Integrator Manuelle V2.5, 2012-04-01

Contents

The LinuxCNC Team

Part I

LinuxCNC Introduction

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.

HINWEIS

Aufgrund der in jüngster Zeit zunehmend in das Interesse an anderen Übersetzungen, die EMC2 Team hat vor kurzem diese Bemühungen begonnen, eine zu liefern Deutsch-Übersetzung der EMC2 Dokumentation.
Wenn Sie möchten, einen Freiwilligen-Editor für die sein Deutsch-Übersetzung von EMC2, kontaktieren Sie uns bitte.

NOTICE

Because of a recent increase in interest in other translations, the EMC2 team has recently begun this effort to deliver a German Translation of the EMC2 documentation.
If you would like to be a volunteer editor for the German translation of EMC2, please contact us.

Chapter 1

Integrator Concepts

1.1 Stepper Systems

1.1.1 Base Period

BASE_PERIOD is the *heartbeat* of your LinuxCNC computer.[1] Every period, the software step generator decides if it is time for another step pulse. A shorter period will allow you to generate more pulses per second, within limits. But if you go too short, your computer will spend so much time generating step pulses that everything else will slow to a crawl, or maybe even lock up. Latency and stepper drive requirements affect the shortest period you can use.

Worst case latencies might only happen a few times a minute, and the odds of bad latency happening just as the motor is changing direction are low. So you can get very rare errors that ruin a part every once in a while and are impossible to troubleshoot.

The simplest way to avoid this problem is to choose a BASE_PERIOD that is the sum of the longest timing requirement of your drive, and the worst case latency of your computer. This is not always the best choice. For example, if you are running a drive with a 20 us direction signal hold time requirement, and your latency test said you have a maximum latency of 11 us , then if you set the BASE_PERIOD to 20+11 = 31 us you get a not-so-nice 32,258 steps per second in one mode and 16,129 steps per second in another mode.

The problem is with the 20 us hold time requirement. That plus the 11 us latency is what forces us to use a slow 31 us period. But the LinuxCNC software step generator has some parameters that let you increase the various times from one period to several. For example, if *steplen* [2] is changed from 1 to 2, then there will be two periods between the beginning and end of the step pulse. Likewise, if *dirhold* [3] is changed from 1 to 3, there will be at least three periods between the step pulse and a change of the direction pin.

If we can use *dirhold* to meet the 20 us hold time requirement, then the next longest time is the 4.5 us high time. Add the 11 us latency to the 4.5 us high time, and you get a minimum period of 15.5 us . When you try 15.5 us , you find that the computer is sluggish, so you settle on 16 us . If we leave *dirhold* at 1 (the default), then the minimum time between step and direction is the 16 us period minus the 11 us latency = 5 us , which is not enough. We need another 15 us . Since the period is 16 us , we need one more period. So we change *dirhold* from 1 to 2. Now the minimum time from the end of the step pulse to the changing direction pin is 5+16=21 us , and we don't have to worry about the drive stepping the wrong direction because of latency.

For more information on stepgen see the stepgen section of the HAL manual.

1.1.2 Step Timing

Step Timing and Step Space on some drives are different. In this case the Step point becomes important. If the drive steps on the falling edge then the output pin should be inverted.

---

[1] This section refers to using **stepgen**, LinuxCNC's built-in step generator. Some hardware devices have their own step generator and do not use LinuxCNC's built-in one. In that case, refer to your hardware manual.

[2] steplen refers to a parameter that adjusts the performance of LinuxCNC's built-in step generator, *stepgen*, which is a HAL component. This parameter adjusts the length of the step pulse itself. Keep reading, all will be explained eventually.

[3] dirhold refers to a parameter that adjusts the length of the direction hold time.

1.2 Servo Systems

1.2.1 Basic Operation

Servo systems are capable of greater speed and accuracy than equivalent stepper systems, but are more costly and complex. Unlike stepper systems, servo systems require some type of position feedback device, and must be adjusted or *tuned*, as they don't quite work right out of the box as a stepper system might. These differences exist because servos are a *closed loop* system, unlike stepper motors which are generally run *open loop*. What does *closed loop* mean? Let's look at a simplified diagram of how a servomotor system is connected.

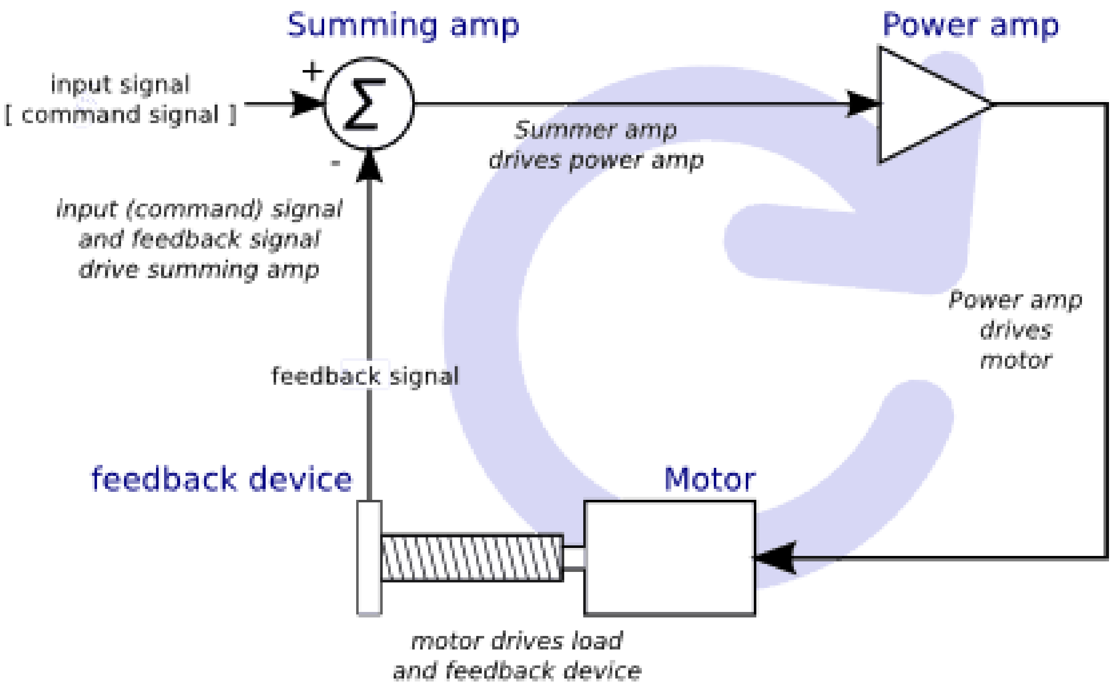


Figure 1.1: Servo Loop

This diagram shows that the input signal (and the feedback signal) drive the summing amplifier, the summing amplifier drives the power amplifier, the power amplifier drives the motor, the motor drives the load (and the feedback device), and the feedback device (and the input signal) drive the motor. This looks very much like a circle (a closed loop) where A controls B, B controls C, C controls D, and D controls A.

If you have not worked with servo systems before, this will no doubt seem a very strange idea at first, especially as compared to more normal electronic circuits, where the inputs proceed smoothly to the outputs, and never go back.[4] If *everything* controls *everything else*, how can that ever work, who's in charge? The answer is that LinuxCNC *can* control this system, but it has to do it by choosing one of several control methods. The control method that LinuxCNC uses, one of the simplest and best, is called PID.

PID stands for Proportional, Integral, and Derivative. The Proportional value determines the reaction to the current error, the Integral value determines the reaction based on the sum of recent errors, and the Derivative value determines the reaction based on the rate at which the error has been changing. They are three common mathematical techniques that are applied to the task of getting a working process to follow a set point. In the case of LinuxCNC the process we want to control is actual axis position and the set point is the commanded axis position.

[4] If it helps, the closest equivalent to this in the digital world are *state machines*, *sequential machines* and so forth, where what the outputs are doing *now* depends on what the inputs (and the outputs) were doing *before*. If it doesn't help, then nevermind.

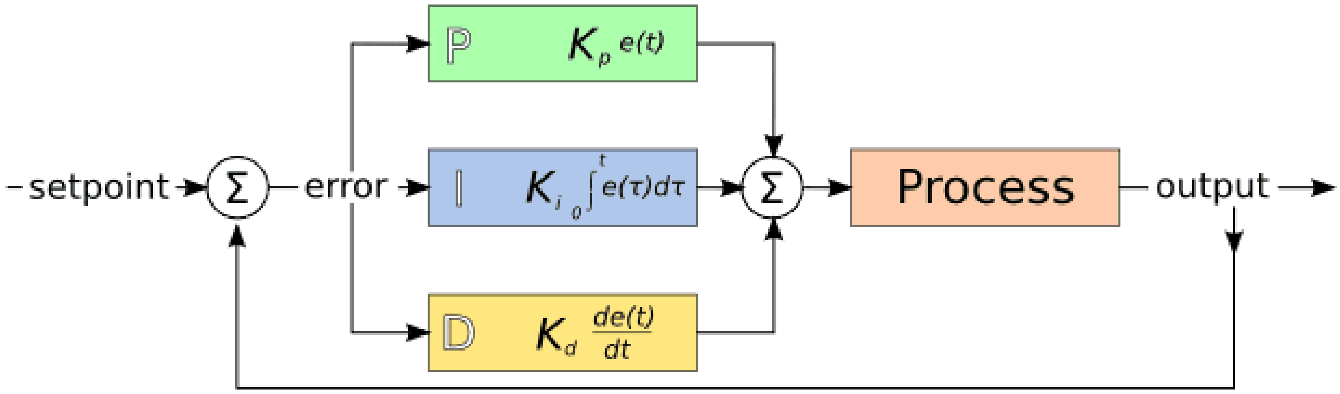


Figure 1.2: PID Loop

By *tuning* the three constants in the PID controller algorithm, the controller can provide control action designed for specific process requirements. The response of the controller can be described in terms of the responsiveness of the controller to an error, the degree to which the controller overshoots the set point and the degree of system oscillation.

1.2.2 Proportional term

The proportional term (sometimes called gain) makes a change to the output that is proportional to the current error value. A high proportional gain results in a large change in the output for a given change in the error. If the proportional gain is too high, the system can become unstable. In contrast, a small gain results in a small output response to a large input error. If the proportional gain is too low, the control action may be too small when responding to system disturbances.

In the absence of disturbances, pure proportional control will not settle at its target value, but will retain a steady state error that is a function of the proportional gain and the process gain. Despite the steady-state offset, both tuning theory and industrial practice indicate that it is the proportional term that should contribute the bulk of the output change.

1.2.3 Integral term

The contribution from the integral term (sometimes called reset) is proportional to both the magnitude of the error and the duration of the error. Summing the instantaneous error over time (integrating the error) gives the accumulated offset that should have been corrected previously. The accumulated error is then multiplied by the integral gain and added to the controller output.

The integral term (when added to the proportional term) accelerates the movement of the process towards set point and eliminates the residual steady-state error that occurs with a proportional only controller. However, since the integral term is responding to accumulated errors from the past, it can cause the present value to overshoot the set point value (cross over the set point and then create a deviation in the other direction).

1.2.4 Derivative term

The rate of change of the process error is calculated by determining the slope of the error over time (i.e. its first derivative with respect to time) and multiplying this rate of change by the derivative gain.

The derivative term slows the rate of change of the controller output and this effect is most noticeable close to the controller set point. Hence, derivative control is used to reduce the magnitude of the overshoot produced by the integral component and improve the combined controller-process stability.

1.2.5 Loop tuning

If the PID controller parameters (the gains of the proportional, integral and derivative terms) are chosen incorrectly, the controlled process input can be unstable, i.e. its output diverges, with or without oscillation, and is limited only by saturation or mechanical breakage. Tuning a control loop is the adjustment of its control parameters (gain/proportional band, integral gain/reset, derivative gain/rate) to the optimum values for the desired control response.

1.2.6 Manual tuning

A simple tuning method is to first set the I and D values to zero. Increase the P until the output of the loop oscillates, then the P should be set to be approximately half of that value for a *quarter amplitude decay* type response. Then increase I until any offset is correct in sufficient time for the process. However, too much I will cause instability. Finally, increase D, if required, until the loop is acceptably quick to reach its reference after a load disturbance. However, too much D will cause excessive response and overshoot. A fast PID loop tuning usually overshoots slightly to reach the set point more quickly; however, some systems cannot accept overshoot, in which case an *over-damped* closed-loop system is required, which will require a P setting significantly less than half that of the P setting causing oscillation.

1.3 RTAI

The Real Time Application Interface (RTAI) is used to provide the best Real Time (RT) performance. The RTAI patched kernel lets you write applications with strict timing constraints. RTAI gives you the ability to have things like software step generation which require precise timing.

1.3.1 ACPI

The Advanced Configuration and Power Interface (ACPI) has a lot of different functions, most of which interfere with RT performance (for example: power management, CPU power down, CPU frequency scaling, etc). The LinuxCNC kernel (and probably all RTAI-patched kernels) has ACPI disabled. ACPI also takes care of powering down the system after a shutdown has been started, and that’s why you might need to push the power button to completely turn off your computer. The RTAI group has been improving this in recent releases, so your LinuxCNC system may shut off by itself after all.

Part II

Configuration

Chapter 2

Latency Test

This test is the first test that should be performed on a PC to see if it is able to drive a CNC machine.

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

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

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

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

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

The best way to find out how well your PC will lrun LinuxCNC is to run the HAL latency test. To run the test, open a terminal window (In Ubuntu, from Applications → Accessories → Terminal) and run the following command:

```
latency-test
```

You should see something like this:

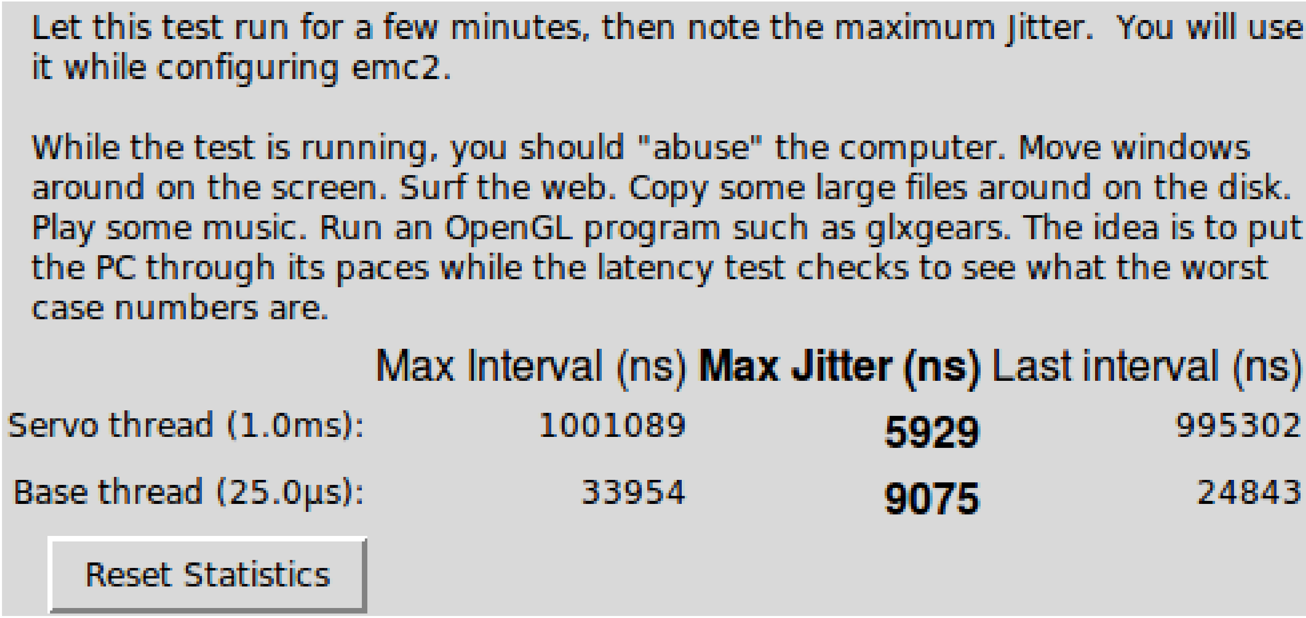


Figure 2.1: HAL Latency Test

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

Note
Do not run LinuxCNC or Stepconf while the latency test is running.

The important numbers are the *max jitter*. In the example above, that is 9075 nanoseconds, or 9.075 microseconds. Record this number, and enter it in Stepconf when it is requested.

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

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

Note that if you get high numbers, there may be ways to improve them. Another PC had very bad latency (several milliseconds) when using the onboard video. But a $5 used video card solved the problem.

Note
LinuxCNC does not require bleeding edge hardware.

For more information on stepper tuning see the Stepper Tuning Chapter.

2.1 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 LinuxCNC.

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

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

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 LinuxCNC 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 LinuxCNC.

Chapter 3

INI Configuration

3.1 The INI File Components

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

- comments
- sections
- variables

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

3.1.1 Comments

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

```
; This is my mill configuration file.
# I set it up on January 12, 2012
```

Comments can also be used to *turn off* a variable. This makes it easier to pick between different variables.

```
DISPLAY = axis
# DISPLAY = touchy
```

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

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

```
INCORRECT = value     # and a comment

# Correct Comment
CORRECT = value
```

3.1.2 Sections

Related parts of an ini file are separated into sections. A section name is enclosed in brackets like this *[THIS_SECTION]* The order of sections is unimportant. Sections begin at the section name and end at the next section name.

The following sections are used by LinuxCNC:

- *[EMC]* general information
- *[DISPLAY]* settings related to the graphical user interface
- *[FILTER]* settings input filter programs
- *[RS274NGC]* settings used by the g-code interpreter
- *[EMCMOT]* settings used by the real time motion controller
- *[TASK]* settings used by the task controller
- *[HAL]* specifies .hal files
- *[HALUI]* MDI commands used by HALUI
- *[TRAJ]* additional settings used by the real time motion controller
- *[AXIS_n]* individual axis variables
- *[EMCIO]* settings used by the I/O Controller

3.1.3 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.

Variables that are used by LinuxCNC must always use the section names and variable names as shown. In the following example the variable *MACHINE* is assigned the value *My Machine*.

Variable Example

```
MACHINE = My Machine
```

3.1.4 Custom Sections and Variables

Most sample configurations use custom sections and variables to put all of the settings into one location for convenience.

To use a custom section variable in your HAL file add the section and variable to the INI file.

Custom Section Example

```
[OFFSETS]
OFFSET_1 = 0.1234
```

To add a custom variable to a LinuxCNC section simply include the variable in that section.

Custom Variable Example

```
[AXIS_0]
TYPE = LINEAR
...
SCALE = 16000
```

To use the custom variables in your HAL file put the section and variable name in place of the value.

HAL Example

```
setp offset.1.offset [OFFSETS]OFFSET_1
setp stepgen.0.position-scale [AXIS_0]SCALE
```

Note
The value stored in the variable must match the type specificied by the component pin.

3.2 INI File Sections

3.2.1 [EMC] Section

- *VERSION = $Revision: 1.3 $* - The version number for the INI file. The value shown here looks odd because it is automatically updated when using the Revision Control System. It's a good idea to change this number each time you revise your file. If you want to edit this manually just change the number and leave the other tags alone.
- *MACHINE = My Controller* - This is the name of the controller, which is printed out at the top of most graphical interfaces. You can put whatever you want here as long as you make it a single line long.
- *DEBUG = 0* - Debug level 0 means no messages will be printed when LinuxCNC is run from a terminal. Debug flags are usually only useful to developers. See src/emc/nml_intf/emcglb.h for other settings.

3.2.2 [DISPLAY] Section

Different user interface programs use different options, and not every option is supported by every user interface. The main two interfaces for LinuxCNC are AXIS and Touchy. Axis is an interface for use with normal computer and monitor, Touchy is for use with touch screens. Descriptions of the interfaces are in the Interfaces section of the User Manual.

- *DISPLAY = axis* - The name of the user interface to use. Valid options may include: axis, touchy, keystick, mini, tklinuxcnc, 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 LinuxCNC. 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. This location is searched for the file name before the subroutine path and user M path if specified in the [RS274NGC] section.
- *INTRO_GRAPHIC = emc2.gif* - The image shown on the splash screen.
- *INTRO_TIME = 5* - The maximum time to show the splash screen, in seconds.
- *CYCLE_TIME = 0.05* - Cycle time in seconds that display will sleep between polls.

Note
The following [DISPLAY] items are for the AXIS interface only.

- *DEFAULT_LINEAR_VELOCITY = .25* - The default velocity for linear jogs, in , machine units per second.
- *MIN_VELOCITY = .01* - The approximate lowest value the jog slider.
- *MAX_LINEAR_VELOCITY = 1.0* - The maximum velocity for linear jogs, in machine units per second.
- *MIN_LINEAR_VELOCITY = .01* - The approximate lowest value the jog slider.
- *DEFAULT_ANGULAR_VELOCITY = .25* - The default velocity for angular jogs, in machine units per second.
- *MIN_ANGULAR_VELOCITY = .01* - The approximate lowest value the jog slider.
- *MAX_ANGULAR_VELOCITY = 1.0* - The maximum velocity for angular jogs, in machine units per second.
- *INCREMENTS = 1 mm, .5 in, …* - Defines the increments available for incremental jogs. The INCREMENTS can be used to override the default. The values can be decimal numbers (e.g., 0.1000) or fractional numbers (e.g., 1/16), optionally followed by a unit (cm, mm, um, inch, in or mil). If a unit is not specified the machine unit is assumed. Metric and imperial distances may be mixed: INCREMENTS = 1 inch, 1 mil, 1 cm, 1 mm, 1 um is a valid entry.
- *OPEN_FILE = /full/path/to/file.ngc* - The file to show in the preview plot when AXIS starts. Use a blank string "" and no file will be loaded at start up.
- *EDITOR = gedit* - The editor to use when selecting File > Edit to edit the gcode from the AXIS menu. This must be configured for this menu item to work. Another valid entry is gnome-terminal -e vim.
- *TOOL_EDITOR = tooledit* - The editor to use when editing the tool table (for example by selecting "File > Edit tool table…" in Axis). Other valid entries are "gedit", "gnome-terminal -e vim", and "gvim".
- *PYVCP = /filename.xml* - The PyVCP panel description file. See the PyVCP section for more information.
- *LATHE = 1* - This displays in lathe mode with a top view and with Radius and Diameter on the DRO.
- *GEOMETRY = XYZABCUVW* - Controls the preview and backplot of rotary motion. This item consists of a sequence of axis letters, optionally preceded by a "-" sign. Only axes defined in *[TRAJ]AXES* should be used. This sequence specifies the order in which the effect of each axis is applied, with a "-" inverting the sense of the rotation. The proper GEOMETRY string depends on the machine configuration and the kinematics used to control it. The example string GEOMETRY=XYZBCUVW is for a 5-axis machine where kinematics causes UVW to move in the coordinate system of the tool and XYZ to move in the coordinate system of the material. The order of the letters is important, because it expresses the order in which the different transformations are applied. For example rotating around C then B is different than rotating around B then C. Geometry has no effect without a rotary axis.
- *ARCDIVISION = 64* - Set the quality of preview of arcs. Arcs are previewed by dividing them into a number of straight lines; a semicircle is divided into **ARCDIVISION** parts. Larger values give a more accurate preview, but take longer to load and result in a more sluggish display. Smaller values give a less accurate preview, but take less time to load and may result in a faster display. The default value of 64 means a circle of up to 3 inches will be displayed to within 1 mil (.03%).[1]
- *MDI_HISTORY_FILE =* - The name of a local MDI history file. If this is not specified Axis will save the MDI history in **.axis_mdi_history** in the user's home directory. This is useful if you have multiple configurations on one computer.

Note
The following [DISPLAY] item is used by the TKLinuxCNC interface only.

- *HELP_FILE = tklinucnc.txt* - Path to help file.

[1] In LinuxCNC 2.4 and earlier, the default value was 128.

3.2.3 [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 LinuxCNC when Run.

- *PROGRAM_EXTENSION = .extension Description*

If your post processor outputs files in all caps you might want to add the following line:

- *PROGRAM_EXTENSION = .NGC XYZ Post Processor*

The following lines add support for the image-to-gcode converter included with LinuxCNC:

- *PROGRAM_EXTENSION = .png,.gif,.jpg Greyscale Depth Image*
  - *png = image-to-gcode*
  - *gif = image-to-gcode*
  - *jpg = 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 more g-code generators are on the LinuxCNC 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.

3.2.4 [RS274NGC] Section

- *PARAMETER_FILE = myfile.var* - The file located in the same directory as the ini file which contains the parameters used by the interpreter (saved between runs).
- *RS274NGC_STARTUP_CODE = G01 G17 G20 G40 G49 G64 P0.001 G80 G90 G92 G94 G97 G98* - A string of NC codes that the interpreter is initialized with. This is not a substitute for specifying modal g-codes at the top of each ngc file, because the modal codes of machines differ, and may be changed by g-code interpreted earlier in the session.
- *SUBROUTINE_PATH = ncsubroutines:/tmp/testsubs:lathesubs:millsubs* - Specifies a colon (:) separated list of up to 10 directories to be searched when single-file subroutines are specified in gcode. These directories are searched after searching [DISPLAY]PROGRAM_PREFIX (if it is specified) and before searching [WIZARD]WIZARD_ROOT (if specified). The paths are searched in the order that they are listed. The first matching subroutine file found in the search is used. Directories are specified relative to the current directory for the inifile or as absolute paths. The list must contain no intervening whitespace.
- *USER_M_PATH = myfuncs:/tmp/mcodes:experimentalmcodes* - Specifies a list of colon (:) separated directories for user defined functions. Directories are specified relative to the current directory for the inifile or as absolute paths. The list must contain no intervening whitespace.

  A search is made for each possible user defined function, typically (M100-M199). The search order is:

1. [DISPLAY]PROGRAM_PREFIX (if specified)
2. If [DISPLAY]PROGRAM_PREFIX is not specified, search the default location: nc_files
3. Then search each directory in the list [RS274NGC]USER_M_PATH
   The first executable M1xx found in the search is used for each M1xx.

- *USER_DEFINED_FUNCTION_MAX_DIRS=5*. The maximum number of directories defined at compile time.

Note
[WIZARD]WIZARD_ROOT is a valid search path but the Wizard has not been fully implemented and the results of using it are unpredictable.

3.2.5 [EMCMOT] Section

This section is a custom section and is not used by LinuxCNC directly. Most configurations use values from this section to load the motion controller. For more information on the motion controller see the Motion Section.

- *EMCMOT = motmod* - the motion controller name is typically used here.
- *BASE_PERIOD = 50000* - the *Base* task period in nanoseconds.
- *SERVO_PERIOD = 1000000* - This is the "Servo" task period in nanoseconds.
- *TRAJ_PERIOD = 100000* - This is the *Trajectory Planner* task period in nanoseconds.

3.2.6 [TASK] Section

- *TASK = milltask* - Specifies the name of the *task* executable. The *task* executable does various things, such as communicate with the UIs over NML, communicate with the realtime motion planner over non-HAL shared memory, and interpret gcode. Currently there is only one task executable that makes sense for 99.9% of users, milltask.
- *CYCLE_TIME = 0.010* - The period, in seconds, at which TASK will run. This parameter affects the polling interval when waiting for motion to complete, when executing a pause instruction, and when accepting a command from a user interface. There is usually no need to change this number.

3.2.7 [HAL] section

- *TWOPASS=ON* - Use two pass processing for loading HAL comps. With TWOPASS processing, all [HAL]HALFILES are first read and multiple appearances of loadrt directives for each moduleb are accumulated. No hal commands are executed in this initial pass.
- *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 LinuxCNC is exiting. Depending on the hardware drivers used, this may make it possible to set outputs to defined values when LinuxCNC is exited normally. However, because there is no guarantee this file will be executed (for instance, in the case of a computer crash) it is not a replacement for a proper physical e-stop chain or other protections against software failure.
- *POSTGUI_HALFILE = example2.hal* - (Only with the TOUCHY and AXIS GUI) Execute *example2.hal* after the GUI has created its HAL pins. See section pyVCP with Axis Section for more information.
- *HALUI = halui* - adds the HAL user interface pins. For more information see the HAL User Interface chapter.

3.2.8 [HALUI] section

- *MDI_COMMAND = G53 G0 X0 Y0 Z0* - An MDI command can be executed by using halui.mdi-command-00. Increment the number for each command listed in the [HALUI] section.

3.2.9 [TRAJ] Section

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

- *COORDINATES = X Y Z* - The names of the axes being controlled. Only X, Y, Z, A, B, C, U, V, W are valid. Only axes named in *COORDINATES* are accepted in g-code. This has no effect on the mapping from G-code axis names (X- Y- Z-) to joint numbers—for *trivial kinematics*, X is always joint 0, A is always joint 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.
- *JOINTS = 3* - (This config variable is used by the Axis GUI only, not by the trajectory planner in the motion controller.) Specifies the number of joints (motors) in the system. For example, an XYZ machine with a single motor for each axis has 3 joints. A gantry machine with one motor on each of two of the axes, and two motors on the third axis, has 4 joints.
- *HOME = 0 0 0* - Coordinates of the homed position of each axis. Again for a fourth axis you will need 0 0 0 0. This value is only used for machines with nontrivial kinematics. On machines with trivial kinematics this value is ignored.
- *LINEAR_UNITS = <units>* - Specifies the *machine units* for linear axes. Possible choices are (in, inch, imperial, metric, mm). This does not affect the linear units in NC code (the G20 and G21 words do this).
- *ANGULAR_UNITS = <units>* - Specifies the *machine units* for rotational axes. Possible choices are *deg*, *degree* (360 per circle), *rad*, *radian* (2pi per circle), *grad*, or *gon* (400 per circle). This does not affect the angular units of NC code. In RS274NGC, A-, B- and C- words are always expressed in degrees.
- *DEFAULT_VELOCITY = 0.0167* - The initial rate for jogs of linear axes, in machine units per second. The value shown in *Axis* equals machine units per minute.
- *DEFAULT_ACCELERATION = 2.0* - In machines with nontrivial kinematics, the acceleration used for "teleop" (Cartesian space) jogs, in *machine units* per second per second.
- *MAX_VELOCITY = 5.0* - The maximum velocity for any axis or coordinated move, in *machine units* per second. The value shown equals 300 units per minute.
- *MAX_ACCELERATION = 20.0* - The maximum acceleration for any axis or coordinated axis move, in *machine units* per second per second.
- *POSITION_FILE = position.txt* - If set to a non-empty value, the joint positions are stored between runs in this file. This allows the machine to start with the same coordinates it had on shutdown. This assumes there was no movement of the machine while powered off. If unset, joint positions are not stored and will begin at 0 each time LinuxCNC is started. This can help on smaller machines without home switches.
- *NO_FORCE_HOMING = 1* - The default behavior is for LinuxCNC to force the user to home the machine before any MDI command or a program is run. Normally jogging only is allowed before homing. Setting NO_FORCE_HOMING = 1 allows the user to make MDI moves and run programs without homing the machine first. Interfaces without homing ability will need to have this option set to 1.

Warning
Using this will allow the machine to go beyond the soft limits while in operation. It is not generally desirable to allow this.

3.2.10 [AXIS_<num>] Section

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

Typically (but not always):

- AXIS_0 = X
- AXIS_1 = Y
- AXIS_2 = Z
- AXIS_3 = A
- AXIS_4 = B
- AXIS_5 = C
- AXIS_6 = U
- AXIS_7 = V
- AXIS_8 = W
- *TYPE = LINEAR* - The type of axes, either LINEAR or ANGULAR.
- *WRAPPED_ROTARY = 1* - When this is set to 1 for an ANGULAR axis the axis will move 0-359.999 degrees. Positive Numbers will move the axis in a positive direction and negative numbers will move the axis in the negative direction.
- *LOCKING_INDEXER = 1* - When this is set to 1 a G0 move for this axis will initiate an unlock with axis.N.unlock pin then wait for the axis.N.is-unlocked pin then move the axis at the rapid rate for that axis. After the move the axis.N.unlock will be false and motion will wait for axis.N.is-unlocked to go false. Moving with other axes is not allowed when moving a locked rotary axis.
- *UNITS = INCH* - If specified, this setting overrides the related [TRAJ] UNITS setting. (e.g., [TRAJ]LINEAR_UNITS if the TYPE of this axis is LINEAR, [TRAJ]ANGULAR_UNITS if the TYPE of this axis is ANGULAR)
- *MAX_VELOCITY = 1.2* - Maximum velocity for this axis in machine units per second.
- *MAX_ACCELERATION = 20.0* - Maximum acceleration for this axis in machine units per second squared.
- *BACKLASH = 0.0000* - Backlash in machine units. Backlash compensation value can be used to make up for small deficiencies in the hardware used to drive an axis. If backlash is added to an axis and you are using steppers the STEPGEN_MAXACCEL must be increased to 1.5 to 2 times the MAX_ACCELERATION for the axis.
- *COMP_FILE = file.extension* - A file holding compensation structure for the axis. The file could be named xscrew.comp, for example, for the X axis. File names are case sensitive and can contain letters and/or numbers. The values are triplets per line separated by a space. The first value is nominal (where it should be). The second and third values depend on the setting of COMP_FILE_TYPE. Currently the limit inside LinuxCNC is for 256 triplets per axis. If COMP_FILE is specified, BACKLASH is ignored. Compensation file values are in machine units.
- *COMP_FILE_TYPE = 0 or 1* -
  - *If 0:* The second and third values specify the forward position (where the axis is while traveling forward) and the reverse position (where the axis is while traveling reverse), positions which correspond to the nominal position.’
  - *If 1:* The second and third values specify the forward trim (how far from nominal while traveling forward) and the reverse trim (how far from nominal while traveling in reverse), positions which correspond to the nominal position.

    ```
    Example triplet with COMP_FILE_TYPE = 0: 1.00 1.01 0.99 +
    Example triplet with COMP_FILE_TYPE = 1: 1.00 0.01 -0.01
    ```
- *MIN_LIMIT = -1000* - The minimum limit (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 LinuxCNC Servo Axis Calibration utility program (in the AXIS interface menu Machine/Calibration and in TkLinuxCNC 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.

3.2.10.1 Homing

These parameters are Homing related, for a better explanation read the Homing Configuration Chapter.

- *HOME = 0.0* - The position that the joint will go to upon completion of the homing sequence.
- *HOME_OFFSET = 0.0* - The axis position of the home switch or index pulse, in machine units. In other words, when the home point is found during the homing process, this is the value or position that should be assigned to the home point, or initialized at the home point.
- *HOME_SEARCH_VEL = 0.0* - Initial homing velocity in machine units per second. Sign denotes direction of travel. A value of zero means assume that the current location is the home position for the machine. If your machine has no home switches you will want to leave this value at zero.
- *HOME_LATCH_VEL = 0.0* - Homing velocity in machine units per second to the home switch latch position. Sign denotes direction of travel.
- *HOME_FINAL_VEL = 0.0* - Velocity in machine units per second from home latch position to home position. If left at 0 or not included in the axis rapid velocity is used. Must be a positive number.
- *HOME_USE_INDEX = NO* - If the encoder used for this axis has an index pulse, and the motion card has provision for this signal you may set it to yes. When it is yes, it will affect the kind of home pattern used. Currently, you can't home to index with steppers unless you're using stepgen in velocity mode and PID.
- *HOME_IGNORE_LIMITS = NO* - Some machines use a single switch per axis as a home switch and limit switch. This variable should be set to YES if the machine configured this way. When set to YES the limit switch for this axis only is ignored when homing.
- *HOME_IS_SHARED = <n>* - If the home input is shared by more than one axis set <n> to 1 to prevent homing from starting if the one of the shared switches is already closed. Set <n> to 0 to permit homing if a switch is closed.
- *HOME_SEQUENCE = <n>* - Used to define the "Home All" sequence. <n> starts at 0 and no numbers may be skipped. If left out or set to -1 the joint will not be homed by the "Home All" function. More than one axis can be homed at the same time.
- *VOLATILE_HOME = 0* - When enabled (set to 1) this joint will be unhomed if the Machine Power is off or if E-Stop is on. This is useful if your machine has home switches and does not have position feedback such as a step and direction driven machine.

> **Warning**
> The following are custom INI file entries that you may find in a sample INI file or a wizard generated file. These are not used by the LinuxCNC software. They are only there to put all the settings in one place. For more information on custom INI file entries see the Custom Sections and Variables subsection.

The following items might be used by a PID component and the assumption is that the output is volts.

- *DEADBAND = 0.000015* - How close is close enough to consider the motor in position, in machine units. This is often set to a distance equivalent to 1, 1.5, 2, or 3 encoder counts, but there are no strict rules. Looser (larger) settings allow less servo *hunting* at the expense of lower accuracy. Tighter (smaller) settings attempt higher accuracy at the expense of more servo *hunting*. Is it really more accurate if it's also more uncertain? As a general rule, it's good to avoid, or at least limit, servo *hunting* if you can.

Be careful about going below 1 encoder count, since you may create a condition where there is no place that your servo is happy. This can go beyond *hunting* (slow) to *nervous* (rapid), and even to *squealing* which is easy to confuse with oscillation caused by improper tuning. Better to be a count or two loose here at first, until you've been through *gross tuning* at least.

Example of calculating machine units per encoder pulse to use in deciding DEADBAND value:

$$\frac{1\,revolution}{1000\,lines} X \frac{1\,line}{4\,pulse/line} X \frac{0.2\,units}{1\,revolution} = \frac{0.200\,units}{4000\,pulses} = \frac{0.00005\,units}{1\,pulse}$$

- *BIAS = 0.000* - This is used by hm2-servo and some others. Bias is a constant amount that is added to the output. In most cases it should be left at zero. However, it can sometimes be useful to compensate for offsets in servo amplifiers, or to balance the weight of an object that moves vertically. bias is turned off when the PID loop is disabled, just like all other components of the output.
- *P = 50* - The proportional gain for the axis servo. This value multiplies the error between commanded and actual position in machine units, resulting in a contribution to the computed voltage for the motor amplifier. The units on the P gain are volts per machine unit, e.g., $\frac{volts}{unit}$
- *I = 0* - The integral gain for the axis servo. The value multiplies the cumulative error between commanded and actual position in machine units, resulting in a contribution to the computed voltage for the motor amplifier. The units on the I gain are volts per machine unit second, e.g., $\frac{volts}{unit\,second}$
- *D = 0* - The derivative gain for the axis servo. The value multiplies the difference between the current and previous errors, resulting in a contribution to the computed voltage for the motor amplifier. The units on the D gain are volts per machine unit per second, e.g., $\frac{volts}{unit\,second}$
- *FF0 = 0* - The 0th order feed forward gain. This number is multiplied by the commanded position, resulting in a contribution to the computed voltage for the motor amplifier. The units on the FF0 gain are volts per machine unit, e.g., $\frac{volts}{unit}$
- *FF1 = 0* - The 1st order feed forward gain. This number is multiplied by the change in commanded position per second, resulting in a contribution to the computed voltage for the motor amplifier. The units on the FF1 gain are volts per machine unit per second, e.g., $\frac{volts}{unit\,second}$

- *FF2 = 0* - The 2nd order feed forward gain. This number is multiplied by the change in commanded position per second per second, resulting in a contribution to the computed voltage for the motor amplifier. The units on the FF2 gain are volts per machine unit per second per second, e.g., $\frac{volts}{unit\ second^2}$
- *OUTPUT_SCALE = 1.000* -
- *OUTPUT_OFFSET = 0.000* - These two values are the scale and offset factors for the axis output to the motor amplifiers. The second value (offset) is subtracted from the computed output (in volts), and divided by the first value (scale factor), before being written to the D/A converters. The units on the scale value are in true volts per DAC output volts. The units on the offset value are in volts. These can be used to linearize a DAC. Specifically, when writing outputs, the LinuxCNC first converts the desired output in quasi-SI units to raw actuator values, e.g., volts for an amplifier DAC. This scaling looks like: $raw = \frac{output - offset}{scale}$

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

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

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

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

To do this, follow this procedure.

1. Build a calibration table for the output, driving the DAC with a desired voltage and measuring the result.
2. Do a least-squares linear fit to get coefficients a, b such that $measured = a * raw + b$
3. Note that we want raw output such that our measured result is identical to the commanded output. This means
   a. $command = a * raw + b$
   b. $raw = (command - b)/a$
4. As a result, the a and b coefficients from the linear fit can be used as the scale and offset for the controller directly.

See the following table for an example of voltage measurements.

Table 3.1: Output Voltage Measurements

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

- *MAX_OUTPUT = 10* - The maximum value for the output of the PID compensation that is written to the motor amplifier, in volts. The computed output value is clamped to this limit. The limit is applied before scaling to raw output units. The value is applied symmetrically to both the plus and the minus side.
- *INPUT_SCALE = 20000* - in Sample configs
- *ENCODER_SCALE = 20000* - in PNCconf built configs Specifies the number of pulses that corresponds to a move of one machine unit as set in the [TRAJ] section. For a linear axis one machine unit will be equal to the setting of LINEAR_UNITS. For an angular axis one unit is equal to the setting in ANGULAR_UNITS. A second number, if specified, is ignored. For example, on a 2000 counts per rev encoder, and 10 revs/inch gearing, and desired units of inch, we have:

$$input\,scale = 2000\,\frac{counts}{rev} * 10\,\frac{rev}{inch} = 20000\,\frac{counts}{inch}$$

Warning
The following are custom INI file entries that you may find in a sample INI file or a wizard generated file. These are not used by the LinuxCNC software. They are only there to put all the settings in one place. For more information on custom INI file entries see the Custom Sections and Variables subsection.

The following items might be used by a stepgen component.

- *SCALE = 4000* - in Sample configs
- *STEP_SCALE = 4000* - in PNCconf built configs Specifies the number of pulses that corresponds to a move of one machine unit as set in the [TRAJ] section. For stepper systems, this is the number of step pulses issued per machine unit. For a linear axis one machine unit will be equal to the setting of LINEAR_UNITS. For an angular axis one unit is equal to the setting in ANGULAR_UNITS. For servo systems, this is the number of feedback pulses per machine unit. A second number, if specified, is ignored.

For example, on a 1.8 degree stepper motor with half-stepping, and 10 revs/inch gearing, and desired machine units of inch, we have:

$$input\,scale = \frac{2\,steps}{1.8\,degrees} * 360\,\frac{degree}{rev} * 10\,\frac{rev}{inch} = 4000\,\frac{steps}{inch}$$

- *ENCODER_SCALE = 20000* (Optionally used in PNCconf built configs) - Specifies the number of pulses that corresponds to a move of one machine unit as set in the [TRAJ] section. For a linear axis one machine unit will be equal to the setting of LINEAR_UNITS. For an angular axis one unit is equal to the setting in ANGULAR_UNITS. A second number, if specified, is ignored. For example, on a 2000 counts per rev encoder, and 10 revs/inch gearing, and desired units of inch, we have:

$$input\,scale = 2000\,\frac{counts}{rev} * 10\,\frac{rev}{inch} = 20000\,\frac{counts}{inch}$$

- *STEPGEN_MAXACCEL = 21.0* - Acceleration limit for the step generator. This should be 1% to 10% larger than the axis MAX_ACCELERATION. This value improves the tuning of stepgen's "position loop". If you have added backlash compensation to an axis then this should be 1.5 to 2 times greater than MAX_ACCELERATION.
- *STEPGEN_MAXVEL = 1.4* - Older configuration files have a velocity limit for the step generator as well. If specified, it should also be 1% to 10% larger than the axis MAX_VELOCITY. Subsequent testing has shown that use of STEPGEN_MAXVEL does not improve the tuning of stepgen's position loop.

3.2.11 [EMCIO] Section

- *CYCLE_TIME = 0.100* - The period, in seconds, at which EMCIO will run. Making it 0.0 or a negative number will tell EMCIO not to sleep at all. There is usually no need to change this number.
- *TOOL_TABLE = tool.tbl* - The file which contains tool information, described in the User Manual.
- *TOOL_CHANGE_POSITION = 0 0 2* - Specifies the XYZ location to move to when performing a tool change if three digits are used. Specifies the XYZABC location when 6 digits are used. Specifies the XYZABCUVW location when 9 digits are used. Tool Changes can be combined. For example if you combine the quill up with change position you can move the Z first then the X and Y.
- *TOOL_CHANGE_WITH_SPINDLE_ON = 1* - The spindle will be left on during the tool change when the value is 1. Useful for lathes or machines where the material is in the spindle, not the tool.
- *TOOL_CHANGE_QUILL_UP = 1* - The Z axis will be moved to machine zero prior to the tool change when the value is 1. This is the same as issuing a G0 G53 Z0.
- *TOOL_CHANGE_AT_G30 = 1* - The machine is moved to reference point defined by parameters 5181-5186 for G30 if the value is 1. For more information on G30 and Parameters see the G Code Manual.
- *RANDOM_TOOLCHANGER = 1* - This is for machines that cannot place the tool back into the pocket it came from. For example, machines that exchange the tool in the active pocket with the tool in the spindle.

Chapter 4

Homing Configuration

4.1 Overview

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

4.2 Homing Sequence

There are four possible homing sequences, along with the associated configuration parameters as shown in the following table. For a more detailed description of what each configuration parameter does, see the following section.

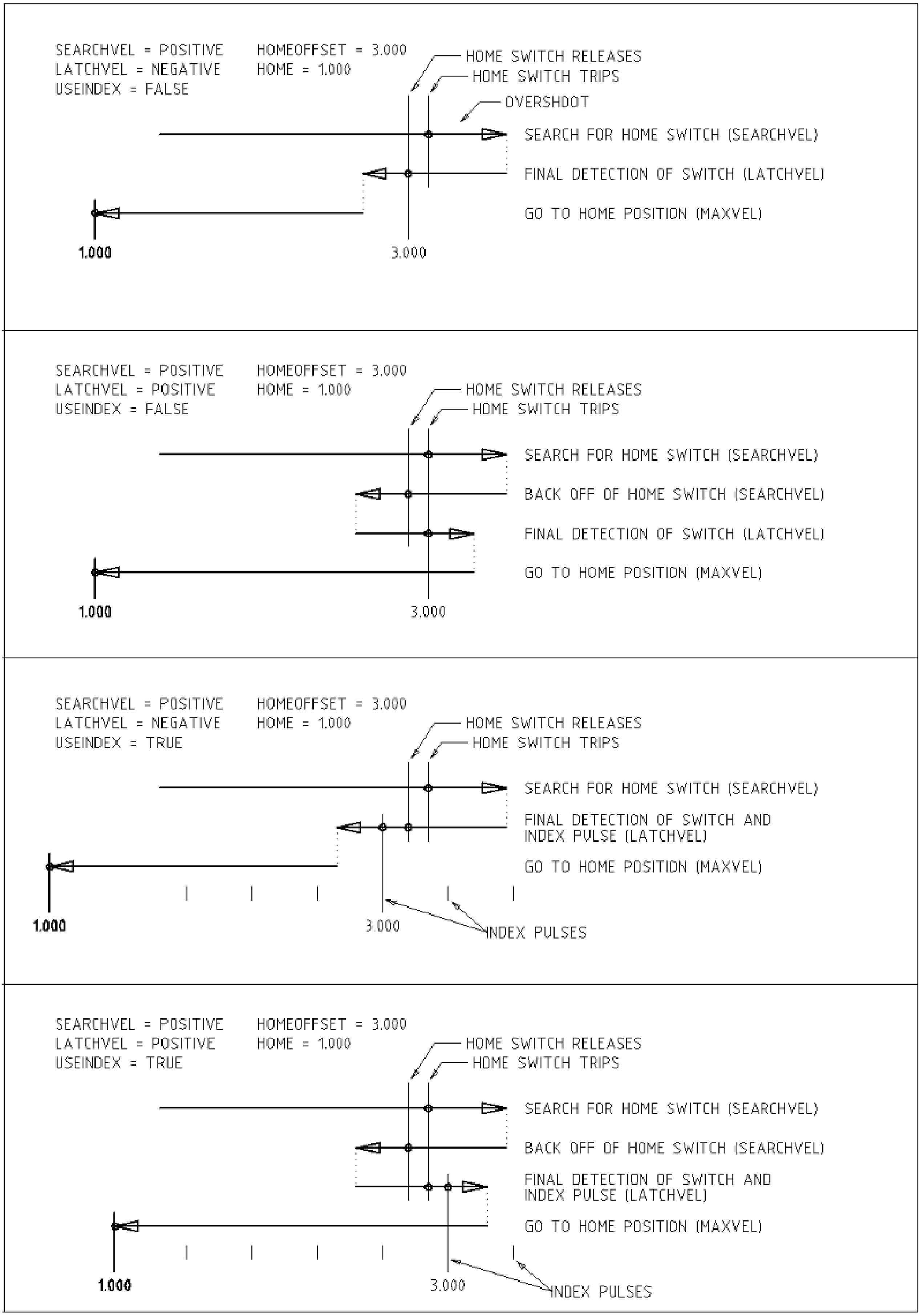


Figure 4.1: Homing Sequences

4.3 Configuration

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

Homing Type	SEARCH_VEL	LATCH_VEL	USE_INDEX
Immediate	0	0	NO
Index-only	0	nonzero	YES
Switch-only	nonzero	nonzero	NO
Switch and Index	nonzero	nonzero	YES

Note
Any other combinations may result in an error.

4.3.1 HOME_SEARCH_VEL

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

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

4.3.2 HOME_LATCH_VEL

Specifies the speed and direction that LinuxCNC uses when it makes its final accurate determination of the home switch (if present) and index pulse location (if present). It will usually be slower than the search velocity to maximize accuracy. If HOME_SEARCH_VEL and HOME_LATCH_VEL have the same sign, then the latch phase is done while moving in the same direction as the search phase. (In that case, LinuxCNC first backs off the switch, before moving towards it again at the latch velocity.) If HOME_SEARCH_VEL and HOME_LATCH_VEL have opposite signs, the latch phase is done while moving in the opposite direction from the search phase. That means LinuxCNC will latch the first pulse after it moves off the switch. If HOME_SEARCH_VEL is zero (meaning there is no home switch), and this parameter is nonzero, LinuxCNC goes ahead to the index pulse search. If HOME_SEARCH_VEL is non-zero and this parameter is zero, it is an error and the homing operation will fail. The default value is zero.

4.3.3 HOME_FINAL_VEL

It specifies the speed that LinuxCNC uses when it makes its move from HOME_OFFSET to the HOME position. If the HOME_FINAL_VEL is missing from the ini file, then the maximum joint speed is used to make this move. The value must be a positive number.

4.3.4 HOME_IGNORE_LIMITS

Can hold the values YES / NO. This flag determines whether LinuxCNC will ignore the limit switch input for this axis while homing. 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, LinuxCNC needs to ignore that limit during homing. The default value for this parameter is NO. To use only one input for all homing and limits you will have to block the limits of the axis not homing with axis that is homing in HAL.

4.3.5 HOME_USE_INDEX

Specifies whether or not there is an index pulse. If the flag is true (HOME_USE_INDEX = YES), LinuxCNC will latch on the rising edge of the index pulse. If false, LinuxCNC will latch on either the rising or falling edge of the home switch (depending on the signs of HOME_SEARCH_VEL and HOME_LATCH_VEL). The default value is NO.

4.3.6 HOME_OFFSET

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

4.3.7 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, LinuxCNC makes a move to HOME as the final step of the homing process. The default value is zero. Note that even if this parameter is the same as HOME_OFFSET, the joint will slightly overshoot the latched position as it stops. Therefore there will always be a small move at this time (unless HOME_SEARCH_VEL is zero, and the entire search/latch stage was skipped). This final move will be made at the joint's maximum velocity. Since the joint 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. [1]

4.3.8 HOME_IS_SHARED

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

4.3.9 HOME_SEQUENCE

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

4.3.10 VOLATILE_HOME

If this setting is true, this axis becomes unhomed whenever the machine transitions into the OFF state. This is appropriate for any axis that does not maintain position when the axis drive is off. Some stepper drives, especially microstep drives, may need this.

4.3.11 LOCKING_INDEXER

If this axis is a locking rotary indexer, it will unlock before homing, and lock afterward.

[1] The distinction between *home_offset* and *home* is that *home_offset* first establishes the scale location on the machine by applying the *home_offset* value to the location where home was found, and then *home* says where the joint should move to on that scale.

Chapter 5

Lathe Configuration

5.1 Default Plane

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

```
RS274NGC_STARTUP_CODE = G18
```

The above can be overwritten in a g code program so always set important things in the preamble of the g code file.

5.2 INI Settings

The following .ini settings are needed for lathe mode in Axis in addition to or replacing normal settings in the .ini file.

```
[DISPLAY]
DISPLAY = axis
LATHE = 1
[TRAJ]
AXES = 3
COORDINATES = X Z
[AXIS_0]
...
[AXIS_2]
...
```

Chapter 6

HAL TCL Files

The halcmd language excels in specifiying components and connections but offers no computational capabilities. As a result, ini files are limited in the clarity and brevity that is possible with higher level languages.

The haltcl facility provides a means to use tcl scripting and its features for computation, looping, branching, procedures, etc. in ini files. To use this functionality, you use the tcl language and the extension .tcl for halfiles.

The .tcl extension is understood by the main script (linuxcnc) that processes ini files. Haltcl files are identified in the the HAL section of ini files (just like .hal files).

Example

```
[HAL]
HALFILE = conventional_file.hal
HALFILE = tcl_based_file.tcl
```

With appropriate care, .hal and .tcl files can be intermixed.

6.1 Compatibility

The halcmd language used in .hal files has a simple syntax that is actually a subset of the more powerful general-purpose tcl scripting language.

6.2 Haltcl Commands

Haltcl files use the tcl scripting language augmented with the specific commands of the LinuxCNC hardware abstraction layer (HAL). The hal-specific commands are.

```
addf, alias,
delf, delsig,
getp, gets
ptype,
stype,
help,
linkpp, linkps, linksp, list, loadrt, loadusr, lock,
net, newsig,
save, setp, sets, show, source, start, status, stop,
unalias, unlinkp, unload, unloadrt, unloadusr, unlock,
waitusr
```

Two special cases occur for the *gets* and *list* commands due to conflicts with tcl builtin commands. For haltcl, these commands must be preceded with the keyword *hal*.

```
halcmd   haltcl
------   ------
gets     hal gets
list     hal list
```

6.3 Haltcl Inifile variables

Inifile variables are accessible by both halcmd and haltcl but with differing syntax.

LinuxCNC ini files use SECTION and ITEM specifiers to identify configuration items.

```
[SECTION_A]
ITEM1 = value_1
ITEM2 = value_2
...
[SECTION_B]
...
```

The ini file values are accessible by text substition in .hal files using the form.

```
[SECTION]ITEM
```

The same ini file values are accessible in .tcl files using the form of a tcl global array variable.

```
$::SECTION(ITEM)
```

For example, an ini file item like:

```
[AXIS_0]
MAX_VELOCITY = 4
```

is expressed as [AXIS_0]MAX_VELOCITY in .hal files for halcmd and as $::AXIS_0(MAX_VELOCITY) in .tcl files for haltcl

6.4 Converting .hal files to .tcl files

Existing .hal files can be converted to .tcl files by hand editing to adapt to the differences mentioned above. The process can be automated with scripts that convert using these substitutions.

```
[SECTION]ITEM ---> $::SECTION(ITEM)
gets           ---> hal gets
list           ---> hal list
```

6.5 Haltcl Notes

In haltcl, the value argument for the *sets* and *setp* commands is implicitly treated as an expression in the tcl language.

Example

```
# set gain to convert deg/sec to units/min for AXIS_0 radius
setp scale.0.gain 6.28/360.0*$::AXIS_0(radius)*60.0
```

Whitespace in the bare expression is not allowed, use quotes for that:

```
setp scale.0.gain "6.28 / 360.0 * $::AXIS_0(radius) * 60.0"
```

In other contexts, such as *loadrt*, you must explicitly use the tcl expr command ([expr {}]) for computational expressions.

Example

```
loadrt motion base_period=[expr {500000000/$::TRAJ(MAX_PULSE_RATE)}]
```

6.6 Haltcl Examples

Consider the topic of *stepgen headroom*. Software stepgen runs best with an acceleration constraint that is "a bit higher" than the one used by the motion planner. So, when using halcmd files, we force inifiles to have a manually calculated value.

```
[AXIS_0]
MAXACCEL = 10.0
STEPGEN_MAXACCEL = 10.5
```

With haltcl, you can use tcl commands to do the computation and eliminate the STEPGEN_MAXACCEL inifile item altogether.

```
setp stepgen.0.maxaccel $::AXIS_0(MAXACCEL)*1.05
```

Another haltcl feature is looping and testing. For example, many simulator configurations use "core_sim.hal" or "core_sim9.hal" hal files. These differ because of the requirement to connect more or fewer axes. The following haltcl code would work for any combination of axes in a trivkins machine.

```
# Create position, velocity and acceleration signals for each axis
set ddt 0
foreach axis {X Y Z A B C U V W} axno {0 1 2 3 4 5 6 7 8} {
  # 'list pin' returns an empty list if the pin doesn't exist
  if {[hal list pin axis.$axno.motor-pos-cmd] == {}} {
    continue
  }
  net ${axis}pos axis.$axno.motor-pos-cmd => axis.$axno.motor-pos-fb \
                                          => ddt.$ddt.in
  net ${axis}vel <= ddt.$ddt.out
  incr ddt
  net ${axis}vel => ddt.$ddt.in
  net ${axis}acc <= ddt.$ddt.out
  incr ddt
}
puts [show sig *vel]
puts [show sig *acc]
```

6.7 Haltcl Interactive

The halrun command recognizes haltcl files. With the -T option, haltcl can be run interaactively as a tcl interpreter. This capability is useful for testing and for standalone hal applications.

Example

```
$ halrun -T haltclfile.tcl
```

6.8 Haltcl Distribution Examples (sim)

The configs/sim/simtcl directory includes an ini file that uses a .tcl file to demonstrate a haltcl configuration in conjunction with the usage of twopass processing. The example shows the use of tcl procedures, looping, the use of comments, and output to the terminal.

Chapter 7

Core Components

See also the man pages *motion(9)*.

7.1 Motion

These pins and parameters are created by the realtime *motmod* module. This module provides a HAL interface for LinuxCNC's motion planner. Basically motmod takes in a list of waypoints and generates a nice blended and constraint-limited stream of joint positions to be fed to the motor drives.

Optionally the number of Digital I/O is set with num_dio. The number of Analog I/O is set with num_aio. The default is 4 each.

Pin names starting with *axis* are actually joint values, but the pins and parameters are still called *axis.N*. They are read and updated by the motion-controller function.

Motion is loaded with the motmod command. A kins should be loaded before motion.

```
loadrt motmod [base_period_nsec=period] [servo_period_nsec=period]
[traj_period_nsec=period] [num_joints=[0-9] ([num_dio=1-64] num_aio=1-16])]
```

7.1.1 Options

If the number of digital I/O needed is more than the default of 4 you can add up to 64 digital I/O by using the num_dio option when loading motmod.

If the number of analog I/O needed is more than the default of 4 you can add up to 16 analog I/O by using the num_aio option when loading motmod.

7.1.2 Pins

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

- *motion.adaptive-feed* - (float, in) When adaptive feed is enabled with *M52 P1* , the commanded velocity is multiplied by this value. This effect is multiplicative with the NML-level feed override value and *motion.feed-hold*.
- *motion.analog-in-00* - (float, in) These pins (00, 01, 02, 03 or more if configured) are controlled by M66.
- *motion.analog-out-00* - (float, out) These pins (00, 01, 02, 03 or more if configured) are controlled by M67 or M68.
- *motion.coord-error* - (bit, out) TRUE when motion has encountered an error, such as exceeding a soft limit
- *motion.coord-mode* - (bit, out) TRUE when motion is in *coordinated mode*, as opposed to *teleop mode*

- *motion.current-vel* - (float, out) The current tool velocity in user units per second.
- *motion.digital-in-00* - (bit, in) These pins (00, 01, 02, 03 or more if configured) are controlled by M62-65.
- *motion.digital-out-00* - (bit, out) These pins (00, 01, 02, 03 or more if configured) are controlled by the *M62-65*.
- *motion.distance-to-go* - (float,out) The distance remaining in the current move.
- *motion.enable* - (bit, in) If this bit is driven FALSE, motion stops, the machine is placed in the *machine off* state, and a message is displayed for the operator. For normal motion, drive this bit TRUE.
- *motion.feed-hold* - (bit, in) When Feed Stop Control is enabled with *M53 P1*, and this bit is TRUE, the feed rate is set to 0.
- *motion.in-position* - (bit, out) TRUE if the machine is in position.
- *motion.motion-enabled* - (bit, out) TRUE when in *machine on* state.
- *motion.on-soft-limit* - (bit, out) TRUE when the machine is on a soft limit.
- *motion.probe-input* - (bit, in) *G38.x* uses the value on this pin to determine when the probe has made contact. TRUE for probe contact closed (touching), FALSE for probe contact open.
- *motion.program-line* - (s32, out) The current program line while executing. Zero if not running or between lines while single stepping.
- *motion.requested-vel* - (float, out) The current requested velocity in user units per second from the F=n setting in the G Code file. No feed overrides or any other adjustments are applied to this pin.
- *motion.spindle-at-speed* - (bit, in) Motion will pause until this pin is TRUE, under the following conditions: before the first feed move after each spindle start or speed change; before the start of every chain of spindle-synchronized moves; and if in CSS mode, at every rapid to feed transition. This input can be used to ensure that the spindle is up to speed before starting a cut, or that a lathe spindle in CSS mode has slowed down after a large to small facing pass before starting the next pass at the large diameter. Many VFDs have an *at speed* output. Otherwise, it is easy to generate this signal with the *HAL near* component, by comparing requested and actual spindle speeds.
- *motion.spindle-brake* - (bit, out) TRUE when the spindle brake should be applied.
- *motion.spindle-forward* - (bit, out) TRUE when the spindle should rotate forward.
- *motion.spindle-index-enable* - (bit, I/O) For correct operation of spindle synchronized moves, this pin must be hooked to the index-enable pin of the spindle encoder.
- *motion.spindle-on* - (bit, out) TRUE when spindle should rotate.
- *motion.spindle-reverse* - (bit, out) TRUE when the spindle should rotate backward
- *motion.spindle-revs* - (float, in) For correct operation of spindle synchronized moves, this signal must be hooked to the position pin of the spindle encoder. The spindle encoder position should be scaled such that spindle-revs increases by 1.0 for each rotation of the spindle in the clockwise (*M3*) direction.
- *motion.spindle-speed-in* - (float, in) Feedback of actual spindle speed in rotations per second. This is used by feed-per-revolution motion (*G95*). If your spindle encoder driver does not have a velocity output, you can generate a suitable one by sending the spindle position through a *ddt* component. If you do not have a spindle encoder, you can loop back *motion.spindle-speed-out-rps*.
- *motion.spindle-speed-out* - (float, out) Commanded spindle speed in rotations per minute. Positive for spindle forward (*M3*), negative for spindle reverse (*M4*).
- *motion.spindle-speed-out-rps* - (float, out) Commanded spindle speed in rotations per second. Positive for spindle forward (*M3*), negative for spindle reverse (*M4*).
- *motion.teleop-mode* - (bit, out) TRUE when motion is in *teleop mode*, as opposed to *coordinated mode*
- *motion.tooloffset.x* … *motion.tooloffset.w* - (float, out, one per axis) shows the tool offset in effect; it could come from the tool table (*G43* active), or it could come from the gcode (*G43.1* active)

7.1.3 Parameters

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

- *motion-command-handler.time* - (s32, RO)
- *motion-command-handler.tmax* - (s32, RW)
- *motion-controller.time* - (s32, RO)
- *motion-controller.tmax* - (s32, RW)
- *motion.debug-bit-0* - (bit, RO) This is used for debugging purposes.
- *motion.debug-bit-1* - (bit, RO) This is used for debugging purposes.
- *motion.debug-float-0* - (float, RO) This is used for debugging purposes.
- *motion.debug-float-1* - (float, RO) This is used for debugging purposes.
- *motion.debug-float-2* - (float, RO) This is used for debugging purposes.
- *motion.debug-float-3* - (float, RO) This is used for debugging purposes.
- *motion.debug-s32-0* - (s32, RO) This is used for debugging purposes.
- *motion.debug-s32-1* - (s32, RO) This is used for debugging purposes.
- *motion.servo.last-period* - (u32, RO) The number of CPU cycles between invocations of the servo thread. Typically, this number divided by the CPU speed gives the time in seconds, and can be used to determine whether the realtime motion controller is meeting its timing constraints
- *motion.servo.last-period-ns* - (float, RO)
- *motion.servo.overruns* - (u32, RW) 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.

7.1.4 Functions

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

- *motion-command-handler* - Processes motion commands coming from user space
- *motion-controller* - Runs the LinuxCNC motion controller

7.2 Axis (Joints)

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

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

7.2.1 Pins

- *axis.N.active* - (bit, out)
- *axis.N.amp-enable-out* - (bit, out) TRUE if the amplifier for this joint should be enabled
- *axis.N.amp-fault-in* - (bit, in) Should be driven TRUE if an external fault is detected with the amplifier for this joint
- *axis.N.backlash-corr* - (float, out)
- *axis.N.backlash-filt* - (float, out)
- *axis.N.backlash-vel* - (float, out)
- *axis.N.coarse-pos-cmd* - (float, out)
- *axis.N.error* - (bit, out)
- *axis.N.f-error* - (float, out)
- *axis.N.f-error-lim* - (float, out)
- *axis.N.f-errored* - (bit, out)
- *axis.N.faulted* - (bit, out)
- *axis.N.free-pos-cmd* - (float, out)
- *axis.N.free-tp-enable* - (bit, out)
- *axis.N.free-vel-lim* - (float, out)
- *axis.N.home-sw-in* - (bit, in) Should be driven TRUE if the home switch for this joint is closed.
- *axis.N.homed* - (bit, out)
- *axis.N.homing* - (bit, out) TRUE if the joint is currently homing
- *axis.N.in-position* - (bit, out)
- *axis.N.index-enable* - (bit, I/O)
- *axis.N.jog-counts* - (s32, in) Connect to the *counts* pin of an external encoder to use a physical jog wheel.
- *axis.N.jog-enable* - (bit, in) When TRUE (and in manual mode), any change in *jog-counts* will result in motion. When false, *jog-counts* is ignored.
- *axis.N.jog-scale* - (float, in) Sets the distance moved for each count on *jog-counts*, in machine units.
- *axis.N.jog-vel-mode* - (bit, in) When FALSE (the default), the jogwheel operates in position mode. The axis will move exactly jog-scale units for each count, regardless of how long that might take. When TRUE, the wheel operates in velocity mode - motion stops when the wheel stops, even if that means the commanded motion is not completed.
- *axis.N.joint-pos-cmd* - (float, out) The joint (as opposed to motor) commanded position. There may be an offset between the joint and motor positions—for example, the homing process sets this offset.
- *axis.N.joint-pos-fb* - (float, out) The joint (as opposed to motor) feedback position.
- *axis.N.joint-vel-cmd* - (float, out)
- *axis.N.kb-jog-active* - (bit, out)
- *axis.N.motor-pos-cmd* - (float, out) The commanded position for this joint.
- *axis.N.motor-pos-fb* - (float, in) The actual position for this joint.
- *axis.N.neg-hard-limit* - (bit, out)

- *axis.N.pos-lim-sw-in* - (bit, in) Should be driven TRUE if the positive limit switch for this joint is closed.
- *axis.N.pos-hard-limit* - (bit, out)
- *axis.N.neg-lim-sw-in* - (bit, in) Should be driven TRUE if the negative limit switch for this joint is closed.
- *axis.N.wheel-jog-active* - (bit, out)

7.2.2 Parameters

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

7.3 iocontrol

iocontrol – accepts NML I/O commands, interacts with HAL in userspace.

The signals are turned on and off in userspace - if you have strict timing requirements or simply need more i/o, consider using the realtime synchronized i/o provided by motion instead.

7.3.1 Pins

- *iocontrol.0.coolant-flood* - (bit, out) TRUE when flood coolant is requested.
- *iocontrol.0.coolant-mist* - (bit, out) TRUE when mist coolant is requested.
- *iocontrol.0.emc-enable-in* - (bit, in) Should be driven FALSE when an external E-Stop condition exists.
- *iocontrol.0.lube* - (bit, out) TRUE when lube is commanded.
- *iocontrol.0.lube_level* - (bit, in) Should be driven TRUE when lube level is high enough.
- *iocontrol.0.tool-change* - (bit, out) TRUE when a tool change is requested.
- *iocontrol.0.tool-changed* - (bit, in) Should be driven TRUE when a tool change is completed.
- *iocontrol.0.tool-number* - (s32, out) The current tool number.
- *iocontrol.0.tool-prep-number* - (s32, out) The number of the next tool, from the RS274NGC T-word.
- *iocontrol.0.tool-prepare* - (bit, out) TRUE when a tool prepare is requested.
- *iocontrol.0.tool-prepared* - (bit, in) Should be driven TRUE when a tool prepare is completed.
- *iocontrol.0.user-enable-out* - (bit, out) FALSE when an internal E-Stop condition exists.
- *iocontrol.0.user-request-enable* - (bit, out) TRUE when the user has requested that E-Stop be cleared.

Chapter 8

Stepper Configuration

8.1 Introduction

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

This chapter describes some of the more common settings for manually setting up a stepper based system. Because of the various possibilities of configuring LinuxCNC, it is very hard to document them all, and keep this document relatively short.

The most common LinuxCNC 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 LinuxCNC. The config is called stepper, and usually it is found in */etc/emc2/sample-configs/stepper*.

8.2 Maximum step rate

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

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

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

If no valid combination of BASE_PERIOD, INPUT_SCALE, and MAX_VELOCITY is acceptable, then consider using hardware step generation (such as with the LinuxCNC-supported Universal Stepper Controller, Mesa cards, and others.)

8.3 Pinout

One of the major flaws in LinuxCNC was that you couldn't specify the pinout without recompiling the source code. LinuxCNC is far more flexible, and now (thanks to the Hardware Abstraction Layer) you can easily specify which signal goes where. See the HAL manual for more detailed information on HAL.

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

Note
We are presenting one axis to keep it short, all others are similar.

The ones relevant for our pinout are:

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

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

8.3.1 standard_pinout.hal

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

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

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

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

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

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

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

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

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

###
```

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

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

###
### Shared home switches all on one parallel port pin?
### that's ok, hook the same signal to all the axes, but be sure to
### set HOME_IS_SHARED and HOME_SEQUENCE in the ini file.  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, LinuxCNC will stop but can't tell
### you which switch/axis has faulted.  Use caution when recovering from this.
###

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

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

8.3.2 Overview

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

- The Parport driver gets loaded (see the Parport section of the HAL Manual for details)
- The read & write functions of the parport driver get assigned to the base thread [1]
- The step & direction signals for axes X,Y,Z get linked to pins on the parport
- Further I/O signals get connected (estop loopback, toolchanger loopback)
- A spindle-on signal gets defined and linked to a parport pin

8.3.3 Changing the standard_pinout.hal

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

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

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

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

can be changed to:

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

or basically any other *out* pin you like.

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

8.3.4 Changing polarity of a signal

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

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

8.3.5 Adding PWM Spindle Speed Control

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

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

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

8.3.6 Adding an enable signal

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

To connect them use the following example:

```
net Xen parport.0.pin-08-out
```

You can either have one single pin that enables all drives; or several, depending on the setup you have. Note, however, that usually when one axis faults, all the other drives will be disabled as well, so having only one enable signal / pin for all drives is a common practice.

8.3.7 External ESTOP button

As you can see in the standard_pinout.hal file by default the stepper configuration assumes no external ESTOP button. [2]

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

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

```
net estop-loop iocontrol.0.user-enable-out iocontrol.0.emc-enable-in
```

with

```
net estop-loop parport.0.pin-01-in iocontrol.0.emc-enable-in
```

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

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

Part III

GUI

Chapter 9

Python Virtual Control Panel

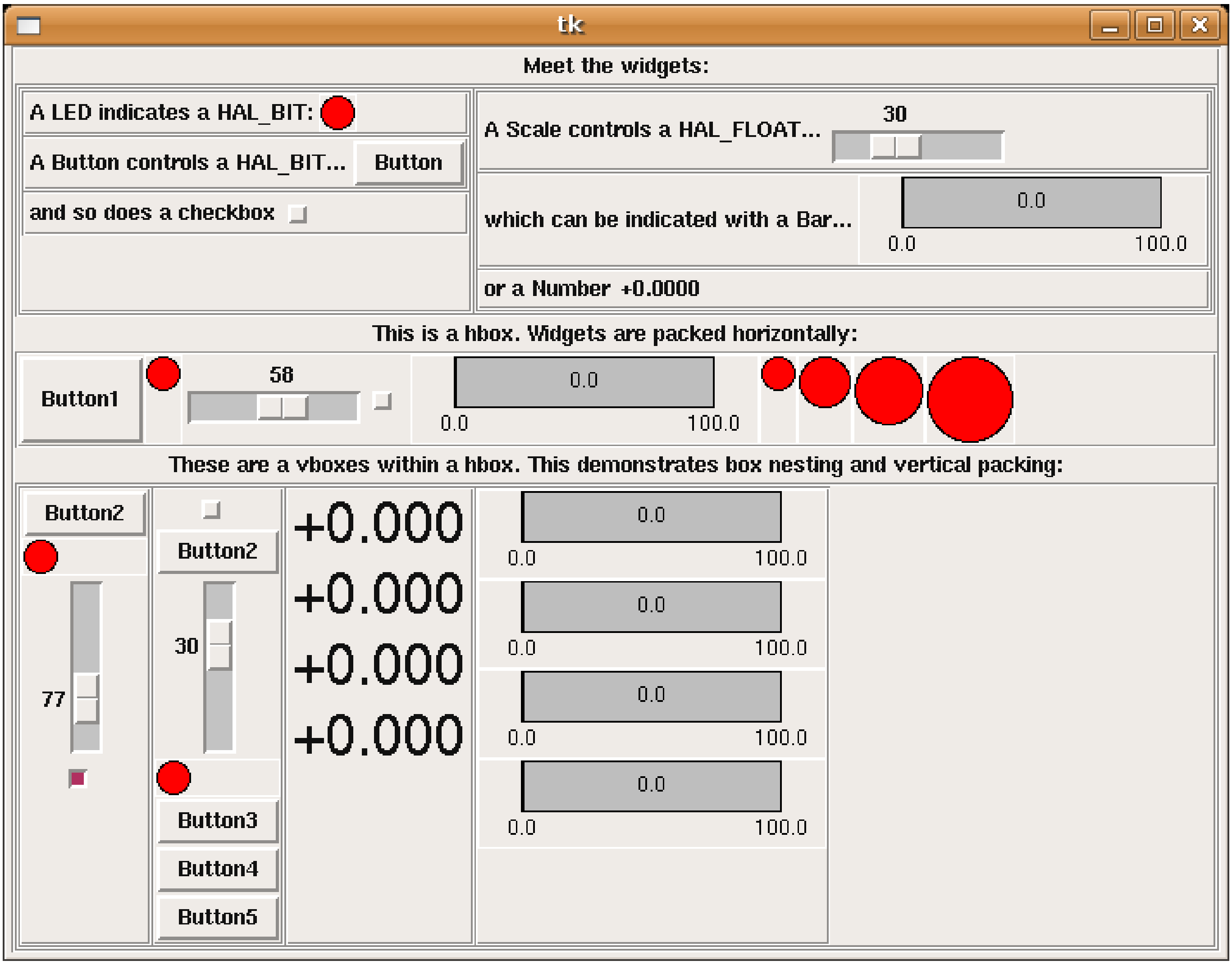
9.1 Introduction

Python Virtual Control Panel The PyVCP (Python Virtual Control Panel) is designed to give the integrator the ability to customize the AXIS interface with buttons and indicators to do special tasks.

Hardware machine control panels can use up a lot of I/O pins and can be expensive. That is where Virtual Control Panels have the advantage as well as it cost nothing to build a PyVCP.

Virtual Control Panels can be used for testing or monitoring things to temporarily replace real I/O devices while debugging ladder logic, or to simulate a physical panel before you build it and wire it to an I/O board.

The following graphic displays many of the PyVCP widgets.



9.2 Panel Construction

The layout of a PyVCP panel is specified with an XML file that contains widget tags between <pyvcp> and </pyvcp>. For example:

```
<pyvcp>
    <label text="This is a LED indicator"/>
    <led/>
</pyvcp>
```

If you place this text in a file called tiny.xml, and run

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

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

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

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

```
net <signal-name> <pin-name> <opt-direction> <opt-pin-name>signal-name
```

If you are new to HAL, the HAL basics chapter in the Integrator Manual is a good place to start.

9.3 Security

Parts of PyVCP files are evaluated as Python code, and can take any action available to Python programs. Only use PyVCP .xml files from a source that you trust.

9.4 AXIS

Since AXIS uses the same GUI toolkit (Tkinter) as PyVCP, it is possible to include a PyVCP panel on the right side of the normal AXIS user interface. A typical example is explained below.

Place your PyVCP XML file describing the panel in the same directory where your .ini file is. Say we we want to display the current spindle speed using a Bar widget. Place the following in a file called spindle.xml:

```
<pyvcp>
    <label>
        <text>"Spindle speed:"</text>
    </label>
    <bar>
        <halpin>"spindle-speed"</halpin>
        <max_>5000</max_>
    </bar>
</pyvcp>
```

Here we've made a panel with a Label and a Bar widget, specified that the HAL pin connected to the Bar should be named *spindle-speed*, and set the maximum value of the bar to 5000 (see widget reference below for all options). To make AXIS aware of this file, and call it at start up, we need to specify the following in the [DISPLAY] section of the .ini file:

```
PYVCP = spindle.xml
```

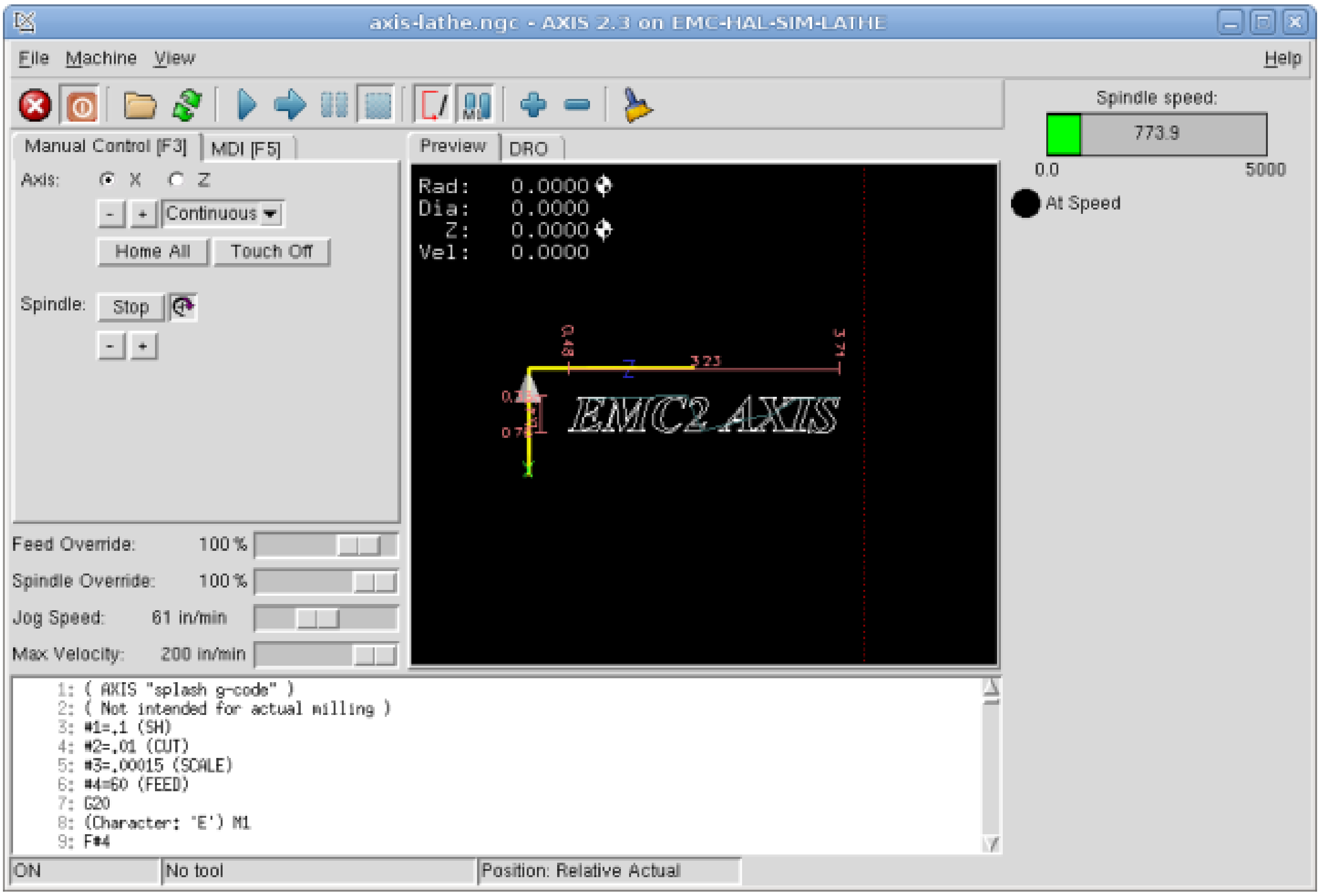
To make our widget actually display the spindle-speed it needs to be hooked up to the appropriate HAL signal. A .hal file that will be run once AXIS and PyVCP have started can be specified in the [HAL] section of the .ini file:

```
POSTGUI_HALFILE = spindle_to_pyvcp.hal
```

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

```
net spindle-rpm-filtered  => pyvcp.spindle-speed
```

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



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

9.5 Stand Alone

This section describes how PyVCP panels can be displayed on their own with or without LinuxCNC's machine controller.

To load a stand alone PyVCP panel with LinuxCNC use these commands:

```
loadusr -Wn mypanel pyvcp -g WxH+X+Y -c mypanel <path/>panel_file.xml
```

You would use this if you wanted a floating panel or a panel with a GUI other than AXIS.

- *-Wn panelname* - makes HAL wait for the component *panelname* to finish loading (*become ready* in HAL speak) before processing more HAL commands. This is important because PyVCP panels export HAL pins, and other HAL components will need them present to connect to them. Note the capital W and lowercase n. If you use the -Wn option you must use the -c option to name the panel.
- *pyvcp* < *-g*> < *-c*> *panel.xml* - builds the panel with the optional geometry and/or panelname from the xml panel file. The panel.xml can be any name that ends in .xml. The .xml file is the file that describes how to build the panel. You must add the path name if the panel is not in the directory that the HAL script is in.
- *-g <WxH><+X+Y>* - specifies the geometry to be used when constructing the panel. The syntax is *Width x Height + X Anchor + Y Anchor*. You can set the size or position or both. The anchor point is the upper left corner of the panel. An example is -g 250x500+800+0 This sets the panel at 250 pixels wide, 500 pixels tall, and anchors it at X800 Y0.
- *-c panelname* - tells PyVCP what to call the component and also the title of the window. The panelname can be any name without spaces.

To load a *stand alone* PyVCP panel without LinuxCNC use this command:

```
loadusr -Wn mypanel pyvcp -g 250x500+800+0 -c mypanel mypanel.xml
```

The minimum command to load a pyvcp panel is:

```
loadusr pyvcp mypanel.xml
```

You would use this if you want a panel without LinuxCNC's machine controller such as for testing or a standalone DRO.

The loadusr command is used when you also load a component that will stop HAL from closing until it's done. If you loaded a panel and then loaded Classic Ladder using *loadusr -w classicladder*, CL would hold HAL open (and the panel) until you closed CL. The *-Wn* above means wait for the component *-Wn "name"* to become ready. (*name* can be any name. Note the capital W and lowercase n.) The -c tells PyVCP to build a panel with the name *panelname* using the info in *panel_file_name.xml*. The name *panel_file_name.xml* can be any name but must end in .xml - it is the file that describes how to build the panel. You must add the path name if the panel is not in the directory that the HAL script is in.

An optional command to use if you want the panel to stop HAL from continuing commands / shutting down. After loading any other components you want the last HAL command to be:

```
waituser panelname
```

This tells HAL to wait for component *panelname* to close before continuing HAL commands. This is usually set as the last command so that HAL shuts down when the panel is closed.

9.6 Widgets

HAL signals come in two variants, bits and numbers. Bits are off/on signals. Numbers can be *float*, *s32* or *u32*. For more information on HAL data types see the HAL manual. The PyVCP widget can either display the value of the signal with an indicator widget, or modify the signal value with a control widget. Thus there are four classes of PyVCP widgets that you can connect to a HAL signal. A fifth class of helper widgets allow you to organize and label your panel.

1. Widgets for indicating *bit* signals: led, rectled
2. Widgets for controlling *bit* signals: button, checkbutton, radiobutton
3. Widgets for indicating *number* signals: number, s32, u32, bar, meter
4. Widgets for controlling *number* signals: spinbox, scale, jogwheel
5. Helper widgets: hbox, vbox, table, label, labelframe

9.6.1 Syntax

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

9.6.2 General Notes

At the present time, both a tag-based and an attribute-based syntax are supported. For instance, the following XML fragments are treated identically:

```
<led halpin="my-led"/>
```

and

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

When the attribute-based syntax is used, the following rules are used to turn the attributes value into a Python value:

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

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

The examples below show a mix of formats.

9.6.2.1 Comments

To add a comment use the xml syntax for a comment.

```
<!-- My Comment -->
```

9.6.2.2 Editing the XML file

Edit the XML file with a text editor. In most cases you can right click on the file and select *open with text editor* or similar.

9.6.2.3 Colors

Colors can be specified using the X11 rgb colors by name *gray75* or hex *#0000ff*. A complete list is located here http://sedition.com/-perl/rgb.html.

Common Colors (colors with numbers indicate shades of that color)

- white
- black
- blue and blue1 - 4
- cyan and cyan1 - 4
- green and green1 - 4
- yellow and yellow1 - 4
- red and red1 - 4
- purple and purple1 - 4
- gray and gray0 - 100

9.6.2.4 HAL Pins

HAL pins provide a means to *connect* the widget to something. Once you create a HAL pin for your widget you can *connect* it to another HAL pin with a *net* command in a .hal file. For more information on the *net* command see the HAL Commands section of the HAL manual.

9.6.3 Label

A label is a piece of text on your panel.

The label has an optional disable pin that is created when you add *<disable_pin>True</disable_pin>*.

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

The above code produced this example.

9.6.4 LEDs

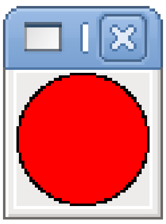
A LED is used to indicate the status of a *bit* halpin. The LED color will be on_color when the halpin is true, and off_color otherwise.

- *<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 *red*
- *<disable_pin>* - when true adds a disable pin to the led.
- *<disabled_color>* - sets the color of the LED when the pin is disabled.

9.6.4.1 Round LED

```
<led>
    <halpin>"my-led"</halpin>
    <size>50</size>
    <on_color>"green"</on_color>
    <off_color>"red"</off_color>
</led>
```

The above code produced this example.



9.6.4.2 Rectangle LED

This is a variant of the *led* widget.

```
<vbox>
    <relief>RIDGE</relief>
    <bd>6</bd>
    <rectled>
        <halpin>"my-led"</halpin>
        <height>"50"</height>
        <width>"100"</width>
        <on_color>"green"</on_color>
        <off_color>"red"</off_color>
    </rectled>
</vbox>
```

The above code produced this example. Also showing a vertical box with relief.

9.6.5 Buttons

A button is used to control a BIT pin. The pin will be set True when the button is pressed and held down, and will be set False when the button is released. Buttons can use the following formatting options

- *<padx>n</padx>* - where *n* is the amount of extra horizontal extra space
- *<pady>n</pady>* - where *n* is the amount of extra vertical extra space
- *<activebackground>"color"</activebackground>* - the cursor over color
- *<bg>"color"</bg>* - the color of the button

9.6.5.1 Text Button

A text button controls a *bit* halpin. The halpin is false until the button is pressed then it is true. The button is a momentary button.

The text button has an optional disable pin that is created when you add <disable_pin>True</disable_pin>.

```
<button>
    <halpin>"ok-button"</halpin>
    <text>"OK"</text>
</button>
<button>
    <halpin>"abort-button"</halpin>
    <text>"Abort"</text>
</button
```

The above code produced this example.

9.6.5.2 Checkbutton

A checkbutton controls a *bit* halpin. The halpin will be set True when the button is checked, and false when the button is unchecked. The checkbutton is a toggle type button.

```
<checkbutton>
    <halpin>"coolant-chkbtn"</halpin>
    <text>"Coolant"</text>
</checkbutton>
<checkbutton>
    <halpin>"chip-chkbtn"</halpin>
    <text>"Chips    "</text>
</checkbutton>
```

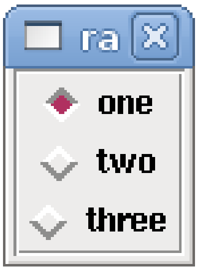
The above code produced this example. The coolant checkbutton is checked. Notice the extra spaces in the Chips text to keep the checkbuttons aligned.

9.6.5.3 Radiobutton

A radiobutton will set one of the halpins true. The other pins are set false.

```
<radiobutton>
    <choices>["one","two","three"]</choices>
    <halpin>"my-radio"</halpin>
</radiobutton>
```

The above code produced this example.



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

9.6.6 Number Displays

Number displays can use the following formatting options

- ("Font Name",n) where *n* is the font size
- <width>n</width> where *n* is the overall width of the space used
- <justify>pos</justify> where *pos* is LEFT, CENTER, or RIGHT (doesn't work)
- <padx>n</padx> where *n* is the amount of extra horizontal extra space
- <pady>n</pady> where *n* is the amount of extra vertical extra space

9.6.6.1 Number

The number widget displays the value of a float signal.

```
<number>
    <halpin>"my-number"</halpin>
    <font>("Helvetica",24)</font>
    <format>"+4.4f"</format>
</number>
```

The above code produced this example.

- *<font>* - is a Tkinter font type and size specification. One font that will show up to at least size 200 is *courier 10 pitch*, so for a really big Number widget you could specify:

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

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

9.6.6.2 s32 Number

The s32 number widget displays the value of a s32 number. The syntax is the same as *number* except the name which is <s32>. Make sure the width is wide enough to cover the largest number you expect to use.

```
<s32>
    <halpin>"my-number"</halpin>
    <font>("Helvetica",24)</font>
    <format>"6d"</format>
    <width>6</width>
</s32>
```

The above code produced this example.

9.6.6.3 u32 Number

The u32 number widget displays the value of a u32 number. The syntax is the same as *number* except the name which is <u32>.

9.6.6.4 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>
```

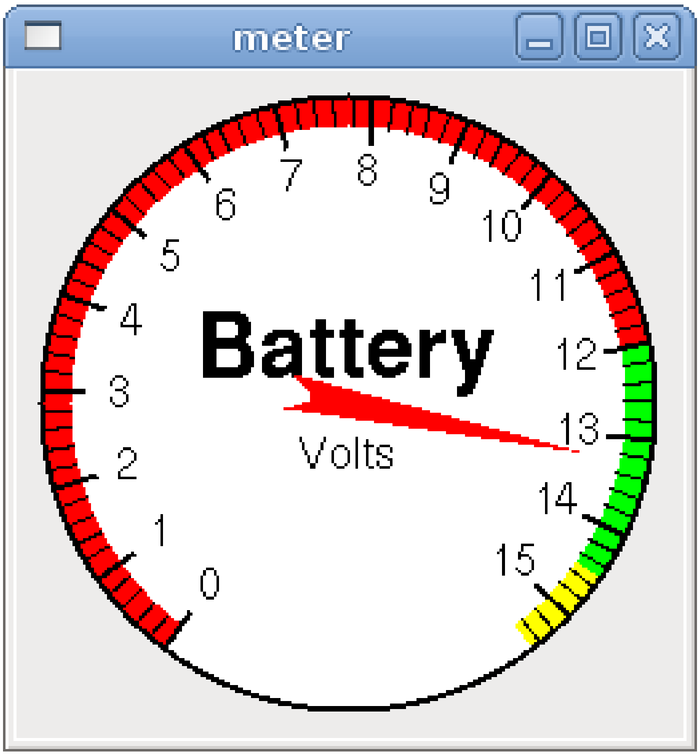
The above code produced this example.

9.6.6.5 Meter

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

```
<meter>
    <halpin>"mymeter"</halpin>
    <text>"Battery"</text>
    <subtext>"Volts"</subtext>
    <size>250</size>
    <min_>0</min_>
    <max_>15.5</max_>
    <majorscale>1</majorscale>
    <minorscale>0.2</minorscale>
    <region1>(14.5,15.5,"yellow")</region1>
    <region2>(12,14.5,"green")</region2>
    <region3>(0,12,"red")</region3>
</meter>
```

The above code produced this example.



9.6.7 Number Inputs

9.6.7.1 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_>
    <initval>0</initval>
    <resolution>0.1</resolution>
    <format>"2.3f"</format>
    <font>("Arial",30)</font>
</spinbox>
```

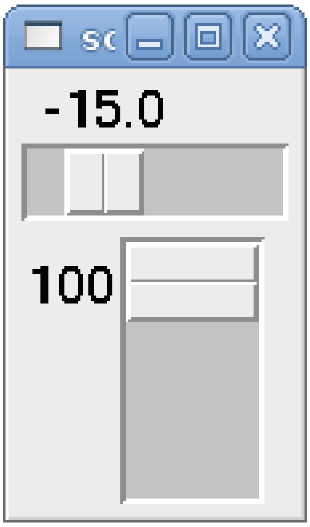
The above code produced this example.

9.6.7.2 Scale

Scale controls a float or a s32 pin. You increase or decrease the value of the pin be either dragging the slider, or pointing at the scale and rolling your mouse-wheel. The *halpin* will have both *-f* and *-i* added to it to form the float and s32 pins. Width is the width of the slider in vertical and the height of the slider in horizontal orientation.

```
<scale>
    <font>("Helvetica",16)</font>
    <width>"25"</width>
    <halpin>"my-hscale"</halpin>
    <resolution>0.1</resolution>
    <orient>HORIZONTAL</orient>
    <initval>-15</initval>
    <min_>-33</min_>
    <max_>26</max_>
</scale>
<scale>
    <font>("Helvetica",16)</font>
    <width>"50"</width>
    <halpin>"my-vscale"</halpin>
    <resolution>1</resolution>
    <orient>VERTICAL</orient>
    <min_>100</min_>
    <max_>0</max_>
</scale>
```

The above code produced this example.

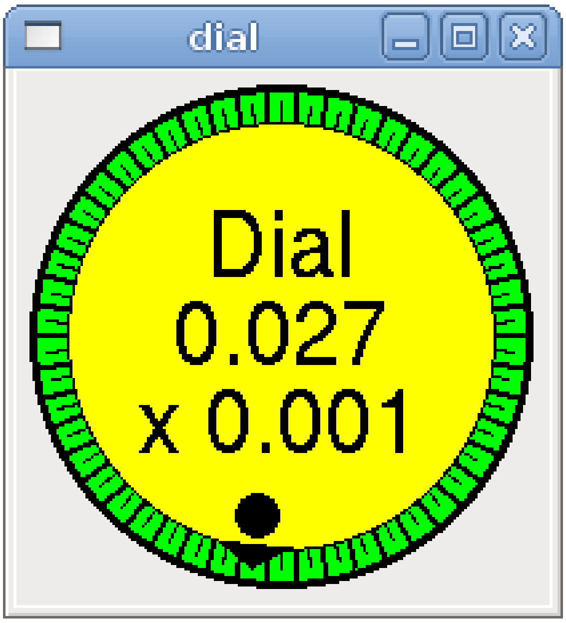


9.6.7.3 Dial

The Dial outputs a HAL float and reacts to both mouse wheel and dragging. Double left click to increase the resolution and double right click to reduce the resolution by one digit. The output is capped by the min and max values. The <cpr> is how many tick marks are on the outside of the ring (beware of high numbers).

```
<dial>
    <size>200</size>
    <cpr>100</cpr>
    <min_>-15</min_>
    <max_>15</max_>
    <text>"Dial"</text>
    <initval>0</initval>
    <resolution>0.001</resolution>
    <halpin>"anaout"</halpin>
    <dialcolor>"yellow"</dialcolor>
    <edgecolor>"green"</edgecolor>
    <dotcolor>"black"</dotcolor>
</dial>
```

The above code produced this example.

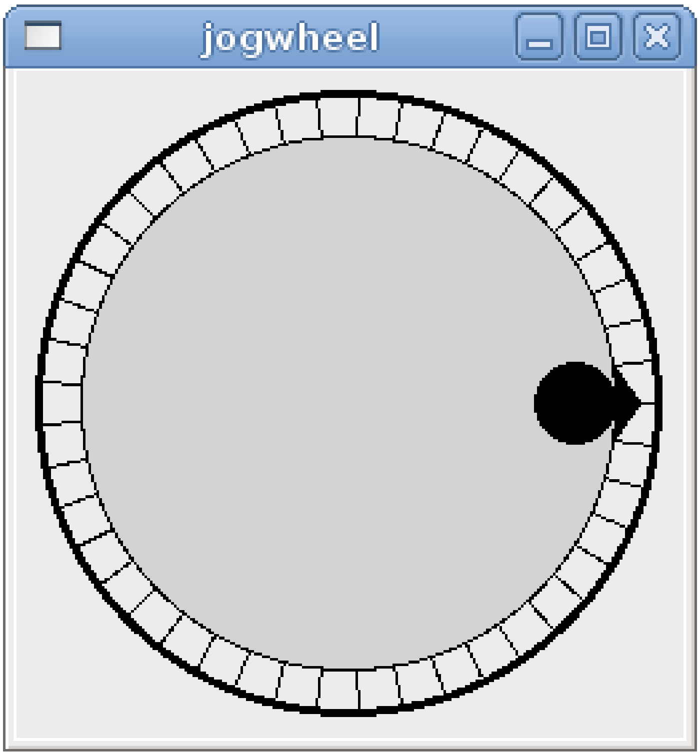


9.6.7.4 Jogwheel

Jogwheel mimics a real jogwheel by outputting a FLOAT pin which counts up or down as the wheel is turned, either by dragging in a circular motion, or by rolling the mouse-wheel.

```
<jogwheel>
    <halpin>"my-wheel"</halpin>
    <cpr>45</cpr>
    <size>250</size>
</jogwheel>
```

The above code produced this example.



9.6.8 Images

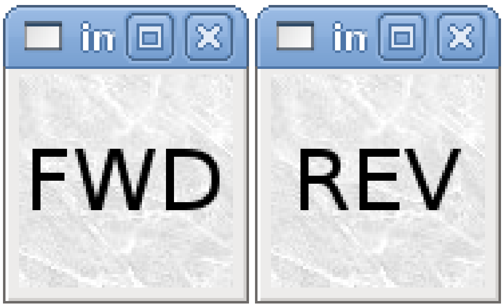
Image displays use only .gif image format. All of the images must be the same size. The images must be in the same directory as your ini file (or in the current directory if running from the command line with halrun/halcmd).

9.6.8.1 Image Bit

The *image_bit* toggles between two images by setting the halpin to true or false.

```
<image name='fwd' file='fwd.gif'/>
<image name='rev' file='rev.gif'/>
<vbox>
    <image_bit halpin='selectimage' images='fwd rev'/>
</vbox>
```

This example was produced from the above code. Using the two image files fwd.gif and rev.gif. FWD is displayed when *selectimage* is false and REV is displayed when *selectimage* is true.

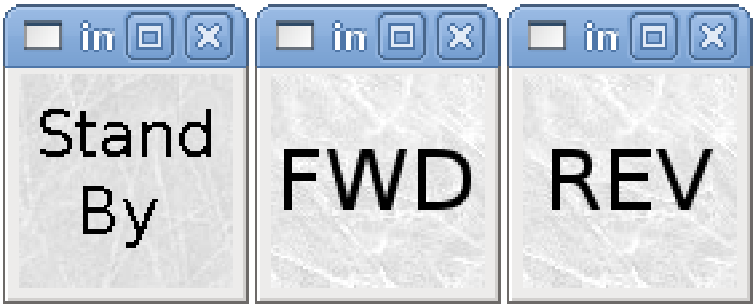


9.6.8.2 Image u32

The *image_u32* is the same as *image_bit* except you have essentially an unlimited number of images and you *select* the image by setting the halpin to a integer value with 0 for the first image in the images list and 1 for the second image etc.

```
<image name='stb' file='stb.gif'/>
<image name='fwd' file='fwd.gif'/>
<image name='rev' file='rev.gif'/>
<vbox>
    <image_u32 halpin='selectimage' images='stb fwd rev'/>
</vbox>
```

The above code produced the following example by adding the stb.gif image.



Notice that the default is the min even though it is set higher than max unless there is a negative min.

9.6.9 Containers

Containers are widgets that contain other widgets. Containers are used to group other widgets.

9.6.9.1 Borders

Container borders are specified with two tags used together. The <relief> tag specifies the type of border and the <bd> specifies the width of the border.

- *<relief>type</relief>* - Where *type* is FLAT, SUNKEN, RAISED, GROOVE, or RIDGE
- *<bd>n</bd>* - Where *n* is the width of the border.

```
<hbox>
    <button>
        <relief>FLAT</relief>
        <text>"FLAT"</text>
        <bd>3</bd>
    </button>
    <button>
        <relief>SUNKEN</relief>
        <text>"SUNKEN"</text>
        <bd>3</bd>
    </button>
    <button>
        <relief>RAISED</relief>
        <text>"RAISED"</text>
        <bd>3</bd>
    </button>
    <button>
        <relief>GROOVE</relief>
        <text>"GROOVE"</text>
        <bd>3</bd>
    </button>
    <button>
        <relief>RIDGE</relief>
```

```
        <text>"RIDGE"</text>
        <bd>3</bd>
    </button>
</hbox>
```

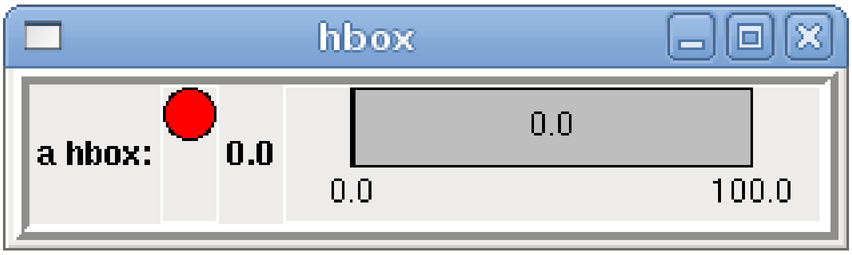
The above code produced this example.

9.6.9.2 Hbox

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

```
<hbox>
    <relief>RIDGE</relief>
    <bd>6</bd>
    <label><text>"a hbox:"</text></label>
    <led></led>
    <number></number>
    <bar></bar>
</hbox>
```

The above code produced this example.



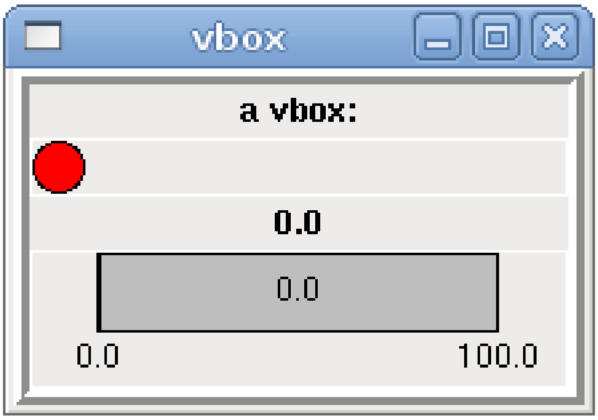
Inside an Hbox, you can use the *<boxfill fill=""/>*, *<boxanchor anchor=""/>*, and *<boxexpand expand=""/>* tags to choose how items in the box behave when the window is re-sized. For details of how fill, anchor, and expand behave, refer to the Tk *pack* manual page, *pack(3tk)*. By default, *fill="y"*,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

9.6.9.3 Vbox

Use a Vbox when you want to stack widgets vertically on top of each other.

```
<vbox>
    <relief>RIDGE</relief>
    <bd>6</bd>
    <label><text>"a vbox:"</text></label>
    <led></led>
    <number></number>
    <bar></bar>
</vbox>
```

The above code produced this example.



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

9.6.9.4 Labelframe

A labelframe is a frame with a groove and a label at the upper-left corner.

```
<labelframe text="Group Title">
    <font>("Helvetica",16)</font>
    <hbox>
    <led/>
    <led/>
    </hbox>
</labelframe>
```

The above code produced this example.

9.6.9.5 Table

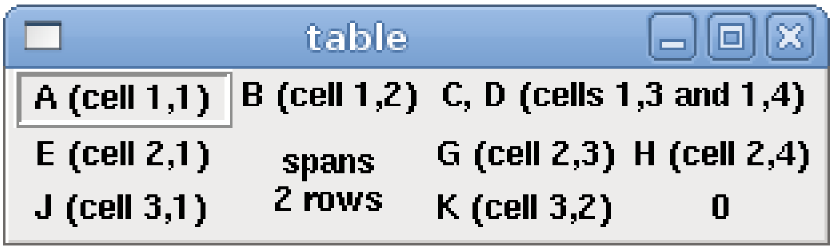
A table is a container that allows layout in a grid of rows and columns. Each row is started by a *<tablerow/>* tag. A contained widget may span rows or columns through the use of the *<tablespan rows= cols=/>* tag. The sides of the cells to which the contained widgets “stick” may be set through the 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) "</text>
        <relief>RIDGE</relief>
        <bd>3</bd>
    </label>
    <label text="B (cell 1,2)"/>
```

```
    <tablespan columns="2"/>
    <label text="C, D (cells 1,3 and 1,4)"/>
<tablerow/>
    <label text="E (cell 2,1)"/>
    <tablesticky sticky="nsew"/>
    <tablespan rows="2"/>
    <label text="'spans\n2 rows'"/>
    <tablesticky sticky="new"/>
    <label text="G (cell 2,3)"/>
    <label text="H (cell 2,4)"/>
<tablerow/>
    <label text="J (cell 3,1)"/>
    <label text="K (cell 3,2)"/>
    <u32 halpin="test"/>
</table>
```

The above code produced this example.

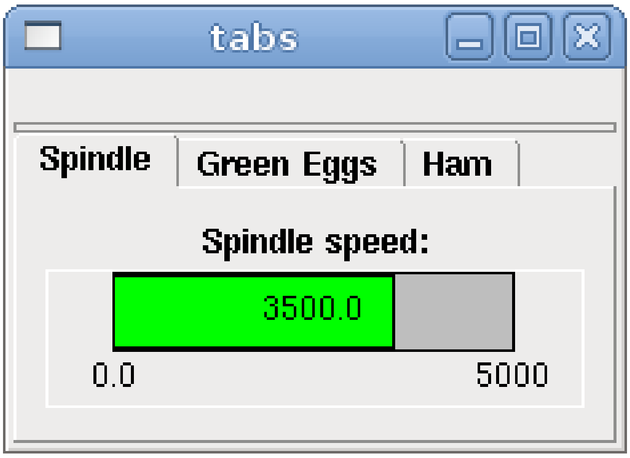


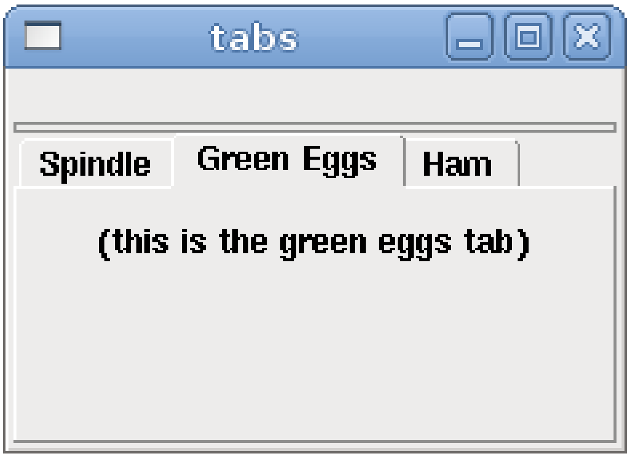
9.6.9.6 Tabs

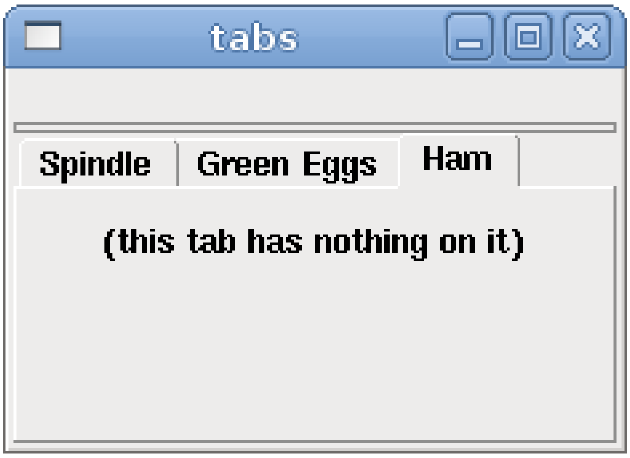
A tabbed interface can save quite a bit of space.

```
<tabs>
    <names> ["spindle","green eggs"]</names>
</tabs>
<tabs>
    <names>["Spindle", "Green Eggs", "Ham"]</names>
    <vbox>
        <label>
            <text>"Spindle speed:"</text>
        </label>
        <bar>
            <halpin>"spindle-speed"</halpin>
            <max_>5000</max_>
        </bar>
    </vbox>
    <vbox>
        <label>
            <text>"(this is the green eggs tab)"</text>
        </label>
    </vbox>
    <vbox>
        <label>
            <text>"(this tab has nothing on it)"</text>
        </label>
    </vbox>
</tabs>
```

The above code produced this example showing each tab selected.


tabs
Spindle
Green Eggs
Ham
Spindle speed:
3500.0
0.0
5000


tabs
Spindle
Green Eggs
Ham
(this is the green eggs tab)


tabs
Spindle
Green Eggs
Ham
(this tab has nothing on it)

Chapter 10

PyVCP Examples

10.1 AXIS

To create a PyVCP panel to use with the AXIS interface that is attached to the right of AXIS you need to do the following basic things.

- Create an .xml file that contains your panel description and put it in your config directory.
- Add the PyVCP entry to the [DISPLAY] section of the ini file with your .xml file name.
- Add the POSTGUI_HALFILE entry to the [HAL] section of the ini file with the name of your postgui HAL file name.
- Add the links to HAL pins for your panel in the postgui.hal file to *connect* your PyVCP panel to EMC.

10.2 Floating

To create floating PyVCP panels that can be used with any interface you need to do the following basic things.

- Create an .xml file that contains your panel description and put it in your config directory.
- Add a loadusr line to your .hal file to load each panel.
- Add the links to HAL pins for your panel in the postgui.hal file to *connect* your PyVCP panel to EMC.

The following is an example of a loadusr command to load two PyVCP panels and name each one so the connection names in HAL will be known.

```
loadusr -Wn btnpanel pyvcp -c btnpanel panel1.xml
loadusr -Wn sppanel pyvcp -c sppanel panel2.xml
```

The -Wn makes HAL *Wait for name* to be loaded before proceeding. The pyvcp -c makes PyVCP name the panel.

The HAL pins from panel1.xml will be named btnpanel.<pin name>

The HAL pins from panel2.xml will be named sppanel.<pin name>

Make sure the loadusr line is before any nets that make use of the PyVCP pins.

10.3 Jog Buttons

In this example we will create a PyVCP panel with jog buttons for X, Y, and Z. This configuration will be built upon a Stepconf Wizard generated configuration. First we run the Stepconf Wizard and configure our machine, then on the Advanced Configuration Options page we make a couple of selections to add a blank PyVCP panel as shown in the following figure. For this example we named the configuration *pyvcp_xyz* on the Basic Machine Information page of the Stepconf Wizard.

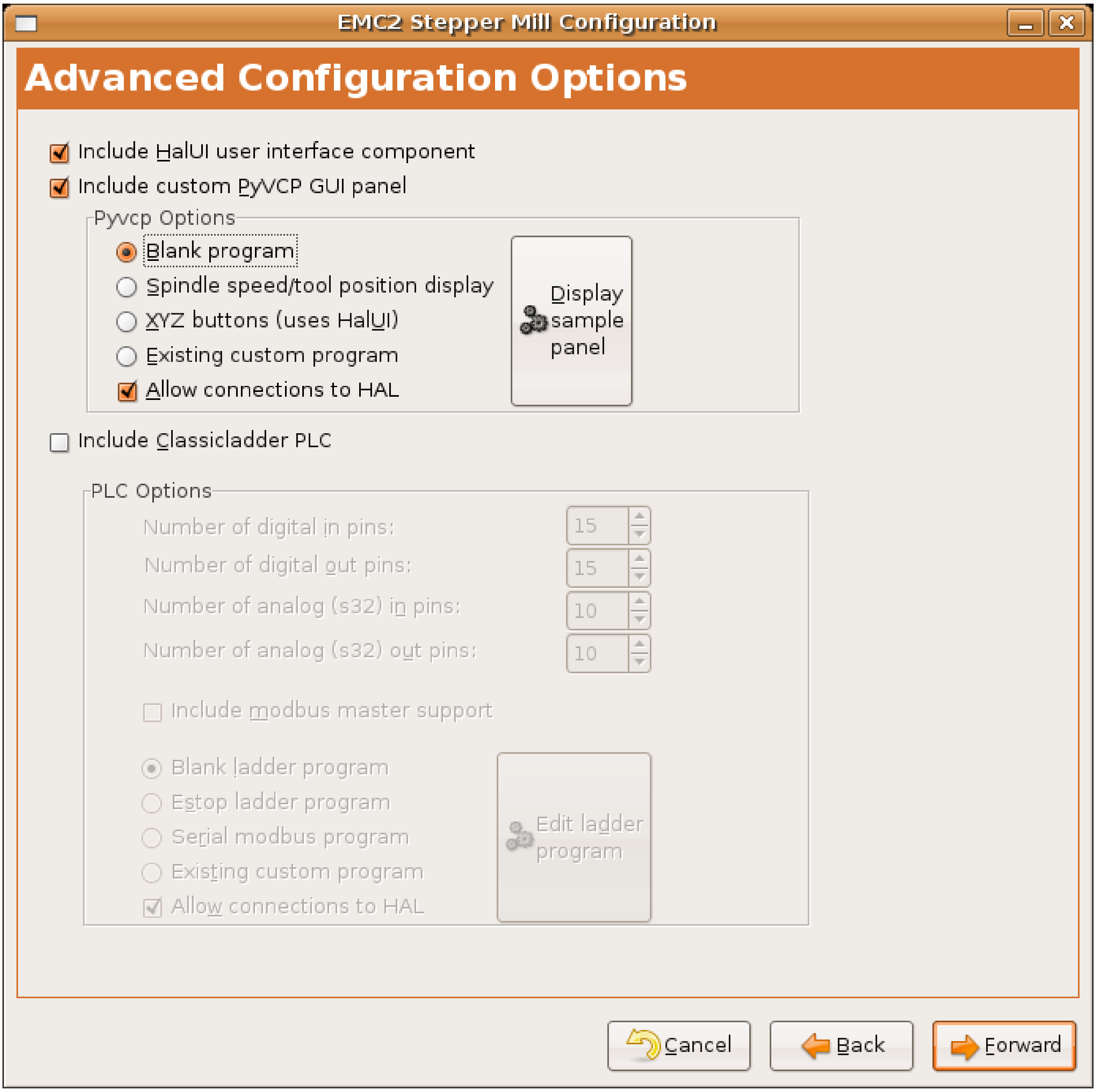


Figure 10.1: XYZ Wizard Configuration

The Stepconf Wizard will create several files and place them in the /emc/configs/pyvcp_xyz directory. If you left the create link checked you will have a link to those files on your desktop.

10.3.1 Create the Widgets

Open up the custompanel.xml file by right clicking on it and selecting *open with text editor*. Between the <pyvcp></pyvcp> tags we will add the widgets for our panel.

Look in the PyVCP Widgets Reference section of the manual for more detailed information on each widget.

In your custompanel.xml file we will add the description of the widgets.

```
<pyvcp>
  <labelframe text="Jog Buttons">
    <font>("Helvetica",16)</font>

      <!-- the X jog buttons -->
      <hbox>
      <relief>RAISED</relief>
      <bd>3</bd>
      <button>
        <font>("Helvetica",20)</font>
        <width>3</width>
        <halpin>"x-plus"</halpin>
        <text>"X+"</text>
      </button>
      <button>
        <font>("Helvetica",20)</font>
        <width>3</width>
        <halpin>"x-minus"</halpin>
        <text>"X-"</text>
      </button>
      </hbox>

      <!-- the Y jog buttons -->
      <hbox>
      <relief>RAISED</relief>
      <bd>3</bd>
      <button>
        <font>("Helvetica",20)</font>
        <width>3</width>
        <halpin>"y-plus"</halpin>
        <text>"Y+"</text>
      </button>
      <button>
        <font>("Helvetica",20)</font>
        <width>3</width>
        <halpin>"y-minus"</halpin>
        <text>"Y-"</text>
      </button>
      </hbox>

      <!-- the Z jog buttons -->
      <hbox>
      <relief>RAISED</relief>
      <bd>3</bd>
      <button>
        <font>("Helvetica",20)</font>
        <width>3</width>
        <halpin>"z-plus"</halpin>
        <text>"Z+"</text>
      </button>
      <button>
        <font>("Helvetica",20)</font>
        <width>3</width>
        <halpin>"z-minus"</halpin>
```

```
        <text>"Z-"</text>
      </button>
      </hbox>

      <!-- the jog speed slider -->
      <vbox>
      <relief>RAISED</relief>
      <bd>3</bd>
      <label>
        <text>"Jog Speed"</text>
        <font>("Helvetica",16)</font>
      </label>
      <scale>
        <font>("Helvetica",14)</font>
        <halpin>"jog-speed"</halpin>
        <resolution>1</resolution>
        <orient>HORIZONTAL</orient>
        <min_>0</min_>
        <max_>80</max_>
      </scale>
      </vbox>
  </labelframe>
</pyvcp>
```

After adding the above you now will have a PyVCP panel that looks like the following attached to the right side of AXIS. It looks nice but it does not do anything until you *connect* the buttons to halui. If you get an error when you try and run scroll down to the bottom of the pop up window and usually the error is a spelling or syntax error and it will be there.

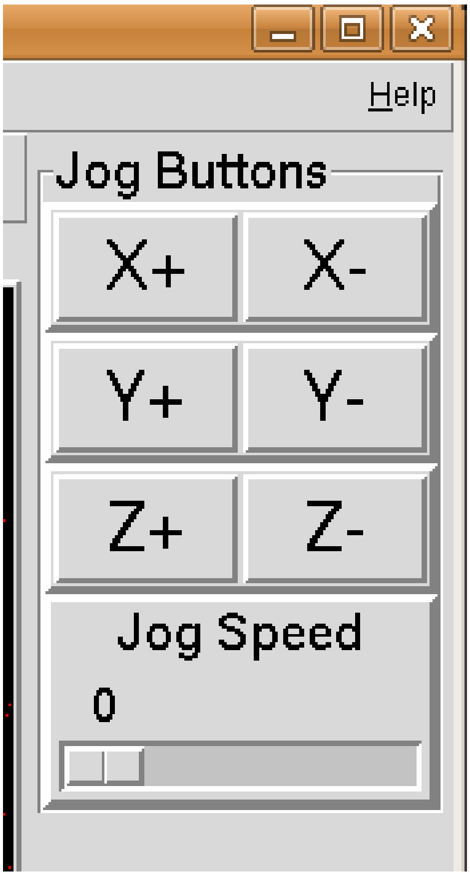


Figure 10.2: Jog Buttons

10.3.2 Make Connections

To make the connections needed open up your custom_postgui.hal file and add the following.

```
# connect the X PyVCP buttons
net my-jogxminus halui.jog.0.minus <= pyvcp.x-minus
net my-jogxplus halui.jog.0.plus <= pyvcp.x-plus

# connect the Y PyVCP buttons
net my-jogyminus halui.jog.1.minus <= pyvcp.y-minus
net my-jogyplus halui.jog.1.plus <= pyvcp.y-plus

# connect the Z PyVCP buttons
net my-jogzminus halui.jog.2.minus <= pyvcp.z-minus
net my-jogzplus halui.jog.2.plus <= pyvcp.z-plus

# connect the PyVCP jog speed slider
  net my-jogspeed halui.jog-speed <= pyvcp.jog-speed-f
```

After resetting the E-Stop and putting it into jog mode and moving the jog speed slider in the PyVCP panel to a value greater than zero the PyVCP jog buttons should work. You can not jog when running a g code file or while paused or while the MDI tab is selected.

10.4 Port Tester

This example shows you how to make a simple parallel port tester using PyVCP and HAL.

First create the ptest.xml file with the following code to create the panel description.

```
<!-- Test panel for the parallel port cfg for out -->
<pyvcp>
  <hbox>
    <relief>RIDGE</relief>
    <bd>2</bd>
    <button>
      <halpin>"btn01"</halpin>
      <text>"Pin 01"</text>
    </button>
    <led>
      <halpin>"led-01"</halpin>
      <size>25</size>
      <on_color>"green"</on_color>
      <off_color>"red"</off_color>
    </led>
  </hbox>
  <hbox>
    <relief>RIDGE</relief>
    <bd>2</bd>
    <button>
      <halpin>"btn02"</halpin>
      <text>"Pin 02"</text>
    </button>
    <led>
      <halpin>"led-02"</halpin>
      <size>25</size>
      <on_color>"green"</on_color>
      <off_color>"red"</off_color>
    </led>
  </hbox>
  <hbox>
```

```
    <relief>RIDGE</relief>
    <bd>2</bd>
    <label>
      <text>"Pin 10"</text>
      <font>("Helvetica",14)</font>
    </label>
    <led>
      <halpin>"led-10"</halpin>
      <size>25</size>
      <on_color>"green"</on_color>
      <off_color>"red"</off_color>
    </led>
  </hbox>
  <hbox>
    <relief>RIDGE</relief>
    <bd>2</bd>
    <label>
      <text>"Pin 11"</text>
      <font>("Helvetica",14)</font>
    </label>
    <led>
      <halpin>"led-11"</halpin>
      <size>25</size>
      <on_color>"green"</on_color>
      <off_color>"red"</off_color>
    </led>
  </hbox>
</pyvcp>
```

This will create the following floating panel which contains a couple of in pins and a couple of out pins.

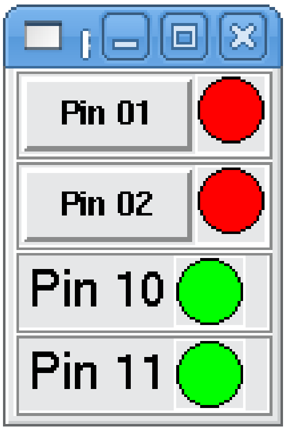


Figure 10.3: Port Tester Panel

To run the HAL commands that we need to get everything up and running we put the following in our ptest.hal file.

```
loadrt hal_parport cfg="0x378 out"
loadusr -Wn ptest pyvcp -c ptest ptest.xml
loadrt threads name1=porttest period1=1000000
addf parport.0.read porttest
addf parport.0.write porttest
net pin01 ptest.btn01 parport.0.pin-01-out ptest.led-01
net pin02 ptest.btn02 parport.0.pin-02-out ptest.led-02
net pin10 parport.0.pin-10-in ptest.led-10
net pin11 parport.0.pin-11-in ptest.led-11
start
```

To run the HAL file we use the following command from a terminal window.

```
~$ halrun -I -f ptest.hal
```

The following figure shows what a complete panel might look like.

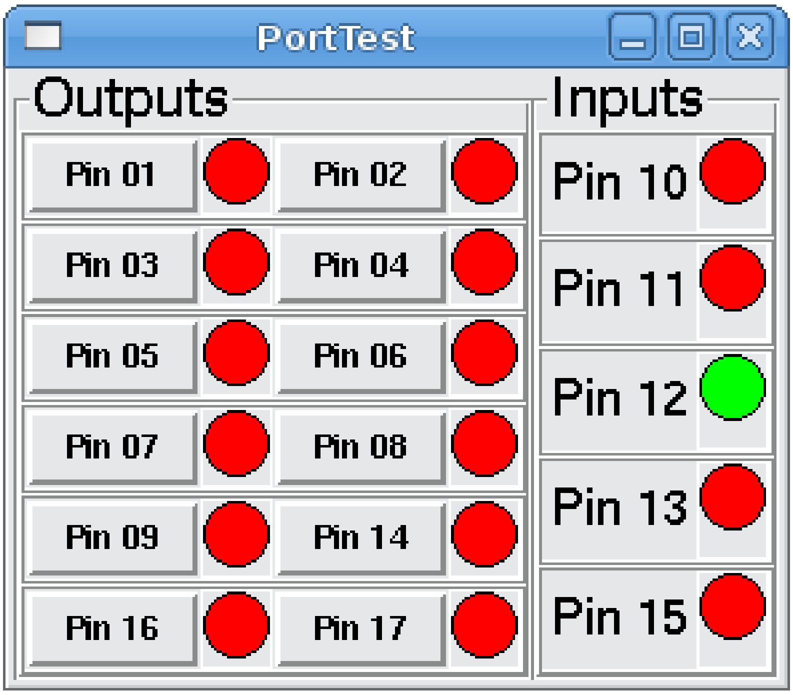


Figure 10.4: Port Tester Complete

To add the rest of the parallel port pins just modify the .xml and .hal files.

To show the pins after running the HAL script use the following command at the halcmd prompt:

```
halcmd: show pin
Component Pins:
Owner Type  Dir Value  Name
    2 bit   IN  FALSE  parport.0.pin-01-out <== pin01
    2 bit   IN  FALSE  parport.0.pin-02-out <== pin02
    2 bit   IN  FALSE  parport.0.pin-03-out
    2 bit   IN  FALSE  parport.0.pin-04-out
    2 bit   IN  FALSE  parport.0.pin-05-out
    2 bit   IN  FALSE  parport.0.pin-06-out
    2 bit   IN  FALSE  parport.0.pin-07-out
    2 bit   IN  FALSE  parport.0.pin-08-out
    2 bit   IN  FALSE  parport.0.pin-09-out
    2 bit   OUT TRUE   parport.0.pin-10-in ==> pin10
    2 bit   OUT FALSE  parport.0.pin-10-in-not
    2 bit   OUT TRUE   parport.0.pin-11-in ==> pin11
    2 bit   OUT FALSE  parport.0.pin-11-in-not
    2 bit   OUT TRUE   parport.0.pin-12-in
    2 bit   OUT FALSE  parport.0.pin-12-in-not
    2 bit   OUT TRUE   parport.0.pin-13-in
    2 bit   OUT FALSE  parport.0.pin-13-in-not
    2 bit   IN  FALSE  parport.0.pin-14-out
    2 bit   OUT TRUE   parport.0.pin-15-in
    2 bit   OUT FALSE  parport.0.pin-15-in-not
    2 bit   IN  FALSE  parport.0.pin-16-out
```

```
    2 bit    IN   FALSE   parport.0.pin-17-out
    4 bit    OUT FALSE   ptest.btn01 ==> pin01
    4 bit    OUT FALSE   ptest.btn02 ==> pin02
    4 bit    IN   FALSE   ptest.led-01 <== pin01
    4 bit    IN   FALSE   ptest.led-02 <== pin02
    4 bit    IN   TRUE    ptest.led-10 <== pin10
    4 bit    IN   TRUE    ptest.led-11 <== pin11
```

This will show you what pins are IN and what pins are OUT as well as any connections.

10.5 GS2 RPM Meter

The following example uses the Automation Direct GS2 VDF driver and displays the RPM and other info in a PyVCP panel. This example is based on the GS2 example in the Hardware Examples section this manual.

10.5.1 The Panel

To create the panel we add the following to the .xml file.

```
<pyvcp>

  <!-- the RPM meter -->
  <hbox>
    <relief>RAISED</relief>
    <bd>3</bd>
    <meter>
      <halpin>"spindle_rpm"</halpin>
      <text>"Spindle"</text>
      <subtext>"RPM"</subtext>
      <size>200</size>
      <min_>0</min_>
      <max_>3000</max_>
      <majorscale>500</majorscale>
      <minorscale>100</minorscale>
      <region1>0,10,"yellow"</region1>
    </meter>
  </hbox>

  <!-- the On Led -->
  <hbox>
  <relief>RAISED</relief>
  <bd>3</bd>
  <vbox>
  <relief>RAISED</relief>
  <bd>2</bd>
  <label>
  <text>"On"</text>
  <font>("Helvetica",18)</font>
  </label>
  <width>5</width>
   <hbox>
  <label width="2"/> <!-- used to center the led -->
  <rectled>
  <halpin>"on-led"</halpin>
  <height>"30"</height>
  <width>"30"</width>
  <on_color>"green"</on_color>
  <off_color>"red"</off_color>
```

```
  </rectled>
  </hbox>
  </vbox>

  <!-- the FWD Led -->
  <vbox>
    <relief>RAISED</relief>
    <bd>2</bd>
    <label>
      <text>"FWD"</text>
      <font>("Helvetica",18)</font>
      <width>5</width>
    </label>
    <label width="2"/>
    <rectled>
      <halpin>"fwd-led"</halpin>
      <height>"30"</height>
      <width>"30"</width>
      <on_color>"green"</on_color>
      <off_color>"red"</off_color>
    </rectled>
  </vbox>

  <!-- the REV Led -->
  <vbox>
  <relief>RAISED</relief>
  <bd>2</bd>
    <label>
      <text>"REV"</text>
      <font>("Helvetica",18)</font>
       <width>5</width>
    </label>
    <label width="2"/>
    <rectled>
      <halpin>"rev-led"</halpin>
      <height>"30"</height>
      <width>"30"</width>
      <on_color>"red"</on_color>
      <off_color>"green"</off_color>
    </rectled>
  </vbox>
  </hbox>
</pyvcp>
```

The above gives us a PyVCP panel that looks like the following.

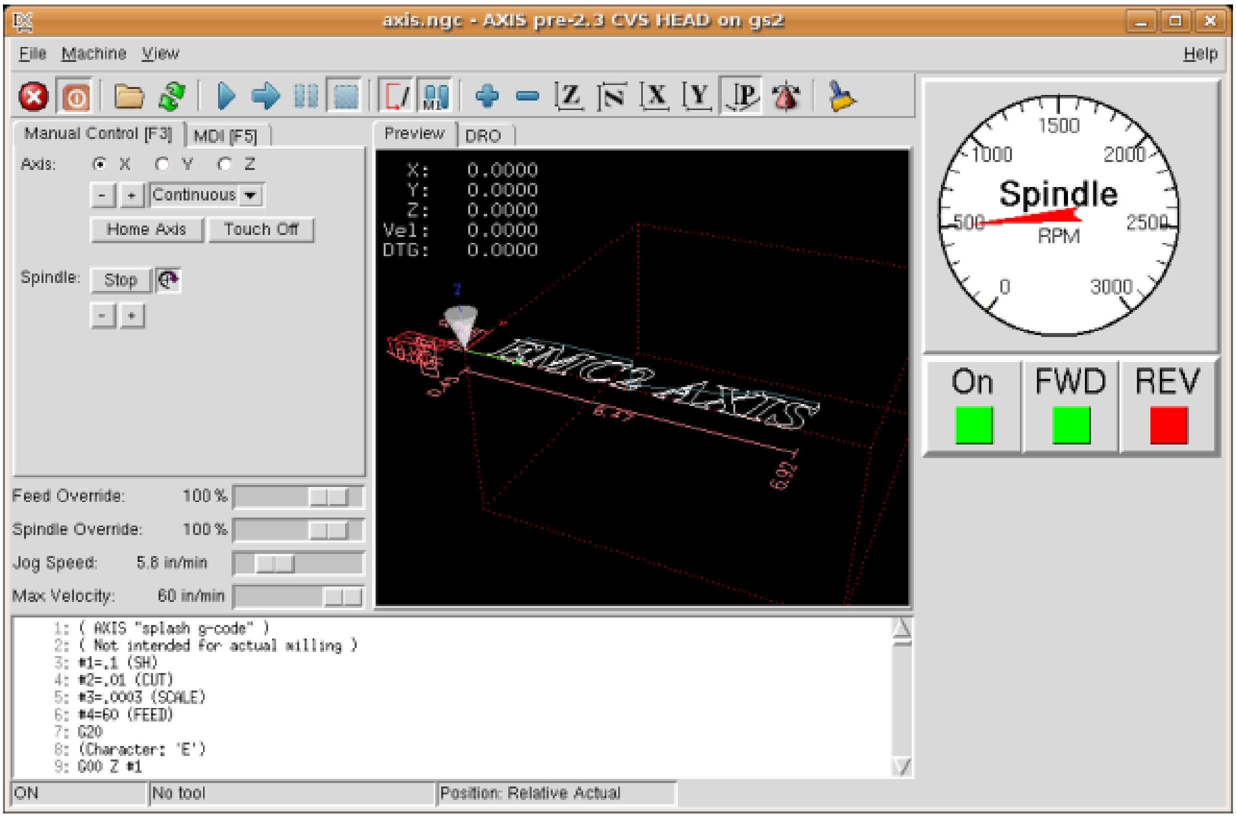


Figure 10.5: GS2 Panel

10.5.2 The Connections

To make it work we add the following code to the custom_postgui.hal file.

```
# display the rpm based on freq * rpm per hz
loadrt mult2
addf mult2.0 servo-thread
setp mult2.0.in1 28.75
net cypher_speed mult2.0.in0 <= spindle-vfd.frequency-out
net speed_out pyvcp.spindle_rpm <= mult2.0.out

# run led
net gs2-run => pyvcp.on-led

# fwd led
net gs2-fwd => pyvcp.fwd-led

# rev led
net running-rev spindle-vfd.spindle-rev => pyvcp.rev-led
```

Some of the lines might need some explanations. The fwd led line uses the signal created in the custom.hal file whereas the rev led needs to use the spindle-rev bit. You can't link the spindle-fwd bit twice so you use the signal that it was linked to.

Chapter 11

Glade Virtual Control Panel

11.1 What is GladeVCP?

GladeVCP is an LinuxCNC component which adds the ability to add a new user interface panel to LinuxCNC user interfaces like Axis or Touchy. Unlike PyVCP, GladeVCP is not limitied to displaying and setting HAL pins, as arbitrary actions can be executed in Python code - in fact, a complete LinuxCNC user interface could be built with GladeVCP and Python.

GladeVCP users the Glade WYSIWYG user interface editor, which makes it easy to create visually pleasing panels. It relies on the PyGTK bindings to the rich GTK+ widget set, and in fact all of these may be used in a GladeVCP application - not just the specialized widgets for interacting with HAL and LinuxCNC, which are documented here.

11.1.1 PyVCP versus GladeVCP at a glance

Both support the creation of panels with *HAL widgets* - user interface elements like LED's, buttons, sliders etc whose values are linked to a HAL pin, which in turn interfaces to the rest of LinuxCNC.

PyVCP:

- widget set: uses TkInter widgets
- user interface creation: "edit XML file / run result / evaluate looks" cycle
- no support for embedding user-defined event handling
- no LinuxCNC interaction beyond HAL pin I/O supported

GladeVCP:

- widget set: relies on the GTK+ widget set.
- user interface creation: uses the Glade WYSIWYG user interface editor
- any HAL pin change may be directed to call back into a user-defined Python event handler
- any GTK signal (key/button press, window, I/O, timer, network events) may be associated with user-defined handlers in Python
- direct LinuxCNC interaction: arbitrary command execution, like initiating MDI commands to call a G-code subroutine, plus support for status change operations through Action Widgets
- several independent GladeVCP panels may be run in different tabs
- separation of user interface appearance and functionality: change appearance without touching any code

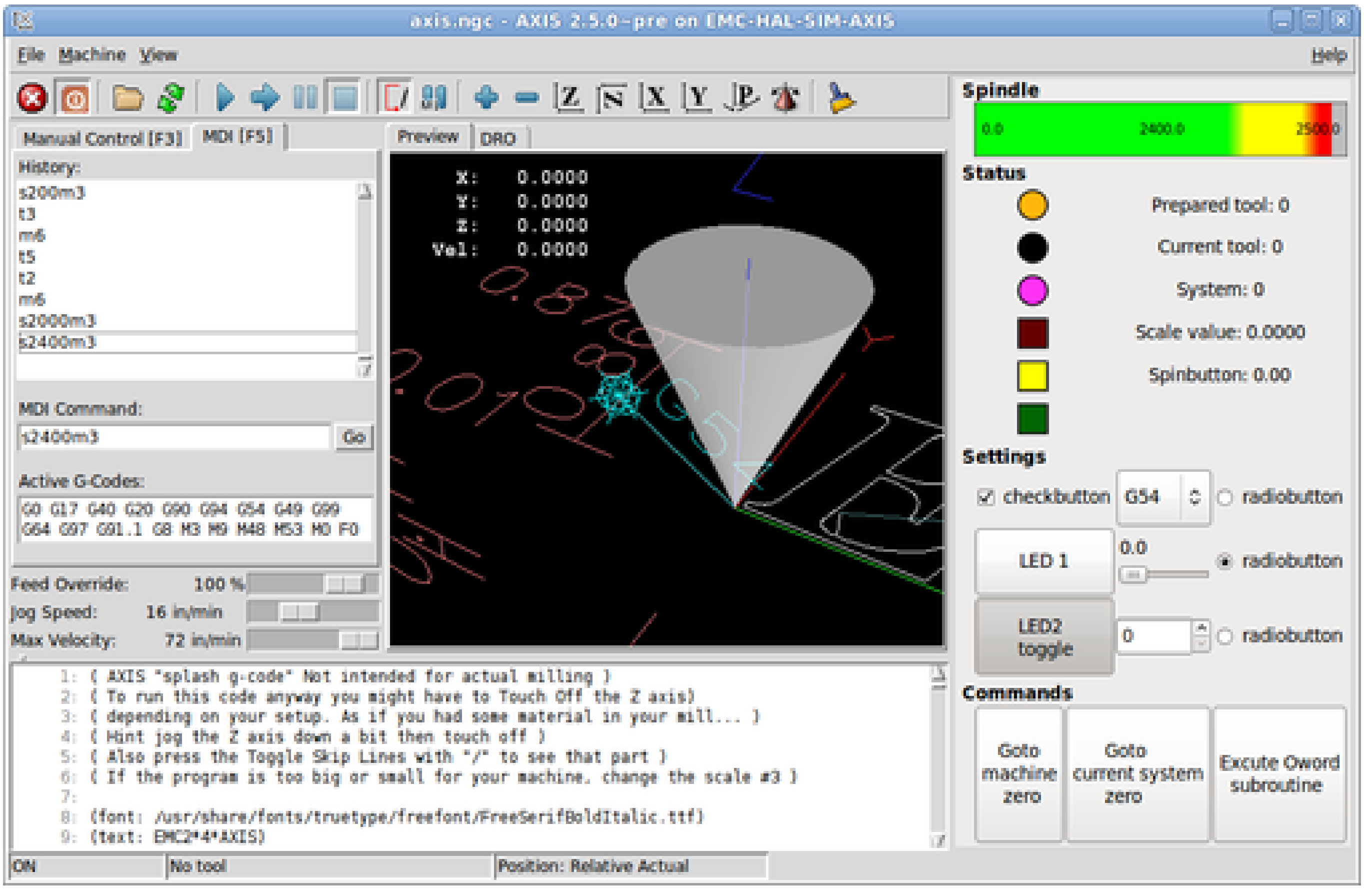
11.2 A Quick Tour with the Example Panel

GladeVCP panel windows may be run in three different setups:

- always visible integrated into Axis at the right side, exactly like PyVCP panels
- as a tab in Axis and Touchy; in Axis this would create a third tab besides the Preview and DRO tabs which must be raised explicitly
- as a standalone toplevel window, which can be iconifyed/deiconified independent of the main window.

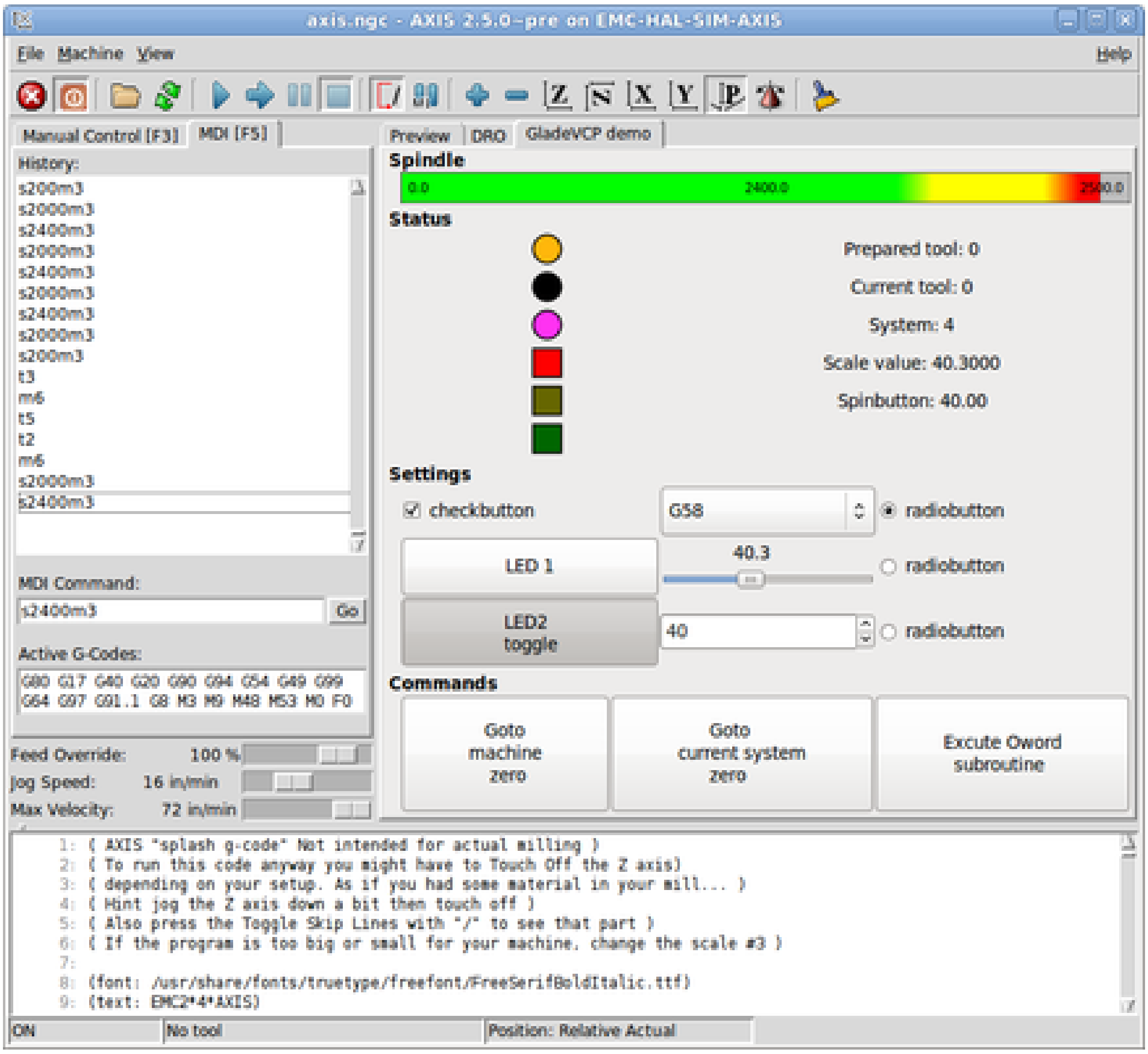
Run the sample GladeVCP panel integrated into Axis like PyVCP as follows:

```
$ cd configs/sim
$ emc gladevcp_panel.ini
```



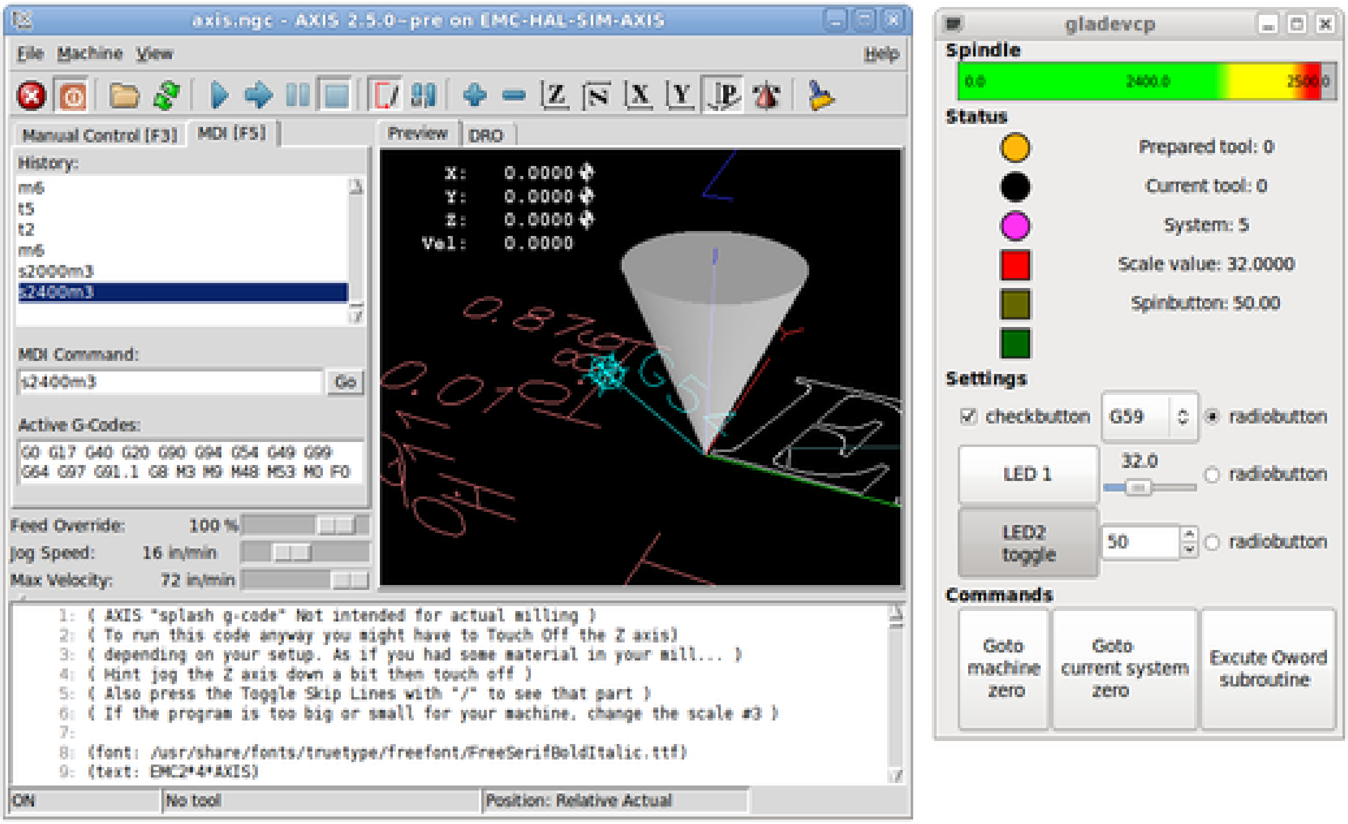
Run the same panel, but as a tab inside Axis:

```
$ cd configs/sim
$ emc gladevcp_tab.ini
```



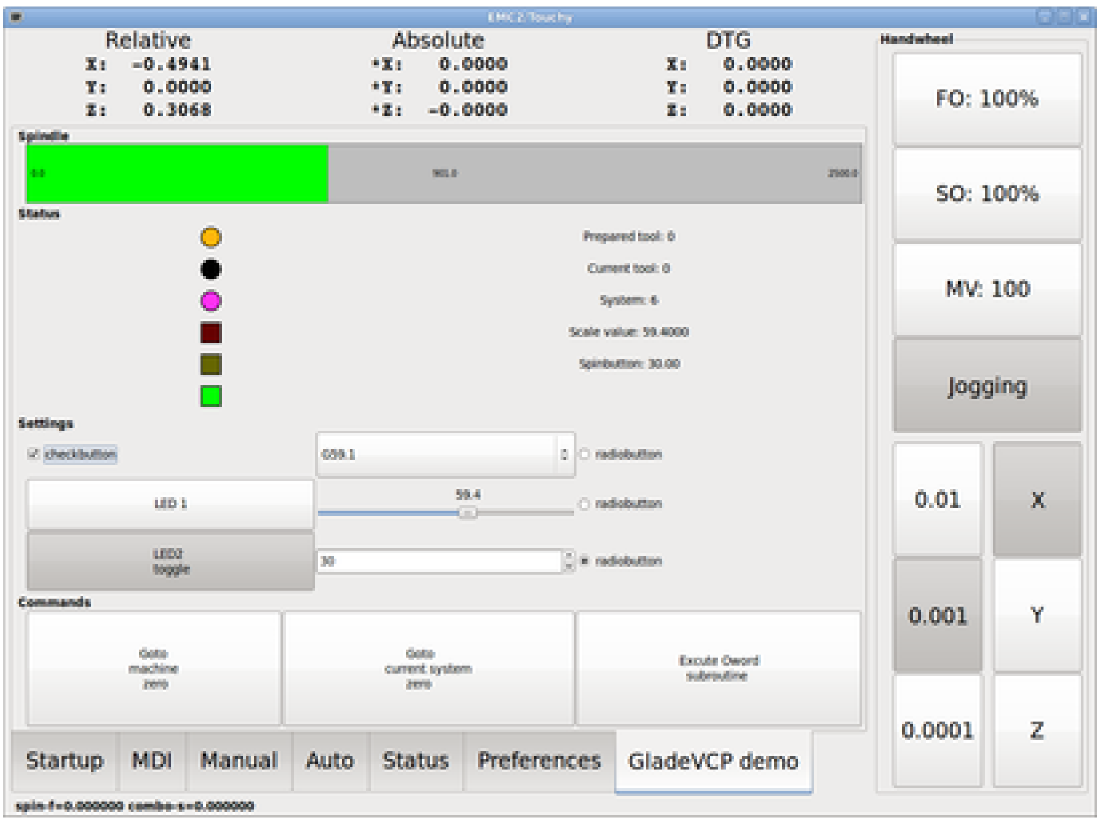
To run this panel as a standalone toplevel window besides Axis, just start Axis in the background and start gladevcp as follows:

```
$ cd configs/sim
$ emc axis.ini &
$ gladevcp -c gladevcp -u ../gladevcp/hitcounter.py -H ../gladevcp/manual-example.hal ../g
```



To run this panel inside *Touchy*:

```
$ cd configs/sim
$ emc gladevcp_touchy.ini
```



Functionally these setups are identical - they only differ in screen real estate requirements and visibility. Since it is possible to run several GladeVCP components in parallel (with different HAL component names), mixed setups are possible as well - for instance a panel on the right hand side, and one or more tabs for less-frequently used parts of the interface.

11.2.1 Exploring the example panel

While running Axis, explore *Show HAL Configuration* - you will find the *gladevcp* HAL component and may observe their pin values while interacting with the widgets in the panel. The HAL setup can be found in *configs/gladevcp/manual-example.hal*.

The example panel has two frames at the bottom. The panel is configured so that resetting ESTOP activates the Settings frame and turning the machine on enables the Commands frame at the bottom. The HAL widgets in the Settings frame are linked to LEDs and labels in the *Status* frame, and to the current and prepared tool number - play with them to see the effect. Executing the *T<toolnumber>* and *M6* commands in the MDI window will change the current and prepared tool number fields.

The buttons in the *Commands* frame are *MDI Action widgets* - pressing them will execute an MDI command in the interpreter. The third button *Execute Oword subroutine* is an advanced example - it takes several HAL pin values from the *Settings* frame, and passes them as parameters to the Oword subroutine. The actual parameters received by the routine are displayed by *(DEBUG,)* commands - see *configs/gladevcp/nc_files/oword.ngc* for the subroutine body.

To see how the panel is integrated into Axis, see the *[DISPLAY]GLADEVCP* statements in gladevcp_panel.ui, and the *[DISPLAY]EMBED** and *[HAL]POSTGUI_HALFILE* statements in gladevcp_tab.ini respectively.

11.2.2 Exploring the User Interface description

The user interface is created with the glade UI editor - to explore it, you need to have glade installed. To edit the user interface, run the command

```
$ glade configs/gladevcp/manual-example.ui
```

The center window shows the appearance of the UI. All user interface objects and support objects are found in the right top window, where you can select a specific widget (or by clicking on it in the center window). The properties of the selected widget are displayed, and can be changed, in the right bottom window.

To see how MDI commands are passed from the MDI Action widgets, explore the widgets listed under *Actions* in the top right window, and in the right bottom window, unter the *General* tab, the *MDI command* property.

11.2.3 Exploring the Python callback

See how a Python callback is integrated into the example:

- in glade, see the `hits` label widget (a plain GTK+ widget)
- in the `button1` widget, look at the *Signals* tab, and find the signal *pressed* associated with the handler *on_button_press*
- in ../gladevcp/hitcounter.py, see the method *on_button_press* and see how it sets the label property in the *hits* object

The is just touching upon the concept - the callback mechanism will be handled in more detail in the GladeVCP Programming section.

11.3 Creating and Integrating a Glade user interface

11.3.1 Prerequisite: Glade installation

To view or modify Glade UI files, you need glade installed - it is not needed just to run a GladeVCP panel. If the glade command is missing, install it with the command:

```
$ sudo apt-get install glade
```

Verify the version number to be greater than 3.6.7:

```
$ glade --version
glade3 3.6.7
```

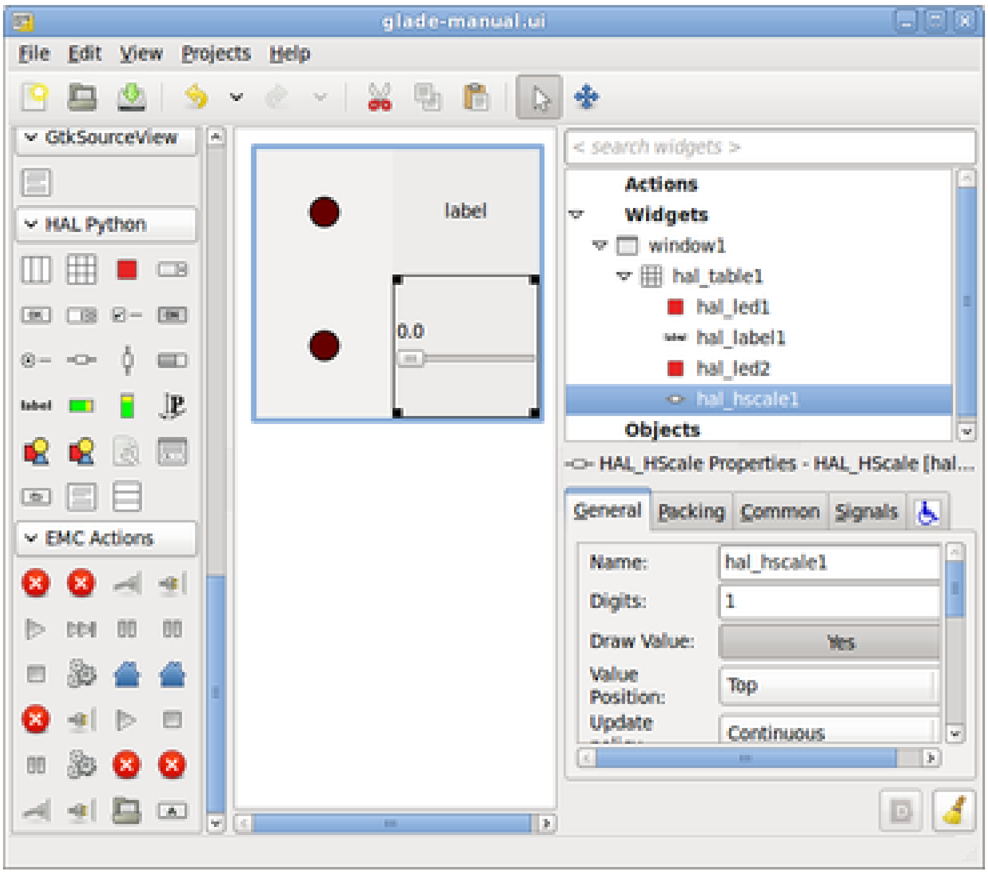
11.3.2 Running Glade to create a new user interface

This section just outlines the initial LinuxCNC-specific steps. For more information and a tutorial on glade, see http://glade.gnome.org. Some glade tips & tricks may also be found on youtube.

Either modify an existing UI component by running `glade <file>.ui` or start a new one by just running the `glade` command from the shell.

- If LinuxCNC was not installed from a package, the LinuxCNC shell environment needs to be set up with `. <emcdir>/scripts/` otherwise glade won't find the LinuxCNC-specific widgets.
- When asked for unsaved Preferences, just accept the defaults and hit *Close*.
- From *Toplevel* (left pane), pick *Window* (first icon) as top level window, which by default will be named *window1*. Do not change this name - GladeVCP relies on it.
- In the left tab, scroll down and expand *HAL Python* and *EMC Actions*.
- add a container like a HAL_Box or a HAL_Table from *HAL Python* to the frame
- pick and place some elements like LED, button, etc. within a container

This will look like so:



Glade tends to write a lot of messages to the shell window, which mostly can be ignored. Select *File→Save as*, give it a name like *myui.ui* and make sure it's saved as *GtkBuilder* file (radio button left bottom corner in Save dialog). GladeVCP will also process the older *libglade* format correctly but there is no point in using it. The convention for GtkBuilder file extension is *.ui*.

11.3.3 Testing a panel

You're now ready to give it a try (while LinuxCNC, e.g. Axis is running) it with:

```
gladevcp myui.ui
```

GladeVCP creates a HAL component named like the basename of the UI file - *myui* in this case - unless overriden by the `-c <component name>` option. If running Axis, just try *Show HAL configuration* and inspect its pins.

You might wonder why widgets contained a *HAL_Hbox* or *HAL_Table* appear greyed out (inactive). HAL containers have an associated HAL pin which is off by default, which causes all contained widgets to render inactive. A common use case would be to associate these container HAL pins with `halui.machine.is-on` or one of the `halui.mode.` signals, to assure some widgets appear active only in a certain state.

To just activate a container, execute the HAL command `setp gladevcp.<container-name> 1`.

11.3.4 Preparing the HAL command file

The suggested way of linking HAL pins in a GladeVCP panel is to collect them in a separate file with extension `.hal`. This file is passed via the `POSTGUI_HALFILE=` option in the `HAL` section of your ini file.

Caution
Do not add the GladeVCP HAL command file to the Axis `[HAL]HALFILE=` ini section, this will not have the desired effect - see the following sections.

11.3.5 Integrating into Axis like PyVCP

Place the GladeVCP panel in the righthand side panel by specifying the following in the ini file:

```
[DISPLAY]
# add GladeVCP panel where PyVCP used to live:
GLADEVCP= -u ../gladevcp/hitcounter.py ../gladevcp/manual-example.ui

[HAL]
# HAL commands for GladeVCP components in a tab must be executed via POSTGUI_HALFILE
POSTGUI_HALFILE =  ../gladevcp/manual-example.hal

[RS274NGC]
# gladevcp Demo specific Oword subs live here
SUBROUTINE_PATH = ../gladevcp/nc_files/
```

The HAL component name of a GladeVCP application started with the the GLADEVCP option is fixed: `gladevcp`. The command line actually run by Axis in the above configuration is as follows:

```
halcmd loadusr -Wn gladevcp gladevcp -c gladevcp -x {XID} <arguments to GLADEVCP>
```

This means you may add arbitrary gladevcp options here, as long as they dont collide with the above command line options.

Note
The `[RS274NGC]SUBROUTINE_PATH=` option is only set so the example panel will find the Oword subroutine for the MDI Command widget. It might not be needed in your setup.

11.3.6 Integrating into Axis as a tab next to DRO and Preview

To do so, edit your .ini file and add to the DISPLAY and HAL sections of ini file as follows:

```
[DISPLAY]
# add GladeVCP panel as a tab next to Preview/DRO:
EMBED_TAB_NAME=GladeVCP demo
EMBED_TAB_COMMAND=halcmd loadusr -Wn gladevcp gladevcp -c gladevcp -x {XID} -u ../gladevcp/ ↩
    hitcounter.py ../gladevcp/manual-example.ui

[HAL]
# HAL commands for GladeVCP components in a tab must be executed via POSTGUI_HALFILE
POSTGUI_HALFILE =  ../gladevcp/manual-example.hal

[RS274NGC]
# gladevcp Demo specific Oword subs live here
SUBROUTINE_PATH = ../gladevcp/nc_files/
```

Note the *halcmd loadusr* way of starting the tab command - this assures that *POSTGUI_HALFILE* will only be run after the HAL component is ready. In rare cases you might run a a command here which uses a tab but does not have an associated HAL component. Such a command can be started without *halcmd loadusr*, and this signifies to Axis that it does not have to wait for a HAL component since there is none.

When changing the component name in the above example, note that the names used in `-Wn <component>` and `-c <component>` must be identical.

Try it out by running Axis - there should be a new tab called *GladeVCP demo* near the DRO tab. Select that tab, you should see the example panel nicely fit within Axis.

> **Note**
> Make sure the UI file is the last option passed to GladeVCP in both the `GLADEVCP=` and `EMBED_TAB_COMMAND=` statements.

11.3.7 Integrating into Touchy

To do add a GladeVCP tab to *Touchy*, edit your .ini file as follows:

```
[DISPLAY]
# add GladeVCP panel as a tab
EMBED_TAB_NAME=GladeVCP demo
EMBED_TAB_COMMAND=gladevcp -c gladevcp -x {XID} -u ../gladevcp/hitcounter.py -H ../gladevcp ↩
    /gladevcp-touchy.hal  ../gladevcp/manual-example.ui

[RS274NGC]
# gladevcp Demo specific Oword subs live here
SUBROUTINE_PATH = ../gladevcp/nc_files/
```

Note the following differences to the Axis tab setup:

- The HAL command file is slightly modified since *Touchy* does not use the *halui* components so its signals are not available and some shortcuts have been taken.
- there is no *POSTGUI_HALFILE=* ini option, but passing the HAL command file on the *EMBED_TAB_COMMAND=* line is ok
- the *halcmd loaduser -Wn …* incantation is not needed.

11.4 GladeVCP command line options

See also *man gladevcp* . These are the gladevcp command line options:

Usage: gladevcp [options] myfile.ui

Options:

-h, --help
show this help message and exit

-c NAME
Set component name to NAME. Default is base name of UI file

-d
Enable debug output

-g GEOMETRY
Set geometry WIDTHxHEIGHT+XOFFSET+YOFFSET. Values are in pixel units, XOFFSET/YOFFSET is referenced from top left of screen. Use -g WIDTHxHEIGHT for just setting size or -g +XOFFSET+YOFFSET for just position

-H FILE
execute hal statements from FILE with halcmd after the component is set up and ready

-m MAXIMUM
force panel window to maximize. Together with the -g geometry option one can move the panel to a second monitor and force it to use all of the screen

-t THEME
set gtk theme. Default is system theme. Different panels can have different themes. An example theme can be found in the EMC Wiki.

-x XID
Re-parent GladeVCP into an existing window XID instead of creating a new top level window

-u FILE
Use File's as additional user defined modules with handlers

-U USEROPT
pass USEROPTs to Python modules

11.5 HAL Widget reference

GladeVcp includes a collection of Gtk widgets with attached HAL pins called HAL Widgets, intended to control, display or otherwise interact with the LinuxCNC HAL layer. They are intended to be used with the Glade user interface editor. With proper installation, the HAL Widgets should show up in Glade's *HAL Python* widget group. Many HAL specific fields in the Glade *General* section have an associated mouse-over tool tip.

HAL signals come in two variants, bits and numbers. Bits are off/on signals. Numbers can be "float", "s32" or "u32". For more information on HAL data types see the HAL manual. The GladeVcp widgets can either display the value of the signal with an indicator widget, or modify the signal value with a control widget. Thus there are four classes of GladeVcp widgets that you can connect to a HAL signal. Another class of helper widgets allow you to organize and label your panel.

- Widgets for indicating "bit" signals: HAL_LED
- Widgets for controlling "bit" signals: HAL_Button, HAL_RadioButton, HAL_CheckButton
- Widgets for indicating "number" signals: HAL_Label, HAL_ProgressBar, HAL_HBar, HAL_VBar, HAL_Meter
- Widgets for controlling "number" signals: HAL_SpinButton, HAL_HScale, HAL_VScale

- Helper widgets: HAL_Table, HAL_HBox
- Tool Path preview: HAL_Gremlin

HAL Widgets inherit methods, properties and signals from the underlying Gtk widgets, so it is helpful to consult the GTK+ and PyGTK bindings documentation as well.

11.5.1 Widget and HAL pin naming

Most HAL widgets have a single associated HAL pin with the same name as the widget (glade: General→Name).

Exceptions to this rule currently are.

- *HAL_Spinbutton* and *HAL_ComboBox*, which have two pins: a `<widgetname>-f` (float) and a `<widgetname>-s` (s32) pin
- *HAL_ProgressBar*, which has a `<widgetname>-value` input pin, and a `<widgetname>-scale` input pin.

11.5.2 Setting pin and widget values

As a general rule, if you need to set a HAL output widget's value from Python code, do so by calling the underlying Gtk *setter* (e.g. `set_active()`, `set_value()`) - do not try to set the associated pin's value by `halcomp[pinname] = value` directly because the widget will not take notice of the change!.

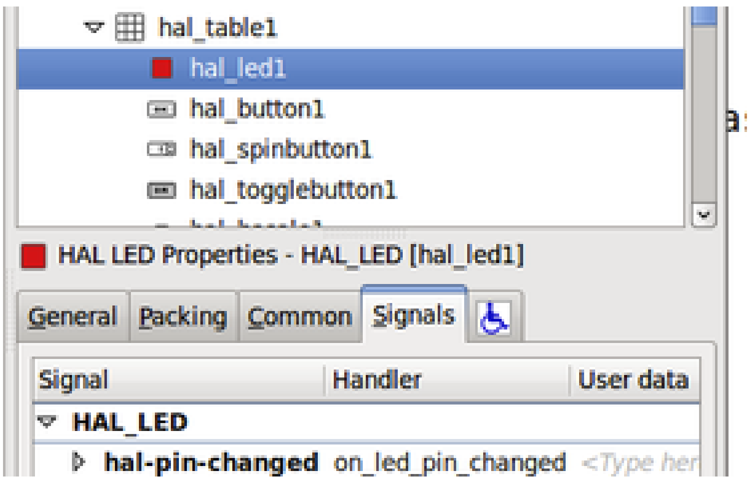
It might be tempting to *set HAL widget input pins* programmatically. Note this defeats the purpose of an input pin in the first place - it should be linked to, and react to signals generated by other HAL components. While there is currently no write protection on writing to input pins in HAL Python, this doesn't make sense. You might use setp pinname value in the associated halfile for testing though.

It is perfectly OK to set an output HAL pin's value with `halcomp[pinname] = value` provided this HAL pin is not associated with a widget, that is, has been created by the `hal_glib.GPin(halcomp.newpin(<name>,<type>,<direction>)` method (see GladeVCP Programming for an example).

11.5.3 The hal-pin-changed signal

Event-driven programming means that the UI tells your code when "something happens" - through a callback, like when a button was pressed. The output HAL widgets (those which display a HAL pin's value) like LED, Bar, VBar, Meter etc, support the *hal-pin-changed* signal which may cause a callback into your Python code when - well, a HAL pin changes its value. This means there's no more need for permanent polling of HAL pin changes in your code, the widgets do that in the background and let you know.

Here is an example how to set a `hal-pin-changed` signal for a HAL_LED in the Glade UI editor:

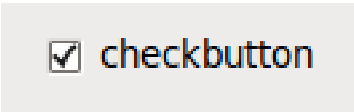


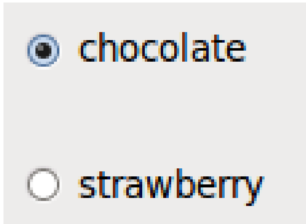
The example in `configs/gladevcp/examples/complex` shows how this is handled in Python.

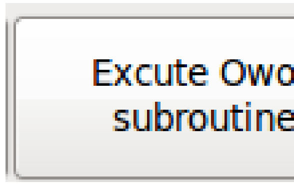
11.5.4 Buttons

This group of widgets are derived from various Gtk buttons and consists of HAL_Button, HAL_ToggleButton, HAL_RadioButton and CheckButton widgets. All of them have a single output BIT pin named identical to the widget. Buttons have no additional properties compared to their base Gtk classes.

- HAL_Button: instantaneous action, does not retain state. Important signal: `pressed`
- HAL_ToggleButton, HAL_CheckButton: retains on/off state. Important signal: `toggled`
- HAL_RadioButton: a one-of-many group. Important signal: `toggled` (per button).
- Important common methods: `set_active()`, `get_active()`
- Important properties: `label`, `image`

Check button: 

Radio buttons: 

Toggle button: 

.

Tip
Defining radio button groups in Glade:

- decide on default active button
- in the other button's *General→Group* select the default active button's name in the *Choose a Radio Button in this project* dialog.

See `configs/gladevcp/by-widget/radiobutton` for a GladeVCP application and UI file for working with radio buttons.

11.5.5 Scales

HAL_HScale and HAL_VScale are derived from the GtkHScale and GtkVScale respectively. They have one output FLOAT pin with name equal to widget name. Scales have no additional properties.

To make a scale useful in Glade, add an *Adjustment* (General→Adjustment→New or existing adjustment) and edit the adjustment object. It defines the default/min/max/increment values. Also, set adjustment *Page size* and *Page increment* to zero to avoid warnings.

Example HAL_HScale: 

.

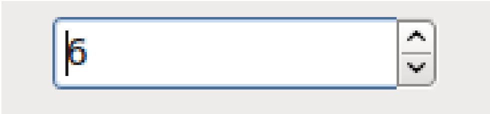
11.5.6 SpinButton

HAL SpinButton is derived from GtkSpinButton and holds two pins:

<widgetname>-f
out FLOAT pin

<widgetname>-s
out S32 pin

To be useful, Spinbuttons need an adjustment value like scales, see above.

Example SpinButton: 

.

11.5.7 Label

HAL_Label is a simple widget based on GtkLabel which represents a HAL pin value in a user-defined format.

label_pin_type
The pin's HAL type (0:S32, 1:float, 2:U32), see also the tooltip on 'General→HAL pin type '(note this is different from PyVCP which has three label widgets, one for each type).

text_template
Determines the text displayed - a Python format string to convert the pin value to text. Defaults to `%s` (values are converted by the str() function) but may contain any legit as an argument to Pythons format() method.
Example: `Distance:  %.03f` will display the text and the pin value with 3 fractional digits padded with zeros for a FLOAT pin.

11.5.8 Containers: HAL_HBox and HAL_Table

Compared to their Gtk counterparts they have one input BIT pin which controls if their child widgets are sensitive or not. If the pin is low then child widgets are inactive which is the default.

Tip
If you find some part of your GladeVCP application is *grayed out* (insensitive), see whether a container's pin is unset.

11.5.9 LED

The hal_led simulates a real indicator LED. It has a single input BIT pin which controls it's state: ON or OFF. LEDs have several properties which control their look and feel:

on_color
a String defining ON color of LED. May be any valid gtk.gdk.Color name. Not working on Ubuntu 8.04.

off_color
String defining OFF color of LED. May be any valid gtk.gdk.Color name or special value `dark`. `dark` means that OFF color will be set to 0.4 value of ON color. Not working on Ubuntu 8.04.

pick_color_on, pick_color_off
Colors for ON and OFF states may be represented as `#RRRRGGGGBBBB` strings. These are optional properties which have precedence over `on_color` and `off_color`.

led_size
LED radius (for square - half of LED's side)

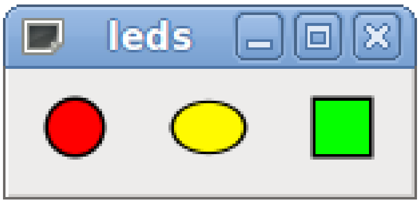
led_shape
LED Shape. Valid values are 0 for round, 1 for oval and 2 for square shapes.

led_blink_rate
if set and LED is ON then it's blinking. Blink period is equal to "led_blink_rate" specified in milliseconds.

As an input widget, LED also supports the `hal-pin-changed signal`. If you want to get a notification in your code when the LED's HAL pin was changed, then connect this signal to a handler, for example `on_led_pin_changed` and provide the handler as follows:

```
def on_led_pin_changed(self,hal_led,data=None):
    print "on_led_pin_changed() - HAL pin value:",hal_led.hal_pin.get()
```

This will be called at any edge of the signal and also during program start up to report the current value.

Example LEDs: 

.

11.5.10 ProgressBar

Note
This widget might go away. Use the HAL_HBar and HAL_VBar widgets instead.

The HAL_ProgressBar is derived from gtk.ProgressBar and has two float HAL input pins:

<widgetname>
the current value to be displayed

<widgetname>-scale
the maximum absolute value of input

It has the following properties:

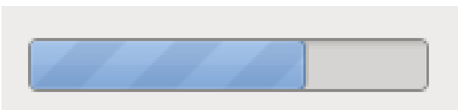
scale
value scale. set maximum absolute value of input. Same as setting the <widgetname>.scale pin. A float, range from $-2^{24 \text{ to } +2}24$.

green_limit
green zone limit lower limit

yellow_limit
yellow zone limit lower limit

red_limit
red zone limit lower limit

text_template
Text template to display the current value of the `<widgetname>` pin. Python formatting may be used for dict `{"value":valu`

Example HAL_ProgressBar: 
.

11.5.11 ComboBox

HAL_ComboBox is derived from gtk.ComboBox. It enables choice of a value from a dropdown list.

It exports two HAL pins:

<widgetname>-f
: the current value, type FLOAT

<widgetname>-s
: the current value, type S32

It has the following property which can be set in Glade:

column
: the column index, type S32, defaults to -1, range from -1..100 .

In default mode this widgets sets the pins to the index of the chosen list entry. So if your widget has three labels, it may only assume values 0,1 and 2.

In column mode (column > -1), the value reported is chosen from the ListStore array as defined in Glade. So typically your widget definition would have two columns in the ListStore , one with text displayed in the dropdown, and an int or float value to use for that choice.

There's an example in `configs/gladevcp/by-widget/combobox/combobox.{py,ui}` which uses column mode to pick a float value from the ListStore.

If you're confused like me about how to edit ComboBox ListStores and CellRenderer, see http://www.youtube.com/watch?v=Z5_F-rW2cL8.

11.5.12 Bars

HAL Bar and VBar widgets for horizontal and vertical bars representing float values. They have one input FLOAT hal pin. Both bars have the following properties:

invert
: Swap min and max direction. An inverted HBar grows from right to left, an inverted VBar from top to bottom.

min, max
: Minimum and maximum value of desired range. It is not an error condition if the current value is outside this range.

zero
: Zero point of range. If it's inside of min/max range then the bar will grow from that value and not from the left (or right) side of the widget. Useful to represent values that may be both positive or negative.

force_width, force_height
: Forced width or height of widget. If not set then size will be deduced from packing or from fixed widget size and bar will fill whole area.

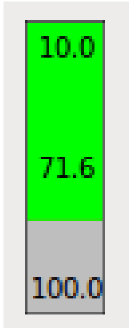
text_template
: Like in Label sets text format for min/max/current values. Can be used to turn off value display.

bg_color
: Background (inactive) color of bar.

z0_color, z1_color, z2_color
: Colors of different value zones. Defaults are `green`, `yellow` and `red`. For description of zones see `z*_border` properties.

z0_border, z1_border
Define up bounds of color zones. By default only one zone is enabled. If you want more then one zone set `z0_border` and `z1_border` to desired values so zone 0 will fill from 0 to first border, zone 1 will fill from first to second border and zone 2 — from last border to 1. Borders are set as fractions, values from 0 to 1.

Horizontal bar: 

Vertical bar: 

.

11.5.13 Meter

HAL Meter is a widget similar to PyVCP meter - it represents a float value and has one input FLOAT hal pin. HAL Meter has the following properties:

min, max
Minimum and maximum value of desired range. It is not an error condition if the current value is outside this range.

force_size
Forced diameter of widget. If not set then size will be deduced from packing or from fixed widget size and meter will fill all available space with respect to aspect ratio.

text_template
Like in Label sets text format for current value. Can be used to turn off value display.

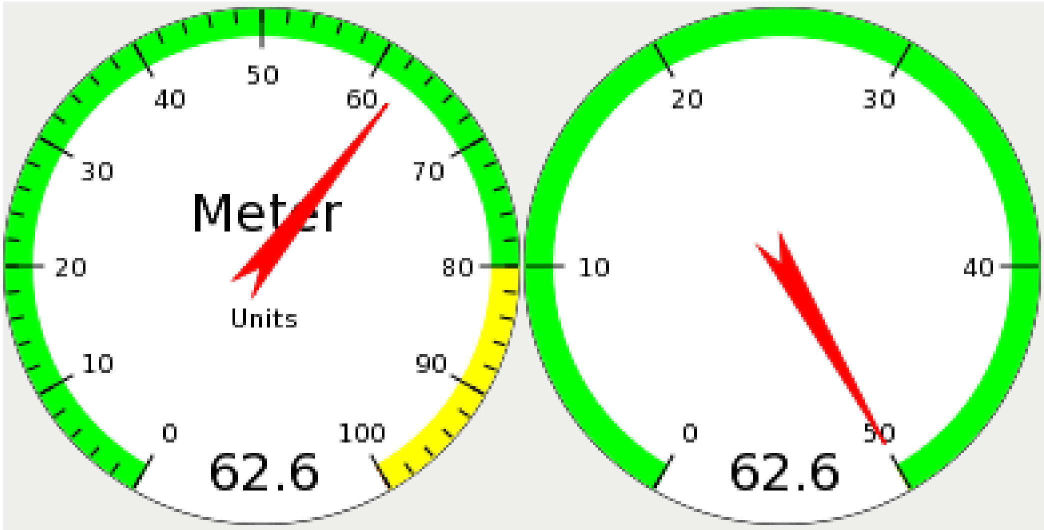
label
Large label above center of meter.

sublabel
Small label below center of meter.

bg_color
Background color of meter.

z0_color, z1_color, z2_color
Colors of different value zones. Defaults are `green`, `yellow` and `red`. For description of zones see `z*_border` properties.

z0_border, z1_border
Define up bounds of color zones. By default only one zone is enabled. If you want more then one zone set `z0_border` and `z1_border` to desired values so zone 0 will fill from min to first border, zone 1 will fill from first to second border and zone 2 — from last border to max. Borders are set as values in range min-max.

Example HAL Meters: 

.

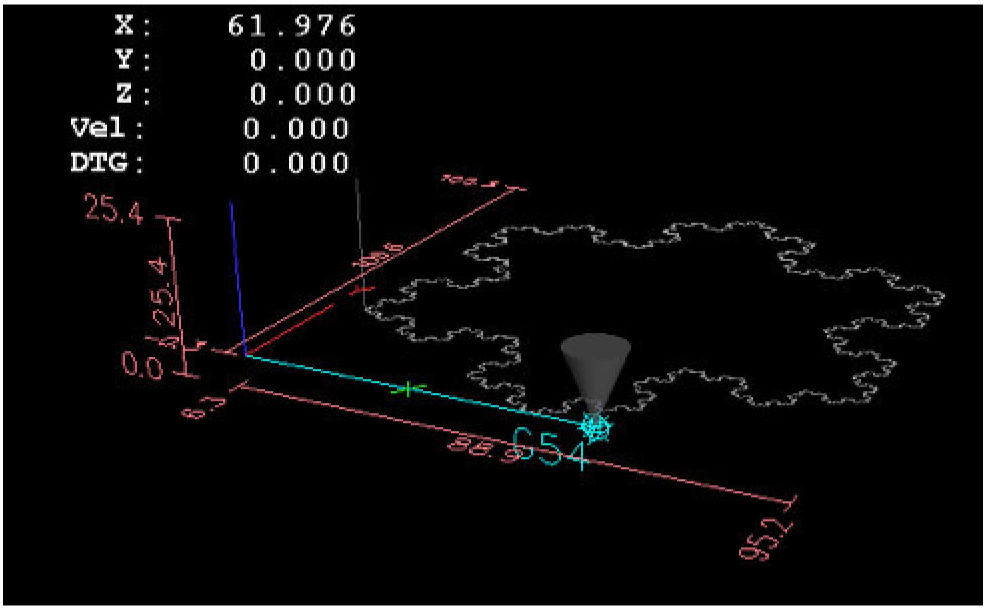
11.5.14 Gremlin tool path preview for .ngc files

Gremlin is a plot preview widget similar to the Axis preview window. It assumes a running LinuxCNC environment like Axis or Touchy. To connect to it, inspects the INI_FILE_NAME environment variable. Gremlin displays the current .ngc file - it does monitor for changes and reloads the ngc file if the file name in Axis/Touchy changes. If you run it in a GladeVCP application when LinuxCNC is not running, you might get a traceback because the Gremlin widget can't find LinuxCNC status, like the current file name.

Gremlin does not export any HAL pins. It has the following properties:

view
: may be any of `x`, `y`, `z`, `p` (perspective) . Defaults to `z` view.

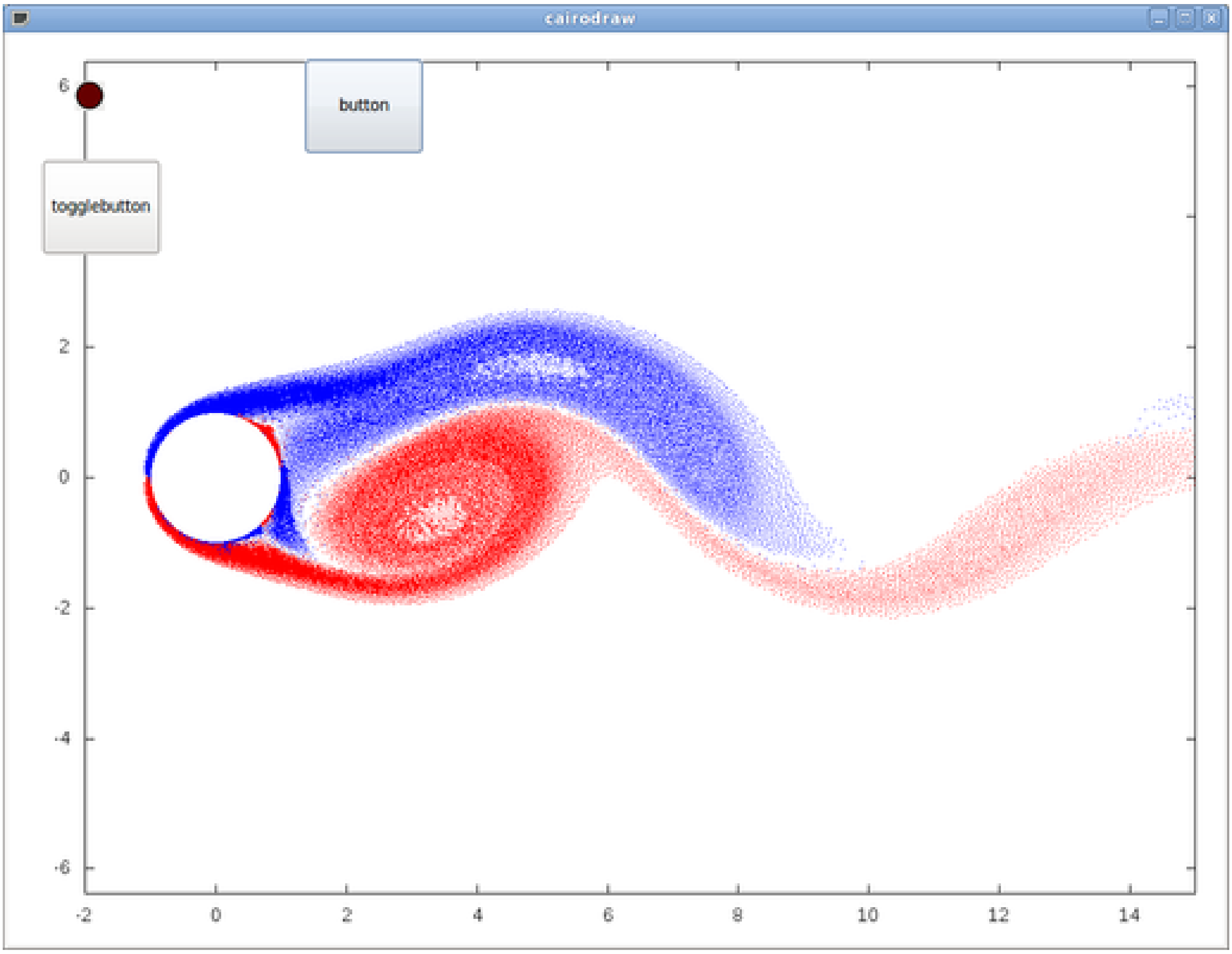
enable_dro
: boolean; whether to draw a DRO on the plot or not. Defaults to `True`.

Example: 

11.5.15 Animated function diagrams: HAL widgets in a bitmap

For some applications it might be desirable to have background image - like a functional diagram - and position widgets at appropriate places in that diagram. A good combination is setting a bitmap background image, like from a .png file, making the gladevcp window fixed-size, and use the glade Fixed widget to position widgets on this image.

The code for the below example can be found in `configs/gladevcp/animated-backdrop`:



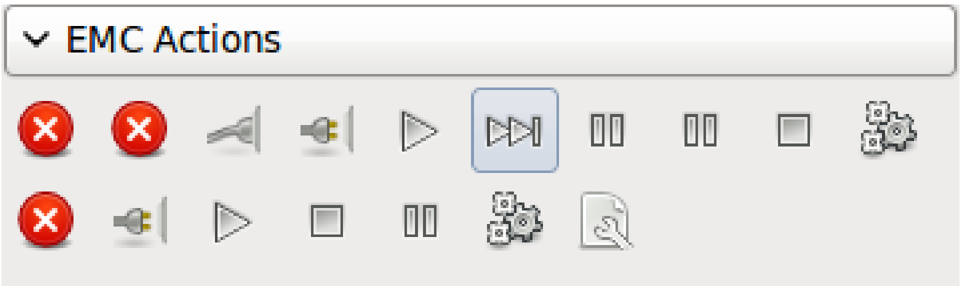
11.6 Action Widgets reference

GladeVcp includes a collection of "canned actions" called EMC Action Widgets for the Glade user interface editor. Other than HAL widgets, which interact with HAL pins, EMC Actions interact with LinuxCNC and the G-code interpreter.

EMC Action Widgets are derived from the Gtk.Action widget. The Action widget in a nutshell:

- it is an object available in Glade
- it has no visual appearance by itself
- it's purpose: associate a visible, sensitive UI component like menu, toolbutton, button with a command. See these widget's *General→Related Action* property.
- the "canned action" will be executed when the associated UI component is triggered (button press, menu click..)
- it provides an easy way to execute commands without resorting to Python programming.

The appearance of EMC Actions in Glade is roughly as follows:



Tooltip hovers provide a description.

11.6.1 EMC Action widgets

EMC Action widgets are one-shot type widgets. They implement a single action and are for use in simple buttons, menu entries or radio/check groups.

11.6.2 EMC ToggleAction widgets

These are bi-modal widgets. They implement two actions or use a second (usually pressed) state to indicate that currently an action is running. Toggle actions are aimed for use in ToggleButtons, ToggleToolButtons or toggling menu items. A simplex example is the ESTOP toggle button.

Currently the following widgets are available:

- The ESTOP toggle sends ESTOP or ESTOP_RESET commands to LinuxCNC depending on it's state.
- The ON/OFF toggle sends STATE_ON and STATE_OFF commands.
- Pause/Resume sends AUTO_PAUSE or AUTO_RESUME commands.

The following toggle actions have only one associated command and use the *pressed* state to indicate that the requested operation is running:

- The Run toggle sends an AUTO_RUN command and waits in the pressed state until the interpreter is idle again.
- The Stop toggle is inactive until the interpreter enters the active state (is running G-code) and then allows user to send AUTO_ABORT command.
- The MDI toggle sends given MDI command and waits for its completion in *pressed* inactive state.

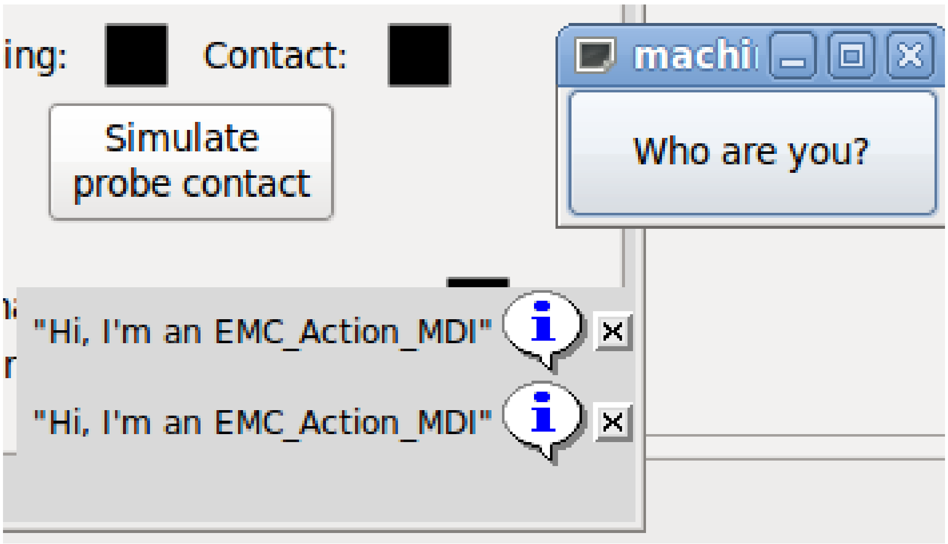
11.6.3 The Action_MDI Toggle and Action_MDI widgets

These widgets provide a means to execute arbitrary MDI commands. The Action_MDI widget does not wait for command completion as the Action_MDI Toggle does, which remains disabled until command complete.

11.6.4 A simple example: Execute MDI command on button press

`configs/gladevcp/mdi-command-example/whoareyou.ui` is a Glade UI file which conveys the basics:

Open it in Glade and study how it's done. Start Axis, and then start this from a terminal window with `gladevcp whoareyou.ui`. See the `hal_action_mdi1` Action and it's `MDI command` property - this just executes `(MSG, "Hi, I'm an EMC_Action_MI` so there should be a message popup in Axis like so:



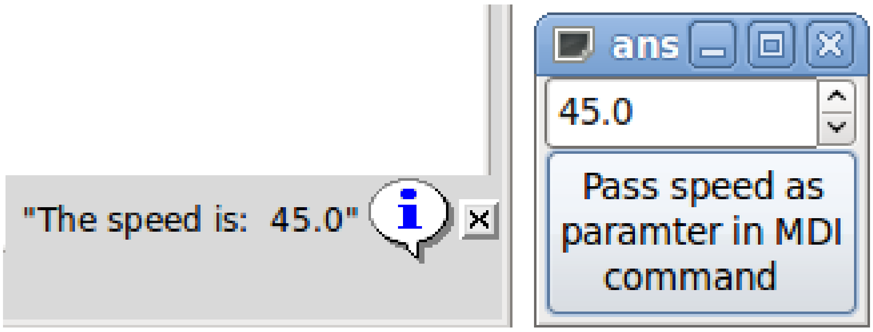
You'll notice that the button associated with the Action_MDI action is grayed out if the machine is off, in E-Stop or the interpreter is running. It will automatically become active when the machine is turned on and out of E-Stop, and the program is idle.

11.6.5 Parameter passing with Action_MDI and ToggleAction_MDI widgets

Optionally, *MDI command* strings may have parameters substituted before they are passed to the interpreter. Parameters currently may be names of HAL pins in the GladeVCP component. This is how it works:

- assume you have a *HAL SpinBox* named `speed`, and you want to pass it's current value as a parameter in an MDI command.
- The HAL SpinBox will have a float-type HAL pin named speed-f (see HalWidgets description).
- To substitute this value in the MDI command, insert the HAL pin name enclosed like so: `${pin-name}`
- for the above HAL SpinBox, we could use `(MSG, "The speed is:  ${speed-f}")` just to show what's happening.

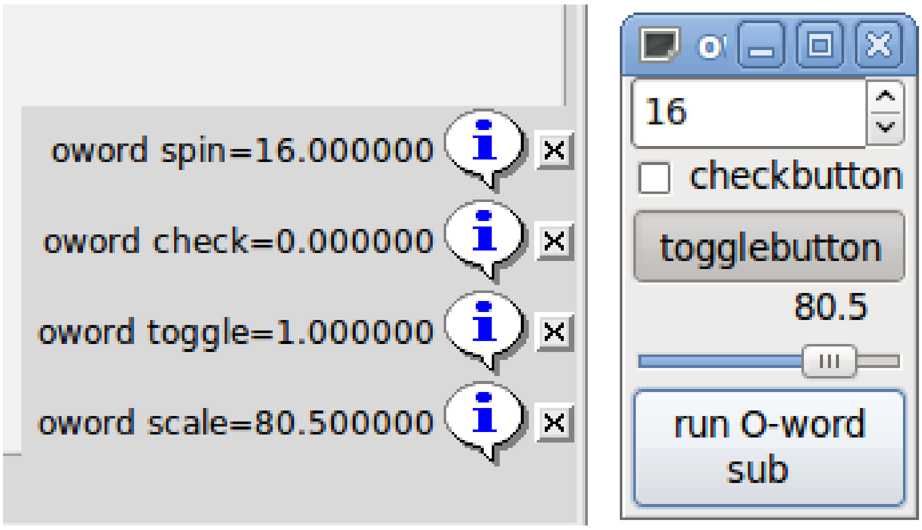
The example UI file is `configs/gladevcp/mdi-command-example/speed.ui`. Here's what you get when running it:



11.6.6 An advanced example: Feeding parameters to an O-word subroutine

It's perfectly OK to call an O-word subroutine in an MDI command, and pass HAL pin values as actual parameters. An example UI file is in `configs/gladevcp/mdi-command-example/owordsub.ui`.

Place `configs/gladevcp/nc_files/oword.ngc` so Axis can find it, and run `gladevcp owordsub.ui` from a terminal window. This looks like so:



11.6.7 Preparing for an MDI Action, and cleaning up afterwards

The LinuxCNC G-Code interpreter has a single global set of variables, like feed, spindle speed, relative/absolute mode and others. If you use G code commands or O-word subs, some of these variables might get changed by the command or subroutine - for example, a probing subroutine will very likely set the feed value quite low. With no further precautions, your previous feed setting will be overwritten by the probing subroutine's value.

To deal with this surprising and undesirable side effect of a given O-word subroutine or G-code statement executed with an LinuxCNC ToggleAction_MDI, you might associate pre-MDI and post-MDI handlers with a given LinuxCNC ToggleAction_MDI. These handlers are optional and provide a way to save any state before executing the MDI Action, and to restore it to previous values afterwards. The signal names are `mdi-command-start` and `mdi-command-stop`; the handler names can be set in Glade like any other handler.

Here's an example how a feed value might be saved and restored by such handlers (note that LinuxCNC command and status channels are available as `self.emc` and `self.stat` through the EMC_ActionBase class:

```
def on_mdi_command_start(self, action, userdata=None):
    action.stat.poll()
    self.start_feed = action.stat.settings[1]

def on_mdi_command_stop(self, action, userdata=None):
    action.emc.mdi('F%.1f' % (self.start_feed))
    while action.emc.wait_complete() == -1:
        pass
```

Only the Action_MDI Toggle widget supports these signals.

Note
In a later release of LinuxCNC, the new M-codes M70-M72 are available which make it saving state before a subroutine call, and restoring state on return much easier.

11.6.8 Using the LinuxCNC Stat object to deal with status changes

Many actions depend on LinuxCNC status - is it in manual, MDI or auto mode? is a program running, paused or idle? You cannot start an MDI command while a G-code program is running, so this needs to be taken care of. Many LinuxCNC actions take care of this themselves, and related buttons and menu entries are deactivated when the operation is currently impossible.

When using Python event handlers - which are at a lower level than Actions - one needs to take care of dealing with status dependencies oneself. For this purpose, there's the LinuxCNC Stat widget: to associate LinuxCNC status changes with event handlers.

LinuxCNC Stat has no visible component - you just add it to your UI with Glade. Once added, you can associate handlers with its following signals:

- state-related: emitted when E-Stop condition occurs, is reset, machine is turned on, or is turned off
  - `state-estop`
  - `state-estop-reset`
  - `state-on`,
  - `state-off`
- mode-related: emitted when LinuxCNC enters that particular mode
  - `mode-manual`
  - `mode-mdi`
  - `mode-auto`
- interpreter-related: emitted when the G-code interpreter changes into that mode
  - `interp-run`
  - `interp-idle`
  - `interp-paused`
  - `interp-reading`
  - `interp-waiting`

11.7 GladeVCP Programming

11.7.1 User Defined Actions

Most widget sets, and their associated user interface editors, support the concept of callbacks - functions in user-written code which are executed when *something happens* in the UI - events like mouse clicks, characters typed, mouse movement, timer events, window hiding and exposure and so forth.

HAL output widgets typically map input-type events like a button press to a value change of the associated HAL pin by means of such a - predefined - callback. Within PyVCP, this is really the only type of event handling supported - doing something more complex, like executing MDI commands to call a G-code subroutine, is not supported.

Within GladeVCP, HAL pin changes are just one type of the general class of events (called signals) in GTK+. Most widgets may originate such signals, and the Glade editor supports associating such a signal with a Python method or function name.

If you decide to use user-defined actions, your job is to write a Python module whose class methods - or in the simple case, just functions - can be referred to in Glade as event handlers. GladeVCP provides a way to import your module(s) at startup and will automatically link your event handlers with the widget signals as set in the Glade UI description.

11.7.2 An example: adding custom user callbacks in Python

This is just a minimal example to convey the idea - details are laid out in the rest of this section.

GladeVCP can not only manipulate or display HAL pins, you can also write regular event handlers in Python. This could be used, among others, to execute MDI commands. Here's how you do it:

Write a Python module like so and save as e.g. handlers.py:

```
nhits = 0
def on_button_press(gtkobj,data=None):
    global nhits nhits += 1 gtkobj.set_label("hits: %d" % nhits)
```

In Glade, define a button or HAL button, select the *Signals* tab, and in the GtkButton properties select the *pressed* line. Enter *on_button_press* there, and save the Glade file.

Then add the option *-u handlers.py* to the gladevcp command line. If your event handlers are spread over several files, just add multiple *-u <pyfilename>* options.

Now, pressing the button should change its label since it's set in the callback function.

What the `-u` flag does: all Python functions in this file are collected and setup as potential callback handlers for your Gtk widgets - they can be referenced from Glade *Signals* tabs. The callback handlers are called with the particular object instance as parameter, like the GtkButton instance above, so you can apply any GtkButton method from there.

Or do some more useful stuff, like calling an MDI command!

11.7.3 HAL value change events

HAL input widgets, like a LED, automatically associate their HAL pin state (on/off) with the optical appearance of the widget (LED lit/dark).

Beyond this builtin functionality, one may associate a change callback with any HAL pin, including those of predefined HAL widgets. This fits nicely with the event-driven structure of a typical widget application: every activity, be it mouse click, key, timer expired, or the change of a HAL pin's value, generates a callback and is handled by the same orthogonal mechanism.

For user-defined HAL pins not associated with a particular HAL widget, the signal name is *value-changed*. See the Adding HAL pins section below for details.

HAL widgets come with a pre-defined signal called *hal-pin-changed*. See the Hal Widgets section for details.

11.7.4 Programming model

The overall approach is as follows:

- design your UI with Glade, and set signal handlers where you want actions associated with a widget
- write a Python module which contains callable objects (see *handler models* below)
- pass your module's path name to gladevcp with the *-u <module>* option
- gladevcp imports the module, inspects it for signal handlers and connects them to the widget tree
- the main event loop is run.

11.7.4.1 The simple handler model

For simple tasks it's sufficient to define functions named after the Glade signal handlers. These will be called when the corresponding event happens in the widget tree. Here's a trivial example - it assumes that the *pressed* signal of a Gtk Button or HAL Button is linked to a callback called *on_button_press*:

```
nhits = 0
def on_button_press(gtkobj,data=None):
    global nhits
    nhits += 1
    gtkobj.set_label("hits: %d" % nhits)
```

Add this function to a Python file and run as follows:

```
gladevcp -u <myhandler>.py mygui.ui
```

Note communication between handlers has to go through global variables, which does not scale well and is positively unpythonic. This is why we came up with the class-based handler model.

11.7.4.2 The class-based handler model

The idea here is: handlers are linked to class methods. The underlying class(es) are instantiated and inspected during GladeVCP startup and linked to the widget tree as signal handlers. So the task now is to write:

- one or more several class definition(s) with one or several methods, in one module or split over several modules,
- a function *get_handlers* in each module which will return a list of class instances to GladeVCP - their method names will be linked to signal handlers

Here is a minimum user-defined handler example module:

```
class MyCallbacks :
    def on_this_signal(self,obj,data=None):
        print "this_signal happened, obj=",obj
    def get_handlers(halcomp,builder,useropts):
        return [MyCallbacks ()]
```

Now, *on_this_signal* will be available as signal handler to your widget tree.

11.7.4.3 The get_handlers protocol

If during module inspection GladeVCP finds a function `get_handlers`, it calls it as follows:

`get_handlers(halcomp,builder,useropts)`

the arguments are:

- halcomp - refers to the HAL component under construction
- builder - widget tree - result of reading the UI definition (either referring to a GtkBuilder or libglade-type object)
- useropts - a list of strings collected from the gladevcp command line `-U <useropts>` option

GladeVCP then inspects the list of class instances and retrieves their method names. Qualifying method names are connected to the widget tree as signal handlers. Only method names which do not begin with an _ (underscore) are considered.

Note that regardless whether you're using the libglade or the new GtkBuilder format for your Glade UI, widgets can always be referred to as `builder.get_object(<widgetname>)`. Also, the complete list of widgets is available as `builder.get_object` regardless of UI format.

11.7.5 Initialization sequence

It is important to know in which state of affairs your `get_handlers()` function is called so you know what is safe to do there and what not. First, modules are imported and initialized in command line order. After successful import, `get_handlers()` is called in the following state:

- the widget tree is created, but not yet realized (no toplevel `window.show()` has been executed yet)
- the halcomp HAL component is set up and all HAL widget's pins have already been added to it
- it is safe to add more HAL pins because `halcomp.ready()` has not yet been called at this point, so you may add your own pins, for instance in the class `__init__()` method.

Once all modules have been imported and method names extracted, the following steps happen:

- all qualifying method names will be connected to the widget tree with `connect_signals()/signal_autoconnect()` (depending on the type of UI imported - GtkBuilder vs the old libglade format).

- the HAL component is finalized with halcomp.ready()
- if a window ID was passed as argument, the widget tree is re-parented to run in this window, and Glade's toplevel window1 is abandoned (see FAQ)
- if a HAL command file was passed with `-H halfile`, it is executed with halcmd
- the Gtk main loop is run.

So when your handler class is initialized, all widgets are existent but not yet realized (displayed on screen). And the HAL component isn't ready as well, so its unsafe to access pins values in your `__init__()` method.

If you want to have a callback to execute at program start after it is safe to access HAL pins, then a connect a handler to the realize signal of the top level window1 (which might be its only real purpose). At this point GladeVCP is done with all setup tasks, the halfile has been run, and GladeVCP is about to enter the Gtk main loop.

11.7.6 Multiple callbacks with the same name

Within a class, method names must be unique. However, it is OK to have multiple class instances passed to GladeVCP by get_handlers() with identically named methods. When the corresponding signal occurs, these methods will be called in definition order - module by module, and within a module, in the order class instances are returned by `get_handlers()`.

11.7.7 The GladeVCP `-U <useropts>` flag

Instead of extending GladeVCP for any conceivable option which could potentially be useful for a handler class, you may use the -U <useroption> flag (repeatedly if you wish). This flag collects a list of <useroption> strings. This list is passed to the get_handlers() function (useropts argument). Your code is free to interpret these strings as you see fit. An possible usage would be to pass them to the Python exec function in your `get_handlers()` as follows:

```
debug = 0
...
def get_handlers(halcomp,builder,useropts):
    ...
    global debug # assuming there's a global var
    for cmd in useropts:
        exec cmd in globals()
```

This way you can pass arbitrary Python statements to your module through the gladevcp -U option, for example:

```
gladevcp -U debug=42 -U "print 'debug=%d' % debug" ...
```

This should set debug to 2 and confirm that your module actually did it.

11.7.8 Persistent variables in GladeVCP

A annoying aspect of GladeVCP in its earlier form and pyvcp is the fact that you may change values and HAL pins through text entry, sliders, spin boxes, toggle buttons etc, but their settings are not saved and restored at the next run of LinuxCNC - they start at the default value as set in the panel or widget definition.

GladeVCP has an easy-to-use mechanism to save and restore the state of HAL widgets, and program variables (in fact any instance attribute of type int, float, bool or string).

This mechanism uses the popular *.ini* file format to save and reload persistent attributes.

11.7.8.1 Persistence, program versions and the signature check

Imagine renaming, adding or deleting widgets in Glade: an .ini file lying around from a previous program version, or an entirely different user interface, would be not be able to restore the state properly since variables and types might have changed.

GladeVCP detects this situation by a signature which depends on all object names and types which are saved and to be restored. In the case of signature mismatch, a new .ini file with default settings is generated.

11.7.9 Using persistent variables

If you want any of Gtk widget state, HAL widgets output pin's values and/or class attributes of your handler class to be retained across invocations, proceed as follows:

- import the `gladevcp.persistence` module
- decide which instance attributes, and their default values you want to have retained, if any
- decide which widgets should have their state retained
- describe these decisions in your handler class' `init()` method through a nested dictionary as follows:

```
def __init__(self, halcomp,builder,useropts):
    self.halcomp = halcomp
    self.builder = builder
    self.useropts = useropts
    self.defaults = {
        # the following names will be saved/restored as method attributes
        # the save/restore mechanism is strongly typed - the variables type will be derived ↩
            from the type of the
        # initialization value. Currently supported types are: int, float, bool, string
        IniFile.vars : { 'nhits' : 0, 'a': 1.67, 'd': True ,'c' : "a string"},
        # to save/restore all widget's state which might remotely make sense, add this:
        IniFile.widgets : widget_defaults(builder.get_objects())
        # a sensible alternative might be to retain only all HAL output widgets' state:
        # IniFile.widgets: widget_defaults(select_widgets(self.builder.get_objects(),  ↩
           hal_only=True,output_only = True)),
    }
```

Then associate an .ini file with this descriptor:

```
self.ini_filename = __name__ + '.ini'
self.ini = IniFile(self.ini_filename,self.defaults,self.builder)
self.ini.restore_state(self)
```

After `restore_state()`, self will have attributes set if as running the following:

```
self.nhits = 0
self.a = 1.67
self.d = True
self.c = "a string"
```

Note that types are saved and preserved on restore. This example assumes that the ini file didn't exist or had the default values from self.defaults.

After this incantation, you can use the following IniFil methods:

ini.save_state(obj)
saves objs's attributes as per IniFil.vars dictionary and the widget state as described in IniFile.widgets in self.defaults

ini.create_default_ini()
create a .ini file with default values

ini.restore_state(obj)
restore HAL out pins and obj's attributes as saved/initialized to default as above

To save the widget and/or variable state on exit, connect a signal handler to the `window1` (toplevel) destroy event:

```
def on_destroy(self,obj,data=None):
    self.ini.save_state(self)
```

Next time you start the GladeVCP application, the widgets should come up in the state when the application was closed.

11.7.10 Hand-editing .ini files

You can do that, but note that the values in self.defaults override your edits if there is a syntax or type error in your edit. The error is detected, a console message will hint about that happened, and the bad inifile will be renamed to have the .BAD suffix. Subsequent bad ini files overwrite earlier .BAD files.

11.7.11 Adding HAL pins

If you need HAL pins which are not associated with a specific HAL widget, add them as follows:

```
import hal_glib
...
# in your handler class __init__():
self.example_trigger = hal_glib.GPin(halcomp.newpin('example-trigger', hal.HAL_BIT, hal. ↩
    HAL_IN))
```

To get a callback when this pin's value changes, associate a `value-change` callback with this pin, add:

```
self.example_trigger.connect('value-changed', self._on_example_trigger_change)
```

and define a callback method (or function, in this case leave out the `self` parameter):

```
# note '_' - this method will not be visible to the widget tree
def _on_example_trigger_change(self,pin,userdata=None):
    print "pin value changed to:" % (pin.get())
```

11.7.12 Adding timers

Since GladeVCP uses Gtk widgets which rely on the GObject base class, the full glib functionally is available. Here is an example for a timer callback:

```
def _on_timer_tick(self,userdata=None):
    ...
    return True # to restart the timer; return False for on-shot
...
# demonstrate a slow background timer - granularity is one second
# for a faster timer (granularity 1 ms), use this:
# glib.timeout_add(100, self._on_timer_tick,userdata) # 10Hz
glib.timeout_add_seconds(1, self._on_timer_tick)
```

11.7.13 Examples, and rolling your own GladeVCP application

Visit `emc2/configs/gladevcp` for running examples and starters for your own projects.

11.8 FAQ

1. *I get an unexpected unmap event in my handler function right after startup. What's this?*

   This is a consequence of your Glade UI file having the window1 Visible property set to True, together with re-parenting the GladeVCP window into Axis or touchy. The GladeVCP widget tree is created, including a top level window, and then *reparented into Axis*, leaving that toplevel window laying around orphaned. To avoid having this useless empty window hanging around, it is unmapped (made invisible), which is the cause of the unmap signal you get. Suggested fix: set window1.visible to False, and ignore an initial unmap event.

2. *My GladeVCP program starts, but no window appears where I expect it to be?*

   The window Axis allocates for GladeVCP will obtain the *natural size* of all its child widgets combined. It's the child widget's job to request a size (width and/or height). However, not all widgets do request a width greater than 0, for instance the Graph widget in its current form. If there's such a widget in your Glade file and it's the one which defines the layout you might want to set its width explicitly. Note that setting the window1 width and height properties in Glade does not make sense because this window will be orphaned during re-parenting and hence its geometry will have no impact on layout (see above). The general rule is: if you manually run a UI file with *gladevcp <uifile>* and its window has reasonable geometry, it should come up in Axis properly as well.

I want a blinking LED, so I ticked the checkbutton to let it blink with 100msec interval. It wont blink, and I get a startup warning: Warning: value "0" of type 'gint' is invalid or out of range for property 'led-blink-rate' of type 'gint'?:: This seems to be a glade bug. Just type over the blink rate field, and save again - this works for me.

My gladevcp panel in Axis doesnt save state when I close Axis, although I defined an on_destroy handler linked to the window destroy signal:: Very likely this handler is linked to window1, which due to reparenting isnt usable for this purpose. Please link the on_destroy handler to the destroy signal of an interior window. For instance, I have a notebook inside window1, and linked on_destroy to the notebooks destroy signal, and that works fine. It doesnt work for window1.

11.9 Troubleshooting

- make sure your have the development version of LinuxCNC installed. You don't need the axisrc file any more, this was mentioned in the old GladeVcp wiki page.
- run GladeVCP or Axis from a terminal window. If you get Python errors, check whether there's still a `/usr/lib/python2.6/dis` file lying around besides the newer `/usr/lib/python2.6/dist-packages/_hal.so` (note underscore); if yes, remove the `hal.so` file. It has been superseded by hal.py in the same directory and confuses the import mechanism.
- if you're using run-in-place, do a *make clean* to remove any accidentally left over hal.so file, then *make*.
- if you're using *HAL_table* or *HAL_HBox* widgets, be aware they have an HAL pin associated with it which is off by default. This pin controls whether these container's children are active or not.

11.10 Implementation note: Key handling in Axis

We believe key handling works OK, but since it is new code, we're telling about it you so you can watch out for problems; please let us know of errors or odd behavior. This is the story:

Axis uses the TkInter widget set. GladeVCP applications use Gtk widgets and run in a separate process context. They are hooked into Axis with the Xembed protocol. This allows a child application like GladeVCP to properly fit in a parent's window, and - in theory - have integrated event handling.

However, this assumes that both parent and child application properly support the Xembed protocol, which Gtk does, but TkInter doesn't. A consequence of this is that certain keys would not be forwarded from a GladeVCP panel to Axis properly under all circumstances. One of these situations was the case when an Entry, or SpinButton widget had focus: in this case, for instance an Escape key would not have been forwarded to Axis and cause an abort as it should, with potentially disastrous consequences.

Therefore, key events in GladeVCP are explicitly handled, and selectively forwarded to Axis, to assure that such situations cannot arise. For details, see the `keyboard_forward()` function in `lib/python/gladevcp/xembed.py`.

Chapter 12

HAL User Interface

12.1 Introduction

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

The easiest way to add halui is to add the following to the [HAL] section of the ini file.

```
HALUI = halui
```

An alternate way to invoke it is to include the following in your .hal file. Make sure you use the actual path to your ini file.

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

12.2 Halui pin reference

ABORT

- *halui.abort* (bit, in) - pin to send an abort message (clears out most errors)

AXIS

- *halui.axis.n.pos-commanded* (float, out) - Commanded axis position in machine coordinates
- *halui.axis.n.pos-feedback* (float, out) - Feedback axis position in machine coordinates
- *halui.axis.n.pos-relative* (float, out) - Commanded axis position in relative coordinates

E-STOP

- *halui.estop.activate* (bit, in) - pin for requesting E-Stop
- *halui.estop.is-activated* (bit, out) - indicates E-stop reset
- *halui.estop.reset* (bit, in) - pin for requesting E-Stop reset

FEED OVERRIDE

- *halui.feed-override.count-enable* (bit, in) - must be true for counts to work.

- *halui.feed-override.counts* (s32, in) - counts * scale = FO percentage
- *halui.feed-override.decrease* (bit, in) - pin for decreasing the FO (-=scale)
- *halui.feed-override.increase* (bit, in) - pin for increasing the FO (+=scale)
- *halui.feed-override.direct-value* (bit, in) - false when using encoder to change counts, true when setting counts directly. The *count-enable* pin must be true. [1]
- *halui.feed-override.scale* (float, in) - pin for setting the scale for increase and decrease
- *halui.feed-override.value* (float, out) - current FO value

MIST

- *halui.mist.is-on* (bit, out) - indicates mist is on
- *halui.mist.off* (bit, in) - pin for requesting mist off
- *halui.mist.on* (bit, in) - pin for requesting mist on

FLOOD

- *halui.flood.is-on* (bit, out) - indicates flood is on
- *halui.flood.off* (bit, in) - pin for requesting flood off
- *halui.flood.on* (bit, in) - pin for requesting flood on

HOMING

- *halui.home-all* (bit, in) - pin for requesting all axis to home. This pin will only be there if HOME_SEQUENCE is set in the ini file.

Jog <n> is a number between 0 and 8 and *selected*.

- *halui.jog-deadband* (float, in) - deadband for analog jogging (smaller jogging speed requests are not performed)
- *halui.jog-speed* (float, in) - pin for setting jog speed for minus/plus jogging
- *halui.jog.<n>.analog* (float, in) - analog velocity input for jogging (useful with joysticks or other analog devices)
- *halui.jog.<n>.increment* (float,in) - pin for setting the jog increment for axis <n> when using increment-minus or increment-plus to jog.
- *halui.jog.<n>.increment-minus* (bit, in) - pin for moving the <n> axis one increment in the minus direction for each off to on transition.
- *halui.jog.<n>.increment-plus* (bit, in) - pin for moving the <n> axis one increment in the plus direction for each off to on transition.
- *halui.jog.<n>.minus* (bit, in) - pin for jogging axis <n> in negative direction at the halui.jog.speed velocity
- *halui.jog.<n>.plus* (bit, in) - pin for jogging axis <n> in positive direction at the halui.jog.speed velocity
- *halui.jog.selected.increment* (float,in) - pin for setting the jog increment for the selected axis when using increment-minus or incremet-plus to jog.
- *halui.jog.selected.increment-minus* (bit, in) - pin for moving the selected axis one increment in the minus direction for each off to on transition.

[1] This function had been released, but is currently withdrawn for further development. It may or may not be present in your version of LinuxCNC.

- *halui.jog.selected.increment-plus* (bit, in) - pin for moving the selected axis one increment in the plus direction for each off to on transition.
- *halui.jog.selected.minus* (bit, in) - pin for jogging the selected axis in negative direction at the halui.jog.speed velocity
- *halui.jog.selected.plus* (bit, in) - pin for jogging the selected axis in positive direction at the halui.jog.speed velocity

Joint <n> is a number between 0 and 8 and *selected*.

- *halui.joint.<n>.has-fault* (bit, out) - status pin telling the joint has a fault
- *halui.joint.<n>.home* (bit, in) - pin for homing the specific joint
- *halui.joint.<n>.is-homed* (bit, out) - status pin telling that the joint is homed
- *halui.joint.<n>.is-selected bit* (bit, out) - status pin a joint is selected* internal halui
- *halui.joint.<n>.on-hard-max-limit* (bit, out) - status pin telling joint <n> is on the positive hardware limit switch
- *halui.joint.<n>.on-hard-min-limit* (bit, out) - status pin telling joint <n> is on the negative hardware limit switch
- *halui.joint.<n>.on-soft-max-limit* (bit, out) - status pin telling joint <n> is at the positive software limit
- *halui.joint.<n>.on-soft-min-limit* (bit, out) - status pin telling joint <n> is at the negative software limit
- *halui.joint.<n>.select* (bit, in) - select joint (0..8) - internal halui
- *halui.joint.<n>.unhome* (bit, in) - unhomes this joint
- *halui.joint.selected* (u32, out) - selected joint (0..8) - internal halui
- *halui.joint.selected.has-fault* (bit, out) - status pin telling that the joint <n> has a fault
- *halui.joint.selected.home* (bit, in) - pin for homing the selected joint
- *halui.joint.selected.is-homed* (bit, out) - status pin telling that the selected joint is homed
- *halui.joint.selected.on-hard-max-limit* (bit, out) - status pin telling that the selected joint is on the positive hardware limit
- *halui.joint.selected.on-hard-min-limit* (bit, out) - status pin telling that the selected joint is on the negative hardware limit
- *halui.joint.selected.on-soft-max-limit* (bit, out) - status pin telling that the selected joint is on the positive software limit
- *halui.joint.selected.on-soft-min-limit* (bit, out) - status pin telling that the selected joint is on the negative software limit
- *halui.joint.selected.unhome* (bit, in) - pin for unhoming the selected joint.

LUBE

- *halui.lube.is-on* (bit, out) - indicates lube is on
- *halui.lube.off* (bit, in) - pin for requesting lube off
- *halui.lube.on* (bit, in) - pin for requesting lube on

MACHINE

- *halui.machine.is-on* (bit, out) - indicates machine on
- *halui.machine.off* (bit, in) - pin for requesting machine off
- *halui.machine.on* (bit, in) - pin for requesting machine on

Max Velocity The maximum linear velocity can be adjusted from 0 to the MAX_VELOCITY that is set in the [TRAJ] section of the ini file.

- *halui.max-velocity.count-enable* (bit, in) - when TRUE, modify max velocity when counts changes
- *halui.max-velocity.counts* (s32, in) - lets you hook up an encoder to change the max velocity
- *halui.max-velocity.decrease* (bit, in) - pin for decreasing max velocity
- *halui.max-velocity.increase* (bit, in) - pin for increasing max velocity
- *halui.max-velocity.scale* (float, in) - the amount applied to the current maximum velocity with each transition from off to on of the increase or decrease pin in machine units per second.
- *halui.max-velocity.value* (float, out) - is the maximum linear velocity in machine units per second.

MDI

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

- The MDI_COMMAND is added to the ini file in the [HALUI] section.

```
[HALUI]
MDI_COMMAND = G0 X0
```

- When halui starts it will read the MDI_COMMAND fields in the ini, and export pins from 00 to the number of MDI_COMMAND's found in the ini up to a maximum of 64 commands.
- *halui.mdi-command-<nn>* (bit, in) - halui will try to send the MDI command defined in the ini. This will not always succeed, depending on the operating mode LinuxCNC is in (e.g. while in AUTO halui can't successfully send MDI commands). If the command succeeds then it will place LinuxCNC in the MDI mode and then back to Manual mode.

JOINT SELECTION

- *halui.joint.select* (u32, in) - select joint (0..8) - internal halui
- *halui.joint.selected* (u32, out) - joint (0..8) selected* internal halui
- *halui.joint.x.select bit* (bit, in) - pins for selecting a joint* internal halui
- *halui.joint.x.is-selected bit* (bit, out) - indicates joint selected* internal halui

MODE

- *halui.mode.auto* (bit, in) - pin for requesting auto mode
- *halui.mode.is-auto* (bit, out) - indicates auto mode is on
- *halui.mode.is-joint* (bit, out) - indicates joint by joint jog mode is on
- *halui.mode.is-manual* (bit, out) - indicates manual mode is on
- *halui.mode.is-mdi* (bit, out) - indicates mdi mode is on
- *halui.mode.is-teleop* (bit, out) - indicates coordinated jog mode is on
- *halui.mode.joint* (bit, in) - pin for requesting joint by joint jog mode
- *halui.mode.manual* (bit, in) - pin for requesting manual mode

- *halui.mode.mdi* (bit, in) - pin for requesting mdi mode
- *halui.mode.teleop* (bit, in) - pin for requesting coordinated jog mode

PROGRAM

- *halui.program.block-delete.is-on* (bit, out) - status pin telling that block delete is on
- *halui.program.block-delete.off* (bit, in) - pin for requesting that block delete is off
- *halui.program.block-delete.on* (bit, in) - pin for requesting that block delete is on
- *halui.program.is-idle* (bit, out) - status pin telling that no program is running
- *halui.program.is-paused* (bit, out) - status pin telling that a program is paused
- *halui.program.is-running* (bit, out) - status pin telling that a program is running
- *halui.program.optional-stop.is-on* (bit, out) - status pin telling that the optional stop is on
- *halui.program.optional-stop.off* (bit, in) - pin requesting that the optional stop is off
- *halui.program.optional-stop.on* (bit, in) - pin requesting that the optional stop is on
- *halui.program.pause* (bit, in) - pin for pausing a program
- *halui.program.resume* (bit, in) - pin for resuming a paused program
- *halui.program.run* (bit, in) - pin for running a program
- *halui.program.step* (bit, in) - pin for stepping in a program
- *halui.program.stop* (bit, in) - pin for stopping a program

SPINDLE OVERRIDE

- *halui.spindle-override.count-enable* (bit, in) - when TRUE, modify spindle override when counts changes.
- *halui.spindle-override.counts* (s32, in) - counts * scale = SO percentage
- *halui.spindle-override.decrease* (bit, in) - pin for decreasing the SO (-=scale)
- *halui.spindle-override.direct-value* (bit, in) - false when using encoder to change counts, true when setting counts directly. The *count-enable* pin must be true.
- *halui.spindle-override.increase* (bit, in) - pin for increasing the SO (+=scale)
- *halui.spindle-override.scale* (float, in) - pin for setting the scale on changing the SO
- *halui.spindle-override.value* (float, out) - current SO value

SPINDLE

- *halui.spindle.brake-is-on* (bit, out) - indicates brake is on
- *halui.spindle.brake-off* (bit, in) - pin for deactivating spindle/brake
- *halui.spindle.brake-on* (bit, in) - pin for activating spindle-brake
- *halui.spindle.decrease* (bit, in) - decreases spindle speed
- *halui.spindle.forward* (bit, in) - starts the spindle with CW motion
- *halui.spindle.increase* (bit, in)- increases spindle speed

- *halui.spindle.is-on* (bit, out) - indicates spindle is on (either direction)
- *halui.spindle.reverse* (bit, in)- starts the spindle with a CCW motion
- *halui.spindle.runs-backward* (bit, out) - indicates spindle is on, and in reverse
- *halui.spindle.runs-forward* (bit, out) - indicates spindle is on, and in forward
- *halui.spindle.start* (bit, in) - starts the spindle
- *halui.spindle.stop* (bit, in) - stops the spindle

TOOL

- *halui.tool.length-offset* (float, out) - indicates current applied tool-length-offset
- *halui.tool.number* (u32, out) - indicates current selected tool

Part IV

Hardware Drivers

Chapter 13

Parallel Port Driver

13.1 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 outputs, and 9 inputs. In other parallel ports, the control group has push-pull drivers and cannot be used as an input.

HAL and Open Collectors
HAL cannot automatically determine if the *x* mode bidirectional pins are actually open collectors (OC). If they are not, they cannot be used as inputs, and attempting to drive them LOW from an external source can damage the hardware.
To determine whether your port has *open collector* pins, load hal_parport in *x* mode. With no device attached, HAL should read the pin as TRUE. Next, insert a 470 ohm resistor from one of the control pins to GND. If the resulting voltage on the control pin is close to 0V, and HAL now reads the pin as FALSE, then you have an OC port. If the resulting voltage is far from 0V, or HAL does not read the pin as FALSE, then your port cannot be used in *x* mode.
The external hardware that drives the control pins should also use open collector gates (e.g., 74LS05).
On some machines, BIOS settings may affect whether *x* mode can be used. *SPP* mode is most likely to work.

No other combinations are supported, and a port cannot be changed from input to output once the driver is installed. The Parport Block Diagram shows two block diagrams, one showing the driver when the data group is configured for output, and one showing it configured for input. For *x* mode, refer to the pin listing of *halcmd show pin* for pin direction assignment.

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

13.1.1 Installing

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

Using the Port Index I/O addresses below 16 are treated as port indexes. This is the simplest way to install the parport driver and cooperates with the Linux parport_pc driver if it is loaded. This will use the address Linux has detected for parport 0.

```
loadrt hal_parport cfg="0"
```

Using the Port Address The configure 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:

```
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 usually 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.

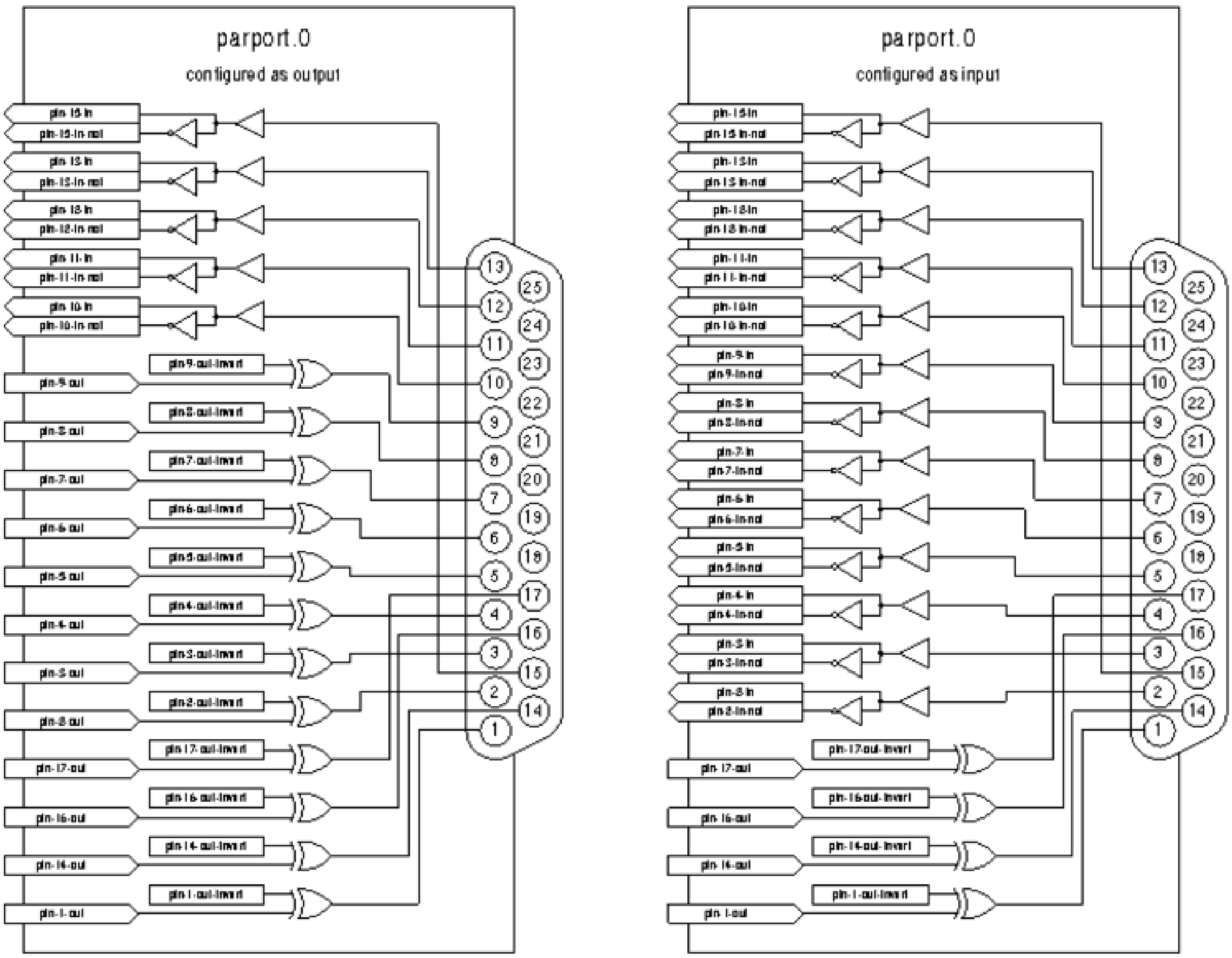


Figure 13.1: Parport Block Diagram

13.1.2 Pins

- *parport.<p>.pin-<n>-out* (bit) Drives a physical output pin.
- *parport.<p>.pin-<n>-in* (bit) Tracks a physical input pin.
- *parport.<p>.pin-<n>-in-not* (bit) Tracks a physical input pin, but inverted.

For each pin, *<p>* is the port number, and *<n>* is the physical pin number in the 25 pin D-shell connector.

For each physical output pin, the driver creates a single HAL pin, for example: *parport.0.pin-14-out*.

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.

13.1.3 Parameters

- *parport.<p>.pin-<n>-out-invert* (bit) Inverts an output pin.
- *parport.<p>.pin-<n>-out-reset* (bit) (only for *out* pins) TRUE if this pin should be reset when the *-reset* function is executed.
- parport.<p>.reset-time' (U32) The time (in nanoseconds) between a pin is set by *write* and reset by the *reset* function if it is enabled.

The *-invert* parameter determines whether an output pin is active high or active low. If *-invert* is FALSE, setting the HAL *-out* pin TRUE drives the physical pin high, and FALSE drives it low. If *-invert* is TRUE, then setting the HAL *-out* pin TRUE will drive the physical pin low.

13.1.4 Functions

- *parport.<p>.read* (funct) Reads physical input pins of port <*portnum*> and updates HAL *-in* and *-in-not* pins.
- *parport.read-all* (funct) Reads physical input pins of all ports and updates HAL *-in* and *-in-not* pins.
- *parport.<p>.write* (funct) Reads HAL *-out* pins of port <*p*> and updates that port's physical output pins.
- *parport.write-all* (funct) Reads HAL *-out* pins of all ports and updates all physical output pins.
- *parport.<p>.reset* (funct) Waits until *reset-time* has elapsed since the associated *write*, then resets pins to values indicated by *-out-invert* and *-out-invert* settings. *reset* must be later in the same thread as *write. 'If '-reset* is TRUE, then the *reset* function will set the pin to the value of *-out-invert*. This can be used in conjunction with stepgen's *doublefreq* to produce one step per period. The stepgen stepspace for that pin must be set to 0 to enable doublefreq.

The individual functions are provided for situations where one port needs to be updated in a very fast thread, but other ports can be updated in a slower thread to save CPU time. It is probably not a good idea to use both an *-all* function and an individual function at the same time.

13.1.5 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[1] and that no other device in the system has claimed the I/O ports.

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

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

13.1.6 Using DoubleStep

To setup DoubleStep on the parallel port you must add the function parport.n.reset after parport.n.write and configure stepspace to 0 and the reset time wanted. So that step can be asserted on every period in HAL and then toggled off by parport after being asserted for time specificed by parport.n.reset-time.

For example:

```
loadrt hal_parport cfg="0x378 out"
setp parport.0.reset-time 5000
loadrt stepgen step_type=0,0,0
addf parport.0.read base-thread
addf stepgen.make-pulses base-thread
addf parport.0.write base-thread
addf parport.0.reset base-thread
addf stepgen.capture-position servo-thread
...
setp stepgen.0.steplen 1
setp stepgen.0.stepspace 0
```

More information on DoubleStep can be found on the wiki.

13.2 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.

13.2.1 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.

Chapter 14

AX5214H Driver

The Axiom Measurement & Control AX5214H is a 48 channel digital I/O board. It plugs into an ISA bus, and resembles a pair of 8255 chips. In fact it may be a pair of 8255 chips, but I'm not sure. If/when someone starts a driver for an 8255 they should look at the ax5214 code, much of the work is already done.

14.1 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.

14.2 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.

14.3 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.

14.4 Functions

- *(funct) ax5214.<boardnum>.read* — Reads all digital inputs on one board.
- *(funct) ax5214.<boardnum>.write* — Writes all digital outputs on one board.

Chapter 15

GS2 VFD Driver

This is a userspace HAL program for the GS2 series of VFD's at Automation Direct.

This component is loaded using the halcmd "loadusr" command:

```
loadusr -Wn spindle-vfd gs2_vfd -n spindle-vfd
```

The above command says: loadusr, wait for named to load, component gs2_vfd, named spindle-vfd

15.1 Command Line Options

- *-b or --bits <n>* (default 8) Set number of data bits to <n>, where n must be from 5 to 8 inclusive
- *-d or --device <path>* (default /dev/ttyS0) Set the name of the serial device node to use
- *-g or --debug* Turn on debugging messages. This will also set the verbose flag. Debug mode will cause all modbus messages to be printed in hex on the terminal.
- *-n or --name <string>* (default gs2_vfd) Set the name of the HAL module. The HAL comp name will be set to <string>, and all pin and parameter names will begin with <string>.
- *-p or --parity {even,odd,none}* (default odd) Set serial parity to even, odd, or none.
- *-r or --rate <n>* (default 38400) Set baud rate to <n>. It is an error if the rate is not one of the following: 110, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200
- *-s or --stopbits {1,2}* (default 1) Set serial stop bits to 1 or 2
- *-t or --target <n>* (default 1) Set MODBUS target (slave) number. This must match the device number you set on the GS2.
- *-v or --verbose* Turn on debug messages.

Note
That if there are serial configuration errors, turning on verbose may result in a flood of timeout errors.

15.2 Pins

Where <n> is gs2_vfd or the name given during loading with the -n option.

- *<n>.DC-bus-volts* (float, out) The DC bus voltage of the VFD

- <n>*.at-speed* (bit, out) when drive is at commanded speed
- <n>*.err-reset* (bit, in) reset errors sent to VFD
- <n>*.firmware-revision* (s32, out) from the VFD
- <n>*.frequency-command* (float, out) from the VFD
- <n>*.frequency-out* (float, out) from the VFD
- <n>*.is-stopped* (bit, out) when the VFD reports 0 Hz output
- <n>*.load-percentage* (float, out) from the VFD
- <n>*.motor-RPM* (float, out) from the VFD
- <n>*.output-current* (float, out) from the VFD
- <n>*.output-voltage* (float, out) from the VFD
- <n>*.power-factor* (float, out) from the VFD
- <n>*.scale-frequency* (float, out) from the VFD
- <n>*.speed-command* (float, in) speed sent to VFD in RPM It is an error to send a speed faster than the Motor Max RPM as set in the VFD
- <n>*.spindle-fwd* (bit, in) 1 for FWD and 0 for REV sent to VFD
- <n>*.spindle-rev* (bit, in) 1 for REV and 0 if off
- <n>*.spindle-on* (bit, in) 1 for ON and 0 for OFF sent to VFD
- <n>*.status-1* (s32, out) Drive Status of the VFD (see the GS2 manual)
- <n>*.status-2* (s32, out) Drive Status of the VFD (see the GS2 manual)

Note
The status value is a sum of all the bits that are on. So a 163 which means the drive is in the run mode is the sum of 3 (run) + 32 (freq set by serial) + 128 (operation set by serial).

15.3 Parameters

Where <n> is gs2_vfd or the name given during loading with the -n option.

- <n>*.error-count* (s32, RW)
- <n>*.loop-time* (float, RW) how often the modbus is polled (default 0.1)
- <n>*.nameplate-HZ* (float, RW) Nameplate Hz of motor (default 60)
- <n>*.nameplate-RPM* (float, RW) Nameplate RPM of motor (default 1730)
- <n>*.retval* (s32, RW) the return value of an error in HAL
- <n>*.tolerance* (s32, RW) speed tolerance (default 0.01)

For an example of using this component to drive a spindle see the GS2 Spindle example.

Chapter 16

Mesa HostMot2 Driver

16.1 Introduction

HostMot2 is an FPGA configuration developed by Mesa Electronics for their line of *Anything I/O* motion control cards. The firmware is open source, portable and flexible. It can be configured (at compile-time) with zero or more instances (an object created at runtime) of each of several Modules: encoders (quadrature counters), PWM generators, and step/dir generators. The firmware can be configured (at run-time) to connect each of these instances to pins on the I/O headers. I/O pins not driven by a Module instance revert to general-purpose bi-directional digital I/O.

16.2 Firmware Binaries

Several pre-compiled HostMot2 firmware binaries are available for the different Anything I/O boards. (This list is incomplete, check the hostmot2-firmware distribution for up-to-date firmware lists.)

- 3x20 (144 I/O pins): using hm2_pci module
  - 24-channel servo
  - 16-channel servo plus 24 step/dir generators
- 5i22 (96 I/O pins): using hm2_pci module
  - 16-channel servo
  - 8-channel servo plus 24 step/dir generators
- 5i20, 5i23, 4i65, 4i68 (72 I/O pins): using hm2_pci module
  - 12-channel servo
  - 8-channel servo plus 4 step/dir generators
  - 4-channel servo plus 8 step/dir generators
- 7i43 (48 I/O pins): using hm2_7i43 module
  - 8-channel servo (8 PWM generators & 8 encoders)
  - 4-channel servo plus 4 step/dir generators

16.3 Installing Firmware

Depending on how you installed LinuxCNC you may have to open the Synaptic Package Manager from the System menu and install the package for your Mesa card. The quickest way to find them is to do a search for *hostmot2* in the Synaptic Package Manager. Mark the firmware for installation, then apply.

16.4 Loading HostMot2

The LinuxCNC support for the HostMot2 firmware is split into a generic driver called *hostmot2* and two low-level I/O drivers for the Anything I/O boards. The low-level I/O drivers are *hm2_7i43* and *hm2_pci* (for all the PCI- and PC-104/Plus-based Anything I/O boards). The hostmot2 driver must be loaded first, using a HAL command like this:

```
loadrt hostmot2
```

See the hostmot2(9) man page for details.

The hostmot2 driver by itself does nothing, it needs access to actual boards running the HostMot2 firmware. The low-level I/O drivers provide this access. The low-level I/O drivers are loaded with commands like this:

```
loadrt hm2_pci config="firmware=hm2/5i20/SVST8_4.BIT
       num_encoders=3 num_pwmgens=3 num_stepgens=1"
```

The config parameters are described in the hostmot2 man page.

16.5 Watchdog

The HostMot2 firmware may include a watchdog Module; if it does, the hostmot2 driver will use it.

The watchdog must be petted by LinuxCNC periodically or it will bite.

When the watchdog bites, all the board's I/O pins are disconnected from their Module instances and become high-impedance inputs (pulled high), and all communication with the board stops. The state of the HostMot2 firmware modules is not disturbed (except for the configuration of the I/O Pins). Encoder instances keep counting quadrature pulses, and pwm- and step-generators keep generating signals (which are not relayed to the motors, because the I/O Pins have become inputs).

Resetting the watchdog resumes communication and resets the I/O pins to the configuration chosen at load-time.

If the firmware includes a watchdog, the following HAL objects will be exported:

16.5.1 Pins:

- *has_bit* - (bit i/o) True if the watchdog has bit, False if the watchdog has not bit. If the watchdog has bit and the has_bit bit is True, the user can reset it to False to resume operation.

16.5.2 Parameters:

- *timeout_ns* - (u32 read/write) Watchdog timeout, in nanoseconds. This is initialized to 1,000,000,000 (1 second) at module load time. If more than this amount of time passes between calls to the pet_watchdog() function, the watchdog will bite.

16.5.3 Functions:

- *pet_watchdog()* - Calling this function resets the watchdog timer and postpones the watchdog biting until timeout_ns nanoseconds later. This function should be added to the servo thread.

16.6 HostMot2 Functions

- *hm2_<BoardType>.<BoardNum>.read* - Read all inputs, update input HAL pins.
- *hm2_<BoardType>.<BoardNum>.write* - Write all outputs.
- *hm2_<BoardType>.<BoardNum>.pet-watchdog* - Pet the watchdog to keep it from biting us for a while.
- *hm2_<BoardType>.<BoardNum>.read_gpio* - Read the GPIO input pins only. (This function is not available on the 7i43 due to limitations of the EPP bus.)
- *hm2_<BoardType>.<BoardNum>.write_gpio* - Write the GPIO control registers and output pins only. (This function is not available on the 7i43 due to limitations of the EPP bus.)

Note

The above *read_gpio* and *write_gpio* functions should not normally be needed, since the GPIO bits are read and written along with everything else in the standard *read* and *write* functions above, which are normally run in the servo thread.
The *read_gpio* and *write_gpio* functions were provided in case some very fast (frequently updated) I/O is needed. These functions should be run in the base thread. If you have need for this, please send an email and tell us about it, and what your application is.

16.7 Pinouts

The hostmot2 driver does not have a particular pinout. The pinout comes from the firmware that the hostmot2 driver sends to the Anything I/O board. Each firmware has different pinout, and the pinout depends on how many of the available encoders, pwmgens, and stepgens are used. To get a pinout list for your configuration after loading LinuxCNC in the terminal window type:

```
dmesg > hm2.txt
```

The resulting text file will contain lots of information as well as the pinout for the HostMot2 and any error and warning messages.

To reduce the clutter by clearing the message buffer before loading LinuxCNC type the following in the terminal window:

```
sudo dmesg -c
```

Now when you run LinuxCNC and then do a *dmesg > hm2.txt* in the terminal only the info from the time you loaded LinuxCNC will be in your file along with your pinout. The file will be in the current directory of the terminal window. Each line will contain the card name, the card number, the I/O Pin number, the connector and pin, and the usage. From this printout you will know the physical connections to your card based on your configuration.

An example of a 5i20 configuration:

```
[HOSTMOT2]
DRIVER=hm2_pci
BOARD=5i20
CONFIG="firmware=hm2/5i20/SVST8_4.BIT num_encoders=1 num_pwmgens=1 num_stepgens=3"
```

The above configuration produced this printout.

```
[ 1141.053386] hm2/hm2_5i20.0: 72 I/O Pins used:
[ 1141.053394] hm2/hm2_5i20.0: IO Pin 000 (P2-01): IOPort
[ 1141.053397] hm2/hm2_5i20.0: IO Pin 001 (P2-03): IOPort
[ 1141.053401] hm2/hm2_5i20.0: IO Pin 002 (P2-05): Encoder #0, pin B (Input)
[ 1141.053405] hm2/hm2_5i20.0: IO Pin 003 (P2-07): Encoder #0, pin A (Input)
[ 1141.053408] hm2/hm2_5i20.0: IO Pin 004 (P2-09): IOPort
[ 1141.053411] hm2/hm2_5i20.0: IO Pin 005 (P2-11): Encoder #0, pin Index (Input)
[ 1141.053415] hm2/hm2_5i20.0: IO Pin 006 (P2-13): IOPort
```

```
[ 1141.053418] hm2/hm2_5i20.0: IO Pin 007 (P2-15): PWMGen #0, pin Out0 (PWM or Up) (Output)
[ 1141.053422] hm2/hm2_5i20.0: IO Pin 008 (P2-17): IOPort
[ 1141.053425] hm2/hm2_5i20.0: IO Pin 009 (P2-19): PWMGen #0, pin Out1 (Dir or Down) ( ←
    Output)
[ 1141.053429] hm2/hm2_5i20.0: IO Pin 010 (P2-21): IOPort
[ 1141.053432] hm2/hm2_5i20.0: IO Pin 011 (P2-23): PWMGen #0, pin Not-Enable (Output)
<snip>...
[ 1141.053589] hm2/hm2_5i20.0: IO Pin 060 (P4-25): StepGen #2, pin Step (Output)
[ 1141.053593] hm2/hm2_5i20.0: IO Pin 061 (P4-27): StepGen #2, pin Direction (Output)
[ 1141.053597] hm2/hm2_5i20.0: IO Pin 062 (P4-29): StepGen #2, pin (unused) (Output)
[ 1141.053601] hm2/hm2_5i20.0: IO Pin 063 (P4-31): StepGen #2, pin (unused) (Output)
[ 1141.053605] hm2/hm2_5i20.0: IO Pin 064 (P4-33): StepGen #2, pin (unused) (Output)
[ 1141.053609] hm2/hm2_5i20.0: IO Pin 065 (P4-35): StepGen #2, pin (unused) (Output)
[ 1141.053613] hm2/hm2_5i20.0: IO Pin 066 (P4-37): IOPort
[ 1141.053616] hm2/hm2_5i20.0: IO Pin 067 (P4-39): IOPort
[ 1141.053619] hm2/hm2_5i20.0: IO Pin 068 (P4-41): IOPort
[ 1141.053621] hm2/hm2_5i20.0: IO Pin 069 (P4-43): IOPort
[ 1141.053624] hm2/hm2_5i20.0: IO Pin 070 (P4-45): IOPort
[ 1141.053627] hm2/hm2_5i20.0: IO Pin 071 (P4-47): IOPort
[ 1141.053811] hm2/hm2_5i20.0: registered
[ 1141.053815] hm2_5i20.0: initialized AnyIO board at 0000:02:02.0
```

Note

That the I/O Pin nnn will correspond to the pin number shown on the HAL Configuration screen for GPIOs. Some of the Stepgen, Encoder and PWMGen will also show up as GPIOs in the HAL Configuration screen.

16.8 PIN Files

The default pinout is described in a .PIN file (human-readable text). When you install a firmware package the .PIN files are installed in

```
/usr/share/doc/hostmot2-firmware-<board>/
```

16.9 Firmware

The selected firmware (.BIT file) and configuration is uploaded from the PC motherboard to the Mesa mothercard on LinuxCNC startup. If you are using Run In Place, you must still install a hostmot2-firmware-<board> package. There is more information about firmware and configuration in the *Configurations* section.

16.10 HAL Pins

The HAL pins for each configuration can be seen by opening up *Show HAL Configuration* from the Machine menu. All the HAL pins and parameters can be found there. The following figure is of the 5i20 configuration used above.

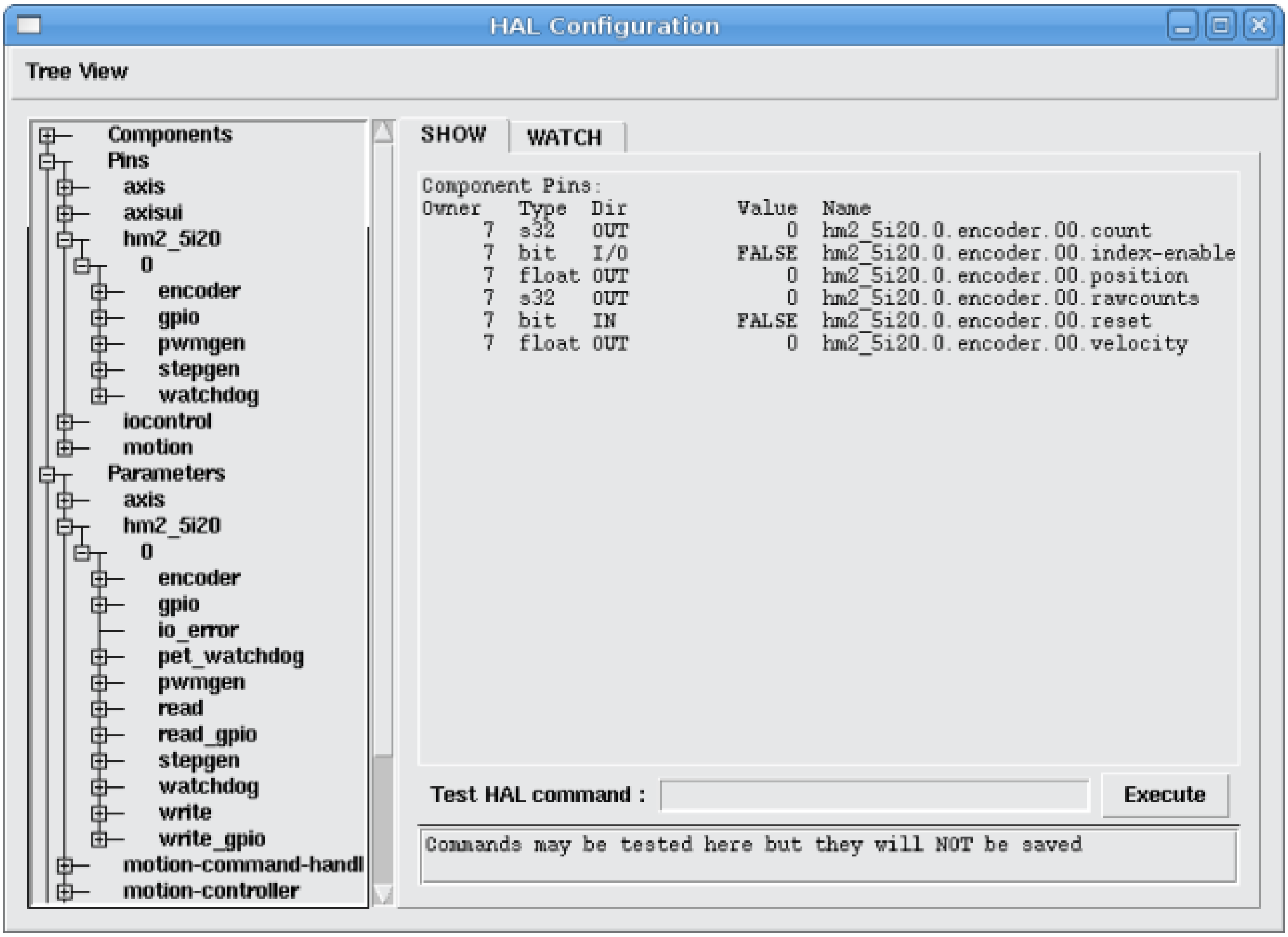


Figure 16.1: 5i20 HAL Pins

16.11 Configurations

The Hostmot2 firmware is available in several versions, depending on what you are trying to accomplish. You can get a reminder of what a particular firmware is for by looking at the name. Let's look at a couple of examples.

In the 7i43 (two ports), SV8 (*Servo 8*) would be for having 8 servos or fewer, using the *classic* 7i33 4-axis (per port) servo board. So 8 servos would use up all 48 signals in the two ports. But if you only needed 3 servos, you could say *num_encoders=3* and *num_pwmgens=3* and recover 5 servos at 6 signals each, thus gaining 30 bits of GPIO.

Or, in the 5i22 (four ports), SVST8_24 (*Servo 8, Stepper 24*) would be for having 8 servos or fewer (7i33 x2 again), and 24 steppers or fewer (7i47 x2). This would use up all four ports. If you only needed 4 servos you could say *num_encoders=4* and *num_pwmgens=4* and recover 1 port (and save a 7i33). And if you only needed 12 steppers you could say *num_stepgens=12* and free up one port (and save a 7i47). So in this way we can save two ports (48 bits) for GPIO.

Here are tables of the firmwares available in the official packages. There may be additional firmwares available at the Mesanet.com website that have not yet made it into the LinuxCNC official firmware packages, so check there too.

3x20 (6-port various) Default Configurations (The 3x20 comes in 1M, 1.5M, and 2M gate versions. So far, all firmware is available in all gate sizes.)

Firmware	Encoder	PWMGen	StepGen	GPIO
SV24	24	24	0	0
SVST16_24	16	16	24	0

5i22 (4-port PCI) Default Configurations (The 5i22 comes in 1M and 1.5M gate versions. So far, all firmware is available in all gate sizes.)

Firmware	Encoder	PWM	StepGen	GPIO
SV16	16	16	0	0
SVST2_4_7I47	4	2	4	72
SVST8_8	8	8	8	0
SVST8_24	8	8	24	0

5i23 (3-port PCI) Default Configurations (The 5i23 has 400k gates.)

Firmware	Encoder	PWM	StepGen	GPIO
SV12	12	12	0	0
SVST2_8	2	2	8 (tbl5)	12
SVST2_4_7I47	4	2	4	48
SV12_2X7I48_72	12	12	0	24
SV12IM_2X7I48_72	12 (+IM)	12	0	12
SVST4_8	4	4	8 (tbl5)	0
SVST8_4	8	8	4 (tbl5)	0
SVST8_4IM2	8 (+IM)	8	4	8
SVST8_8IM2	8 (+IM)	8	8	0
SVTP6_7I39	6	0 (6 BLDC)	0	0

5i20 (3-port PCI) Default Configurations (The 5i20 has 200k gates.)

Firmware	Encoder	PWM	StepGen	GPIO
SV12	12	12	0	0
SVST2_8	2	2	8 (tbl5)	12
SVST2_4_7I47	4	2	4	48
SV12_2X7I48_72	12	12	0	24
SV12IM_2X7I48_72	12 (+IM)	12	0	12
SVST8_4	8	8	4 (tbl5)	0
SVST8_4IM2	8 (+IM)	8	4	8

4i68 (3-port PC/104) Default Configurations (The 4i68 has 400k gates.)

Firmware	Encoder	PWM	StepGen	GPIO
SV12	12	12	0	0
SVST2_4_7I47	4	2	4	48
SVST4_8	4	4	8	0
SVST8_4	8	8	4	0
SVST8_4IM2	8 (+IM)	8	4	8
SVST8_8IM2	8 (+IM)	8	8	0

4i65 (3-port PC/104) Default Configurations (The 4i65 has 200k gates.)

Firmware	Encoder	PWM	StepGen	GPIO
SV12	12	12	0	0
SVST8_4	8	8	4	0
SVST8_4IM2	8 (+IM)	8	4	8

7i43 (2-port parallel) 400k gate versions, Default Configurations

Firmware	Encoder	PWM	StepGen	GPIO
SV8	8	8	0	0
SVST4_4	4	4	4 (tbl5)	0
SVST4_6	4	4	6 (tbl3)	0
SVST4_12	4	4	12	0
SVST2_4_7I47	4	2	4	24

7i43 (2-port parallel) 200k gate versions, Default Configurations

Firmware	Encoder	PWM	StepGen	GPIO
SV8	8	8	0	0
SVST4_4	4	4	4 (tbl5)	0
SVST4_6	4	4	6 (tbl3)	0
SVST2_4_7I47	4	2	4	24

Even though several cards may have the same named .BIT file you cannot use a .BIT file that is not for that card. Different cards have different clock frequencies so make sure you load the proper .BIT file for your card. Custom hm2 firmwares can be created for special applications and you may see some custom hm2 firmwares in the directories with the default ones.

When you load the board-driver (hm2_pci or hm2_7i43), you can tell it to disable instances of the three primary modules (pwmgen, stepgen, and encoder) by setting the count lower. Any I/O pins belonging to disabled module instances become GPIOs.

16.12 GPIO

General Purpose I/O pins on the board which are not used by a module instance are exported to HAL as *full* GPIO pins. Full GPIO pins can be configured at run-time to be inputs, outputs, or open drains, and have a HAL interface that exposes this flexibility. I/O pins that are owned by an active module instance are constrained by the requirements of the owning module, and have a restricted HAL interface.

GPIOs have names like *hm2_<BoardType>.<BoardNum>.gpio.<IONum>*. IONum. is a three-digit number. The mapping from IONum to connector and pin-on-that-connector is written to the syslog when the driver loads, and it's documented in Mesa's manual for the Anything I/O boards.

The hm2 GPIO representation is modeled after the Digital Inputs and Digital Outputs described in the Canonical Device Interface (part of the HAL General Reference document).

GPIO pins default to input.

16.12.1 Pins

- *in* - (Bit, Out) Normal state of the hardware input pin. Both full GPIO pins and I/O pins used as inputs by active module instances have this pin.
- *in_not* - (Bit, Out) Inverted state of the hardware input pin. Both full GPIO pins and I/O pins used as inputs by active module instances have this pin.
- *out* - (Bit, In) Value to be written (possibly inverted) to the hardware output pin. Only full GPIO pins have this pin.

16.12.2 Parameters

- *invert_output* - (Bit, RW) This parameter only has an effect if the *is_output* parameter is true. If this parameter is true, the output value of the GPIO will be the inverse of the value on the *out* HAL pin. Only full GPIO pins and I/O pins used as outputs by active module instances have this parameter. To invert an active module pin you have to invert the GPIO pin not the module pin.

- *is_opendrain* - (Bit, RW) This parameter only has an effect if the *is_output* parameter is true. If this parameter is false, the GPIO behaves as a normal output pin: the I/O pin on the connector is driven to the value specified by the *out* HAL pin (possibly inverted), and the value of the *in* and *in_not* HAL pins is undefined. If this parameter is true, the GPIO behaves as an open-drain pin. Writing 0 to the *out* HAL pin drives the I/O pin low, writing 1 to the *out* HAL pin puts the I/O pin in a high-impedance state. In this high-impedance state the I/O pin floats (weakly pulled high), and other devices can drive the value; the resulting value on the I/O pin is available on the *in* and *in_not* pins. Only full GPIO pins and I/O pins used as outputs by active module instances have this parameter.
- *is_output* - (Bit, RW) If set to 0, the GPIO is an input. The I/O pin is put in a high-impedance state (weakly pulled high), to be driven by other devices. The logic value on the I/O pin is available in the *in* and *in_not* HAL pins. Writes to the *out* HAL pin have no effect. If this parameter is set to 1, the GPIO is an output; its behavior then depends on the *is_opendrain* parameter. Only full GPIO pins have this parameter.

16.13 StepGen

Stepgens have names like *hm2_<BoardType>.<BoardNum>.stepgen.<Instance>.*. *Instance* is a two-digit number that corresponds to the HostMot2 stepgen instance number. There are *num_stepgens* instances, starting with 00.

Each stepgen allocates 2-6 I/O pins (selected at firmware compile time), but currently only uses two: Step and Direction outputs.[1]

The stepgen representation is modeled on the stepgen software component. Stepgen default is active high step output (high during step time low during step space). To invert a StepGen output pin you invert the corresponding GPIO pin that is being used by StepGen. To find the GPIO pin being used for the StepGen output run dmesg as shown above.

Each stepgen instance has the following pins and parameters:

16.13.1 Pins

- *control-type* - (Bit, In) Switches between position control mode (0) and velocity control mode (1). Defaults to position control (0).
- *counts* - (s32, Out) Feedback position in counts (number of steps).
- *enable* - (Bit, In) Enables output steps. When false, no steps are generated.
- *position-cmd* - (Float, In) Target position of stepper motion, in user-defined position units.
- *position-fb* - (Float, Out) Feedback position in user-defined position units (counts / position_scale).
- *velocity-cmd* - (Float, In) Target velocity of stepper motion, in user-defined position units per second. This pin is only used when the stepgen is in velocity control mode (control-type=1).
- *velocity-fb* - (Float, Out) Feedback velocity in user-defined position units per second.

16.13.2 Parameters

- *dirhold* - (u32, RW) Minimum duration of stable Direction signal after a step ends, in nanoseconds.
- *dirsetup* - (u32, RW) Minimum duration of stable Direction signal before a step begins, in nanoseconds.
- *maxaccel* - (Float, RW) Maximum acceleration, in position units per second per second. If set to 0, the driver will not limit its acceleration.
- *maxvel* - (Float, RW) Maximum speed, in position units per second. If set to 0, the driver will choose the maximum velocity based on the values of steplen and stepspace (at the time that maxvel was set to 0).
- *position-scale* - (Float, RW) Converts from counts to position units. position = counts / position_scale

[1] At present, the firmware supports multi-phase stepper outputs, but the driver doesn't. Interested volunteers are solicited.

- *step_type* - (u32, RW) Output format, like the step_type modparam to the software stegen(9) component. 0 = Step/Dir, 1 = Up/Down, 2 = Quadrature. In Quadrature mode (step_type=2), the stepgen outputs one complete Gray cycle (00 -> 01 -> 11 -> 10 -> 00) for each *step* it takes.
- *steplen* - (u32, RW) Duration of the step signal, in nanoseconds.
- *stepspace* - (u32, RW) Minimum interval between step signals, in nanoseconds.

16.13.3 Output Parameters

The Step and Direction pins of each StepGen have two additional parameters. To find which I/O pin belongs to which step and direction output run dmesg as described above.

- *invert_output* - (Bit, RW) This parameter only has an effect if the *is_output* parameter is true. If this parameter is true, the output value of the GPIO will be the inverse of the value on the *out* HAL pin.
- *is_opendrain* - (Bit, RW) If this parameter is false, the GPIO behaves as a normal output pin: the I/O pin on the connector is driven to the value specified by the *out* HAL pin (possibly inverted). If this parameter is true, the GPIO behaves as an open-drain pin. Writing 0 to the *out* HAL pin drives the I/O pin low, writing 1 to the *out* HAL pin puts the I/O pin in a high-impedance state. In this high-impedance state the I/O pin floats (weakly pulled high), and other devices can drive the value; the resulting value on the I/O pin is available on the *in* and *in_not* pins. Only full GPIO pins and I/O pins used as outputs by active module instances have this parameter.

16.14 PWMGen

PWMgens have names like *hm2_<BoardType>.<BoardNum>.pwmgen.<Instance>.*. *Instance* is a two-digit number that corresponds to the HostMot2 pwmgen instance number. There are *num_pwmgens* instances, starting with 00.

In HM2, each pwmgen uses three output I/O pins: Not-Enable, Out0, and Out1. To invert a PWMGen output pin you invert the corresponding GPIO pin that is being used by PWMGen. To find the GPIO pin being used for the PWMGen output run dmesg as shown above.

The function of the Out0 and Out1 I/O pins varies with output-type parameter (see below).

The hm2 pwmgen representation is similar to the software pwmgen component. Each pwmgen instance has the following pins and parameters:

16.14.1 Pins

- *enable* - (Bit, In) If true, the pwmgen will set its Not-Enable pin false and output its pulses. If *enable* is false, pwmgen will set its Not-Enable pin true and not output any signals.
- *value* - (Float, In) The current pwmgen command value, in arbitrary units.

16.14.2 Parameters

- *output-type* - (s32, RW) This emulates the output_type load-time argument to the software pwmgen component. This parameter may be changed at runtime, but most of the time you probably want to set it at startup and then leave it alone. Accepted values are 1 (PWM on Out0 and Direction on Out1), 2 (Up on Out0 and Down on Out1), 3 (PDM mode, PDM on Out0 and Dir on Out1), and 4 (Direction on Out0 and PWM on Out1, *for locked antiphase*).
- *scale* - (Float, RW) Scaling factor to convert *value* from arbitrary units to duty cycle: dc = value / scale. Duty cycle has an effective range of -1.0 to +1.0 inclusive, anything outside that range gets clipped.

- *pdm_frequency* - (u32, RW) This specifies the PDM frequency, in Hz, of all the pwmgen instances running in PDM mode (mode 3). This is the *pulse slot frequency*; the frequency at which the pdm generator in the Anything I/O board chooses whether to emit a pulse or a space. Each pulse (and space) in the PDM pulse train has a duration of 1/pdm_frequency seconds. For example, setting the pdm_frequency to 2e6 (2 MHz) and the duty cycle to 50% results in a 1 MHz square wave, identical to a 1 MHz PWM signal with 50% duty cycle. The effective range of this parameter is from about 1525 Hz up to just under 100 MHz. Note that the max frequency is determined by the ClockHigh frequency of the Anything I/O board; the 5i20 and 7i43 both have a 100 MHz clock, resulting in a 100 Mhz max PDM frequency. Other boards may have different clocks, resulting in different max PDM frequencies. If the user attempts to set the frequency too high, it will be clipped to the max supported frequency of the board.
- *pwm_frequency* - (u32, RW) This specifies the PWM frequency, in Hz, of all the pwmgen instances running in the PWM modes (modes 1 and 2). This is the frequency of the variable-duty-cycle wave. Its effective range is from 1 Hz up to 193 KHz. Note that the max frequency is determined by the ClockHigh frequency of the Anything I/O board; the 5i20 and 7i43 both have a 100 MHz clock, resulting in a 193 KHz max PWM frequency. Other boards may have different clocks, resulting in different max PWM frequencies. If the user attempts to set the frequency too high, it will be clipped to the max supported frequency of the board. Frequencies below about 5 Hz are not terribly accurate, but above 5 Hz they're pretty close.

16.14.3 Output Parameters

The output pins of each PWMGen have two additional parameters. To find which I/O pin belongs to which output run dmesg as described above.

- *invert_output* - (Bit, RW) This parameter only has an effect if the *is_output* parameter is true. If this parameter is true, the output value of the GPIO will be the inverse of the value on the *out* HAL pin.
- *is_opendrain* - (Bit, RW) If this parameter is false, the GPIO behaves as a normal output pin: the I/O pin on the connector is driven to the value specified by the *out* HAL pin (possibly inverted). If this parameter is true, the GPIO behaves as an open-drain pin. Writing 0 to the *out* HAL pin drives the I/O pin low, writing 1 to the *out* HAL pin puts the I/O pin in a high-impedance state. In this high-impedance state the I/O pin floats (weakly pulled high), and other devices can drive the value; the resulting value on the I/O pin is available on the *in* and *in_not* pins. Only full GPIO pins and I/O pins used as outputs by active module instances have this parameter.

16.15 Encoder

Encoders have names like *hm2_<BoardType>.<BoardNum>.encoder.<Instance>.*. *Instance* is a two-digit number that corresponds to the HostMot2 encoder instance number. There are *num_encoders* instances, starting with 00.

Each encoder uses three or four input I/O pins, depending on how the firmware was compiled. Three-pin encoders use A, B, and Index (sometimes also known as Z). Four-pin encoders use A, B, Index, and Index-mask.

The hm2 encoder representation is similar to the one described by the Canonical Device Interface (in the HAL General Reference document), and to the software encoder component. Each encoder instance has the following pins and parameters:

16.15.1 Pins

- *count* - (s32, Out) Number of encoder counts since the previous reset.
- *index-enable* - (Bit, I/O) When this pin is set to True, the count (and therefore also position) are reset to zero on the next Index (Phase-Z) pulse. At the same time, index-enable is reset to zero to indicate that the pulse has occurred.
- *position* - (Float, Out) Encoder position in position units (count / scale).
- *rawcounts* - (s32, Out) Total number of encoder counts since the start, not adjusted for index or reset.
- *reset* - (Bit, In) When this pin is TRUE, the count and position pins are set to 0. (The value of the velocity pin is not affected by this.) The driver does not reset this pin to FALSE after resetting the count to 0, that is the user's job.
- *velocity* - (Float, Out) Estimated encoder velocity in position units per second.

16.15.2 Parameters

- *counter-mode* - (Bit, RW) Set to False (the default) for Quadrature. Set to True for Up/Down or for single input on Phase A. Can be used for a frequency to velocity converter with a single input on Phase A when set to true.
- *filter* - (Bit, RW) If set to True (the default), the quadrature counter needs 15 clocks to register a change on any of the three input lines (any pulse shorter than this is rejected as noise). If set to False, the quadrature counter needs only 3 clocks to register a change. The encoder sample clock runs at 33 MHz on the PCI Anything I/O cards and 50 MHz on the 7i43.
- *index-invert* - (Bit, RW) If set to True, the rising edge of the Index input pin triggers the Index event (if index-enable is True). If set to False, the falling edge triggers.
- *index-mask* - (Bit, RW) If set to True, the Index input pin only has an effect if the Index-Mask input pin is True (or False, depending on the index-mask-invert pin below).
- *index-mask-invert* - (Bit, RW) If set to True, Index-Mask must be False for Index to have an effect. If set to False, the Index-Mask pin must be True.
- *scale* - (Float, RW) Converts from *count* units to *position* units. A quadrature encoder will normally have 4 counts per pulse so a 100 PPR encoder would be 400 counts per revolution. In *.counter-mode* a 100 PPR encoder would have 100 counts per revelution as it only uses the rising edge of A and direction is B.
- *vel-timeout* - (Float, RW) When the encoder is moving slower than one pulse for each time that the driver reads the count from the FPGA (in the hm2_read() function), the velocity is harder to estimate. The driver can wait several iterations for the next pulse to arrive, all the while reporting the upper bound of the encoder velocity, which can be accurately guessed. This parameter specifies how long to wait for the next pulse, before reporting the encoder stopped. This parameter is in seconds.

16.16 Examples

Several example configurations are included with LinuxCNC for both stepper and servo applications. The configurations are located in the hm2-servo and hm2-stepper sections of the LinuxCNC Configuration Selector window. You will need the same board installed for the configuration you pick to load. The examples are a good place to start and will save you time. Just pick the proper example from the LinuxCNC Configuration Selector and save a copy to your computer so you can edit it. To see the exact pins and parameters that your configuration gave you, open the Show HAL Configuration window from the Machine menu, or do dmesg as outlined above.

Chapter 17

Motenc Driver

Vital Systems Motenc-100 and Motenc-LITE

The Vital Systems Motenc-100 and Motenc-LITE are 8- and 4-channel servo control boards. The Motenc-100 provides 8 quadrature encoder counters, 8 analog inputs, 8 analog outputs, 64 (68?) digital inputs, and 32 digital outputs. The Motenc-LITE has only 4 encoder counters, 32 digital inputs and 16 digital outputs, but it still has 8 analog inputs and 8 analog outputs. The driver automatically identifies the installed board and exports the appropriate HAL objects.

Installing:

```
loadrt hal_motenc
```

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

Up to 4 boards may be used in one system.

17.1 Pins

In the following pins, parameters, and functions, <board> is the board ID. According to the naming conventions the first board should always have an ID of zero. However this driver sets the ID based on a pair of jumpers on the board, so it may be non-zero even if there is only one board.

- *(s32) motenc.<board>.enc-<channel>-count* - Encoder position, in counts.
- *(float) motenc.<board>.enc-<channel>-position* - Encoder position, in user units.
- *(bit) motenc.<board>.enc-<channel>-index* - Current status of index pulse input.
- *(bit) motenc.<board>.enc-<channel>-idx-latch* - Driver sets this pin true when it latches an index pulse (enabled by latch-index). Cleared by clearing latch-index.
- *(bit) motenc.<board>.enc-<channel>-latch-index* - If this pin is true, the driver will reset the counter on the next index pulse.
- *(bit) motenc.<board>.enc-<channel>-reset-count* - If this pin is true, the counter will immediately be reset to zero, and the pin will be cleared.
- *(float) motenc.<board>.dac-<channel>-value* - Analog output value for DAC (in user units, see -gain and -offset)
- *(float) motenc.<board>.adc-<channel>-value* - Analog input value read by ADC (in user units, see -gain and -offset)
- *(bit) motenc.<board>.in-<channel>* - State of digital input pin, see canonical digital input.
- *(bit) motenc.<board>.in-<channel>-not* - Inverted state of digital input pin, see canonical digital input.

- *(bit) motenc.<board>.out-<channel>* - Value to be written to digital output, seen canonical digital output.
- *(bit) motenc.<board>.estop-in* - Dedicated estop input, more details needed.
- *(bit) motenc.<board>.estop-in-not* - Inverted state of dedicated estop input.
- *(bit) motenc.<board>.watchdog-reset* - Bidirectional, - Set TRUE to reset watchdog once, is automatically cleared.

17.2 Parameters

- *(float) motenc.<board>.enc-<channel>-scale* - The number of counts / user unit (to convert from counts to units).
- *(float) motenc.<board>.dac-<channel>-offset* - Sets the DAC offset.
- *(float) motenc.<board>.dac-<channel>-gain* - Sets the DAC gain (scaling).
- *(float) motenc.<board>.adc-<channel>-offset* - Sets the ADC offset.
- *(float) motenc.<board>.adc-<channel>-gain* - Sets the ADC gain (scaling).
- *(bit) motenc.<board>.out-<channel>-invert* - Inverts a digital output, see canonical digital output.
- *(u32) motenc.<board>.watchdog-control* - Configures the watchdog. The value may be a bitwise OR of the following values:

Bit #	Value	Meaning
0	1	Timeout is 16ms if set, 8ms if unset
1	2	
2	4	Watchdog is enabled
3	8	
4	16	Watchdog is automatically reset by DAC writes (the HAL dac-write function)

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

- *(u32) motenc.<board>.led-view* - Maps some of the I/O to onboard LEDs.

17.3 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.

Chapter 18

Opto22 Driver

PCI AC5 ADAPTER CARD / HAL DRIVER

18.1 The Adapter Card

This is a card made by Opto22 for adapting the PCI port to solid state relay racks such as their standard or G4 series. It has 2 ports that can control up to 24 points each and has 4 on board LEDs. The ports use 50 pin connectors the same as Mesa boards. Any relay racks/breakout boards thats work with Mesa Cards should work with this card with the understanding any encoder counters, PWM, etc., would have to be done in software. The AC5 does not have any *smart* logic on board, it is just an adapter.

See the manufacturer's website for more info:

http://www.opto22.com/site/pr_details.aspx?item=PCI-AC5&qs=100110021011,,,1,3

I would like to thank Opto22 for releasing info in their manual, easing the writing of this driver!

18.2 The Driver

This driver is for the PCI AC5 card and will not work with the ISA AC5 card. The HAL driver is a realtime module. It will support 4 cards as is (more cards are possible with a change in the source code). Load the basic driver like so:

```
loadrt opto_ac5
```

This will load the driver which will search for max 4 boards. It will set I/O of each board's 2 ports to a default setting. The default configuration is for 12 inputs then 12 outputs. The pin name numbers correspond to the position on the relay rack. For example the pin names for the default I/O setting of port 0 would be:

- *opto_ac5.0.port0.in-00* - They would be numbered from 00 to 11
- *opto_ac5.0.port0.out-12* - They would be numbered 12 to 23 port 1 would be the same.

18.3 Pins

- *opto_ac5.[BOARDNUMBER].port[PORTNUMBER].in-[PINNUMBER] OUT bit* -
- *opto_ac5.[BOARDNUMBER].port[PORTNUMBER].in-[PINNUMBER]-not OUT bit* - Connect a HAL bit signal to this pin to read an I/O point from the card. The PINNUMBER represents the position in the relay rack. Eg. PINNUMBER 0 is position 0 in a Opto22 relay rack and would be pin 47 on the 50 pin header connector. The -not pin is inverted so that LOW gives TRUE and HIGH gives FALSE.

- *opto_ac5.[BOARDNUMBER].port[PORTNUMBER].out-[PINNUMBER] IN bit* - Connect a HAL bit signal to this pin to write to an I/O point of the card. The PINNUMBER represents the position in the relay rack.Eg. PINNUMBER 23 is position 23 in a Opto22 relay rack and would be pin 1 on the 50 pin header connector.
- *opto_ac5.[BOARDNUMBER].led[NUMBER] OUT bit* - Turns one of the 4 onboard LEDs on/off. LEDs are numbered 0 to 3.

BOARDNUMBER can be 0-3 PORTNUMBER can be 0 or 1. Port 0 is closest to the card bracket.

18.4 Parameters

- *opto_ac5.[BOARDNUMBER].port[PORTNUMBER].out-[PINNUMBER]-invert W bit* - When TRUE, invert the meaning of the corresponding -out pin so that TRUE gives LOW and FALSE gives HIGH.

18.5 FUNCTIONS

- *opto_ac5.0.digital-read* - Add this to a thread to read all the input points.
- *opto_ac5.0.digital-write* - Add this to a thread to write all the output points and LEDs.

For example the pin names for the default I/O setting of port 0 would be:

```
opto_ac5.0.port0.in-00
```

They would be numbered from 00 to 11

```
opto_ac5.0.port0.out-12
```

They would be numbered 12 to 23 port 1 would be the same.

18.6 Configuring I/O Ports

To change the default setting load the driver something like so:

```
loadrt opto_ac5 portconfig0=0xffff portconfig1=0xff0000
```

Of course changing the numbers to match the I/O you would like. Each port can be set up different.

Here's how to figure out the number: The configuration number represents a 32 bit long code to tell the card which I/O points are output vrs input. The lower 24 bits are the I/O points of one port. The 2 highest bits are for 2 of the on board LEDs. A one in any bit position makes the I/O point an output. The two highest bits must be output for the LEDs to work. The driver will automatically set the two highest bits for you, we won't talk about them.

The easiest way to do this is to fire up the calculator under APPLICATIONS/ACCESSORIES. Set it to scientific (click view). Set it BINARY (radio button Bin). Press 1 for every output you want and/or zero for every input. Remember that HAL pin 00 corresponds to the rightmost bit. 24 numbers represent the 24 I/O points of one port. So for the default setting (12 inputs then 12 outputs) you would push 1 twelve times (thats the outputs) then 0 twelve times (thats the inputs). Notice the first I/O point is the lowest (rightmost) bit. (that bit corresponds to HAL pin 00 .looks backwards) You should have 24 digits on the screen. Now push the Hex radio button. The displayed number (fff000) is the configport number (put a *0x* in front of it designating it as a HEX number).

Another example: To set the port for 8 outputs and 16 inputs (the same as a Mesa card). Here is the 24 bits represented in a BINARY number. Bit 1 is the rightmost number.

000000000000000011111111

16 zeros for the 16 inputs and 8 ones for the 8 outputs

Which converts to FF on the calculator so 0xff is the number to use for portconfig0 and/or portconfig1 when loading the driver.

18.7 Pin Numbering

HAL pin 00 corresponds to bit 1 (the rightmost) which represents position 0 on an Opto22 relay rack. HAL pin 01 corresponds to bit 2 (one spot to the left of the rightmost) which represents position 1 on an Opto22 relay rack. HAL pin 23 corresponds to bit 24 (the leftmost) which represents position 23 on an Opto22 relay rack.

HAL pin 00 connects to pin 47 on the 50 pin connector of each port. HAL pin 01 connects to pin 45 on the 50 pin connector of each port. HAL pin 23 connects to pin 1 on the 50 pin connector of each port.

Note that Opto22 and Mesa use opposite numbering systems: Opto22 position 23 = connector pin 1, and the position goes down as the connector pin number goes up. Mesa Hostmot2 position 1 = connector pin 1, and the position number goes up as the connector pin number goes up.

Chapter 19

Pico Drivers

Pico Systems has a family of boards for doing analog servo, stepper, and PWM (digital) servo control. The boards connect to the PC through a parallel port working in EPP mode. Although most users connect one board to a parallel port, in theory any mix of up to 8 or 16 boards can be used on a single parport. One driver serves all types of boards. The final mix of I/O depends on the connected board(s). The driver doesn't distinguish between boards, it simply numbers I/O channels (encoders, etc) starting from 0 on the first board. The driver is named hal_ppmc.ko The analog servo interface is also called the PPMC for Parallel Port Motion Control. There is also the Universal Stepper Controller, abbreviated the USC. And the Universal PWM Controller, or UPC.

Installing:

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

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

Up to 3 parport busses may be used, and each bus may have up to 8 (or possibly 16 PPMC) devices on it.

19.1 Pins

In the following pins, parameters, and functions, <port> is the parallel port ID. According to the naming conventions the first port should always have an ID of zero. All the boards have some method of setting the address on the EPP bus. USC and UPC have simple provisions for only two addresses, but jumper foil cuts allow up to 4 boards to be addressed. The PPMC boards have 16 possible addresses. In all cases, the driver enumerates the boards by type and exports the appropriate HAL pins. For instance, the encoders will be enumerated from zero up, in the same order as the address switches on the board specify. So, the first board will have encoders 0 — 3, the second board would have encoders 4 — 7. The first column after the bullet tells which boards will have this HAL pin or parameter associated with it. All means that this pin is available on all three board types. Option means that this pin will only be exported when that option is enabled by an optional parameter in the loadrt HAL command. These options require the board to have a sufficient revision level to support the feature.

- *(All s32 output) ppmc.<port>.encoder.<channel>.count* - Encoder position, in counts.
- *(All s32 output) ppmc.<port>.encoder.<channel>.delta* - Change in counts since last read, in raw encoder count units.
- *(All float output) 'ppmc.<port>.encoder.<channel>.velocity* - Velocity scaled in user units per second. On PPMC and USC this is derived from raw encoder counts per servo period, and hence is affected by encoder granularity. On UPC boards with the 8/21/09 and later firmware, velocity estimation by timestamping encoder counts can be used to improve the smoothness of this velocity output. This can be fed to the PID HAL component to produce a more stable servo response. This function has to be enabled in the HAL command line that starts the PPMC driver, with the timestamp=0x00 option.

- *(All float output) ppmc.<port>.encoder.<channel>.position* - Encoder position, in user units.
- *(All bit bidir) ppmc.<port>.encoder.<channel>.index-enable* - Connect to axis.#.index-enable for home-to-index. This is a bidirectional HAL signal. Setting it to true causes the encoder hardware to reset the count to zero on the next encoder index pulse. The driver will detect this and set the signal back to false.
- *(PPMC float output) ppmc.<port>.DAC.<channel>.value* - sends a signed value to the 16-bit Digital to Analog Converter on the PPMC DAC16 board commanding the analog output voltage of that DAC channel.
- *(UPC bit input) ppmc.<port>.pwm.<channel>.enable* - Enables a PWM generator.
- *(UPC float input) ppmc.<port>.pwm.<channel>.value* - Value which determines the duty cycle of the PWM waveforms. The value is divided by *pwm.<channel>.scale*, and if the result is 0.6 the duty cycle will be 60%, and so on. Negative values result in the duty cycle being based on the absolute value, and the direction pin is set to indicate negative.
- *(USC bit input) ppmc.<port>.stepgen.<channel>.enable* - Enables a step pulse generator.
- *(USC float input) ppmc.<port>.stepgen.<channel>.velocity* - Value which determines the step frequency. The value is multiplied by *stepgen.<channel>.scale* , and the result is the frequency in steps per second. Negative values result in the frequency being based on the absolute value, and the direction pin is set to indicate negative.
- *(All bit output) ppmc.<port>.din.<channel>.in* - State of digital input pin, see canonical digital input.
- *(All bit output) ppmc.<port>.din.<channel>.in-not* - Inverted state of digital input pin, see canonical digital input.
- *(All bit input) ppmc.<port>.dout.<channel>.out* - Value to be written to digital output, see canonical digital output.
- *(Option float input) ppmc.<port>.DAC8-<channel>.value* - Value to be written to analog output, range from 0 to 255. This sends 8 output bits to J8, which should have a Spindle DAC board connected to it. 0 corresponds to zero Volts, 255 corresponds to 10 Volts. The polarity of the output can be set for always minus, always plus, or can be controlled by the state of SSR1 (plus when on) and SSR2 (minus when on). You must specify extradac = 0x00 on the HAL command line that loads the PPMC driver to enable this function on the first USC ur UPC board.
- *(Option bit input) ppmc.<port>.dout.<channel>.out* - Value to be written to one of the 8 extra digital output pins on J8. You must specify extradout = 0x00 on the HAL command line that loads the ppmc driver to enable this function on the first USC or UPC board. extradac and extradout are mutually exclusive features as they use the same signal lines for different purposes. These output pins will be enumerated after the standard digital outputs of the board.

19.2 Parameters

- *(All float) ppmc.<port>.enc.<channel>.scale* - The number of counts / user unit (to convert from counts to units).
- *(UPC float) ppmc.<port>.pwm.<channel-range>.freq* - The PWM carrier frequency, in Hz. Applies to a group of four consecutive PWM generators, as indicated by *<channel-range>*. Minimum is 610Hz, maximum is 500KHz.
- *(PPMC float) ppmc.<port>.DAC.<channel>.scale* - Sets scale of DAC16 output channel such that an output value equal to the 1/scale value will produce an output of + or - value Volts. So, if the scale parameter is 0.1 and you send a value of 0.5, the output will be 5.0 Volts.
- *(UPC float) ppmc.<port>.pwm.<channel>.scale* - Scaling for PWM generator. If *scale* is X, then the duty cycle will be 100% when the *value* pin is X (or -X).
- *(UPC float) ppmc.<port>.pwm.<channel>.max-dc* - Maximum duty cycle, from 0.0 to 1.0.
- *(UPC float) ppmc.<port>.pwm.<channel>.min-dc* - Minimum duty cycle, from 0.0 to 1.0.
- *(UPC float) ppmc.<port>.pwm.<channel>.duty-cycle* - Actual duty cycle (used mostly for troubleshooting.)
- *(UPC bit) ppmc.<port>.pwm.<channel>.bootstrap* - If true, the PWM generator will generate a short sequence of pulses of both polarities when E-stop goes false, to charge the bootstrap capacitors used on some MOSFET gate drivers.

- *(USC u32) ppmc.<port>.stepgen.<channel-range>.setup-time* - Sets minimum time between direction change and step pulse, in units of 100ns. Applies to a group of four consecutive step generators, as indicated by <*channel-range*>.
- *(USC u32) ppmc.<port>.stepgen.<channel-range>.pulse-width* - Sets width of step pulses, in units of 100ns. Applies to a group of four consecutive step generators, as indicated by <*channel-range*>.
- *(USC u32) ppmc.<port>.stepgen.<channel-range>.pulse-space-min* - Sets minimum time between pulses, in units of 100ns. Applies to a group of four consecutive step generators, as indicated by <*channel-range*>. The maximum step rate is: $\frac{1}{100\text{ns} * (pulsewidth + pulsespacemin)}$
- *(USC float) ppmc.<port>.stepgen.<channel>.scale* - Scaling for step pulse generator. The step frequency in Hz is the absolute value of *velocity* * *scale*.
- *(USC float) ppmc.<port>.stepgen.<channel>.max-vel* - The maximum value for *velocity*. Commands greater than *max-vel* will be clamped. Also applies to negative values. (The absolute value is clamped.)
- *(USC float) ppmc.<port>.stepgen.<channel>.frequency* - Actual step pulse frequency in Hz (used mostly for troubleshooting.)
- *(Option float) ppmc.<port>.DAC8.<channel>.scale* - Sets scale of extra DAC output such that an output value equal to scale gives a magnitude of 10.0 V output. (The sign of the output is set by jumpers and/or other digital outputs.)
- *(Option bit) ppmc.<port>.dout.<channel>.invert* - Inverts a digital output, see canonical digital output.
- *(Option bit) ppmc.<port>.dout.<channel>.invert* - Inverts a digital output pin of J8, see canonical digital output.

19.3 Functions

- *(All funct) ppmc.<port>.read* - Reads all inputs (digital inputs and encoder counters) on one port. These reads are organized into blocks of contiguous registers to be read in a block to minimize CPU overhead.
- *(All funct) ppmc.<port>.write* - Writes all outputs (digital outputs, stepgens, PWMs) on one port. These writes are organized into blocks of contiguous registers to be written in a block to minimize CPU overhead.

Chapter 20

Pluto P Driver

20.1 General Info

The Pluto-P is a FPGA board featuring the ACEX1K chip from Altera.

20.1.1 Requirements

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

Note
The Pluto P board requires EPP mode. Netmos98xx chips do not work in EPP mode. The Pluto P board will work on some computers and not on others. There is no known pattern to which computers work and which don't work.

For more information on PCI EPP compatible parallel port cards see the LinuxCNC Supported Hardware page on the wiki.

20.1.2 Connectors

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

20.1.3 Physical Pins

- Read the ACEX1K datasheet for information about input and output voltage thresholds. The pins are all configured in *LVTTL/LVCMOS* mode and are generally compatible with 5V TTL logic.
- Before configuration and after properly exiting LinuxCNC, all Pluto-P pins are tristated with weak pull-ups (20k-ohms min, 50k-ohms max). If the watchdog timer is enabled (the default), these pins are also tristated after an interruption of communication between LinuxCNC and the board. The watchdog timer takes approximately 6.5ms to activate. However, software bugs in the pluto_servo firmware or LinuxCNC can leave the Pluto-P pins in an undefined state.
- In pwm+dir mode, by default dir is HIGH for negative values and LOW for positive values. To select HIGH for positive values and LOW for negative values, set the corresponding dout-NN-invert parameter TRUE to invert the signal.

- The index input is triggered on the rising edge. Initial testing has shown that the QZx inputs are particularly noise sensitive, due to being polled every 25ns. Digital filtering has been added to filter pulses shorter than 175ns (seven polling times). Additional external filtering on all input pins, such as a Schmitt buffer or inverter, RC filter, or differential receiver (if applicable) is recommended.
- The IN1. . . IN7 pins have 22-ohm series resistors to their associated FPGA pins. No other pins have any sort of protection for out-of-spec voltages or currents. It is up to the integrator to add appropriate isolation and protection. Traditional parallel port optoisolator boards do not work with pluto_servo due to the bidirectional nature of the EPP protocol.

20.1.4 LED

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

20.1.5 Power

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

20.1.6 PC interface

- Only a single pluto_servo or pluto_step board is supported.

20.1.7 Rebuilding the FPGA firmware

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

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

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

20.1.8 For more information

Some additional information about it is available from KNJC LLC and from the developer's blog.

20.2 Pluto Servo

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

This driver features:

- 4 quadrature channels with 40MHz sample rate. The counters operate in *4x* mode. The maximum useful quadrature rate is 8191 counts per LinuxCNC servo cycle, or about 8MHz for LinuxCNC's default 1ms servo rate.
- 4 PWM channels, *up/down* or *pwm+dir* style. 4095 duty cycles from -100% to +100%, including 0%. The PWM period is approximately 19.5kHz (40MHz / 2047). A PDM-like mode is also available.

- 18 digital outputs: 10 dedicated, 8 shared with PWM functions. (Example: A lathe with unidirectional PWM spindle control may use 13 total digital outputs)
- 20 digital inputs: 8 dedicated, 12 shared with Quadrature functions. (Example: A lathe with index pulse only on the spindle may use 13 total digital inputs)
- EPP communication with the PC. The EPP communication typically takes around 100 us on machines tested so far, enabling servo rates above 1kHz.

20.2.1 Pinout

- *UPx* - The *up* (up/down mode) or *pwm* (pwm+direction mode) signal from PWM generator X. May be used as a digital output if the corresponding PWM channel is unused, or the output on the channel is always negative. The corresponding digital output invert may be set to TRUE to make UPx active low rather than active high.
- *DNx* - The *down* (up/down mode) or *direction* (pwm+direction mode) signal from PWM generator X. May be used as a digital output if the corresponding PWM channel is unused, or the output on the channel is never negative. The corresponding digital ouput invert may be set to TRUE to make DNx active low rather than active high.
- *QAx, QBx* - The A and B signals for Quadrature counter X. May be used as a digital input if the corresponding quadrature channel is unused.
- *QZx* - The Z (index) signal for quadrature counter X. May be used as a digital input if the index feature of the corresponding quadrature channel is unused.
- *INx* - Dedicated digital input #x
- *OUTx* - Dedicated digital output #x
- *GND* - Ground
- *VCC* - +3.3V regulated DC

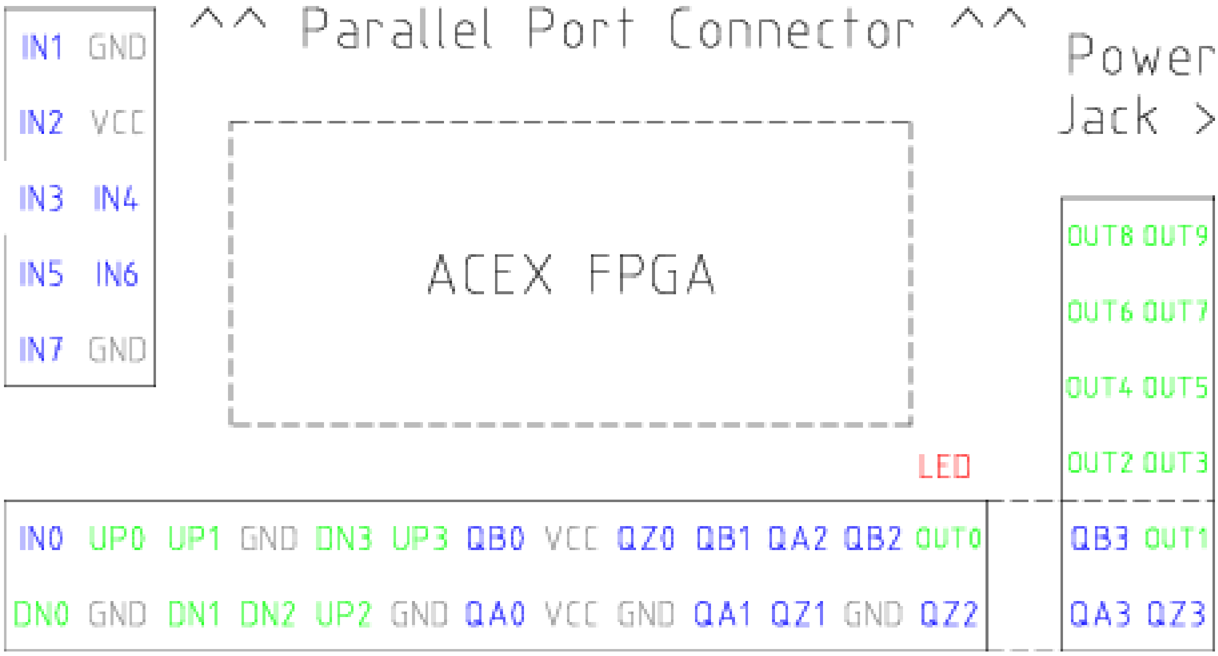


Figure 20.1: Pluto-Servo Pinout

Table 20.1: 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

20.2.2 Input latching and output updating

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

20.2.3 HAL Functions, Pins and Parameters

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

20.2.4 Compatible driver hardware

A schematic for a 2A, 2-axis PWM servo amplifier board is available from the (the software developer). The L298 H-Bridge can be used for motors up to 4A (one motor per L298) or up to 2A (two motors per L298) with the supply voltage up to 46V. However, the L298 does not have built-in current limiting, a problem for motors with high stall currents. For higher currents and voltages, some users have reported success with International Rectifier's integrated high-side/low-side drivers.

20.3 Pluto Step

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

The board features:

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

20.3.1 Pinout

- *STEPx* - The *step* (clock) output of stepgen channel *x*
- *DIRx* - The *direction* output of stepgen channel *x*
- *INx* - Dedicated digital input #x
- *OUTx* - Dedicated digital output #x
- *GND* - Ground
- *VCC* - +3.3V regulated DC

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

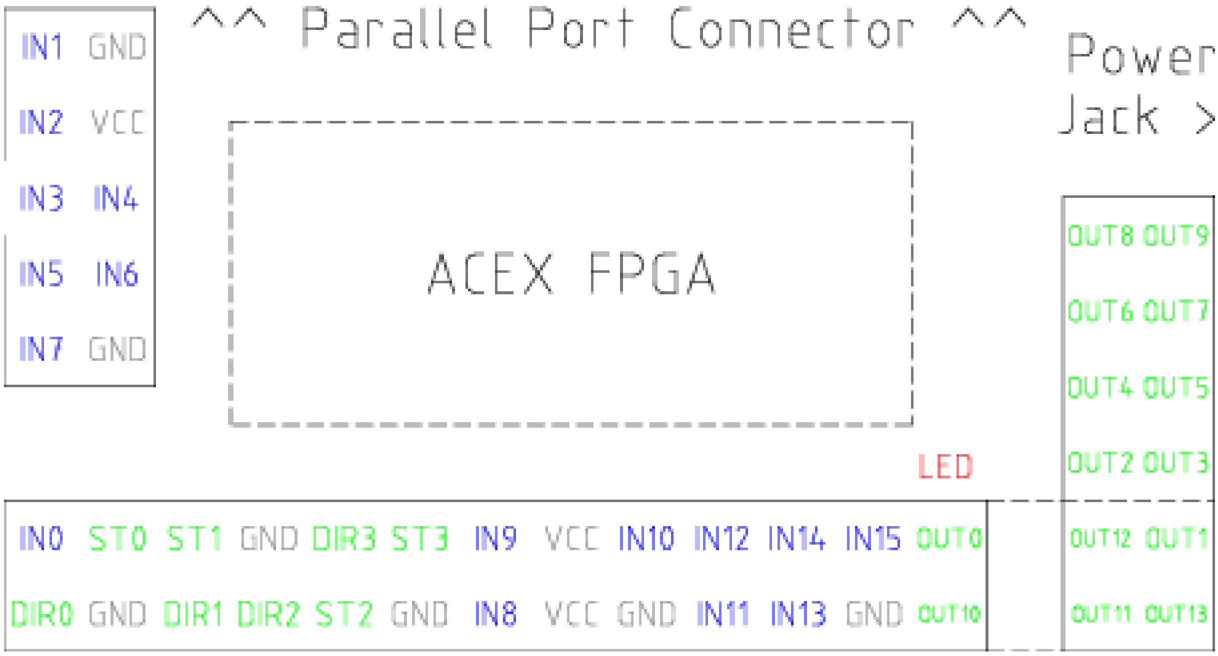


Figure 20.2: Pluto-Step Pinout

20.3.2 Input latching and output updating

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

20.3.3 Step Waveform Timings

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

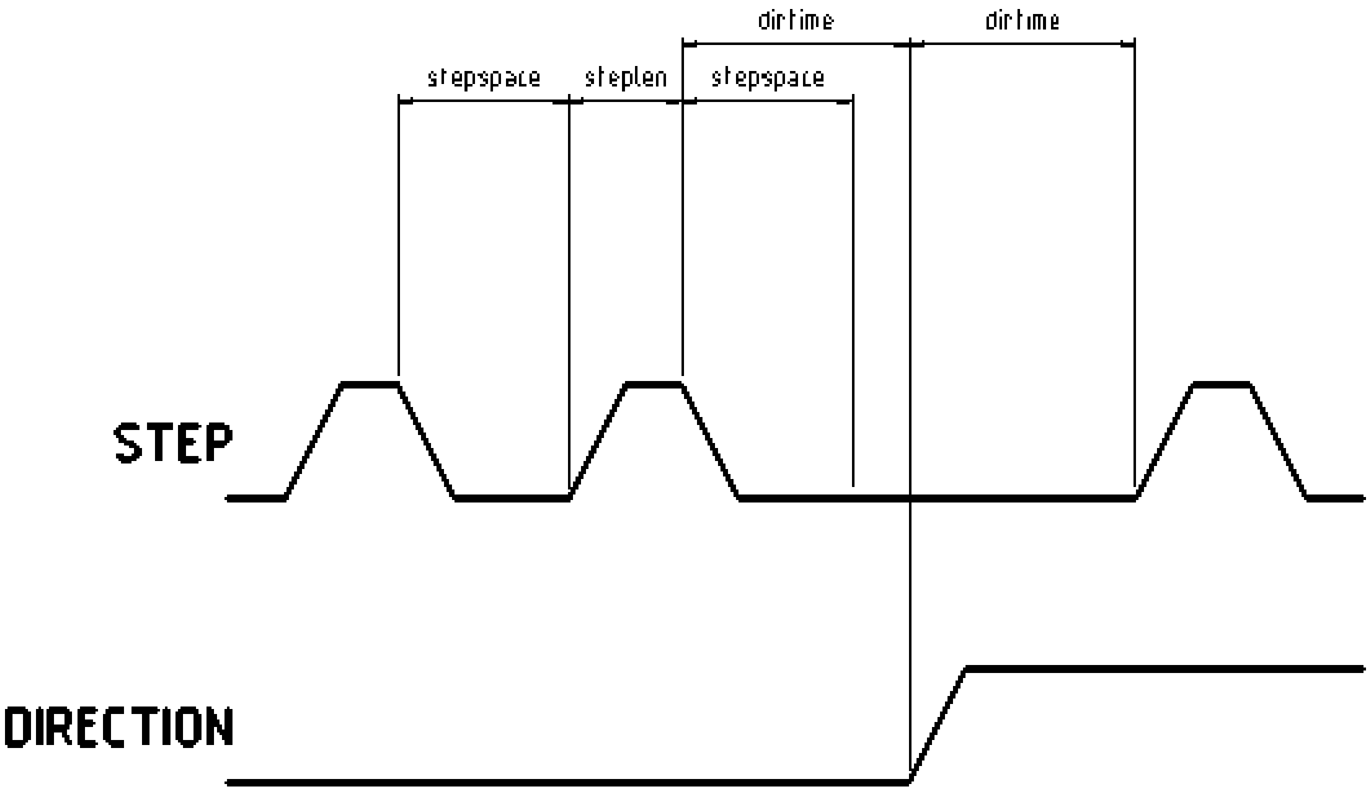


Figure 20.3: Pluto-Step Timings

20.3.4 HAL Functions, Pins and Parameters

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

Chapter 21

Servo To Go Driver

The Servo-To-Go (STG) is one of the first PC motion control cards supported by LinuxCNC. It is an ISA card and it exists in different flavors (all supported by this driver). The board includes up to 8 channels of quadrature encoder input, 8 channels of analog input and output, 32 bits digital I/O, an interval timer with interrupt and a watchdog. For more information see the Servo To Go web page.

21.1 Installing

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

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

hint: after starting up the driver, *dmesg* can be consulted for messages relevant to the driver (e.g. autodetected version number and base address). For example:

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

This example installs the STG driver for a card found at the base address of 0x300, 4 channels of encoder feedback, DACs and ADCs, along with 32 bits of I/O configured like this: the first 8 (Port A) configured as Input, the next 8 (Port B) configured as Output, the next 8 (Port C) configured as Input, and the last 8 (Port D) configured as Output

```
loadrt hal_stg
```

This example installs the driver and attempts to autodetect the board address and board model, it installs 8 axes by default along with a standard I/O setup: Port A & B configured as Input, Port C & D configured as Output.

21.2 Pins

- *stg.<channel>.counts* - (s32) Tracks the counted encoder ticks.
- *stg.<channel>.position* - (float) Outputs a converted position.
- *stg.<channel>.dac-value* - (float) Drives the voltage for the corresponding DAC.

- *stg.<channel>.adc-value* - (float) Tracks the measured voltage from the corresponding ADC.
- *stg.in-<pinnum>* - (bit) Tracks a physical input pin.
- *stg.in-<pinnum>-not* - (bit) Tracks a physical input pin, but inverted.
- *stg.out-<pinnum>* - (bit) Drives a physical output pin

For each pin, <channel> is the axis number, and <pinnum> is the logic pin number of the STG if `IIOO` is defined, there are 16 input pins (in-00 .. in-15) and 16 output pins (out-00 .. out-15), and they correspond to PORTs ABCD (in-00 is PORTA.0, out-15 is PORTD.7).

The in-<pinnum> HAL pin is TRUE if the physical pin is high, and FALSE if the physical pin is low. The in-<pinnum>-not HAL pin is inverted — it is FALSE if the physical pin is high. By connecting a signal to one or the other, the user can determine the state of the input.

21.3 Parameters

- *stg.<channel>.position-scale* - (float) The number of counts / user unit (to convert from counts to units).
- *stg.<channel>.dac-offset* - (float) Sets the offset for the corresponding DAC.
- *stg.<channel>.dac-gain* - (float) Sets the gain of the corresponding DAC.
- *stg.<channel>.adc-offset* - (float) Sets the offset of the corresponding ADC.
- *stg.<channel>.adc-gain* - (float) Sets the gain of the corresponding ADC.
- *stg.out-<pinnum>-invert* - (bit) Inverts an output pin.

The -invert parameter determines whether an output pin is active high or active low. If -invert is FALSE, setting the HAL out-pin TRUE drives the physical pin high, and FALSE drives it low. If -invert is TRUE, then setting the HAL out- pin TRUE will drive the physical pin low.

21.3.1 Functions

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

Chapter 22

ShuttleXpress

22.1 Description

shuttlexpress is a userspace HAL component that interfaces Contour Design's ShuttleXpress device with LinuxCNC's HAL. The ShuttleXpress has five momentary buttons, a 10 counts/revolution jog wheel with detents, and a 15-position spring-loaded outer wheel that returns to center when released.

If it is started without command-line arguments, it will probe all /dev/hidraw* device files for ShuttleXpress devices, and use all devices found. If it is started with command-line arguments, only will only probe the devices specified.

> **Warning**
> The ShuttleXpress device has an internal 8-bit counter for the current jog-wheel position. The shuttlexpress driver can not know this value until the ShuttleXpress device sends its first event. When the first event comes into the driver, the driver uses the device's reported jog-wheel position to initialize counts to 0.

This means that if the first event is generated by a jog-wheel move, that first move will be lost.

Any user interaction with the ShuttleXpress device will generate an event, informing the driver of the jog-wheel position. So if you (for example) push one of the buttons at startup, the jog-wheel will work fine and notice the first click.

22.2 Setup

The shuttlexpress module needs read permission on the /dev/hidraw* device files. This can be accomplished by adding a file /etc/udev/rules.d/99-shuttlexpress.rules, with the following contents:

```
SUBSYSTEM=="hidraw", ATTRS{idVendor}=="0b33", ATTRS{idProduct}=="0020", MODE="0444"
```

22.3 Pins

(bit out) shuttlexpress.0.button-0 (bit out) shuttlexpress.0.button-0-not (bit out) shuttlexpress.0.button-1 (bit out) shuttlexpress.0.button-1-not (bit out) shuttlexpress.0.button-2 (bit out) shuttlexpress.0.button-2-not (bit out) shuttlexpress.0.button-3 (bit out) shuttlexpress.0.button-3-not (bit out) shuttlexpress.0.button-4 (bit out) shuttlexpress.0.button-4-not

The five buttons around the outside, starting with the counter-clockwise-most one.

(s32 out) shuttlexpress.0.counts

Accumulated counts from the jog wheel (the inner wheel).

(s32 out) shuttlexpress.0.spring-wheel-s32

The current deflection of the spring-wheel (the outer wheel). It's 0 at rest, and ranges from -7 at the counter-clockwise extreme to +7 at the clockwise extreme.

(float out) shuttlexpress.0.spring-wheel-f

The current deflection of the spring-wheel (the outer wheel). It's 0 at rest, -1 at the counter-clockwise extreme, and +1 at the clockwise extreme. (The ShuttleXpress device reports the spring-wheel position quantized from -7 to +7, so this pin reports only 15 discrete values in it's range.)

Part V

Advanced Topics

Chapter 23

Kinematics

23.1 Introduction

When we talk about CNC machines, we usually think about machines that are commanded to move to certain locations and perform various tasks. In order to have an unified view of the machine space, and to make it fit the human point of view over 3D space, most of the machines (if not all) use a common coordinate system called the Cartesian Coordinate System.

The Cartesian Coordinate system is composed of three axes (X, Y, Z) each perpendicular to the other two. [1]

When we talk about a G-code program (RS274/NGC) we talk about a number of commands (G0, G1, etc.) which have positions as parameters (X- Y- Z-). These positions refer exactly to Cartesian positions. Part of the LinuxCNC motion controller is responsible for translating those positions into positions which correspond to the machine kinematics. [2]

23.1.1 Joints vs. Axes

A joint of a CNC machine is a one of the physical degrees of freedom of the machine. This might be linear (leadscrews) or rotary (rotary tables, robot arm joints). There can be any number of joints on a given machine. For example, one popular robot has 6 joints, and a typical simple milling machine has only 3.

There are certain machines where the joints are laid out to match kinematics axes (joint 0 along axis X, joint 1 along axis Y, joint 2 along axis Z), and these machines are called Cartesian machines (or machines with Trivial Kinematics). These are the most common machines used in milling, but are not very common in other domains of machine control (e.g. welding: puma-typed robots).

23.2 Trivial Kinematics

The simplest machines are those in which which each joint is placed along one of the Cartesian axes. On these machines the mapping from Cartesian space (the G-code program) to the joint space (the actual actuators of the machine) is trivial. It is a simple 1:1 mapping:

```
pos->tran.x = joints[0];
pos->tran.y = joints[1];
pos->tran.z = joints[2];
pos->a = joints[3];
pos->b = joints[4];
pos->c = joints[5];
```

[1] The word “axes” is also commonly (and wrongly) used when talking about CNC machines, and referring to the moving directions of the machine.

[2] Kinematics: a two way function to transform from Cartesian space to joint space

In the above code snippet one can see how the mapping is done: the X position is identical with the joint 0, the Y posittion with with joint 1, etc. The above refers to the direct kinematics (one direction of the transformation). The next code snippet refers to the inverse kinematics (or the inverse direction of the transformation):

```
joints[0] = pos->tran.x;
joints[1] = pos->tran.y;
joints[2] = pos->tran.z;
joints[3] = pos->a;
joints[4] = pos->b;
joints[5] = pos->c;
```

As one can see, it's pretty straightforward to do the transformation for a trivial "kins" (kinematics) or Cartesian machine. It gets a bit more complicated if the machine is missing one of the axes.[3] [4]

23.3 Non-trivial kinematics

There can be quite a few types of machine setups (robots: puma, scara; hexapods etc.). Each of them is set up using linear and rotary joints. These joints don't usually match with the Cartesian coordinates, therefore we need a kinematics function which does the conversion (actually 2 functions: forward and inverse kinematics function).

To illustrate the above, we will analyze a simple kinematics called bipod (a simplified version of the tripod, which is a simplified version of the hexapod).

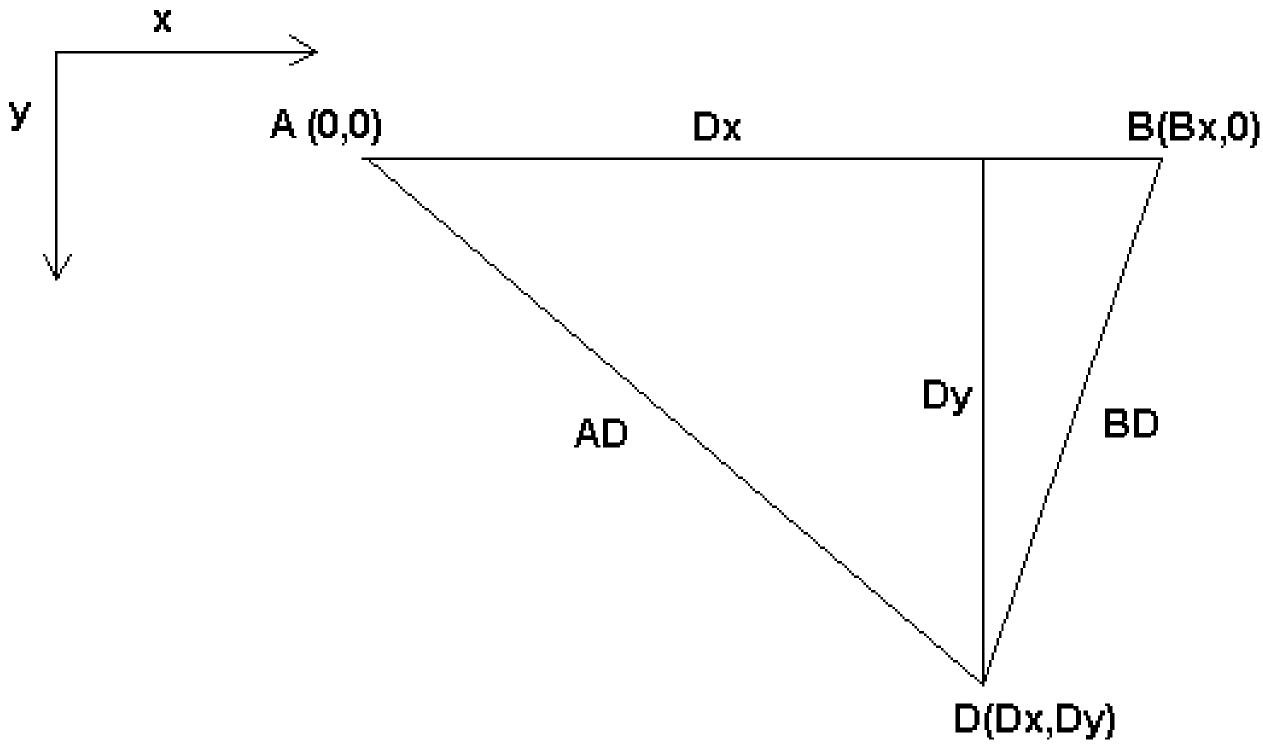


Figure 23.1: Bipod setup

[3]If a machine (e.g. a lathe) is set up with only the axes X,Z & A, and the LinuxCNC inifile holds only these 3 joints defined, then the above matching will be faulty. That is because we actually have (joint0=x, joint1=Z, joint2=A) whereas the above assumes joint1=Y. To make it easily work in LinuxCNC one needs to define all axes (XYZA), then use a simple loopback in HAL for the unused Y axis.

[4]One other way of making it work, is by changing the matching code and recompiling the software.

The Bipod we are talking about is a device that consists of 2 motors placed on a wall, from which a device is hung using some wire. The joints in this case are the distances from the motors to the device (named AD and BD in the figure).

The position of the motors is fixed by convention. Motor A is in (0,0), which means that its X coordinate is 0, and its Y coordinate is also 0. Motor B is placed in (Bx, 0), which means that its X coordinate is Bx.

Our tooltip will be in point D which gets defined by the distances AD and BD, and by the Cartesian coordinates Dx, Dy.

The job of the kinematics is to transform from joint lengths (AD, BD) to Cartesian coordinates (Dx, Dy) and vice-versa.

23.3.1 Forward transformation

To transform from joint space into Cartesian space we will use some trigonometry rules (the right triangles determined by the points (0,0), (Dx,0), (Dx,Dy) and the triangle (Dx,0), (Bx,0) and (Dx,Dy).

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

If we subtract one from the other we will get:

$$AD^2 - BD^2 = x^2 + y^2 - x^2 + 2 * x * Bx - Bx^2 - y^2$$

and therefore:

$$x = \frac{AD^2 - BD^2 + Bx^2}{2 * Bx}$$

From there we calculate:

$$y = \sqrt{AD2 - x2}$$

Note that the calculation for y involves the square root of a difference, which may not result in a real number. If there is no single Cartesian coordinate for this joint position, then the position is said to be a singularity. In this case, the forward kinematics return -1.

Translated to actual code:

```
double AD2 = joints[0] * joints[0];
double BD2 = joints[1] * joints[1];
double x = (AD2 - BD2 + Bx * Bx) / (2 * Bx);
double y2 = AD2 - x * x;
if(y2 < 0) return -1;
pos->tran.x = x;
pos->tran.y = sqrt(y2);
return 0;
```

23.3.2 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;
```

23.4 Implementation details

A kinematics module is implemented as a HAL component, and is permitted to export pins and parameters. It consists of several "C" functions (as opposed to HAL functions):

```
int kinematicsForward(const double *joint, EmcPose *world,
const KINEMATICS_FORWARD_FLAGS *fflags,
KINEMATICS_INVERSE_FLAGS *iflags)
```

Implements the forward kinematics function.

```
int kinematicsInverse(const EmcPose * world, double *joints,
const KINEMATICS_INVERSE_FLAGS *iflags,
KINEMATICS_FORWARD_FLAGS *fflags)
```

Implements the inverse kinematics function.

```
KINEMATICS_TYPE kinematicsType(void)
```

Returns the kinematics type identifier, typically *KINEMATICS_BOTH*.

```
int kinematicsHome(EmcPose *world, double *joint,
KINEMATICS_FORWARD_FLAGS *fflags,
KINEMATICS_INVERSE_FLAGS *iflags)
```

The home kinematics function sets all its arguments to their proper values at the known home position. When called, these should be set, when known, to initial values, e.g., from an INI file. If the home kinematics can accept arbitrary starting points, these initial values should be used.

```
int rtapi_app_main(void)
void rtapi_app_exit(void)
```

These are the standard setup and tear-down functions of RTAPI modules.

When they are contained in a single source file, kinematics modules may be compiled and installed by *comp*. See the *comp(1)* manpage or the HAL manual for more information.

Chapter 24

Stepper Tuning

24.1 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.

24.1.1 Run a Latency Test

Run the latency test as described in the Latency Test chapter.

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

The last number in the column labeled *ovl max* is the most important. Write it down - you will need it later. It contains the worst latency measurement during the entire run of the test. In the example above, that is 10636 nano-seconds, or 10.6 micro-seconds, which is excellent. However the example only ran for a few seconds (it prints one line every second). You should run the test for at least several minutes; sometimes the worst case latency doesn't happen very often, or only happens when you do some particular action. I had one Intel motherboard that worked pretty well most of the time, but every 64 seconds it had a very bad 300 us latency. Fortunately that is fixable, see FixingDapperSMIIssues in the wiki found at wiki.linuxcnc.org.

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

Note that if you get high numbers, there may be ways to improve them. For example, one PC had very bad latency (several milliseconds) when using the onboard video. But a $5 used Matrox video card solved the problem - EMC does not require bleeding edge hardware.

24.1.2 Figure out what your drives expect

Different brands of stepper drives have different timing requirements on their step and direction inputs. So you need to dig out (or Google for) the data sheet that has your drive's specs.

From the Gecko G202 manual:

```
Step Frequency: 0 to 200 kHz
Step Pulse "0" Time: 0.5 us min (Step on falling edge)
Step Pulse "1" Time: 4.5 us min
Direction Setup: 1 us min (20 us min hold time after Step edge)
```

From the Gecko G203V manual:

```
Step Frequency: 0 to 333 kHz
Step Pulse "0" Time: 2.0 us min (Step on rising edge)
Step Pulse "1" Time: 1.0 us min

Direction Setup:
    200 ns (0.2 us) before step pulse rising edge
    200 ns (0.2 us) hold after step pulse rising edge
```

From the Xylotex datasheet:

```
Minimum DIR setup time before rising edge of STEP Pulse 200 ns Minimum
DIR hold time after rising edge of STEP pulse 200 ns
Minimum STEP pulse high time 2.0 us
Minimum STEP pulse low time 1.0 us
Step happens on rising edge
```

Once you find the numbers, write them down too - you need them in the next step.

24.1.3 Choose your BASE_PERIOD

BASE_PERIOD is the *heartbeat* of your EMC computer. Every period, the software step generator decides if it is time for another step pulse. A shorter period will allow you to generate more pulses per second, within limits. But if you go too short, your computer will spend so much time generating step pulses that everything else will slow to a crawl, or maybe even lock up. Latency and stepper drive requirements affect the shortest period you can use, as we will see in a minute.

Let's look at the Gecko example first. The G202 can handle step pulses that go low for 0.5 us and high for 4.5 us, it needs the direction pin to be stable 1 us before the falling edge, and remain stable for 20 us after the falling edge. The longest timing requirement is the 20 us hold time. A simple approach would be to set the period at 20 us. That means that all changes on the STEP and DIR lines are separated by 20 us. All is good, right?

Wrong! If there was ZERO latency, then all edges would be separated by 20 us, and everything would be fine. But all computers have some latency. Latency means lateness. If the computer has 11 us of latency, that means sometimes the software runs as much as 11 us later than it was supposed to. If one run of the software is 11 us late, and the next one is on time, the delay from the first to the second is only 9 us. If the first one generated a step pulse, and the second one changed the direction bit, you just violated the 20 us G202 hold time requirement. That means your drive might have taken a step in the wrong direction, and your part will be the wrong size.

The really nasty part about this problem is that it can be very very rare. Worst case latencies might only happen a few times a minute, and the odds of bad latency happening just as the motor is changing direction are low. So you get very rare errors that ruin a part every once in a while and are impossible to troubleshoot.

The simplest way to avoid this problem is to choose a BASE_PERIOD that is the sum of the longest timing requirement of your drive, and the worst case latency of your computer. If you are running a Gecko with a 20 us hold time requirement, and your latency test said you have a maximum latency of 11 us, then if you set the BASE_PERIOD to 20+11 = 31 us (31000 nano-seconds in the ini file), you are guaranteed to meet the drive's timing requirements.

But there is a tradeoff. Making a step pulse requires at least two periods. One to start the pulse, and one to end it. Since the period is 31 us, it takes 2x31 = 62 us to create a step pulse. That means the maximum step rate is only 16,129 steps per second. Not so good. (But don't give up yet, we still have some tweaking to do in the next section.)

For the Xylotex, the setup and hold times are very short, 200 ns each (0.2 us). The longest time is the 2 us high time. If you have 11 us latency, then you can set the BASE_PERIOD as low as 11+2=13 us. Getting rid of the long 20 us hold time really helps! With a period of 13 us, a complete step takes 2x13 = 26 us, and the maximum step rate is 38,461 steps per second!

But you can't start celebrating yet. Note that 13 us is a very short period. If you try to run the step generator every 13 us, there might not be enough time left to run anything else, and your computer will lock up. If you are aiming for periods of less than 25 us, you should start at 25 us or more, run EMC, and see how things respond. If all is well, you can gradually decrease the period. If the mouse pointer starts getting sluggish, and everything else on the PC slows down, your period is a little too short. Go back to the previous value that let the computer run smoothly.

In this case, sppose you started at 25 us, trying to get to 13 us, but you find that around 16 us is the limit - any less and the computer doesn't respond very well. So you use 16 us. With a 16 us period and 11 us latency, the shortest output time will be 16-11 = 5 us. The drive only needs 2 us, so you have some margin. Margin is good - you don't want to lose steps because you cut the timing too close.

What is the maximum step rate? Remember, two periods to make a step. You settled on 16 us for the period, so a step takes 32 us. That works out to a not bad 31,250 steps per second.

24.1.4 Use steplen, stepspace, dirsetup, and/or dirhold

In the last section, we got the Xylotex drive to a 16 us period and a 31,250 step per second maximum speed. But the Gecko was stuck at 31 us and a not-so-nice 16,129 steps per second. The Xylotex example is as good as we can make it. But the Gecko can be improved.

The problem with the G202 is the 20 us hold time requirement. That plus the 11 us latency is what forces us to use a slow 31 us period. But the LinuxCNC software step generator has some parameters that let you increase the various time from one period to several. For example, if steplen is changed from 1 to 2, then it there will be two periods between the beginning and end of the step pulse. Likewise, if dirhold is changed from 1 to 3, there will be at least three periods between the step pulse and a change of the direction pin.

If we can use dirhold to meet the 20 us hold time requirement, then the next longest time is the 4.5 us high time. Add the 11 us latency to the 4.5 us high time, and you get a minimum period of 15.5 us. When you try 15.5 us, you find that the computer is sluggish, so you settle on 16 us. If we leave dirhold at 1 (the default), then the minimum time between step and direction is the 16 us period minus the 11 us latency = 5 us, which is not enough. We need another 15 us. Since the period is 16 us, we need one more period. So we change dirhold from 1 to 2. Now the minimum time from the end of the step pulse to the changing direction pin is 5+16=21 us, and we don't have to worry about the Gecko stepping the wrong direction because of latency.

If the computer has a latency of 11 us, then a combination of a 16 us base period, and a dirhold value of 2 ensures that we will always meet the timing requirements of the Gecko. For normal stepping (no direction change), the increased dirhold value has no effect. It takes two periods totalling 32 us to make each step, and we have the same 31,250 step per second rate that we got with the Xylotex.

The 11 us latency number used in this example is very good. If you work through these examples with larger latency, like 20 or 25 us, the top step rate for both the Xylotex and the Gecko will be lower. But the same formulas apply for calculating the optimum BASE_PERIOD, and for tweaking dirhold or other step generator parameters.

24.1.5 No Guessing!

For a fast AND reliable software based stepper system, you cannot just guess at periods and other configuration paremeters. You need to make measurements on your computer, and do the math to ensure that your drives get the signals they need.

To make the math easier, I've created an Open Office spreadsheet http://wiki.linuxcnc.org/uploads/StepTimingCalculator.ods You enter your latency test result and your stepper drive timing requirements and the spreadsheet calculates the optimum

BASE_PERIOD. Next, you test the period to make sure it won't slow down or lock up your PC. Finally, you enter the actual period, and the spreadsheet will tell you the stepgen parameter settings that are needed to meet your drive's timing requirements. It also calculates the maximum step rate that you will be able to generate.

I've added a few things to the spreadsheet to calculate max speed and stepper electrical calculations.

Chapter 25

PID Tuning

25.1 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 set point value. The difference (or *error* signal) is then used to calculate a new value for a manipulable input to the process that brings the process measured value back to its desired set point.

Unlike simpler control algorithms, the PID controller can adjust process outputs based on the history and rate of change of the error signal, which gives more accurate and stable control. (It can be shown mathematically that a PID loop will produce accurate, stable control in cases where a simple proportional control would either have a steady-state error or would cause the process to oscillate).

25.1.1 Control loop basics

Intuitively, the PID loop tries to automate what an intelligent operator with a gauge and a control knob would do. The operator would read a gauge showing the output measurement of a process, and use the knob to adjust the input of the process (the *action*) until the process's output measurement stabilizes at the desired value on the gauge.

In older control literature this adjustment process is called a *reset* action. The position of the needle on the gauge is a *measurement*, *process value* or *process variable*. The desired value on the gauge is called a *set point* (also called *set value*). The difference between the gauge's needle and the set point is the *error*.

A control loop consists of three parts:

1. Measurement by a sensor connected to the process (e.g. encoder),
2. Decision in a controller element,
3. Action through an output device such as an motor.

As the controller reads a sensor, it subtracts this measurement from the *set point* to determine the *error*. It then uses the error to calculate a correction to the process's input variable (the *action*) so that this correction will remove the error from the process's output measurement.

In a PID loop, correction is calculated from the error in three ways: cancel out the current error directly (Proportional), the amount of time the error has continued uncorrected (Integral), and anticipate the future error from the rate of change of the error over time (Derivative).

[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* controller's output, which is controlling based upon a related variable.

25.1.2 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:

25.1.2.1 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.

25.1.2.2 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 set point. A simple proportional system either oscillates, moving back and forth around the set point because there's nothing to remove the error when it overshoots, or oscillates and/or stabilizes at a too low or too high value. By adding a negative proportion of (i.e. subtracting part of) the average error from the process input, the average difference between the process output and the set point is always being reduced. Therefore, eventually, a well-tuned PID loop's process output will settle down at the set point.

25.1.2.3 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 set point.

25.1.3 Loop Tuning

Tuning a control loop is the adjustment of its control parameters (gain/proportional band, integral gain/reset, derivative gain/rate) to the optimum values for the desired control response. The optimum behavior on a process change or set point change varies depending on the application. Some processes must not allow an overshoot of the process variable from the set point. Other processes must minimize the energy expended in reaching a new set point. Generally stability of response is required and the process must not oscillate for any combination of process conditions and set points.

Tuning of loops is made more complicated by the response time of the process; it may take minutes or hours for a set point change to produce a stable effect. Some processes have a degree of non-linearity and so parameters that work well at full-load conditions don't work when the process is starting up from no-load. This section describes some traditional manual methods for loop tuning.

There are several methods for tuning a PID loop. The choice of method will depend largely on whether or not the loop can be taken *offline* for tuning, and the response speed of the system. If the system can be taken offline, the best tuning method often involves subjecting the system to a step change in input, measuring the output as a function of time, and using this response to determine the control parameters.

25.1.3.1 Simple method

If the system must remain on line, one tuning method is to first set the I and D values to zero. Increase the P until the output of the loop oscillates. Then increase I until oscillation stops. Finally, increase D until the loop is acceptably quick to reach its reference. A fast PID loop tuning usually overshoots slightly to reach the set point more quickly; however, some systems cannot accept overshoot.

Parameter	Rise Time	Overshoot	Settling Time	Steady State Error
P	Decrease	Increase	Small Change	Decrease
I	Decrease	Increase	Increase	Eliminate
D	Small Change	Decrease	Decrease	Small Change

Effects of increasing parameters

25.1.3.2 Ziegler-Nichols method

Another tuning method is formally known as the *Ziegler-Nichols method*, introduced by John G. Ziegler and Nathaniel B. Nichols. It starts in the same way as the method described before: first set the I and D gains to zero and then increase the P gain and expose the loop to external interference for example knocking the motor axis to cause it to move out of equilibrium in order to determine critical gain and period of oscillation until the output of the loop starts to oscillate. Write down the critical gain (K_c) and the oscillation period of the output (P_c). Then adjust the P, I and D controls as the table shows:

Control type	P	I	D
P	$.5K_c$		
PI	$.45K_c$	$P_c/1.2$	
PID	$.6K_c$	$P_c/2$	$P_c/8$

25.1.3.3 Final Steps

After tuning the axis check the following error with Halscope to make sure it is within your machine requirements. More information on Halscope is in the HAL User manual.

Part VI

Ladder Logic

Chapter 26

Classicladder Introduction

26.1 History

Classic Ladder is a free implementation of a ladder interpreter, released under the LGPL. It was written by Marc Le Douarain.

He describes the beginning of the project on his website:

> I decided to program a ladder language only for test purposes at the start, in February 2001. It was planned, that I would have to participate to a new product after leaving the enterprise in which I was working at that time. And I was thinking that to have a ladder language in those products could be a nice option to considerate. And so I started to code the first lines for calculating a rung with minimal elements and displaying dynamically it under Gtk, to see if my first idea to realize all this works.
>
> And as quickly I've found that it advanced quite well, I've continued with more complex elements: timer, multiples rungs, etc…
>
> Voila, here is this work… and more: I've continued to add features since then.
>
> — Marc Le Douarain *from "Genesis" at the Classic Ladder website*

Classic Ladder has been adapted to work with LinuxCNC's HAL, and is currently being distributed along with LinuxCNC. If there are issues/problems/bugs please report them to the Enhanced Machine Controller project.

26.2 Introduction

Ladder logic or the Ladder programming language is a method of drawing electrical logic schematics. It is now a graphical language very popular for programming Programmable Logic Controllers (PLCs). It was originally invented to describe logic made from relays. The name is based on the observation that programs in this language resemble ladders, with two vertical *rails* and a series of horizontal *rungs* between them. In Germany and elsewhere in Europe, the style is to draw the rails horizontally along the top and bottom of the page while the rungs are drawn vertically from left to right.

A program in ladder logic, also called a ladder diagram, is similar to a schematic for a set of relay circuits. Ladder logic is useful because a wide variety of engineers and technicians can understand and use it without much additional training because of the resemblance.

Ladder logic is widely used to program PLCs, where sequential control of a process or manufacturing operation is required. Ladder logic is useful for simple but critical control systems, or for reworking old hardwired relay circuits. As programmable logic controllers became more sophisticated it has also been used in very complex automation systems.

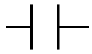
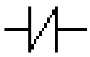
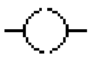
Ladder logic can be thought of as a rule-based language, rather than a procedural language. A *rung* in the ladder represents a rule. When implemented with relays and other electromechanical devices, the various rules *execute* simultaneously and immediately. When implemented in a programmable logic controller, the rules are typically executed sequentially by software, in a loop. By executing the loop fast enough, typically many times per second, the effect of simultaneous and immediate execution is obtained.

Ladder logic follows these general steps for operation.

- Read Inputs
- Solve Logic
- Update Outputs

26.3 Example

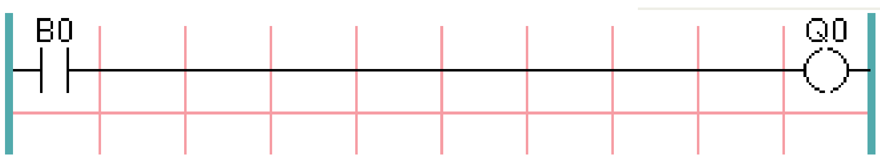
The most common components of ladder are contacts (inputs), these usually are either NC (normally closed) or NO (normally open), and coils (outputs).

- the NO contact 
- the NC contact 
- the coil (output) 

Of course there are many more components to a full ladder language, but understanding these will help you grasp the overall concept.

The ladder consists of one or more rungs. These rungs are horizontal traces (representing wires), with components on them (inputs, outputs and other), which get evaluated left to right.

This example is the simplest rung:



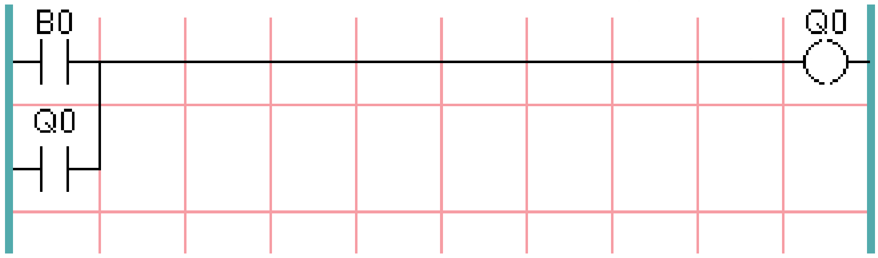
The input on the left, B0, a normally open contact, is connected to the coil (output) on the right, Q0. Now imagine a voltage gets applied to the leftmost end, because the input B0 turns true (e.g. the input is activated, or the user pushed the NO contact). The voltage has a direct path to reach the coil (output) on the right, Q0. As a consequence, the Q0 coil (output) will turn from 0/off/false to 1/on/true. If the user releases B0, the Q0 output quickly returns to 0/off/false.

26.4 Basic Latching On-Off Circuit

Building on the above example, suppose we add a switch that closes whenever the coil Q0 is active. This would be the case in a relay, where the coil can activate the switch contacts; or in a contactor, where there are often several small auxilliary contacts in addition to the large 3-phase contacts that are the primary feature of the contactor.

Since this auxilliary switch is driven from coil Q0 in our earlier example, we will give it the same number as the coil that drives it. This is the standard practice followed in all ladder programming, although it may seem strange at first to see a switch labeled the same as a coil. So let's call this auxilliary contact Q0 and connect it across the B0 *pushbutton* contact from our earlier example.

Let's take a look at it:

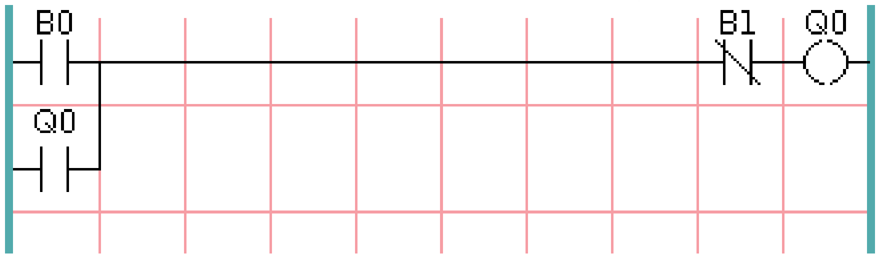


As before, when the user presses pushbutton B0, coil Q0 comes on. And when coil Q0 comes on, switch Q0 comes on. Now the interesting part happens. When the user releases pushbutton B0, coil Q0 does not stop as it did before. This is because switch Q0 of this circuit is effectively holding the user's pushbutton pressed. So we see that switch Q0 is still holding coil Q0 on after the *start* pushbutton has been released.

This type of contact on a coil or relay, used in this way, is often called a *holding contact*, because it *holds on* the coil that it is associated with. It is also occasionally called a *seal* contact, and when it is active it is said that the circuit is *sealed*.

Unfortunately, our circuit so far has little practical use, because, although we have an *on* or *start* button in the form of pushbutton B0, we have no way to shut this circuit off once it is started. But that's easy to fix. All we need is a way to interrupt the power to coil Q0. So let's add a normally-closed (NC) pushbutton just ahead of coil Q0.

Here's how that would look:



Now we have added *off* or *stop* pushbutton B1. If the user pushes it, contact from the rung to the coil is broken. When coil Q0 loses power, it drops to 0/off/false. When coil Q0 goes off, so does switch Q0, so the *holding contact* is broken, or the circuit is *unsealed*. When the user releases the *stop* pushbutton, contact is restored from the rung to coil Q0, but the rung has gone dead, so the coil doesn't come back on.

This circuit has been used for decades on virtually every machine that has a three-phase motor controlled by a contactor, so it was inevitable that it would be adopted by ladder/PLC programmers. It is also a very safe circuit, in that if *start* and *stop* are both pressed at the same time, the *stop* function always wins.

This is the basic building block of much of ladder programming, so if you are new to it, you would do well to make sure that you understand how this circuit operates.

Chapter 27

Classicladder Programming

27.1 Ladder Concepts

Classic Ladder is a type of programming language originally implemented on industrial PLCs (it's called Ladder Programming). It is based on the concept of relay contacts and coils, and can be used to construct logic checks and functions in a manner that is familiar to many systems integrators. Ladder consists of rungs that may have branches and resembles an electrical circuit. It is important to know how ladder programs are evaluated when running.

It seems natural that each line would be evaluated left to right, then the next line down, etc., but it doesn't work this way in ladder logic. Ladder logic *scans* the ladder rungs 3 times to change the state of the outputs.

- the inputs are read and updated
- the logic is figured out
- the outputs are set

This can be confusing at first if the output of one line is read by the input of a another rung. There will be one scan before the second input becomes true after the output is set.

Another gotcha with ladder programming is the "Last One Wins" rule. If you have the same output in different locations of your ladder the state of the last one will be what the output is set to.

27.2 Languages

The most common language used when working with Classic Ladder is *ladder*. Classic Ladder also supports Sequential Function Chart (Grafcet).

27.3 Components

There are 2 components to Classic Ladder.

- The real time module classicladder_rt
- The user space module (including a GUI) classicladder

27.3.1 Files

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

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

27.3.2 Realtime Module

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

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

The speed of the thread that Classic Ladder is running in directly affects the responsiveness to inputs and outputs. If you can turn a switch on and off faster than Classic Ladder can notice it then you may need to speed up the thread. The fastest that Classic Ladder can update the rungs is one millisecond. You can put it in a faster thread but it will not update any faster. If you put it in a slower than one millisecond thread then Classic Ladder will update the rungs slower. The current scan time will be displayed on the section display, it is rounded to microseconds. If the scan time is longer than one millisecond you may want to shorten the ladder or put it in a slower thread.

27.3.3 Variables

It is possible to configure the number of each type of ladder object while loading the Classic Ladder real time module. If you do not configure the number of ladder objects Classic Ladder will use the default values.

Table 27.1: Default Variable Count

Object Name	Variable Name	Default Value
Number of rungs	(numRungs)	100
Number of bits	(numBits)	20
Number of word variables	(numWords)	20
Number of timers	(numTimers)	10
Number of timers IEC	(numTimersIec)	10
Number of monostables	(numMonostables)	10
Number of counters	(numCounters)	10
Number of HAL inputs bit pins	(numPhysInputs)	15
Number of HAL output bit pins	(numPhysOutputs)	15
Number of arithmetic expressions	(numArithmExpr)	50
Number of Sections	(numSections)	10
Number of Symbols	(numSymbols)	Auto
Number of S32 inputs	(numS32in)	10
Number of S32 outputs	(numS32out)	10
Number of Float inputs	(numFloatIn)	10
Number of Float outputs	(numFloatOut)	10

Objects of most interest are numPhysInputs, numPhysOutputs, numS32in, and numS32out.

Changing these numbers will change the number of HAL bit pins available. numPhysInputs and numPhysOutputs control how many HAL bit (on/off) pins are available. numS32in and numS32out control how many HAL signed integers (+- integer range) pins are available.

For example (you don't need all of these to change just a few):

```
loadrt classicladder_rt numRungs=12 numBits=100 numWords=10
numTimers=10 numMonostables=10 numCounters=10 numPhysInputs=10
numPhysOutputs=10 numArithmExpr=100 numSections=4 numSymbols=200
numS32in=5 numS32out=5
```

To load the default number of objects:

```
loadrt classicladder_rt
```

27.4 Loading the Classic Ladder user module

Classic Ladder HAL commands must executed before the GUI loads or the menu item Ladder Editor will not function. If you used the Stepper Config Wizard place any Classic Ladder HAL commands in the custom.hal file.

To load the user module:

```
loadusr classicladder
```

To load a ladder file:

```
loadusr classicladder myladder.clp
```

Classic Ladder Loading Options

- *--nogui* - (loads without the ladder editor) normally used after debugging is finished.
- *--modbus_port=port* - (loads the modbus port number)
- *--modmaster* - (initializes MODBUS master) should load the ladder program at the same time or the TCP is default port.
- *--modslave* - (initializes MODBUS slave) only TCP

To use Classic Ladder with HAL without EMC:

```
loadusr -w classicladder
```

The -w tells HAL not to close down the HAL environment until Classic Ladder is finished.

If you first load ladder program with the *--nogui* option then load Classic Ladder again with no options the GUI will display the last loaded ladder program.

In AXIS you can load the GUI from File/Ladder Editor…

27.5 Classic Ladder GUI

If you load Classic Ladder with the GUI it will display two windows: section display, and section manager.

27.5.1 Sections Manager

When you first start up Classic Ladder you get an empty Sections Manager window.

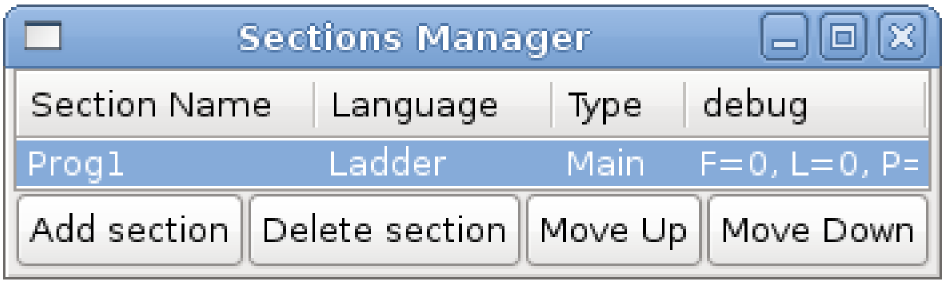


Figure 27.1: Sections Manager Default Window

This window allows you to name, create or delete sections and choose what language that section uses. This is also how you name a subroutine for call coils.

27.5.2 Section Display

When you first start up Classic Ladder you get an empty Section Display window.

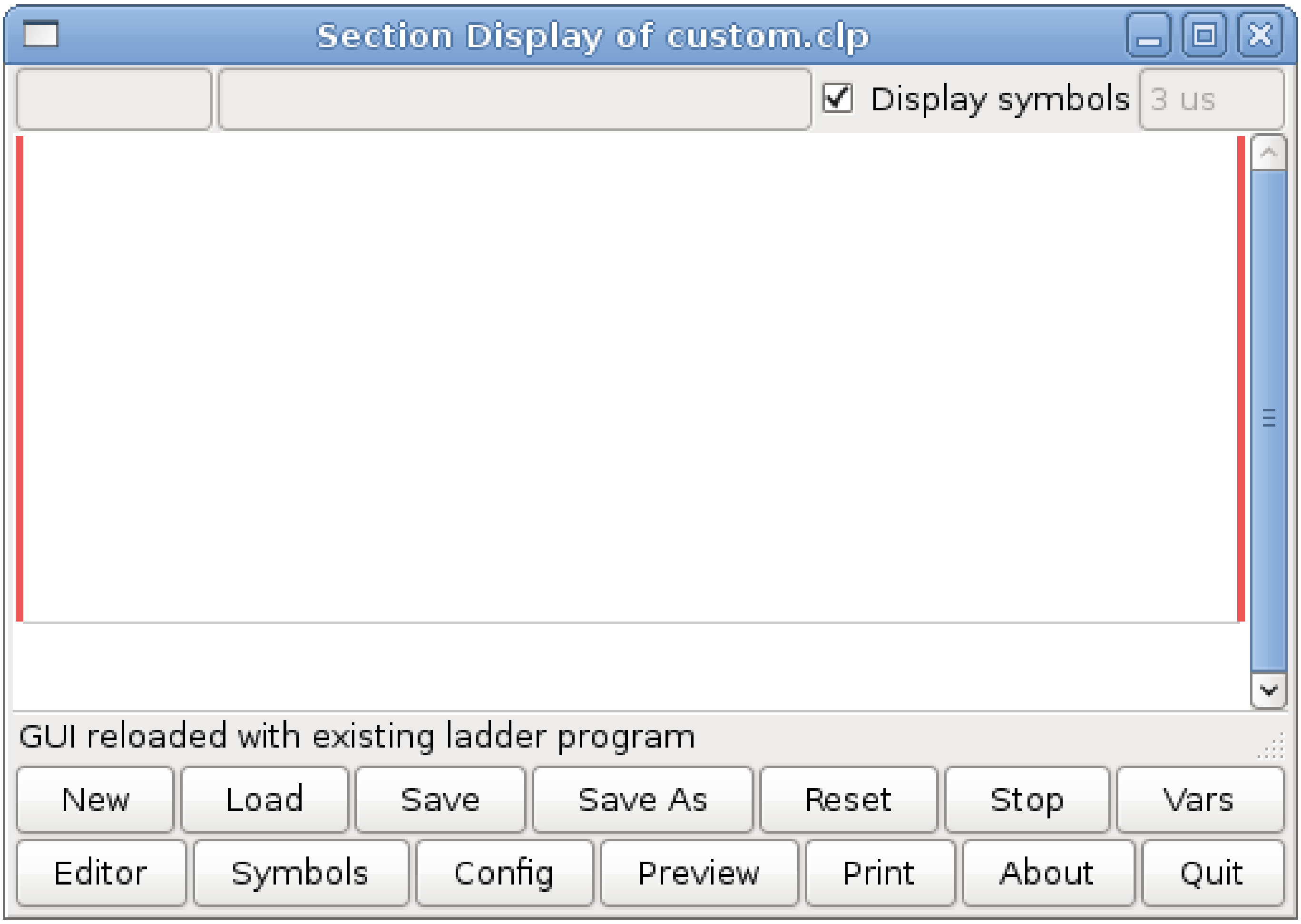


Figure 27.2: Section Display Default Window

Most of the buttons are self explanatory:

The Vars button is for looking at variables, toggle it to display one, the other, both, then none of the windows.

The Config button is used for modbus and shows the max number of ladder elements that was loaded with the real time module.

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

The Quit button will shut down the user program meaning Modbus and the display. The real time ladder program will still run in the background.

The check box at the top right allows you to select whether variable names or symbol names are displayed

You might notice that there is a line under the ladder program display that reads "Project failed to load…" That is the status bar that gives you info about elements of the ladder program that you click on in the display window. This status line will now display HAL signal names for variables %I, %Q and the first %W (in an equation) You might see some funny labels, such as (103) in the rungs. This is displayed (on purpose) because of an old bug- when erasing elements older versions sometimes didn't erase the object with the right code. You might have noticed that the long horizontal connection button sometimes didn't work in the older versions. This was because it looked for the *free* code but found something else. The number in the brackets is the unrecognized code. The ladder program will still work properly, to fix it erase the codes with the editor and save the program.

27.5.3 The Variable Windows

This are two variable windows: the Bit Status Window (boolean) and the Watch Window (signed integer). The Vars button is in the Section Display Window, toggle the Vars button to display one, the other, both, then none of the variable windows.

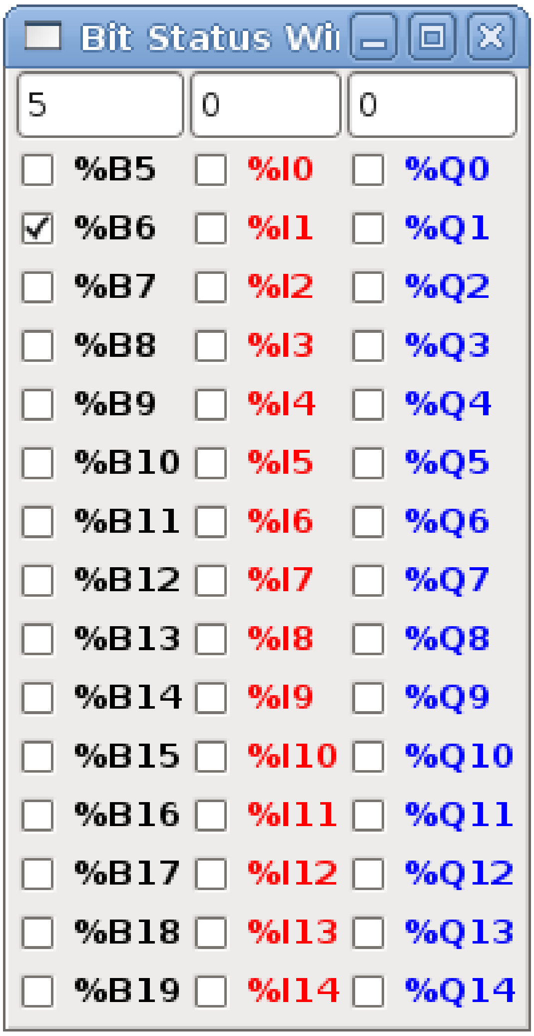


Figure 27.3: Bit Status Window

The Bit Status Window displays some of the boolean (on/off) variable data. Notice all variables start with the % sign. The %I variables represent HAL input bit pins. The %Q represents the relay coil and HAL output bit pins. The %B represents an internal relay coil or internal contact. The three edit areas at the top allow you to select what 15 variables will be displayed in each column. For instance, if the %B Variable column were 15 entries high, and you entered 5 at the top of the column, variables %B5 to %B19 would be displayed. The check boxes allow you to set and unset %B variables manually as long as the ladder program isn't setting them as outputs. Any Bits that are set as outputs by the program when Classic Ladder is running can not be changed and will be displayed as checked if on and unchecked if off.

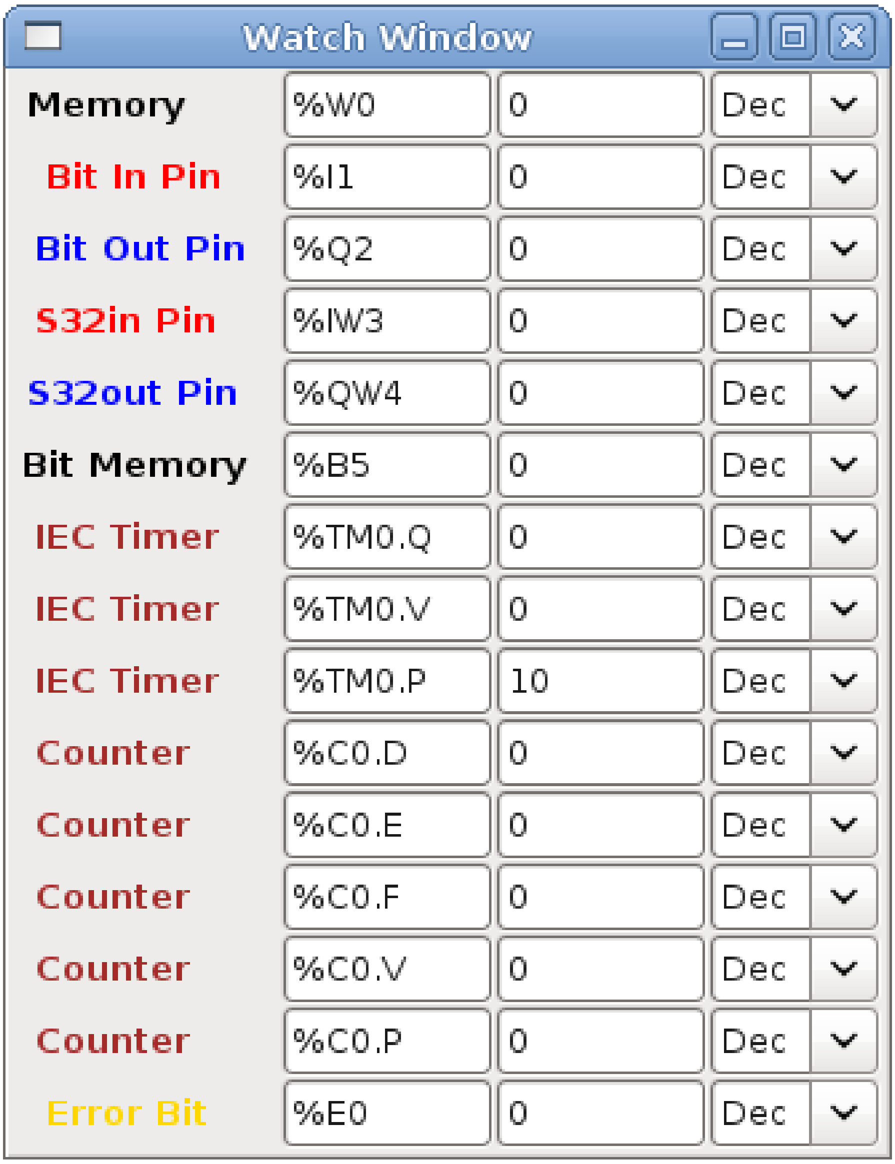


Figure 27.4: Watch Window

The Watch Window displays variable status. The edit box beside it is the number stored in the variable and the drop-down box beside that allow you to choose whether the number to be displayed in hex, decimal or binary. If there are symbol names defined in the symbols window for the word variables showing and the *display symbols* checkbox is checked in the section display window, symbol names will be displayed. To change the variable displayed, type the variable number, e.g. %W2 (if the display symbols check box is not checked) or type the symbol name (if the display symbols checkbox is checked) over an existing variable number/name and press the Enter Key.

27.5.4 Symbol Window

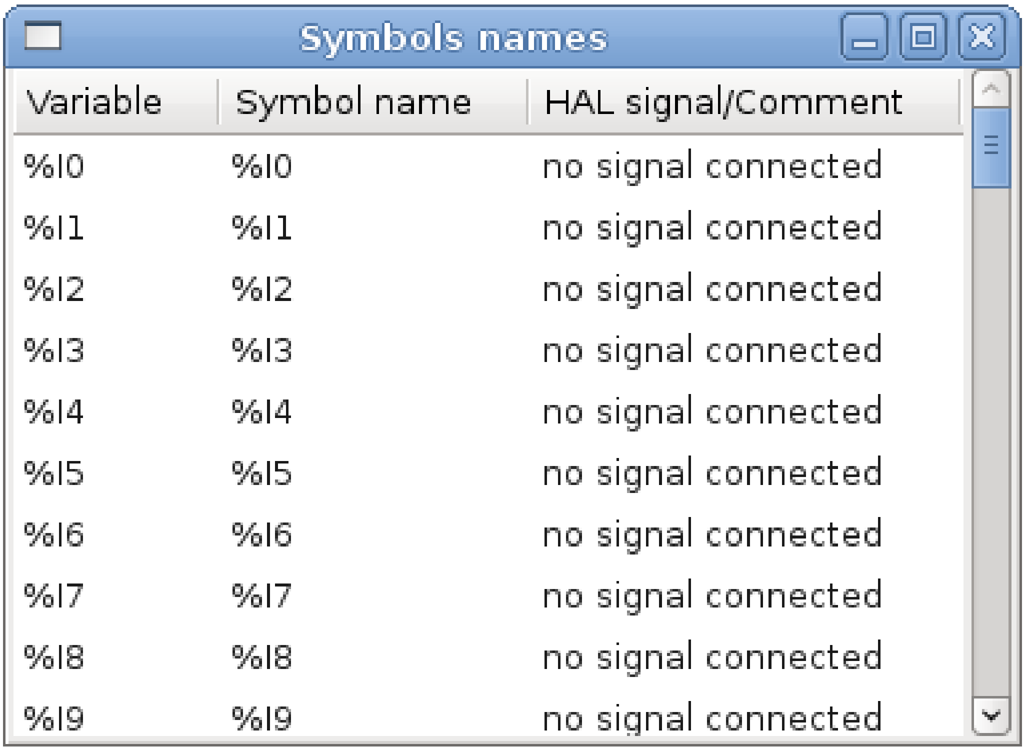


Figure 27.5: Symbol Names window

This is a list of *symbol* names to use instead of variable names to be displayed in the section window when the *display symbols* check box is checked. You add the variable name (remember the % symbol and capital letters), symbol name . If the variable can have a HAL signal connected to it (%I, %Q, and %W-if you have loaded s32 pin with the real time module) then the comment section will show the current HAL signal name or lack thereof. Symbol names should be kept short to display better. Keep in mind that you can display the longer HAL signal names of %I, %Q and %W variable by clicking on them in the section window. Between the two, one should be able to keep track of what the ladder program is connected to!

27.5.5 The Editor window

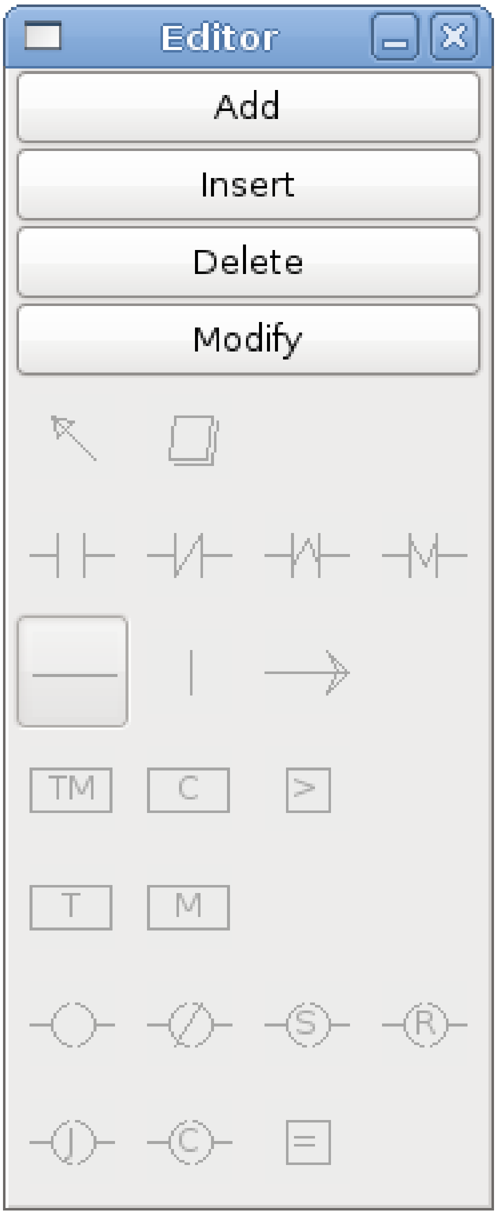


Figure 27.6: Editor Window

Starting from the top left image:

1. Object Selector, Eraser
2. N.O. Input, N.C. Input, Rising Edge Input , Falling Edge Input
3. Horizontal Connection, Vertical Connection , Long Horizontal Connection
4. Timer IEC Block, Counter Block, Compare Variable
5. Old Timer Block, Old Monostable Block (These have been replaced by the IEC Timer)
6. COILS - N.O. Output, N.C. Output, Set Output, Reset Output

7. Jump Coil, Call Coil, Variable Assignment

A short description of each of the buttons:

- *Selector* - allows you to select existing objects and modify the information.
- *Eraser* - erases an object.
- *N.O. Contact* - creates a normally open contact. It can be an external HAL-pin (%I) input contact, an internal-bit coil (%B) contact or a external coil (%Q) contact. The HAL-pin input contact is closed when the HAL-pin is true. The coil contacts are closed when the corresponding coil is active (%Q2 contact closes when %Q2 coil is active).
- *N.C. Contact* - creates a normally closed contact. It is the same as the N.O. contact except that the contact is open when the HAL-pin is true or the coil is active.
- 'Rising Edge Contact - creates a contact that is closed when the HAL-pin goes from False to true, or the coil from not-active to active.
- *Falling Edge Contact* - creates a contact that is closed when the HAL-pin goes from true to false or the coil from active to not.
- *Horizontal Connection* - creates a horizontal connection to objects.
- *Vertical Connection* - creates a vertical connection to horizontal lines.
- *Horizontal Running Connection* - creates a horizontal connection between two objects and is a quick way to connect objects that are more than one block apart.
- *IEC Timer* - creates a timer and replaces the *Timer*.
- *Timer* - creates a Timer Module (depreciated use IEC Timer instead).
- *Monostable* - creates a one-shot monostable module
- *Counter* - creates a counter module.
- *Compare* - creates a compare block to compare variable to values or other variables. (eg %W1<=5 or %W1=%W2) Compare cannot be placed in the right most side of the section display.
- *Variable Assignment* - creates an assignment block so you to assign values to variables. (eg %W2=7 or %W1=%W2) ASSIGNMENT functions can only be placed at the right most side of the section display.

27.5.6 Config Window

The config window shows the current project status and has the Modbus setup tabs.

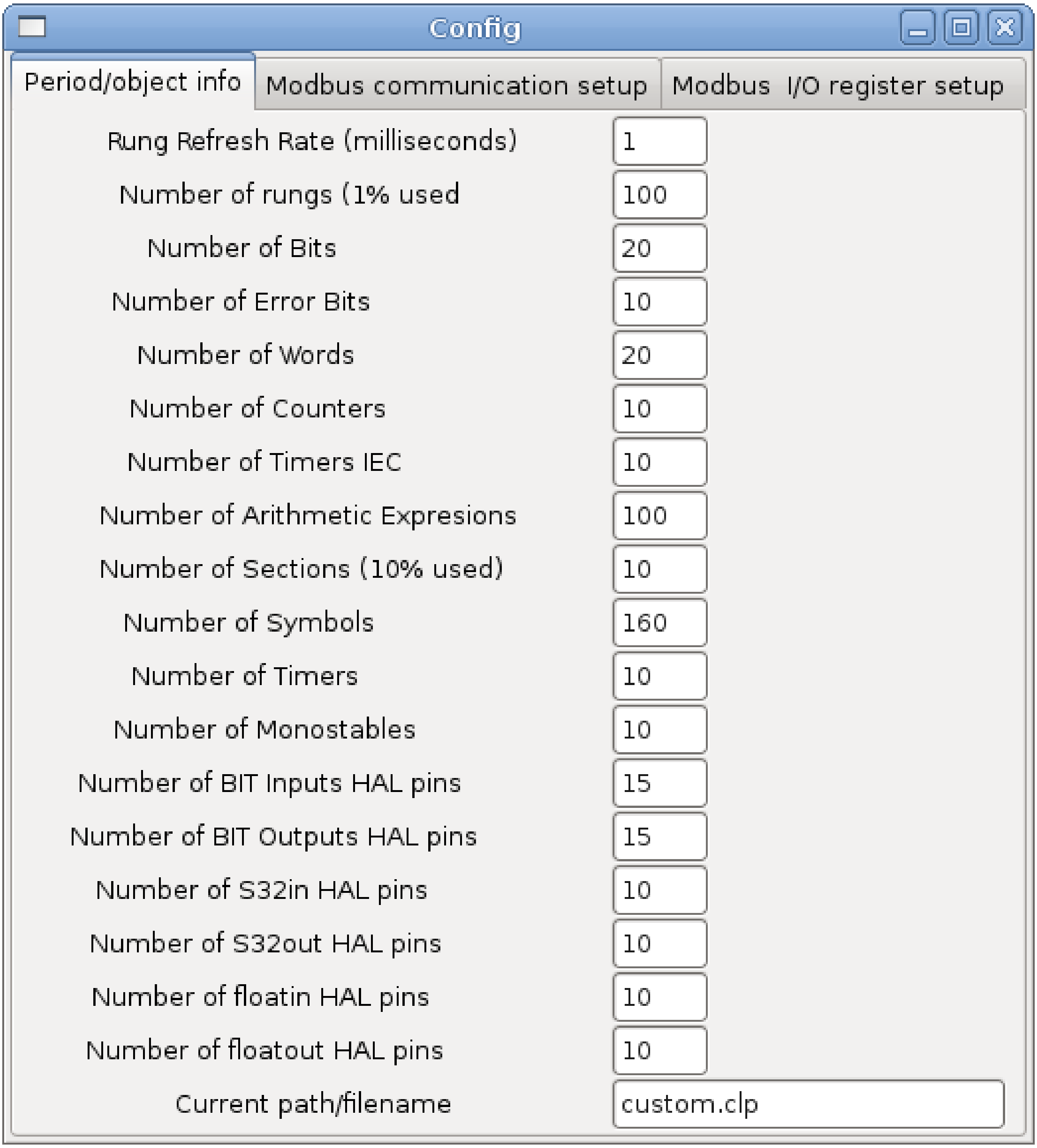


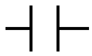
Figure 27.7: Config Window

27.6 Ladder objects

27.6.1 CONTACTS

Represent switches or relay contacts. They are controlled by the variable letter and number assigned to them.

The variable letter can be B, I, or Q and the number can be up to a three digit number eg. %I2, %Q3, or %B123. Variable I is controlled by a HAL input pin with a corresponding number. Variable B is for internal contacts, controlled by a B coil with a corresponding number. Variable Q is controlled by a Q coil with a corresponding number. (like a relay with multiple contacts). E.g. if HAL pin classicladder.0.in-00 is true then %I0 N.O. contact would be on (closed, true, whatever you like to call it). If %B7 coil is *energized* (on, true, etc) then %B7 N.O. contact would be on. If %Q1 coil is *energized* then %Q1 N.O. contact would be on (and HAL pin classicladder.0.out-01 would be true.)

- *N.O. Contact* -  (Normally Open) When the variable is false the switch is off.
- *N.C. Contact* -  (Normally Closed) When the variable is false the switch is on.
- *Rising Edge Contact* - When the variable changes from false to true, the switch is PULSED on.
- *Falling Edge Contact* - When the variable changes from true to false, the switch is PULSED on.

27.6.2 IEC TIMERS

Represent new count down timers. IEC Timers replace Timers and Monostables.

IEC Timers have 2 contacts.

- *I* - input contact
- *Q* - output contact

There are three modes - TON, TOF, TP.

- *TON* - When timer input is true countdown begins and continues as long as input remains true. After countdown is done and as long as timer input is still true the output will be true.
- *TOF* - When timer input is true, sets output true. When the input is false the timer counts down then sets output false.
- *TP* - When timer input is pulsed true or held true timer sets output true till timer counts down. (one-shot)

The time intervals can be set in multiples of 100ms, seconds, or minutes.

There are also Variables for IEC timers that can be read and/or written to in compare or operate blocks.

- *%TMxxx.Q* - timer done (Boolean, read write)
- *%TMxxx.P* - timer preset (read write)
- *%TMxxx.V* - timer value (read write)

27.6.3 TIMERS

Represent count down timers. This is deprecated and replaced by IEC Timers.

Timers have 4 contacts.

- *E* - enable (input) starts timer when true, resets when goes false
- *C* - control (input) must be on for the timer to run (usually connect to E)
- *D* - done (output) true when timer times out and as long as E remains true
- *R* - running (output) true when timer is running

The timer base can be multiples of milliseconds, seconds, or minutes.

There are also Variables for timers that can be read and/or written to in compare or operate blocks.

- *%Txx.R* - Timer xx running (Boolean, read only)
- *%Txx.D* - Timer xx done (Boolean, read only)
- *%Txx.V* - Timer xx current value (integer, read only)
- *%Txx.P* - Timer xx preset (integer, read or write)

27.6.4 MONOSTABLES

Represent the original one-shot timers. This is now deprecated and replaced by IEC Timers.

Monostables have 2 contacts, I and R.

- *I* - input (input) will start the mono timer running.
- *R* - running (output) will be true while timer is running.

The I contact is rising edge sensitive meaning it starts the timer only when changing from false to true (or off to on). While the timer is running the I contact can change with no effect to the running timer. R will be true and stay true till the timer finishes counting to zero. The timer base can be multiples of milliseconds, seconds, or minutes.

There are also Variables for monostables that can be read and/or written to in compare or operate blocks.

- *%Mxx.R* - Monostable xx running (Boolean, read only)
- *%Mxx.V* - Monostable xx current value (integer, read only)
- *%Mxx.P* - Monostable xx preset (integer, read or write)

27.6.5 COUNTERS

Represent up/down counters.

There are 7 contacts:

- *R* - reset (input) will reset the count to 0.
- *P* - preset (input) will set the count to the preset number assigned from the edit menu.
- *U* - up count (input) will add one to the count.
- *D* - down count (input) will subtract one from the count.

- *E* - under flow (output) will be true when the count rolls over from 0 to 9999.
- *D* - done (output) will be true when the count equals the preset.
- *F* - overflow (output) will be true when the count rolls over from 9999 to 0.

The up and down count contacts are edge sensitive meaning they only count when the contact changes from false to true (or off to on if you prefer).

The range is 0 to 9999.

There are also Variables for counters that can be read and/or written to in compare or operate blocks.

- *%Cxx.D* - Counter xx done (Boolean, read only)
- *%Cxx.E* - Counter xx empty overflow (Boolean, read only)
- *%Cxx.F* - Counter xx full overflow (Boolean, read only)
- *%Cxx.V* - Counter xx current value (integer, read or write)
- *%Cxx.P* - Counter xx preset (integer, read or write)

27.6.6 COMPARE

For arithmetic comparison. Is variable %XXX = to this number (or evaluated number)

The compare block will be true when comparison is true. you can use most math symbols:

- +, - ,* , /, = (standard math symbols)
- < (less than), > (greater than), <= (less or equal), >= (greater or equal), <> (not equal)
- (,) grouping
- ˆ (exponent),% (modulus),& (and),| (or),. -
- ABS (absolute), MOY (french for average) ,AVG (average)

For example ABS(%W2)=1, MOY(%W1,%W2)<3.

No spaces are allowed in the comparison equation. For example %C0.V>%C0.P is a valid comparison expression while %C0.V > %CO.P is not a valid expression.

There is a list of Variables down the page that can be used for reading from and writing to ladder objects. When a new compare block is opened be sure and delete the # symbol when you enter a compare.

To find out if word variable #1 is less than 2 times the current value of counter #0 the syntax would be:

```
%W1<2*%C0.V
```

To find out if S32in bit 2 is equal to 10 the syntax would be:

```
%IW2=10
```

Note: Compare uses the arithmetic equals not the double equals that programmers are used to.

27.6.7 VARIABLE ASSIGNMENT

For variable assignment, e.g. assign this number (or evaluated number) to this variable %xxx, there are two math functions MINI and MAXI that check a variable for maximum (0x80000000) and minimum values (0x07FFFFFFF) (think signed values) and keeps them from going beyond.

When a new variable assignment block is opened be sure to delete the # symbol when you enter an assignment.

To assign a value of 10 to the timer preset of IEC Timer 0 the syntax would be:

```
%TM0.P=10
```

To assign the value of 12 to s32out bit 3 the syntax would be:

```
%QW3=12
```

The following figure shows an Assignment and a Comparison Example. %QW0 is a S32out bit and %IW0 is a S32in bit. In this case the HAL pin classicladder.0.s32out-00 will be set to a value of 5 and when the HAL pin classicladder.0.s32in-00 is 0 the HAL pin classicladder.0.out-00 will be set to True.

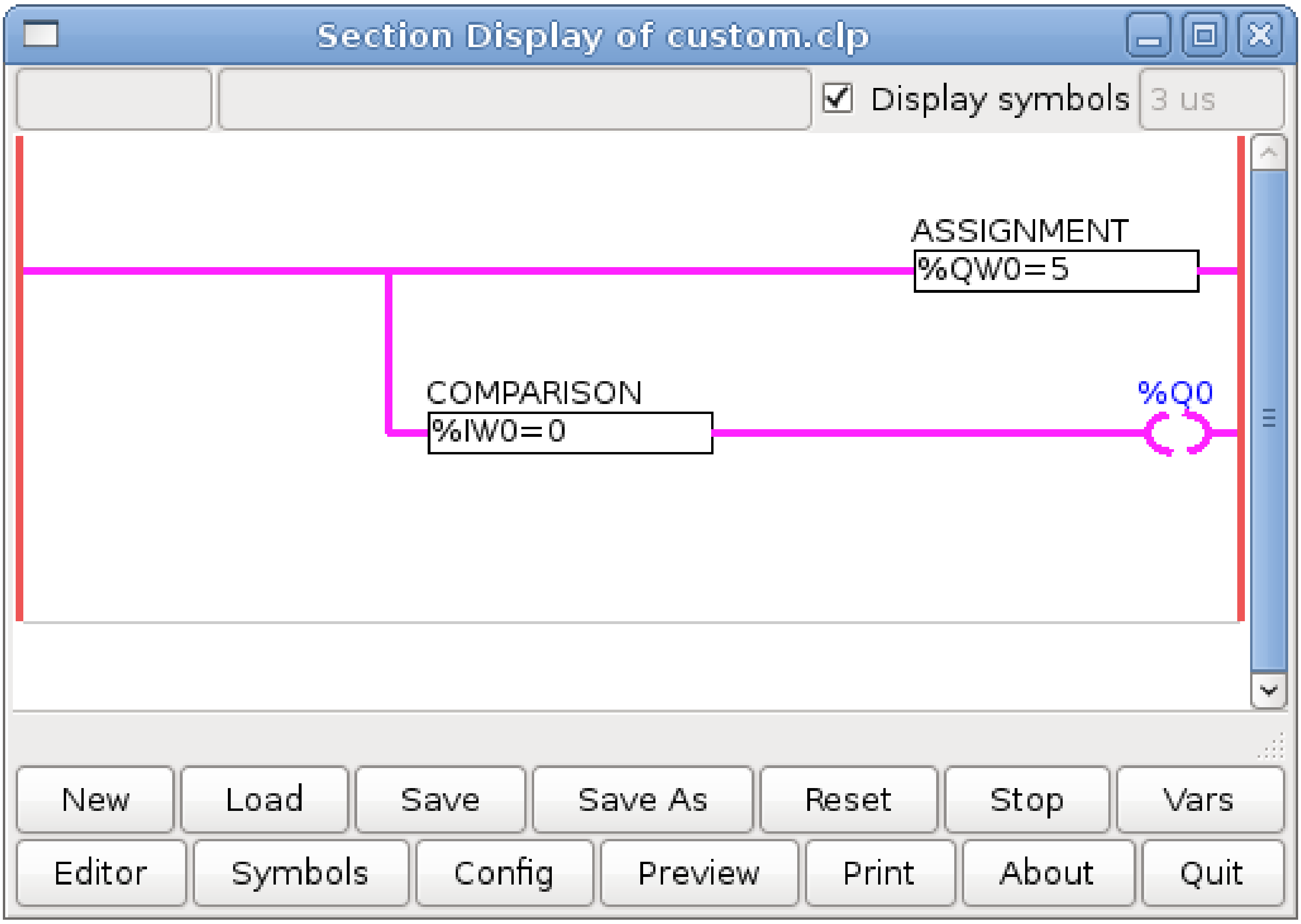
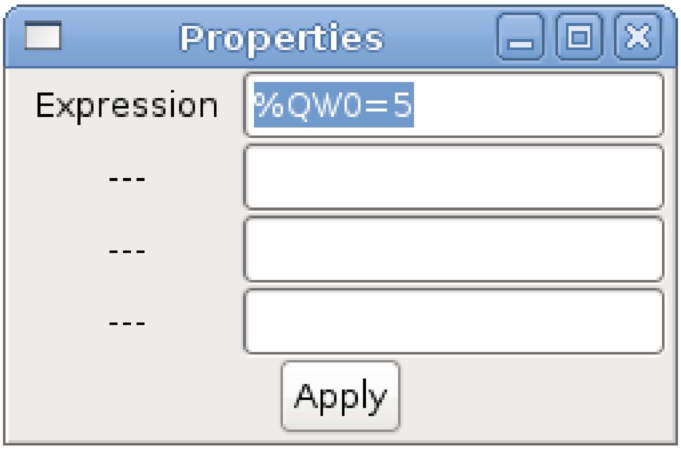
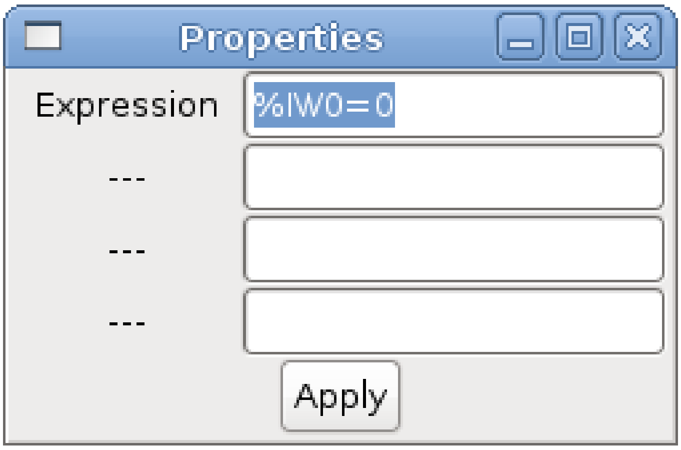


Figure 27.8: Assign/Compare Example





27.6.8 COILS

Coils represent relay coils. They are controlled by the variable letter and number assigned to them.

The variable letter can be B or Q and the number can be up to a three digit number eg. %Q3, or %B123. Q coils control HAL out pins, e.g. if %Q15 is energized then HAL pin classicladder.0.out-15 will be true. B coils are internal coils used to control program flow.

- *N.O. COIL* - (a relay coil.) When coil is energized it's N.O. contact will be closed (on, true, etc)
- *N.C. COIL* - (a relay coil that inverses its contacts.) When coil is energized it"s N.O. contact will be open (off, false, etc)
- *SET COIL* - (a relay coil with latching contacts) When coil is energized it's N.O. contact will be latched closed.
- *RESET COIL* - (a relay coil with latching contacts) When coil is energized It's N.0. contact will be latched open.
- *JUMP COIL* - (a *goto* coil) when coil is energized ladder program jumps to a rung (in the CURRENT section) -jump points are designated by a rung label. (Add rung labels in the section display, top left label box)
- *CALL COIL* - (a *gosub* coil) when coil is energized program jumps to a subroutine section designated by a subroutine number -subroutines are designated SR0 to SR9 (designate them in the section manager)

Warning
If you use a N.C. contact with a N.C. coil the logic will work (when the coil is energized the contact will be closed) but that is really hard to follow!

27.6.8.1 JUMP COIL

A JUMP COIL is used to *JUMP* to another section, like a goto in BASIC programming language.

If you look at the top left of the sections display window you will see a small label box and a longer comment box beside it. Now go to Editor→Modify then go back to the little box, type in a name.

Go ahead and add a comment in the comment section. This label name is the name of this rung only and is used by the JUMP COIL to identify where to go.

When placing a JUMP COIL, add it in the rightmost position and change the label to the rung you want to JUMP to.

27.6.8.2 CALL COIL

A CALL COIL is used to go to a subroutine section then return, like a gosub in BASIC programming language.

If you go to the sections manager window hit the add section button. You can name this section, select what language it will use (ladder or sequential), and select what type (main or subroutine).

Select a subroutine number (SR0 for example). An empty section will be displayed and you can build your subroutine.

When you've done that, go back to the section manager and click on the your main section (default name prog1).

Now you can add a CALL COIL to your program. CALL COILs are to be placed at the rightmost position in the rung.

Remember to change the label to the subroutine number you chose before.

27.7 Classic Ladder Variables

These Variables are used in COMPARE or OPERATE to get information about, or change specs of, ladder objects such as changing a counter preset, or seeing if a timer is done running.

List of variables :

- *%Bxxx* - Bit memory xxx (Boolean)
- *%Wxxx* - Word memory xxx (32 bits signed integer)
- *%IWxxx* - Word memory xxx (S32 in pin)
- *%QWxxx* - Word memory xxx (S32 out pin)
- *%IFxx* - Word memory xx (Float in pin) (**converted to S32 in Classic Ladder**)
- *%QFxx* - Word memory xx (Float out pin) (**converted to S32 in Classic Ladder**)
- *%Txx.R* - Timer xx running (Boolean, user read only)
- *%Txx.D* - Timer xx done (Boolean, user read only)
- *%Txx.V* - Timer xx current value (integer, user read only)
- *%Txx.P* - Timer xx preset (integer)
- *%TMxxx.Q* - Timer xxx done (Boolean, read write)
- *%TMxxx.P* - Timer xxx preset (integer, read write)
- *%TMxxx.V* - Timer xxx value (integer, read write)
- *%Mxx.R* - Monostable xx running (Boolean)
- *%Mxx.V* - Monostable xx current value (integer, user read only)

- *%Mxx.P* - Monostable xx preset (integer)
- *%Cxx.D* - Counter xx done (Boolean, user read only)
- *%Cxx.E* - Counter xx empty overflow (Boolean, user read only)
- *%Cxx.F* - Counter xx full overflow (Boolean, user read only)
- *%Cxx.V* - Counter xx current value (integer)
- *%Cxx.P* - Counter xx preset (integer)
- *%Ixxx* - Physical input xxx (Boolean) (HAL input bit)
- *%Qxxx* - Physical output xxx (Boolean) (HAL output bit)
- *%Xxxx* - Activity of step xxx (sequential language)
- *%Xxxx.V* - Time of activity in seconds of step xxx (sequential language)
- *%Exx* - Errors (Boolean, read write(will be overwritten))
- *Indexed or vectored variables* - These are variables indexed by another variable. Some might call this vectored variables. Example: %W0[%W4] => if %W4 equals 23 it corresponds to %W23

27.8 GRAFCET Programming

> **Warning**
> This is probably the least used and most poorly understood feature of Classic Ladder. Sequential programming is used to make sure a series of ladder events always happen in a prescribed order. Sequential programs do not work alone. There is always a ladder program as well that controls the variables. Here are the basic rules governing sequential programs:

- Rule 1 : Initial situation - The initial situation is characterized by the initial steps which are by definition in the active state at the beginning of the operation.There shall be at least one initial step.
- Rule 2 : R2, Clearing of a transition - A transition is either enabled or disabled. It is said to be enabled when all immediately preceding steps linked to its corresponding transition symbol are active, otherwise it is disabled. A transition cannot be cleared unless it is enabled, and its associated transition condition is true.
- Rule 3 : R3, Evolution of active steps - The clearing of a transition simultaneously leads to the active state of the immediately following step(s) and to the inactive state of the immediately preceding step(s).
- Rule 4 : R4, Simultaneous clearing of transitions - All simultaneous cleared transitions are simultaneously cleared.
- Rule 5 : R5, Simultaneous activation and deactivation of a step - If during operation, a step is simultaneously activated and deactivated, priority is given to the activation.

This is the SEQUENTIAL editor window Starting from the top left image: Selector arrow , Eraser Ordinary step , Initial (Starting) step Transition , Step and Transition Transition Link-Downside , Transition Link-Upside Pass-through Link-Downside , Pass-through Link-Upside Jump Link Comment Box [show sequential program]

- *ORDINARY STEP* - has a unique number for each one
- *STARTING STEP* - a sequential program must have one. This is where the program will start.
- *TRANSITION* - This shows the variable that must be true for control to pass through to the next step.
- *STEP AND TRANSITION* - Combined for convenience

- *TRANSITION LINK-DOWNSIDE* - splits the logic flow to one of two possible lines based on which of the next steps is true first (Think OR logic)
- *TRANSITION LINK=UPSIDE* - combines two (OR) logic lines back in to one
- *PASS-THROUGH LINK-DOWNSIDE* - splits the logic flow to two lines that BOTH must be true to continue (Think AND logic)
- *PASS-THROUGH LINK-UPSIDE* - combines two concurrent (AND logic) logic lines back together
- *JUMP LINK* - connects steps that are not underneath each other such as connecting the last step to the first
- *COMMENT BOX* - used to add comments

To use links, you must have steps already placed. Select the type of link, then select the two steps or transactions one at a time. It takes practice!

With sequential programming: The variable %Xxxx (eg. %X5) is used to see if a step is active. The variable %Xxxx.V (eg. %X5.V) is used to see how long the step has been active. The %X and %X.v variables are use in LADDER logic. The variables assigned to the transitions (eg. %B) control whether the logic will pass to the next step. After a step has become active the transition variable that caused it to become active has no control of it anymore. The last step has to JUMP LINK back only to the beginning step.

27.9 Modbus

Things to consider:

- Modbus is a userspace program so it might have latency issues on a heavily laden computer.
- Modbus is not really suited to Hard real time events such as position control of motors or to control E-stop.
- The Classic Ladder GUI must be running for Modbus to be running.
- Modbus is not fully finished so it does not do all modbus functions.

To get MODBUS to initialize you must specify that when loading the Classic Ladder userspace program.

Loading Modbus

```
loadusr -w classicladder --modmaster myprogram.clp
```

The -w makes HAL wait until you close Classic Ladder before closing realtime session. Classic Ladder also loads a TCP modbus slave if you add *--modserver* on command line.

MODBUS FUNCTIONS

- *1* - read coils
- *2* - read inputs
- *3* - read holding registers
- *4* - read input registers
- *5* - write single coils
- *6* - write single register
- *8* - echo test
- *15* - write multiple coils

- *16* - write multiple registers

If you do not specify a -- *modmaster* when loading the Classic Ladder user program this page will not be displayed.

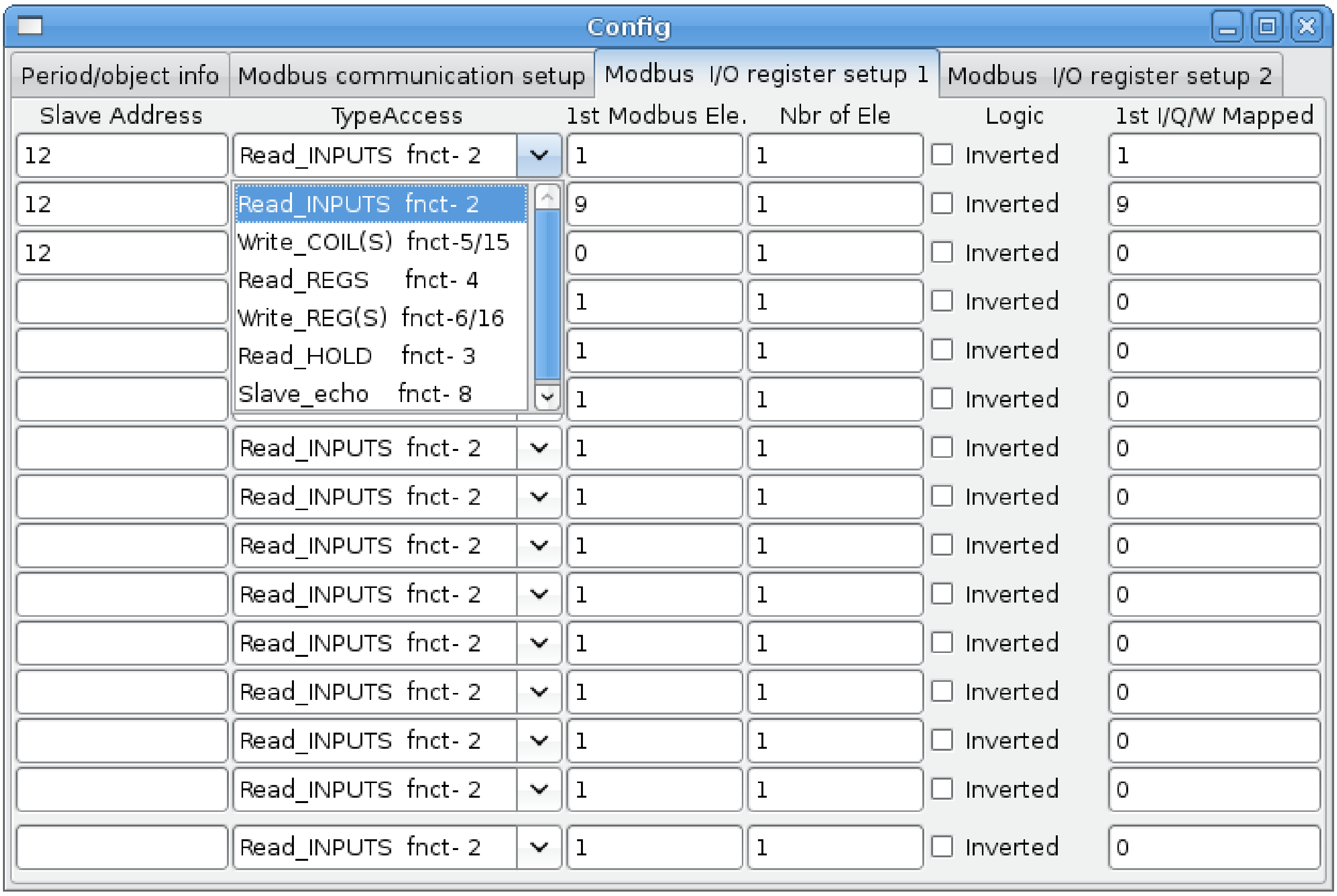


Figure 27.9: Config I/O

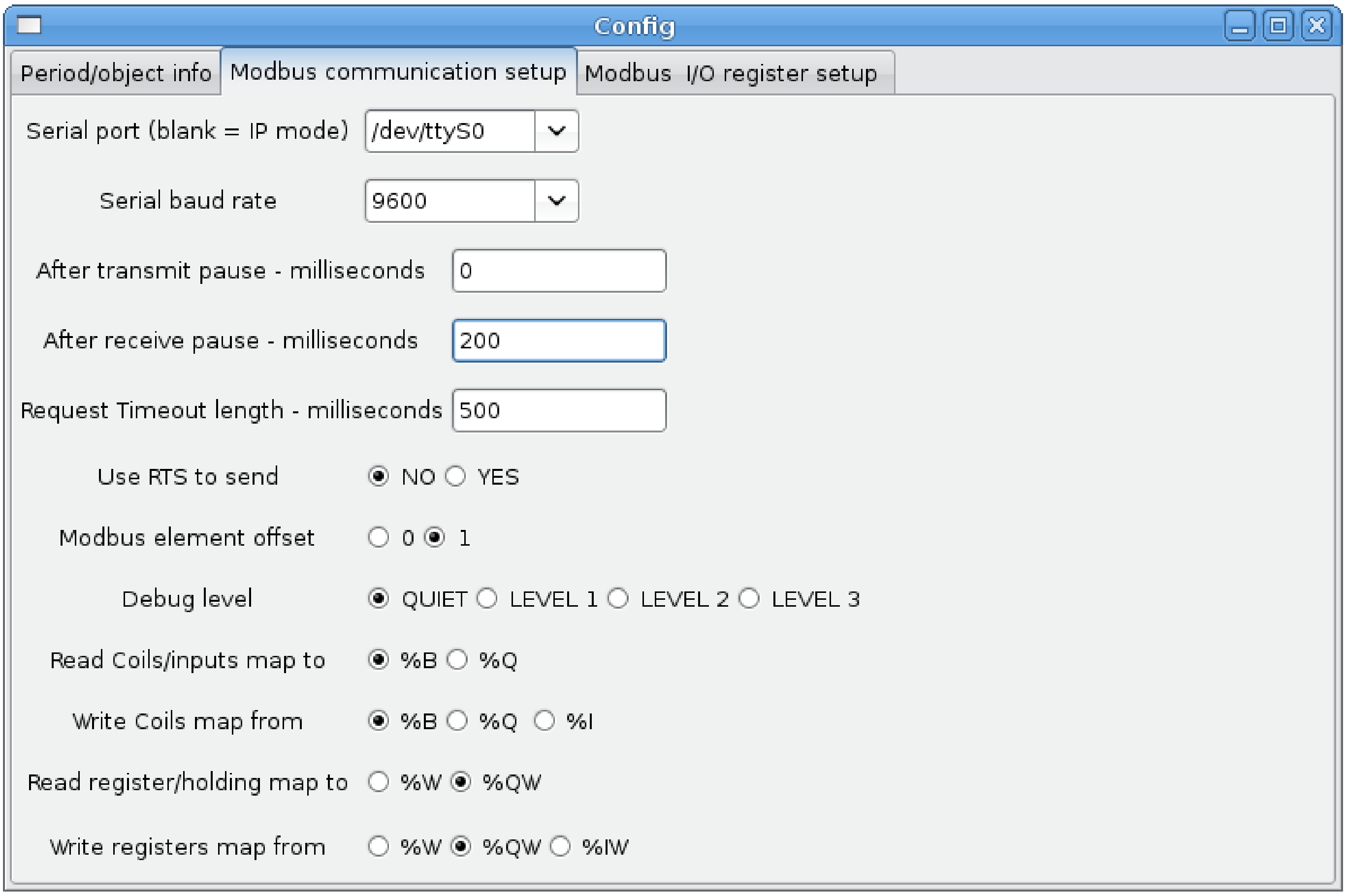


Figure 27.10: Config Coms

- *SERIAL PORT* - For IP blank. For serial the location/name of serial driver eg. /dev/ttyS0 (or /dev/ttyUSB0 for a USB-to-serial converter).
- *SERIAL SPEED* - Should be set to speed the slave is set for - 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 are supported.
- *PAUSE AFTER TRANSMIT* - Pause (milliseconds) after transmit and before receiving answer, some devices need more time (e.g., USB-to-serial converters).
- *PAUSE INTER-FRAME* - Pause (milliseconds) after receiving answer from slave. This sets the duty cycle of requests (it's a pause for EACH request).
- *REQUEST TIMEOUT LENGTH* - Length (milliseconds) of time before we decide that the slave didn't answer.
- *MODBUS ELEMENT OFFSET* - used to offset the element numbers by 1 (for manufacturers numbering differences).
- *DEBUG LEVEL* - Set this to 0-3 (0 to stop printing debug info besides no-response errors).
- *READ COILS/INPUTS MAP TO* - Select what variables that read coils/inputs will update. (B or Q).
- *WRITE COILS MAP TO* - Select what variables that write coils will updated.from (B,Q,or I).
- *READ REGISTERS/HOLDING* - Select what variables that read registers will update. (W or QW).
- *WRITE REGISTERS MAP TO* - Select what variables that read registers will updated from. (W, QW, or IW).
- *SLAVE ADDRESS* - For serial the slaves ID number usually settable on the slave device (usually 1-256) For IP the slave IP address plus optionally the port number.

- *TYPE ACCESS* - This selects the MODBUS function code to send to the slave (eg what type of request).
- *COILS / INPUTS* - Inputs and Coils (bits) are read from/written to I, B, or Q variables (user selects).
- *REGISTERS (WORDS)* - Registers (Words/Numbers) map to IW, W, or QW variables (user selects).
- *1st MODBUS ELEMENT* - The address (or register number) of the first element in a group. (remember to set MODBUS ELEMENT OFFSET properly).
- *NUMBER OF ELEMENTS* - The number of elements in this group.
- *LOGIC* - You can invert the logic here.
- *1st%I%Q IQ WQ MAPPED* - This is the starting number of %B, %I, %Q, %W, %IW, or %QW variables that are mapped onto/from the modbus element group (starting at the first modbus element number).

In the example above: Port number - for my computer /dev/ttyS0 was my serial port.

The serial speed is set to 9600 baud.

Slave address is set to 12 (on my VFD I can set this from 1-31, meaning I can talk to 31 VFDs maximum on one system).

The first line is set up for 8 input bits starting at the first register number (register 1). So register numbers 1-8 are mapped onto Classic Ladder's %B variables starting at %B1 and ending at %B8.

The second line is set for 2 output bits starting at the ninth register number (register 9) so register numbers 9-10 are mapped onto Classic Ladder's %Q variables starting at %Q9 ending at %Q10.

The third line is set to write 2 registers (16 bits each) starting at the 0th register number (register 0) so register numbers 0-1 are mapped onto Classic Ladder's %W variables starting at %W0 ending at %W1.

It's easy to make an off-by-one error as sometimes the modbus elements are referenced starting at one rather then 0 (actually by the standard that is the way it's supposed to be!) You can use the modbus element offset radio button to help with this.

The documents for your modbus slave device will tell you how the registers are set up- there is no standard way.

The SERIAL PORT, PORT SPEED, PAUSE, and DEBUG level are editable for changes (when you close the config window values are applied, though Radio buttons apply immediately).

To use the echo function select the echo function and add the slave number you wish to test. You don't need to specify any variables.

The number 257 will be sent to the slave number you specified and the slave should send it back. you will need to have Classic Ladder running in a terminal to see the message.

27.9.1 MODBUS Settings

Serial:

- Classic Ladder uses RTU protocol (not ASCII).
- 8 data bits, No parity is used, and 1 stop bit is also known as 8-N-1.
- Baud rate must be the same for slave and master. Classic Ladder can only have one baud rate so all the slaves must be set to the same rate.
- Pause inter frame is the time to pause after receiving an answer.
- MODBUS_TIME_AFTER_TRANSMIT is the length of pause after sending a request and before receiving an answer (this apparently helps with USB converters which are slow).

27.9.2 MODBUS Info

- Classic Ladder can use distributed inputs/outputs on modules using the modbus protocol ("master": polling slaves).
- The slaves and theirs I/O can be configured in the config window.
- 2 exclusive modes are available : ethernet using Modbus/TCP and serial using Modbus/RTU.
- No parity is used.
- If no port name for serial is set, TCP/IP mode will be used…
- The slave address is the slave address (Modbus/RTU) or the IP address.
- The IP address can be followed per the port number to use (xx.xx.xx.xx:pppp) else the port 9502 will be used per default.
- 2 products have been used for tests: a Modbus/TCP one (Adam-6051, http://www.advantech.com) and a serial Modbus/RTU one (http://www.ipac.ws).
- See examples: adam-6051 and modbus_rtu_serial.
- Web links: http://www.modbus.org and this interesting one: http://www.iatips.com/modbus.html
- MODBUS TCP SERVER INCLUDED
- Classic Ladder has a Modbus/TCP server integrated. Default port is 9502. (the previous standard 502 requires that the application must be launched with root privileges).
- List of Modbus functions code supported are: 1, 2, 3, 4, 5, 6, 15 and 16.
- Modbus bits and words correspondence table is actually not parametric and correspond directly to the %B and %W variables.

More information on modbus protocol is available on the internet.

http://www.modbus.org/

27.9.3 Communication Errors

If there is a communication error, a warning window will pop up (if the GUI is running) and %E0 will be true. Modbus will continue to try to communicate. The %E0 could be used to make a decision based on the error. A timer could be used to stop the machine if timed out, etc.

27.9.4 MODBUS Bugs

- In compare blocks the function %W=ABS(%W1-%W2) is accepted but does not compute properly. only %W0=ABS(%W1) is currently legal.
- When loading a ladder program it will load Modbus info but will not tell Classic Ladder to initialize Modbus. You must initialize Modbus when you first load the GUI by adding *--modmaster*.
- If the section manager is placed on top of the section display, across the scroll bar and exit is clicked the user program crashes.
- When using *--modmaster* you must load the ladder program at the same time or else only TCP will work.
- reading/writing multiple registers in Modbus has checksum errors.

27.10 Setting up Classic Ladder

In this section we will cover the steps needed to add Classic Ladder to a Stepconf Wizard generated config. On the advanced Configuration Options page of Stepconf Wizard check off "Include Classic Ladder PLC".

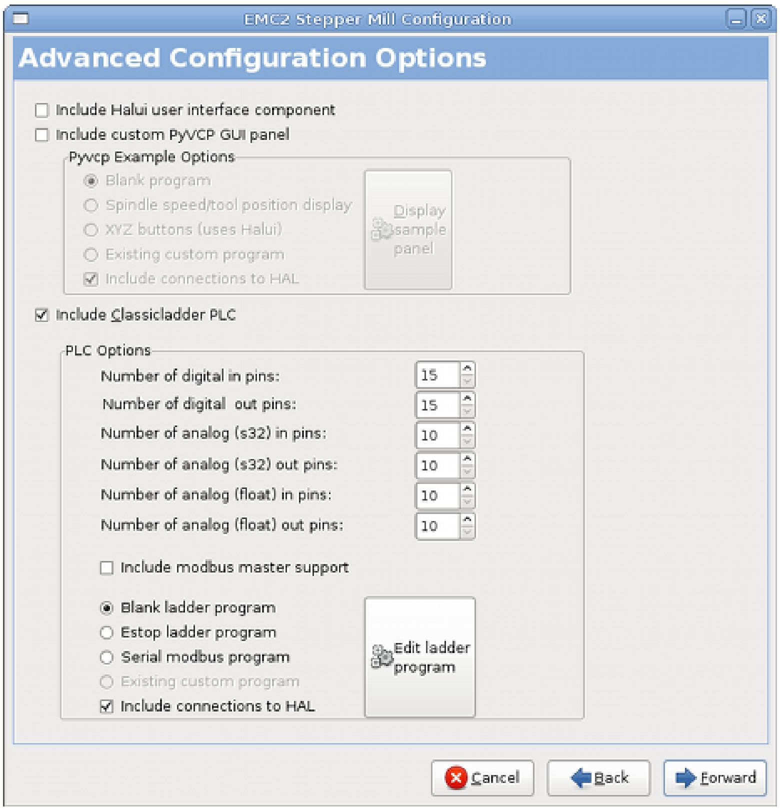


Figure 27.11: Stepconf Classic Ladder

27.10.1 Add the Modules

If you used the Stepconf Wizard to add Classic Ladder you can skip this step.

To manually add Classic Ladder you must first add the modules. This is done by adding a couple of lines to the custom.hal file.

This line loads the real time module:

```
loadrt classicladder_rt
```

This line adds the Classic Ladder function to the servo thread:

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

27.10.2 Adding Ladder Logic

Now start up your config and select "File/Ladder Editor" to open up the Classic Ladder GUI. You should see a blank Section Display and Sections Manager window as shown above. In the Section Display window open the Editor. In the Editor window select Modify. Now a Properties window pops up and the Section Display shows a grid. The grid is one rung of ladder. The rung can contain branches. A simple rung has one input, a connector line and one output.

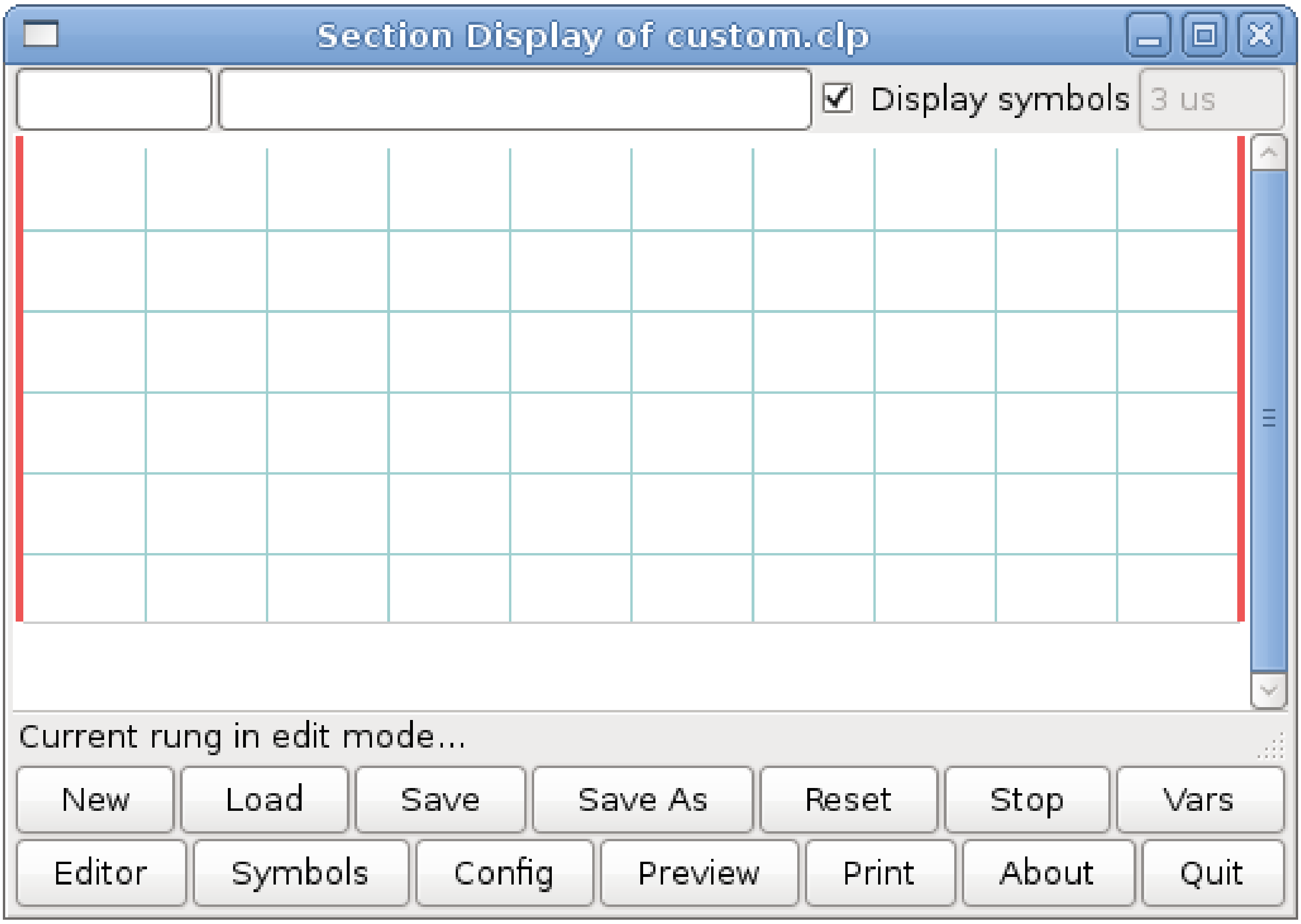


Figure 27.12: Section Display with Grid

Now click on the N.O. Input in the Editor Window.

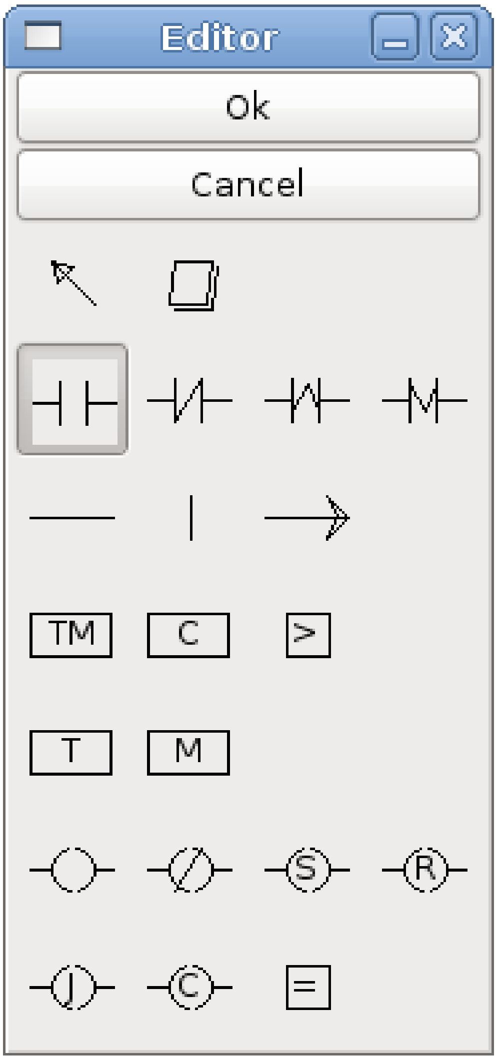


Figure 27.13: Editor Window

Now click in the upper left grid to place the N.O. Input into the ladder.

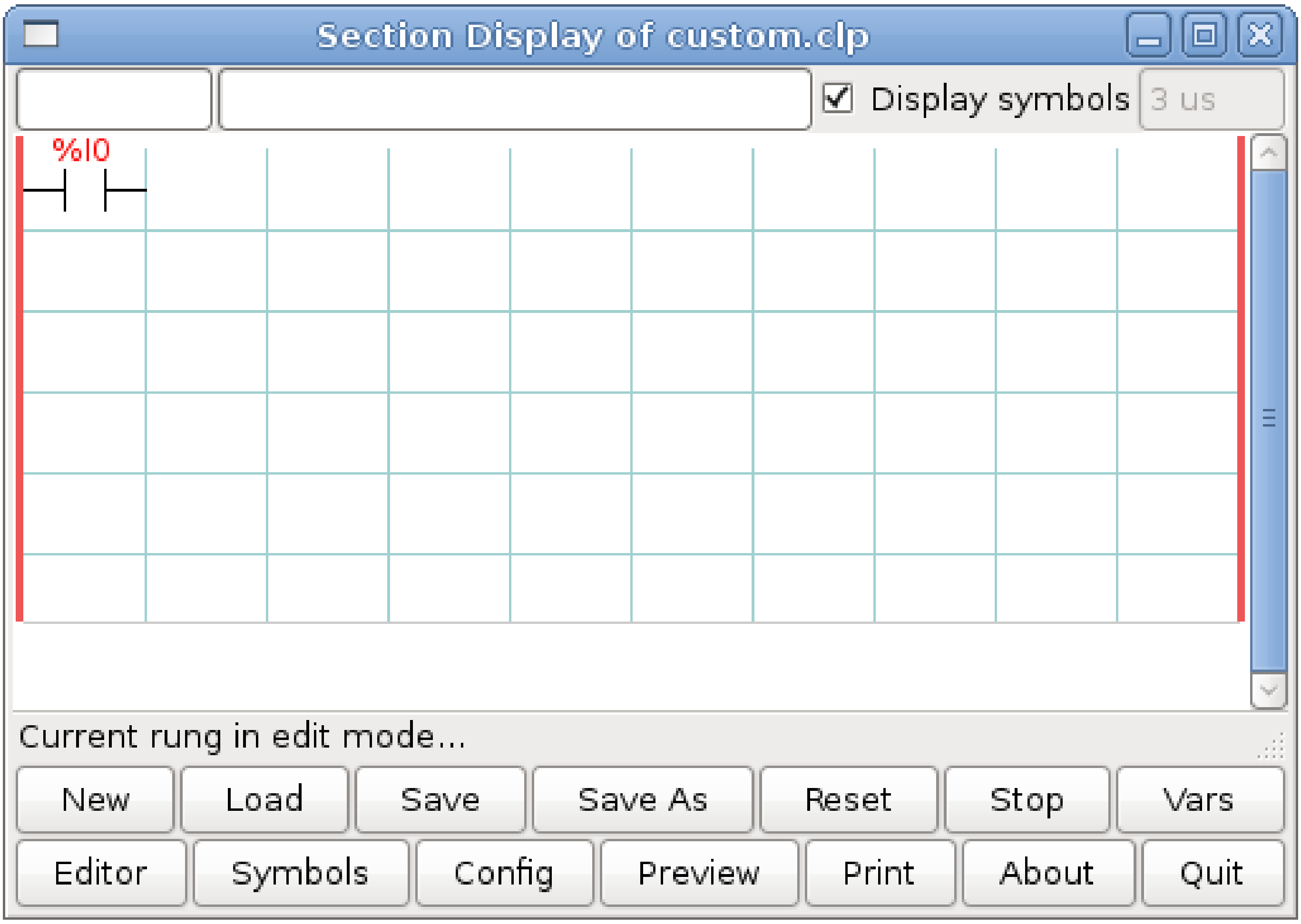


Figure 27.14: Section Display with Input

Repeat the above steps to add a N.O. Output to the upper right grid and use the Horizontal Connection to connect the two. It should look like the following. If not, use the Eraser to remove unwanted sections.

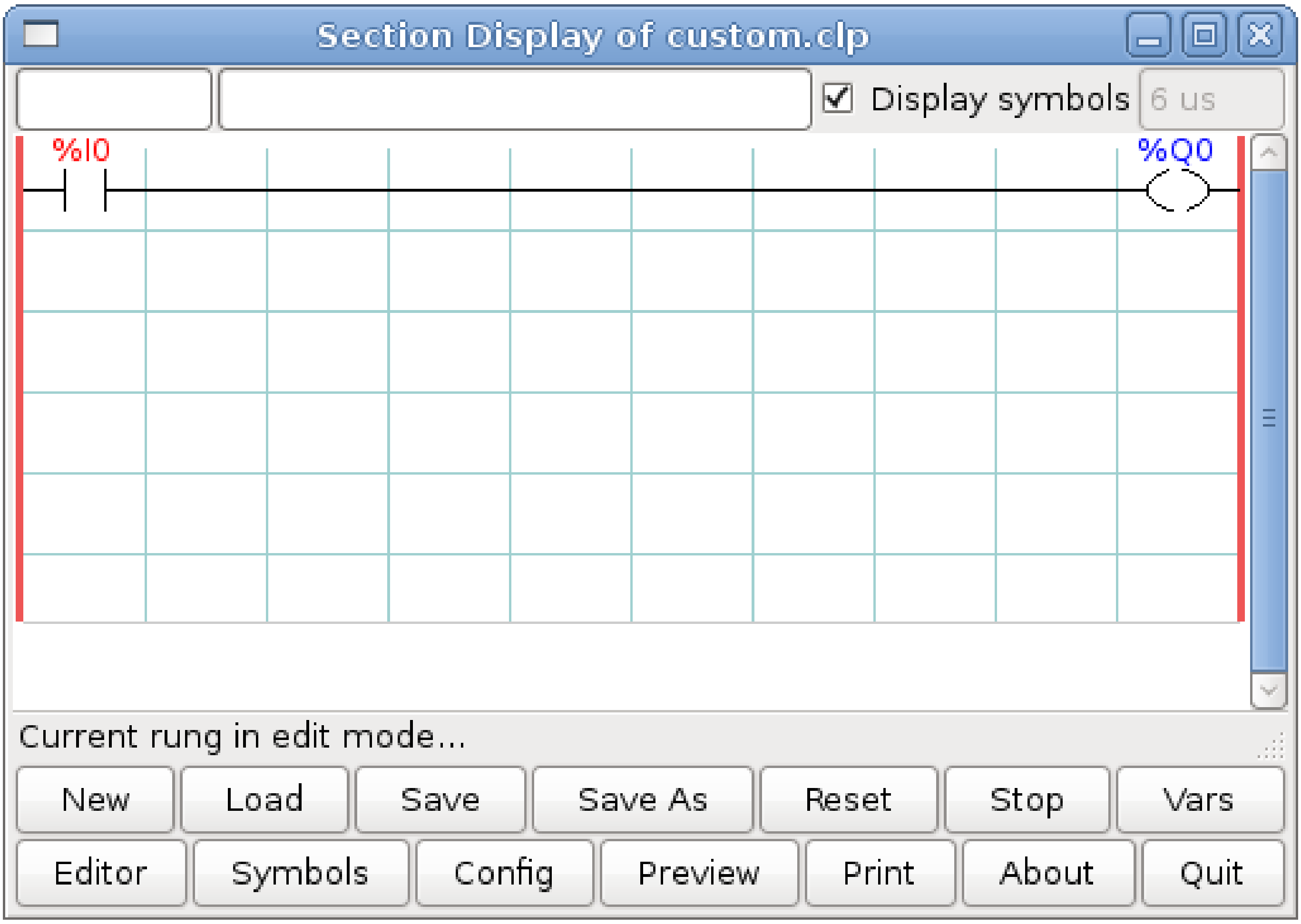


Figure 27.15: Section Display with Rung

Now click on the OK button in the Editor window. Now your Section Display should look like this.

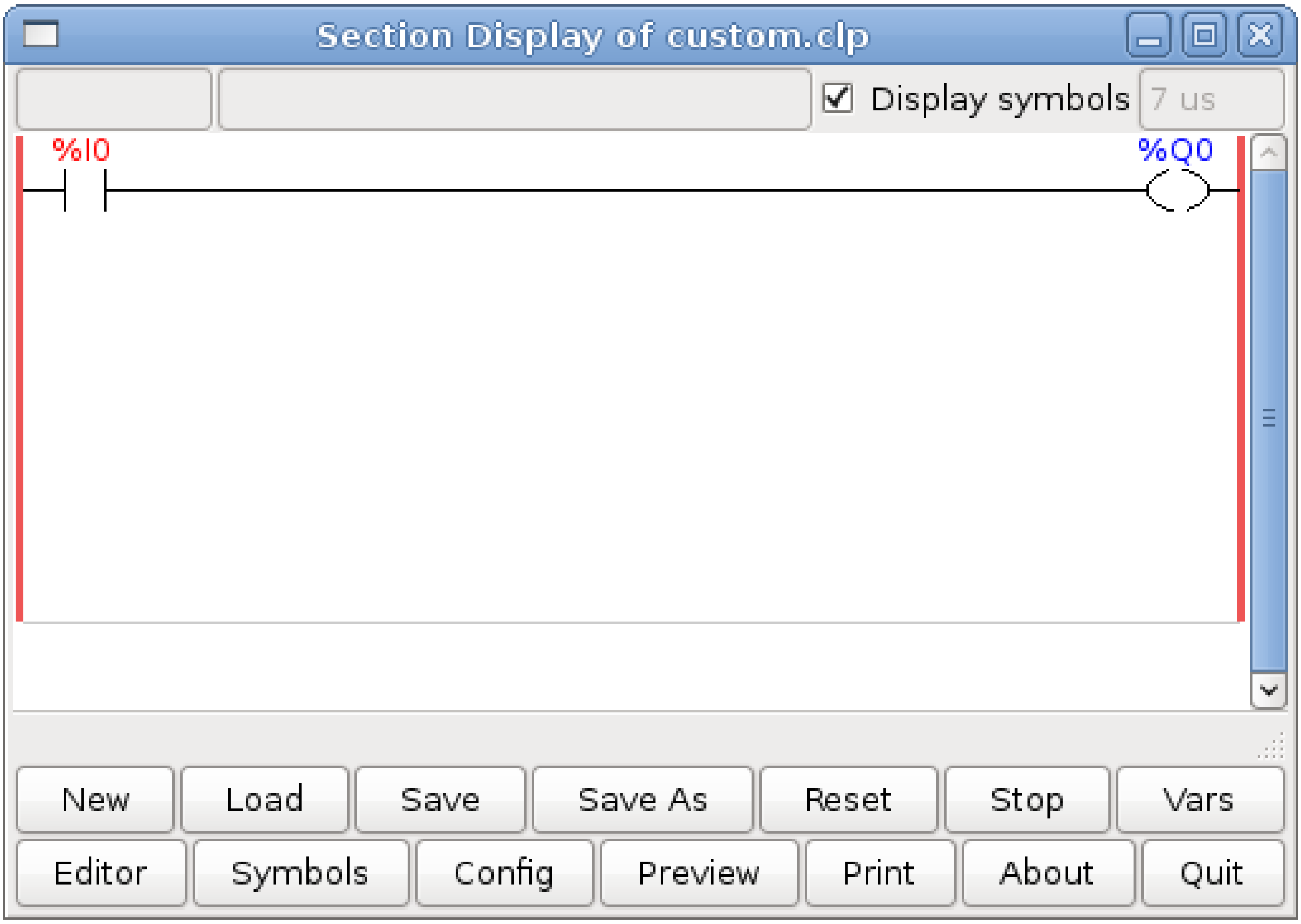


Figure 27.16: Section Display Finished

To save the new file select Save As and give it a name. The .clp extension will be added automatically. It should default to the running config directory as the place to save it.

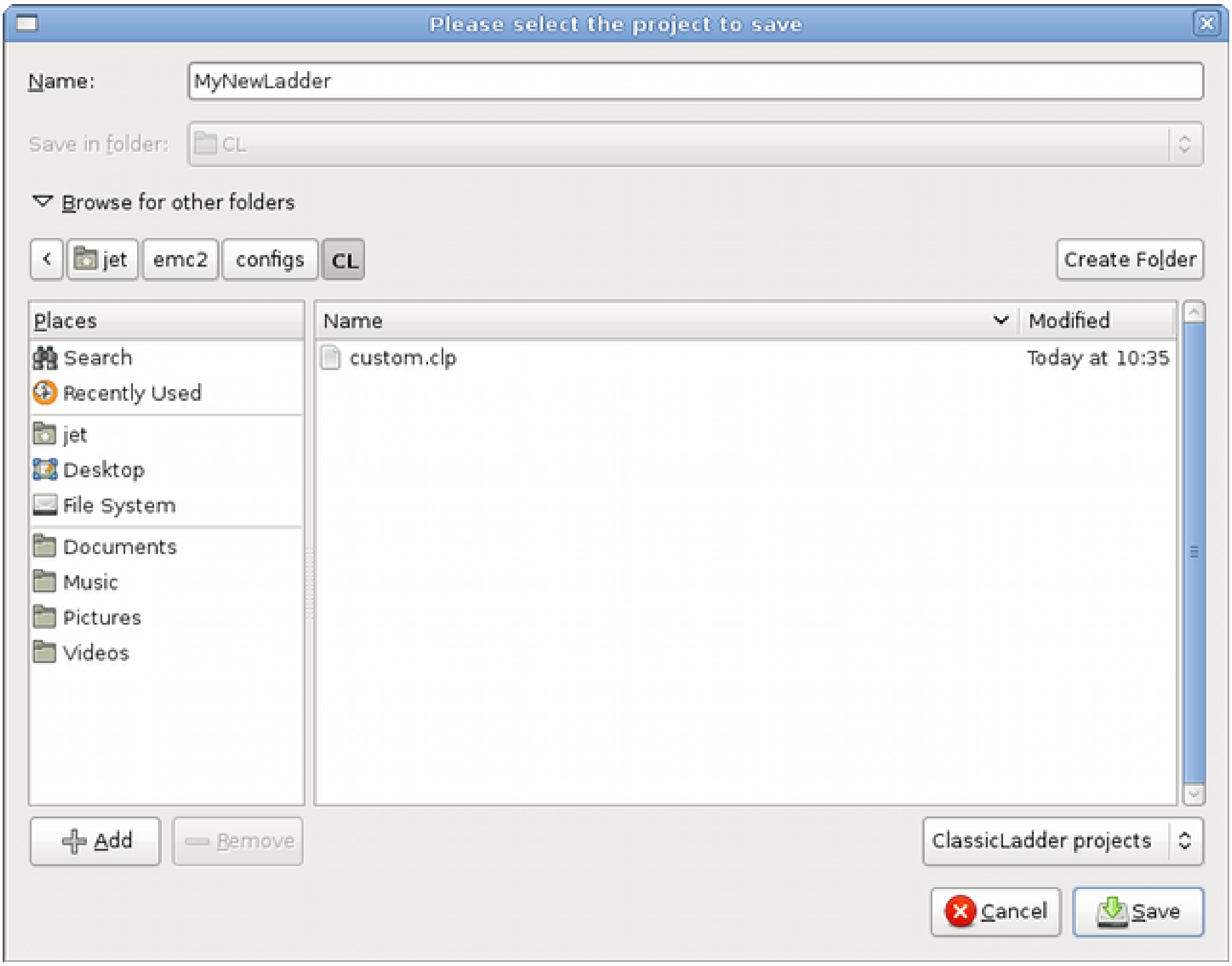


Figure 27.17: Save As Dialog

Again if you used the Stepconf Wizard to add Classic Ladder you can skip this step.

To manually add a ladder you need to add add a line to your custom.hal file that will load your ladder file. Close your LinuxCNC session and add this line to your custom.hal file.

```
loadusr -w classicladder --nogui MyLadder.clp
```

Now if you start up your LinuxCNC config your ladder program will be running as well. If you select "File/Ladder Editor", the program you created will show up in the Section Display window.

Chapter 28

Classicladder Examples

28.1 Wrapping Counter

To have a counter that *wraps around* you have to use the preset pin and the reset pin. When you create the counter set the preset at the number you wish to reach before wrapping around to 0. The logic is if the counter value is over the preset then reset the counter and if the underflow is on then set the counter value to the preset value. As you can see in the example when the counter value is greater than the counter preset the counter reset is triggered and the value is now 0. The underflow output %Q2 will set the counter value at the preset when counting backwards.

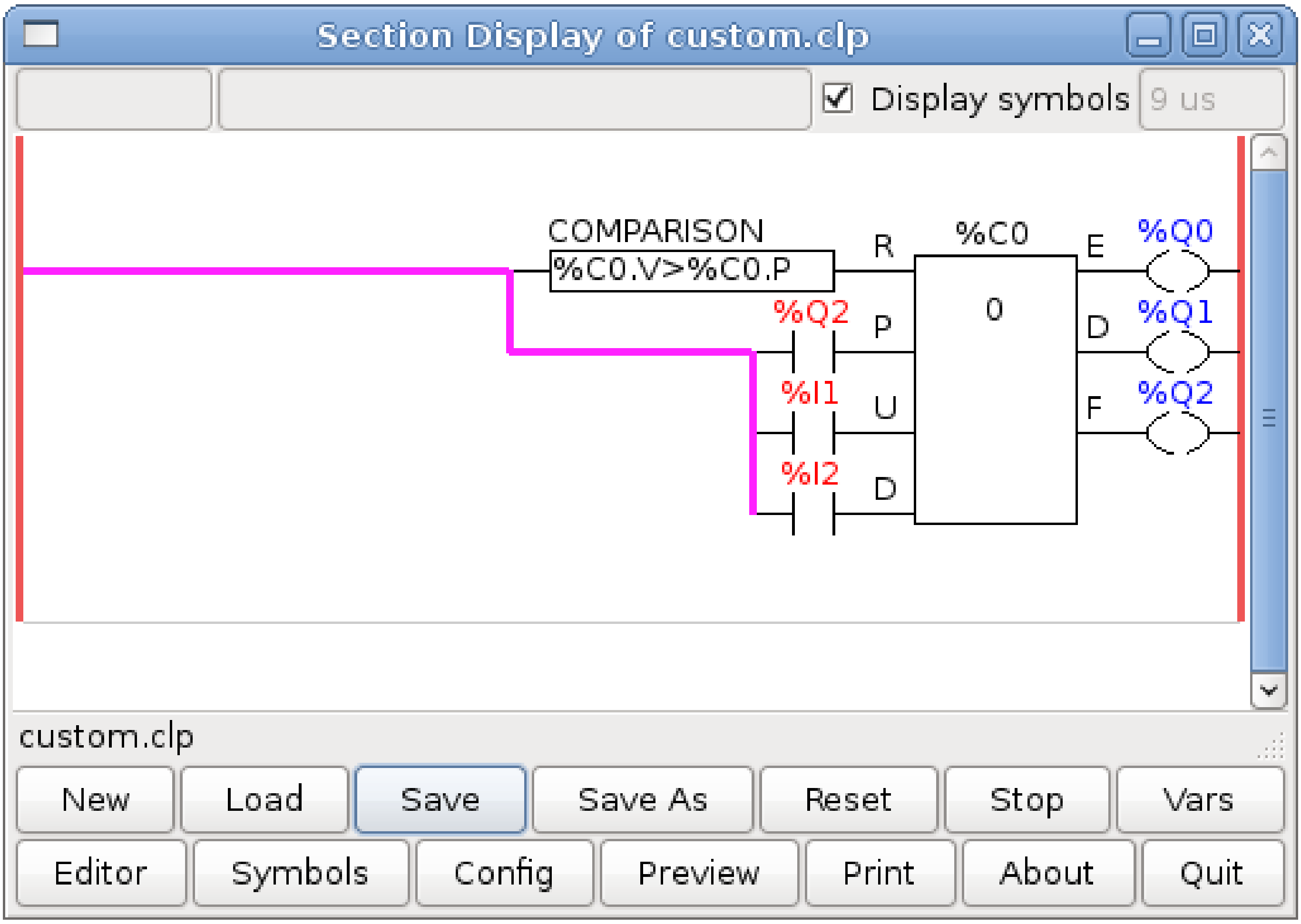


Figure 28.1: Wrapping Counter

28.2 Reject Extra Pulses

This example shows you how to reject extra pulses from an input. Suppose the input pulse %I0 has an annoying habit of giving an extra pulse that spoils our logic. The TOF (Timer Off Delay) prevents the extra pulse from reaching our cleaned up output %Q0. How this works is when the timer gets an input the output of the timer is on for the duration of the time setting. Using a normally closed contact %TM0.Q the output of the timer blocks any further inputs from reaching our output until it times out.

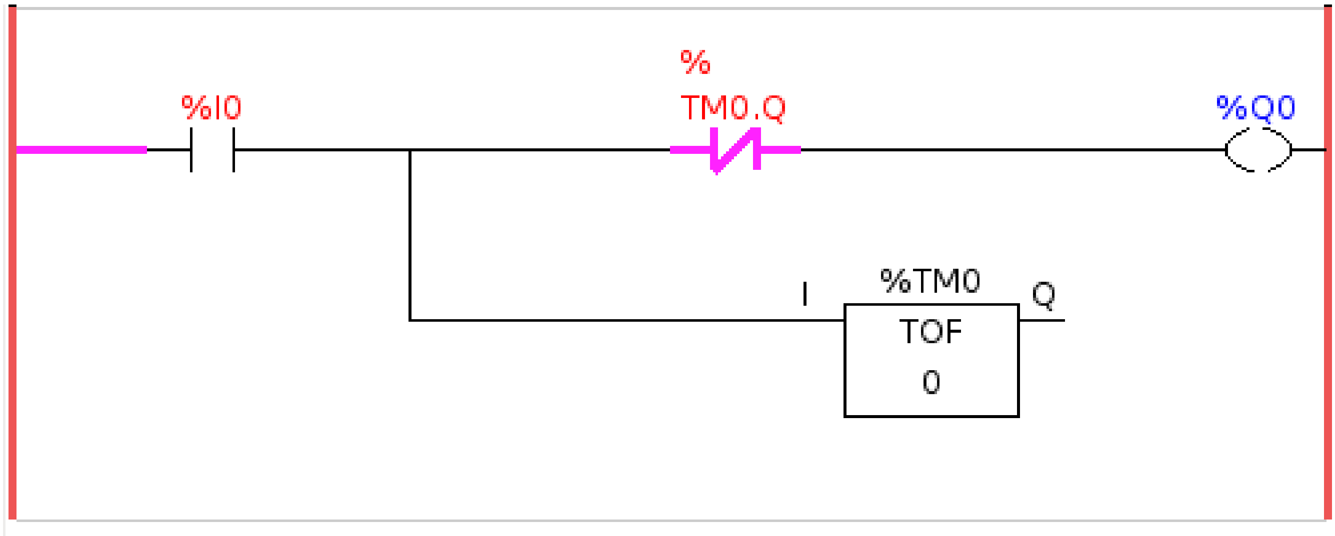


Figure 28.2: Reject Extra Pulse

28.3 External E-Stop

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

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

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

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

Next we add Classic Ladder to our custom.hal file by adding these two lines:

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

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

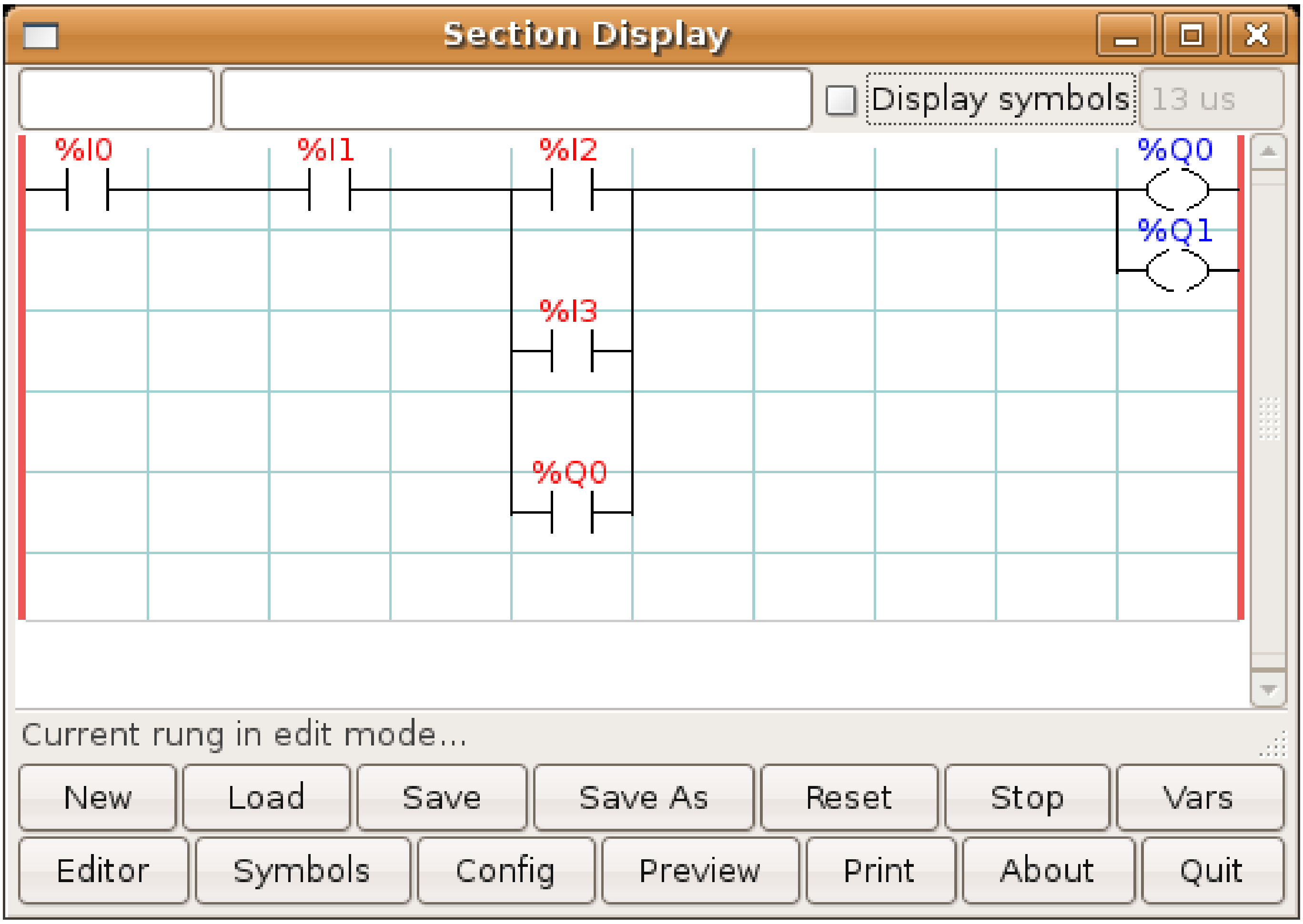


Figure 28.3: E-Stop Section Display

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

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

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

I/O assignments

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

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

```
# E-Stop example using pyVCP buttons to simulate external components

# The pyVCP checkbutton simulates a normally closed external E-Stop
net ext-estop classicladder.0.in-00 <= pyvcp.py-estop

# Request E-Stop Enable from LinuxCNC
net estop-all-ok iocontrol.0.emc-enable-in <= classicladder.0.out-00

# Request E-Stop Enable from pyVCP or external source
net ext-estop-reset classicladder.0.in-03 <= pyvcp.py-reset

# This line resets the E-Stop from LinuxCNC
 net emc-reset-estop iocontrol.0.user-request-enable =>
classicladder.0.in-02

# This line enables LinuxCNC to unlatch the E-Stop in Classic Ladder
net emc-estop iocontrol.0.user-enable-out => classicladder.0.in-01

# This line turns on the green indicator when out of E-Stop
net estop-all-ok => pyvcp.py-es-status
```

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

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

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

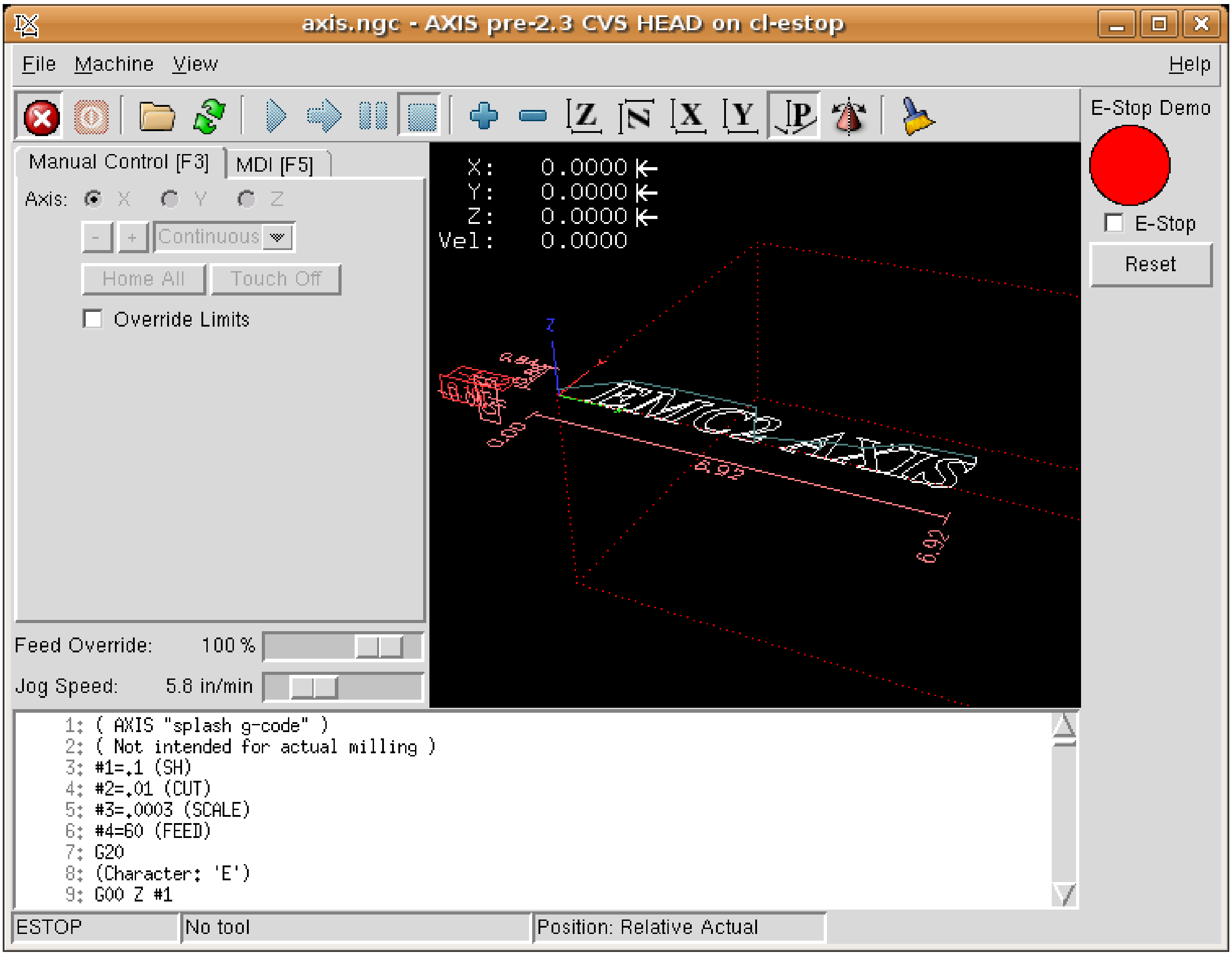


Figure 28.4: AXIS E-Stop

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

28.4 Timer/Operate Example

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

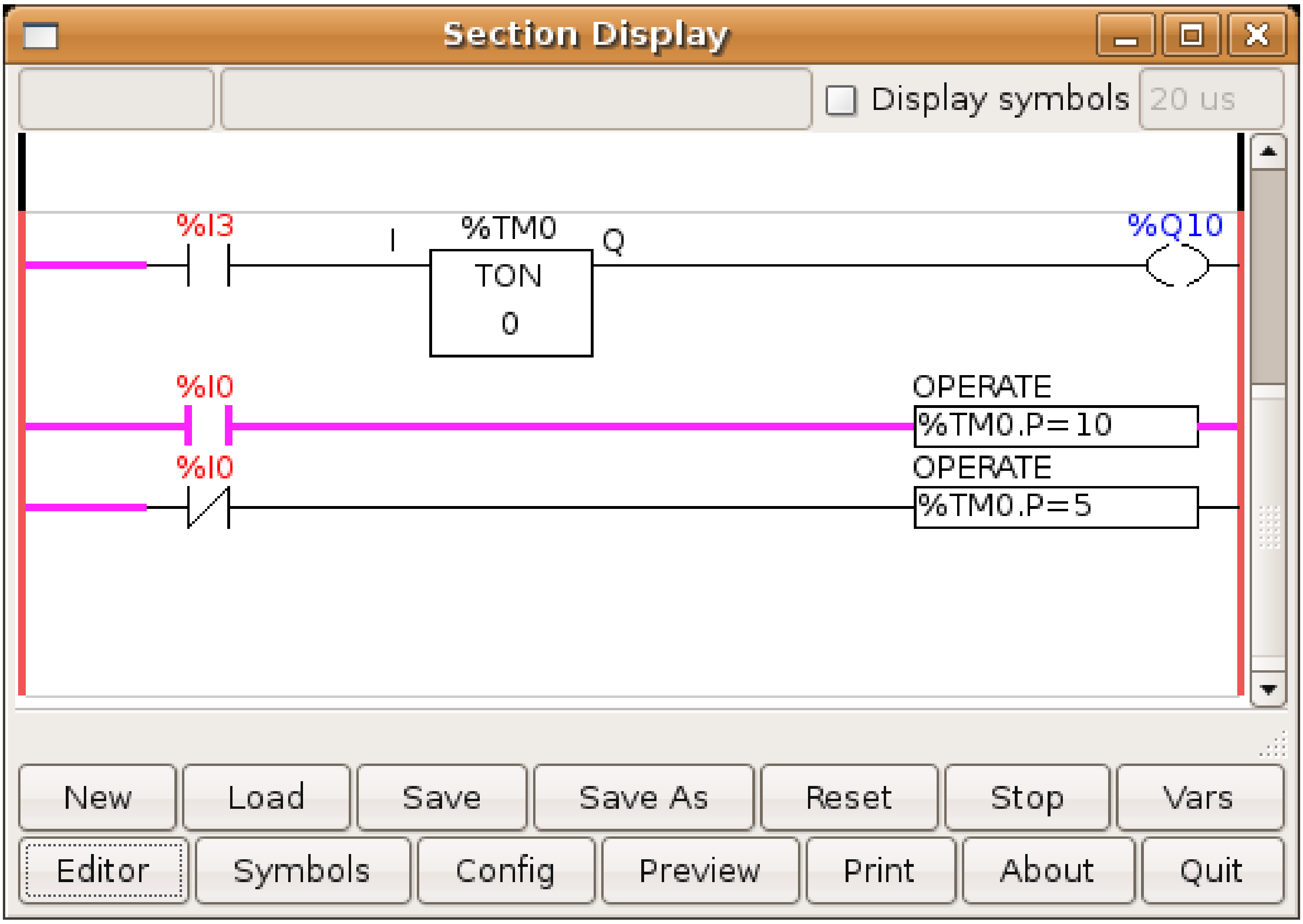


Figure 28.5: Timer/Operate Example

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

Part VII

Hardware Examples

Chapter 29

PCI Parallel Port

When you add a second parallel port to your PCI bus you have to find out the address before you can use it with LinuxCNC.

To find the address of your parallel port card open a terminal window and type

```
lspci -v
```

You will see something similar to this as well as info on everything else on the PCI bus:

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

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

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

to

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

(Note the double quotes surrounding the addresses.)

and then added the following lines so the parport will be read and written:

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

After doing the above then run your config and verify that the parallel port got loaded in Machine/Show HAL Configuration window.

Chapter 30

Spindle Control

30.1 0-10v Spindle Speed

If your spindle speed is controlled by an analog signal, (for example, by a VFD with a 0 to 10 volt signal) and you're using a DAC card like the m5i20 to output the control signal:

First you need to figure the scale of spindle speed to control signal. For this example the spindle top speed of 5000 RPM is equal to 10 volts.

$$\frac{10\,Volts}{5000\,RPM}=\frac{0.002\,Volts}{1\,RPM}$$

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

```
loadrt scale count=1
addf scale.0 servo-thread
setp scale.0.gain 0.002
net spindle-speed-scale motion.spindle-speed-out => scale.0.in
net spindle-speed-DAC scale.0.out => <your DAC pin name>
```

30.2 PWM Spindle Speed

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

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

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

30.3 Spindle Enable

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

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

30.4 Spindle Direction

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

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

```
net spindle-fwd motion.spindle-forward => parport.0.pin-16-out
net spindle-rev motion.spindle-reverse => parport.0.pin-17-out
```

30.5 Spindle Soft Start

If you need to ramp your spindle speed command and your control does not have that feature it can be done in HAL. Basically you need to hijack the output of motion.spindle-speed-out and run it through a limit2 component with the scale set so it will ramp the rpm from motion.spindle-speed-out to your device that receives the rpm. The second part is to let LinuxCNC know when the spindle is at speed so motion can begin.

In the 0-10 volt example the line *net spindle-speed-scale motion.spindle-speed-out => scale.0.in* is changed as shown in the following example:

Intro to HAL components limit2 and near:

In case you have not run across them before, here's a quick introduction to the two HAL components used in the following example.

- A "limit2" is a HAL component (floating point) that accepts an input value and provides an output that has been limited to a max/min range, and also limited to not exceed a specified rate of change.
- A "near" is a HAL component (floating point) with a binary output that says whether two inputs are approximately equal.

More info is available in the documentation for HAL components, or from the man pages, just say *man limit2* or *man near* in a terminal.

```
# load real time a limit2 and a near with names so it is easier to follow
loadrt limit2 names=spindle-ramp
loadrt near names=spindle-at-speed

# add the functions to a thread
addf spindle-ramp servo-thread
addf spindle-at-speed servo-thread

# set the parameter for max rate-of-change
# (max spindle accel/decel in units per second)
setp spindle-ramp.maxv 60

# hijack the spindle speed out and send it to spindle ramp in
net spindle-cmd <= motion.spindle-speed-out => spindle-ramp.in
```

```
# the output of spindle ramp is sent to the scale in
net spindle-ramped <= spindle-ramp.out => scale.0.in

# to know when to start the motion we send the near component
# (named spindle-at-speed) to the spindle commanded speed from
# the signal spindle-cmd and the actual spindle speed
# provided your spindle can accelerate at the maxv setting.
net spindle-cmd => spindle-at-speed.in1
net spindle-ramped => spindle-at-speed.in2

# the output from spindle-at-speed is sent to motion.spindle-at-speed
# and when this is true motion will start
net spindle-ready <= spindle-at-speed.out => motion.spindle-at-speed
```

30.6 Spindle Feedback

30.6.1 Spindle Synchronized Motion

Spindle feedback is needed by LinuxCNC to perform any spindle coordinated motions like threading and constant surface speed. The StepConf Wizard can perform the connections for you if you select Encoder Phase A and Encoder Index as inputs.

Hardware assumptions:

- An encoder is connected to the spindle and puts out 100 pulses per revolution on phase A
- The encoder A phase is connected to the parallel port pin 10
- The encoder index pulse is connected to the parallel port pin 11

Basic Steps to add the components and configure them:

```
# add the encoder to HAL and attach it to threads.footnote:[In this
example, we will assume that some encoders have already been issued
to axes/joints 0, 1, and 2. So the next encoder available for us to
attach to the spindle would be number 3. Your situation may differ.]
loadrt encoder num_chan=1
addf encoder.update-counters base-thread
addf encoder.capture-position servo-thread

# set the HAL encoder to 100 pulses per revolution.
setp encoder.3.position-scale 100

# set the HAL encoder to non-quadrature simple counting using A only.
setp encoder.3.counter-mode true

# connect the HAL encoder outputs to LinuxCNC.footnote:[The HAL encoder
index-enable is an exception to the rule in that it behaves
as both an input and an output, see manual for details]
net spindle-position encoder.3.position => motion.spindle-revs
net spindle-velocity encoder.3.velocity => motion.spindle-speed-in
net spindle-index-enable encoder.3.index-enable <=> motion.spindle-index-enable

# connect the HAL encoder inputs to the real encoder.
net spindle-phase-a encoder.3.phase-A <= parport.0.pin-10-in
net spindle-phase-b encoder.3.phase-B footnote:[It is because
we selected 'non-quadrature simple counting...' above that we
can get away with 'quadrature' counting without having any
B quadrature input.]
net spindle-index encoder.3.phase-Z <= parport.0.pin-11-in
```

30.6.2 Spindle At Speed

To enable LinuxCNC to wait for the spindle to be at speed before executing a series of moves you need to set motion.spindle-at-speed to true when the spindle is at the commanded speed. To do this you need spindle feedback from an encoder. Since the feedback and the commanded speed are not usually *exactly* the same you need to use the *near* component to say that the two numbers are close enough.

The connections needed are from the spindle velocity command signal to near.n.in1 and from the spindle velocity from the encoder to near.n.in2. Then the near.n.out is connected to motion.spindle-at-speed. The near.n.scale needs to be set to say how close the two numbers must be before turning on the output. Depending on your setup you may need to adjust the scale to work with your hardware.

The following is typical of the additions needed to your HAL file to enable Spindle At Speed. If you already have near in your HAL file then increase the count and adjust code to suit. Check to make sure the signal names are the same in your HAL file.

```
# load a near component and attach it to a thread
loadrt near
addf near.0 servo-thread

# connect one input to the commanded spindle speed
net spindle-cmd => near.0.in1

# connect one input to the encoder-measured spindle speed
net spindle-velocity => near.0.in2

# connect the output to the spindle-at-speed input
net spindle-at-speed motion.spindle-at-speed <= near.0.out

# set the spindle speed inputs to agree if within 1%
setp near.0.scale 1.01
```

Chapter 31

MPG Pendant

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

In your custom.hal file or 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 and encoder can be found in the HAL manual.

```
# Jog Pendant
loadrt encoder num_chan=1
loadrt mux4 count=1
addf encoder.capture-position servo-thread
addf encoder.update-counters base-thread
addf mux4.0 servo-thread

# If your MPG outputs a quadrature signal per click set x4 to 1
# If your MPG puts out 1 pulse per click set x4 to 0
setp encoder.0.x4-mode 0

# For velocity mode, set n to 1
# In velocity mode the axis stops when the dial is stopped
# even if that means the commanded motion is not completed,
# For position mode (the default), set n to 0
# In position mode the axis will move exactly jog-scale
# units for each count, regardless of how long that might take,
# This must be set for each axis you want in velocity mode
setp axis.N.jog-vel-mode n

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
```

Chapter 32

GS2 Spindle

This example shows the connections needed to use an Automation Direct GS2 VFD to drive a spindle. The spindle speed and direction is controlled by LinuxCNC.

Using the GS2 component involves very little to set up. We start with a Stepconf Wizard generated config. Make sure the pins with "Spindle CW" and "Spindle PWM" are set to unused in the parallel port setup screen.

In the custom.hal file we place the following to connect LinuxCNC to the GS2 and have LinuxCNC control the drive.

GS2 Example

```
# load the user space component for the Automation Direct GS2 VFD’s
loadusr -Wn spindle-vfd gs2_vfd -r 9600 -p none -s 2 -n spindle-vfd

# connect the spindle direction pin to the GS2
net gs2-fwd spindle-vfd.spindle-fwd <= motion.spindle-forward

# connect the spindle on pin to the GS2
net gs2-run spindle-vfd.spindle-on <= motion.spindle-on

# connect the GS2 at speed to the motion at speed
net gs2-at-speed motion.spindle-at-speed <= spindle-vfd.at-speed

# connect the spindle RPM to the GS2
net gs2-RPM spindle-vfd.speed-command <= motion.spindle-speed-out
```

Note
The transmission speed might be able to be faster depending on the exact envirnment. Both the drive and the command line options must match. To check for transmission errors add the *-v* command line option and run from a terminal.

On the GS2 drive itself you need to set a couple of things before the modbus communications will work. Other parameters might need to be set based on your physical requirements but these are beyond the scope of this manual. Refer to the GS2 manual that came with the drive for more information on the drive parameters.

- The communications switches must be set to RS-232C
- The motor parameters must be set to match the motor
- P3.00 (Source of Operation Command) must be set to Operation determined by RS-485 interface, 03 or 04
- P4.00 (Source of Frequency Command) must be set to Frequency determined by RS232C/RS485 communication interface, 05
- P9.01 (Transmission Speed) must be set to 9600 baud, 02
- P9.02 (Communication Protocol) must be set to "Modbus RTU mode, 8 data bits, no parity, 2 stop bits", 03

A PyVCP panel based on this example is here.

Part VIII

Diagnostics & FAQ

Chapter 33

Stepper Diagnostics

If what you get is not what you expect many times you just got some experience. Learning from the experience increases your understanding of the whole. Diagnosing problems is best done by divide and conquer. By this I mean if you can remove 1/2 of the variables from the equation each time you will find the problem the fastest. In the real world this is not always the case, but it's usually a good place to start.

33.1 Common Problems

33.1.1 Stepper Moves One Step

The most common reason in a new installation for a stepper motor not to move is that the step and direction signals are exchanged. If you press the jog forward and jog backward keys, alternately , and the stepper moves one step each time, and in the same direction, there is your clue.

33.1.2 No Steppers Move

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

33.1.3 Distance Not Correct

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

33.2 Error Messages

33.2.1 Following Error

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

- FERROR too small
- MIN_FERROR too small
- MAX_VELOCITY too fast
- MAX_ACCELERATION too fast

- BASE_PERIOD set too long
- Backlash added to an axis

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

If you added backlash you need to increase the STEPGEN_MAXACCEL up to double the MAX_ACCELERATION in the AXIS section of the INI file for each axis you added backlash to. LinuxCNC uses "extra acceleration" at a reversal to take up the backlash. Without backlash correction, step generator acceleration can be just a few percent above the motion planner acceleration.

33.2.2 RTAPI Error

When you get this error:

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

This error is generated by rtapi based on an indication from RTAI that a deadline was missed. It is usually an indication that the BASE_PERIOD in the [EMCMOT] section of the ini file is set too low. You should run the Latency Test for an extended period of time to see if you have any delays that would cause this problem. If you used the Stepconf Wizard, run it again, and test the Base Period Jitter again, and adjust the Base Period Maximum Jitter on the Basic Machine Information page. You might have to leave the test running for an extended period of time to find out if some hardware causes intermittent problems.

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

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

33.3 Testing

33.3.1 Step Timing

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

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

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

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

Chapter 34

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.

34.1 Automatic Login

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

34.2 Automatic Startup

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

Example:

```
emc /home/mill/emc2/config/mill/mill.ini
```

34.3 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.

34.4 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.

34.5 Editing a Root File

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

34.5.1 The Command Line Way

Open up *Applications* > *Accessories* > *Terminal*.

In the terminal window type

```
sudo gedit
```

Open the file with File > Open > Edit

34.5.2 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

34.5.3 Root Access

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

34.6 Terminal Commands

34.6.1 Working Directory

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

```
pwd
```

34.6.2 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
```

34.6.3 Listing files in a directory

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

```
dir
```

or

```
ls
```

34.6.4 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
```

And pwd might return the following result:

```
/home/joe
```

With this information put the command together like this:

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

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

34.6.5 Searching for Text

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

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

34.6.6 Bootup Messages

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

```
dmesg > bootmsg.txt
```

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

To clear the message buffer type this:

```
sudo dmesg -c
```

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

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

After boot up open a terminal and type:

```
dmesg|grep parport
```

34.7 Convenience Items

34.7.1 Terminal Launcher

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

34.8 Hardware Problems

34.8.1 Hardware Info

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

```
lspci -v
```

34.8.2 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

34.9 Paths

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

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

Chapter 35

Glossary

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

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

Axis
: One of the computer controlled movable parts of the machine. For a typical vertical mill, the table is the X axis, the saddle is the Y axis, and the quill or knee is the Z axis. Angular axes like rotary tables are referred to as A, B, and C. Additional linear axes relative to the tool are called U, V, and W respectively.

Axis(GUI)
: One of the Graphical User Interfaces available to users of LinuxCNC. It features the modern use of menus and mouse buttons while automating and hiding some of the more traditional LinuxCNC controls. It is the only open-source interface that displays the entire tool path as soon as a file is opened.

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

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

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

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

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

Comp
A tool used to build, compile and install LinuxCNC HAL components.

Configuration(n)
A directory containing a set of configuration files. Custom configurations are normally saved in the users home/emc2/configs directory. These files include EMC's traditional INI file and HAL files. A configuration may also contain several general files that describe tools, parameters, and NML connections.

Configuration(v)
The task of setting up LinuxCNC so that it matches the hardware on a machine tool.

Coordinate Measuring Machine
A Coordinate Measuring Machine is used to make many accurate measurements on parts. These machines can be used to create CAD data for parts where no drawings can be found, when a hand-made prototype needs to be digitized for moldmaking, or to check the accuracy of machined or molded parts.

Display units
The linear and angular units used for onscreen display.

DRO
A Digital Read Out is a system of position-measuring devices attached to the slides of a machine tool, which are connected to a numeric display showing the current location of the tool with respect to some reference position. DROs are very popular on hand-operated machine tools because they measure the true tool position without backlash, even if the machine has very loose Acme screws. Some DROs use linear quadrature encoders to pick up position information from the machine, and some use methods similar to a resolver which keeps rolling over.

EDM
EDM is a method of removing metal in hard or difficult to machine or tough metals, or where rotating tools would not be able to produce the desired shape in a cost-effective manner. An excellent example is rectangular punch dies, where sharp internal corners are desired. Milling operations can not give sharp internal corners with finite diameter tools. A *wire* EDM machine can make internal corners with a radius only slightly larger than the wire's radius. A *sinker* EDM can make internal corners with a radius only slightly larger than the radius on the corner of the sinking electrode.

EMC
The Enhanced Machine Controller. Initially a NIST project. 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 LinuxCNC.

Feed
Relatively slow, controlled motion of the tool used when making a cut.

Feed rate
The speed at which a cutting motion occurs. In auto or mdi mode, feed rate is commanded using an F word. F10 would mean ten machine units per minute.

Feedback
A method (e.g., quadrature encoder signals) by which LinuxCNC receives information about the position of motors

Feedrate Override
A manual, operator controlled change in the rate at which the tool moves while cutting. Often used to allow the operator to adjust for tools that are a little dull, or anything else that requires the feed rate to be "tweaked".

Floating Point Number
A number that has a decimal point. (12.300) In HAL it is known as float.

G-Code
The generic term used to refer to the most common part programming language. There are several dialects of G-code, EMC uses RS274/NGC.

GUI
Graphical User Interface.

General
A type of interface that allows communications between a computer and a human (in most cases) via the manipulation of icons and other elements (widgets) on a computer screen.

EMC
An application that presents a graphical screen to the machine operator allowing manipulation of the machine and the corresponding controlling program.

HAL
Hardware Abstraction Layer. At the highest level, it is simply a way to allow a number of building blocks to be loaded and interconnected to assemble a complex system. Many of the building blocks are drivers for hardware devices. However, HAL can do more than just configure hardware drivers.

Home
A specific location in the machine's work envelope that is used to make sure the computer and the actual machine both agree on the tool position.

ini file
A text file that contains most of the information that configures EMC for a particular machine

Instance
One can have an instance of a class or a particular object. The instance is the actual object created at runtime. In programmer jargon, the Lassie object is an instance of the Dog class.

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

Jog
Manually moving an axis of a machine. Jogging either moves the axis a fixed amount for each key-press, or moves the axis at a constant speed as long as you hold down the key. In manual mode, jog speed can be set from the graphical interface.

kernel-space
See real-time.

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

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

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

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

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

NML
Neutral Message Language provides a mechanism for handling multiple types of messages in the same buffer as well as simplifying the interface for encoding and decoding buffers in neutral format and the configuration mechanism.

Offsets
An arbitrary amount, added to the value of something to make it equal to some desired value. For example, gcode programs are often written around some convenient point, such as X0, Y0. Fixture offsets can be used to shift the actual execution point of that gcode program to properly fit the true location of the vise and jaws. Tool offsets can be used to shift the "uncorrected" length of a tool to equal that tool's actual length.

Part Program
A description of a part, in a language that the controller can understand. For EMC, that language is RS-274/NGC, commonly known as G-code.

Program Units
The linear and angular units used in a part program. The linear program units do not have to be the same as the linear machine units. See G20 and G21 for more information. The angular program units are always measured in degrees.

Python
General-purpose, very high-level programming language. Used in LinuxCNC for the Axis GUI, the Stepconf configuration tool, and several G-code programming scripts.

Rapid
Fast, possibly less precise motion of the tool, commonly used to move between cuts. If the tool meets the workpiece or the fixturing during a rapid, it is probably a bad thing!

Rapid rate
The speed at which a rapid motion occurs. In auto or mdi mode, rapid rate is usually the maximum speed of the machine. It is often desirable to limit the rapid rate when testing a g-code program for the first time.

Real-time
Software that is intended to meet very strict timing deadlines. Under Linux, in order to meet these requirements it is necessary to install 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
Generally, any motor that is used with error-sensing feedback to correct the position of an actuator. Also, a motor which is specially-designed to provide improved performance in such applications.

Servo Loop
A control loop used to control position or velocity of an motor equipped with a feedback device.

Signed Integer
A whole number that can have a positive or negative sign. In HAL it is known as s32. (A signed 32-bit integer has a usable range of -2,147,483,647 to +2,147,483,647.)

Spindle
The part of a machine tool that spins to do the cutting. On a mill or drill, the spindle holds the cutting tool. On a lathe, the spindle holds the workpiece.

Spindle Speed Override
A manual, operator controlled change in the rate at which the tool rotates while cutting. Often used to allow the operator to adjust for chatter caused by the cutter's teeth. Spindle Speed Override assumes that the LinuxCNC software has been configured to control spindle speed.

Stepconf
An LinuxCNC configuration wizard. It is able to handle many step-and-direction motion command based machines. It writes a full configuration after the user answers a few questions about the computer and machine that LinuxCNC is to run on.

Stepper Motor
A type of motor that turns in fixed steps. By counting steps, it is possible to determine how far the motor has turned. If the load exceeds the torque capability of the motor, it will skip one or more steps, causing position errors.

TASK
The module within EMC that coordinates the overall execution and interprets the part program.

Tcl/Tk
A scripting language and graphical widget toolkit with which several of LinuxCNCs GUIs and selection wizards were written.

Traverse Move
A move in a straight line from the start point to the end point.

Units
See "Machine Units", "Display Units", or "Program Units".

Unsigned Integer
A whole number that has no sign. In HAL it is known as u32. (An unsigned 32-bit integer has a usable range of zero to 4,294,967,296.)

World Coordinates
This is the absolute frame of reference. It gives coordinates in terms of a fixed reference frame that is attached to some point (generally the base) of the machine tool.

Chapter 36

Legal Section

36.1 Copyright Terms

36.2 GNU Free Documentation License

GNU Free Documentation License Version 1.1, March 2000

Copyright © 2000 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five). C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. H. Include an unaltered copy of this License. I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version. N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Chapter 37

Index