are not saved. They will persist as long as the emc remains up but are gone as soon as it is.

The entry box labeled Test Hal Command : will accept any of the commands listed for halcmd. These include;

- loadrt, unloadrt
- addf, delf
- newsig, delsig
- linkpp, linksp, linkps, unlinkp
- setp, sets

This little editor will enter a command any time you press <enter> or push the execute button. An error message from halcmd will show below this entry widget when these commands are not properly formed. If you are not certain how to set up a proper command you'll need to read again the documentation on halcmd and the specific modules that you are working with.

Let's use this editor to add a differential module to a hal and connect it to axis position so that we could see the rate of change in position, ie acceleration. We first need to load a hal module named blocks, add it to the servo thread, then connect it to the position pin of an axis. Once that is done we can find the output of the differentiator in halscope. So let's go. (yes I looked this one up.)

```
loadrt blocks ddt=1
```

Now look at the components node and you should see blocks in there someplace.

```
Loaded HAL Components:
ID Type Name
10 User halcmd29800
09 User halcmd29374
08 RT blocks
06 RT hal_parport
05 RT scope_rt
04 RT stepgen
03 RT motmod
02 User iocontrol
```

Sure enough there it is. Notice that its id is 08. Next we need to find out what functions are available with it so we look at functions.

```
Exported Functions:
Owner CodeAddr Arg FP Users Name
08 E0B97630 E0DC7674 YES 0 ddt.0
03 E0DEF83C 00000000 YES 1 motion-command-handler
03 E0DF0BF3 00000000 YES 1 motion-controller
06 E0B541FE E0DC75B8 NO 1 parport.0.read
06 E0B54270 E0DC75B8 NO 1 parport.0.write
06 E0B54309 E0DC75B8 NO 0 parport.read-all
06 E0B5433A E0DC75B8 NO 0 parport.write-all
05 E0AD712D 00000000 NO 0 scope.sample
04 E0B618C1 E0DC7448 YES 1 stepgen.capture-position
04 E0B612F5 E0DC7448 NO 1 stepgen.make-pulses
04 E0B614AD E0DC7448 YES 1 stepgen.update-freq
```

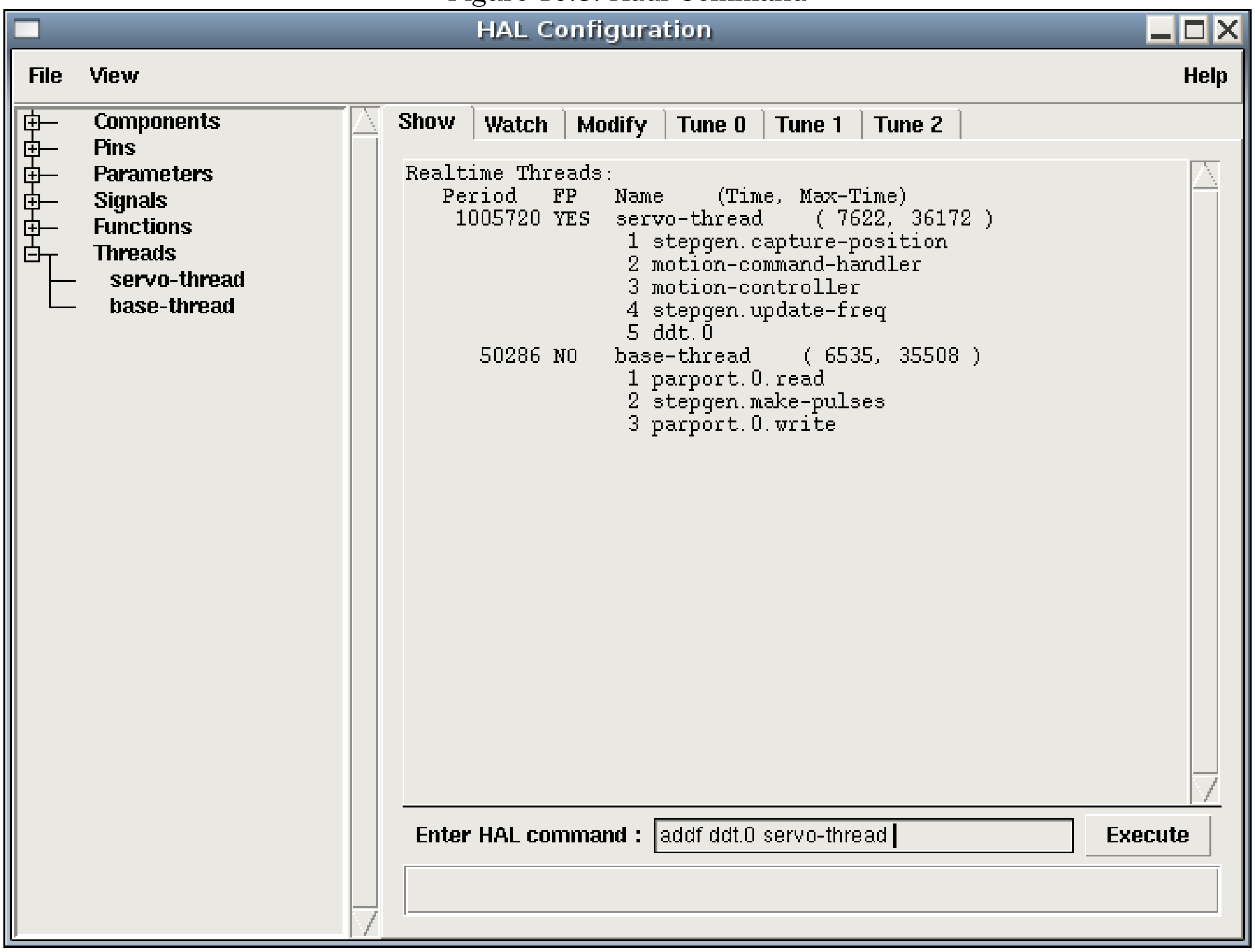
Here we look for owner #08 and see that blocks has exported a function named ddt.0. We should be able to add ddt.0 to the servo thread and it will do its math each time the servo thread is updated. Once again I look up the addf command and find that it uses three arguments like this.

```
addf <functname> <threadname> [<position>]
```

We already know the functname=ddt.0 so let's get the thread name right by expanding the thread node in the tree. Here we see two threads servo-thread and base-thread. The position of ddt.0 in the thread is not critical.

```
addf ddt.0 servo-thread
```

Figure 10.3: Addf Command



This is just for viewing so we leave position blank and get the last position in the thread. Figure 10.3 shows the state of halshow after this command has been issued.

Next we need to connect this block to something. But how do we know what pins are available. The answer is look under pins. There we find ddt and see this.

```
Component Pins:
Owner Type Dir Value Name
08 float R- 0.00000e+00 ddt.0.in
08 float -W 0.00000e+00 ddt.0.out
```

That looks easy enough to understand but what signal or pin do we want to connect to it. It could be an axis pin, a stepgen pin, or a signal. I see this when I look at axis.0.

```
Component Pins:
Owner Type Dir Value Name
03 float -W 0.00000e+00 axis.0.motor-pos-cmd ==> Xpos-cmd
```

So it looks like Xpos-cmd should be a good signal to use. Back to the editor where I enter the following command.

```
linksp Xpos-cmd ddt.0.in
```

Now if I look at the Xpos-cmd signal using the tree node I'll see what I've done.

```
Signals:
Type Value Name
float 0.00000e+00 Xpos-cmd
<== axis.0.motor-pos-cmd
==> ddt.0.in
==> stepgen.0.position-cmd
```

We see that this signal comes from axis.o.motor-pos-cmd and goes to both ddt.0.in and stepgen.0.position-cmd. By connecting our block to the signal we have avoided any complications with the normal flow of this motion command.

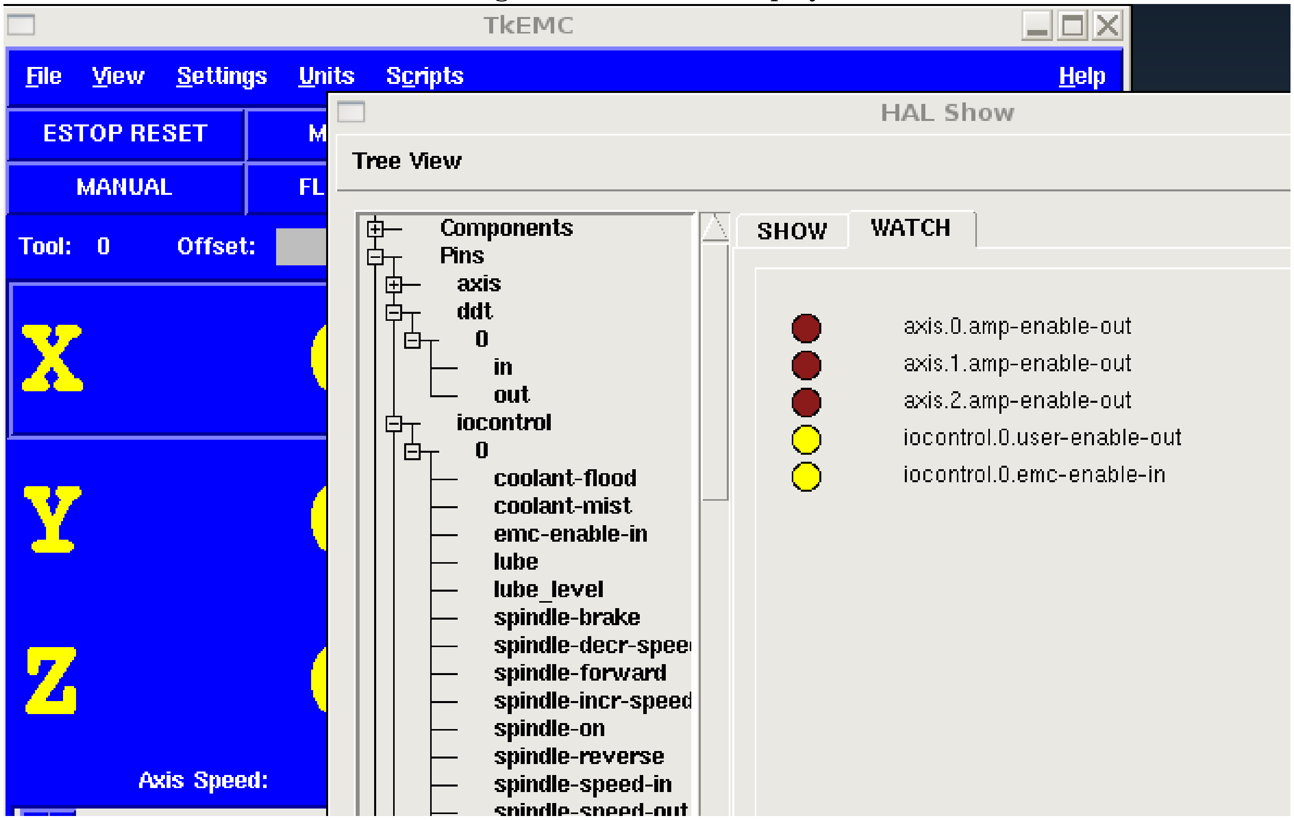
The Hal Show Area uses halcmd to discover what is happening in a running HAL. It gives you complete information about what it has discovered. It also updates as you issue commands from the little editor panel to modify that HAL. There are times when you want a different set of things displayed without all of the information available in this area. That is where the Hal Watch Area is of value.

10.4.3 Hal Watch Area

Clicking the watch tab produces a blank canvas. You can add signals and pins to this canvas and watch their values.[1] You can add signals or pins when the watch tab is displayed by clicking on the name of it. Figure 10.4 shows this canvas with several "bit" type signals. These signals include enable-out for the first three axes and two of the three iocontrol "estop" signals. Notice that the axes are not enabled even though the estop signals say that the EMC is not in estop. A quick look at themc shows that the condition of the EMC is ESTOP RESET. The amp enables do not turn true until the machine has been turned on.

[1]The refresh rate of the watch display is much lower than Halmeter or Halscope. If you need good resolution of the timing of signals these tools are much more effective.

Figure 10.4: Watch Display



The two colors of circles, aka leds, always show dark brown when a bit signal or pin is false. They show the light yellow whenever that signal is true. If you select a pin or signal that is not a bit typed signal, watch will show the numerical value.

Watch will quickly allow you to test switches or see the effect of changes that you make to EMC while using the graphical interface. Watch's refresh rate is a bit slow to see stepper pulses but you can use it for these if you move an axis very slowly or in very small increments of distance. If you've used IO_Show in EMC, the watch page in halshow can be setup to watch a parport much as it did.

Chapter 11

comp: a tool for creating HAL modules

11.1 Introduction

Writing a HAL component can be a tedious process, most of it in setup calls to `rtapi_` and `hal_` functions and associated error checking. *comp* will write all this code for you, automatically.

Compiling a HAL component is also much easier when using *comp*, whether the component is part of the emc2 source tree, or outside it.

For instance, the “ddt” portion of `blocks` is around 80 lines of code. The equivalent component is very short when written using the *comp* preprocessor:

```
component ddt "Compute the derivative of the input function";
pin in float in;
pin out float out;
variable float old;
function _;
license "GPL";
;;
float tmp = in;
out = (tmp - old) / fperiod;
old = tmp;
```

and it can be compiled and installed very easily: by simply placing `ddt.comp` in `src/hal/components` and running ’`make`’, or by placing it anywhere on the system and running `comp --install ddt.comp`

11.2 Definitions

component A component is a single real-time module, which is loaded with `halcmd loadrt`. One `.comp` file specifies one component.

instance A component can have zero or more instances. Each instance of a component is created equal (they all have the same pins, parameters, functions, and data) but behave independently when their pins, parameters, and data have different values.

singleton It is possible for a component to be a ’singleton’, in which case exactly one instance is created. It seldom makes sense to write a `singleton` component, unless there can literally only be a single object of that kind in the system (for instance, a component whose purpose is to provide a pin with the current UNIX time, or a hardware driver for the internal PC speaker)

11.3 Instance creation

For a singleton, the one instance is created when the component is loaded.

For a non-singleton, the 'count' module parameter determines how many numbered instances are created.

11.4 Syntax

A .comp file consists of a number of declarations, followed by ;; on a line of its own, followed by C code implementing the module's functions.

Declarations include:

- component *HALNAME (DOC)*;
- pin *PINDIRECTION TYPE HALNAME* ([*SIZE*]|[*MAXSIZE* : *CONDSIZE*]) (if *CONDITION*) (= *STARTVALUE*) (*DOC*);
- param *PARAMDIRECTION TYPE HALNAME* ([*SIZE*]|[*MAXSIZE* : *CONDSIZE*]) (if *CONDITION)* (= *STARTVALUE*) (*DOC*) ;
- function *HALNAME* (fp | nofp) (*DOC*);
- option *OPT* (*VALUE*);
- variable *CTYPE NAME* ([*SIZE*]);
- description DOC;
- see_also DOC;
- license LICENSE;

Parentheses indicate optional items. A vertical bar indicates alternatives. Words in *CAPITALS* indicate variable text, as follows:

HALNAME An identifier.

When used to create a HAL identifier, any underscores are replaced with dashes, and any trailing dash or period is removed, so that "this_name_" will be turned into "this-name", and if the name is "_", then a trailing period is removed as well, so that "function _" gives a HAL function name like component.<num> instead of component.<num>.

If present, the prefix hal_ is removed from the beginning of the component name when creating pins, parameters and functions.

In the HAL identifier for a pin or parameter, # denotes an array item, and must be used in conjunction with a [SIZE] declaration. The hash marks are replaced with a 0-padded number with the same length as the number of # characters.

When used to create a C identifier, the following changes are applied to the HALNAME:

1. Any # characters, and any ".", "_" or "-" characters immediately before them, are removed.
2. Any remaining "." and "-" characters are replaced with "_"
3. Repeated "_" characters are changed to a single "_" character.

A trailing _ is retained, so that HAL identifiers which would otherwise collide with reserved names or keywords (e.g., 'min') can be used.

HALNAME	C Identifier	HAL Identifier
x_y_z	x_y_z	x-y-z
x-y.z	x_y_z	x-y.z
x_y_z_	x_y_z	x-y-z
x.##.y	x_y(MM)	x.MM.z
x.##	x(MM)	x.MM

if CONDITION An expression involving the variable *personality* which is nonzero when the pin or parameter should be created

SIZE A number that gives the size of an array. The array items are numbered from 0 to *SIZE*-1.

MAXSIZE : CONDSIZE A number that gives the maximum size of the array followed by an expression involving the variable *personality* and which always evaluates to less than *MAXSIZE*. When the array is created its size will be *CONDSIZE*.

DOC A string that documents the item. String can be a C-style "double quoted" string, like `"Selects the desired edge: TRUE means falling, FALSE means rising"` or a Python-style "triple quoted" string, which may include embedded newlines and quote characters, such as:

```
param rw bit zot=TRUE
"""The effect of this parameter, also known as "the orb of zot",
will require at least two paragraphs to explain.

Hopefully these paragraphs have allowed you to understand "zot"
better.""";
```

The documentation string is in "groff -man" format. For more information on this markup format, see `groff_man(7)`. Remember that comp interprets backslash escapes in strings, so for instance to set the italic font for the word *example*, write `"\\fIexample\\fB"`.

TYPE One of the HAL types: `bit`, `signed`, `unsigned`, or `float`. The old names `s32` and `u32` may also be used, but `signed` and `unsigned` are preferred.

PINDIRECTION One of the following: `in`, `out`, or `io`. A component sets a value for an `out` pin, it reads a value from an `in` pin, and it may read or set the value of an `io` pin.

PARAMDIRECTION One of the following: `r` or `rw`. A component sets a value for a `r` parameter, and it may read or set the value of a `rw` parameter.

STARTVALUE Specifies the initial value of a pin or parameter. If it is not specified, then the default is `0` or `FALSE`, depending on the type of the item.

fp Indicates that the function performs floating-point calculations.

nofp Indicates that it only performs integer calculations. If neither is specified, `fp` is assumed. Neither comp nor gcc can detect the use of floating-point calculations in functions that are tagged `nofp`.

OPT, VALUE Depending on the option name OPT, the valid VALUEs vary. The currently defined options are:

option singleton yes (default: no)

Do not create a `count` module parameter, and always create a single instance. With `singleton`, items are named `component-name.item-name` and without `singleton`, items for numbered instances are named `component-name.<num>.item-name`.

option default_count *number* (default: 1)

Normally, the module parameter `count` defaults to 0. If specified, the `count` will default to this value instead.

option count_function yes (default: no)

Normally, the number of instances to create is specified in the module parameter `count`; if `count_function` is specified, the value returned by the function `int get_count(void)` is used instead, and the `count` module parameter is not defined.

option rtapi_app no (default: yes)

Normally, the functions `rtapi_app_main` and `rtapi_app_exit` are automatically defined. With `option rtapi_app no`, they are not, and must be provided in the C code.

When implementing your own `rtapi_app_main`, call the function `int export(char *prefix, long extra_arg)` to register the pins, parameters, and functions for `prefix`.

option data *type* (default: none) **DEPRECATED**

If specified, each instance of the component will have an associated data block of *type* (which can be a simple type like `float` or the name of a type created with `typedef`).

In new components, *variable* should be used instead.

option extra_setup yes (default: no)

If specified, call the function defined by `EXTRA_SETUP` for each instance. If using the automatically defined `rtapi_app_main`, `extra_arg` is the number of this instance.

option extra_cleanup yes (default: no)

If specified, call the function defined by `EXTRA_CLEANUP` from the automatically defined `rtapi_app_exit`, or if an error is detected in the automatically defined `rtapi_app_main`.

option userspace yes (default: no)

If specified, this file describes a userspace component, rather than a real one. A userspace component may not have functions defined by the `function` directive. Instead, after all the instances are constructed, the C function `user_mainloop()` is called. When this function returns, the component exits. Typically, `user_mainloop()` will use `FOR_ALL_INSTS()` to perform the update action for each instance, then sleep for a short time. Another common action in `user_mainloop()` may be to call the event handler loop of a GUI toolkit.

option userinit yes (default: no)

If specified, the function `userinit(argc,argv)` is called before `rtapi_app_main()` (and thus before the call to `hal_init()`). This function may process the commandline arguments or take other actions. Its return type is `void`; it may call `exit()` if it wishes to terminate rather than create a hal component (for instance, because the commandline arguments were invalid).

If an option's VALUE is not specified, then it is equivalent to specifying `option ... yes`. The result of assigning an inappropriate value to an option is undefined. The result of using any other option is undefined.

LICENSE Specify the license of the module, for the documentation and for the MODULE_LICENSE() module declaration.

11.5 Per-instance data storage

variable *CTYPE NAME*;

variable *CTYPE NAME*[*SIZE*];

variable *CTYPE NAME* = *DEFAULT*;

variable *CTYPE NAME*[*SIZE*] = *DEFAULT*;

Declare a per-instance variable *NAME* of type *CTYPE*, optionally as an array of *SIZE* items, and optionally with a default value *DEFAULT*. Items with no *DEFAULT* are initialized to all-bits-zero. *CTYPE* is a simple one-word C type, such as `float`, `u32`, `s32`, etc.

C++-style one-line comments (`// ...`) and C-style multi-line comments (`/* ... */`) are both supported in the declaration section.

11.6 Other restrictions on comp files

Though HAL permits a pin, a parameter, and a function to have the same name, comp does not.

11.7 Convenience Macros

Based on the items in the declaration section, *comp* creates a C structure called `struct state`. However, instead of referring to the members of this structure (e.g., `*(inst->name)`), they will generally be referred to using the macros below. The details of `struct state` and these macros may change from one version of *comp* to the next.

FUNCTION(name) Use this macro to begin the definition of a realtime function which was previously declared with '`function NAME`'. The function includes a parameter '`period`' which is the integer number of nanoseconds between calls to the function.

EXTRA_SETUP() Use this macro to begin the definition of the function called to perform extra setup of this instance. Return a negative Unix `errno` value to indicate failure (e.g., `return -EBUSY` on failure to reserve an I/O port), or 0 to indicate success.

EXTRA_CLEANUP() Use this macro to begin the definition of the function called to perform extra cleanup of the component. Note that this function must clean up all instances of the component, not just one. The 'pin_name', 'parameter_name', and 'data' macros may not be used here.

pin_name

parameter_name For each pin `pin_name` or param `parameter_name` there is a macro which allows the name to be used on its own to refer to the pin or parameter.

When `pin_name` or `parameter_name` is an array, the macro is of the form *pin_name(idx)* or *param_name(idx)* where *idx* is the index into the pin array. When the array is a variable-sized array, it is only legal to refer to items up to its *condsize*.

When the item is a conditional item, it is only legal to refer to it when its *condition* evalued to a nonzero value.

variable_name For each variable `variable_name` there is a macro which allows the name to be used on its own to refer to the variable. When `variable_name` is an array, the normal C-style subscript is used: *variable_name[idx]*

data If 'option data' is specified, this macro allows access to the instance data.

fperiod The floating-point number of seconds between calls to this realtime function.

FOR_ALL_INSTS() `{...}` For userspace components. This macro uses the variable `struct state *inst` to iterate over all the defined instances. Inside the body of the loop, the ***pin_name***, ***parameter_name***, and **data** macros work as they do in realtime functions.

11.8 Components with one function

If a component has only one function and the string "FUNCTION" does not appear anywhere after `;;`, then the portion after `;;` is all taken to be the body of the component's single function.

11.9 Component "Personality"

If a component has any pins or parameters with an "if *condition*" or "[*maxsize : condsize*]", it is called a component with "*personality*". The "personality" of each instance is specified when the module is loaded. "Personality" can be used to create pins only when needed. For instance, personality is used in the `logic` component, to allow for a variable number of input pins to each logic gate and to allow for a selection of any of the basic boolean logic functions **and**, **or**, and **xor**.

11.10 Compiling `.comp` files in the source tree

Place the `.comp` file in the source directory `emc2/src/hal/components` and re-run `make`. `Comp` files are automatically detected by the build system.

If a `.comp` file is a driver for hardware, it may be placed in `emc2/src/hal/components` and will be built except if emc2 is configured as a userspace simulator.

11.11 Compiling realtime components outside the source tree

`comp` can process, compile, and install a realtime component in a single step, placing `rtexample.ko` in the emc2 realtime module directory:

```
comp --install rtexample.comp
```

Or, it can process and compile in one step, leaving `example.ko` (or `example.so` for the simulator) in the current directory:

```
comp --compile rtexample.comp
```

Or it can simply process, leaving `example.c` in the current directory:

```
comp rtexample.comp
```

`comp` can also compile and install a component written in C, using the `--install` and `--compile` options shown above:

```
comp --install rtexample2.c
```

man-format documentation can also be created from the information in the declaration section:

```
comp --document rtexample.comp
```

The resulting manpage, `example.9` can be viewed with

```
man ./example.9
```

or copied to a standard location for manual pages.

11.12 Compiling userspace components outside the source tree

`comp` can process, compile, install, and document userspace components:

```
comp usrexample.comp
comp --compile usrexample.comp
comp --install usrexample.comp
comp --document usrexample.comp
```

This only works for `.comp` files, not for `.c` files.

11.13 Examples

11.13.1 constant

This component functions like the one in 'blocks', including the default value of 1.0. The declaration "function _" creates functions named 'constant.0', etc.

```
component constant;
pin out float out;
param r float value = 1.0;
function _;
option extra_setup yes;
;;
FUNCTION(_) { out = value; }
```

11.13.2 sincos

This component computes the sine and cosine of an input angle in radians. It has different capabilities than the 'sine' and 'cosine' outputs of siggen, because the input is an angle, rather than running freely based on a 'frequency' parameter.

The pins are declared with the names `sin_` and `cos_` in the source code so that they do not interfere with the functions `sin()` and `cos()`. The HAL pins are still called `sincos.<num>.sin`.

```
component sincos;
pin out float sin_ out;
pin out float cos_ out;
pin in float theta in;
function _;
;;
#include <rtapi_math.h>
FUNCTION(_) { sin_ = sin(theta); cos_ = cos(theta); }
```

11.13.3 out8

This component is a driver for a *fictional* card called "out8", which has 8 pins of digital output which are treated as a single 8-bit value. There can be a varying number of such cards in the system, and they can be at various addresses. The pin is called `out_` because `out` is an identifier used in `<asm/io.h>`. It illustrates the use of `EXTRA_SETUP` and `EXTRA_CLEANUP` to request an I/O region and then free it in case of error or when the module is unloaded.

```
component out8;
pin out unsigned out_ "Output value; only low 8 bits are used";
param r unsigned ioaddr;

function _;

option count_function;
option extra_setup;
option extra_cleanup;
option constructable no;

;;
#include <asm/io.h>

#define MAX 8
int io[MAX] = {0,};
RTAPI_MP_ARRAY_INT(io, MAX, "I/O addresses of out8 boards");

int get_count(void) {
    int i = 0;
    for(i=0; i<MAX && io[i]; i++) { /* Nothing */ }
    return i;
}

EXTRA_SETUP() {
    if(!rtapi_request_region(io[extra_arg], 1, "out8")) {
// set this I/O port to 0 so that EXTRA_CLEANUP does not release the IO
// ports that were never requested.
        io[extra_arg] = 0;
        return -EBUSY;
    }
    ioaddr = io[extra_arg];
    return 0;
}

EXTRA_CLEANUP() {
    int i;
    for(i=0; i < MAX && io[i]; i++) {
        rtapi_release_region(io[i], 1);
    }
}

FUNCTION(_) { outb(out_, ioaddr); }
```

11.13.4 hal_loop

```
component hal_loop;
pin out float example;
```

This fragment of a component illustrates the use of the `hal_` prefix in a component name. `loop` is the name of a standard Linux kernel module, so a `loop` component might not successfully load if the Linux `loop` module was also present on the system.

When loaded, `halcmd show comp` will show a component called `hal_loop`. However, the pin shown by `halcmd show pin` will be `loop.0.example`, not `hal-loop.0.example`.

11.13.5 arraydemo

This realtime component illustrates use of fixed-size arrays:

```
component arraydemo "4-bit Shift register";
pin in bit in;
pin out bit out-# [4];
function _ nofp;
;;
int i;
for(i=3; i>0; i--) out(i) = out(i-1);
out(0) = in;
```

11.13.6 rand

This userspace component changes the value on its output pin to a new random value in the range $[0, 1)$ about once every 1ms.

```
component rand;
option userspace;

pin out float out;
;;
#include <unistd.h>

void user_mainloop(void) {
    while(1) {
        usleep(1000);
        FOR_ALL_INSTS() out = drand48();
    }
}
```

11.13.6.1 logic

This realtime component shows how to use “personality” to create variable-size arrays and optional pins.

```
component logic;
pin in bit in-##[16 : personality & 0xff];
pin out bit and if personality & 0x100;
pin out bit or if personality & 0x200;
pin out bit xor if personality & 0x400;
function _ nofp;
description """
Experimental general ‘logic function’ component.  Can perform ‘and’, ‘or’
and ‘xor’ of up to 16 inputs.  Determine the proper value for ‘personality’
by adding:
.IP \\(bu 4
The number of input pins, usually from 2 to 16
.IP \\(bu
256 (0x100)  if the ‘and’ output is desired
.IP \\(bu
512 (0x200)  if the ‘or’ output is desired
```

```
.IP \\(bu
1024 (0x400)  if the ‘xor’ (exclusive or) output is desired""";
license "GPL";
;;
FUNCTION(_) {
    int i, a=1, o=0, x=0;
    for(i=0; i < (personality & 0xff); i++) {
        if(in(i)) { o = 1; x = !x; }
        else { a = 0; }
    }
    if(personality & 0x100) and = a;
    if(personality & 0x200) or = o;
    if(personality & 0x400) xor = x;
}
```

A typical load line for this component might be

```
loadrt logic count=3 personality=0x102,0x305,0x503
```

which creates the following pins:

- A 2-input AND gate: logic.0.and, logic.0.in-00, logic.0.in-01
- 5-input AND and OR gates: logic.1.and, logic.1.or, logic.1.in-00, logic.1.in-01, logic.1.in-02, logic.1.in-03, logic.1.in-04,
- 3-input AND and XOR gates: logic.2.and, logic.2.xor, logic.2.in-00, logic.2.in-01, logic.2.in-02

Chapter 12

Creating Userspace Python Components with the 'hal' module

12.1 Basic usage

A userspace component begins by creating its pins and parameters, then enters a loop which will periodically drive all the outputs from the inputs. The following component copies the value seen on its input pin (`passthrough.in`) to its output pin (`passthrough.out`) approximately once per second.

```
#!/usr/bin/python
import hal, time
h = hal.component("passthrough")
h.newpin("in", hal.HAL_FLOAT, hal.HAL_IN)
h.newpin("out", hal.HAL_FLOAT, hal.HAL_OUT)
h.ready()
try:
    while 1:
        time.sleep(1)
        h['out'] = h['in']
except KeyboardInterrupt:
    raise SystemExit
```

Copy the above listing into a file named "passthrough", make it executable (`chmod +x`), and place it on your `$PATH`. Then try it out:

```
$ halrun
halcmd: loadusr passthrough
halcmd: show pin
Component Pins:
Owner  Type  Dir     Value      Name
 03    float IN              0  passthrough.in
 03    float OUT             0  passthrough.out
halcmd: setp passthrough.in 3.14
halcmd: show pin
Component Pins:
Owner  Type  Dir     Value      Name
 03    float IN           3.14  passthrough.in
 03    float OUT          3.14  passthrough.out
```

12.2 Userspace components and delays

If you typed "show pin" quickly, you may see that `passthrough.out` still had its old value of 0. This is because of the call to 'time.sleep(1)', which makes the assignment to the output pin occur at most once per second. Because this is a userspace component, the actual delay between assignments can be much longer–for instance, if the memory used by the passthrough component is swapped to disk, the assignment could be delayed until that memory is swapped back in.

Thus, userspace components are suitable for user-interactive elements such as control panels (delays in the range of milliseconds are not noticed, and longer delays are acceptable), but not for sending step pulses to a stepper driver board (delays must always be in the range of microseconds, no matter what).

12.3 Create pins and parameters

```
h = hal.component("passthrough")
```

The component itself is created by a call to the constructor '`hal.component`'. The arguments are the HAL component name and (optionally) the prefix used for pin and parameter names. If the prefix is not specified, the component name is used.

```
h.newpin("in", hal.HAL_FLOAT, hal.HAL_IN)
```

Then pins are created by calls to methods on the component object. The arguments are: pin name suffix, pin type, and pin direction. For parameters, the arguments are: parameter name suffix, parameter type, and parameter direction.

Table 12.1: HAL Option Names

Pin and Parameter Types:	HAL_BIT	HAL_FLOAT	HAL_S32	HAL_U32
Pin Directions:	HAL_IN	HAL_OUT	HAL_IO	
Parameter Directions:	HAL_RO	HAL_RW		

The full pin or parameter name is formed by joining the prefix and the suffix with a ".", so in the example the pin created is called `passthrough.in`.

```
h.ready()
```

Once all the pins and parameters have been created, call the `.ready()` method.

12.3.1 Changing the prefix

The prefix can be changed by calling the `.setprefix()` method. The current prefix can be retrieved by calling the `.getprefix()` method.

12.4 Reading and writing pins and parameters

For pins and parameters which are also proper Python identifiers, the value may be accessed or set using the attribute syntax:

```
h.out = h.in
```

For all pins, whether or not they are also proper Python identifiers, the value may be accessed or set using the subscript syntax:

```
h['out'] = h['in']
```

12.4.1 Driving output (HAL_OUT) pins

Periodically, usually in response to a timer, all HAL_OUT pins should be "driven" by assigning them a new value. This should be done whether or not the value is different than the last one assigned. When a pin is connected to a signal, its old output value is not copied into the signal, so the proper value will only appear on the signal once the component assigns a new value.

12.4.2 Driving bidirectional (HAL_IO) pins

The above rule does not apply to bidirectional pins. Instead, a bidirectional pin should only be driven by the component when the component wishes to change the value. For instance, in the canonical encoder interface, the encoder component only sets the **index-enable** pin to **FALSE** (when an index pulse is seen and the old value is **TRUE**), but never sets it to **TRUE**. Repeatedly driving the pin **FALSE** might cause the other connected component to act as though another index pulse had been seen.

12.5 Exiting

A "`halcmd unload`" request for the component is delivered as a `KeyboardInterrupt` exception. When an unload request arrives, the process should either exit in a short time, or call the `.exit()` method on the component if substantial work (such as reading or writing files) must be done to complete the shutdown process.

12.6 Project ideas

- Create an external control panel with buttons, switches, and indicators. Connect everything to a microcontroller, and connect the microcontroller to the PC using a serial interface. Python has a very capable serial interface module called pyserial `http://pyserial.sourceforge.net/` (Ubuntu package name "python-serial", in the universe repository)
- Attach a LCDProc `http://lcdproc.omnipotent.net/`-compatible LCD module and use it to display a digital readout with information of your choice (Ubuntu package name "lcdproc", in the universe repository)
- Create a virtual control panel using any GUI library supported by Python (gtk, qt, wxwindows, etc)

Part V

EMC related HAL

Chapter 13

Basic configurations for a stepper based system

13.1 Introduction

This chapter describes 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`.

13.2 Maximum step rate

With software step generation, the maximum step rate is one step per two BASE_PERIODs for step-and-direction output. The maximum requested step rate is the product of an axis's MAX_VELOCITY and its INPUT_SCALE. If the requested step rate is not attainable, following errors will occur, particularly during fast jogs and G0 moves.

If your stepper driver can accept quadrature input, use this mode. With a quadrature signal, one step is possible for each BASE_PERIOD, doubling the maximum step rate.

The other remedies are to decrease one or more of: the BASE_PERIOD (setting this too low will cause the machine to become unresponsive or even lock up), the INPUT_SCALE (if you can select different step sizes on your stepper driver, change pulley ratios, or leadscrew pitch), or the MAX_VELOCITY and STEPGEN_MAXVEL.

If no valid combination of BASE_PERIOD, INPUT_SCALE, and MAX_VELOCITY is acceptable, then hardware step generation (such as with the emc2-supported Universal Stepper Controller)

13.3 Pinout

One of the majour 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 6.1 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[1]:

```
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.

13.3.1 standard_pinout.hal

This file contains several HAL commands, and usually looks like this:

```
# standard pinout config file for 3-axis steppers
# using a parport for I/O
#
# first load the parport driver
loadrt hal_parport cfg="0x0378"
#
# next connect the parport functions to threads
# read inputs first
addf parport.0.read base-thread 1
# write outputs last
addf parport.0.write base-thread -1
#
# finally connect physical pins to the signals
net Xstep => parport.0.pin-03-out
net Xdir  => parport.0.pin-02-out
net Ystep => parport.0.pin-05-out
net Ydir  => parport.0.pin-04-out
net Zstep => parport.0.pin-07-out
net Zdir  => parport.0.pin-06-out

# create a signal for the estop loopback
net estop-loop iocontrol.0.user-enable-out iocontrol.0.emc-enable-in

# create signals for tool loading loopback
net tool-prep-loop iocontrol.0.tool-prepare iocontrol.0.tool-prepared
net tool-change-loop iocontrol.0.tool-change iocontrol.0.tool-changed

# connect "spindle on" motion controller pin to a physical pin
net spindle-on motion.spindle-on => parport.0.pin-09-out

###
### You might use something like this to enable chopper drives when machine ON
### the Xen signal is defined in core_stepper.hal
###
```

[1] Note: we are only presenting one axis to keep it short, all others are similar.
[2] Refer to section 15.1 for additional information

```
# net Xen => parport.0.pin-01-out

###
### If you want active low for this pin, invert it like this:
###

# setp parport.0.pin-01-out-invert 1

###
### A sample home switch on the X axis (axis 0).  make a signal,
### link the incoming parport pin to the signal, then link the signal
### to EMC's axis 0 home switch input pin
###

# net Xhome parport.0.pin-10-in => axis.0.home-sw-in

###
### Shared home switches all on one parallel port pin?
### that's ok, hook the same signal to all the axes, but be sure to
### set HOME_IS_SHARED and HOME_SEQUENCE in the ini file.  See the
### user manual!
###

# net homeswitches <= parport.0.pin-10-in
# net homeswitches => axis.0.home-sw-in
# net homeswitches => axis.1.home-sw-in
# net homeswitches => axis.2.home-sw-in

###
### Sample separate limit switches on the X axis (axis 0)
###

# net X-neg-limit parport.0.pin-11-in => axis.0.neg-lim-sw-in
# net X-pos-limit parport.0.pin-12-in => axis.0.pos-lim-sw-in

###
### Just like the shared home switches example, you can wire together
### limit switches.  Beware if you hit one, EMC will stop but can't tell
### you which switch/axis has faulted.  Use caution when recovering from this.
###

# net Xlimits parport.0.pin-13-in => axis.0.neg-lim-sw-in axis.0.pos-lim-sw-in
```

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

13.3.2 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 15.1 for details)
2. The read & write functions of the parport driver get assigned to the Base thread [3]

[3] the fastest thread in the EMC2 setup, usually the code gets executed every few microseconds

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

13.3.3 Changing the standard_pinout.hal

If you want to change the standard_pinout.hal file, all you need is a text editor. Open the file and locate the parts you want to change.

If you want for example to change the pin for the X-axis Step & Directions signals, all you need to do is to change the number in the 'parport.0.pin-XX-out' name:

```
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 basicly any other numbers you like.

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

13.3.4 Changing the polarity of a signal

If external hardware expects an "active low" signal, set the corresponding `-invert` parameter. For instance, to invert the spindle control signal:

```
setp parport.0.pin-09-invert TRUE
```

13.3.5 Adding PWM Spindle Speed Control

If your spindle can be controlled by a PWM signal, use the `pwmgen` component to create the signal:

```
loadrt pwmgen output_type=0
addf pwmgen.update servo-thread
addf pwmgen.make-pulses base-thread
net spindle-speed-cmd motion.spindle-speed-out => pwmgen.0.value
net spindle-on motion.spindle-on => pwmgen.0.enable
net spindle-pwm pwmgen.0.pwm => parport.0.pin-09-out
setp pwmgen.0.scale 1800 # Change to your spindle's top speed in RPM
```

This assumes that the spindle controller's response to PWM is simple: 0% PWM gives 0RPM, 10% PWM gives 180 RPM, etc. If there is a minimum PWM required to get the spindle to turn, follow the example in the *nist-lathe* sample configuration to use a `scale` component.

13.3.6 Adding an enable signal

Some amplifiers (drives) require an enable signal before they accept and command movement of the motors. For this reason there are already defined signals called 'Xen', 'Yen', 'Zen'.

To connect them use the following example:

```
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.

13.3.7 Adding an external ESTOP button

As you can see in 13.3.1 by default the stepper configuration assumes no external ESTOP button. [4]

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[5], EMC2 will be in the ESTOP state. When the external button gets released EMC2 will imediately 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.

[4] An extensive explanation of hooking up ESTOP circuitry is explained in the wiki.linuxcnc.org and in the Integrator Manual

[5] make sure you use a maintained switch for ESTOP.

Chapter 14

Internal Components

Most components have unix-style manual pages. To view manual pages for real-time components, type "man 9 *componentname*" at the terminal prompt.

This document focuses on more complicated components which have figures which are hard to reproduce in the manual page format.

14.1 Stepgen

This component provides software based generation of step pulses in response to position or velocity commands. In position mode, it has a built in pre-tuned position loop, so PID tuning is not required. In velocity mode, it drives a motor at the commanded speed, while obeying velocity and acceleration limits. It is a realtime component only, and depending on CPU speed, etc, is capable of maximum step rates of 10kHz to perhaps 50kHz. Figure 14.1 shows three block diagrams, each is a single step pulse generator. The first diagram is for step type '0', (step and direction). The second is for step type '1' (up/down, or pseudo-PWM), and the third is for step types 2 through 14 (various stepping patterns). The first two diagrams show position mode control, and the third one shows velocity mode. Control mode and step type are set independently, and any combination can be selected.

14.1.1 Installing

```
emc2$ halcmd loadrt stepgen step_type=<type-array> [ctrl_type=<ctrl_array>]
```

`<type-array>` is a series of comma separated decimal integers. Each number causes a single step pulse generator to be loaded, the value of the number determines the stepping type. `<ctrl_array>` is a comma separated series of "**p**" or "**v**" characters, to specify position or velocity mode. **ctrl_type** is optional, if ommitted, all of the step generators will be position mode. For example:

```
emc2# halcmd loadrt stepgen.o step_type=0,0,2 ctrl_type=p,p,v
```

will install three step generators. The first two use step type '0' (step and direction) and run in position mode. The last one uses step type '2' (quadrature) and runs in velocity mode. The default value for `<config-array>` is "`0,0,0`" which will install three type '0' (step/dir) generators. The maximum number of step generators is 8 (as defined by MAX_CHAN in stepgen.c). Each generator is independent, but all are updated by the same function(s) at the same time. In the following descriptions, `<chan>` is the number of a specific generator. The first generator is number 0.

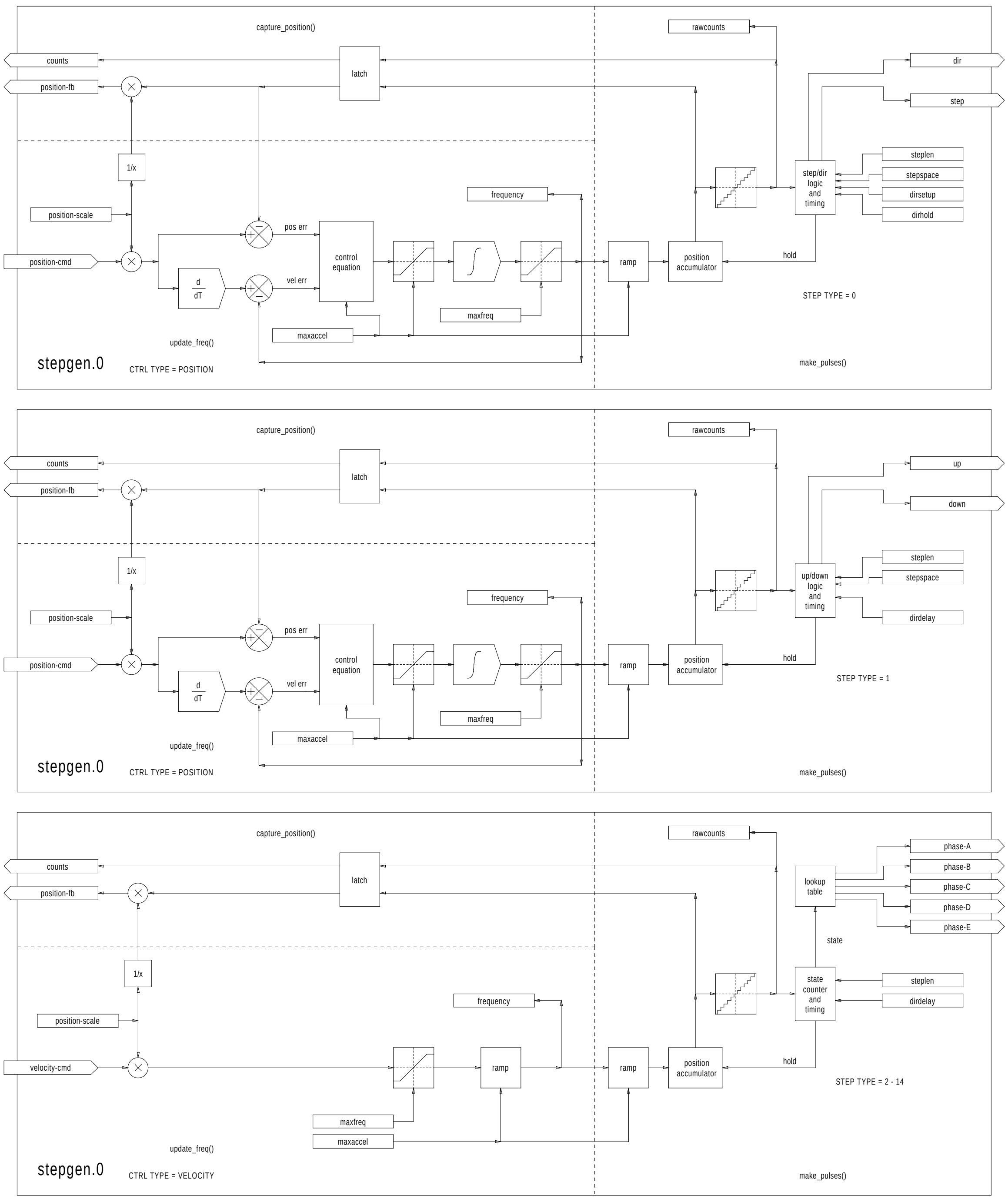


Figure 14.1: Step Pulse Generator Block Diagram (position mode)

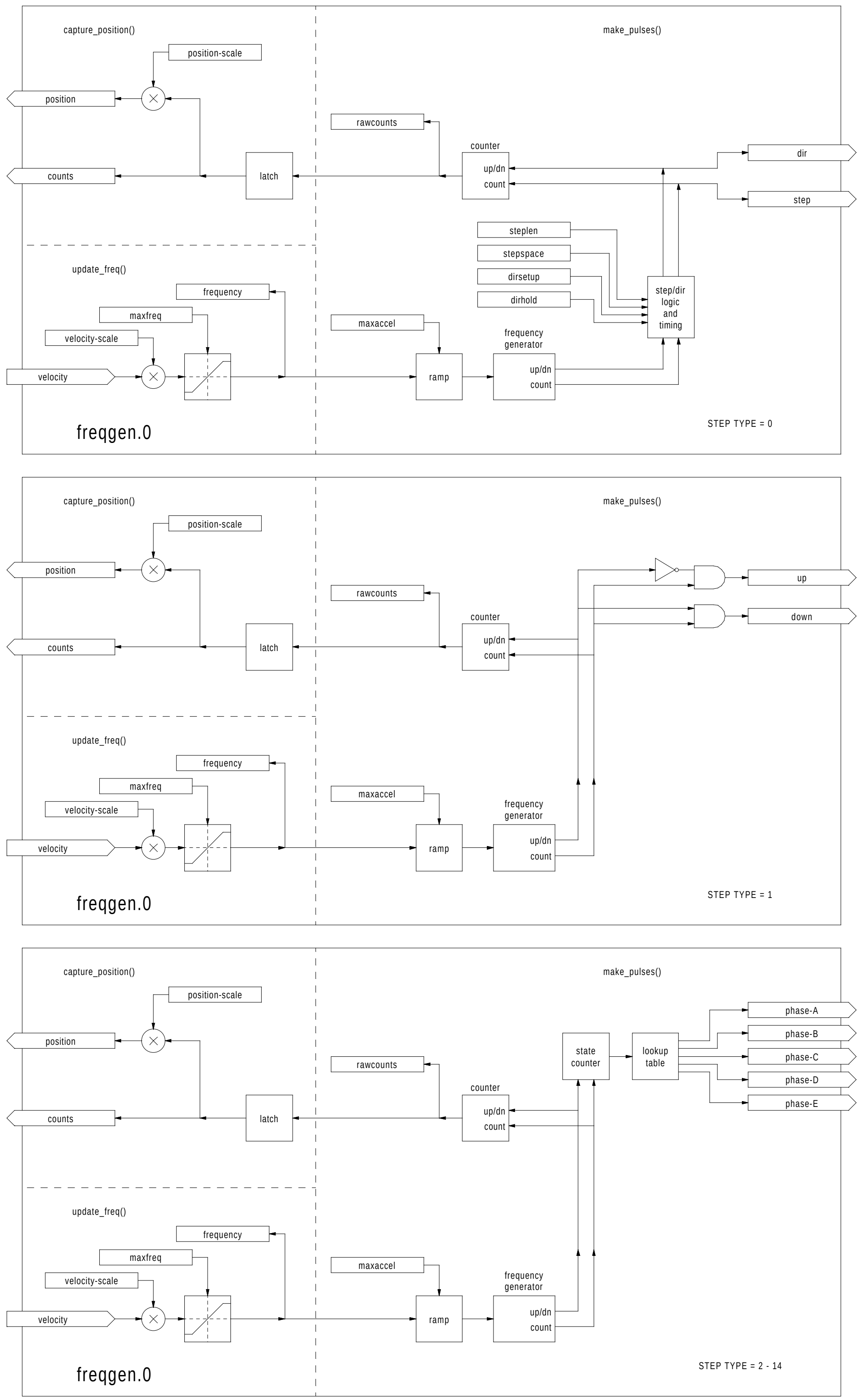


Figure 14.2: Step Pulse Generator Block Diagram (velocity mode)

14.1.2 Removing

```
emc2$ halcmd unloadrt stepgen
```

14.1.3 Pins

Each step pulse generator will have only some of these pins, depending on the step type and control type selected.

- (FLOAT) `stepgen.<chan>.position-cmd` – Desired motor position, in position units (position mode only).
- (FLOAT) `stepgen.<chan>.velocity-cmd` – Desired motor velocity, in position units per second (velocity mode only).
- (S32) `stepgen.<chan>.counts` – Feedback position in counts, updated by `capture_position()`.
- (FLOAT) `stepgen.<chan>.position-fb` – Feedback position in position units, updated by `capture_position()`.
- (BIT) `stepgen.<chan>.step` – Step pulse output (step type 0 only).
- (BIT) `stepgen.<chan>.dir` – Direction output (step type 0 only).
- (BIT) `stepgen.<chan>.up` – UP pseudo-PWM output (step type 1 only).
- (BIT) `stepgen.<chan>.down` – DOWN pseudo-PWM output (step type 1 only).
- (BIT) `stepgen.<chan>.phase-A` – Phase A output (step types 2-14 only).
- (BIT) `stepgen.<chan>.phase-B` – Phase B output (step types 2-14 only).
- (BIT) `stepgen.<chan>.phase-C` – Phase C output (step types 3-14 only).
- (BIT) `stepgen.<chan>.phase-D` – Phase D output (step types 5-14 only).
- (BIT) `stepgen.<chan>.phase-E` – Phase E output (step types 11-14 only).

14.1.4 Parameters

- (FLOAT) `stepgen.<chan>.position-scale` – Steps per position unit. This parameter is used for both output and feedback.
- (FLOAT) `stepgen.<chan>.maxvel` – Maximum velocity, in position units per second. If 0.0, has no effect.
- (FLOAT) `stepgen.<chan>.maxaccel` – Maximum accel/decel rate, in positions units per second squared. If 0.0, has no effect.
- (FLOAT) `stepgen.<chan>.frequency` – The current step rate, in steps per second.
- (FLOAT) `stepgen.<chan>.steplen` – Length of a step pulse (step type 0 and 1) or minimum time in a given state (step types 2-14), in nano-seconds.
- (FLOAT) `stepgen.<chan>.stepspace` – Minimum spacing between two step pulses (step types 0 and 1 only), in nano-seconds.
- (FLOAT) `stepgen.<chan>.dirsetup` – Minimum time from a direction change to the beginning of the next step pulse (step type 0 only), in nanoseconds.
- (FLOAT) `stepgen.<chan>.dirhold` – Minmum time from the end of a step pulse to a direction change (step type 0 only), in nanoseconds.

- (FLOAT) stepgen.<chan>.dirdelay – Minmum time any step to a step in the opposite direction (step types 1-14 only), in nano-seconds.
- (s32) stepgen.<chan>.rawcounts – The raw feedback count, updated by make_pulses().

In position mode, the values of maxvel and maxaccel are used by the internal position loop to avoid generating step pulse trains that the motor cannot follow. When set to values that are appropriate for the motor, even a large instantaneous change in commanded position will result in a smooth trapezoidal move to the new location. The algorithm works by measuring both position error and velocity error, and calculating an acceleration that attempts to reduce both to zero at the same time. For more details, including the contents of the "control equation" box, consult the code.

In velocity mode, maxvel is a simple limit that is applied to the commanded velocity, and maxaccel is used to ramp the actual frequency if the commanded velocity changes abruptly. As in position mode, proper values for these parameters ensure that the motor can follow the generated pulse train.

14.1.5 Step Types

The step generator supports 15 different "step types". Step type 0 is the most familiar, standard step and direction. When configured for step type 0, there are four extra parameters that determine the exact timing of the step and direction signals. See figure 14.3 for the meaning of these parameters. The parameters are in nanoseconds, but will be rounded up to an integer multiple of the thread period for the threaed that calls make_pulses(). For example, if make_pulses() is called every 16uS, and steplen is 20000, then the step pulses will be 2 x 16 = 32uS long. The default value for all four of the parameters is 1nS, but the automatic rounding takes effect the first time the code runs. Since one step requires steplen nS high and stepspace nS low, the maximum frequency is 1,000,000,000 divided by (steplen+stepspace). If maxfreq is set higher than that limit, it will be lowered automatically. If maxfreq is zero, it will remain zero, but the output frequency will still be limited.

Step type 1 has two outputs, up and down. Pulses appear on one or the other, depending on the direction of travel. Each pulse is steplen nS long, and the pulses are separated by at least stepspace nS. The maximum frequency is the same as for step type 0. If maxfreq is set higher than the limit it will be lowered. If maxfreq is zero, it will remain zero but the output frequency will still be limited.

Step types 2 through 14 are state based, and have from two to five outputs. On each step, a state counter is incremented or decremented. Figures 14.4, 14.5, and 14.6 show the output patterns as a function of the state counter. The maximum frequency is 1,000,000,000 divided by steplen, and as in the other modes, maxfreq will be lowered if it is above the limit.

14.1.6 Functions

The component exports three functions. Each function acts on all of the step pulse generators - running different generators in different threads is not supported.

- (FUNCT) stepgen.make-pulses – High speed function to generate and count pulses (no floating point).
- (FUNCT) stepgen.update-freq – Low speed function does position to velocity conversion, scaling and limiting.
- (FUNCT) stepgen.capture-position – Low speed function for feedback, updates latches and scales position.

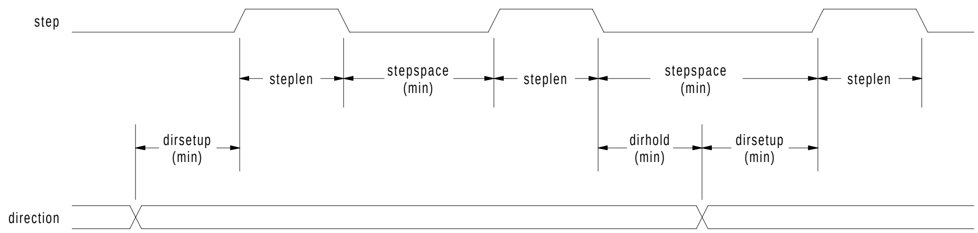


Figure 14.3: Step and Direction Timing

The high speed function `stepgen.make-pulses` should be run in a very fast thread, from 10 to 50uS depending on the capabilities of the computer. That thread's period determines the maximum step frequency, since `steplen`, `stepspace`, `dirsetup`, `dirhold`, and `dirdelay` are all rounded up to a integer multiple of the thread periond in nanoseconds. The other two functions can be called at a much lower rate.

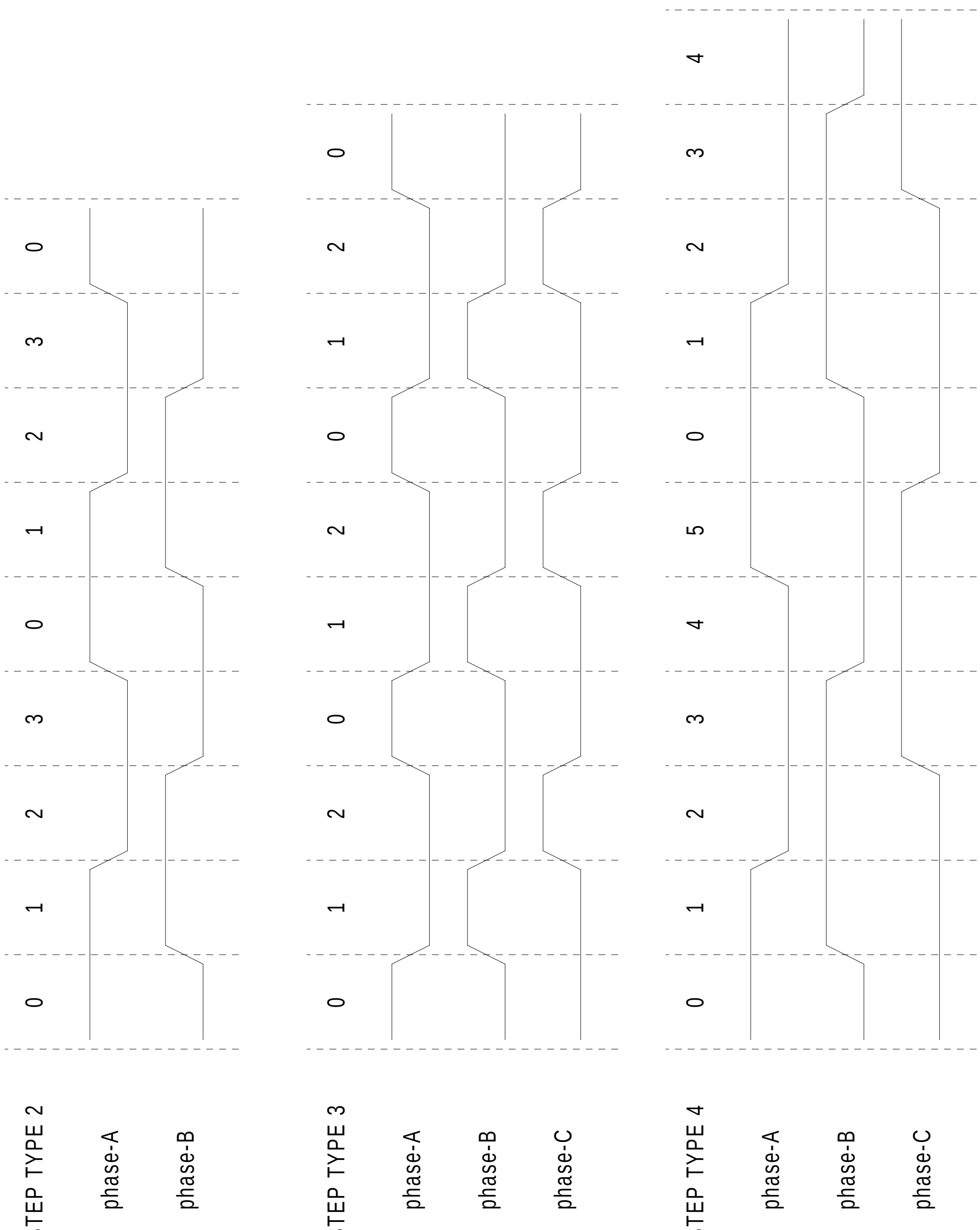


Figure 14.4: Three-Phase step types

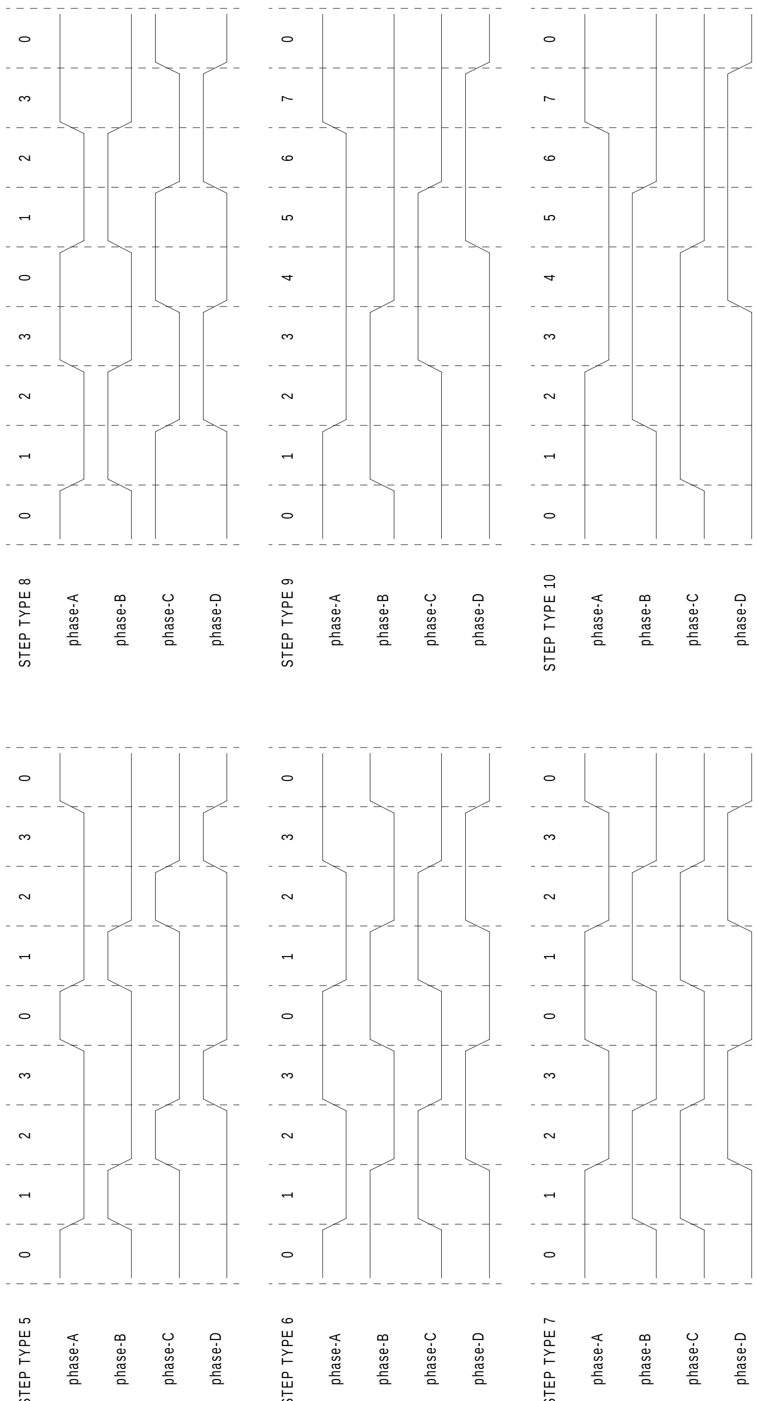


Figure 14.5: Four-Phase Step Types

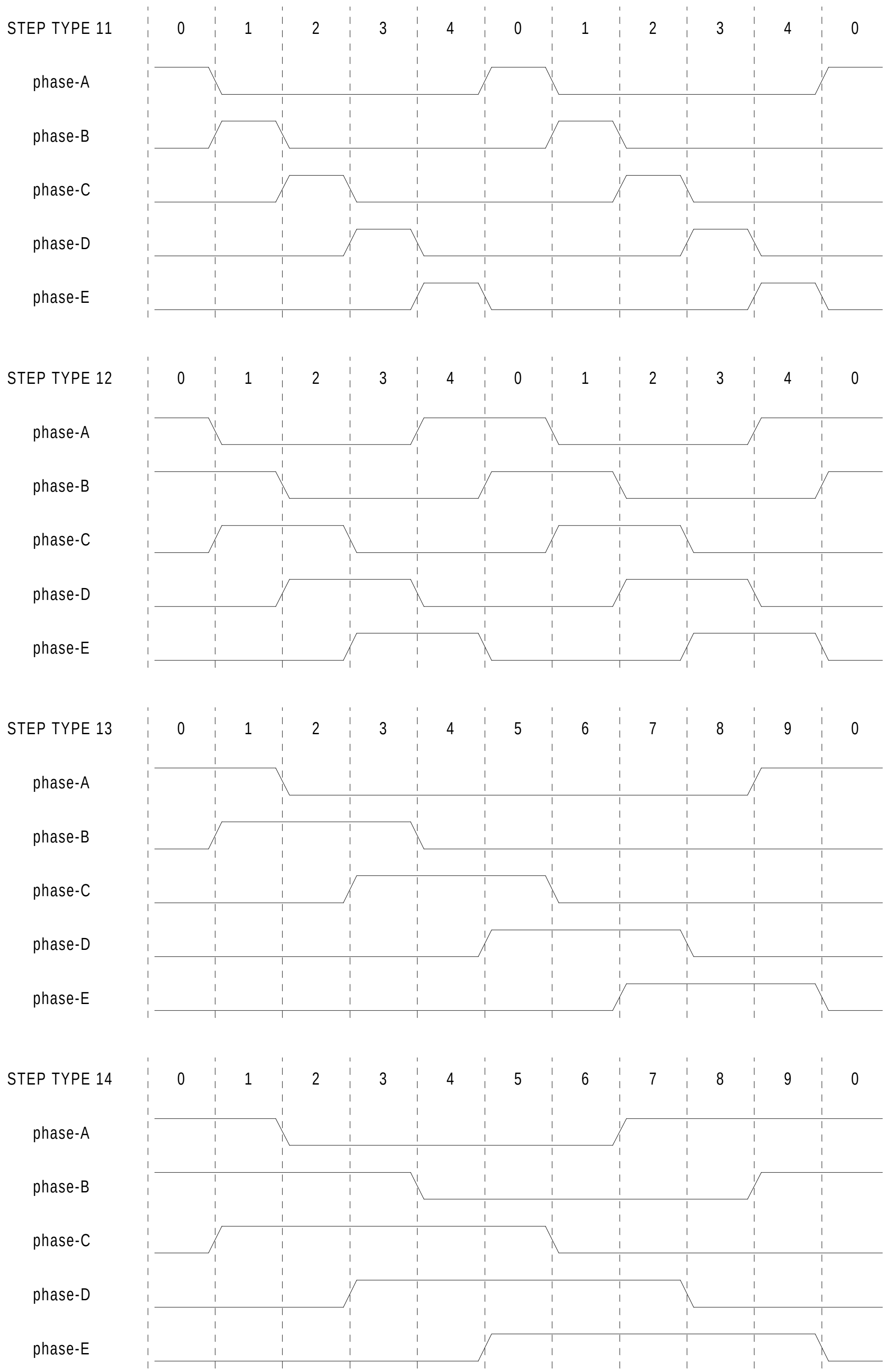


Figure 14.6: Five-Phase Step Types

14.2 PWMgen

This component provides software based generation of PWM (Pulse Width Modulation) and PDM (Pulse Density Modulation) waveforms. It is a realtime component only, and depending on CPU speed, etc, is capable of PWM frequencies from a few hundred Hertz at pretty good resolution, to perhaps 10KHz with limited resolution.

14.2.1 Installing

```
emc2$ halcmd loadrt pwmgen output_type=<config-array>
```

`<config-array>` is a series of comma separated decimal integers. Each number causes a single PWM generator to be loaded, the value of the number determines the output type. For example:

```
emc2$ halcmd loadrt pwmgen step_type=0,1,2
```

will install three PWM generators. The first one will use output type '0' (PWM only), the next uses output type 1 (PWM and direction) and the last one uses output type 2 (UP and DOWN). There is no default value, if `<config-array>` is not specified, no PWM generators will be installed. The maximum number of frequency generators is 8 (as defined by MAX_CHAN in pwmgen.c). Each generator is independent, but all are updated by the same function(s) at the same time. In the following descriptions, `<chan>` is the number of a specific generator. The first generator is number 0.

14.2.2 Removing

```
emc2$ halcmd unloadrt pwmgen
```

14.2.3 Pins

Each PWM generator will have the following pins:

- `(FLOAT) pwmgen.<chan>.value` – Command value, in arbitrary units. Will be scaled by the `scale` parameter (see below).
- `(BIT) pwmgen.<chan>.enable` – Enables or disables the PWM generator outputs.

Each PWM generator will also have some of these pins, depending on the output type selected:

- `(BIT) pwmgen.<chan>.pwm` – PWM (or PDM) output, (output types 0 and 1 only).
- `(BIT) pwmgen.<chan>.dir` – Direction output (output type 1 only).
- `(BIT) pwmgen.<chan>.up` – PWM/PDM output for positive input value (output type 2 only).
- `(BIT) pwmgen.<chan>.down` – PWM/PDM output for negative input value (output type 2 only).

14.2.4 Parameters

- (FLOAT) pwmgen.<chan>.scale – Scaling factor to convert value from arbitrary units to duty cycle.
- (FLOAT) pwmgen.<chan>.pwm-freq – Desired PWM frequency, in Hz. If 0.0, generates PDM instead of PWM. If set higher than internal limits, next call of update_freq() will set it to the internal limit. If non-zero, and dither is false, next call of update_freq() will set it to the nearest integer multiple of the make_pulses() function period.
- (BIT) pwmgen.<chan>.dither-pwm – If true, enables dithering to achieve average PWM frequencies or duty cycles that are unobtainable with pure PWM. If false, both the PWM frequency and the duty cycle will be rounded to values that can be achieved exactly.
- (FLOAT) pwmgen.<chan>.min-dc – Minimum duty cycle, between 0.0 and 1.0 (duty cycle will go to zero when disabled, regardless of this setting).
- (FLOAT) pwmgen.<chan>.max-dc – Maximum duty cycle, between 0.0 and 1.0.
- (FLOAT) pwmgen.<chan>.curr-dc – Current duty cycle - after all limiting and rounding (read only).

14.2.5 Output Types

The PWM generator supports three different "output types". Type 0 has a single output pin. Only positive commands are accepted, negative values are treated as zero (and will be affected by min-dc if it is non-zero). Type 1 has two output pins, one for the PWM/PDM signal and one to indicate direction. The duty cycle on the PWM pin is based on the absolute value of the command, so negative values are acceptable. The direction pin is false for positive commands, and true for negative commands. Finally, type 2 also has two outputs, called up and down. For positive commands, the PWM signal appears on the up output, and the down output remains false. For negative commands, the PWM signal appears on the down output, and the up output remains false. Output type 2 is suitable for driving most H-bridges.

14.2.6 Functions

The component exports two functions. Each function acts on all of the PWM generators - running different generators in different threads is not supported.

- (FUNCT) pwmgen.make-pulses – High speed function to generate PWM waveforms (no floating point).
- (FUNCT) pwmgen.update – Low speed function to scale and limit value and handle other paremeters.

The high speed function pwmgen.make-pulses should be run in a very fast thread, from 10 to 50uS depending on the capabilities of the computer. That thread's period determines the maximum PWM carrier frequency, as well as the resolution of the PWM or PDM signals. The other function can be called at a much lower rate.

14.3 Encoder

This component provides software based counting of signals from quadrature encoders. It is a realtime component only, and depending on CPU speed, etc, is capable of maximum count rates of 10kHz to perhaps 50kHz. Figure 14.7 is a block diagram of one channel of encoder counter.

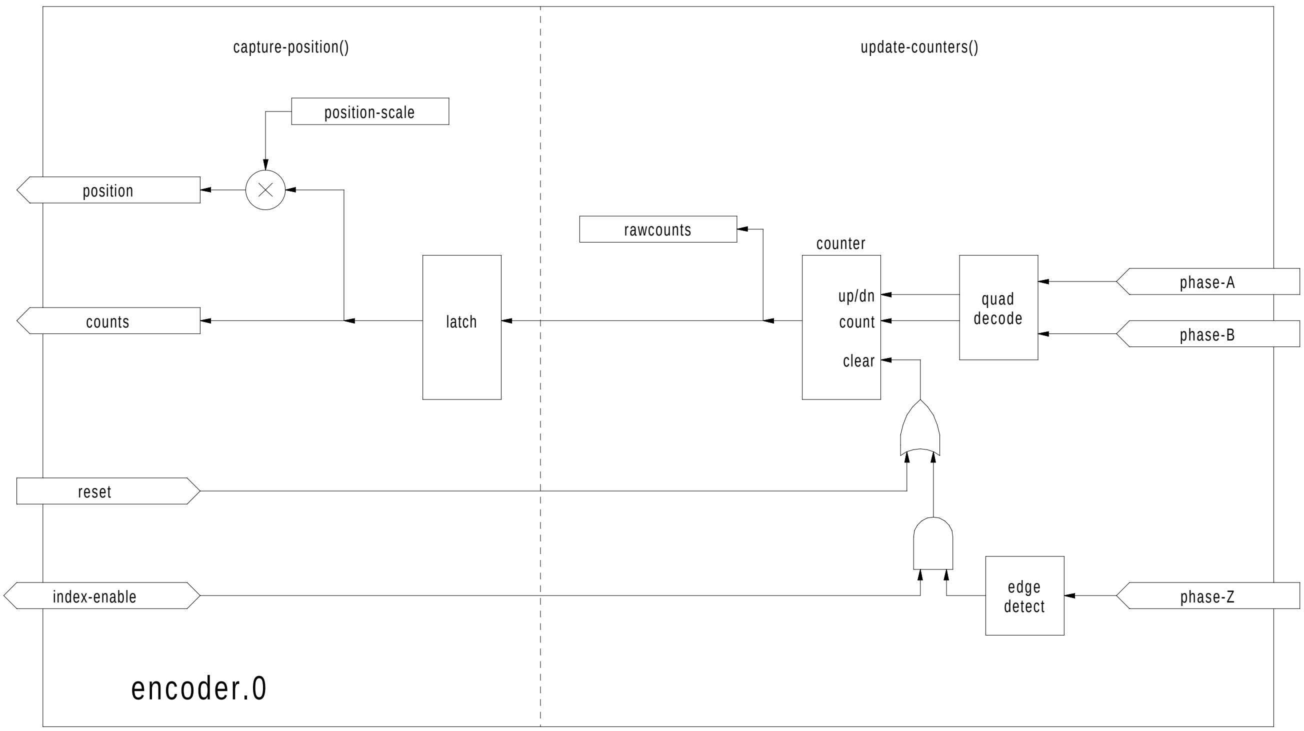


Figure 14.7: Encoder Counter Block Diagram

14.3.1 Installing

```
emc2$ halcmd loadrt encoder [num_chan=<counters>]
```

`<counters>` is the number of encoder counters that you want to install. If `numchan` is not specified, three counters will be installed. The maximum number of counters is 8 (as defined by MAX_CHAN in encoder.c). Each counter is independent, but all are updated by the same function(s) at the same time. In the following descriptions, `<chan>` is the number of a specific counter. The first counter is number 0.

14.3.2 Removing

```
emc2$ halcmd unloadrt encoder
```

14.3.3 Pins

- `(BIT) encoder.<chan>.phase-A` – Phase A of the quadrature encoder signal.
- `(BIT) encoder.<chan>.phase-B` – Phase B of the quadrature encoder signal.
- `(BIT) encoder.<chan>.phase-Z` – Phase Z (index pulse) of the quadrature encoder signal.
- `(BIT) encoder.<chan>.reset` – See canonical encoder interface, section 9.5.
- `(BIT) encoder.<chan>.velocity` – Estimated speed of the quadrature signal.
- `(BIT) encoder.<chan>.index-enable` – See canonical encoder interface.
- `(S32) encoder.<chan>.count` – See canonical encoder interface.
- `(FLOAT) encoder.<chan>.position` – See canonical encoder interface.

14.3.4 Parameters

- `(S32) encoder.<chan>.raw-count` – The raw count value, updated by `update-counters()`.
- `(BIT) encoder.<chan>.x4-mode` – Sets encoder to 4x or 1x mode. The 1x mode is usefull for some jogwheels.
- `(FLOAT) encoder.<chan>.position-scale` – See canonical encoder interface, section 9.5.

14.3.5 Functions

The component exports two functions. Each function acts on all of the encoder counters - running different counters in different threads is not supported.

- `(FUNCT) encoder.update-counters` – High speed function to count pulses (no floating point).
- `(FUNCT) encoder.capture-position` – Low speed function to update latches and scale position.

14.4 PID

This component provides Proportional/Integeral/Derivative control loops. It is a realtime component only. For simplicity, this discussion assumes that we are talking about position loops, however this component can be used to implement other feedback loops such as speed, torch height, temperature, etc. Figure 14.8 is a block diagram of a single PID loop.

14.4.1 Installing

```
emc2$ halcmd loadrt pid [num_chan=<loops>] [debug=1]
```

`<loops>` is the number of PID loops that you want to install. If `numchan` is not specified, one loop will be installed. The maximum number of loops is 16 (as defined by MAX_CHAN in pid.c). Each loop is completely independent. In the following descriptions, `<loopnum>` is the loop number of a specific loop. The first loop is number 0.

If `debug=1` is specified, the component will export a few extra parameters that may be useful during debugging and tuning. By default, the extra parameters are not exported, to save shared memory space and avoid cluttering the parameter list.

14.4.2 Removing

```
emc2$ halcmd unloadrt pid
```

14.4.3 Pins

The three most important pins are

- `(FLOAT) pid.<loopnum>.command` – The desired position, as commanded by another system component.
- `(FLOAT) pid.<loopnum>.feedback` – The present position, as measured by a feedback device such as an encoder.
- `(FLOAT) pid.<loopnum>.output` – A velocity command that attempts to move from the present position to the desired position.

For a position loop, 'command' and 'feedback' are in position units. For a linear axis, this could be inches, mm, meters, or whatever is relevant. Likewise, for an angular axis, it could be degrees, radians, etc. The units of the 'output' pin represent the change needed to make the feedback match the command. As such, for a position loop 'Output' is a velocity, in inches/sec, mm/sec, degrees/sec, etc. Time units are always seconds, and the velocity units match the position units. If command and feedback are in meters, then output is in meters per second.

Each loop has two other pins which are used to monitor or control the general operation of the component.

- `(FLOAT) pid.<loopnum>.error` – Equals `.command` minus `.feedback`.
- `(BIT) pid.<loopnum>.enable` – A bit that enables the loop. If `.enable` is false, all integrators are reset, and the output is forced to zero. If `.enable` is true, the loop operates normally.

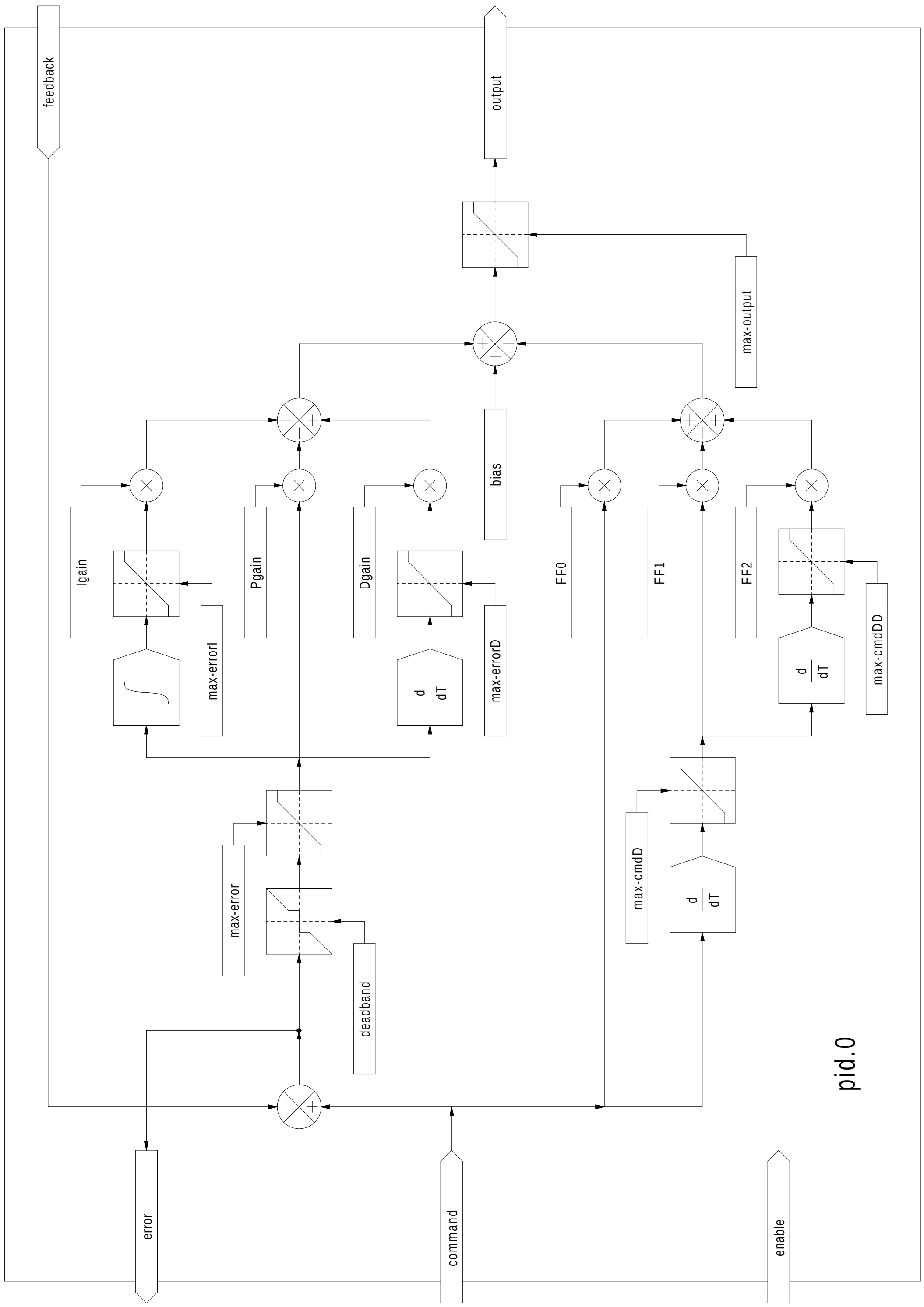


Figure 14.8: PID Loop Block Diagram

14.4.4 Parameters

The PID gains, limits, and other 'tunable' features of the loop are implemented as parameters.

- (FLOAT) pid.<loopnum>.Pgain – Proportional gain
- (FLOAT) pid.<loopnum>.Igain – Integral gain
- (FLOAT) pid.<loopnum>.Dgain – Derivative gain
- (FLOAT) pid.<loopnum>.bias – Constant offset on output
- (FLOAT) pid.<loopnum>.FF0 – Zeroth order feedforward - output proportional to command (position).
- (FLOAT) pid.<loopnum>.FF1 – First order feedforward - output proportional to derivative of command (velocity).
- (FLOAT) pid.<loopnum>.FF2 – Second order feedforward - output proportional to 2nd derivative of command (acceleration)[1].
- (FLOAT) pid.<loopnum>.deadband – Amount of error that will be ignored
- (FLOAT) pid.<loopnum>.maxerror – Limit on error
- (FLOAT) pid.<loopnum>.maxerrorI – Limit on error integrator
- (FLOAT) pid.<loopnum>.maxerrorD – Limit on error derivative
- (FLOAT) pid.<loopnum>.maxcmdD – Limit on command derivative
- (FLOAT) pid.<loopnum>.maxcmdDD – Limit on command 2nd derivative
- (FLOAT) pid.<loopnum>.maxoutput – Limit on output value

All of the max??? limits are implemented such that if the parameter value is zero, there is no limit.

If debug=1 was specified when the component was installed, four additional parameters will be exported:

- (FLOAT) pid.<loopnum>.errorI – Integral of error.
- (FLOAT) pid.<loopnum>.errorD – Derivative of error.
- (FLOAT) pid.<loopnum>.commandD – Derivative of the command.
- (FLOAT) pid.<loopnum>.commandDD – 2nd derivative of the command.

14.4.5 Functions

The component exports one function for each PID loop. This function performs all the calculations needed for the loop. Since each loop has its own function, individual loops can be included in different threads and execute at different rates.

- (FUNCT) pid.<loopnum>.do_pid_calcs – Performs all calculations for a single PID loop.

If you want to understand the exact algorithm used to compute the output of the PID loop, refer to figure 14.8, the comments at the beginning of emc2/src/hal/components/pid.c, and of course to the code itself. The loop calculations are in the C function calc_pid().

[1] FF2 is not currently implemented, but it will be added. Consider this note a "FIXME" for the code

14.5 Simulated Encoder

The simulated encoder is exactly that. It produces quadrature pulses with an index pulse, at a speed controlled by a HAL pin. Mostly useful for testing.

14.5.1 Installing

```
emc2$ halcmd loadrt sim-encoder num_chan=<number>
```

`<number>` is the number of encoders that you want to simulate. If not specified, one encoder will be installed. The maximum number is 8 (as defined by MAX_CHAN in sim_encoder.c).

14.5.2 Removing

```
emc2$ halcmd unloadrt sim-encoder
```

14.5.3 Pins

- `(FLOAT) sim-encoder.<chan-num>.speed` – The speed command for the simulated shaft.
- `(BIT) sim-encoder.<chan-num>.phase-A` – Quadrature output.
- `(BIT) sim-encoder.<chan-num>.phase-B` – Quadrature output.
- `(BIT) sim-encoder.<chan-num>.phase-Z` – Index pulse output.

When `.speed` is positive, `.phase-A` leads `.phase-B`.

14.5.4 Parameters

- (U32) `sim-encoder.<chan-num>.ppr` – Pulses Per Revolution.
- (FLOAT) `sim-encoder.<chan-num>.scale` – Scale Factor for **speed**. The default is 1.0, which means that **speed** is in revolutions per second. Change to 60 for RPM, to 360 for degrees per second, 6.283185 for radians per seconed, etc.

Note that pulses per revolution is not the same as counts per revolution. A pulse is a complete quadrature cycle. Most encoder counters will count four times during one complete cycle.

14.5.5 Functions

The component exports two functions. Each function affects all simulated encoders.

- `(FUNCT) sim-encoder.make-pulses` – High speed function to generate quadrature pulses (no floating point).
- `(FUNCT) sim-encoder.update-speed` – Low speed function to read **speed**, do scaling, and set up **make-pulses**.

14.6 Debounce

Debounce is a realtime component that can filter the glitches created by mechanical switch contacts. It may also be useful in other applications where short pulses are to be rejected.

14.6.1 Installing

emc2$ **halcmd loadrt debounce cfg="<config-string>"**

`<config-string>` is a series of space separated decimal integers. Each number installs a group of identical debounce filters, the number determines how many filters are in the group. For example:

emc2$ **halcmd loadrt debounce cfg="1 4 2"**

will install three groups of filters. Group 0 contains one filter, group 1 contains four, and group 2 contains two filters. The default value for `<config-string>` is "`1`" which will install a single group containing a single filter. The maximum number of groups 8 (as defined by MAX_GROUPS in debounce.c). The maximum number of filters in a group is limited only by shared memory space. Each group is completely independent. All filters in a single group are identical, and they are all updated by the same function at the same time. In the following descriptions, `<G>` is the group number and `<F>` is the filter number within the group. The first filter is group 0, filter 0.

14.6.2 Removing

emc2$ **halcmd unloadrt debounce**

14.6.3 Pins

Each individual filter has two pins.

- `(BIT) debounce.<G>.<F>.in` – Input of filter `<F>` in group `<G>`.
- `(BIT) debounce.<G>.<F>.out` – Output of filter `<F>` in group `<G>`.

14.6.4 Parameters

Each group of filters has one parameter[2].

- `(S32) debounce.<G>.delay` – Filter delay for all filters in group `<G>`.

The filter delay is in units of thread periods. The minimum delay is zero. The output of a zero delay filter exactly follows its input - it doesn't filter anything. As `delay` increases, longer and longer glitches are rejected. If `delay` is 4, all glitches less than or equal to four thread periods will be rejected.

14.6.5 Functions

Each group of filters has one function, which updates all the filters in that group "simultaneously". Different groups of filters can be updated from different threads at different periods.

- `(FUNCT) debounce.<G>` – Updates all filters in group `<G>`.

[2]Each individual filter also has an internal state variable. There is a compile time switch that can export that variable as a parameter. This is intended for testing, and simply wastes shared memory under normal circumstances.

14.7 Siggen

Siggen is a realtime component that generates square, triangle, and sine waves. It is primarily used for testing.

14.7.1 Installing

```
emc2$ halcmd loadrt siggen [num_chan=<chans>]
```

`<chans>` is the number of signal generators that you want to install. If `numchan` is not specified, one signal generator will be installed. The maximum number of generators is 16 (as defined by MAX_CHAN in siggen.c). Each generator is completely independent. In the following descriptions, `<chan>` is the number of a specific signal generator (the numbers start at 0).

14.7.2 Removing

```
emc2$ halcmd unloadrt siggen
```

14.7.3 Pins

Each generator has five output pins.

- (FLOAT) `siggen.<chan>.sine` – Sine wave output.
- (FLOAT) `siggen.<chan>.cosine` – Cosine output.
- (FLOAT) `siggen.<chan>.sawtooth` – Sawtooth output.
- (FLOAT) `siggen.<chan>.triangle` – Triangle wave output.
- (FLOAT) `siggen.<chan>.square` – Square wave output.

All five outputs have the same frequency, amplitude, and offset.

In addition to the output pins, there are three control pins:

- (FLOAT) `siggen.<chan>.frequency` – Sets the frequency in Hertz, default value is 1 Hz.
- (FLOAT) `siggen.<chan>.amplitude` – Sets the peak amplitude of the output waveforms, default is 1.
- (FLOAT) `siggen.<chan>.offset` – Sets DC offset of the output waveforms, default is 0.

For example, if `siggen.0.amplitude` is 1.0 and `siggen.0.offset` is 0.0, the outputs will swing from -1.0 to +1.0. If `siggen.0.amplitude` is 2.5 and `siggen.0.offset` is 10.0, then the outputs will swing from 7.5 to 12.5.

14.7.4 Parameters

None. [3]

14.7.5 Functions

- (FUNCT) `siggen.<chan>.update` – Calculates new values for all five outputs.

[3] Prior to version 2.1, frequency, amplitude, and offset were parameters. They were changed to pins to allow control by other components.

Chapter 15

Hardware Drivers

15.1 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 pins, and the status group consists of 5 input pins.

In the early 1990's, the bidirectional parallel port was introduced, which allows the data group to be used for output or input. The HAL driver supports the bidirectional port, and allows the user to set the data group as either input or output. If configured as output, a port provides a total of 12 outputs and 5 inputs. If configured as input, it provides 4 outputs and 13 inputs.

In some parallel ports, the control group pins are open collectors, which may also be driven low by an external gate. On a board with open collector control pins, the "x" mode allows a more flexible mode with 8 dedicated outputs, 5 dedicated inputs, and 4 open collector pins. In other parallel ports, the control group has push-pull drivers and cannot be used as an input.[1]

No other combinations are supported, and a port cannot be changed from input to output once the driver is installed. Figure 15.1 shows two block diagrams, one showing the driver when the data group is configured for output, and one showing it configured for input.

The parport driver can control up to 8 ports (defined by MAX_PORTS in hal_parport.c). The ports are numbered starting at zero.

15.1.1 Installing

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

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

[1]HAL cannot automatically determine if the "x" mode bidirectional pins are actually open collectors (OC). If they are not, they cannot be used as inputs, and attempting to drive them LOW from an external source can damage the hardware.

To determine whether your port has "open collector" pins, load hal_parport in "x" mode, output a HIGH value on the pin. HAL should read the pin as TRUE. Next, insert a 470Ω resistor from one of the control pins to GND. If the resulting voltage on the control pin is close to 0V, and HAL now reads the pin as FALSE, then you have an OC port. If the resulting voltage is far from 0V, or HAL does not read the pin as FALSE, then your port cannot be used in "x" mode.

The external hardware that drives the control pins should also use open collector gates (e.g., 74LS05). Generally, the -out HAL pins should be set to TRUE when the physical pin is being used as an input.

On some machines, BIOS settings may affect whether "x" mode can be used. "SPP" mode is most most likely to work.

```
loadrt hal_parport cfg="0x278 0x378 in 0x20A0 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 which is typically configured in the system BIOS. The address for a PCI card is usally shown in “lspci -v” in an “I/O ports” line, or in the kernel message log after executing “`sudo modprobe -a parport_pc`”. There is no default address; if `<config-string>` does not contain at least one address, it is an error.

15.1.2 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 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.) Pins 1, 14, 16, and 17 are outputs in all modes. 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. In “x” mode, pins 1, 14, 16, and 17 are also input pins.

15.1.3 Parameters

- `(BIT) parport.<portnum>.pin-<pinnum>-out-invert` – Inverts an output pin.
- `(BIT) parport.<portnum>.pin-<pinnum>-out-reset` (only for pins 2..9) – TRUE if this pin should be reset when the `-reset` function is executed.
- `(U32) parport.<portnum>.reset-time` – The time (in nanoseconds) between a pin is set by `write` and reset by `reset` HAL functions.

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.

If `-reset` is TRUE, then the `reset` function will set the pin to the value of `-out-invert`. This can be used in conjunction with stepgen’s `doublefreq` to produce one step per period.

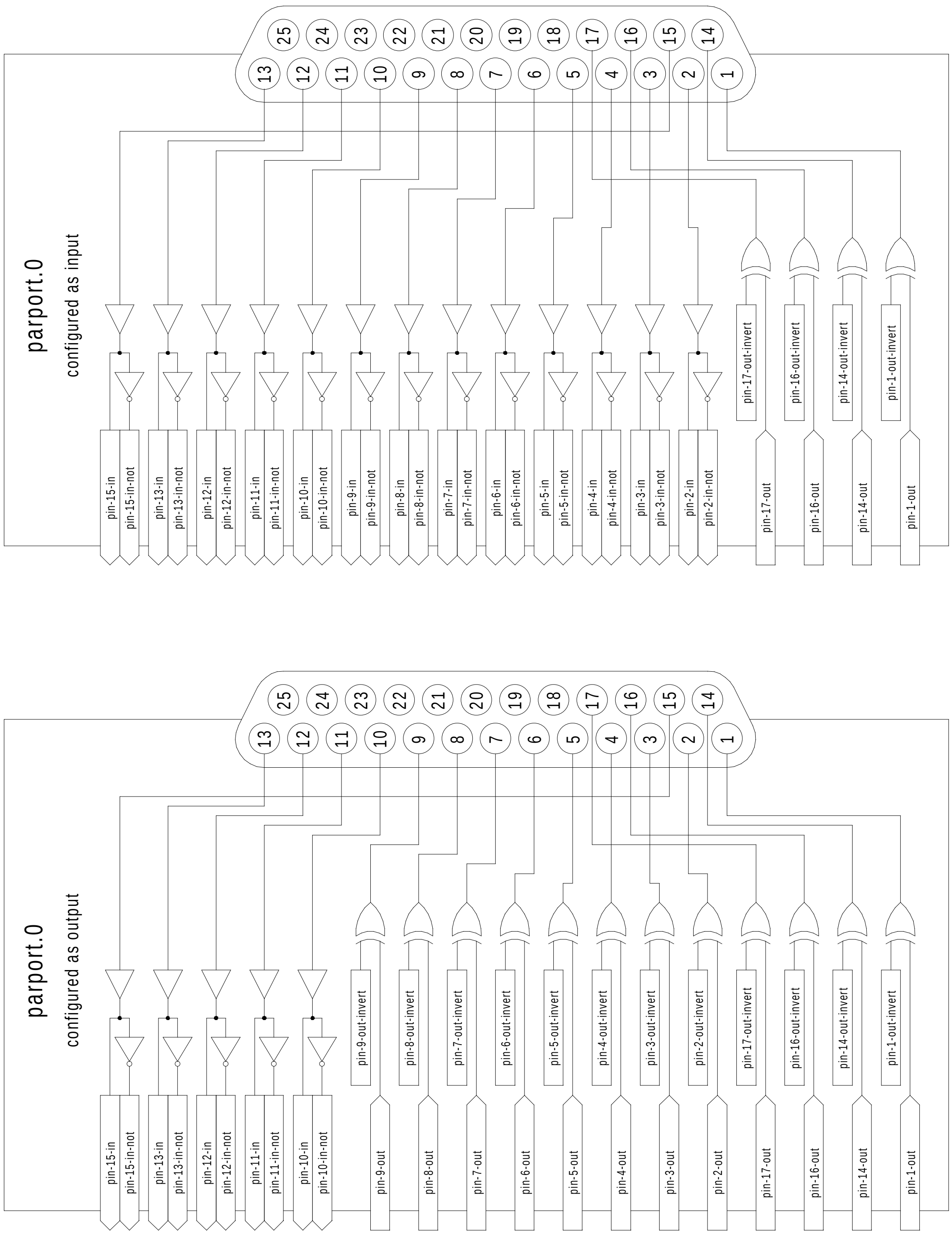


Figure 15.1: Parport Block Diagram

15.1.4 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.
- (FUNCT) parport.<portnum>.reset – Waits until reset-time has elapsed since the associated write, then resets pins to values indicated by -out-invert and -out-invert settings. reset must be later in the same thread as write

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.

15.1.5 Common problems

If loading the module reports

```
insmod: error inserting '/home/jepler/emc2/rtlib/hal_parport.ko':
-1 Device or resource busy
```

then ensure that the standard kernel module parport_pc is not loaded[2] and that no other device in the system has claimed the I/O ports.

If the module loads but does not appear to function, then the port address is incorrect or the probe_parport module is required.

15.2 probe_parport

In modern PCs, the parallel port may require plug and play (PNP) configuration before it can be used. The probe_parport module performs configuration of any PNP ports present, and should be loaded before hal_parport. On machines without PNP ports, it may be loaded but has no effect.

15.2.1 Installing

```
loadrt probe_parport
loadrt hal_parport ...
```

If the Linux kernel prints a message similar to

```
parport: PnPBIOS parport detected.
```

when the parport_pc module is loaded (sudo modprobe -a parport_pc; sudo rmmod parport_pc) then use of this module is probably required.

[2] In the emc packages for Ubuntu, the file /etc/modprobe.d/emc2 generally prevents parport_pc from being automatically loaded.

15.3 AX5214H

The Axiom Measurement & Control AX5214H is a 48 channel digital I/O board. It plugs into an ISA bus, and resembles a pair of 8255 chips.[3]

15.3.1 Installing

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

The config string consists of a hex port address, followed by an 8 character string of "I" and "O" which sets groups of pins as inputs and outputs. The first two character set the direction of the first two 8 bit blocks of pins (0-7 and 8-15). The next two set blocks of 4 pins (16-19 and 20-23). The pattern then repeats, two more blocks of 8 bits (24-31 and 32-39) and two blocks of 4 bits (40-43 and 44-47). If more than one board is installed, the data for the second board follows the first. As an example, the string **`"0x220 IIIOIIOO 0x300 OIOOIOIO"`** installs drivers for two boards. The first board is at address 0x220, and has 36 inputs (0-19 and 24-39) and 12 outputs (20-23 and 40-47). The second board is at address 0x300, and has 20 inputs (8-15, 24-31, and 40-43) and 28 outputs (0-7. 16-23, 32-39, and 44-47). Up to 8 boards may be used in one system.

15.3.2 Pins

- `(BIT) ax5214.<boardnum>.out-<pinnum>` – Drives a physical output pin.
- `(BIT) ax5214.<boardnum>.in-<pinnum>` – Tracks a physical input pin.
- `(BIT) ax5214.<boardnum>.in-<pinnum>-not` – Tracks a physical input pin, inverted.

For each pin, `<boardnum>` is the board number (starts at zero), and `<pinnum>` is the I/O channel number (0 to 47).

Note that the driver assumes active LOW signals. This is so that modules such as OPTO-22 will work correctly (TRUE means output ON, or input energized). If the signals are being used directly without buffering or isolation the inversion needs to be accounted for. The `in-` HAL pin is TRUE if the physical pin is low (OPTO-22 module energized), and FALSE if the physical pin is high (OPTO-22 module off). The `in-<pinnum>-not` HAL pin is inverted – it is FALSE if the physical pin is low (OPTO-22 module energized). By connecting a signal to one or the other, the user can determine the state of the input.

15.3.3 Parameters

- `(BIT) ax5214.<boardnum>.out-<pinnum>-invert` – Inverts an output pin.

The `-invert` parameter determines whether an output pin is active high or active low. If `-invert` is FALSE, setting the HAL `out-` pin TRUE drives the physical pin low, turning ON an attached OPTO-22 module, and FALSE drives it high, turning OFF the OPTO-22 module. If `-invert` is TRUE, then setting the HAL `out-` pin TRUE will drive the physical pin high and turn the module OFF.

15.3.4 Functions

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

[3]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.

15.4 Servo-To-Go

The Servo-To-Go is one of the first PC motion control cards[4] 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.

15.4.1 Installing:

```
loadrt hal_stg [base=<address>] [num_chan=<nr>] [dio="<dio-string>"] [model=<model>]
```

The base address field is optional; if it's not provided the driver attempts to autodetect the board. The num_chan field is used to specify the number of channels available on the card, if not used the 8 axis version is assumed. The digital inputs/outputs configuration is determined by a config string passed to insmod when loading the module. The format consists of a four character string that sets the direction of each group of pins. Each character of the direction string is either "I" or "O". The first character sets the direction of port A (Port A - DIO.0-7), the next sets port B (Port B - DIO.8-15), the next sets port C (Port C - DIO.16-23), and the fourth sets port D (Port D - DIO.24-31). The model field can be used in case the driver doesn't autodetect the right card version[5]. For example:

```
loadrt hal_stg base=0x300 num_chan=4 dio="IOIO"
```

This example installs the stg driver for a card found at the base address of 0x300, 4 channels of encoder feedback, 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

```
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.

15.4.2 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.
- `(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

[4] 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.)

[5] hint: after starting up the driver, 'dmesg' can be consulted for messages relevant to the driver (e.g. autodetected version number and base address)

For each pin, <channel> is the axis number, and <pinnum> is the logic pin number of the STG[6].

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.

15.4.3 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.

15.4.4 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.

15.5 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.[7]

Installing:

```
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. Valid values are from 1 to 32226. The driver prints some useful debugging message to the kernel log, which can be viewed with **dmesg**.

Up to 4 boards may be used in one system.

[6] if IIOO is defined, there are 16 input pins (in-00 .. in-15) and 16 output pins (out-00 .. out-15), and they correspond to PORTs ABCD (in-00 is PORTA.0, out-15 is PORTD.7)

[7] 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.

15.5.1 Pins

In the following pins, parameters, and functions, <board> is the board ID. According to the naming conventions the first board should always have an ID of zero, however this driver 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.
- (FLOAT) `m5i20.<board>.enc-<channel>-position` – Encoder position, in user units.
- (BIT) `m5i20.<board>.enc-<channel>-index` – Current status of index pulse input?
- (BIT) `m5i20.<board>.enc-<channel>-index-enable` – when TRUE, and an index pulse appears on the encoder input, reset counter to zero and clear `index-enable`.
- (BIT) `m5i20.<board>.enc-<channel>-reset` – When true, counter is forced to zero.
- (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. If bit value 16 is set in `watchdog-control` then this value is not used, and the hardware watchdog is cleared every time the `dac-write` function is executed.

15.5.2 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 anaolg voltage.[8]
- (BIT) `m5i20.<board>.out-<channel>-invert` – Inverts a digital output, see canonical digital output.

[8]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).

- (U32) m5i20.<board>.watchdog-control – Configures the watchdog. The value may be a bitwise OR of the following values:

Bit #	Value	Meaning
0	1	Watchdog is enabled
1	2	Watchdog is automatically reset by DAC writes (the HAL dac-write function)

  Typically, the useful values are 0 (watchdog disabled) or 3 (watchdog enabled, cleared by dac-write).

- (U32) m5i20.<board>.led-view – Maps some of the I/O to onboard LEDs. See table below.

15.5.3 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.

15.5.4 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.

15.5.4.1 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

15.5.4.2 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	

15.5.4.3 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[9] for a description of the CCR.

[9] 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	

15.5.4.4 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 viewed channel.

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 ?

15.6 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.[10]

Installing:

[10] 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.

```
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**.

Up to 4 boards may be used in one system.

15.6.1 Pins

In the following pins, parameters, and functions, <board> is the board ID. According to the naming conventions the first board should always have an ID of zero. However this driver sets the ID based on a pair of jumpers on the 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.

15.6.2 Parameters

- (FLOAT) motenc.<board>.enc-<channel>-scale – The number of counts / user unit (to convert from counts to units).
- (FLOAT) motenc.<board>.dac-<channel>-offset – Sets the DAC offset.
- (FLOAT) motenc.<board>.dac-<channel>-gain – Sets the DAC gain (scaling).
- (FLOAT) motenc.<board>.adc-<channel>-offset – Sets the ADC offset.

- (FLOAT) motenc.<board>.adc-<channel>-gain – Sets the ADC gain (scaling).
- (BIT) motenc.<board>.out-<channel>-invert – Inverts a digital output, see canonical digital output.
- (U32) motenc.<board>.watchdog-control – Configures the watchdog. The value may be a bitwise OR of the following values:

Bit #	Value	Meaning
0	1	Timeout is 16ms if set, 8ms if unset
2	4	Watchdog is enabled
4	16	Watchdog is automatically reset by DAC writes (the HAL dac-write function)

  Typically, the useful values are 0 (watchdog disabled) or 20 (8ms watchdog enabled, cleared by dac-write).

- (U32) motenc.<board>.led-view – Maps some of the I/O to onboard LEDs?

15.6.3 Functions

- (FUNCT) motenc.<board>.encoder-read – Reads all encoder counters.
- (FUNCT) motenc.<board>.adc-read – Reads the analog-to-digital converters.
- (FUNCT) motenc.<board>.digital-in-read – Reads digital inputs.
- (FUNCT) motenc.<board>.dac-write – Writes the voltages to the DACs.
- (FUNCT) motenc.<board>.digital-out-write – Writes digital outputs.
- (FUNCT) motenc.<board>.misc-update – Updates misc stuff.

15.7 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:

```
loadrt hal_ppmc port_addr=<addr1>[,<addr2>[,<addr3>...]]
```

The **port_addr** parameter tells the driver what parallel port(s) to check. By default, **<addr1>** is 0x0378, and **<addr2>** and following are not used. The driver searches the entire address space of the enhanced parallel port(s) at **port_addr**, looking for any board(s) in the PPMC family. It then exports HAL pins for whatever it finds. During loading (or attempted loading) the driver prints some usefull debugging message to the kernel log, which can be viewed with **dmesg**.

Up to 3 parport busses may be used, and each bus may have up to 8 devices on it.

15.7.1 Pins

In the following pins, parameters, and functions, <board> is the board ID. According to the naming conventions the first board should always have an ID of zero. However this driver sets the ID based on a pair of jumpers on the baord, 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.[11]
- (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, seen canonical digital output.

15.7.2 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 capacators used on some MOSFET gate drivers.

[11] Index handling does _not_ comply with the canonical encoder interface, and should be changed.

- (U32) ppmc.<port>.stepgen.<channel-range>.setup-time – Sets minimum time between direction change and step pulse, in units of 100nS. Applies to a group fof four consecutive PWM generators, as indicated by <channel-range>.
- (U32) ppmc.<port>.stepgen.<channel-range>.pulse-width – Sets width of step pulses, in units of 100nS. Applies to a group fof four consecutive PWM generators, as indicated by <channel-range>.
- (U32) ppmc.<port>.stepgen.<channel-range>.pulse-space-min – Sets minimum time between pulses, in units of 100nS. The maximum step rate is 1/(100nS * (pulse-width + pulse-space-min)). Applies to a group fof 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.

15.7.3 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.

15.8 Pluto-P: generalities

The Pluto-P is an inexpensive ($60) FPGA board featuring the ACEX1K chip from Altera.

15.8.1 Requirements

1. A Pluto-P board
2. An EPP-compatible parallel port, configured for EPP mode in the system BIOS

15.8.2 Connectors

- The Pluto-P board is shipped with the left connector presoldered, with the key in the indicated position. The other connectors are unpopulated. There does not seem to be a standard 12-pin IDC connector, but some of the pins of a 16P connector can hang off the board next to QA3/QZ3.
- The bottom and right connectors are on the same .1" grid, but the left connector is not. If OUT2. . . OUT9 are not required, a single IDC connector can span the bottom connector and the bottom two rows of the right connector.

15.8.3 Physical Pins

- Read the ACEX1K datasheet for information about input and output voltage thresholds. The pins are all configured in "LVTTL/LVCMOS" mode and are generally compatible with 5V TTL logic.
- Before configuration and after properly exiting emc2, all Pluto-P pins are tristated with weak pull-ups (20kΩ min, 50kΩ max). If the watchdog timer is enabled (the default), these pins are also tristated after an interruption of communication between emc2 and the board. The watchdog timer takes approximately 6.5ms to activate. However, software bugs in the pluto_servo firmware or emc2 can leave the Pluto-P pins in an undefined state.
- In pwm+dir mode, by default dir is HIGH for negative values and LOW for positive values. To select HIGH for positive values and LOW for negative values, set the corresponding dout-NN-invert parameter TRUE to invert the signal.
- The index input is triggered on the rising edge. Initial testing has shown that the QZx inputs are particularly noise sensitive, due to being polled every 25ns. Digital filtering has been added to filter pulses shorter than 175ns (seven polling times). Additional external filtering on all input pins, such as a Schmitt buffer or inverter, RC filter, or differential receiver (if applicable) is recommended.
- The IN1...IN7 pins have 22-ohm series resistors to their associated FPGA pins. No other pins have any sort of protection for out-of-spec voltages or currents. It is up to the integrator to add appropriate isolation and protection. Traditional parallel port optoisolator boards do not work with pluto_servo due to the bidirectional nature of the EPP protocol.

15.8.4 LED

- When the device is unprogrammed, the LED glows faintly. When the device is programmed, the LED glows according to the duty cycle of PWM0 (**LED** = **UP0** *xor* **DOWN0**) or STEPGEN0 (**LED** = **STEP0** *xor* **DIR0**).

15.8.5 Power

- A small amount of current may be drawn from VCC. The available current depends on the unregulated DC input to the board. Alternately, regulated +3.3VDC may be supplied to the FPGA through these VCC pins. The required current is not yet known, but is probably around 50mA plus I/O current.
- The regulator on the Pluto-P board is a low-dropout type. Supplying 5V at the power jack will allow the regulator to work properly.

15.8.6 PC interface

- At present, only a single pluto_servo or pluto_step board is supported. At present there is no provision for multiple boards on one parallel port (because all boards reside at the same EPP address) but supporting one board per parallel port should be possible.

15.8.7 Rebuilding the FPGA firmware

The `src/hal/drivers/pluto_servo_firmware/` and `src/hal/drivers/pluto_step_firmware/` subdirectories contain the Verilog source code plus additional files used by Quartus for the FPGA firmwares. Altera's Quartus II software is required to rebuild the FPGA firmware. To rebuild the firmware from the .hdl and other source files, open the `.qpf` file and press CTRL-L. Then, recompile emc2.

Like the HAL hardware driver, the FPGA firmware is licensed under the terms of the GNU General Public License.

The gratis version of Quartus II runs only on Microsoft Windows, although there is apparently a paid version that runs on Linux.

15.8.8 For more information

The Pluto-P board may be ordered from http://www.knjn.com/ShopBoards_Parallel.html (US based, international shipping is available). Some additional information about it is available from http://www.fpga4fun.com/board_pluto-P.html and from the developer's blog http://emergent.unpy.net/01165081407.

15.9 pluto-servo: Hardware PWM and quadrature counting

The pluto_servo system is suitable for control of a 4-axis CNC mill with servo motors, a 3-axis mill with PWM spindle control, a lathe with spindle encoder, etc. The large number of inputs allows a full set of limit switches.

This driver features:

- 4 quadrature channels with 40MHz sample rate. The counters operate in "4x" mode. The maximum useful quadrature rate is 8191 counts per emc2 servo cycle, or about 8MHz for EMC2's default 1ms servo rate.
- 4 PWM channels, "up/down" or "pwm+dir" style. 4095 duty cycles from -100% to +100%, including 0%. The PWM period is approximately 19.5kHz (40MHz / 2047). A PDM-like mode is also available.
- 18 digital outputs: 10 dedicated, 8 shared with PWM functions. (Example: A lathe with unidirectional PWM spindle control may use 13 total digital outputs)
- 20 digital inputs: 8 dedicated, 12 shared with Quadrature functions. (Example: A lathe with index pulse only on the spindle may use 13 total digital inputs)
- EPP communication with the PC. The EPP communication typically takes around 100uS on machines tested so far, enabling servo rates above 1kHz.

15.9.1 Pinout

UPx The "up" (up/down mode) or "pwm" (pwm+direction mode) signal from PWM generator X. May be used as a digital output if the corresponding PWM channel is unused, or the output on the channel is always negative. The corresponding digital output invert may be set to TRUE to make UPx active low rather than active high.

DNx The "down" (up/down mode) or "direction" (pwm+direction mode) signal from PWM generator X. May be used as a digital output if the corresponding PWM channel is unused, or the output on the channel is never negative. The corresponding digital ouput invert may be set to TRUE to make DNx active low rather than active high.

QAx, QBx The A and B signals for Quadrature counter X. May be used as a digital input if the corresponding quadrature channel is unused.

QZx The Z (index) signal for quadrature counter X. May be used as a digital input if the index feature of the corresponding quadrature channel is unused.

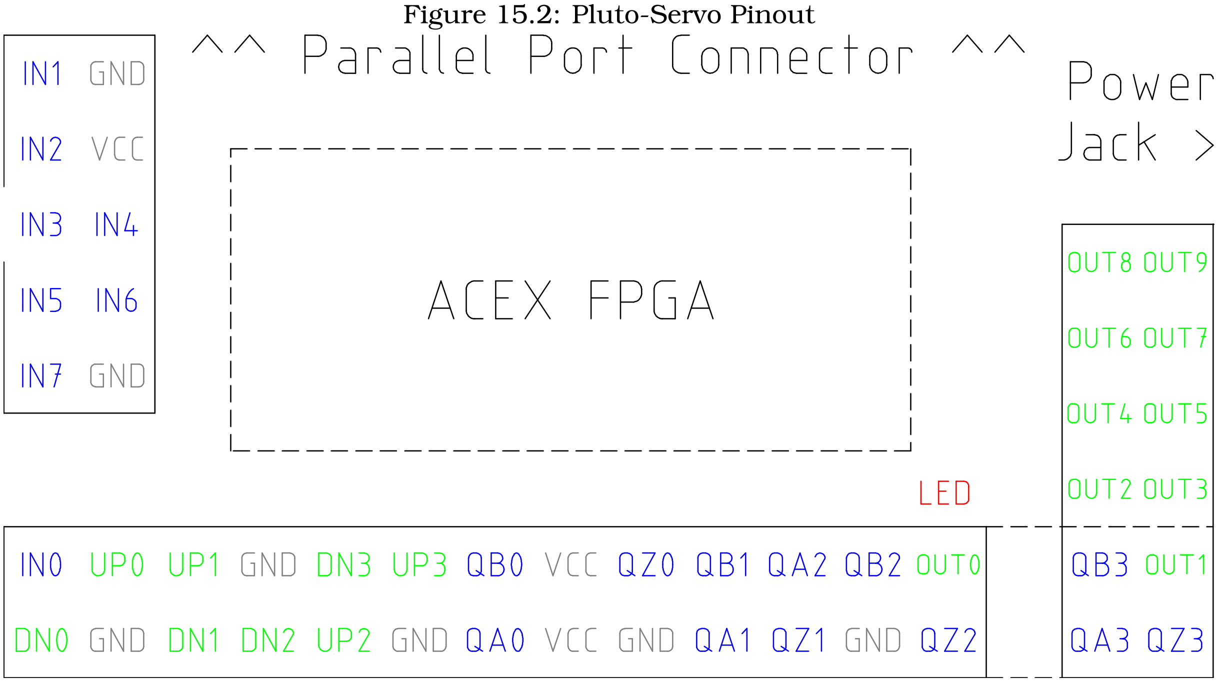
INx Dedicated digital input #x

OUTx Dedicated digital output #x

GND Ground

VCC +3.3V regulated DC

Figure 15.2: Pluto-Servo Pinout



15.9.2 Input latching and output updating

- PWM duty cycles for each channel are updated at different times.
- Digital outputs OUT0 through OUT9 are all updated at the same time. Digital outputs OUT10 through OUT17 are updated at the same time as the pwm function they are shared with.
- Digital inputs IN0 through IN19 are all latched at the same time.
- Quadrature positions for each channel are latched at different times.

15.9.3 HAL Functions, Pins and Parameters

A list of all 'loadrt' arguments, HAL function names, pin names and parameter names is in the manual page, *pluto_servo.9*.

15.9.4 Compatible driver hardware

A schematic for a 2A, 2-axis PWM servo amplifier board is available (http://emergent.unpy.net/projects/01148303608). The L298 H-Bridge (L298 H-bridge http://www.st.com/stonline/books/pdf/docs/1773.pdf) is inexpensive and can easily be used for motors up to 4A (one motor per

Table 15.1: Pluto-Servo Alternate Pin Functions

Primary function	Alternate Function	Behavior if both functions used
UP0	PWM0	When pwm-0-pwmdir is TRUE, this pin is the PWM output
	OUT10	XOR'd with UP0 or PWM0
UP1	PWM1	When pwm-1-pwmdir is TRUE, this pin is the PWM output
	OUT12	XOR'd with UP1 or PWM1
UP2	PWM2	When pwm-2-pwmdir is TRUE, this pin is the PWM output
	OUT14	XOR'd with UP2 or PWM2
UP3	PWM3	When pwm-3-pwmdir is TRUE, this pin is the PWM output
	OUT16	XOR'd with UP3 or PWM3
DN0	DIR0	When pwm-0-pwmdir is TRUE, this pin is the DIR output
	OUT11	XOR'd with DN0 or DIR0
DN1	DIR1	When pwm-1-pwmdir is TRUE, this pin is the DIR output
	OUT13	XOR'd with DN1 or DIR1
DN2	DIR2	When pwm-2-pwmdir is TRUE, this pin is the DIR output
	OUT15	XOR'd with DN2 or DIR2
DN3	DIR3	When pwm-3-pwmdir is TRUE, this pin is the DIR output
	OUT17	XOR'd with DN3 or DIR3
QZ0	IN8	Read same value
QZ1	IN9	Read same value
QZ2	IN10	Read same value
QZ3	IN11	Read same value
QA0	IN12	Read same value
QA1	IN13	Read same value
QA2	IN14	Read same value
QA3	IN15	Read same value
QB0	IN16	Read same value
QB1	IN17	Read same value
QB2	IN18	Read same value
QB3	IN19	Read same value

L298) or up to 2A (two motors per L298) with the supply voltage up to 46V. However, the L298 does not have built-in current limiting, a problem for motors with high stall currents. For higher currents and voltages, some users have reported success with International Rectifier's integrated high-side/low-side drivers. (http://www.cnczone.com/forums/showthread.php?t=25929)

15.10 Pluto-step: 300kHz Hardware Step Generator

Pluto-step is suitable for control of a 3- or 4-axis CNC mill with stepper motors. The large number of inputs allows for a full set of limit switches.

The board features:

- 4 "step+direction" channels with 312.5kHz maximum step rate, programmable step length, space, and direction change times
- 14 dedicated digital outputs
- 16 dedicated digital inputs
- EPP communuication with the PC

15.10.1 Pinout

STEPx The "step" (clock) output of stepgen channel **x**

DIRx The "direction" output of stepgen channel **x**

INx Dedicated digital input #x

OUTx Dedicated digital output #x

GND Ground

VCC +3.3V regulated DC

While the "extended main connector" has a superset of signals usually found on a Step & Direction DB25 connector–4 step generators, 9 inputs, and 6 general-purpose outputs–the layout on this header is different than the layout of a standard 26-pin ribbon cable to DB25 connector.

15.10.2 Input latching and output updating

- Step frequencies for each channel are updated at different times.
- Digital outputs are all updated at the same time.
- Digital inputs are all latched at the same time.
- Feedback positions for each channel are latched at different times.

15.10.3 Step Waveform Timings

The firmware and driver enforce step length, space, and direction change times. Timings are rounded up to the next multiple of $1.6\mu s$, with a maximum of $49.6\mu s$. The timings are the same as for the software stepgen component, except that "dirhold" and "dirsetup" have been merged into a single parameter "dirtime" which should be the maximum of the two, and that the same step timings are always applied to all channels.

Figure 15.3: Pluto-Step Pinout

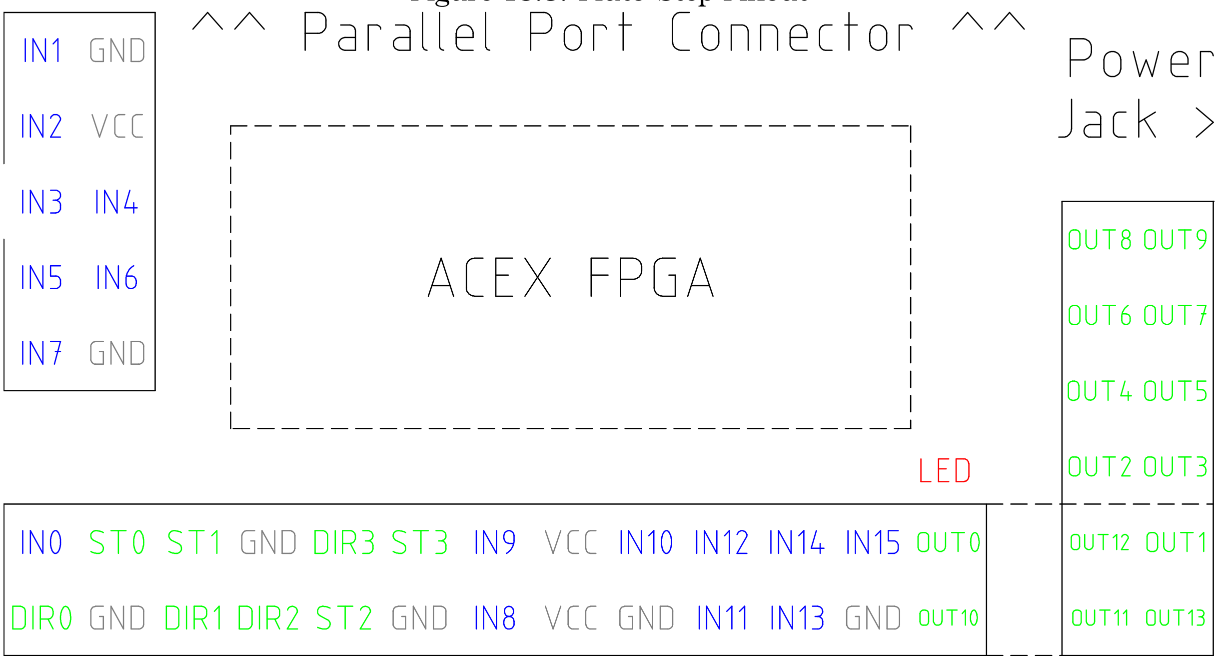
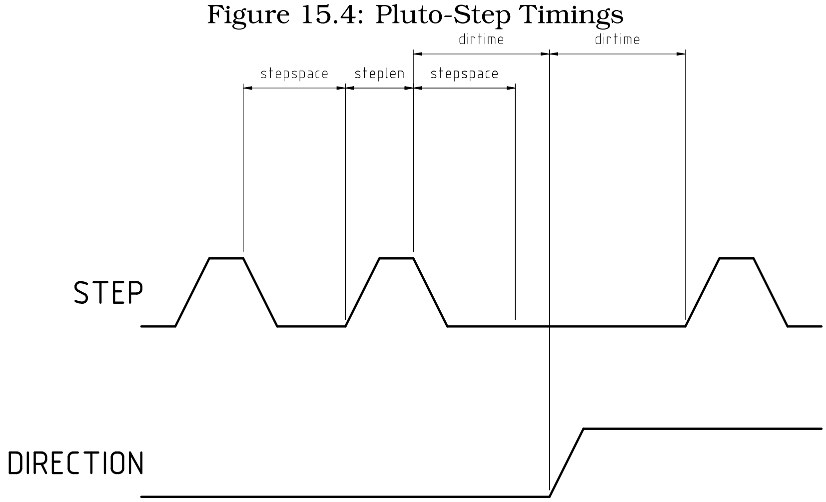


Figure 15.4: Pluto-Step Timings



15.10.4 HAL Functions, Pins and Parameters

A list of all 'loadrt' arguments, HAL function names, pin names and parameter names is in the manual page, *pluto_step.9*.

Chapter 16

Halui

16.1 Introduction

Halui is a HAL based user interface for EMC, it connects HAL pins to NML commands. Most of the functionality (buttons, indicators etc.) that is provided by a traditional GUI (mini, Axis, etc.), is provided by HAL pins in Halui.

The easiest way to use halui is to modify your ini file to include

HALUI = halui

in the [HAL] section.

16.2 Halui pin reference

16.2.1 Machine

- (BIT) halui.machine.on - pin for requestiong machine on
- (BIT) halui.machine.off - pin for requesting machine off
- (BIT) halui.machine.is-on - indicates machine on

16.2.2 E-Stop

- (BIT) halui.estop.activate - pin for requesting E-Stop
- (BIT) halui.estop.reset - pin for requesting E-Stop reset
- (BIT) halui.estop.is-activated - indicates E-stop reset

16.2.3 Mode

- (BIT) halui.mode.manual - pin for requesting manual mode
- (BIT) halui.mode.is_manual - indicates manual mode is on
- (BIT) halui.mode.auto - pin for requesting auto mode
- (BIT) halui.mode.is_auto - indicates auto mode is on
- (BIT) halui.mode.mdi - pin for requesting mdi mode
- (BIT) halui.mode.is_mdi - indicates mdi mode is on

16.2.4 Mist, Flood, Lube

- (BIT) halui.mist.on - pin for requesting mist on
- (BIT) halui.mist.is-on - indicates mist is on
- (BIT) halui.flood.on - pin for requesting flood on
- (BIT) halui.flood.is-on - indicates flood is on
- (BIT) halui.lube.on - pin for requesting lube on
- (BIT) halui.lube.is-on - indicates lube is on

16.2.5 Spindle

- (BIT) halui.spindle.start
- (BIT) halui.spindle.stop
- (BIT) halui.spindle.forward
- (BIT) halui.spindle.reverse
- (BIT) halui.spindle.increase
- (BIT) halui.spindle.decrease
- (BIT) halui.spindle.brake-on - pin for activating spindle-brake
- (BIT) halui.spindle.brake-off - pin for deactivating spindle/brake
- (BIT) halui.spindle.brake-is-on - indicates brake is on

16.2.6 Joints

<channel> is a number between 0 and 7 and 'selected'.

- (BIT) halui.joint.<channel>.home - pin for homing the specific joint
- (BIT) halui.joint.<channel>.on-min-limit-soft - status pin telling joint is at the negative software limit
- (BIT) halui.joint.<channel>.on-max-limit-soft - status pin telling joint is at the positive software limit
- (BIT) halui.joint.<channel>.on-min-limit-hard - status pin telling joint is on the negative hardware limit switch
- (BIT) halui.joint.<channel>.on-max-limit-hard - status pin telling joint is on the positive hardware limit switch
- (BIT) halui.joint.<channel>.fault - status pin telling the joint has a fault
- (BIT) halui.joint.<channel>.homed - status pin telling that the joint is homed

16.2.7 Jogging

<channel> is a number between 0 and 7 and 'selected'.

- (FLOAT) halui.jog.speed - set jog speed
- (BIT) halui.jog.<channel>.minus - jog in negative direction
- (BIT) halui.jog.<channel>.plus - jog in positive direction

16.2.8 Selecting a joint

- (U32) halui.joint.select - select joint (0..7) - internal halui
- (U32) halui.joint.selected - selected joint (0..7) - internal halui
- (BIT) halui.joint.x.select bit - pins for selecting a joint - internal halui
- (BIT) halui.joint.x.is-selected bit - status pin a joint is selected - internal halui

16.2.9 Feed override

- (FLOAT) halui.feed-override.value - current FO value
- (FLOAT) halui.feed-override.scale - pin for setting the scale on changing the FO
- (S32) halui.feed-override.counts - counts from an encoder for example to change FO
- (BIT) halui.feed-override.increase - pin for increasing the FO (+=scale)
- (BIT) halui.feed-override.decrease - pin for decreasing the FO (-=scale)

16.2.10 Spindle override

- (FLOAT) halui.spindle-override.value - current SO value
- (FLOAT) halui.spindle-override.scale - pin for setting the scale on changing the SO
- (S32) halui.spindle-override.counts - counts from an encoder for example to change SO
- (BIT) halui.spindle-override.increase - pin for increasing the SO (+=scale)
- (BIT) halui.spindle-override.decrease - pin for decreasing the SO (-=scale)

16.2.11 Tool

- (U32) halui.tool.number - indicates current selected tool
- (FLOAT) halui.tool.length-offset - indicates current applied tool-length-offset

16.2.12 Program

- (BIT) halui.program.is-idle
- (BIT) halui.program.is-running
- (BIT) halui.program.is-paused
- (BIT) halui.program.run
- (BIT) halui.program.pause
- (BIT) halui.program.resume
- (BIT) halui.program.step

16.2.13 General

- (BIT) halui.abort - pin to send an abort message (clears out most errors)

16.2.14 MDI

Sometimes the user wants to add more complicated tasks to be performed by the activation of a HAL pin. This is possible using the following MDI commands scheme:

- a MDI_COMMAND is added to the ini (in the section [HALUI]) (e.g. [HALUI] MDI_COMMAND = G0 X0
- when halui starts it will read/detect the MDI_COMMAND fields in the ini, and export pins of type (BIT) halui.mdi-command-<nr> (<nr> is a number from 00 to the number of MDI_COMMAND's found in the ini)
- when the pin halui.mdi-command-<nr> is activated halui will try to send the MDI command defined in the ini [1]

16.3 Case - Studies

User descriptions of working halui and hardware EMC control panels here.

[1]This will not always succeed, depending on the operating mode emc2 is in (e.g. while in AUTO halui can't successfully send MDI commands).

Chapter 17

Virtual Control Panels

17.1 Introduction

Traditional machine control panels are large sheets of steel with pushbuttons, knobs, lights and sometimes meters mounted on them. They have many advantages - the buttons are far more rugged than a computer keyboard, and large enough that you can usually operate the correct one by feel while looking elsewhere, for example at the tool. However, they also have disadvantages. The occupy a lot of panel space, they are expensive, and wiring them into the PC can use up a lot of I/O pins. That is where Virtual Control Panels come in.

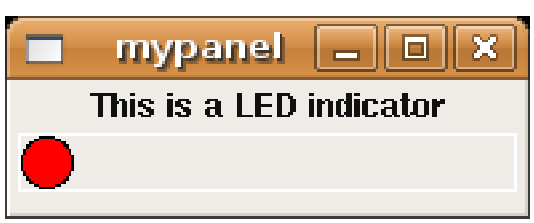
A Virtual Control Panel (VCP) is a window on the computer screen with buttons, meters, switches, etc. When you click on a VCP button, it changes the state of a HAL pin, exactly as if you had pressed a physical button wired to an input pin on an I/O card. Likewise, a VCP LED lights up when a HAL pin goes true, just like a physical indicator lamp wired to an output pin on an I/O card. Virtual control panels are not intended to replace physical panels - sometimes there is just no substiture for a big rugged oil-tight pushbutton. But virtual panels can be used for testing or monitoring things that don't require physical buttons and lights, to temporarily replace real I/O devices while debugging ladder logic, or perhaps to simulate a physical panel before you build it and wire it to an I/O board.

Currently there are two VCP implementations included with EMC2: The older, simply named VCP, which used GTK widgets, and the newer, called pyVCP, which uses Tkinter widgets. VCP is deprecated and should not be used - it may be removed in the future.

17.2 pyVCP

The layout of a pyVCP panel is specified with an XML file that contains widget tags between <pyvcp> and </pyvcp>. For example:

```
<pyvcp>
    <label text="This is a LED indicator"/>
    <led/>
</pyvcp>
```



If you place this text in a file called tiny.xml, and run

```
pyvcp -c mypanel tiny.xml
```

pyVCP will create the panel for you, which includes two widgets, a Label with the text “This is a LED indicator”, and a LED, used for displaying the state of a HAL BIT signal. It will also create a HAL component named “mypanel” (all widgets in this panel are connected to pins that start with “mypanel.”). Since no <halpin> tag was present inside the <led> tag, pyVCP will automatically name the HAL pin for the LED widget mypanel.led.0

For a list of widgets and their tags and options, see the widget reference below.

Once you have created your panel, connecting HAL signals to and form the pyVCP pins is done with ’halcmd linksp’ as usual. If you are new to HAL, the HAL Tutorial7 is recommended.

17.3 Security of pyVCP

Parts of pyVCP files are evaluated as Python code, and can take any action available to Python programs. Only use pyVCP .xml files from a source that you trust.

17.4 Using pyVCP with AXIS

Since AXIS uses the same GUI toolkit (Tkinter) as pyVCP, it is possible to include a pyVCP panel on the right side of the normal AXIS user interface. A typical example is explained below.

Place your pyVCP XML file describing the panel in the same directory where your .ini file is. Say we we want to display the current spindle speed using a Bar widget. Place the following in a file called spindle.xml:

```
<pyvcp>
    <label>
        <text>"Spindle speed:"</text>
    </label>
    <bar>
        <halpin>"spindle-speed"</halpin>
        <max_>5000</max_>
    </bar>
</pyvcp>
```

Here we’ve made a panel with a Label and a Bar widget, specified that the HAL pin connected to the Bar should be named “spindle-speed”, and set the maximum value of the bar to 5000 (see widget reference below for all options). To make AXIS aware of this file, and call it at startup, we need to specify the following in the [DISPLAY] section of the .ini file:

```
PYVCP = spindle.xml
```

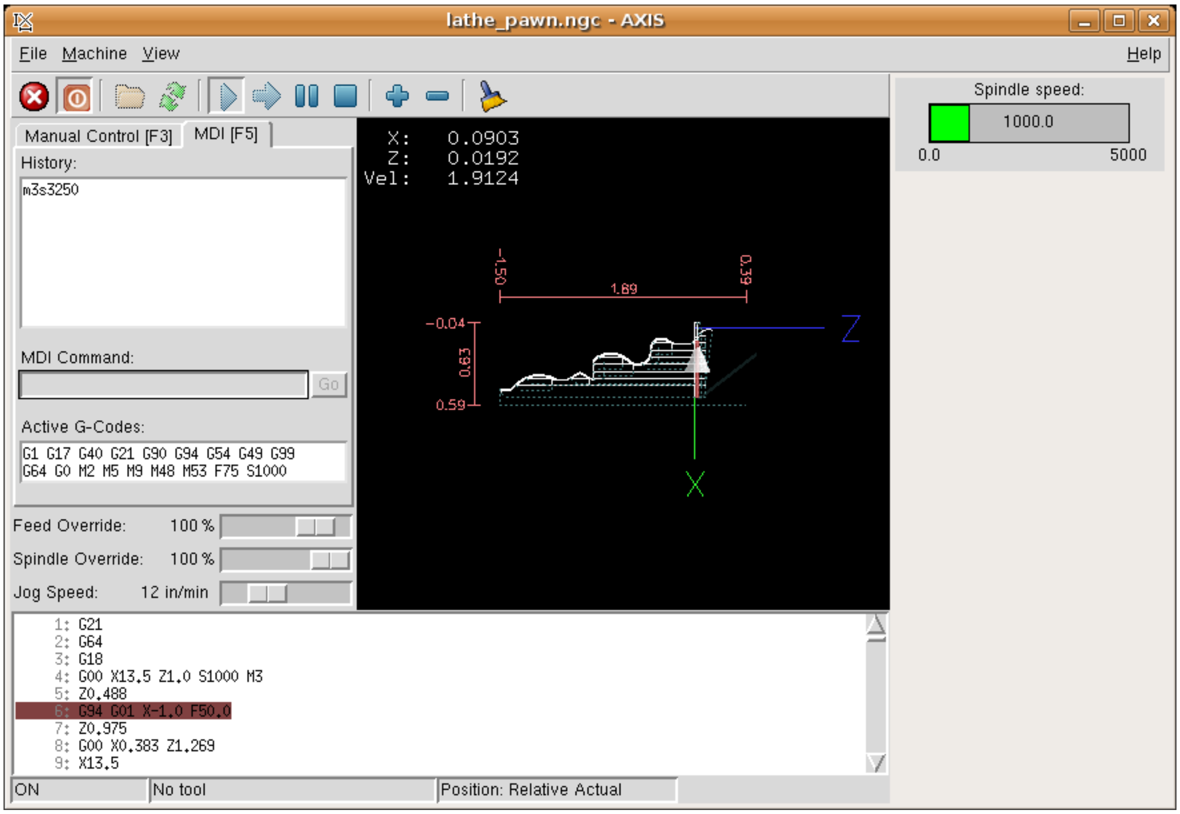
To make our widget actually display the spindle-speed it needs to be hooked up to the appropriate HAL signal. A .hal file that will be run once AXIS and pyVCP have started can be specified in the [HAL] section of the .ini file:

```
POSTGUI_HALFILE = spindle_to_pyvcp.hal
```

This change will run the HAL commands specified in "spindle_to_pyvcp.hal". In our example the contents could look like this:

```
linksp spindle-rpm-filtered  pyvcp.spindle-speed
```

assuming that a singlal called "spindle-rpm-filtered" already exists. Note that when running together with AXIS, all pyVCP widget HAL pins have names that start with "pyvcp.".



This is what the newly created pyVCP panel should look like in AXIS. The `sim/lathe` configuration is already configured this way.

17.5 pyVCP Widget reference

HAL signals come in two variants, BIT and FLOAT. pyVCP can either display the value of the signal with an indicator widget, or modify the signal value with a control widget. Thus there are four classes of pyVCP widgets that you can connect to a HAL signal. A fifth class of helper widgets allow you to organize and label your panel.

1. Widgets for indicating BIT signals: LED
2. Widgets for controlling BIT signals: Button, Checkbutton, Radiobutton
3. Widgets for indicating FLOAT signals: Number, Bar, Meter
4. Widgets for controlling FLOAT signals: Spinbox, Scale, Jogwheel
5. Helper widgets: Hbox, Vbox, Tabel, Label, Labelframe

17.5.0.1 Syntax

Each widget is described briefly, followed by the markup used, and a screenshot. All tags inside the main widget tag are optional.

17.5.0.2 General Notes

At the present time, both a tag-based and an attribute-based syntax are supported. For instance, the following XML fragments are treated identically:

```
<led halpin="my-led"/>
```

and

```
<led><halpin>"my-led"</halpin></led>
```

When the attribute-based syntax is used, the following rules are used to turn the attribute's value into a Python value:

1. If the first character of the attribute is one of the following, it is evaluated as a Python expression: `{(["'`
2. If the string is accepted by int(), the value is treated as an integer
3. If the string is accepted by float(), the value is treated as floating-point
4. Otherwise, the string is accepted as a string.

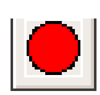
When the tag-based syntax is used, the text within the tag is always evalued as a Python expression.

The examples below show a mix of formats.

17.5.1 LED

A LED is used to indicate the status of a BIT signal. The LED color will be on_color when the BIT signal is true, and off_color otherwise.

```
<led>
    <halpin>"my-led"</halpin>
    <size>50</size>
    <on_color>"blue"</on_color>
    <off_color>"black"</off_color>
</led>
```



<halpin> sets the name of the pin, default is “led.n”, where n is an integer
<size> sets the size of the led, default is 20
<on_color> sets the color of the LED when the pin is true. default is “green”
<off_color> sets the color of the LED when the pin is false. default is “ref”

17.5.2 Button

A button is used to control a BIT pin. The pin will be set True when the button is pressed and held down, and will be set False when the button is released.

```
<button>
    <halpin>"my-button"</halpin>
    <text>"ON"</text>
</button>
```

17.5.3 Checkbutton

A checkbutton controls a BIT pin. The pin will be set True when the button is checked, and false when the button is unchecked.

```
<checkbutton>
    <halpin>"my-checkbutton"</halpin>
</checkbutton>
```

An unchecked checkbutton:  , and a checked one: 

17.5.4 Radiobutton

A radiobutton will set one of a number of BIT pins true. The other pins are set false.

```
<radiobutton>
    <choices>["one","two","three"]</choices>
    <halpin>"my-radio"</halpin>
</radiobutton>
```

Note that the HAL pins in the example above will me named my-radio.one, my-radio.two, and my-radio.three. In the image above, “three” is the selected value.

17.5.5 Number

The number widget displays the value of a FLOAT signal.

```
<number>
    <halpin>"my-number"</halpin>
    <font>('Helvetica',50)</font>
    <format>"+4.3f"</format>
</number>
```

<font> is a Tkinter font type and size specification. Note that on Ubuntu 6.06 'Helvetica' is not available in sizes above ca 40 or 50. One font that will show up to at least size 200 is 'courier 10 pitch', so for a really big Number widget you could specify:

```
<font>('courier 10 pitch',100)</font>
```

<format> is a 'C-style' format specified that determines how the number is displayed.

17.5.6 Bar

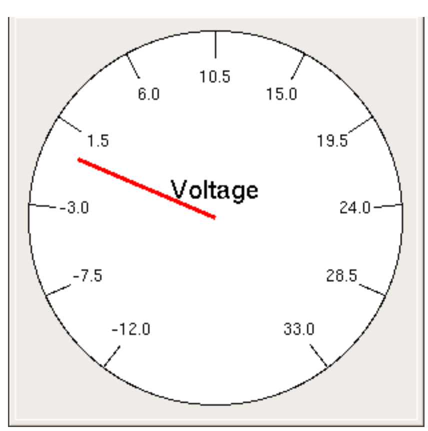
A bar widget displays the value of a FLOAT signal both graphically using a bar display and numerically.

```
<bar>
    <halpin>"my-bar"</halpin>
    <min_>0</min_>
    <max_>123</max_>
    <bgcolor>"grey"</bgcolor>
    <fillcolor>"red"</fillcolor>
</bar>
```

17.5.7 Meter

Meter displays the value of a FLOAT signal using a traditional dial indicator.

```
<meter>
    <halpin>"my-meter"</halpin>
    <text>"Voltage"</text>
    <size>300</size>
    <min_>-12</min_>
    <max_>33</max_>
</meter>
```



17.5.8 Spinbox

Spinbox controls a FLOAT pin. You increase or decrease the value of the pin by 'resolution' by either pressing on the arrows, or pointing at the spinbox and rolling your mouse-wheel.

```
<spinbox>
    <halpin>"my-spibox"</halpin>
    <min_>-12</min_>
    <max_>33</max_>
    <resolution>0.1</resolution>
    <format>"2.3f"</format>
    <font>('Arial',30)</font>
</spinbox>
```

17.5.9 Scale

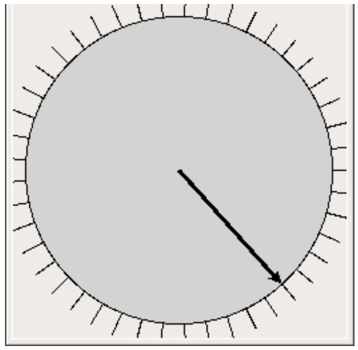
Scale controls a FLOAT pin. You increase or decrease the value of the pin be either dragging the slider, or pointing at the scale and rolling your mouse-wheel.

```
<scale>
    <halpin>"my-scale"</halpin>
    <resolution>0.1</resolution>
    <orient>HORIZONTAL</orient>
    <min_>-33</min_>
    <max_>26</max_>
</scale>
```

17.5.10 Jogwheel

Jogwheel mimics a real jogwheel by outputting a FLOAT pin which counts up or down as the wheel is turned, either by dragging in a circular motion, or by rolling the mouse-wheel.

```
<jogwheel>
    <halpin>"my-wheel"</halpin>
    <cpr>45</cpr>
    <size>250</size>
</jogwheel>
```



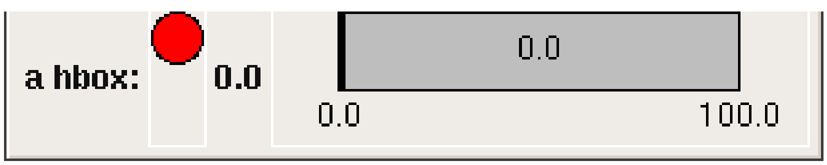
17.6 pyVCP Container reference

Containers are widgets that contain other widgets.

17.6.1 Hbox

Use a Hbox when you want to stack widgets horizontally next to eachother.

```
<hbox>
    <label><text>"a vbox:"</text></label>
    <led></led>
    <number></number>
    <bar></bar>
</hbox>
```

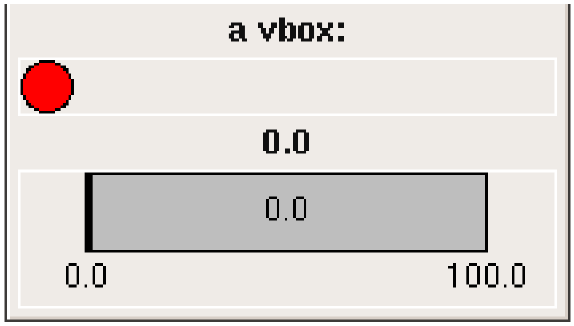


Inside a Hbox, you can use the `<boxfill fill=""/>`, `<boxanchor anchor=""/>`, and `<boxexpand expand=""/>` tags to choose how items in the box behave when the window is resized. For details of how fill, anchor, and expand behave, refer to the Tk `pack` manual page, `pack(3tk)`. By default, `fill='y'`, `anchor='center'`, `expand='yes'`.

17.6.2 Vbox

Use a Vbox when you want to stack widgets vertically on top of each other.

```
<vbox>
    <label><text>"a vbox:"</text></label>
    <led></led>
    <number></number>
    <bar></bar>
</vbox>
```



Inside a Hbox, you can use the `<boxfill fill=""/>`, `<boxanchor anchor=""/>`, and `<boxexpand expand=""/>` tags to choose how items in the box behave when the window is resized. For details of how fill, anchor, and expand behave, refer to the Tk `pack` manual page, `pack(3tk)`. By default, `fill='x', anchor='center', expand='yes'`.

17.6.3 Label

A label is a piece of text on your panel.

```
<label>
    <text>"This is a Label:"</text>
    <font>('Helvetica',20)</font>
</label>
```

17.6.4 Labelframe

A labelframe is a frame with a groove and a label at the upper-left corner.

```
<labelframe text="Group Title">
  <hbox>
    <led/> <led/>
  </hbox>
</labelframe>
```

17.6.5 Table

A table is a container that allows layout in a grid of rows and columns. Each row is started by a `<tablerow/>` tag. A contained widget may span rows or columns through the use of the `<tablespan rows= cols=/>` tag. The sides of the cells to which the contained widgets "stick" may be set through the use of the `<tablesticky sticky=/>` tag. A table expands on its flexible rows and columns.

Example:

```
<table flexible_rows="[2]" flexible_columns="[1,4]">
   <tablesticky sticky="new"/>
   <tablerow/>
    <label text="A (cell 1,1)"/>
    <label text="B (cell 1,2)"/>
```

```
     <tablespan columns="2"/><label text="C, D (cells 1,3 and 1,4)">
    <tablerow/>
     <label text="E (cell 2,1)"/>
     <tablesticky sticky="nsew"/><tablespan rows="2"/>
         <label text="'spans\n2 rows'"/>
     <tablesticky sticky="new"/><label text="G (cell 2,3)"/>
     <label text="H (cell 2,4)"/>
    <tablerow/>
     <label text="J (cell 3,1)"/>
     <label text="K (cell 3,2)"/>
     <label text="M (cell 3,4)"/>
</table>
```

17.7 VCP: A small example

NOTE: VCP is deprecated, and will most likely not be getting any new development or additional widgets. We strongly recommend using pyVCP. However, pyVCP won't be released until version 2.2 comes out, and VCP is in version 2.1. That means some people will wind up using VCP, so we can't simply drop it.[1]

Place the following in the file `tiny.vcp`:

```
vcp {
 main-window {
   box {
     button {
       halpin = vcp.pushbutton
       label { text = "Push Me" }
     }
     LED {
       halpin = vcp.light
     }
   }
 }
}
```

The above file describes a tiny Virtual Control Panel, with one push button, and one light. To see what it looks like, we need to start HAL:

```
$ halrun
```

Next we load halvcp, and give it the name of our .vcp file:

```
halcmd: loadusr halvcp tiny.vcp
halcmd:
```

There may be some text printed as halvcp parses the tiny.vcp file, but when it finishes, there should be a small window on your screen, with a button and an LED. It will look something like figure 17.1.

So, we have a button and an LED, but they aren't connected to anything, so nothing happens when you push the button. However, the LED and the button both have HAL pins associated with them:

[1] A .vcp to .xml translator that takes a vcp file and turns it into one that pyVCP can use is on my to-do list. That would enable VCP users to easily switch over to pyVCP. If such a translator is written, VCP may be removed from the version 2.2 release.

Figure 17.1: tiny.vcp on the screen

```
halcmd: show pin
Component Pins:
Owner  Type  Dir      Value      Name
 03    bit   IN       FALSE      vcp.light
 03    bit   OUT      FALSE      vcp.pushbutton
halcmd:
```

To make something happen, we can connect a HAL signal between the button and the light:

```
halcmd: newsig jumper bit
halcmd: linksp jumper vcp.pushbutton
halcmd: linksp jumper vcp.light
halcmd: show sig
Signals:
Type       Value      Name
bit        FALSE      jumper
                          ==> vcp.light
                          <== vcp.pushbutton
halcmd:
```

Now push the button, and the the LED should light up!

17.8 VCP: Another small example with EMC

Place the following in the file `estop.vcp`:

```
vcp {
   main-window {
      toggle { halpin = vcp.estop }
   }
}
```

In your .hal file, remove any existing signal linked to `iocontrol.0.emc-enable-in` and add the following lines:

```
loadusr -W halvcp estop.vcp
newsig estop bit
linkps vcp.estop => estop
linkps estop => iocontrol.0.emc-enable-in
```

Now, when running your machine, the ESTOP button in the GUI is disabled, and the ESTOP button in the VCP window is used instead.

17.9 VCP Syntax

17.9.1 Block

A block's format is:

```
tag { contents }
```

The contents can consist of attributes that describe the block, or other blocks that nest inside it.

A attribute's format is

```
name = value
```

The attribute names that are acceptable for each block depend on the block tag, and will be listed later.

Part VI

Advanced topics

Chapter 18

Kinematics in EMC2

18.1 Introduction

When we talk about CNC machines, we usually think about machines that are commanded to move to certain locations and perform various tasks. In order to have an unified view of the machine space, and to make it fit the human point of view over 3D space, most of the machines (if not all) use a common coordinate system called the Cartesian Coordinate System.

The Cartesian Coordinate system is composed of 3 axes (X, Y, Z) each perpendicular to the other 2. [1]

When we talk about a G-code program (RS274NGC) we talk about a number of commands (G0, G1, etc.) which have positions as parameters (X- Y- Z-). These positions refer exactly to Cartesian positions. Part of the EMC2 motion controller is responsible for translating those positions into positions which correspond to the machine kinematics[2].

18.1.1 Joints vs. Axes

A joint of a CNC machine is a one of the physical degrees of freedom of the machine. This might be linear (leadscrews) or rotary (rotary tables, robot arm joints). There can be any number of joints on a certain machine. For example a typical robot has 6 joints, and a typical simple milling machine has only 3.

There are certain machines where the joints are layed out to match kinematics axes (joint 0 along axis X, joint 1 along axis Y, joint 2 along axis Z), and these machines are called Cartesian machines (or machines with Trivial Kinematics). These are the most common machines used in milling, but are not very common in other domains of machine control (e.g. welding: puma-typed robots).

18.2 Trivial Kinematics

As we said there is a group of machines in which each joint is placed along one of the Cartesian axes. On these machines the mapping from Cartesian space (the G-code program) to the joint space (the actual actuators of the machine) is trivial. It is a simple 1:1 mapping:

```
pos->tran.x = joints[0];
pos->tran.y = joints[1];
```

[1] The word “axes” is also commonly (and wrongly) used when talking about CNC machines, and referring to the moving directions of the machine.

[2] Kinematics: a two way function to transform from Cartesian space to joint space

```
pos->tran.z = joints[2];
pos->a = joints[3];
pos->b = joints[4];
pos->c = joints[5];
```

In the above code snippet one can see how the mapping is done: the X position is identical with the joint 0, Y with joint 1 etc. The above refers to the direct kinematics (one way of the transformation) whereas the next code part refers to the inverse kinematics (or the inverse way of the transformation):

```
joints[0] = pos->tran.x;
joints[1] = pos->tran.y;
joints[2] = pos->tran.z;
joints[3] = pos->a;
joints[4] = pos->b;
joints[5] = pos->c;
```

As one can see, it's pretty straightforward to do the transformation for a trivial kins (or Cartesian) machine. It gets a bit more complicated if the machine is missing one of the axes.[3][4]

18.3 Non-trivial kinematics

There can be quite a few types of machine setups (robots: puma, scara; hexapods etc.). Each of them is set up using linear and rotary joints. These joints don't usually match with the Cartesian coordinates, therefor there needs to be a kinematics function which does the conversion (actually 2 functions: forward and inverse kinematics function).

To illustrate the above, we will analyze a simple kinematics called bipod (a simplified version of the tripod, which is a simplified version of the hexapod).

The Bipod we are talking about is a device that consists of 2 motors placed on a wall, from which a device is hanged using some wire. The joints in this case are the distances from the motors to the device (named AD and BD in figure 18.1).

The position of the motors is fixed by convention. Motor A is in (0,0), which means that its X coordinate is 0, and its Y coordinate is also 0. Motor B is placed in (Bx, 0), which means that its X coordinate is Bx.

Our tooltip will be in point D which gets defined by the distances AD and BD, and by the Cartesian coordinates Dx, Dy.

The job of the kinematics is to transform from joint lengths (AD, BD) to Cartesian coordinates (Dx, Dy) and vice-versa.

18.3.1 Forward transformation

To transform from joint space into Cartesian space we will use some trigonometry rules (the right triangles determined by the points (0,0), (Dx,0), (Dx,Dy) and the triangle (Dx,0), (Bx,0) and (Dx,Dy).

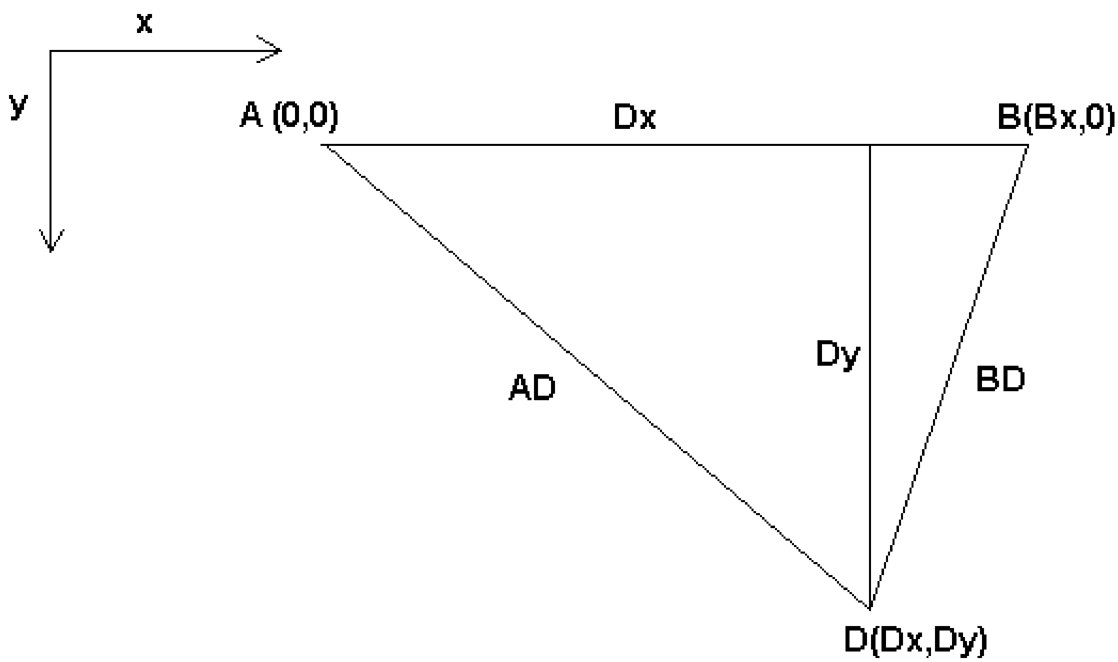
we can easily see that $AD^2 = x^2 + y^2$, likewise $BD^2 = (Bx - x)^2 + y^2$.

If we subtract one from the other we will get:

[3] If a machine (e.g. a lathe) is set up with only the axes X,Z & A, and the EMC2 inifile holds only these 3 joints defined, then the above matching will be faulty. That is because we actually have (joint0=x, joint1=Z, joint2=A) whereas the above assumes joint1=Y. To make it easily work in EMC2 one needs to define all axes (XYZA), then use a simple loopback in HAL for the unused Y axis.

[4] One other way of making it work, is by changing the matching code and recompiling the software.

Figure 18.1: Bipod setup



$$AD^2 - BD^2 = x^2 + y^2 - x^2 + 2 * x * Bx - Bx^2 - y^2$$

and therefore:

$$x = \frac{AD^2 - BD^2 + Bx^2}{2 * Bx}$$

From there we calculate:

$$y = \sqrt{AD^2 - x^2}$$

Note that the calculation for y involves the square root of a difference, which may not result in a real number. If there is no single Cartesian coordinate for this joint position, then the position is said to be a singularity. In this case, the forward kinematics return -1.

Translated to actual code:

```
double AD2 = joints[0] * joints[0];
double BD2 = joints[1] * joints[1];
double x = (AD2 - BD2 + Bx * Bx) / (2 * Bx);
double y2 = AD2 - x * x;
if(y2 < 0) return -1;
pos->tran.x = x;
pos->tran.y = sqrt(y2);
return 0;
```

18.3.2 Inverse transformation

The inverse kinematics is lots easier in our example, as we can write it directly:

$$AD = \sqrt{x^2 + y^2}$$

$$BD = \sqrt{(Bx - x)^2 + y^2}$$

or translated to actual code:

```
double x2 = pos->tran.x * pos->tran.x;
double y2 = pos->tran.y * pos->tran.y;
joints[0] = sqrt(x2 + y2);
joints[1] = sqrt((Bx - pos->tran.x)*(Bx - pos->tran.x) + y2);
return 0;
```

18.4 Implementation details

A kinematics module is implemented as a HAL component, and is permitted to export pins and parameters. It consists of several functions:

- `int kinematicsForward(const double *joint, EmcPose *world, const KINEMATICS_FORWARD_FL *fflags, KINEMATICS_INVERSE_FLAGS *iflags)`

  Implements the forward kinematics function as described in section 18.3.1.

- `extern int kinematicsInverse(const EmcPose * world, double *joints, const KINEMATICS_I *iflags, KINEMATICS_FORWARD_FLAGS *fflags)`

  Implements the inverse kinematics function as described in section 18.3.2.

- `extern KINEMATICS_TYPE kinematicsType(void)`

  Returns the kinematics type identifier.

- `int kinematicsHome(EmcPose *world, double *joint, KINEMATICS_FORWARD_FLAGS *fflags, KINEMATICS_INVERSE_FLAGS *iflags)`

  The home kinematics function sets all its arguments to their proper values at the known home position. When called, these should be set, when known, to initial values, e.g., from an INI file. If the home kinematics can accept arbitrary starting points, these initial values should be used.

- int rtapi_app_main(void)

- void rtapi_app_exit(void)

  These are the standard setup and tear-down functions of RTAPI modules.

Part VII

Tuning

18.5 Tuning servo systems

18.5.1 PID Controller

A proportional-integral-derivative controller (PID controller) is a common feedback loop component in industrial control systems.[5]

The Controller compares a measured value from a process (typically an industrial process) with a reference setpoint value. The difference (or "error" signal) is then used to calculate a new value for a manipulatable input to the process that brings the process' measured value back to its desired setpoint.

Unlike simpler control algorithms, the PID controller can adjust process outputs based on the history and rate of change of the error signal, which gives more accurate and stable control. (It can be shown mathematically that a PID loop will produce accurate, stable control in cases where a simple proportional control would either have a steady-state error or would cause the process to oscillate).

18.5.1.1 Control loop basics

Intuitively, the PID loop tries to automate what an intelligent operator with a gauge and a control knob would do. The operator would read a gauge showing the output measurement of a process, and use the knob to adjust the input of the process (the "action") until the process's output measurement stabilizes at the desired value on the gauge.

In older control literature this adjustment process is called a "reset" action. The position of the needle on the gauge is a "measurement", "process value" or "process variable". The desired value on the gauge is called a "setpoint" (also called "set value"). The difference between the gauge's needle and the setpoint is the "error".

A control loop consists of three parts:

1. Measurement by a sensor connected to the process (e.g. encoder),
2. Decision in a controller element,
3. Action through an output device such as an motor.

As the controller reads a sensor, it subtracts this measurement from the "setpoint" to determine the "error". It then uses the error to calculate a correction to the process's input variable (the "action") so that this correction will remove the error from the process's output measurement.

In a PID loop, correction is calculated from the error in three ways: cancel out the current error directly (Proportional), the amount of time the error has continued uncorrected (Integral), and anticipate the future error from the rate of change of the error over time (Derivative).

A PID controller can be used to control any measurable variable which can be affected by manipulating some other process variable. For example, it can be used to control temperature, pressure, flow rate, chemical composition, speed, or other variables. Automobile cruise control is an example of a process outside of industry which utilizes crude PID control.

Some control systems arrange PID controllers in cascades or networks. That is, a "master" control produces signals used by "slave" controllers. One common situation is motor controls: one often wants the motor to have a controlled speed, with the "slave" controller (often built into a variable frequency drive) directly managing the speed based on a proportional input. This "slave" input is fed by the "master" controllers' output, which is controlling based upon a related variable.

[5] This Subsection is taken from an much more extensive article found at http://en.wikipedia.org/wiki/PID_controller

18.5.1.2 Theory

"PID" is named after its three correcting calculations, which all add to and adjust the controlled quantity. These additions are actually "subtractions" of error, because the proportions are usually negative:

18.5.1.2.0.1 Proportional To handle the present, the error is multiplied by a (negative) constant P (for "proportional"), and added to (subtracting error from) the controlled quantity. P is only valid in the band over which a controller's output is proportional to the error of the system. Note that when the error is zero, a proportional controller's output is zero.

18.5.1.2.0.2 Integral To learn from the past, the error is integrated (added up) over a period of time, and then multiplied by a (negative) constant I (making an average), and added to (subtracting error from) the controlled quantity. I averages the measured error to find the process output's average error from the setpoint. A simple proportional system either oscillates, moving back and forth around the setpoint because there's nothing to remove the error when it overshoots, or oscillates and/or stabilizes at a too low or too high value. By adding a negative proportion of (i.e. subtracting part of) the average error from the process input, the average difference between the process output and the setpoint is always being reduced. Therefore, eventually, a well-tuned PID loop's process output will settle down at the setpoint.

18.5.1.2.0.3 Derivative To handle the future, the first derivative (the slope of the error) over time is calculated, and multiplied by another (negative) constant D, and also added to (subtracting error from) the controlled quantity. The derivative term controls the response to a change in the system. The larger the derivative term, the more rapidly the controller responds to changes in the process's output.

More technically, a PID loop can be characterized as a filter applied to a complex frequency-domain system. This is useful in order to calculate whether it will actually reach a stable value. If the values are chosen incorrectly, the controlled process input can oscillate, and the process output may never stay at the setpoint.

18.5.1.3 Loop Tuning

"Tuning" a control loop is the adjustment of its control parameters (gain/proportional band, integral gain/reset, derivative gain/rate) to the optimum values for the desired control response. The optimum behavior on a process change or setpoint change varies depending on the application. Some processes must not allow an overshoot of the process variable from the setpoint. Other processes must minimize the energy expended in reaching a new setpoint. Generally stability of response is required and the process must not oscillate for any combination of process conditions and setpoints.

Tuning of loops is made more complicated by the response time of the process; it may take minutes or hours for a setpoint change to produce a stable effect. Some processes have a degree of non-linearity and so parameters that work well at full-load conditions don't work when the process is starting up from no-load. This section describes some traditional manual methods for loop tuning.

There are several methods for tuning a PID loop. The choice of method will depend largely on whether or not the loop can be taken "offline" for tuning, and the response speed of the system. If the system can be taken offline, the best tuning method often involves subjecting the system to a step change in input, measuring the output as a function of time, and using this response to determine the control parameters.

18.5.1.3.0.4 Simple method If the system must remain online, one tuning method is to first set the I and D values to zero. Increase the P until the output of the loop oscillates. Then increase I until oscillation stops. Finally, increase D until the loop is acceptably quick to reach its reference. A fast PID loop tuning usually overshoots slightly to reach the setpoint more quickly; however, some systems cannot accept overshoot.

Parameter	Rise Time	Overshoot	Settling Time	S.S. Error
P	Decrease	Increase	Small Change	Decrease
I	Decrease	Increase	Increase	Eliminate
D	Small Change	Decrease	Decrease	Small Change

Effects of increasing parameters

18.5.1.3.0.5 Ziegler-Nichols method Another tuning method is formally known as the "Ziegler-Nichols method", introduced by John G. Ziegler and Nathaniel B. Nichols. It starts in the same way as the method described before: first set the I and D gains to zero and then increase the P gain until the output of the loop starts to oscillate. Write down the critical gain (K_c) and the oscillation period of the output (P_c). Then adjust the P, I and D controls as the table shows:

Control type	P	I	D
P	$.5K_c$		
PI	$.45K_c$	$1.2/P_c$	
PID	$.6K_c$	$2/P_c$	$P \times P_c/8$

18.6 Tuning stepper systems

18.6.1 Getting the most out of Software Stepping

Generating step pulses in software has one very big advantage - it's free. Just about every PC has a parallel port that is capable of outputting step pulses that are generated by the software. However, software step pulses also have some disadvantages:

- limited maximum step rate
- jitter in the generated pulses
- loads the CPU

This chapter has some steps that can help you get the best results from software generated steps.

18.6.1.1 Run a Latency Test

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

Latency is far more important than CPU speed. A lowly Pentium II that responds to interrupts within 10 microseconds each and every time can give better results than the latest and fastest P4 Hyperthreading beast.

The CPU isn't the only factor in determining latency. Motherboards, video cards, USB ports, and a number of other things can hurt the latency. The best way to find out what you are dealing with is to run the RTAI latency test.

> DO NOT TRY TO RUN EMC2 WHILE THE TEST IS RUNNING

On Ubuntu Dapper, you can run the test by opening a shell and doing:

```
sudo mkdir /dev/rtf;
sudo mknod /dev/rtf/3 c 150 3;
sudo mknod /dev/rtf3 c 150 3;
cd /usr/realtime*/testsuite/kern/latency; ./run
```

and then you should see something like this:

```
ubuntu:/usr/realtime-2.6.12-magma/testsuite/kern/latency$ ./run
*
*
* Type ^C to stop this application.
*
*
## RTAI latency calibration tool ##
# period = 100000 (ns)
# avrgtime = 1 (s)
# do not use the FPU
# start the timer
# timer_mode is oneshot
RTAI Testsuite - KERNEL latency (all data in nanoseconds)
```

```
RTH| lat min| ovl min| lat avg| lat max| ovl max| overruns
RTD|   -1571|   -1571|    1622|    8446|    8446|       0
RTD|   -1558|   -1571|    1607|    7704|    8446|       0
RTD|   -1568|   -1571|    1640|    7359|    8446|       0
RTD|   -1568|   -1571|    1653|    7594|    8446|       0
RTD|   -1568|   -1571|    1640|   10636|   10636|       0
RTD|   -1568|   -1571|    1640|   10636|   10636|       0
```

While the test is running, you should "abuse" the computer. Move windows around on the screen. Surf the web. Copy some large files around on the disk. Play some music. Run an OpenGL program such as glxgears. The idea is to put the PC through its paces while the latency test checks to see what the worst case numbers are.

The last number in the column labeled "ovl max" is the most important. Write it down - you will need it later. It contains the worst latency measurement during the entire run of the test. In the example above, that is 10636 nano-seconds, or 10.6 micro-seconds, which is excellent. However the example only ran for a few seconds (it prints one line every second). You should run the test for at least several minutes; sometimes the worst case latency doesn't happen very often, or only happens when you do some particular action. I had one Intel motherboard that worked pretty well most of the time, but every 64 seconds it had a very bad 300uS latency. Fortunately that is fixable, see FixingDapperSMIIssues in the wiki found at wiki.linuxcnc.org.

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

Note that if you get high numbers, there may be ways to improve them. For example, one PC had very bad latency (several milliseconds) when using the onboard video. But a $5 used Matrox video card solved the problem - EMC does not require bleeding edge hardware.

18.6.1.2 Figure out what your drives expect

Different brands of stepper drives have different timing requirements on their step and direction inputs. So you need to dig out (or Google for) the data sheet that has your drive's specs.

> For example, the Gecko G202 manual says this:
> Step Frequency: 0 to 200 kHz
> Step Pulse “0” Time: 0.5 uS min (Step on falling edge)
> Step Pulse “1” Time: 4.5 uS min
> Direction Setup: 1 uS min (20 uS min hold time after Step edge)
>
> The Gecko G203V specifications are:
> Step Frequency: 0 to 333 kHz
> Step Pulse “0” Time: 2.0 uS min (Step on rising edge)
> Step Pulse “1” Time: 1.0 uS min
> Direction Setup:
>
> > 200 nS (0.2uS) before step pulse rising edge
> > 200 nS (0.2uS) hold after step pulse rising edge

A Xylotex drive datasheet has a nice drawing of the timing requirements, which are:

Minimum DIR setup time before rising edge of STEP Pulse 200nS Minimum
DIR hold time after rising edge of STEP pulse 200nS
Minimum STEP pulse high time 2.0uS
Minimum STEP pulse low time 1.0uS
Step happens on rising edge

Once you find the numbers, write them down too - you need them in the next step.

18.6.1.3 Choose your BASE_PERIOD

BASE_PERIOD is the "heartbeat" of your EMC computer. Every period, the software step generator decides if it is time for another step pulse. A shorter period will allow you to generate more pulses per second, within limits. But if you go too short, your computer will spend so much time generating step pulses that everything else will slow to a crawl, or maybe even lock up. Latency and stepper drive requirements affect the shortest period you can use, as we will see in a minute.

Let's look at the Gecko example first. The G202 can handle step pulses that go low for 0.5uS and high for 4.5uS, it needs the direction pin to be stable 1uS before the falling edge, and remain stable for 20uS after the falling edge. The longest timing requirement is the 20uS hold time. A simple approach would be to set the period at 20uS. That means that all changes on the STEP and DIR lines are separated by 20uS. All is good, right?

Wrong! If there was ZERO latency, then all edges would be separated by 20uS, and everything would be fine. But all computers have some latency. Latency means lateness. If the computer has 11uS of latency, that means sometimes the software runs as much as 11uS later than it was supposed to. If one run of the software is 11uS late, and the next one is on time, the delay from the first to the second is only 9uS. If the first one generated a step pulse, and the second one changed the direction bit, you just violated the 20uS G202 hold time requirement. That means your drive might have taken a step in the wrong direction, and your part will be the wrong size.

The really nasty part about this problem is that it can be very very rare. Worst case latencies might only happen a few times a minute, and the odds of bad latency happening just as the motor is changing direction are low. So you get very rare errors that ruin a part every once in a while and are impossible to troubleshoot.

The simplest way to avoid this problem is to choose a BASE_PERIOD that is the sum of the longest timing requirement of your drive, and the worst case latency of your computer. If you are running a Gecko with a 20uS hold time requirement, and your latency test said you have a maximum latency of 11uS, then if you set the BASE_PERIOD to 20+11 = 31uS (31000 nano-seconds in the ini file), you are guaranteed to meet the drive's timing requirements.

But there is a tradeoff. Making a step pulse requires at least two periods. One to start the pulse, and one to end it. Since the period is 31uS, it takes 2x31 = 62uS to create a step pulse. That means the maximum step rate is only 16,129 steps per second. Not so good. (But don't give up yet, we still have some tweaking to do in the next section.)

For the Xylotex, the setup and hold times are very short, 200nS each (0.2uS). The longest time is the 2uS high time. If you have 11uS latency, then you can set the BASE_PERIOD as low as 11+2=13uS. Getting rid of the long 20uS hold time really helps! With a period of 13uS, a complete step takes 2x13 = 26uS, and the maximum step rate is 38,461 steps per second!

But you can't start celebrating yet. Note that 13uS is a very short period. If you try to run the step generator every 13uS, there might not be enough time left to run anything else, and your computer will lock up. If you are aiming for periods of less than 25uS, you should start at 25uS or more, run EMC, and see how things respond. If all is well, you can gradually decrease the period. If the mouse pointer starts getting sluggish, and everything else on the PC slows down, your period is a little too short. Go back to the previous value that let the computer run smoothly.

In this case, sppose you started at 25uS, trying to get to 13uS, but you find that around 16uS is the limit - any less and the computer doesn't respond very well. So you use 16uS. With a 16uS period and 11uS latency, the shortest output time will be 16-11 = 5uS. The drive only needs 2uS, so you have some margin. Margin is good - you don't want to lose steps because you cut the timing too close.

What is the maximum step rate? Remember, two periods to make a step. You settled on 16uS for the period, so a step takes 32uS. That works out to a not bad 31,250 steps per second.

18.6.1.4 Use steplen, stepspace, dirsetup, and/or dirhold

In the last section, we got the Xylotex drive to a 16uS period and a 31,250 step per second maximum speed. But the Gecko was stuck at 31uS and a not-so-nice 16,129 steps per second. The Xylotex example is as good as we can make it. But the Gecko can be improved.

The problem with the G202 is the 20uS hold time requirement. That plus the 11uS latency is what forces us to use a slow 31uS period. But the EMC2 software step generator has some parameters that let you increase the various time from one period to several. For example, if steplen is changed from 1 to 2, then it there will be two periods between the beginning and end of the step pulse. Likewise, if dirhold is changed from 1 to 3, there will be at least three periods between the step pulse and a change of the direction pin.

If we can use dirhold to meet the 20uS hold time requirement, then the next longest time is the 4.5uS high time. Add the 11uS latency to the 4.5uS high time, and you get a minimum period of 15.5uS. When you try 15.5uS, you find that the computer is sluggish, so you settle on 16uS. If we leave dirhold at 1 (the default), then the minimum time between step and direction is the 16uS period minus the 11uS latency = 5uS, which is not enough. We need another 15uS. Since the period is 16uS, we need one more period. So we change dirhold from 1 to 2. Now the minimum time from the end of the step pulse to the changing direction pin is 5+16=21uS, and we don't have to worry about the Gecko stepping the wrong direction because of latency.

If the computer has a latency of 11uS, then a combination of a 16uS base period, and a dirhold value of 2 ensures that we will always meet the timing requirements of the Gecko. For normal stepping (no direction change), the increased dirhold value has no effect. It takes two periods totalling 32uS to make each step, and we have the same 31,250 step per second rate that we got with the Xylotex.

The 11uS latency number used in this example is very good. If you work through these examples with larger latency, like 20 or 25uS, the top step rate for both the Xylotex and the Gecko will be lower. But the same formulas apply for calculating the optimum BASE_PERIOD, and for tweaking dirhold or other step generator parameters.

18.6.1.5 No Guessing!

For a fast AND reliable software based stepper system, you cannot just guess at periods and other configuration paremeters. You need to make measurements on your computer, and do the math to ensure that your drives get the signals they need.

To make the math easier, I've created an Open Office spreadsheet (http://wiki.linuxcnc.org/uploads/StepTiming You enter your latency test result and your stepper drive timing requirements and the spreadsheet calculates the optimum BASE_PERIOD. Next, you test the period to make sure it won't slow down or lock up your PC. Finally, you enter the actual period, and the spreadsheet will tell you the stepgen parameter settings that are needed to meet your drive's timing requirements. It also calculates the maximum step rate that you will be able to generate.

I've added a few things to the spreadsheet to calculate max speed and stepper electrical calculations.

Part VIII

Machine logic

Chapter 19

Ladder programming

19.1 Introduction

Ladder logic or the Ladder programming language is a method of drawing electrical logic schematics. It is now a graphical language very popular for programming Programmable Logic Controllers (PLCs). It was originally invented to describe logic made from relays. The name is based on the observation that programs in this language resemble ladders, with two vertical "rails" and a series of "rungs" between them. In Germany and elsewhere in Europe, the style is to draw the rails horizontal along the top and bottom of the page while the rungs are drawn sequentially from left to right.

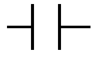
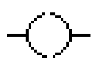
A program in ladder logic, also called a ladder diagram, is similar to a schematic for a set of relay circuits. Ladder logic is useful because a wide variety of engineers and technicians can understand and use it without much additional training because of the resemblance.

Ladder logic is widely used to program PLCs, where sequential control of a process or manufacturing operation is required. Ladder logic is useful for simple but critical control systems, or for reworking old hardwired relay circuits. As programmable logic controllers became more sophisticated it has also been used in very complex automation systems.

Ladder logic can be thought of as a rule-based language, rather than a procedural language. A "rung" in the ladder represents a rule. When implemented with relays and other electromechanical devices, the various rules "execute" simultaneously and immediately. When implemented in a programmable logic controller, the rules are typically executed sequentially by software, in a loop. By executing the loop fast enough, typically many times per second, the effect of simultaneous and immediate execution is obtained.

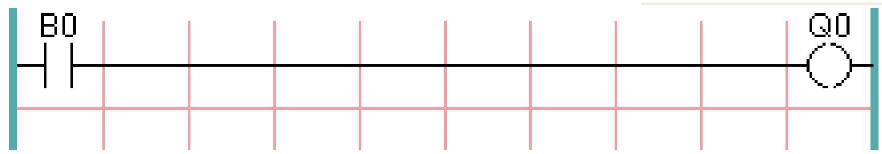
19.2 Example

The most common components of ladder are contacts (inputs), these usually are either NC (normally closed) or NO (normally open), and coils (outputs).

- the NO contact 
- the NC contact 
- the coil (output) 

Of course there are way more components to a full ladder language, but understanding these will help grasp the overall concept.

The ladder consists of one or more rungs. These rungs are horizontal traces, with components on them (inputs, outputs and other), which get evaluated left to right.

This example is the simplest rung: 

The input on the left, a normal open contact is connected to the output on the right Q0. Now imagine a voltage gets applied to the leftmost end, as soon as the B0 turns true (e.g. the input is activated, or the user pushed the NO contact), the voltage reaches the right part Q0. As a consequence, the Q0 output will turn from 0 to 1.

Chapter 20

Ladder in emc2 - ClassicLadder

20.1 Introduction

ClassicLadder is a free implementation of a ladder interpreter, released under the LGPL. It has been written by Marc Le Douarain.

He describes the beginning of the project on his website:

> "I decided to program a ladder language only for test purposes at the start, in february 2001. It was planned, that I would have to participate to a new product after leaving the enterprise in which I was working at that time. And I was thinking that to have a ladder language in thoses products could be a nice option to considerate. And so I started to code the first lines for calculating a rung with minimal elements and displaying dynamically it under Gtk, to see if my first idea to realise all this works.
> And as quickly I've found that it advanced quite well, I've continued with more complex elements : timer, multiples rungs, etc...
> Voila, here is this work... and more : I've continued to add features since then."

ClassicLadder has been adapted to work with emc2's HAL, and is currently beeing distributed along with emc2. If there are issues/problems/bugs please report them to the Enhanced Machine Controller project.

20.2 Languages

The most common language used when working with ClassicLadder is 'ladder'. ClassicLadder allows one to use other variants (like sequential function chart - Grafcet) too, however those aren't covered by the current documentation.

In the next chapters the main components of ClassicLadder will be described.

20.3 Starting ClassicLadder

There are 2 components belonging to ClassicLadder: a realtime part, and a userspace part (along with a GUI).

20.3.1 Loading the ClassicLadder realtime module

Loading the ClassicLadder realtime module (classicladder_rt) is possible from a halfile, or directly using a halcmd instruction.

It is possible to configure the number of each ladder object while loading the classicladder realtime module

Table 20.1: ClassicLadder realtime component options

Object name:	variable name:	Default value:
Number of rungs	(numRungs)	100
Number of bits	(numBits)	500
Number of word variables	(numWords)	100
Number of timers	(numTimers)	10
Number of monostables	(numMonostables)	10
Number of counters	(numCounters)	10
Number of hal inputs bit pins	(numPhysInputs)	50
Number of hal output bit pins	(numPhysOuputs)	50
Number of arithmetic expressions	(numArithmExpr)	50
Number of sections	(numSections)	100
Number of symbols	(numSymbols)	10

If you do not configure the number of ladder objects classicladder will use the default values. Objects of most interest are numPhysInputs and numPhysOutputs.

Changing these numbers will change the number of HAL bit pins available.

For example:

```
loadrt classicladder_rt numRungs=12 numBits=100 numWords=10 numTimers=10
numMonostables=10 numCounters=10 numPhysInputs=10 numPhysOutputs=10
numArithmExpr=100 numSections=4 numSymbols=200
```

20.3.2 Loading the ClassicLadder user module

There are options while loading the user module:

- –help ▷ displays basically this list then exits
- –version ▷ displays the... version ...surprise!! then exits
- –nogui ▷ starts classicladder (while loading a ladder program if specified) with no GUI.
- –modbus_port=port ▷ sets up the modbus port number (EMC doesn't use it)
- –config=file ▷ sets up the number of the each ladder object (only if there is no realtime support.In EMC you load this with the realtime module-I'll get to this in a minute)

Unfortunately as of EMC 2.1.x, –version and –help do not exit HAL properly. Really only –nogui is any use to you. Use the GUI when setting up your system then change it to –nogui when running. The only other thing you can do while loading the user module is specify a ladder program to load. ladder programs are specified by the .clp ending.

for examples:

- loadusr -w classicladder –nogui myladder.clp

  loads the classicladder user module, ladder program myladder and displays nothing.

- loadusr -w classicladder myladder.clp

  loads the classicladder user module, ladder program myladder and starts the classicladder GUI.

- loadusr -w classicladder

  loads the classicladder user module, starts the GUI but loads no laddder program.

*** you can only use this after loading a ladder program previously ***

20.4 ClassicLadder GUI

If you load classicladder with the GUI it will display three windows: vars, section display, and section manager.

20.4.1 The Variables window

It displays some of the variable data and variable names. Notice all variable start with the % sign.

The three edit areas at the top allow you to select what 15 variable will be displayed in each column. For instance if there were 30 %I variable and you entered 10 at the top of the column, variable %I10 to %I25 would be displayed.

The check boxes allow you to set and un set variables but when classicladder is running hal will update the pins and change them.

Near the bottom are the %W variables. These are called word variable and represent positive and negative (signed) numbers and are used with compare and operate. By clicking on the variable, you can edit the number to display which ever you want.The edit box beside it is the number stored in the variable -you can change it- and the drop-down box beside that allow you to choose whether the number to be displayed is in hex, decimal or binary.

The %I variable represents HAL input bit pins. The %Q represents the relay coil and HAL output bit pins. The %B represents an internal relay coil or internal contact.

There is a quirk that the word variables list will not display the symbols unless the check-box in the section manager is checked and you miss-edit the coresponding variable. Meaning erase the numbers after the %W and press return-if there is a symbol for that number you erased and the display symbols check-box in the section display is checked it will show it.

20.4.2 The Section Display window

Most of the buttons are self explanitory:

The config button is not used in EMC.

The symbols button will display an editable list of symbols for the variables (eg you can name the inputs, outputs, coils etc).

The quit button will only shut down the display-the ladder program will still run in the back ground.

The check box at the top right allows you to select whether variable names or symbol names are displayed

Figure 20.1: ClassicLadder Var window

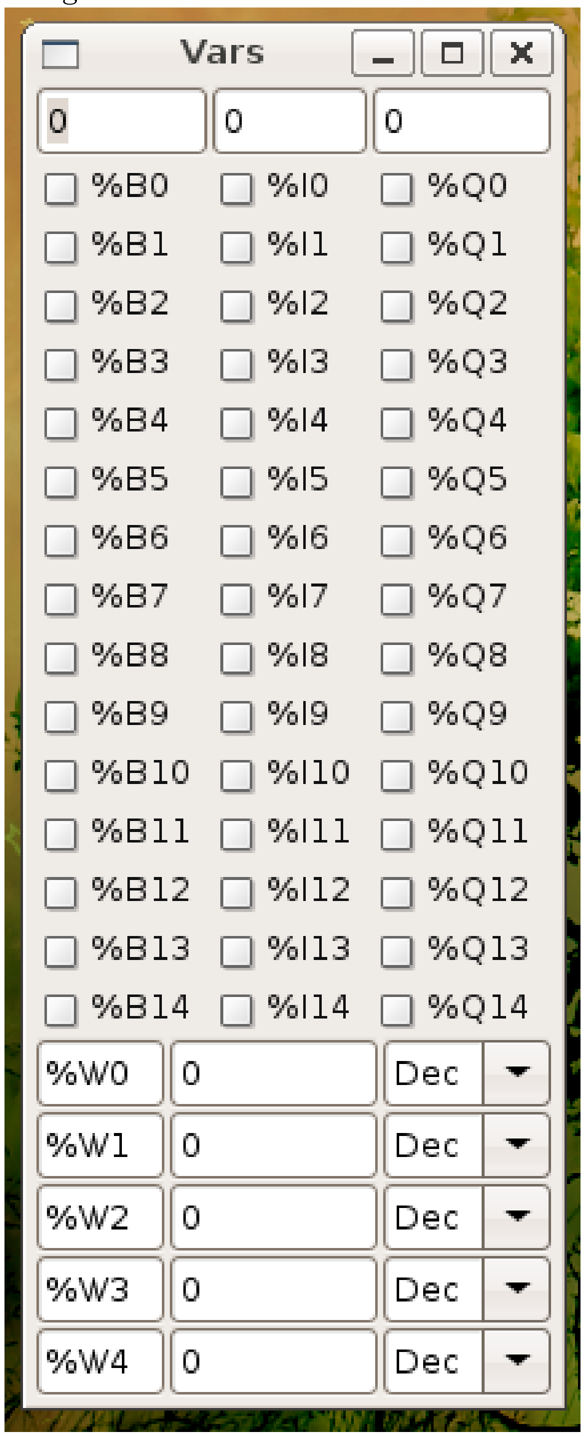
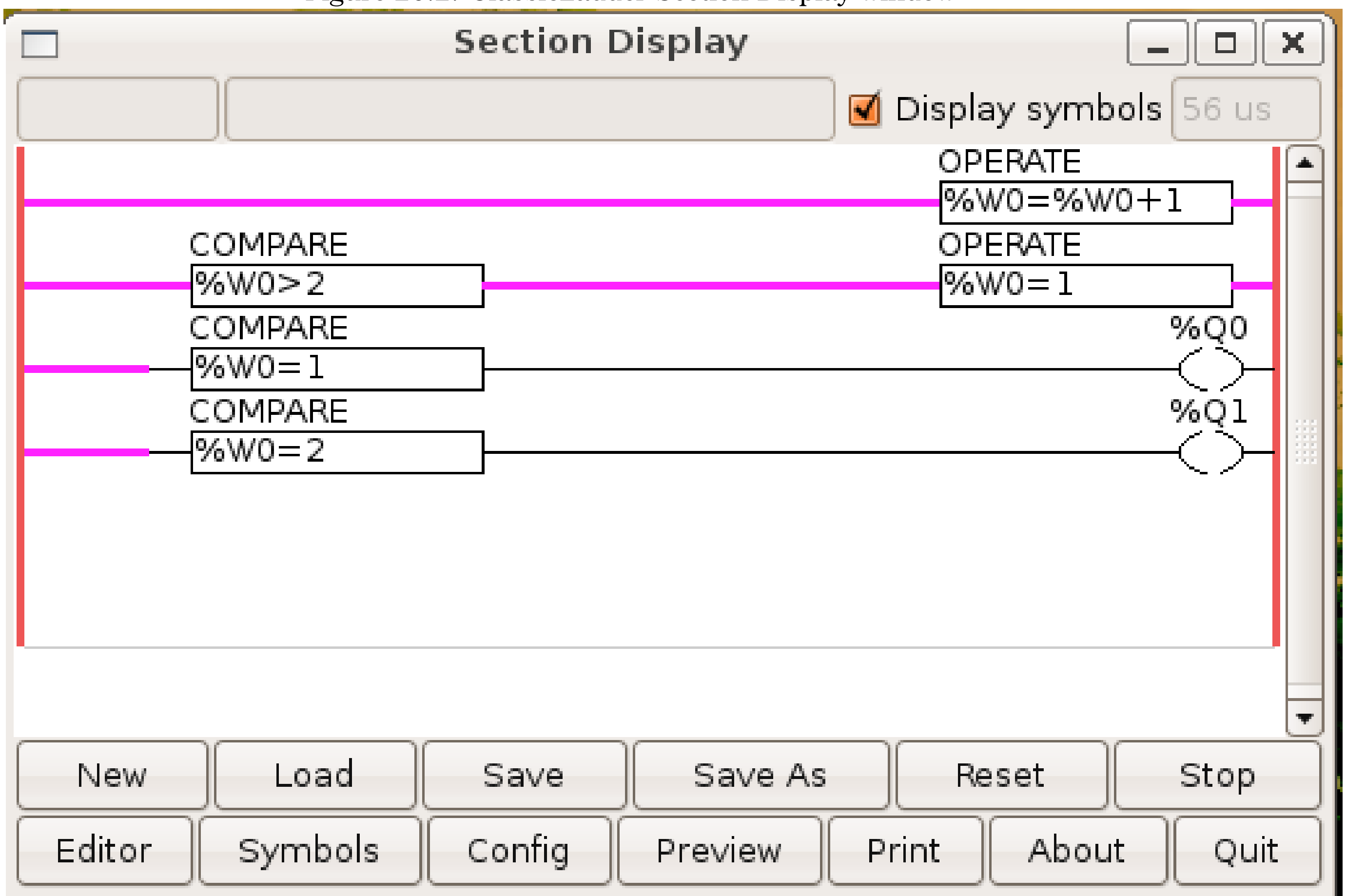


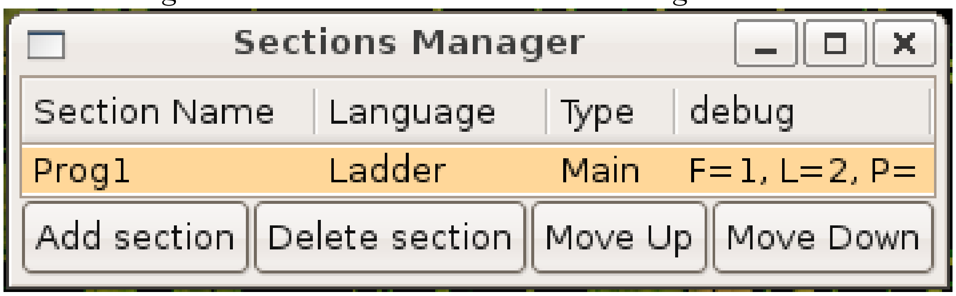
Figure 20.2: ClassicLadder Section Display window



20.4.3 The Section Manager window

This window allows you to name, create or delete sections. This is also how you name a subroutine for call coils.

Figure 20.3: ClassicLadder Section Manager window



20.4.4 The Editor window

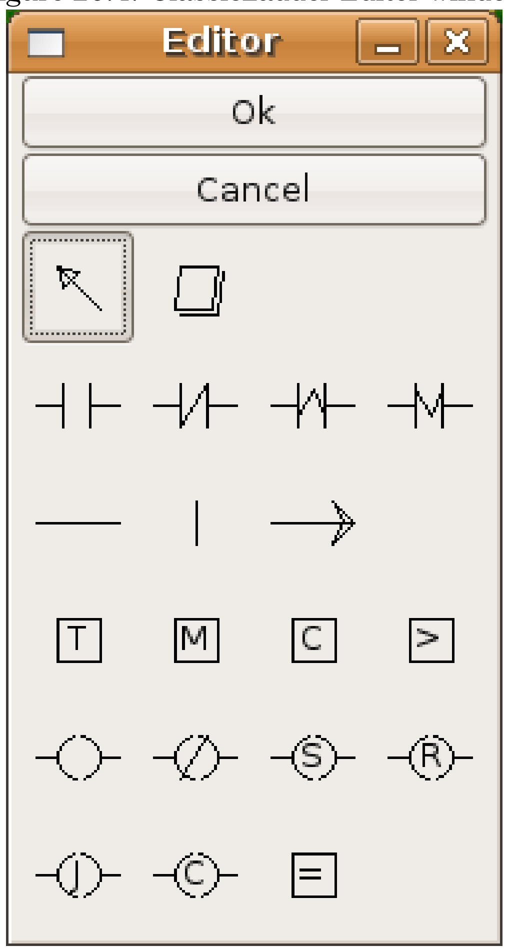
Starting from the top left image:

1. SELECTOR ARROW, ERASER
2. N.O., N.C. , RISING-EDGE ,FALLING-EDGE CONTACTS.
3. HORIZONTAL, VERTICAL , HORIZONTAL RUNNING-CONNECTIONS
4. TIMER, MONOSTABLE, COUNTER, COMPARE
5. N.O. COIL, N.C. COIL, SET COIL, RESET COIL
6. JUMP COIL, CALL COIL, OPERATE

A short description of each of the buttons:

- The SELECTOR ARROW button allows you to select existing objects and modify the information.
- The ERASER erases an object.
- The N.O. CONTACT is a normally open contact. It can be an extenal HAL-pin (%I) input contact, an internal-bit coil (%B) contact or a external coil (%Q) contact. The Hal-pin input contact is closed when the HAL-pin is true. The coil contacts are closed when the coresponding coil is active (%Q2 contact closes when %Q2 coil is active).
- The N.C. CONTACT is a normally closed contact. It is the same as the n.o. contact except that the contact is open when the hal-pin is true or the coil is active.
- The RISING-EDGE CONTACT is a contact that is closed when the HAL-pin goes from False to true, or the coil from not-active to active.
- The FALLING-EDGE CONTACT is a contact that is closed when the HAL-pin goes from true to false or the coil from active to not.
- The HORIZONTAL CONNECTION connects the 'signal' to objects horizontally.
- The VERTICAL CONNECTION connects the 'signal' to objects vertically.
- The HORIZONTAL-RUNNING CONNECTION is a quick way to connect a long run of 'signal wire' horizontally.

Figure 20.4: ClassicLadder Editor window



- The TIMER is a Timer Module.
- The MONOSTABLE is monostable module (one-shot)
- The COUNTER is a counter module.
- The COMPARE button allows you to compare variable to values or other variables. (eg %W1<=5 or %W1=%W2)
  The variable you can use are: W-words,T-timers,M-monostables,C-counters,X-sequential and their attributes D-done, E-empty, F-full, P-preset, R-running, and V-value (not all atributes are available to all variables) eg %T2.D.
  The math symbols are +,-,*,/,=,<,>,<=,>=,(,),^ (exponent),% (modulas),& (and),| (or),! (not).
  Math function are ABS (absolute), MOY (average). eg ABS(%W2)=1, MOY(%W1,%W2)<3 .
  Compare cannot be placed in the right most side of the section display.
- The OPERATE button allows you to assign values to variables. (eg %W2=7 or %W1=%W2) there are two math funtions MINI and MAXI that check a variable for maximum (0x80000000) and minimum values (0x05FFFFFFF) (think signed values) and keeps them from going beyond.
  You may use all the math symbols and functions from above. OPERATE funtions can only be placed at the right most side of the section display.

20.5 ClassicLadder Variables

List of known variables :

Bxxx : Bit memory xxx (boolean)

Wxxx : Word memory xxx (32 bits integer)

Txx,R : Timer xx running (boolean, user read only)

Txx,D : Timer xx done (boolean, user read only)

Txx,V : Timer xx current value (integer, user read only)

Txx,P : Timer xx preset (integer)

Mxx,R : Monostable xx running (boolean)

Mxx,V : Monostable xx current value (integer, user read only)

Mxx,P : Monostable xx preset (integer)

Cxx,D : Counter xx done (boolean, user read only)

Cxx,E : Counter xx empty overflow (boolean, user read only)

Cxx,F : Counter xx full overflow (boolean, user read only)

Cxx,V : Counter xx current value (integer, user read only)

Cxx,P : Counter xx preset (integer)

Ixxx : Physical input xxx (boolean) - HAL input bit -

Qxxx : Physical output xxx (boolean) - HAL output bit -

Xxxx : Activity of step xxx (sequential language)

Xxxx,V : Time of activity in seconds of step xxx (sequential language)

20.6 Using JUMP COILs

JUMP COILs are used to 'JUMP' to another section-like a goto in BASIC programming language.

If you look at the top left of the sections display window you will see a small lable box and a longer comment box beside it. Now go to Editor->Modify then go back to the little box, type in a name.

Go ahead and add a comment in the comment section. This lable name is the name of this rung only and is used by the JUMP COIL to identify where to go.

When placing a JUMP COIL add it in the right most position and change the lable to the rung you want to JUMP to.

JUMP COILs should be placed as the last coil of a rung because of a bug. If there are coils after the JUMP COIL (in the same rung) they will be updated even if the JUMP COIL is true.[1]

20.7 Using CALL COILs

CALL COILs are used to go to a subroutine section then return-like a gosub in BASIC programming language.

If you go to the sections manager window hit the add section button. You can name this section, select what language it will use (ladder or sequential), and select what type (main or subroutine).

Select a subroutine number (SR0 for exampe). An empty section will be displayed and you can build your subroutine.

When your done that, go back to the section manager and click on the your 'main' section (default name prog1).

Now you can add a CALL COIL to your program. CALL COILs are to be placed at the right most position in the rung.

Remember to change the lable to the subroutine number you choose before.

There can only be one CALL COIL per rung-the rest wil not be called.

[1] If the JUMP COIL is true it should JUMP to the new rung right away and not update the rest of the coils of the current rung

Appendix A

Glossary of Common Terms Used in the EMC Documents

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 re-circulate the balls from one end of the screw to the other.

CNC Computer Numerical Control. The general term used to refer to computer control of machinery. Instead of a human operator turning cranks to move a cutting tool, CNC uses a computer and motors to move the tool, based on a part program 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.

Display units The linear and angular units used for onscreen display.

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 cam 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 EMCA that handles general purpose I/O, unrelated to the actual motion of the axes.

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 A device to measure position. Usually a machanical-optical device, which outputs a quadrature signal. The signal can be counted by special hardware, or directly by the parport with emc2.

Feed Relatively slow, controlled motion of the tool used when making a cut.

Feed rate The speed at which a motion occurs. In manual mode, jog speed can be set from the graphical interface. In auto or mdi mode feed rate is commanded using a (f) word. F10 would mean ten units per minute.

Feedback A method (e.g., quadrature encoder signals) by which emc receives information about the position of motors

Feed rate 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. See also KinematicsA

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 See real-time A.

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.

Machine units The linear and angular units used for machine configuration. These units are used in the inifile. HAL pins and parameters are also generally in machine units.

MDI Manual Data Input. This is a mode of operation where the controller executes single lines of G-code A as they are typed by the operator.

NIST National Institute of Standards and Technology. An agency of the Department of Commerce in the United States.

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.

Program Units The linear and angular units used for part programs.

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/ `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.

RTAPI A portable interface to real-time operating systems including RTAIA and RTLINUXA

RS-274/NGC The formal name for the language used by EMC A part programs A. See Chapter **??**

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.

Units See "Machine Units", "Display Units", or "Program Units", above.

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 B

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

B.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.

B.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. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five). C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. H. Include an unaltered copy of this License. I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version. N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http:///www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index