Part I

Integrators Manual

V2.2 Integrators Manual

February 10, 2009

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.

Part II

Contents

Contents

Part III

Introduction

Chapter 1

The Enhanced Machine Control

1.1 Introduction

For normal stepper based installations see the Getting Started Guide. Once EMC is installed and configured see the User Manual for information on using EMC.

The Integrator Manual scope is on more complex machines, configurations and installations. As the system integrator your task is bringing together all the component subsystems into a whole and ensuring that those subsystems function together. EMC being one of the subsystems.

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 hand wheels. 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.

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

[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.

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 2.2

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

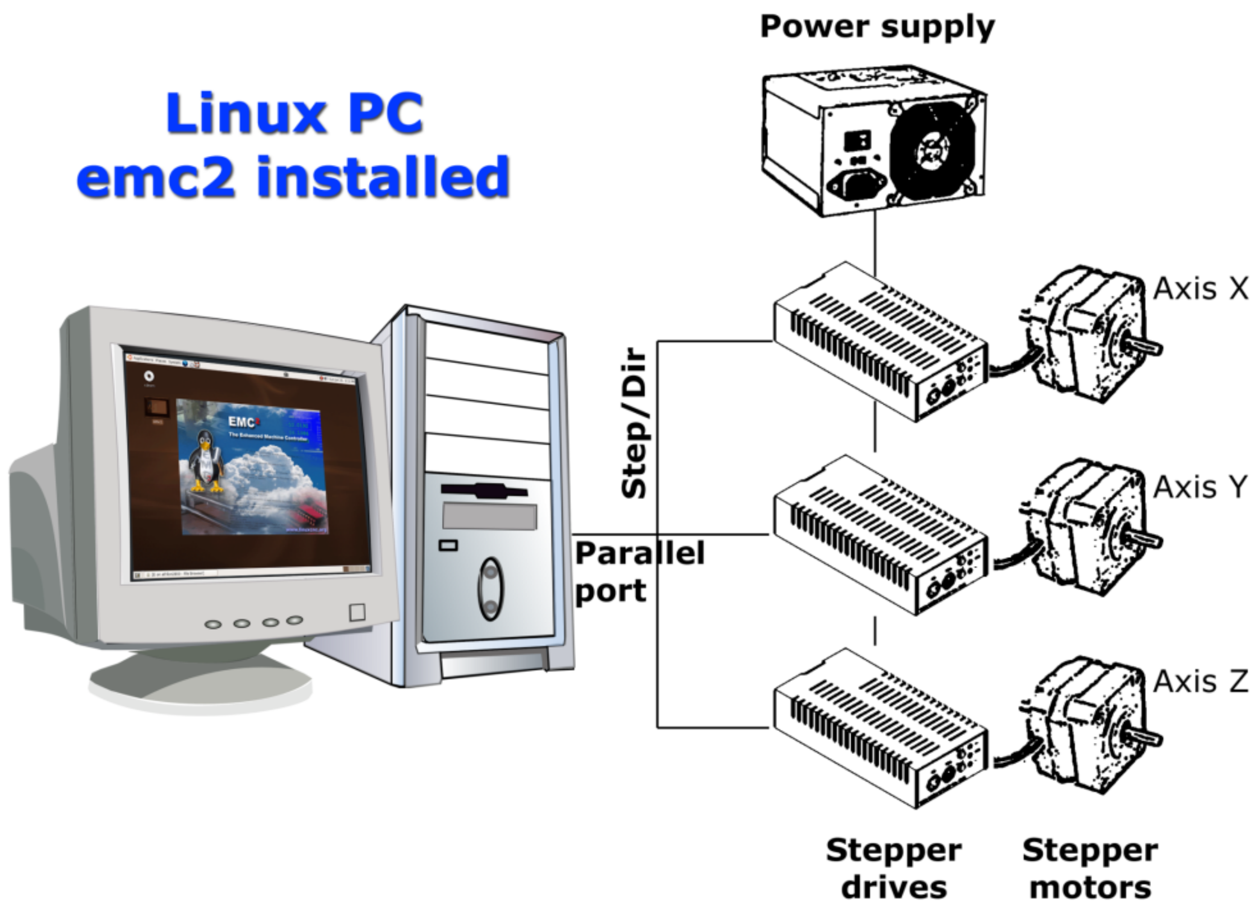
The Enhanced Machine Controller (EMC2) is a lot more than just another CNC mill program. It can control machine tools, robots, or other automated devices. It can control servo motors, stepper motors, relays, and other devices related to machine tools. In this handbook we focus on only a small part of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

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.

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 mini mill 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.

[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



1.5.1 Graphical User Interfaces

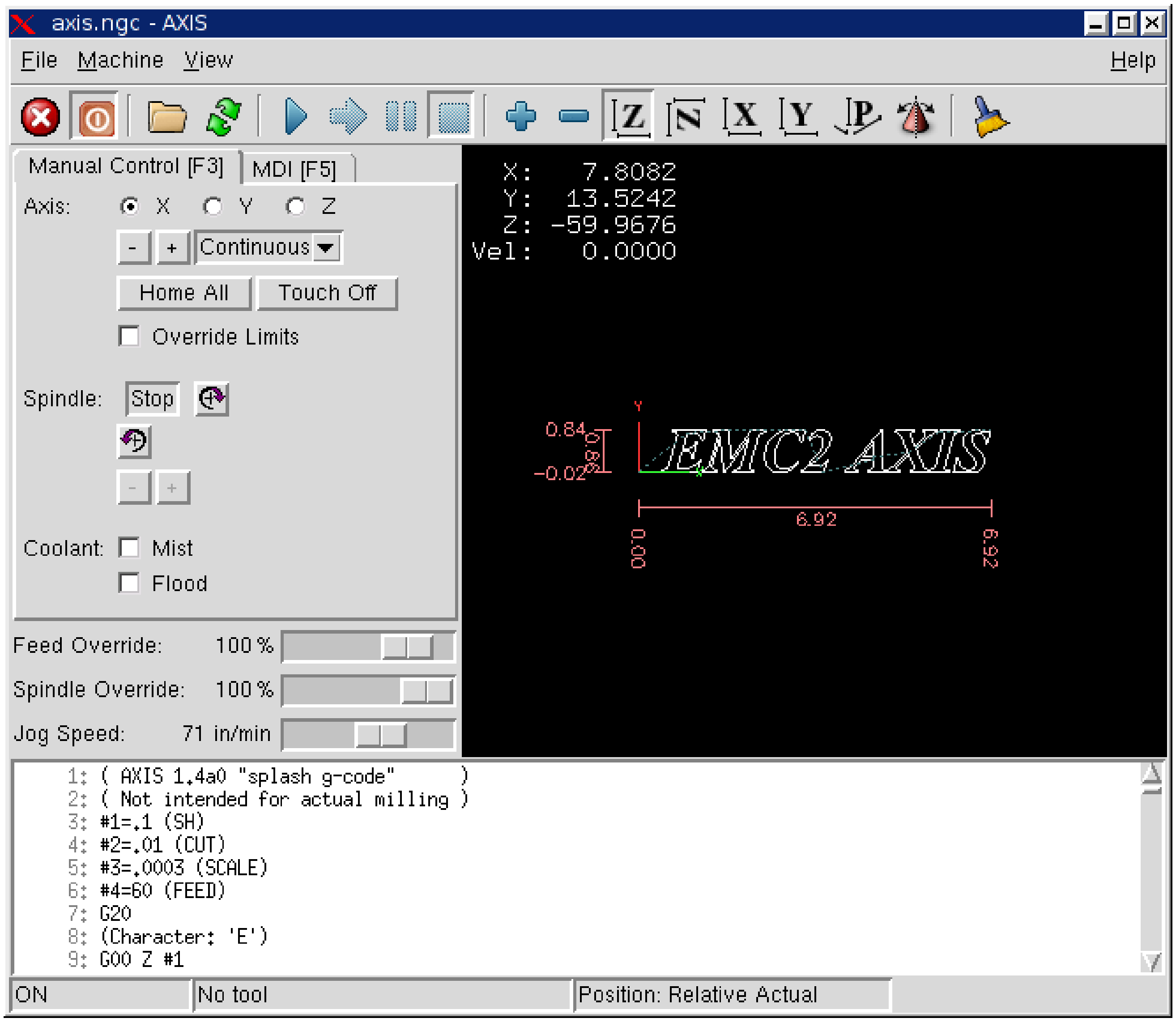
A graphical interface is the part of the EMC2 that the machine tool operator interacts with. The EMC2 comes with several types of user interfaces:

- 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 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



Axis: [✓ X] Y Z

Continuous

Spindle:

Coolant:

Feed Override: 100 %

Spindle Override: 100 %

Jog Speed: 71 in/min

```
X:    7.8082
Y:   13.5242
Z:  -59.9676
Vel:   0.0000
```

```
( AXIS 1.4a0 "splash g-code"      )
( Not intended for actual milling )
#1=.1 (SH)
#2=.01 (CUT)
#3=.0003 (SCALE)
#4=60 (FEED)
G20
(Character: 'E')
G00 Z #1
```

ON | No tool | Position: Relative Actual

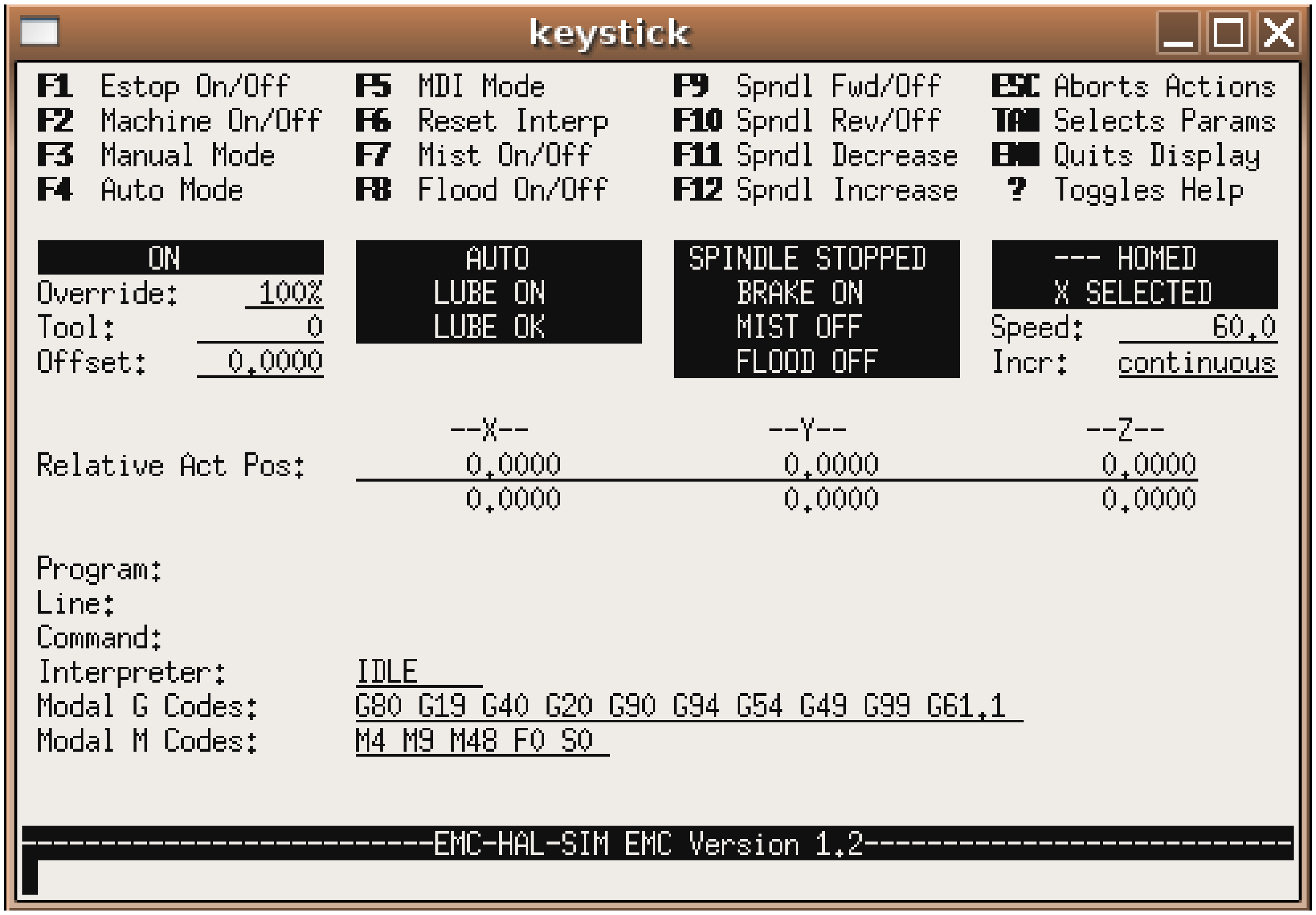
Figure 1.2: The AXIS Graphical Interface

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. 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



```
keystick

F1  Estop On/Off     F5  MDI Mode       F9  Spndl Fwd/Off    ESC Aborts Actions
F2  Machine On/Off   F6  Reset Interp   F10 Spndl Rev/Off    TAB Selects Params
F3  Manual Mode      F7  Mist On/Off    F11 Spndl Decrease   END Quits Display
F4  Auto Mode        F8  Flood On/Off   F12 Spndl Increase    ?  Toggles Help

         ON                AUTO         SPINDLE STOPPED         --- HOMED
Override:      100%       LUBE ON          BRAKE ON            X SELECTED
Tool:             0       LUBE OK          MIST OFF        Speed:        60.0
Offset:      0.0000                        FLOOD OFF       Incr:   continuous

                           --X--              --Y--              --Z--
Relative Act Pos:          0.0000             0.0000             0.0000
                           0.0000             0.0000             0.0000

Program:
Line:
Command:
Interpreter:          IDLE
Modal G Codes:        G80 G19 G40 G20 G90 G94 G54 G49 G99 G61.1
Modal M Codes:        M4 M9 M48 F0 S0

-------------------------EMC-HAL-SIM EMC Version 1.2-------------------------
```

Figure 1.3: The Keystick interface

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.).

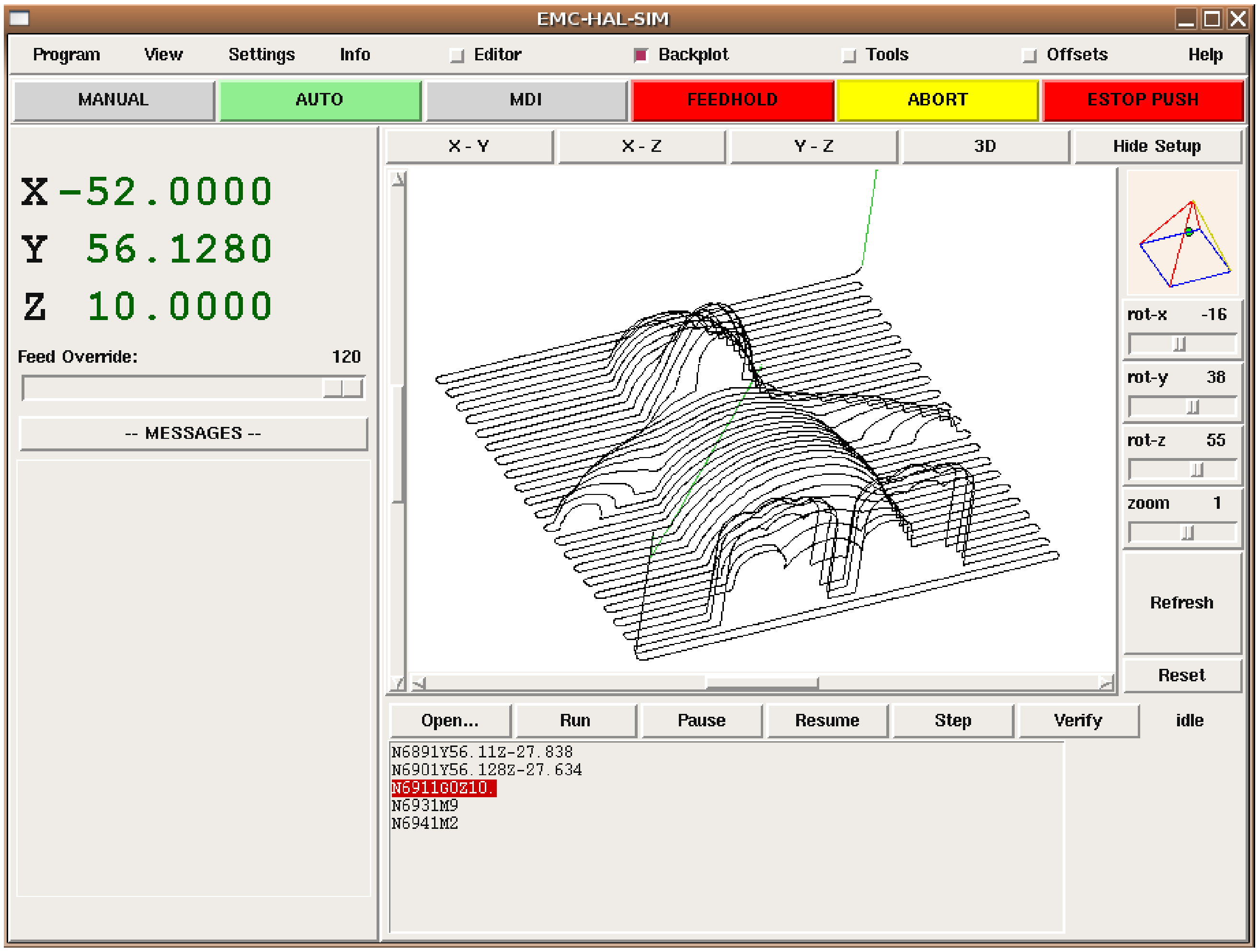
In EMC2 there is only one big I/O controller, which provides support for all kinds of actions and hardware control. All its outputs and inputs are HAL pins (more on this later on), so you can use only the subset that fits your hardware and is necessary for your application.

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 hand wheel 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 hand wheel five turns clockwise in 20 seconds. The same machinist programming that same move for CNC might write the following block of code.

```
G1 F3 X1.000
```

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



```
X -52.0000
Y  56.1280
Z  10.0000
```

Feed Override: 120

rot-x -16
rot-y 38
rot-z 55
zoom 1

```
N6891Y56.11Z-27.838
N6901Y56.128Z-27.634
N6911G0Z10.
N6931M9
N6941M2
```

Figure 1.4: The Mini Graphical Interface

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 .

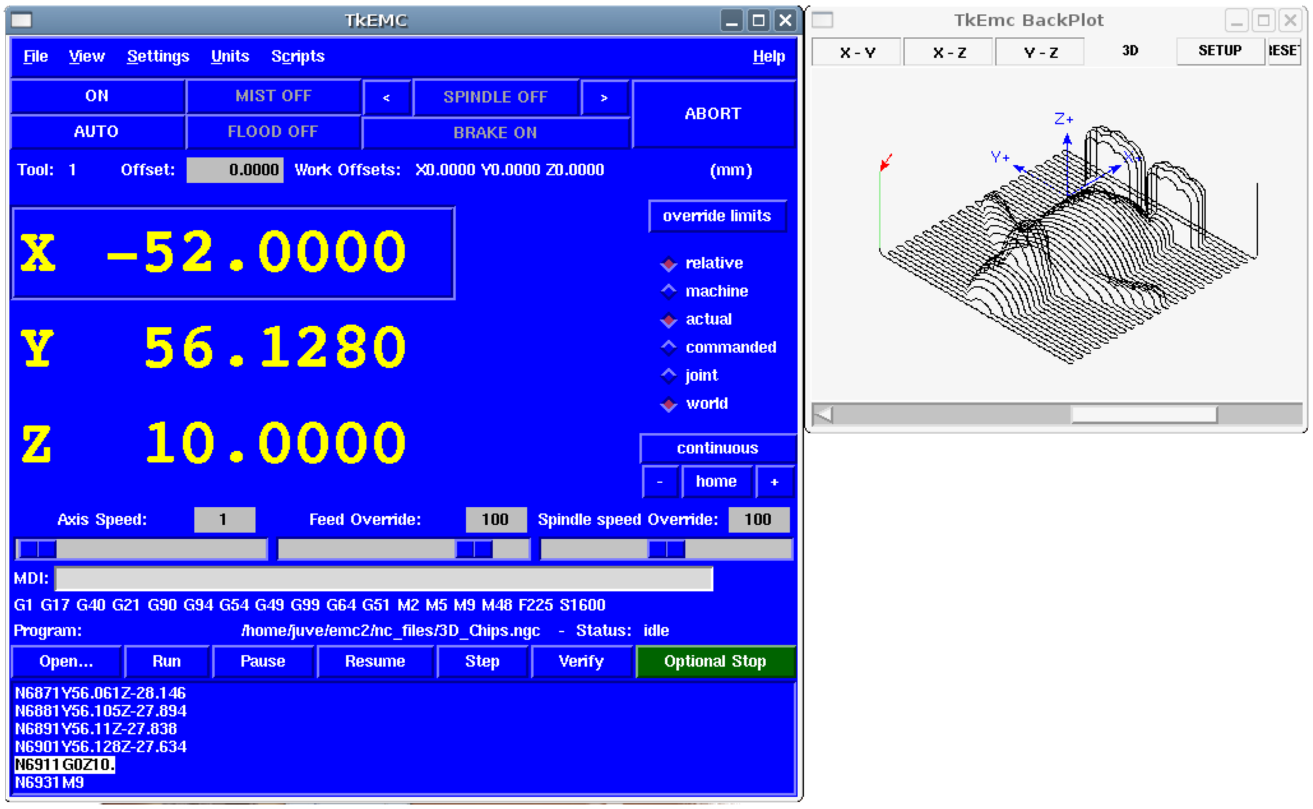
Figure 1.7 is a block diagram of how a personal computer running the EMC2 is used to control a machine with G-code. The actual G-code can be sent using the MDI (Machine Device Interface) mode or it can be sent as a file when the machine is in Auto mode. These choices are made by the operator and entered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

1.5.5 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



```
Tool: 1    Offset: 0.0000  Work Offsets: X0.0000 Y0.0000 Z0.0000    (mm)
X  -52.0000
Y   56.1280
Z   10.0000
Axis Speed: 1    Feed Override: 100  Spindle speed Override: 100
MDI:
G1 G17 G40 G21 G90 G94 G54 G49 G99 G64 G51 M2 M5 M9 M48 F225 S1600
Program:  /home/juve/emc2/nc_files/3D_Chips.ngc  -  Status: idle
N6871Y56.061Z-28.146
N6881Y56.105Z-27.894
N6891Y56.11Z-27.838
N6901Y56.128Z-27.634
N6911G0Z10.
N6931M9
```

Figure 1.5: The TkEmc Graphical Interface

switch or turning the hand wheel for an axis. These commands are normally handled on one of the graphical interfaces by pressing a button with the mouse or holding down a key on the keyboard. In auto mode, a similar button or key press might be used to load or start the running of a whole program of G-code that is stored in a file. In the MDI mode the operator might type in a block of code and tell the machine to execute it by pressing the <return> or <enter> key on the keyboard.

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

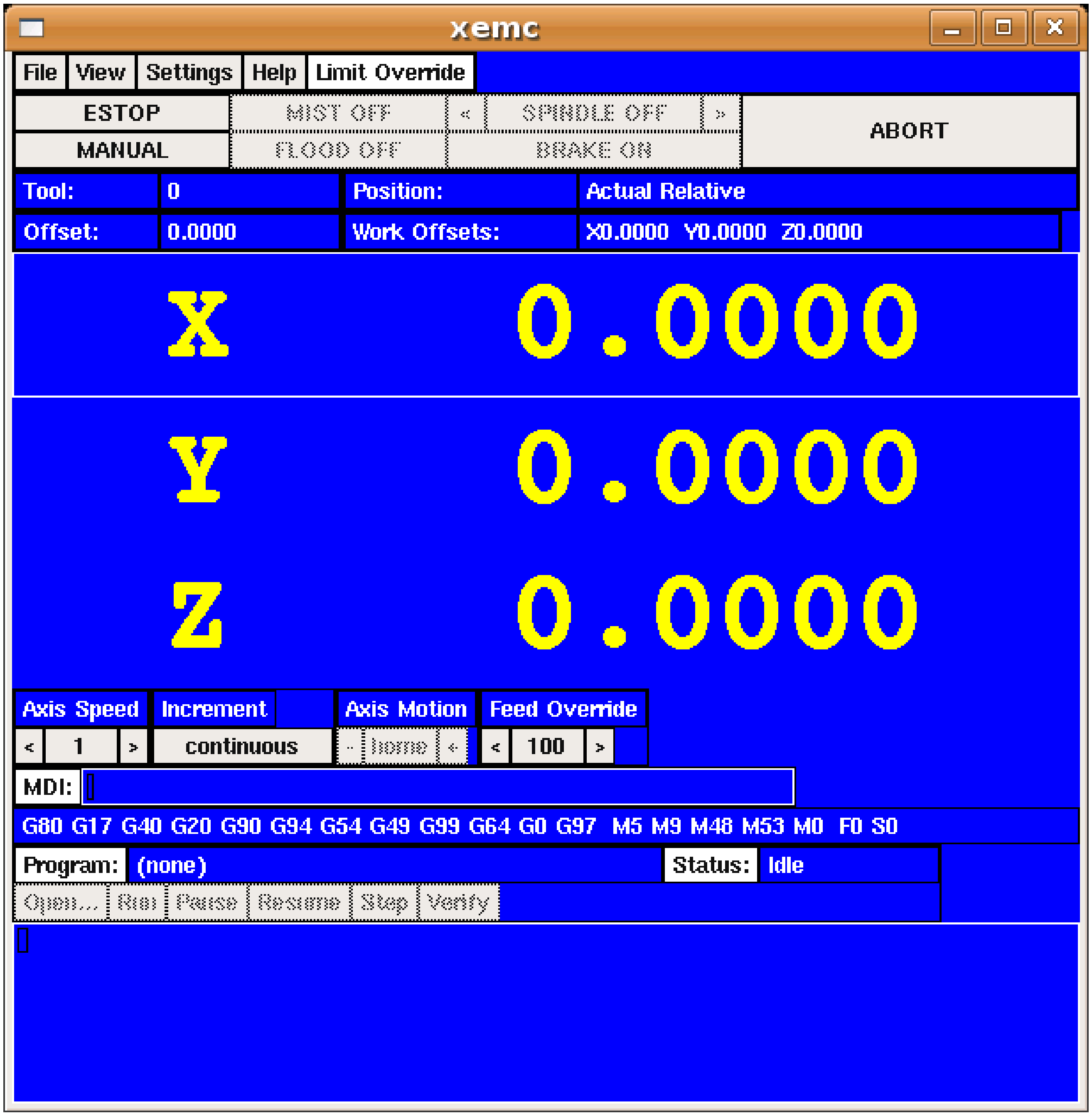
The AXIS user interface removes some of the distinctions between Auto and the other modes by making Auto-commands available at most times. It also blurs the distinction between Manual and MDI because some Manual commands like Touch Off are actually implemented by sending MDI commands.

1.5.6 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.



Tool: 0
Offset: 0.0000
Position: Actual Relative
Work Offsets: X0.0000 Y0.0000 Z0.0000

X 0.0000
Y 0.0000
Z 0.0000

Axis Speed 1, Increment continuous, Feed Override 100

G80 G17 G40 G20 G90 G94 G54 G49 G99 G64 G0 G97 M5 M9 M48 M53 M0 F0 S0

Program: (none) Status: Idle

Figure 1.6: The XEMC Graphical Interface

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

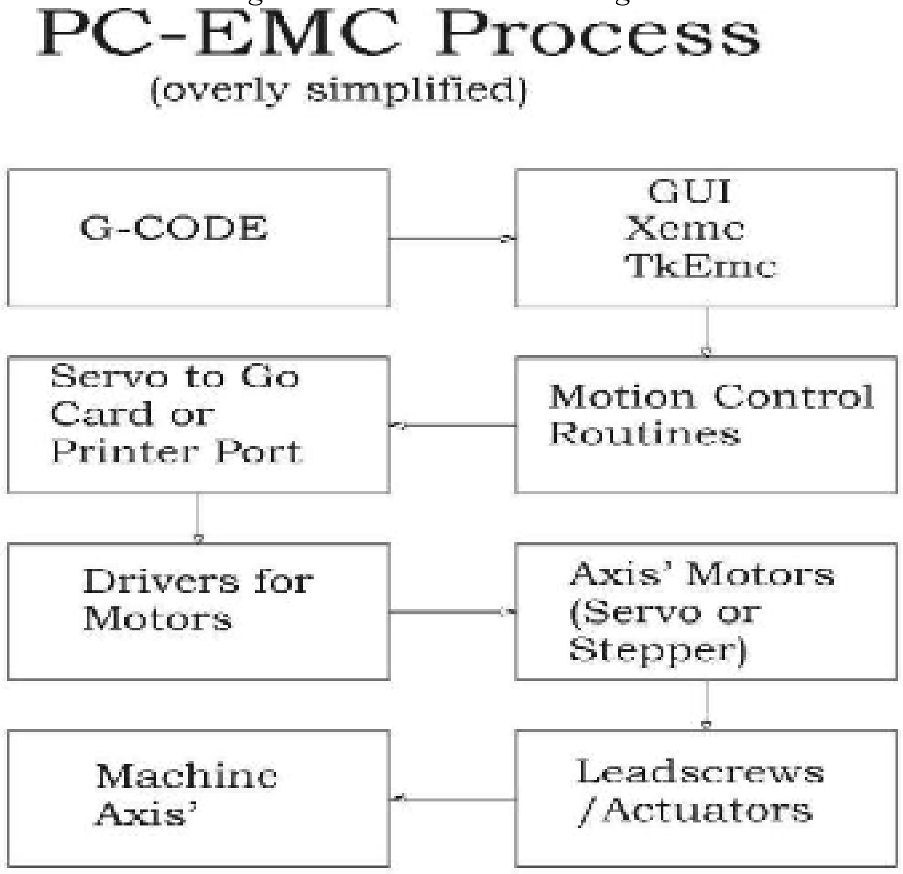
Actual This is the real position of the axis within the machine or relative system.

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

Figure 1.7: EMC2 Process Diagram



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 mini mill here, you will be thinking of other applications and other capabilities. These are the topics of the other linuxcnc.org handbooks.

1.6 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.

1.6.1 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 IV

Installing EMC

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 EMC2 Live CD

The EMC2 team now has a custom Live-CD based on Ubuntu 6.06 and 8.04 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 from www.linuxcnc.org and burn it to a CD.

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.

[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

2.3 Other Methods

You will find information about other install methods on the following web sites. These methods are only needed if you have special needs or you just have to have the bleeding edge version.

http://www.linuxcnc.org (Home of EMC2)

http://wiki.linuxcnc.org/cgi-bin/emcinfo.pl (User maintained Wiki EMC2 site)

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 Drake” 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[2].

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:

[2]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.

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

Chapter 3

Compiling from Source

3.1 Introduction

The third hurdle that you face when you begin to set up the EMC2 manually 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. Again the easiest install is the Live-CD.

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.

3.2 EMC Download Page

You will find the most recent releases of EMC2 announced on the Download page at:`http://www.linuxcnc.org/index.php`

The releases of EMC2 will be done in three ways the Live-CD, sources and binary package. The sources (described further on) 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 consensus 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.

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.

The pre-built packages for Ubuntu Linux use the "Installed" method

3.4.1 Downloading the CVS version

If you wish to use the TRUNK version of EMC2 or the latest branch version of EMC2, please follow the instructions on our wiki site to obtain the source code:

`http://wiki.linuxcnc.org/cgi-bin/emcinfo.pl?CVS`

3.5 Installed

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

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

and issue these commands:

```
~/emc2/src$ ./configure
~/emc2/src$ make && sudo make install
```

To run it simply type 'emc'.

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:

```
~/emc2/src$ ./configure --enable-run-in-place
~/emc2/src$ make && sudo make setuid
```

In a shell session where you want to use the run-in-place version of EMC, execute

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

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.

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:

```
~/emc2/src$ emc
```

3.7 Simulator

To install EMC2 on a system without a real time kernel, add `--enable-simulator` to the `configure` command line. In this mode, EMC2 runs as a purely userspace program. No hardware can be controlled and real time 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 not needed.

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

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

Part V

Configuration

Chapter 4

Hardware

4.1 Latency Test

Latency is how long it takes the PC to stop what it is doing and respond to an external request. For EMC2 the request is BASE_THREAD that makes the periodic "heartbeat" that serves as a timing reference for the step pulses. The lower the latency, the faster you can run the heartbeat, and the faster and smoother the step pulses will be.

Latency is far more important than CPU speed. A lowly Pentium II that responds to interrupts within 10 microseconds each and every time can give better results than the latest and fastest P4 Hyperthreading beast.

The CPU isn't the only factor in determining latency. Motherboards, video cards, USB ports, and a number of other things can hurt the latency. The best way to find out what you are dealing with is to run the RTAI latency test.

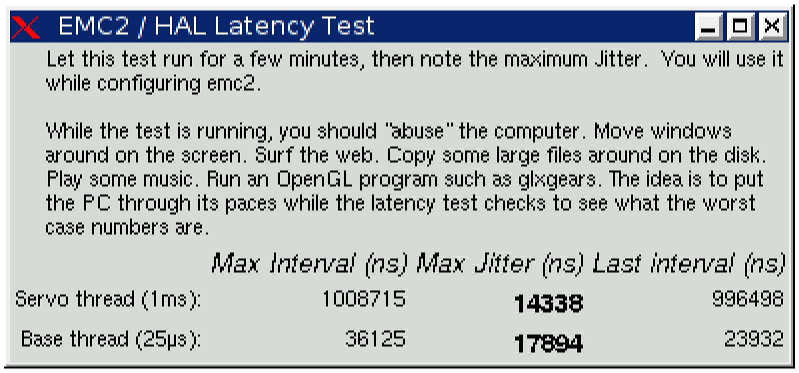
Generating step pulses in software has one very big advantage - it's free. Just about every PC has a parallel port that is capable of outputting step pulses that are generated by the software. However, software step pulses also have some disadvantages:

- limited maximum step rate
- jitter in the generated pulses
- loads the CPU

The best way to find out what you are dealing with is to run the HAL latency test. To run the test, open a terminal window from Applications/Accessories/Terminal (Ubuntu) and run the following command:

```
latency-test
```

You should see something like this:


EMC2 / HAL Latency Test

Let this test run for a few minutes, then note the maximum Jitter. You will use it while configuring emc2.

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.

	Max Interval (ns)	Max Jitter (ns)	Last interval (ns)
Servo thread (1ms):	1008715	**14338**	996498
Base thread (25µs):	36125	**17894**	23932

While the test is running, you should "abuse" the computer. Move windows around on the screen. Surf the web. Copy some large files around on the disk. Play some music. Run an OpenGL program such as glxgears. The idea is to put the PC through its paces while the latency test checks to see what the worst case numbers are.

NOTE: Do not run EMC2 or Stepconf while the latency test is running.

The important numbers are the "max jitter". In the example above, that is 17894 nanoseconds, or 17.9 microseconds. Record this number, and enter it in Stepconf when it is requested.

In the example above, latency-test only ran for a few seconds. You should run the test for at least several minutes; sometimes the worst case latency doesn't happen very often, or only happens when you do some particular action. For instance, one Intel motherboard worked pretty well most of the time, but every 64 seconds it had a very bad 300uS latency. Fortunately that was fixable see ([http://wiki.linuxcnc.org/cgi-bin/emcinfo.pl?FixingSMIIssues | | "Fixing SMI Issues"])

So, what do the results mean? If your Max Jitter number is less than about 15-20 microseconds (15000-20000 nanoseconds), the computer should give very nice results with software stepping. If the max latency is more like 30-50 microseconds, you can still get good results, but your maximum step rate might be a little disappointing, especially if you use microstepping or have very fine pitch leadscrews. If the numbers are 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. Another PC had very bad latency (several milliseconds) when using the onboard video. But a $5 used video card solved the problem - EMC does not require bleeding edge hardware.

4.2 Port Address

For those who build their own hardware, one safeguard against shorting out an on-board parallel port - or even the whole motherboard - is to use an add-on parallel port card. Even if you don't need the extra layer of safety, a parport card is a good way to add extra I/O lines with EMC.

One good PCI parport card is made with the Netmos 9815 chipset. It has good +5V signals, and can come in a single or dual ports.

To find the I/O addresses for these cards open a terminal window and use the list pci command:

```
lspci -v
```

Look for the entry with "NetMos" in it. Example of a 2-port card:

0000:01:0a.0 Communication controller: Netmos Technology PCI 9815 Multi-I/O Controller (rev 01)

```
Subsystem: LSI Losgic / Symbios Logic 2POS (2 port parallel adapter)
Flags: medium devsel, IRQ 5
I/O ports at b800 [size=8]
I/O ports at bc00 [size=8]
I/O ports at c000 [size=8]
I/O ports at c400 [size=8]
I/O ports at c800 [size=8]
I/O ports at cc00 [size=16]
```

From experimentation, I've found the first port (the on-card port) uses the third address listed (c000), and the second port (the one that attaches with a ribbon cable) uses the first address listed (b800).

You can then open an editor and put the addresses into the appropriate place in your .hal file.

```
loadrt hal_parport cfg="0x378 0xc000"
```

You must also direct emc to run the "read" and "write" functions for the second card. For example,

```
addf parport.1.read base-thread 1
addf parport.1.write base-thread -1
```

Please note that your values will differ. The Netmos cards are Plug-N-Play, and might change their settings depending on which slot you put them into, so if you like to 'get under the hood' and re-arrange things, be sure to check these values before you start EMC.

Chapter 5

Config Files

5.1 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 start up 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 is a way for the interpreter to save some values from one run to the next. These values are saved from one run to another but not always saved immediately. See the Parameters section of the G Code Manual for information on what each parameter is.

TBL The tbl file saves tool information. See Tool File section of the G Code Manual.

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.

[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/)

Chapter 6

INI File

6.1 File Layout

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

- comments.
- sections,
- variables.

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

6.1.1 Comments

A comment line is started with a ; or a # mark. When the ini reader sees either of these marks at the start a line, the rest of the line is ignored by the software. Comments can be used to describe what 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
```

6.1.2 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 (6.2.1)
- [DISPLAY] settings related to the graphical user interface (6.2.2)
- [FILTER] settings input filter programs (6.2.3)
- [RS274NGC] settings used by the g-code interpreter ()
- [EMCMOT] settings used by the real time motion controller (6.2.5)
- [HAL] specifies .hal files (6.2.7)
- [TASK] settings used by the task controller (6.2.6)
- [TRAJ] additional settings used by the real time motion controller (6.2.8)
- [AXIS_0] ... [AXIS_n] individual axis variables (6.2.9)
- [EMCIO] settings used by the I/O Controller (6.2.10)

6.1.3 Variables

A variable line is made up of a variable name, an equals sign(=), and a value. Everything from the first non-white space character after the = up to the end of the line is passed as the value, so you can embed spaces in string symbols if you want to or need to. A variable name is often called a keyword.

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.

6.1.4 Definitions

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

6.2 Section Variables

6.2.1 [EMC] Section

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

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

6.2.2 [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

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

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

INTRO_TIME = 5 The maximum time to show the splash screen

6.2.2.1 AXIS Interface

If your using the Axis interface the following can be used with it only.

DEFAULT_LINEAR_VELOCITY = .25 The default velocity for linear jogs, in machine units per second.

MAX_LINEAR_VELOCITY = 1.0 The maximum velocity for linear jogs, in machine units per second.

DEFAULT_ANGULAR_VELOCITY = .25 The default velocity for angular jogs, in machine units per second.

MAX_ANGULAR_VELOCITY = 1.0 The maximum velocity for angular jogs, in machine units per second.

INCREMENTS = 1 mm, .5 in, ... Defines the increments available for incremental jogs. The INCREMENTS can be used to override the default. The values can be decimal numbers (e.g., 0.1000) or fractional numbers (e.g., 1/16), optionally followed by a unit (cm, mm, um, inch, in or mil). If a unit is not specified the machine unit is assumed. Metric and imperial distances may be mixed: INCREMENTS = 1 inch, 1 mil, 1 cm, 1 mm, 1 um is a valid entry.

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

EDITOR = gedit The editor to use when selecting File > Edit or File Edit Tool Table from the AXIS menu. This must be configured for these menu items to work. Another valid entry is gnome-terminal -e vim.

6.2.3 [FILTER] Section

AXIS has the ability to send loaded files through a filter program. This filter can do any desired task: Something as simple as making sure the file ends with M2, or something as complicated as detecting whether the input is a depth image, and generating g-code to mill the shape it defines. The [FILTER] section of the ini file controls how filters work. First, for each type of file, write a PROGRAM_EXTENSION line. Then, specify the program to execute for each type of file. This program is given the name of the input file as its first argument, and must write rs274ngc code to standard output. This output is what will be displayed in the text area, previewed in the display area, and executed by emc when Run. The following lines add support for the image-to-gcode converter included with emc2:

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

It is also possible to specify an interpreter:

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

In this way, any Python script can be opened, and its output is treated as g-code. One such example script is available at nc_files/holecircle.py. This script creates g-code for drilling a series of holes along the circumference of a circle. Many other g-code generators are on the EMC Wiki site http://wiki.linuxcnc.org/cgi-bin/emcinfo.pl.

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

`FILTER_PROGRESS=%d`

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

6.2.4 [RS274NGC] Section

PARAMETER_FILE = file.var The file which contains the parameters used by the interpreter (saved between runs).

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, because the modal codes of machines differ, and may be changed by g-code interpreted earlier in the session.

6.2.5 [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 real time 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**.

6.2.6 [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.

6.2.7 [HAL] section

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

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

SHUTDOWN = shutdown.hal Execute the file 'shutdown.hal' when 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 e-stop 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 14 for more information.

6.2.8 [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.[1] If unset, joint positions are not stored and will begin at 0 each time EMC is started.

6.2.9 [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)

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

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.

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 EMC2 Servo Axis Calibration utility program (in the AXIS interface menu Machine/Calibration and in TkEMC it is under Setting/Calibration) can be used to set these and more interactively and verify their results so that the proper values can be put in the INI file with a minimum of trouble.

6.2.9.1 Homing-related items

The next few parameters are Homing related, for a better explanation read Section 6.3

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 second. 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.

6.2.9.2 Servo-related items

The following items are for servo-based systems and servo-like systems. 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.

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 0th order feed forward gain. This number is multiplied by the commanded position, resulting in a contribution to the computed voltage for the motor amplifier. The units on the FF0 gain are volts per machine unit, e.g., $\frac{volt}{mm}$ if machine units are millimeters.

FF1 = 0 (HAL) The 1st order feed forward gain. This number is multiplied by the change in commanded position per second, resulting in a contribution to the computed voltage for the motor amplifier. The units on the FF1 gain are volts per machine unit per second, e.g., $\frac{volt}{mm\,s}$ if machine units are millimeters.

FF2 = 0 (HAL) The 2nd order feed forward gain. This number is multiplied by the change in commanded position per second per second, resulting in a contribution to the computed voltage for the motor amplifier. The units on the FF2 gain are volts per machine unit per second per second, e.g., $\frac{volt}{mm\,s^2}$ 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 DAC as well, resulting in values that reflect the combined effects of amplifier gain, DAC non-linearity, DAC units, etc. To do this, follow this procedure:

1. Build a calibration table for the output, driving the DAC with a desired voltage and measuring the result. See table 6.2.9.2 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. The value is applied symmetrically to both the plus and the minus side.

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

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

6.2.9.3 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.

6.2.10 [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. For more information see the G Code Manual.

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

6.3 Homing

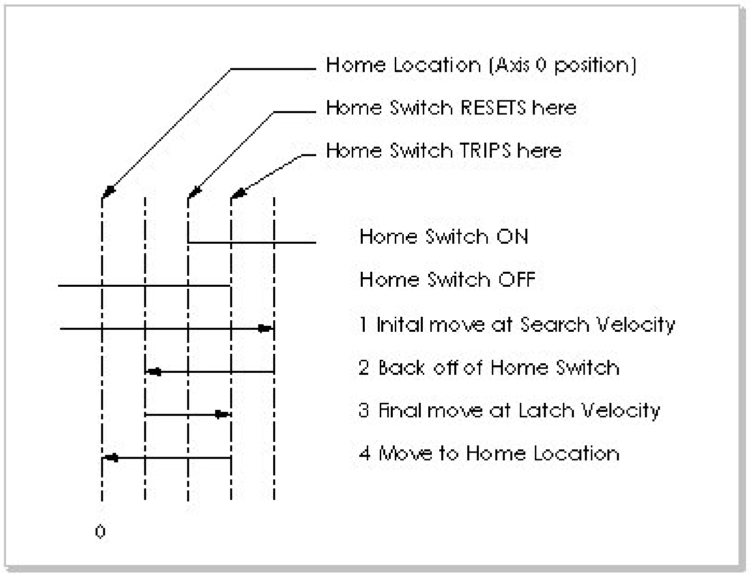
6.3.1 Overview

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 can be quite complicated. When specifying the direction of travel for the axis by using a positive or negative number for SEARCH_VEL and LATCH_VEL the machine will move in the same direction as when you do a + or - jog in manual control. For example if you press the + key to move toward your home switch and you have a typical configuration then both SEARCH_VEL and LATCH_VEL will be positive numbers for that axis.

6.3.2 Homing Sequence

In figure 6.1 you can see the sequence of a typical homing scenario when both SEARCH_VEL and LATCH_VEL are the same sign (positive or negative number). Notice that typical switches will have two different points where the switch will change states the "tripped" point and the "reset" point. Normally you have to travel past the reset point to get to the trip point and you have to travel past the tripped point to reach the reset point.

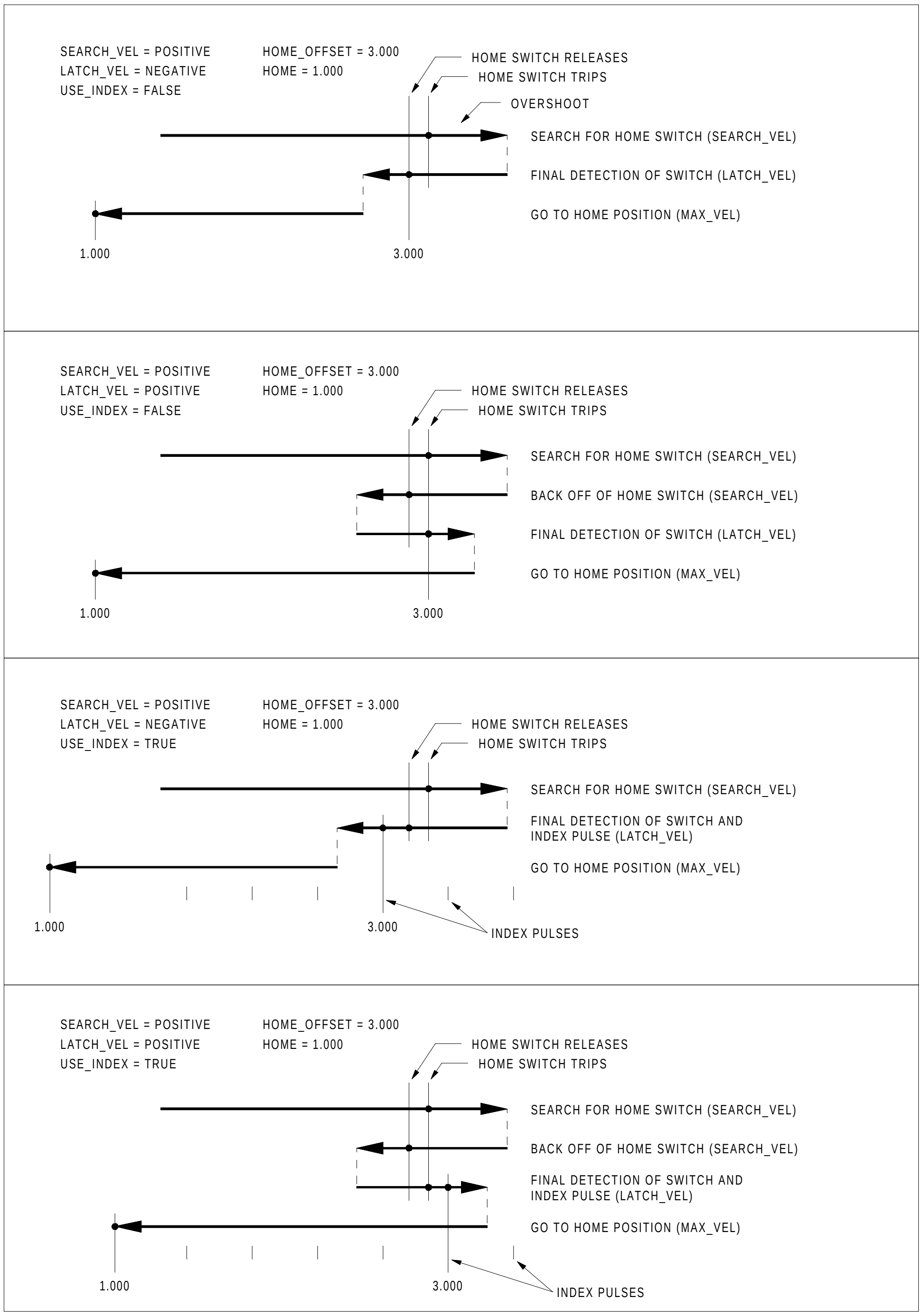
Figure 6.1: Typical Homing Sequence


Home Location (Axis 0 position)
Home Switch RESETS here
Home Switch TRIPS here
Home Switch ON
Home Switch OFF
1 Inital move at Search Velocity
2 Back off of Home Switch
3 Final move at Latch Velocity
4 Move to Home Location
0

1. The axis moves toward the home switch at search velocity as defined by SEARCH_VEL in the [AXIS_n] section of the ini file. Usually there is some overshoot as depicted in the figure. Search velocity is usually fast enough so you won't fall asleep waiting for the axis to traverse from one side to the other, but not so fast that damage could be done.
2. The axis moves in the reverse direction until the home switch changes state (off to on or on to off) plus a little to clear the switch.
3. The axis moves toward the home switch at latch velocity as defined by LATCH_VEL in the [AXIS_n] section of the ini file. Usually this is at a much lower speed than search velocity to get a more accurate position. When the switch changes state if a home offset is defined it is applied.
4. The axis moves to the home position. Usually the home location is not at the switch to prevent false tripping of the switch.

Figure 6.2 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.

Figure 6.2: Homing Sequences



6.3.3 Homing Configuration

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

SEARCH_VEL	LATCH_VEL	USE_INDEX	Homing Type
nonzero	nonzero	NO	Switch-only
nonzero	nonzero	YES	Switch + Index
0	nonzero	YES	Index-only
0	0	NO	None
Other combinations			Error

Table 6.1: Homing Types

6.3.3.1 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.

6.3.3.2 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.

6.3.3.3 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.

6.3.3.4 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.

6.3.3.5 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.

6.3.3.6 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.

6.3.3.7 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.

6.3.3.8 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.

6.4 Lathe

6.4.1 Default Plane

When EMC's interperter was first written, it was designed for mills. That is why the default plane is XY (G17). A normal lathe only uses the XZ plane(G18). To change the default plane place the following line in the .ini file in the RS274NGC section.

```
RS274NGC_STARTUP_CODE = G18
```

Chapter 7

EMC2 and HAL

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

7.1 motion (realtime)

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

7.1.1 Pins

motion.adaptive-feed IN float When adaptive feed is enabled with `M52 P1` (See the G Code Manual), 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 the G Code Manual), 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.

7.1.2 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 real time 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.

7.1.3 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

7.2 axis.N (realtime)

These pins and parameters are created by the real time `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.

7.2.1 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 jog wheel 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.

7.2.2 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

7.3 iocontrol (userspace)

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

7.3.1 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 e-stop 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 e-stop condition exists

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

Part VI

HAL

Chapter 8

Getting Started

8.1 Hal Commands

8.1.1 loadrt

The command "loadrt" loads a real time HAL component. Real time components need to be added to threads to do anything. You can not load a user space component into the real time space.

The syntax and an example:

```
loadrt <component> <options>
loadrt mux4 count=1
```

8.1.2 addf

The command "addf" adds a function to a real time thread. If you used the Stepper Config Wizard to generate your config you will have two threads.

- base-thread (the high speed thread) this thread handles items that need a fast response like making pulses, reading and writing to the parallel port.
- servo-thread (the slow speed thread) this thread handles items that don't quite need as fast of a thread like motion controller, ClassicLadder, motion command handler.

The syntax and an example:

```
addf <component> <thread>
addf mux4 servo-thread
```

8.1.3 loadusr

The command "loadusr" loads a user space HAL component. User space programs are their own separate processes, which optionally talk to other HAL components via pins and parameters. You can not load real time components into user space.

The syntax and an example:

```
loadusr <component> <options>
loadusr or2 count=2
```

8.1.4 net

The command "net" creates a "connection" between a signal and and one or more pins. The direction indicator "<= and =>" is only to make it easier to read for humans and is not used by net.

The syntax and an example:

```
net <signal-name> <pin-name> <opt-direction> <opt-pin-name>
net both-home-y <= paraport.0.pin-11-in
```

8.1.5 setp

The command "setp" sets the value of a pin or parameter. The values will depend on the type of the pin or parameter.

- bit values = true or 1 and false or 0 (True, TRUE, true are all valid)
- float values = a 32 bit floating point value, with approximately 24 bits of resolution and over 200 bits of dynamic range.
- s32 values = integer numbers -2147483648 to 2147483647
- u32 values = integer numbers 0 to 4294967295

For more information on floating point numbers see:

http://en.wikipedia.org/wiki/Floating_point

Some components have parameters that need to be set before use. Parameters can be set before use or while running as needed. You can not use setp on a pin that is connected to a signal.

The syntax and an example:

```
setp <pin/parameter-name> <value>
setp parport.0.pin-08-out TRUE
```

8.2 Hal Files

If you used the Stepper Config Wizard to generate your config you will have up to three HAL files in your config directory.

- my-mill.hal (if your config is named "my-mill") This file is loaded first and should not be changed if you used the Stepper Config Wizard.
- custom.hal This file is loaded next and before the GUI loads. This is where you put your custom HAL commands that you want loaded before the GUI is loaded.
- custom_postgui.hal This file is loaded after the GUI loads. This is where you put your custom HAL commands that you want loaded after the GUI is loaded. Any HAL commands that use pyVCP widgets need to be placed here.

8.3 Logic Components

Hal contains several real time logic components. Logic components follow a "Truth Table" that states what the output is for any given input. Typically these are bit manipulators and follow electrical logic gate truth tables.

8.3.1 and2

The "and2" component is a two input "and" gate. The truth table below shows the ouput based on each combination of input.

Syntax

```
and2 [count=N|names=name1[,name2...]]
```

Functions

```
and2.n
```

Pins

```
and2.N.in0 (bit, in)
and2.N.in1 (bit, in)
and2.N.out (bit, out)
```

Truth Table

in0	in1	out
False	False	False
True	False	False
False	True	False
True	True	True

8.3.2 not

The "not" component is a bit inverter.

Syntax

```
not [count=n|names=name1[,name2...]]
```

Functions

```
not.n
```

Pins

```
not.n.in (bit, in)
not.n.out (bit, out)
```

Truth Table

in	out
True	False
False	True

8.3.3 or2

The "or2" component is a two input OR gate.

Syntax

```
or2[count=,|names=name1[,name2...]]
```

Functions

```
or2.n
```

Pins

```
or2.n.in0 (bit, in)
or2.n.in1 (bit, in)
or2.n.out (bit, out)
```

Truth Table

in0	in1	out
True	False	True
True	True	True
False	True	True
False	False	False

8.3.4 xor2

The "xor2" component is a two input XOR (exclusive OR)gate.

Syntax

```
xor2[count=,|names=name1[,name2...]]
```

Functions

```
xor2.n
```

Pins

```
xor2.n.in0 (bit, in)
xor2.n.in1 (bit, in)
xor2.n.out (bit, out)
```

Truth Table

in0	in1	out
True	False	True
True	True	False
False	True	True
False	False	False

8.3.5 Logic Examples

An "and2" example connecting two inputs to one output.

```
loadrt and2 count=1
addf and2.0 servo-thread
net my-sigin1 and2.0.in0 <= paraport.0.pin-11-in
net my-sigin2 and2.0.in.1 <= paraport.0.pin-12-in
net both-on paraport.0.pin-14-out <= and2.0.out
```

In the above example one copy of and2 is loaded into real time space and added to the servo thread. Next pin 11 of the parallel port is connected to the in0 bit of the and gate. Next pin 12 is connected to the in1 bit of the and gate. Last we connect the and2 out bit to the parallel port pin 14. So following the truth table for and2 if pin 11 and pin 12 are on then the output pin 14 will be on.

Chapter 9

Basic Configuration

9.1 Introduction

The preferred way to set up a standard stepper machine is with the Step Configuration Wizard. See the Getting Started Guide for more information on the Step Configuration Wizard.

This chapter describes some of the more common settings for manually setting up a stepper based system. 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 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`.

9.2 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)

9.3 Pinout

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

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:

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

Depending on what you have chosen in your ini file you are using either standard_pinout.hal or xylotex_pinout.hal. These are two files that instruct the HAL how to link the various signals & pins. Further on we'll investigate the standard_pinout.hal.

9.3.1 standard_pinout.hal

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

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

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

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

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

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

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

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

```
###

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

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

9.3.2 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 12.1 for details)
2. The read & write functions of the parport driver get assigned to the Base thread [1]
3. The step & direction signals for axes X,Y,Z get linked to pins on the parport
4. Further IO signals get connected (e-stop loopback, toolchanger loopback)
5. A spindle On signal gets defined and linked to a parport pin

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

9.3.3 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 basically any other numbers you like.

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

9.3.4 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
```

9.3.5 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.

9.3.6 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 as well, so having only one signal / pin is perfectly safe.

9.3.7 Adding an external E-STOP button

As you can see in 9.3.1 by default the stepper configuration assumes no external E-STOP button. [2]

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 E-STOP switch connected to pin 01 on the parport. As long as the switch will stay pushed[3], EMC2 will be in the E-STOP state. When the external button gets released EMC2 will immediately switch to the E-STOP-RESET state, and all you need to do is switch to Machine On and you'll be able to continue your work with EMC2.

[2]An extensive explanation of hooking up E-STOP circuitry is explained in the wiki.linuxcnc.org and in the Integrator Manual

[3]make sure you use a maintained switch for E-STOP.

Chapter 10

HAL Components

10.1 Commands and Userspace Components

Some of these will have expanded descriptions from the man pages. Some will have limited descriptions. All of the components have man pages. From this list you know what components exist and can use man n name to get additional information. For example in a terminal window type **`man 1 axis`** to view the information in the man page.

axis-remote.1	= AXIS Remote Interface
axis.1	= AXIS EMC (The Enhanced Machine Controller) Graphical User Interface
bfload.1	= A program for loading a Xilinx Bitfile program into the FPGA of an Anything I/O board from Mesa
comp.1	= Build, compile and install EMC HAL components
emc.1	= EMC (The Enhanced Machine Controller)
hal_input.1	= control HAL pins with any Linux input device, including USB HID devices
hal_joystick.1	= control HAL pins with a joystick (deprecated use hal_input)
halcmd.1	= manipulate the Enhanced Machine Controller HAL from the command line
halmeter.1	= observe HAL pins, signals, and parameters
halrun.1	= manipulate the Enhanced Machine Controller HAL from the command line
halsampler.1	= sample data from HAL in realtime
halstreamer.1	= stream file data into HAL in real time
halui.1	= observe HAL pins and command EMC through NML
io.1	= accepts NML I/O commands, interacts with HAL in userspace
iocontrol.1	= accepts NML I/O commands, interacts with HAL in userspace
pyvcp.1	= Virtual Control Panel for EMC2

10.2 Realtime components and kernel modules

Some of these will have expanded descriptions from the man pages. Some will have limited descriptions. All of the components have man pages. From this list you know what components exist and can use man n name to get additional information. For example to find the man page for abs.9 open a terminal window and type:

```
    man abs.9
```

abs.9	= Compute the absolute value and sign of the input signal
and2.9	= Two-input AND gate
at_pid.9	= proportional/integral/derivative controller with auto tuning
axis.9	= accepts NML motion commands, interacts with HAL in realtime
biquad.9	= Biquad IIR filter
blend.9	= Perform linear interpolation between two values
blocks.9	= Old style HAL blocks (deprecated)
charge_pump.9	= Create a square-wave for the âcharge pumpâ input of some controller boards
clarke2.9	= Two input version of Clarke transform
clarke3.9	= Clarke (3 phase to cartesian) transform
clarkeinv.9	= Inverse Clarke transform
classicladder.9	= Realtime software plc based on ladder logic
comp.9	= Two input comparator with hysteresis
constant.9	= Use a parameter to set the value of a pin
conv_bit_s32.9	= Convert a value from bit to s32
conv_bit_u32.9	= Convert a value from bit to u32
conv_float_s32.9	= Convert a value from float to s32
conv_float_u32.9	= Convert a value from float to u32
conv_s32_bit.9	= Convert a value from s32 to bit
conv_s32_float.9	= Convert a value from u32 to bit
conv_s32_u32.9	= Convert a value from s32 to u32
conv_u32_bit.9	= Convert a value from u32 to bit
conv_u32_float.9	= Convert a value from u32 to float
conv_u32_s32.9	= Convert a value from u32 to s32
counter.9	= counts input pulses (deprecated)
ddt.9	= Compute the derivative of the input function
deadzone.9	= Return the center if within the threshold
debounce.9	= filter noisy digital inputs, for more information see 11.6
edge.9	= Edge detector

encoder.9	= software counting of quadrature encoder signals, for more information see 11.3
encoder_ratio.9	= an electronic gear to synchronize two axes
estop_latch.9	= ESTOP latch
flipflop.9	= D type flip-flop
freqgen.9	= software step pulse generation
genhexkins.9	= kinematics definitions for emc2
hypot.9	= Three-input hypotenuse (Euclidean distance) calculator
integ.9	= Integrator
kins.9	= kinematics definitions for emc2
knob2float.9	= Convert counts (probably from an encoder) to a float value
limit1.9	= Limit the output signal to fall between min and max
limit2.9	= Limit the output signal to fall between min and max
limit3.9	= Limit the output signal to fall between min and max
logic.9	=
lowpass.9	= Low-pass filter
lut5.9	= Arbitrary 5-input logic function based on a look-up table
m7i43_hm2.9	= RTAI driver for the Mesa 7i43 EPP Anything IO board with HostMot2 firmware
maj3.9	= Compute the majority of 3 inputs
match8.9	= 8-bit binary match detector
minmax.9	= Track the minimum and maximum values of the input to the outputs
motion.9	= Accepts NML motion commands, interacts with HAL in realtime
mult2.9	= Product of two inputs
mux2.9	= Select from one of two input values based on the status of the sel bit.
mux4.9	= Select from one of four input values based on the status of the two sel bits.
not.9	= Inverter
offset.9	= Adds an offset to an input, and subtracts it from the feedback value
oneshot.9	= one-shot pulse generator
or2.9	= two-input OR gate
pid.9	= proportional/integral/derivative controller, for more information see 11.4
pluto_servo.9	= Hardware driver and firmware for the Pluto-P parallel-port FPGA, for use with servos
pluto_step.9	= Hardware driver and firmware for the Pluto-P parallel-port FPGA, for use with steppers
pwmgen.9	= software PWM/PDM generation, for more information see 11.2
rotatekins.9	= kinematics definitions for emc2

sample_hold.9	= Sample and Hold
sampler.9	= sample data from HAL in real time
scale.9	=
select8.9	= 8-bit binary match detector
serport.9	= Hardware driver for the digital I/O bits of the 8250 and 16550 serial port
siggen.9	= signal generator, for more information see 11.7
sim_encoder.9	= simulated quadrature encoder, for more information see 11.5
stepgen.9	= software step pulse generation, for more information see 11.1
steptest.9	= Used by Stepconf to allow testing of acceleration and velocity values for an axis
streamer.9	= stream file data into HAL in real time
sum2.9	= Sum of two inputs (each with a gain) and an offset
supply.9	= set output pins with values from parameters (deprecated)
threads.9	= creates hard realtime HAL threads
threadtest.9	=
timedelta.9	=
toggle.9	= push-on, push-off from momentary pushbuttons
tripodkins.9	= kinematics definitions for emc2
tristate_bit.9	= Place a signal on an I/O pin only when enabled, similar to a tristate buffer in electronics
tristate_float.9	= Place a signal on an I/O pin only when enabled, similar to a tristate buffer in electronics
trivkins.9	= kinematics definitions for emc2
updown.9	= Counts up or down, with optional limits and wraparound behavior
wcomp.9	= Window comparator
weighted_sum.9	= convert a group of bits to an integer
xor2.9	= Two-input XOR (exclusive OR) gate

Chapter 11

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.

11.1 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 11.1 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.

11.1.1 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.

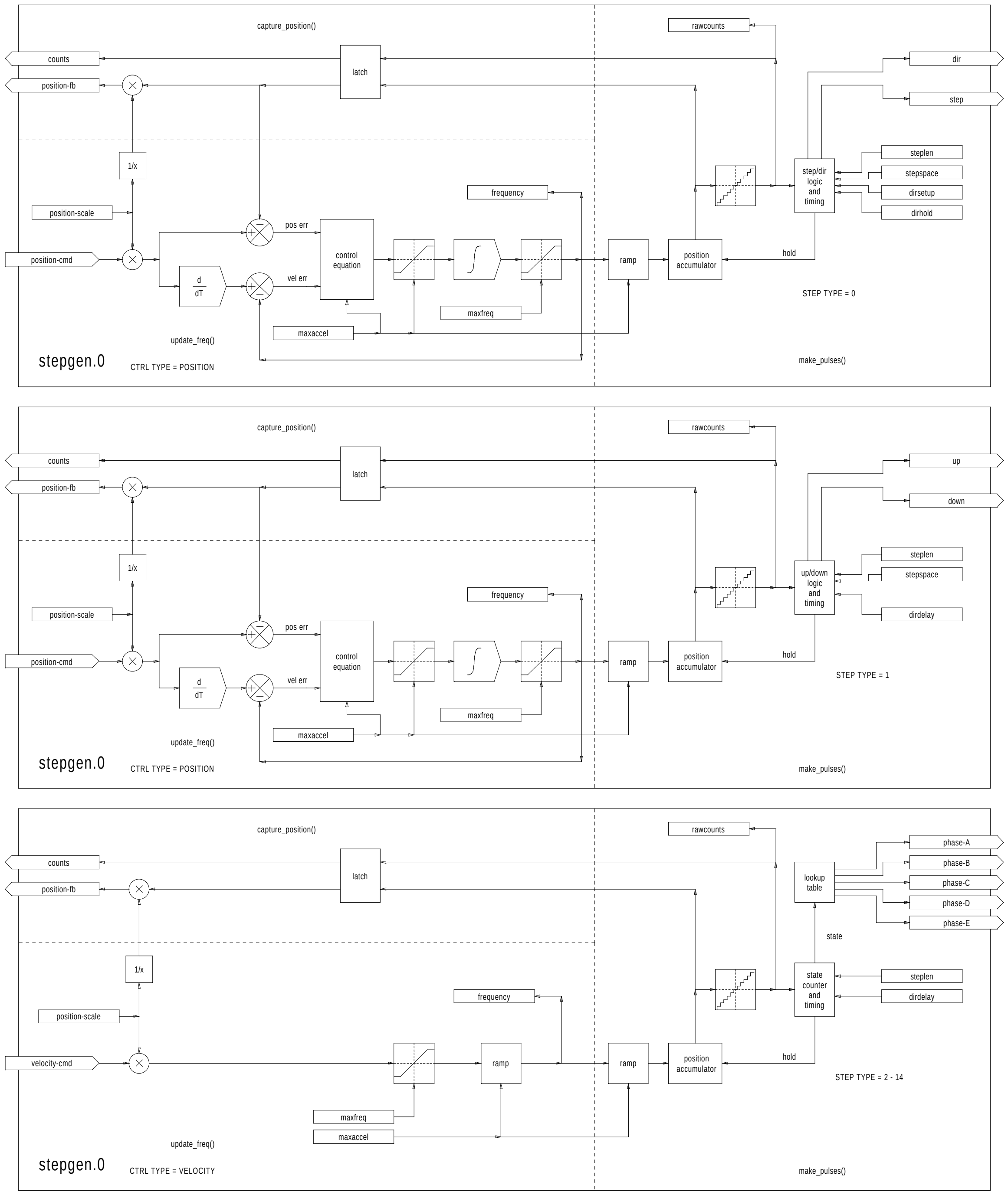


Figure 11.1: Step Pulse Generator Block Diagram (position mode)

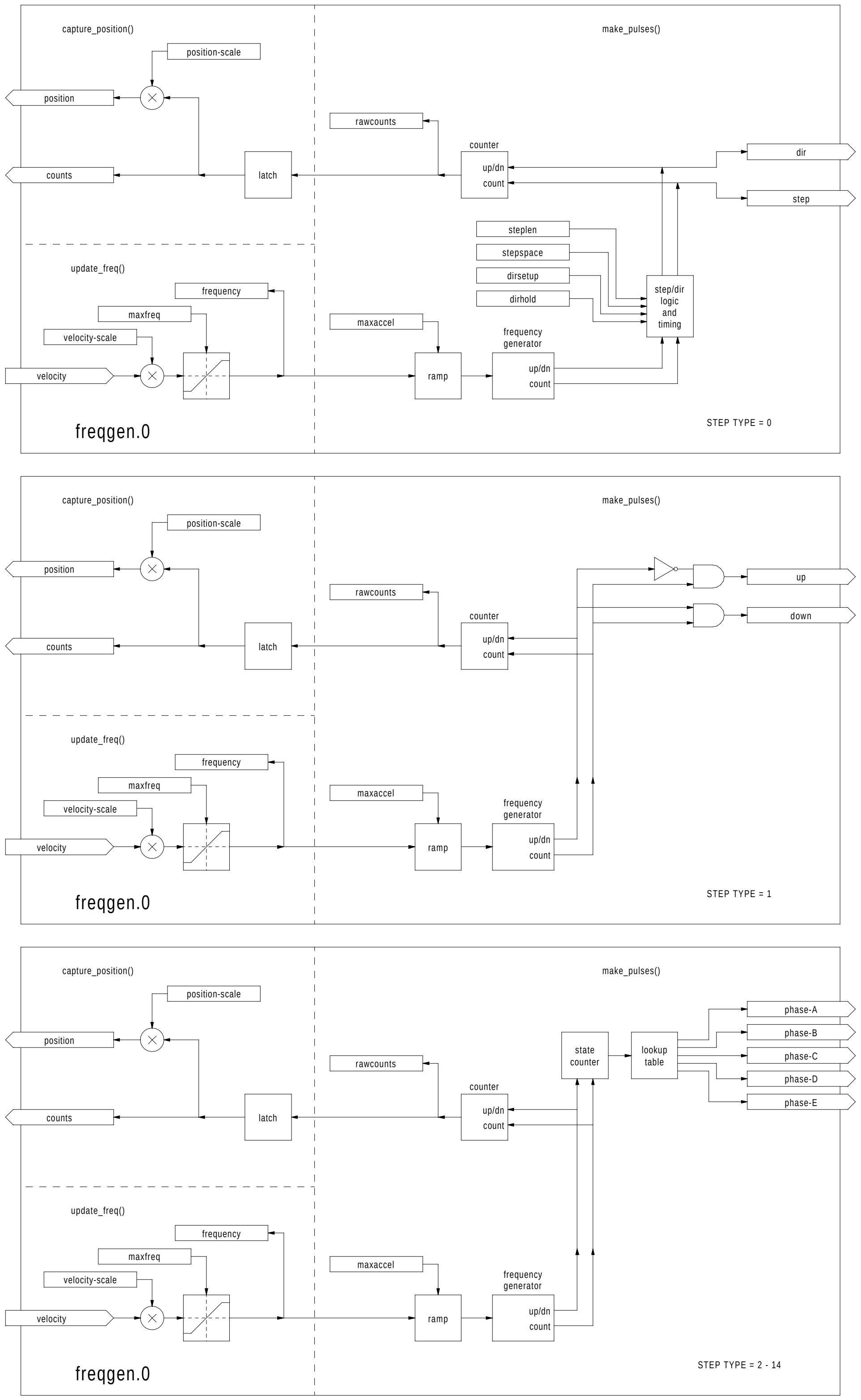


Figure 11.2: Step Pulse Generator Block Diagram (velocity mode)

11.1.2 Removing

```
emc2$ halcmd unloadrt stepgen
```

11.1.3 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).

11.1.4 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.

11.1.5 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 11.3 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 11.4, 11.5, and 11.6 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.

11.1.6 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.

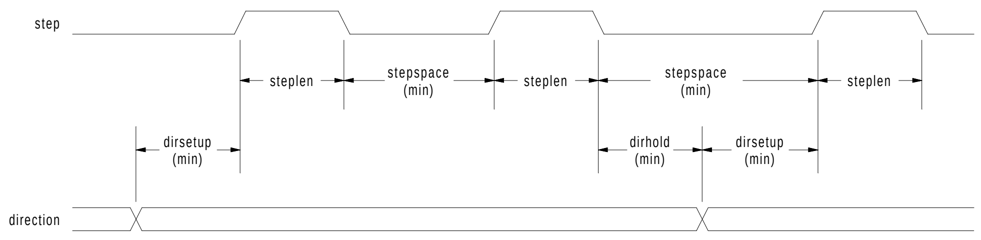


Figure 11.3: Step and Direction Timing

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.

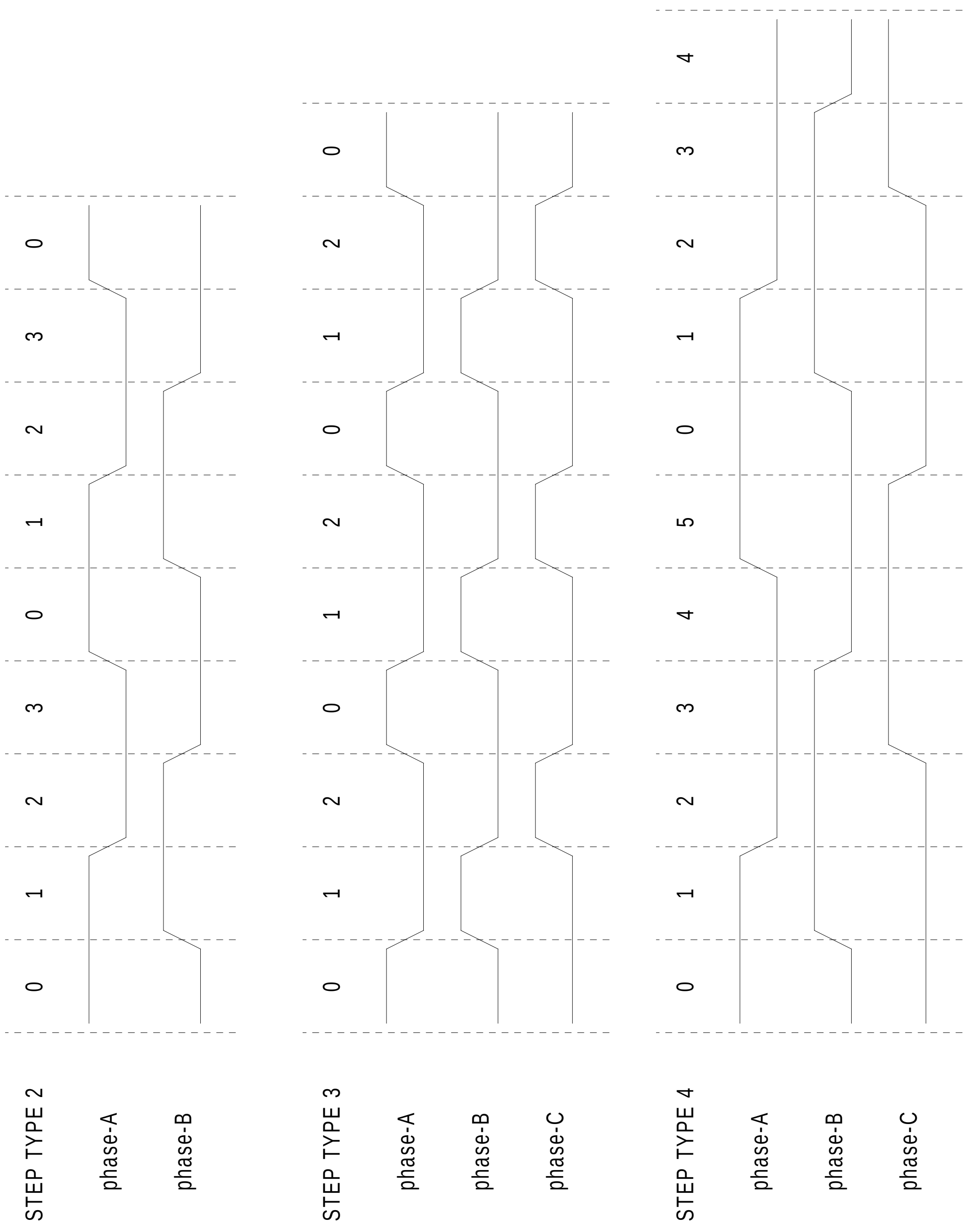


Figure 11.4: Three-Phase step types

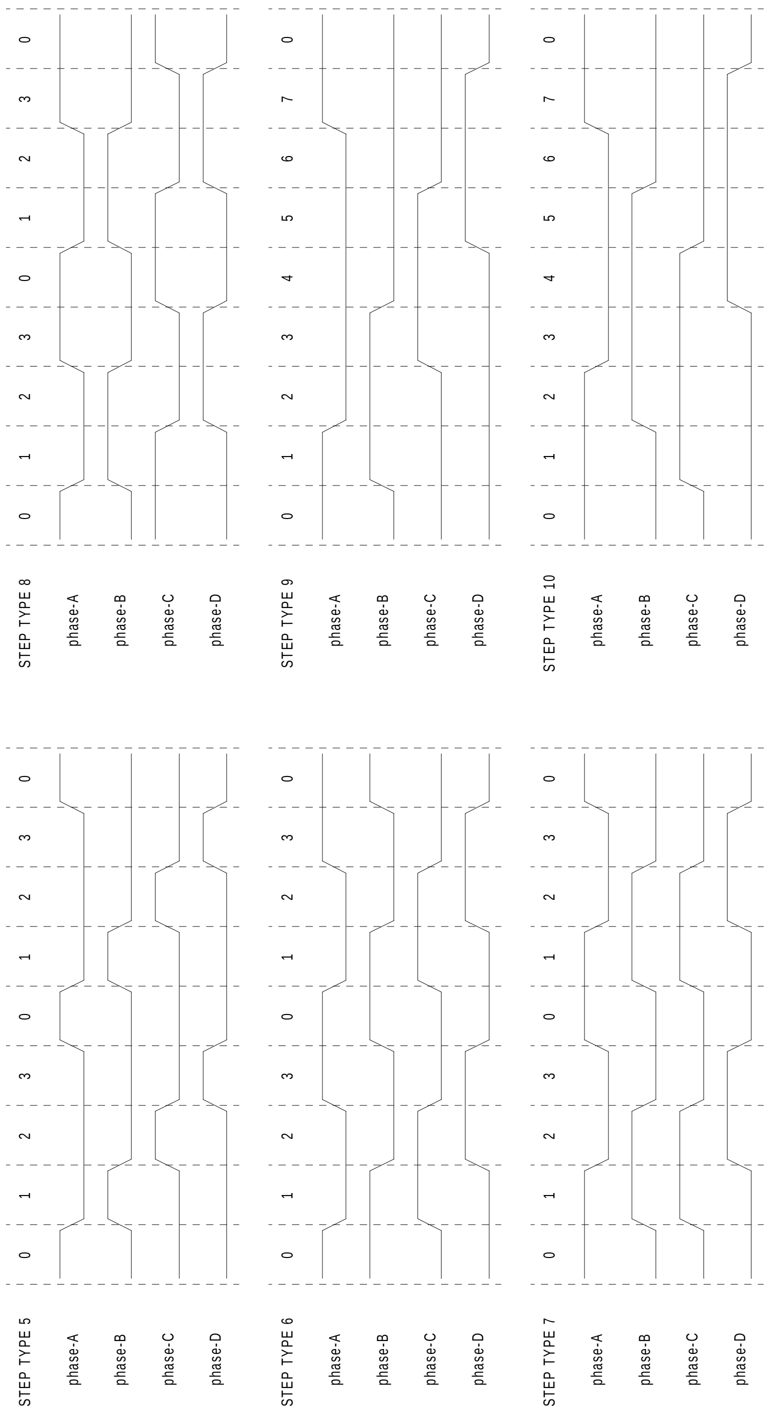


Figure 11.5: Four-Phase Step Types

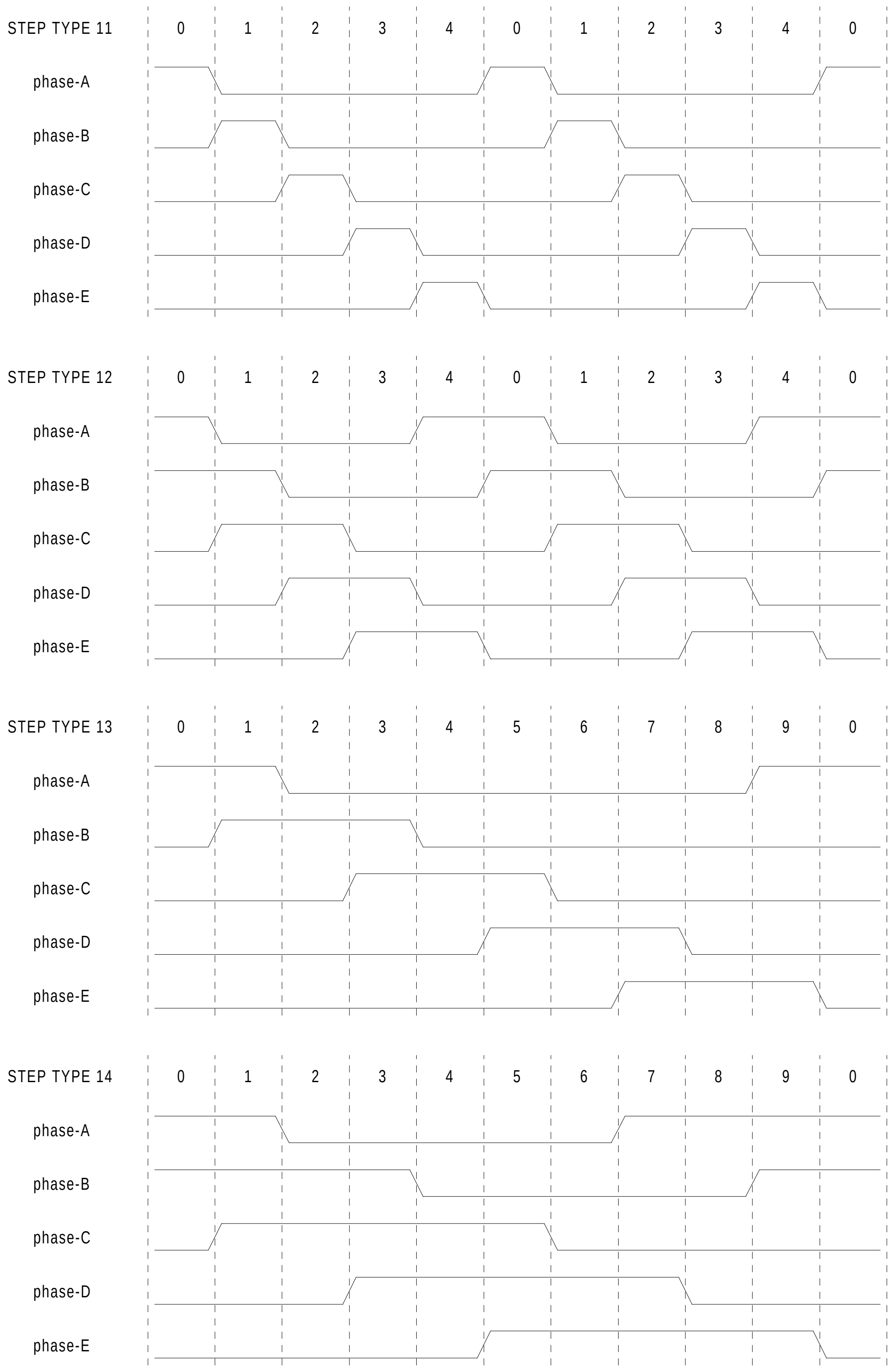


Figure 11.6: Five-Phase Step Types

11.2 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.

11.2.1 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.

11.2.2 Removing

```
emc2$ halcmd unloadrt pwmgen
```

11.2.3 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).

11.2.4 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).

11.2.5 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.

11.2.6 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.

11.3 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 11.7 is a block diagram of one channel of encoder counter.

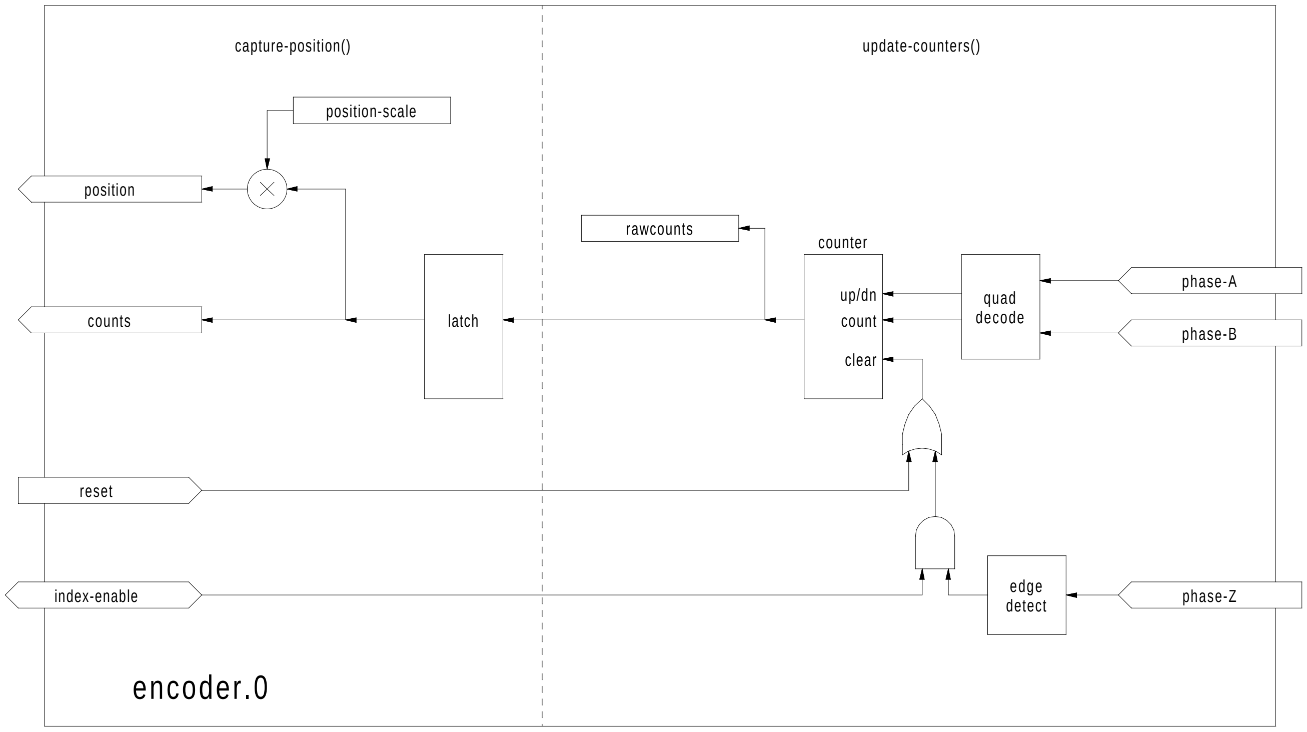


Figure 11.7: Encoder Counter Block Diagram

11.3.1 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.

11.3.2 Removing

```
emc2$ halcmd unloadrt encoder
```

11.3.3 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 **??**.
- `(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.

11.3.4 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 **??**.

11.3.5 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.

11.4 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 11.8 is a block diagram of a single PID loop.

11.4.1 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.

11.4.2 Removing

```
emc2$ halcmd unloadrt pid
```

11.4.3 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.

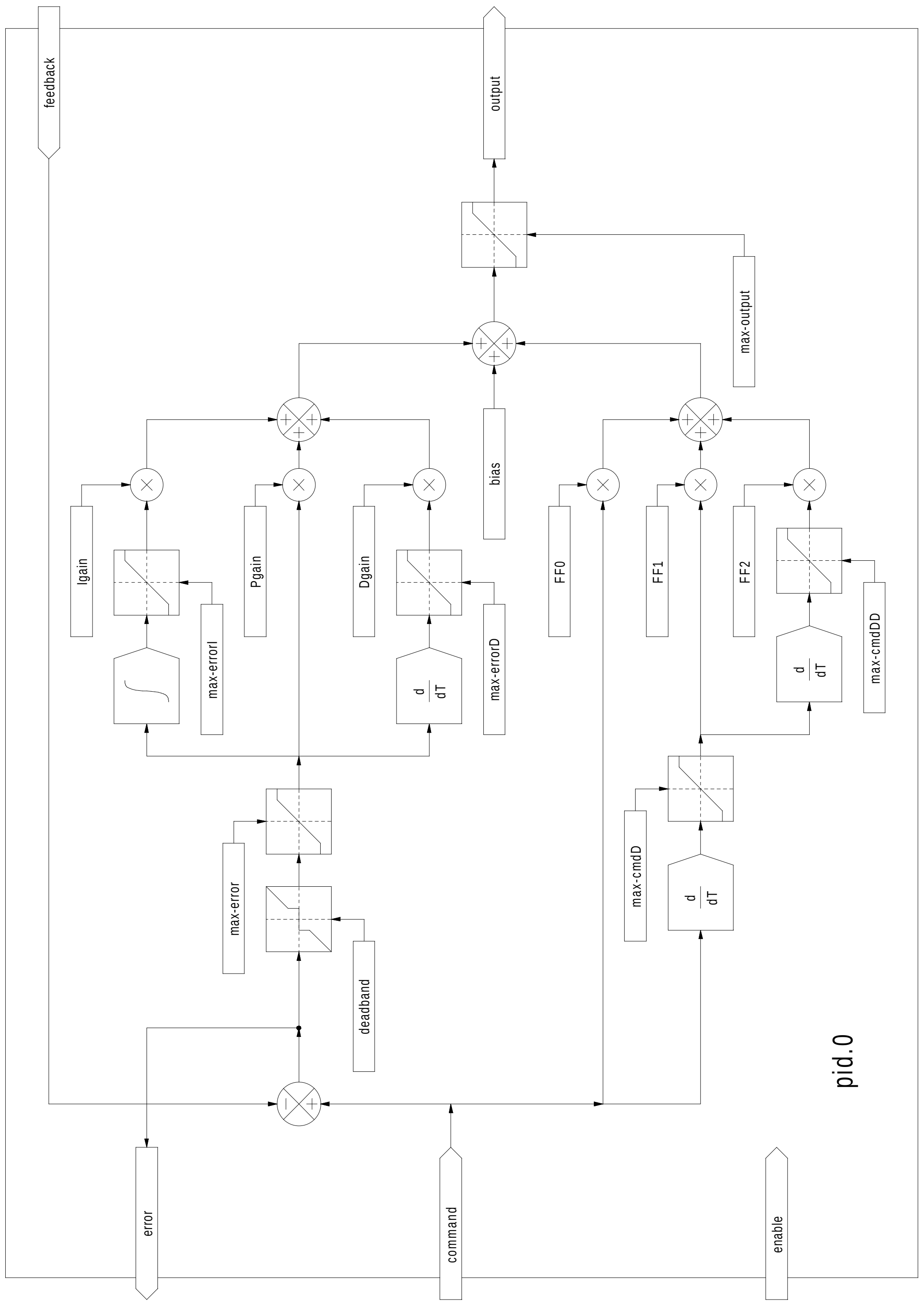


Figure 11.8: PID Loop Block Diagram

11.4.4 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.

11.4.5 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 11.8, 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

11.5 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.

11.5.1 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).

11.5.2 Removing

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

11.5.3 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`.

11.5.4 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.

11.5.5 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**.

11.6 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.

11.6.1 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.

11.6.2 Removing

```
emc2$ halcmd unloadrt debounce
```

11.6.3 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>`.

11.6.4 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.

11.6.5 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.

11.7 Siggen

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

11.7.1 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).

11.7.2 Removing

```
emc2$ halcmd unloadrt siggen
```

11.7.3 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.

11.7.4 Parameters

None. [3]

11.7.5 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 12

Hardware Drivers

12.1 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 12.1 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.

12.1.1 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.

12.1.2 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.

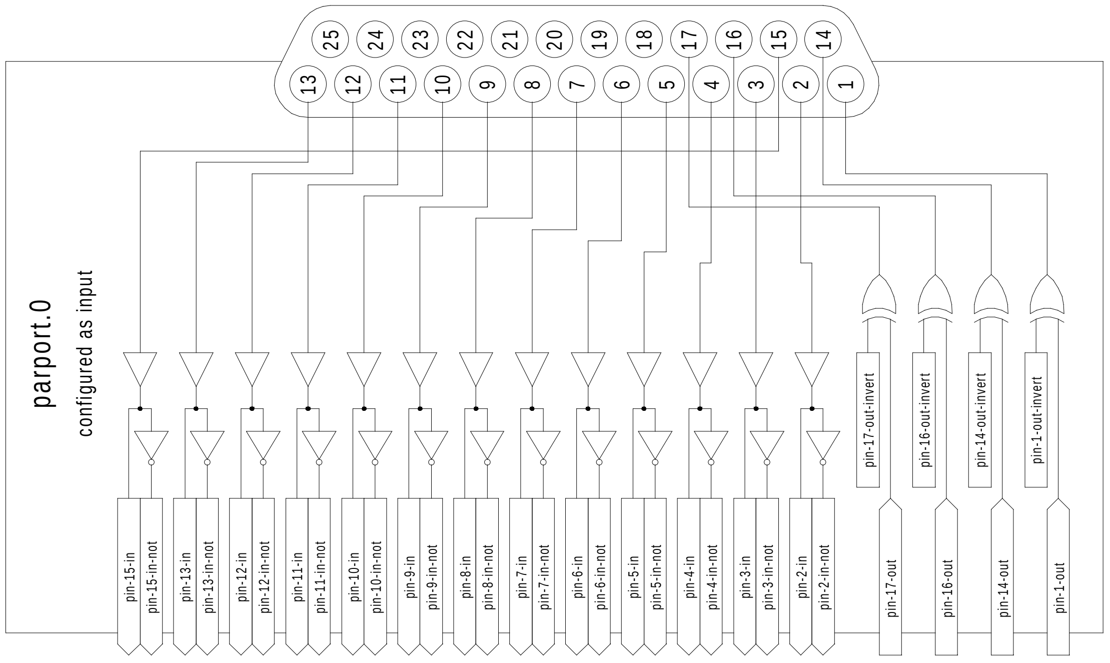
For each physical input pin, the driver creates two HAL pins, for example `parport.0.pin-12-in` and `parport.0.pin-12-in-not`. Pins 10, 11, 12, 13, and 15 are always input pins. Pins 2 through 9 are input pins only if the port is defined as an input port. The `-in` HAL pin is TRUE if the physical pin 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.

12.1.3 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.



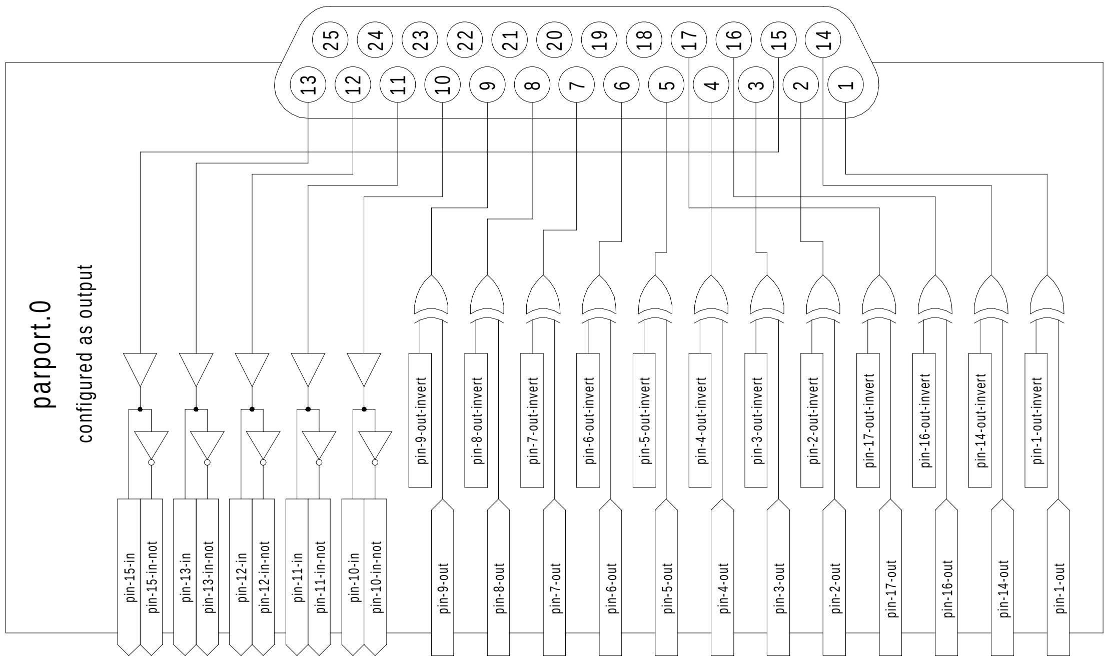


Figure 12.1: Parport Block Diagram

12.1.4 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.

12.1.5 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.

12.2 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.

12.2.1 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.

12.3 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]

12.3.1 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.

12.3.2 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.

12.3.3 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.

12.3.4 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.

12.4 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.

12.4.1 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.

12.4.2 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.

12.4.3 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.

12.4.4 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.

12.5 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.

12.5.1 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.

12.5.2 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.

12.5.3 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.

12.5.4 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.

12.5.4.1 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

12.5.4.2 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	

12.5.4.3 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	

12.5.4.4 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 ?

12.6 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.

12.6.1 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.

12.6.2 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?

12.6.3 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.

12.7 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.

12.7.1 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.

12.7.2 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.[12]

12.7.3 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.

12.8 Pluto-P: generalities

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

12.8.1 Requirements

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

12.8.2 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.

[12] In a future version this will be changed from .invert to -invert to better match the HAL canonical bit interface

12.8.3 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.

12.8.4 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**).

12.8.5 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.

12.8.6 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.

12.8.7 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.

12.8.8 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`.

12.9 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.

12.9.1 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.

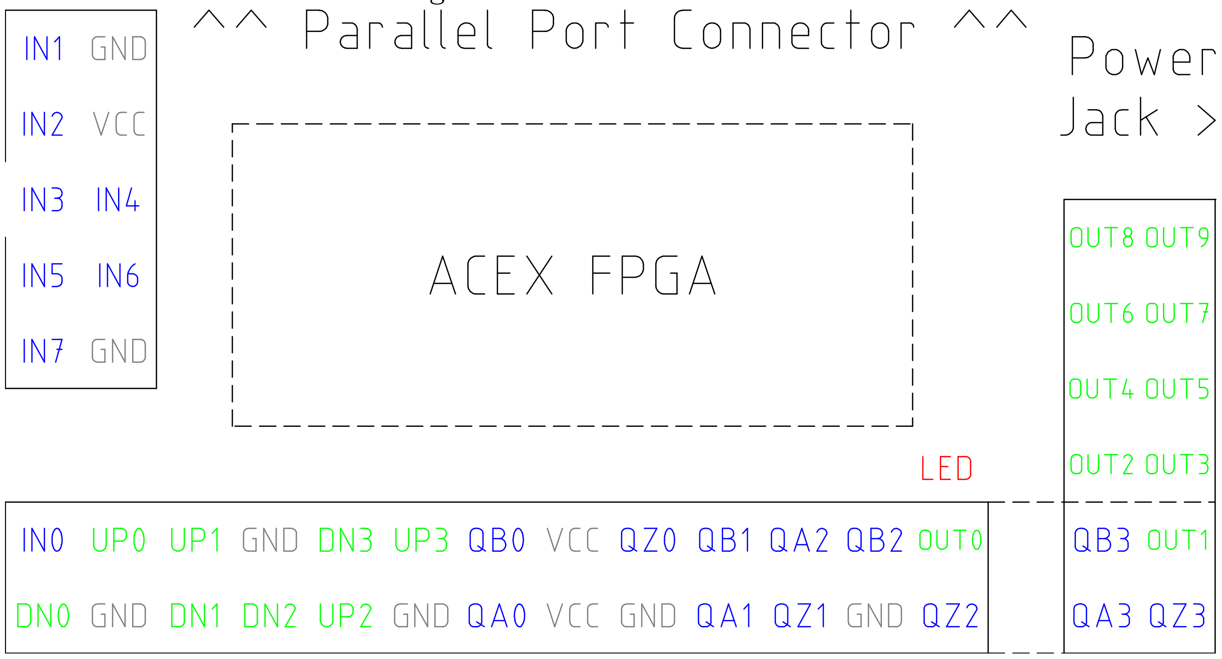
INx Dedicated digital input #x

OUTx Dedicated digital output #x

GND Ground

VCC +3.3V regulated DC

Figure 12.2: Pluto-Servo Pinout


^^ Parallel Port Connector ^^
Power Jack >
ACEX FPGA
LED
IN1 GND
IN2 VCC
IN3 IN4
IN5 IN6
IN7 GND
OUT8 OUT9
OUT6 OUT7
OUT4 OUT5
OUT2 OUT3
IN0 UP0 UP1 GND DN3 UP3 QB0 VCC QZ0 QB1 QA2 QB2 OUT0
DN0 GND DN1 DN2 UP2 GND QA0 VCC GND QA1 QZ1 GND QZ2
QB3 OUT1
QA3 QZ3

12.9.2 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.

12.9.3 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*.

12.9.4 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 12.1: 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)

12.10 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

12.10.1 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.

12.10.2 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.

12.10.3 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 12.3: Pluto-Step Pinout

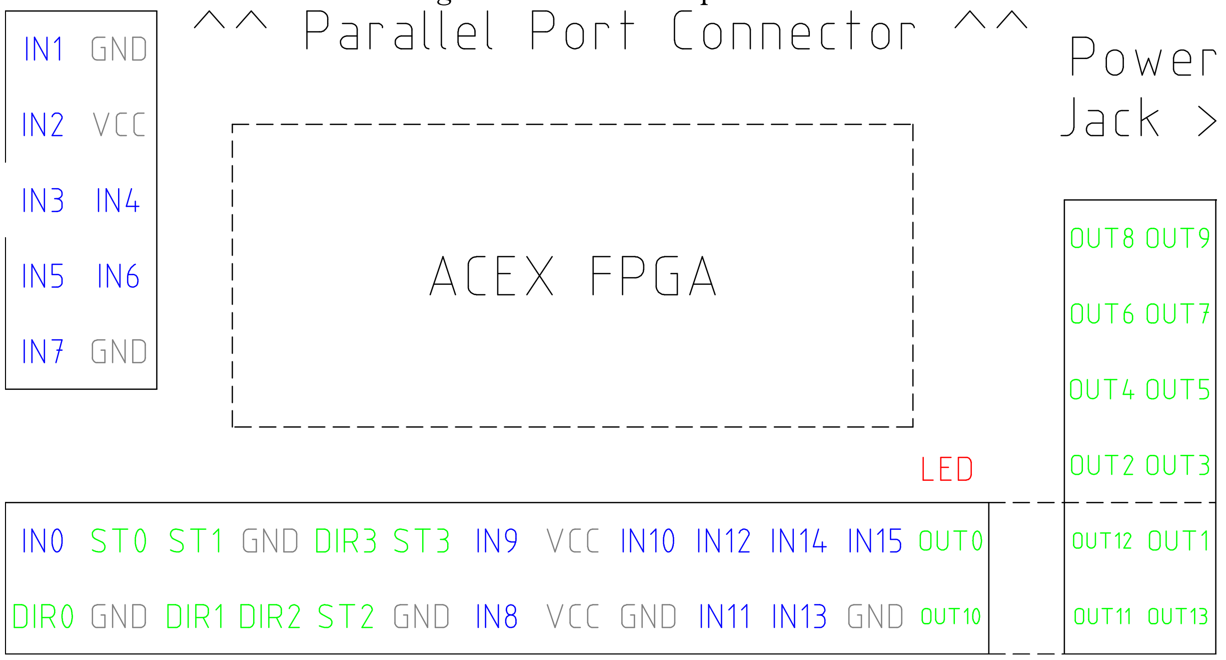
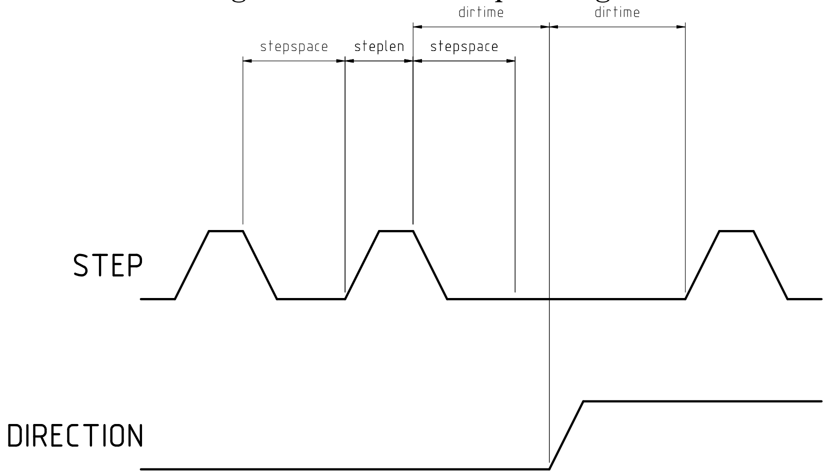

^^ Parallel Port Connector ^^
Power Jack >
IN1 GND
IN2 VCC
IN3 IN4
IN5 IN6
IN7 GND
ACEX FPGA
LED
OUT8 OUT9
OUT6 OUT7
OUT4 OUT5
OUT2 OUT3
OUT12 OUT1
OUT11 OUT13
IN0 ST0 ST1 GND DIR3 ST3 IN9 VCC IN10 IN12 IN14 IN15 OUT0
DIR0 GND DIR1 DIR2 ST2 GND IN8 VCC GND IN11 IN13 GND OUT10

Figure 12.4: Pluto-Step Timings



12.10.4 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 13

Halui

13.1 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 add halui is to add the following to the [HAL] section of the ini file.

```
HALUI = halui
```

An alternate way to invoke it (especially when using a stepconf generated config file) is to include the following in your custom.hal file. Make sure you use the actual path to your ini file.

```
loadusr halui -ini /path/to/inifile.ini
```

in your custom.hal file.

13.2 Halui pin reference

13.2.1 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

13.2.2 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

13.2.3 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

13.2.4 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

13.2.5 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

13.2.6 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

13.2.7 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

13.2.8 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

13.2.9 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)

13.2.10 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)

13.2.11 Tool

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

13.2.12 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

13.2.13 General

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

13.2.14 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. 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).

13.3 Case - Studies

User descriptions of working halui and hardware EMC control panels here.

Chapter 14

Virtual Control Panels

14.1 Introduction

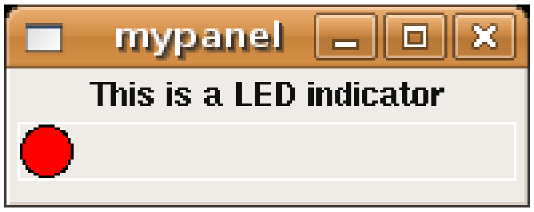
Traditional machine control panels are large sheets of steel with push buttons, knobs, lights and sometimes meters mounted on them. They have many advantages - the buttons are far more rugged than a computer keyboard, and large enough that you can usually operate the correct one by feel while looking 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 substitute for a big rugged oil-tight push button. 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.

14.2 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

```
halrun -I loadusr pyvcp -c mypanel tiny.xml
```

pyVCP will create the panel for you, which includes two widgets, a Label with the text "This is a LED indicator", and a LED, used for displaying the state of a HAL BIT signal. It will also create a HAL component named "mypanel" (all widgets in this panel are connected to pins that start with "mypanel."). Since no <halpin> tag was present inside the <led> tag, pyVCP will automatically name the HAL pin for the LED widget mypanel.led.0

For a list of widgets and their tags and options, see the widget reference below.

Once you have created your panel, connecting HAL signals to and from the pyVCP pins is done with:

```
halcmd linksp
```

If you are new to HAL, the HAL Tutorial**??** is recommended.

14.3 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.

14.4 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 start up, we need to specify the following in the [DISPLAY] section of the .ini file:

```
PYVCP = spindle.xml
```

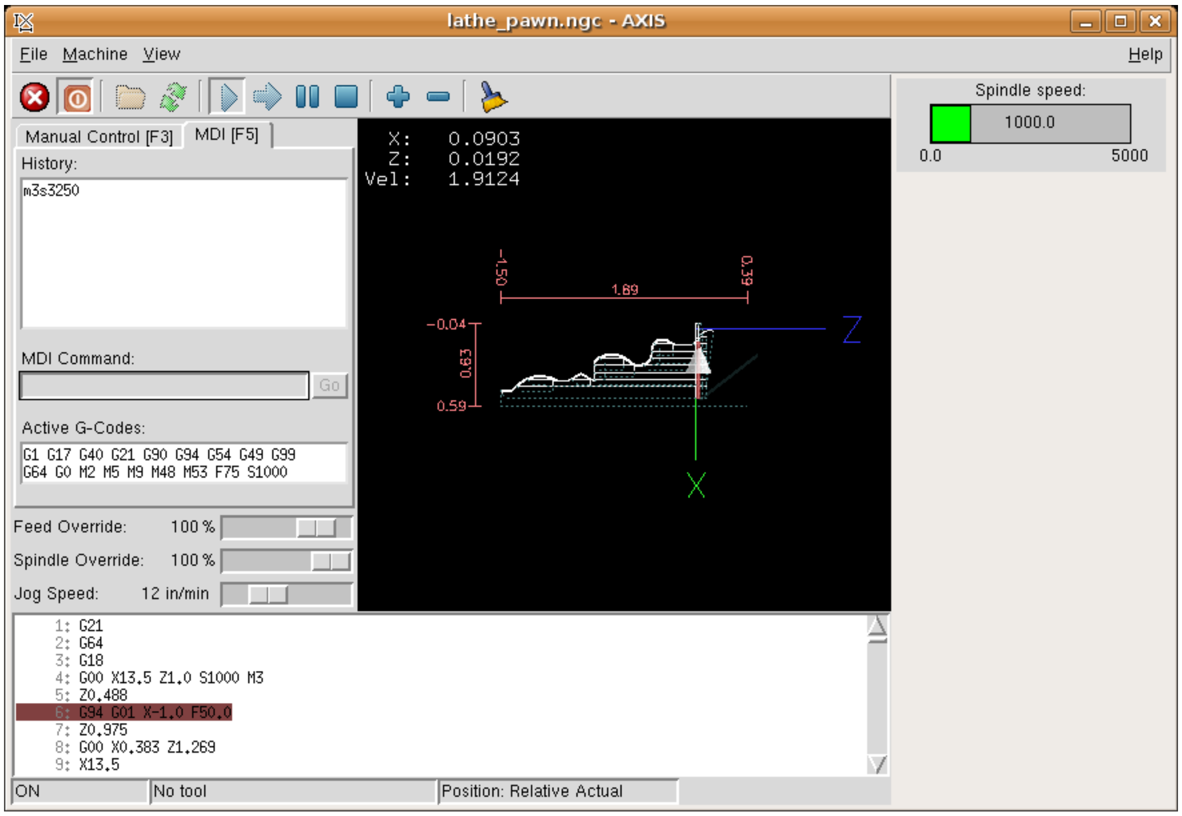
To make our widget actually display the spindle-speed it needs to be hooked up to the appropriate HAL signal. A .hal file that will be run once AXIS and pyVCP have started can be specified in the [HAL] section of the .ini file:

```
POSTGUI_HALFILE = spindle_to_pyvcp.hal
```

This change will run the HAL commands specified in “spindle_to_pyvcp.hal”. In our example the contents could look like this:

```
linksp spindle-rpm-filtered  pyvcp.spindle-speed
```

assuming that a signal called “spindle-rpm-filtered” already exists. Note that when running together with AXIS, all pyVCP widget HAL pins have names that start with “pyvcp.”.



Spindle speed: 1000.0 (0.0 – 5000)

```
  X:   0.0903
  Z:   0.0192
Vel:   1.9124
```

History:
m3s3250

MDI Command:

Active G-Codes:
```
G1 G17 G40 G21 G90 G94 G54 G49 G99
G64 G0 M2 M5 M9 M48 M53 F75 S1000
```

Feed Override: 100 %
Spindle Override: 100 %
Jog Speed: 12 in/min

```
G21
G64
G18
G00 X13.5 Z1.0 S1000 M3
Z0.488
G94 G01 X-1.0 F50.0
Z0.975
G00 X0.383 Z1.269
X13.5
```

ON | No tool | Position: Relative Actual

This is what the newly created pyVCP panel should look like in AXIS. The `sim/lathe` configuration is already configured this way.

14.5 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

14.5.0.1 Syntax

Each widget is described briefly, followed by the markup used, and a screen shot. All tags inside the main widget tag are optional.

14.5.0.2 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 attributes value into a Python value:

1. If the first character of the attribute is one of the following, it is evaluated as a Python expression: `{(["'`
2. If the string is accepted by int(), the value is treated as an integer
3. If the string is accepted by float(), the value is treated as floating-point
4. Otherwise, the string is accepted as a string.

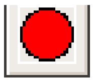
When the tag-based syntax is used, the text within the tag is always evaluated as a Python expression.

The examples below show a mix of formats.

14.5.1 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>
```

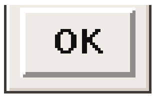


<halpin> sets the name of the pin, default is "led.n", where n is an integer
<size> sets the size of the led, default is 20
<on_color> sets the color of the LED when the pin is true. default is "green"
<off_color> sets the color of the LED when the pin is false. default is "ref"

14.5.2 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>
```



14.5.3 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.

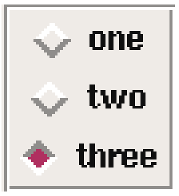
```
<checkbutton>
    <halpin>"my-checkbutton"</halpin>
</checkbutton>
```

An unchecked checkbutton: , and a checked one:

14.5.4 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>
```

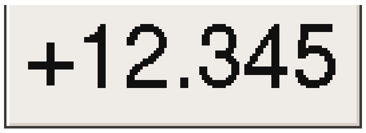


Note that the HAL pins in the example above will me named my-radio.one, my-radio.two, and my-radio.three. In the image above, "three" is the selected value.

14.5.5 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:

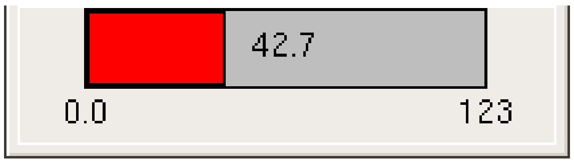
```
<font>("courier 10 pitch",100)</font>
```

<format> is a "C-style" format specified that determines how the number is displayed.

14.5.6 Bar

A bar widget displays the value of a FLOAT signal both graphically using a bar display and numerically.

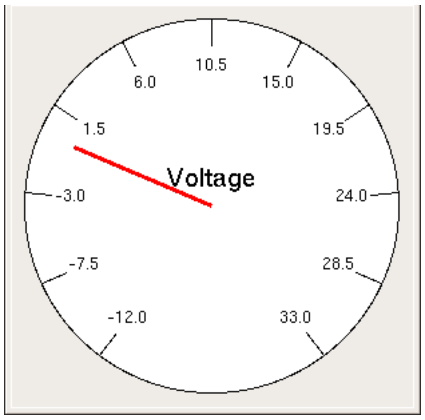
```
<bar>
    <halpin>"my-bar"</halpin>
    <min_>0</min_>
    <max_>123</max_>
    <bgcolor>"grey"</bgcolor>
    <fillcolor>"red"</fillcolor>
</bar>
```



14.5.7 Meter

Meter displays the value of a FLOAT signal using a traditional dial indicator.

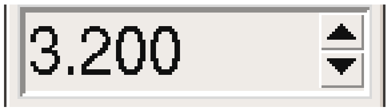
```
<meter>
    <halpin>"my-meter"</halpin>
    <text>"Voltage"</text>
    <size>300</size>
    <min_>-12</min_>
    <max_>33</max_>
</meter>
```



14.5.8 Spinbox

Spinbox controls a FLOAT pin. You increase or decrease the value of the pin by either pressing on the arrows, or pointing at the spinbox and rolling your mouse-wheel.

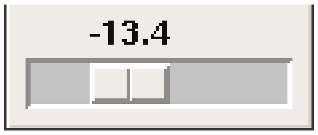
```
<spinbox>
    <halpin>"my-spinbox"</halpin>
    <min_>-12</min_>
    <max_>33</max_>
    <resolution>0.1</resolution>
    <format>"2.3f"</format>
    <font>("Arial",30)</font>
</spinbox>
```



14.5.9 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.

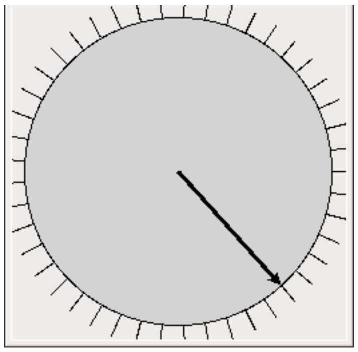
```
<scale>
    <halpin>"my-scale"</halpin>
    <resolution>0.1</resolution>
    <orient>HORIZONTAL</orient>
    <min_>-33</min_>
    <max_>26</max_>
</scale>
```



14.5.10 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>
```



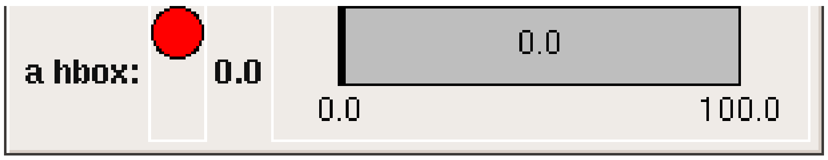
14.6 pyVCP Container reference

Containers are widgets that contain other widgets.

14.6.1 Hbox

Use a Hbox when you want to stack widgets horizontally next to each other.

```
<hbox>
    <label><text>"a vbox:"</text></label>
    <led></led>
    <number></number>
    <bar></bar>
</hbox>
```

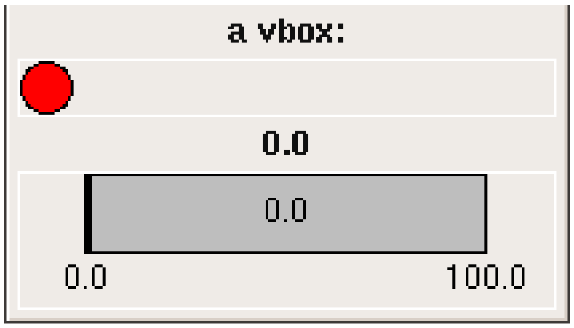


Inside a Hbox, you can use the `<boxfill fill=""/>`, `<boxanchor anchor=""/>`, and `<boxexpand expand=""/>` tags to choose how items in the box behave when the window is re-sized. For details of how fill, anchor, and expand behave, refer to the Tk `pack` manual page, `pack(3tk)`. By default, `fill="y"`, `anchor="center"`, `expand="yes"`.

14.6.2 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>
```

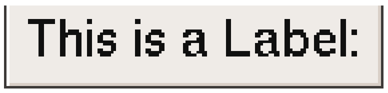


Inside a Hbox, you can use the `<boxfill fill=""/>`, `<boxanchor anchor=""/>`, and `<boxexpand expand=""/>` tags to choose how items in the box behave when the window is re-sized. For details of how fill, anchor, and expand behave, refer to the Tk `pack` manual page, `pack(3tk)`. By default, `fill="x"`, `anchor="center"`, `expand="yes"`.

14.6.3 Label

A label is a piece of text on your panel.

```
<label>
    <text>"This is a Label:"</text>
    <font>("Helvetica",20)</font>
</label>
```



14.6.4 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>
```

14.6.5 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>
```

Chapter 15

VCP

15.1 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 15.1.

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.

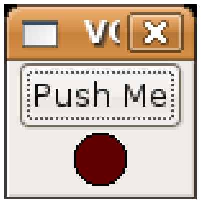


Figure 15.1: tiny.vcp on the screen

```
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!

15.2 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.

15.3 VCP Syntax

15.3.1 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 attributes format is

```
    name = value
```

The attribute names that are acceptable for each block depend on the block tag, and will be listed later.

Part VII

Advanced topics

Chapter 16

Kinematics in EMC2

16.1 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].

16.1.1 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).

16.2 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]

16.3 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 16.1).

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.

16.3.1 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).

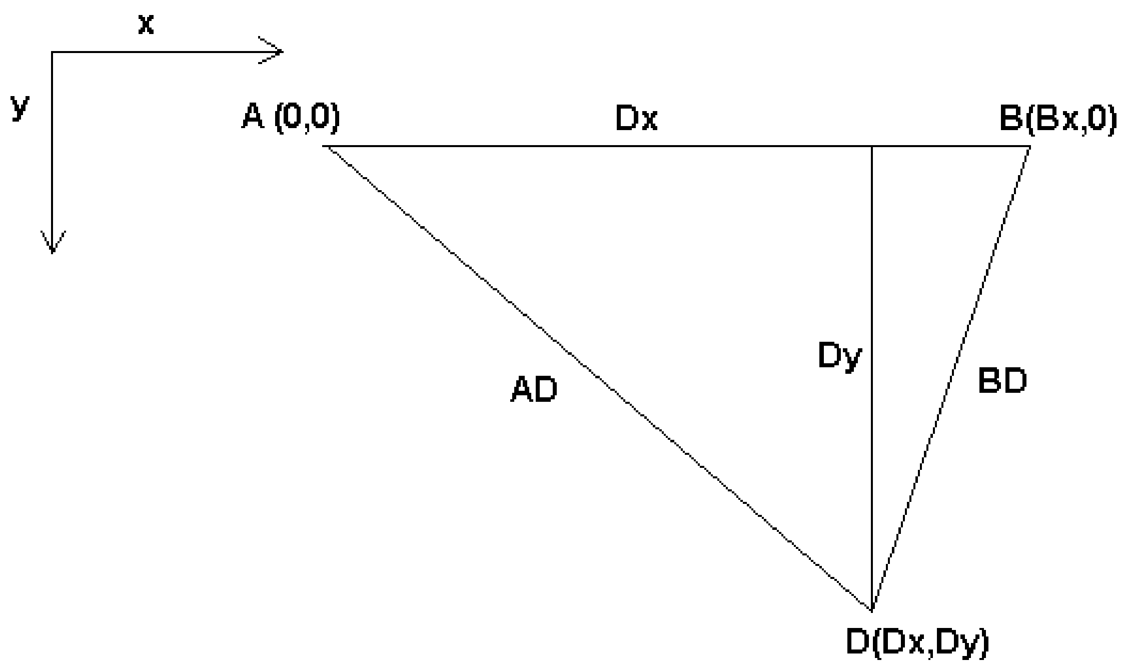
we can easily see that $AD^2 = x^2 + y^2$, likewise $BD^2 = (Bx - x)^2 + y^2$.

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 16.1: 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;
```

16.3.2 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;
```

16.4 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 16.3.1.

- `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 16.3.2.

- `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 VIII

Tuning

Chapter 17

Stepper Tuning

17.1 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.

17.1.1 Run a Latency Test

The new easy way to do a latency test is described in the Getting Started Guide.

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.

17.1.2 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.

17.1.3 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.

17.1.4 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.

17.1.5 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.

Chapter 18

PID Tuning

18.1 PID Controller

A proportional-integral-derivative controller (PID controller) is a common feedback loop component in industrial control systems.[1]

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).

18.1.1 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).

[1]This Subsection is taken from an much more extensive article found at http://en.wikipedia.org/wiki/PID_controller

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.

18.1.2 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:

18.1.2.0.0.1 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.

18.1.2.0.0.2 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.

18.1.2.0.0.3 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.

18.1.3 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.

18.1.3.0.0.4 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

18.1.3.0.0.5 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	$.6K_c$	$2/P_c$	$P \times P_c/8$

Part IX

Ladder 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.

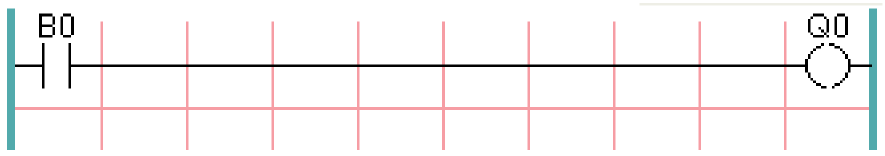
19.2 Example

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

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 Components

There are 2 components to ClassicLadder.

- The realtime module = classicladder_rt
- The userspace module (along with a GUI) = classicladder

20.3.1 Files

Typically classicladder components are placed in the custom.hal file if your working from a Stepconf generated configuration. These must not be placed in the custom_postgui.hal file or the Ladder Editor menu will be grayed out.

Ladder files (.clp) must not contain any blank spaces in the name.

20.3.2 Realtime Module

Loading the ClassicLadder realtime module (classicladder_rt) is possible from a halfile, or directly using a halcmd instruction. The first line loads real time the ClassicLadder module. The second line adds the function classicladder.0.refresh to the servo thread. This makes ClassicLadder update at the servo thread rate.

```
loadrt classicladder_rt
addf classicladder.0.refresh servo-thread
```

The speed of the thread that ClassicLadder is running in directly effects the responsiveness to inputs and outputs. If you can turn a switch on and off faster than ClassicLadder can notice it then you may need to speed up the thread. The fastest that ClassicLadder can refresh the rungs is one millisecond. You can put it in a faster thread but it will not update any faster. If you put it in a slower then one microsecond thread then ClassicLadder will update the rungs slower. The current refresh rate will be displayed on the section display, it is rounded to microseconds.

20.3.3 Variables

It is possible to configure the number of each type of ladder object while loading the classicladder realtime module. If you do not configure the number of ladder objects ClassicLadder will use the default values.

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)	10
Number of symbols	(numSymbols)	100
Number of S32 inputs	(numS32in)	0
Number of S32 outputs	(numS32out)	0

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
```

20.3.4 Loading the ClassicLadder user module

To load the user module:

loadusr classicladder

To load a ladder file (filename must not have any spaces):

loadusr classicladder myladder.clp

To load the user module without the GUI:

loadusr classicladder –nogui

To load the user module with the modbus port number for modbus server over ethernet:

loadusr classicladder –modbus_port=port

To load the user module with a config file:

loadusr classicladder –config=file

Sets up the ClassicLadder for modbus master over serial or ethernet.

loadusr classicladder –config=my modbusfile –nogui myladder.clp

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.

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.

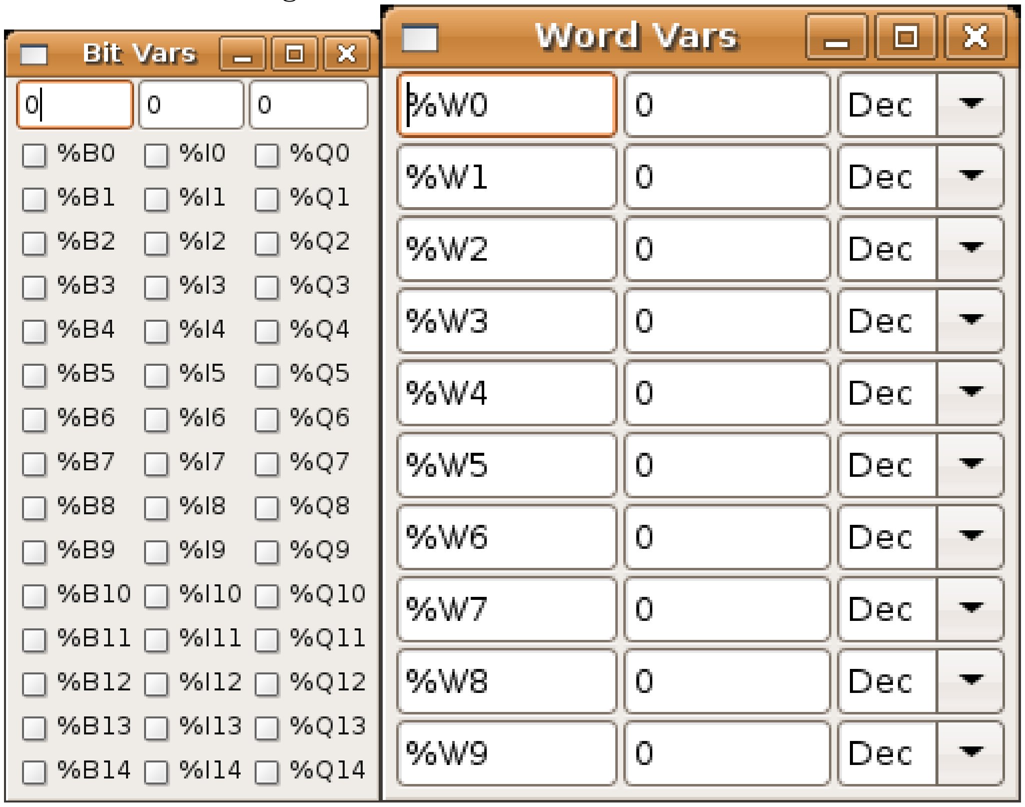
The three edit areas at the top allow you to select what 15 variable will be displayed in each column. For instance if there were 30 %I variable and you entered 10 at the top of the column, variable %I10 to %I25 would be displayed.

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.

Figure 20.1: ClassicLadder Var window



0	0	0
☐ %B0	☐ %I0	☐ %Q0
☐ %B1	☐ %I1	☐ %Q1
☐ %B2	☐ %I2	☐ %Q2
☐ %B3	☐ %I3	☐ %Q3
☐ %B4	☐ %I4	☐ %Q4
☐ %B5	☐ %I5	☐ %Q5
☐ %B6	☐ %I6	☐ %Q6
☐ %B7	☐ %I7	☐ %Q7
☐ %B8	☐ %I8	☐ %Q8
☐ %B9	☐ %I9	☐ %Q9
☐ %B10	☐ %I10	☐ %Q10
☐ %B11	☐ %I11	☐ %Q11
☐ %B12	☐ %I12	☐ %Q12
☐ %B13	☐ %I13	☐ %Q13
☐ %B14	☐ %I14	☐ %Q14

%W0	0	Dec
%W1	0	Dec
%W2	0	Dec
%W3	0	Dec
%W4	0	Dec
%W5	0	Dec
%W6	0	Dec
%W7	0	Dec
%W8	0	Dec
%W9	0	Dec

20.4.2 The Section Display window

Most of the buttons are self explanitory:

The config button is not used in EMC.

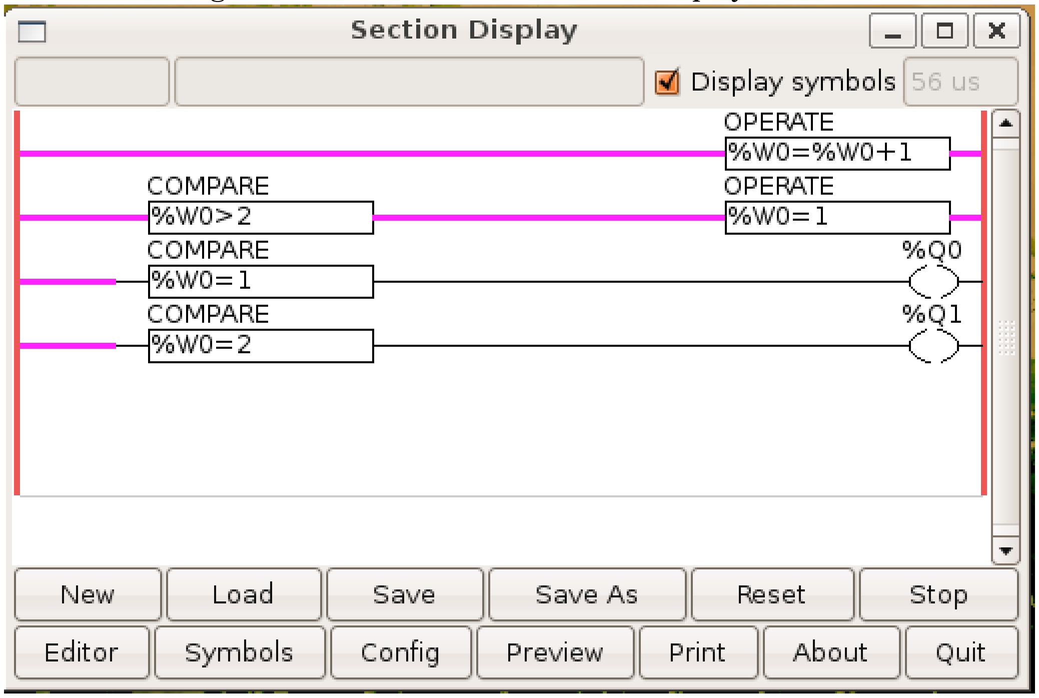
The symbols button will display an editable list of symbols for the variables (eg you can name the inputs, outputs, coils etc).

The symbols window will display the HAL signal names if present for %I, %Q and %W variables.

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

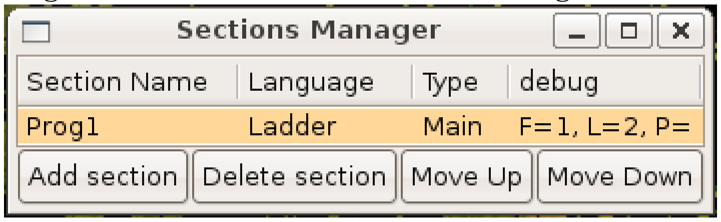
Figure 20.2: ClassicLadder Section Display window



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



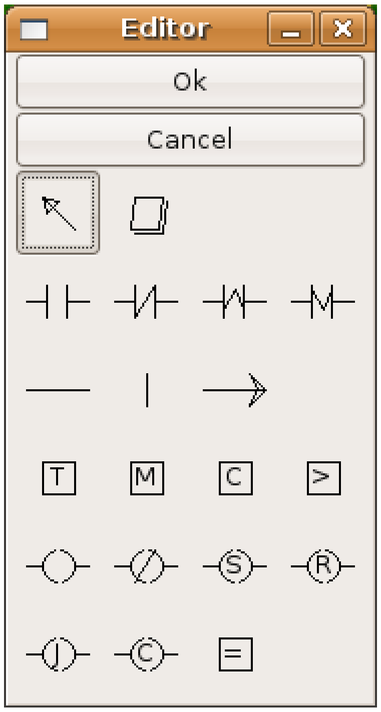
Section Name	Language	Type	debug
Prog1	Ladder	Main	F=1, L=2, P=

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

Figure 20.4: ClassicLadder Editor window



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.
- 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) w32in and w32out also use the word memory. See the s3220.8 section for details.

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.

20.8 w32 Pins

As of EMC 2.2.0 The realtime module can export HAL s32 unsigned integer in/out pins.

You must specify how many s32 in/out pins you want when loading the realtime module (default is 0 of each).

For example to load 5 s32in and 5 s32out pins:

```
loadrt classicladder_rt numS32in=5 numS32out=5
```

This loads the realtime module with the default number of each ladder objects except for it also specifies 5 s32 in pins and 5 s32 out pins.

Values of this type can range from -2,147,483,648 to 2,147,483,647. They would correspond to some of the %W variables. For instance if you requested 5 s32in pins and 5 s32out pins %W0-%W4 would correspond to HAL pin name classicladder.0.s32in.00 -04 %W5-%W9 would be HAL pin name classicladder.0.s32out-00 -04 and %W10-99 would be regular (memory) variables.

[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

If you request only s32out pins they would start at %W0 and regular words would start after them. so it maps %W variables like this. Number of s32in pins first (if any), then number of s32out pins second (if any) then regular memory variables.

For example if you add 2 s32in pins and 3 s32out pins the word relationship would be as shown in the following table.

Table 20.2: Using w32's

Word	Hal Pin
%W0	classicladder.0.s32in.00
%W1	classicladder.0.s32in.01
%W2	classicladder.0.s32out.00
%W3	classicladder.0.s32out.01
%W4	classicladder.0.s32out.02
%W5	start of regular words

20.9 Ladder Examples

20.9.1 External E-Stop

The External E-Stop example is in the /config/classicladder/cl-estop folder. It uses a pyVCP panel to simulate the external components.

To interface an external E-Stop to EMC and have the external E-Stop work together with the internal E-Stop requires a couple of connections through ClassicLadder.

First we have to open the E-Stop loop in the main hal file by commenting out by adding the pound sign as shown or removing the following lines.

```
# net estop-out <= iocontrol.0.user-enable-out
# net estop-out => iocontrol.0.emc-enable-in
```

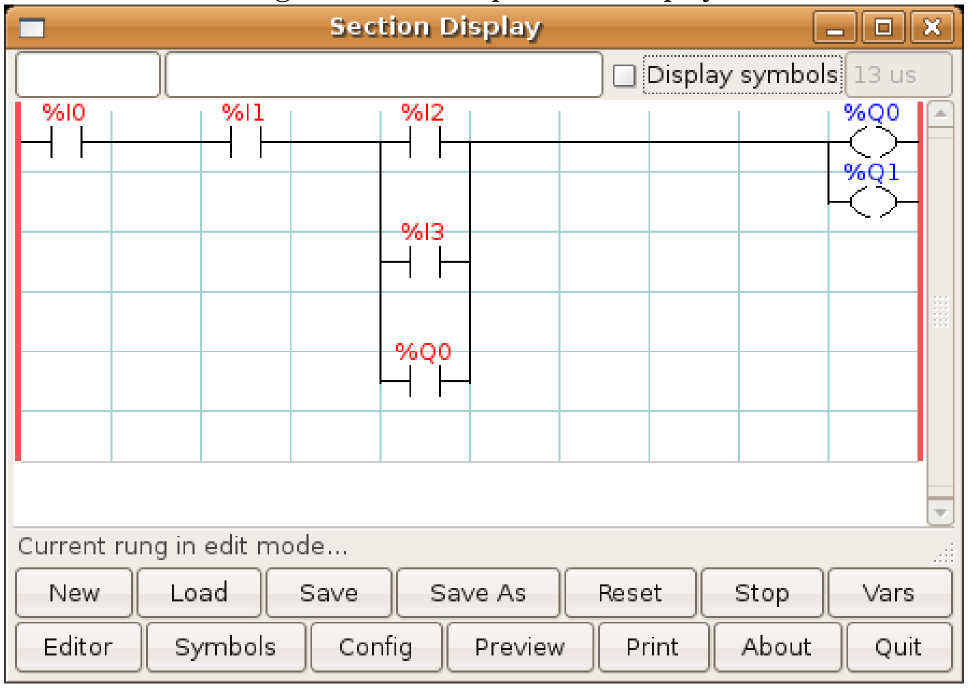
Next we add ClassicLadder to our main .hal file by adding these two lines:

```
loadrt classicladder_rt
addf classicladder.0.refresh servo-thread
```

ClassicLadder must be loaded in the main .hal file or the menu selection will be grayed out.

Next we run our config and build the ladder as shown here.

Figure 20.5: E-Stop Section Display



Current rung in edit mode...

After building the ladder select Save As and save the ladder as estop.clp

Now add the following line to your main .hal file.

```
# Load the ladder
loadusr classicladder --nogui estop.clp
```

I/O assignments

- %I0 = Input from the pyVCP panel simulated E-Stop (the checkbox)
- %I1 = Input from EMC's E-Stop
- %I2 = Input from EMC's E-Stop Reset Pulse
- %I3 = Input from the pyVCP panel reset button
- %Q0 = Ouput to EMC to enable
- %Q1 = Output to external driver board enable pin (use a N/C output if your board had a disable pin)

Next we add the following lines to the custom_postgui.hal file.

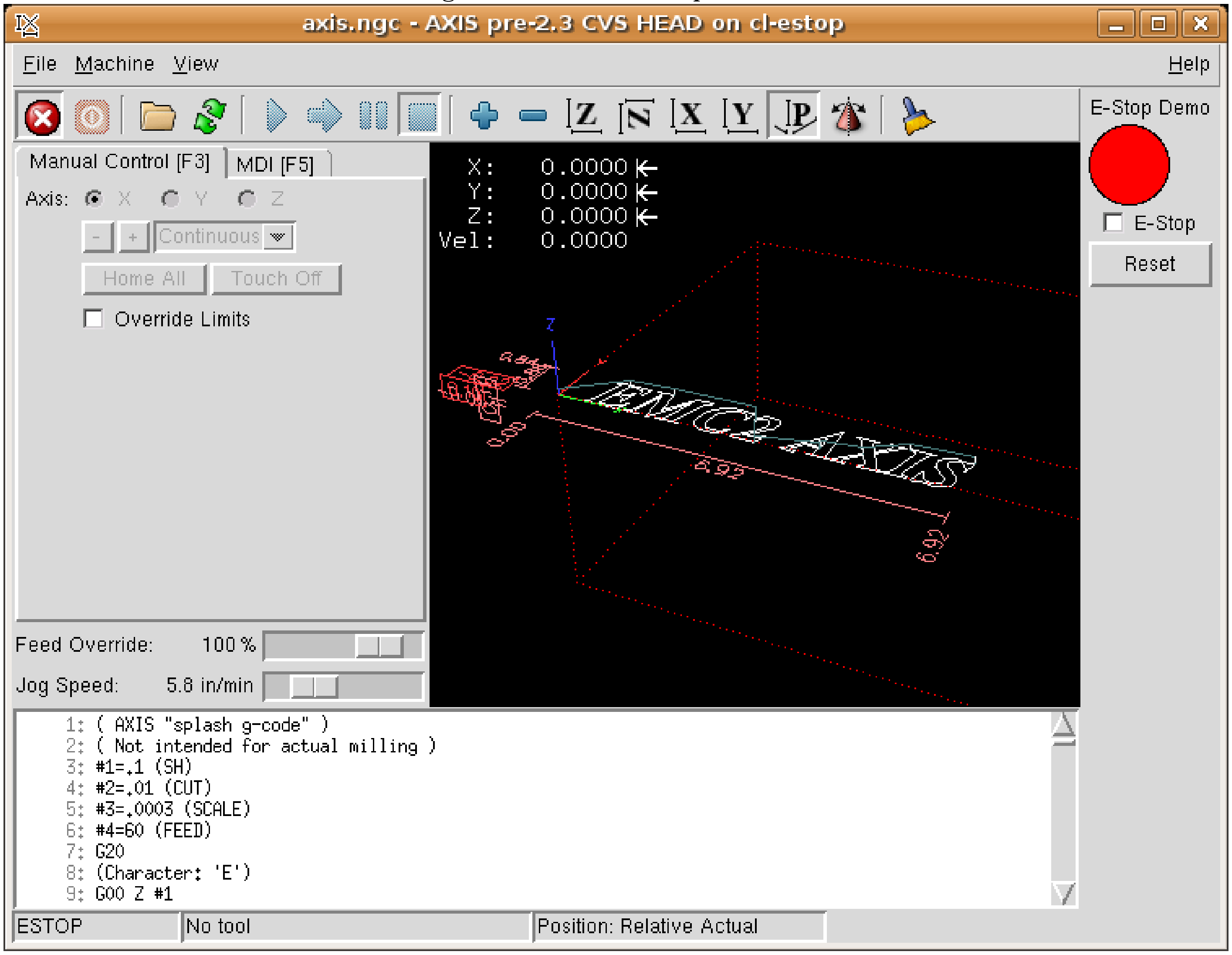
```
# E-Stop example using pyVCP buttons to simulate external components
# The pyVCP checkbutton simulates a normally closed external E-Stop
net ext-estop classicladder.0.in-00 <= pyvcp.py-estop
# Request E-Stop Enable from EMC
net estop-all-ok iocontrol.0.emc-enable-in <= classicladder.0.out-00
# Request E-Stop Enable from pyVCP or external source
net ext-estop-reset classicladder.0.in-03 <= pyvcp.py-reset
# This line resets the E-Stop from EMC
net emc-reset-estop iocontrol.0.user-request-enable => classicladder.0.in-02
# This line enables EMC to unlatch the E-Stop in classicladder
net emc-estop iocontrol.0.user-enable-out => classicladder.0.in-01
# This line turns on the green indicator when out of E-Stop
net estop-all-ok => pyvcp.py-es-status
```

Next we add the following lines to the panel.xml file. Note you have to open it with the text editor not the default html viewer.

```
<pyvcp>
<vbox>
<label><text>"E-Stop Demo"</text></label>
<led>
<halpin>"py-es-status"</halpin>
<size>50</size>
<on_color>"green"</on_color>
<off_color>"red"</off_color>
</led>
<checkbutton>
<halpin>"py-estop"</halpin>
<text>"E-Stop"</text>
</checkbutton>
</vbox>
<button>
<halpin>"py-reset"</halpin>
<text>"Reset"</text>
</button>
</pyvcp>
```

Now start up your config and it should look like this.

Figure 20.6: AXIS E-Stop



```
X:    0.0000
Y:    0.0000
Z:    0.0000
Vel:  0.0000
```

Feed Override: 100 %
Jog Speed: 5.8 in/min

```
( AXIS "splash g-code" )
( Not intended for actual milling )
#1=.1 (SH)
#2=.01 (CUT)
#3=.0003 (SCALE)
#4=60 (FEED)
G20
(Character: 'E')
G00 Z #1
```

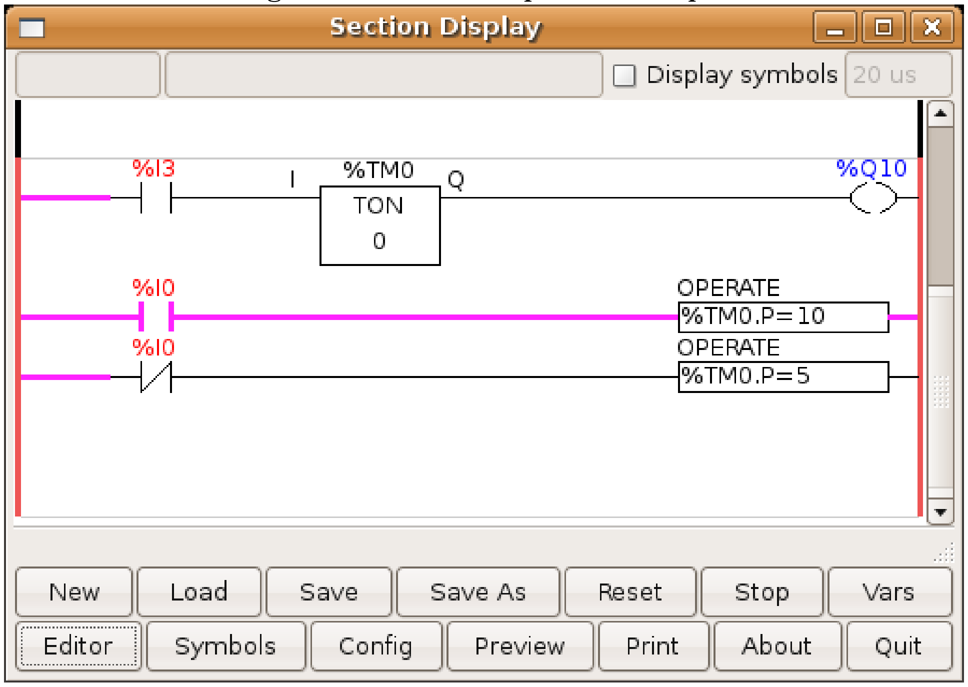
ESTOP | No tool | Position: Relative Actual

Note that in this example like in real life you must clear the remote E-Stop (simulated by the checkbox) before the AXIS E-Stop or the external Reset will put you in OFF mode. If the E-Stop in the AXIS screen was pressed you must press it again to clear it. You cannot reset from the external after you do an E-Stop in AXIS.

20.9.2 Timer/Operate Example

In this example we are using the Operate block to assign a value to the timer preset based on if an input is on or off.

Figure 20.7: Timer/Operate Example



In this case %I0 is true so the timer preset value is 10. If %I0 was false the timer preset would be 5.

20.9.3 Tool Turret

- This Example is not complete yet.

This is a program for one type of tool turret. The turret has a home switch at tool position 1 and another another switch to tell you when the turret is in a lockable position. To keep track of the actual tool number one must count how many positions past home you are. We will use ClassicLadder's counter block '$CO'.The counter is preset to 1 when RESET is true. The counter is increased by one on the rising edge of INDEX. We then 'COMPARE' the counter value (%C0.V) to the tool number we want (in the example only checks for tool 1 and 2 are shown). We also 'OPERATE' the counter value to a word variable (%W0) that (you can assume) is mapped on to a S32 out HAL pin so you can let some other HAL component know what the current tool number is. In the real world another S32 (in) pin would be used to get the requested tool number from EMC.You would have to load ClassicLadder's real time module specifying that you want S32 in and out pins. See 'loading options' above. [display turret sample]

20.9.4 Sequential Example

- This Example is not complete yet.

This is a sequential program when the program is first started step one is active then when %B0 is true then steps 2 and 3 are then active and step one is inactive then when %B1 and/or %B2 are true, step 4 and/or 5 are active and step 2 and/or 3 are inactive Then when either %B3 OR %B4 are true, step 6 is true and steps 4 and 5 are inactive then when %B5 is true step 1 is active and step 6 is inactive and it all starts again As shown the sequence has been: %B0 was true making step 2 and 3 active then %B1 became true (and still is-see the pink line through %B1) making step 4 active and step 2 inactive step 3 is active and waiting for %B2 to be true step 4 is active and is waiting for %B3 to be true WOW that was quite a mouth full!! [display sequential program]

Part X

Hardware Examples

Chapter 21

Spindle Speed Control

21.1 0-10v Spindle Speed

If your spindle is controlled by a VFD with a 0 to 10 volt signal and your using a DAC card like the m5i20 to output the control signal.

First you need to figure the scale of spindle speed to control signal. For this example the spindle top speed of 5000 RPM is equal to 10 volts. $10/5000 = 0.002$ so our scale factor is 0.002

We have to add a scale componet to the hal file to scale the motion.spindle-speed-out to the 0 to 10 needed by the VFD if your DAC card does not do scaling.

loadrt scale count=1

addf scale.0 servo-thread

setp scale.0.gain 0.002

net spindle-speed-scale motion.spindle-speed-out => scale.0.in

net spindle-speed-DAC scale.0.out => <your DAC pin name>

21.2 PWM Spindle Speed

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

loadrt pwmgen output_type=0

addf pwmgen.update servo-thread

addf pwmgen.make-pulses base-thread

net spindle-speed-cmd motion.spindle-speed-out => pwmgen.0.value

net spindle-on motion.spindle-on => pwmgen.0.enable

net spindle-pwm pwmgen.0.pwm => parport.0.pin-09-out

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.

21.3 Spindle Feedback

Add this section

21.4 Spindle Enable

If you need a spindle enable signal link your output pin to motion.spindle-on. To link these pins to a parallel port pin put something like the following in your .hal file making sure you pick the pin that is connected to your control device.

```
net spindle-enable motion.spindle-on => parport.0.pin-14-out
```

21.5 Spindle Direction

If you have direction control of your spindle the hal pins motion.spindle-forward and motion.spindle-reverse are controlled by M3 and M4. S must be set to a positive non zero value for M3/4 to turn on spindle motion.

To link these pins to a parallel port pin put something like the following in your .hal file making sure you pick the pin that is connected to your control device.

```
net spindle-fwd motion.spindle-forward -> parport.0.pin-16-out
```

```
net spindle-rev motion.spindle-reverse => parport.0.pin-17-out
```

Chapter 22

MPG Pendant

This example is to explain how to hook up the common MPG pendants found on the market place today. This example uses a MPG3 pendant and a C22 pendant interface card from CNC4PC and a second parallel port plugged into the PCI slot. This example gives you 3 axis with 3 step increments of 0.1, 0.01, 0.001.

22.1 Second Parallel Port

In this example we are using a second parallel port connected to the MPG Pendant that is plugged into a PCI slot. To find the address of your parallel port card open a terminal window and type

```
lspci -v
```

You will see something similar to this

```
0000:00:10.0 Communication controller: NetMos Technology PCI 1 port parallel adapter (rev 01)
Subsystem: LSI Logic / Symbios Logic: Unknown device 0010
Flags:      medium devsel, IRQ 11
I/O         ports at a800 [size=8]
I/O         ports at ac00 [size=8]
I/O         ports at b000 [size=8]
I/O         ports at b400 [size=8]
I/O         ports at b800 [size=8]
I/O         ports at bc00 [size=16]
```

In my case the address was the first one so I changed my .hal file from

```
loadrt hal_parport cfg=0x378
```

to

```
loadrt hal_parport cfg="0x378 0xa800 in"
```

note the double quotes surrounding the addresses.

and added

```
addf parport.1.read base-thread addf parport.1.write base-thread
```

so the parport will get read and written to.

22.2 Hook it up in HAL

In your custom.hal file or other .hal file add the following making sure you don't have mux4 or an encoder already in use. If you do just increase the counts and change the reference number. More information about mux4 can be found in Section(10) and encoder in Section(11.3).

```
# Jog Pendant

loadrt encoder num_chan=1

loadrt mux4 count=1

addf encoder.capture-position servo-thread

addf encoder.update-counters base-thread

addf mux4.0 servo-thread

setp encoder.0.x4-mode 0

setp mux4.0.in0 0.1

setp mux4.0.in1 0.01

setp mux4.0.in2 0.001

net scale1 mux4.0.sel0 <= parport.1.pin-09-in

net scale2 mux4.0.sel1 <= parport.1.pin-10-in

net pend-scale axis.0.jog-scale <= mux4.0.out

net pend-scale axis.1.jog-scale

net pend-scale axis.2.jog-scale

net mpg-a encoder.0.phase-A <= parport.1.pin-02-in

net mpg-b encoder.0.phase-B <= parport.1.pin-03-in

net mpg-x axis.0.jog-enable <= parport.1.pin-04-in

net mpg-y axis.1.jog-enable <= parport.1.pin-05-in

net mpg-z axis.2.jog-enable <= parport.1.pin-06-in

net pend-counts axis.0.jog-counts <= encoder.0.counts

net pend-counts axis.1.jog-counts

net pend-counts axis.2.jog-counts
```

Part XI

Diagnostics

Chapter 23

Steppers

If what you get is not what you expect if you pay attention you get some experience.

Diagnosing problems is best done by divide and conquer. By this I mean if you can remove 1/2 of the variables from the equation each time you will find the problem the fastest. In the real world this is not always the case but a good place to start usually.

23.1 Common Problems

23.1.1 Stepper Moves One Step

The most common reason in a new installation for the stepper not to move is the step and direction signals are backwards. If you press the jog foward and backward key and the stepper moves one step each time in the same direction there is your sign.

23.1.2 No Steppers Move

Many drives have an enable pin or need a charge pump to enable the output.

23.1.3 Distance Not Correct

If you command the axis to move a specific distance and it does not move that distance then your scale is wrong.

23.2 Error Messages

23.2.1 Following Error

The concept of a following error is funny when talking about stepper motors. Since they are an open loop system, there is no position feedback to let you know if you actually are out of range. EMC calculates if it can keep up with the motion called for and if not then it gives a following error. Following errors usually are the result of one of the following on stepper systems.

- FERROR to small

- MIN_FERROR to small
- MAX_VELOCITY to fast
- MAX_ACCELERATION to fast
- BASE_PERIOD set to long

Any of the above can cause the RT pulsing to not be able to keep up the requested step rate. This can happen if you didn't run the latency test long enough to get a good number to plug into the Stepconf Wizard or if you set the Maximum Velocity or Maximum Acceleration too high.

23.2.2 RTAPI Error

When you get this error:

```
RTAPI: ERROR: Unexpected realtime delay on task n
```

It is usually an indication that the BASE_PERIOD in the [EMCMOT] section of the ini file is set too low. You should run the Latency Test for an extended period of time to see if you have any delays that would cause this problem.

EMC2 tracks the number of CPU cycles between invocations of the real-time thread. If some element of your hardware is causing delays or your realtime threads are set too fast you will get this error.

NOTE: This error is only displayed once per session. If you had your BASE_PERIOD too low you could get hundreds of thousands of error messages per second if more than one was displayed.

23.3 Testing

23.3.1 Step Timing

If you are seeing an axis ending up in the wrong location over multiple moves, it is likely that you do not have the correct direction hold times or step timing for your stepper drivers. Each direction change may be losing a step or more. If the motors are stalling, it is also possible you have either the MAX_ACCELERATION or MAX_VELOCITY set too high for that axis.

The following program will test the Z axis configuration for proper setup. Copy the program to your emc2/nc_files directory and name it TestZ.ngc or similar. Zero your machine with Z = 0.000 at the table top. Load and run the program. It will make 200 moves back and forth from 0.5 to 1". If you have a configuration issue, you will find that the final position will not end up 0.500" that the axis window is showing. To test another axis just replace the Z with your axis in the G0 lines.

```
( test program to see if Z axis loses position )
( msg, test 1 of Z axis configuration )
G20 #1000=100 ( loop 100 times )
( this loop has delays after moves )
( tests acc and velocity settings )
o100 while [#1000]
G0 Z1.000
G4 P0.250
G0 Z0.500
```

```
G4 P0.250
#1000 = [#1000 - 1]
o100 endwhile
( msg, test 2 of Z axis configuration S to continue)
M1 (stop here)
#1000=100 ( loop 100 times )
( the next loop has no delays after moves )
( tests direction hold times on driver config and also max accel setting )
o101 while [#1000]
G0 Z1.000 .
G0 Z0.500
#1000 = [#1000 - 1]
o101 endwhile
( msg, Done...Z should be exactly .5" above table )
M2
```

Part XII

FAQ

Chapter 24

Linux FAQ

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

24.1 Automatic Login

When you install EMC2 with the Ubuntu LiveCD the default is to have to log in each time you turn the computer on. To enable automatic login go to System/Administration/Login Window. If it is a fresh install the Login Window might take a second or three to pop up. You will have to have your password that you used for the install to gain acess to the Login Window Preferences window. In the Security tab check off Enable Automatic Login and pick a user name from the list (that would be you).

24.2 Man Pages

Man pages are automatically generated manual pages in most cases. Man pages are usually available for most programs and commands in Linux.

To view a man page open up a terminal window by going to Applications, Accessories, Terminal. For example if you wanted to find out something about the find command in the terminal window type:

```
man find
```

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

24.3 List Modules

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

```
lsmod
```

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

```
lsmod > mymod.txt
```

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

24.4 Editing a Root File

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

24.4.1 The Command Line Way

Open up Applications, Accessories, Terminal.

In the terminal window type:

```
sudo gedit
```

Open the file with File, Open then edit

24.4.2 The GUI Way

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

24.5 Terminal Commands

24.5.1 Working Directory

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

```
pwd
```

24.5.2 Changing Directories

To move up one level in the terminal window type:

```
cd ..
```

To move up two levels in the terminal window type:

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

To move down to the emc2/configs subdirectory in the terminal window type:

```
cd emc2/configs
```

24.5.3 Listing files in a directory

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

```
dir
```

or

```
ls
```

24.5.4 Finding a File

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

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

For example to find all the .ini files in your EMC2 directory you first need to use the pwd command to find out the directory. Open a new terminal window and type:

```
pwd
```

might return the following result

```
/home/joe
```

With this information put the command together like this:

```
find /home/joe/emc2 -name *.ini -print
```

The -name is the name of the file your looking for and the -print tells it to print out the result to the terminal window. The *.ini tells find to return all files that have the .ini extension.

To find all the files in the directory named and all the subdirectories under that add the -L option to the find command like this:

```
find -L /home/joe/emc2 -name *.ini -print
```

24.5.5 Searching for Text

```
grep -i -r 'text to search for' *
```

To find all the files that contain the 'text to search for' in the current directory and all the subdirectories below the current while ignoring the case. The -i is for ignore case and the -r is for recursive (include all subdirectories in the search). The * is a wild card for search all files.

24.6 Hardware Problems

24.6.1 Hardware Info

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

```
lspci -v
```

24.6.2 Monitor Resolution

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

Instructions for fixing this are located here:

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

Part XIII

Appendices

Appendix A

Glossary

A listing of terms and what they mean. Some terms have a general meaning and several additional meanings for users, installers, and developers.

Acme Screw A type of lead-screw that uses an acme thread form. Acme threads have somewhat lower friction and wear than simple triangular threads, but ball-screws are lower yet. Most manual machine tools use acme lead-screws.

Axis One of the computer 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. or other mechanical motion driving system. It can result from nuts that are loose on leadscrews, slippage in belts, cable slack, "wind-up" in rotary couplings, and other places where the mechanical system is not "tight". Backlash will result in inaccurate motion, or in the case of motion caused by external forces (think cutting tool pulling on the work piece) the result can be broken cutting tools. This can happen because of the sudden increase in chip load on the cutter as the work piece is pulled across the backlash distance by the cutting tool.

Backlash Compensation - Any technique that attempts to reduce the effect of backlash without actually removing it from the mechanical system. This is typically done in software in the controller. This can correct the final resting place of the part in motion but fails to solve problems related to direction changes while in motion (think circular interpolation) and motion that is caused when external forces (think cutting tool pulling on the work piece) are the source of the motion.

Ball Screw A type of lead-screw that uses small hardened steel balls between the nut and screw to reduce friction. Ball-screws have very low friction and backlash, but are usually quite expensive.

Ball Nut A special nut designed for use with a ball-screw. It contains an internal passage to re-circulate the balls from one end of the screw to the other.

CNC Computer Numerical Control. The general term used to refer to computer control of machinery. Instead of a human operator turning cranks to move a cutting tool, CNC uses a computer and motors to move the tool, based on a part program.

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 mold making, 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 project. EMC is able to run a wide range of motion devices.

EMCIO The module within EMC that handles general purpose I/O, unrelated to the actual motion of the axes.

EMCMOT The module within EMC that handles the actual motion of the cutting tool. It runs as a real-time program and directly controls the motors.

Encoder A device to measure position. Usually a mechanical-optical device, which outputs a quadrature signal. The signal can be counted by special hardware, or directly by the parport with 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.

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 for a particular machine

Joint Coordinates These specify the angles between the individual joints of the machine. See also Kinematics

Jog Manually moving an axis of a machine. Jogging either moves the axis a fixed amount for each key-press, or moves the axis at a constant speed as long as you hold down the key.

kernel-space See real-time.

Kinematics The position relationship between world coordinates and joint coordinates of a machine. There are two types of kinematics. Forward kinematics is used to calculate world coordinates from joint coordinates. Inverse kinematics is used for exactly opposite purpose.Note that kinematics does not take into account, the forces, moments etc. on the machine. It is for positioning only.

Lead-screw An screw that is rotated by a motor to move a table or other part of a machine. Lead-screws are usually either ball-screws or acme screws, although conventional triangular threaded screws may be used where accuracy and long life are not as important as low cost.

Machine units The linear and angular units used for machine configuration. These units are used in the ini file. HAL pins and parameters are also generally in machine units.

MDI Manual Data Input. This is a mode of operation where the controller executes single lines of G-code as they are typed by the operator.

NIST National Institute of Standards and Technology. An agency of the Department of Commerce in the United States.

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.

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 or RTLINUX 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 `https://www.rtai.org/`, one of two real-time extensions for Linux that EMC can use to achieve real-time performance.

RTLINUX See `http://www.rtlinux.org`, one of two real-time extensions for Linux that EMC can use to achieve real-time performance.

RTAPI A portable interface to real-time operating systems including RTAI and RTLINUX

RS-274/NGC The formal name for the language used by EMC part programs.

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 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 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

Copyright Terms

GNU Free Documentation License

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.

Index