Portland State University

Capstone Project Progress Report

Department of Electrical & Computer Engineering

3D Metal Printer

Members:
Cameron Tribe
Branden Driver
Brian Andrews
Ahmad Qazi

Faculty Supervisor:
Dr. Marek Perkowski
Industry Sponsor:
Aram Kasparov

April 7, 2015

1 Project Overview

The team will interface a CNC machine with a MIG welder to create a 3D printer.

2 Project Proposal

2.1 Sponsor Proposal

The company is in the process of constructing an innovative 3D metal printer controlled by CNC (Computer Numerical Control). The project will be a combination of two machines:

- CNC mill – (3, 4 or 5 axis CNC mill)
- MIG/TIG welding machine.

The purpose for the CNC motion control (CNC mill) is to program and control motion of machine, in this case, the metal deposition process. The purpose for MIG welder is to deposit liquid metal. Many kinds of wire can be used by the welder to form the parts; carbon steel, titanium, stainless steel or aluminum. Idea for metal deposition and an example that uses a laser can be found at:
https://www.youtube.com/watch?feature=player_embedded&v=s9IdZ2pI5dA
A problem with laser use is its high cost. In this project it is planned to use the welding machine with the cost of 400. Another example can be found here:
http://www.wired.co.uk/magazine/archive/2014/08/play/steel-sketch
AKTechnology plan is to manufacture parts for pump and compressors, and R&D part for all use. The goal is to fabricate low cost and highly usable machines.

Company has CNC PC based CNC mill- motion controller.
https://www.youtube.com/watch?v=Plf3t7o951U&list=UUlGufPQeEKdN1-50F89Ejig
https://www.youtube.com/watch?v=G-jokU7v92E&list=UUlGufPQeEKdN1-50F89Ejig
https://www.youtube.com/watch?v=bPQ5UNiGA4c&list=UUlGufPQeEKdN1-50F89Ejig

The project is to upgrade this CNC motion controller – mill into 3D metal deposition printer by adding MIG welder instead of cutting tool spindle. The CNC motion control was reprogrammed. The MIG welder is operational.
The project will also build control to integrate CNC mill and MIG welding machine. Welder has 2 adjustment—- feed of wire and current. A stepper motor is planned to be used to control those analog data for wire feed and power current. The PLC, programmable logical controller, will join the CNC motion controller and the MIG welding machine.

2.2 The Goal

The end goal of this project is to fully integrate the MIG welder with the LinuxCNC system. Integration will include a way to control all of the functions of the welder, i.e. wire speed, maximum current output, engaging and disengaging the welder at appropriate times. In order for this to be done, electromechanical devices must be used to manipulate the knobs on the MIG welder. At the very least, the machine must be able to deposit material, reproducing a simple single object from a CAD drawing. Our aim is to produce a 1” cube. However, it is desired that the machine will be able to create complex structures on a single base. Precision of the deposition is not the primary concern, however it will be a requirement that the total amount of material deposited is more than the minimum tolerance of the part being created. This will allow for material to be machined away to a more precise tolerance.

2.3 Our Starting Point

The groundwork of this project has been completed by Aram Kasparov, the project sponsor. The project at its current state consists of a PC controlled CNC machine, a MIG welder, an infrared temperature sensor and a current measuring sensor. The PC controlling the CNC machine is running a Linux operating system. LinuxCNC an open-source software is used for programing and interfacing with the physical machine. Additional hardware is installed onto the PC, consisting of Mesa Electronics 5I20 FPGA based PCI Anything I/O card, 7i33 analog servo interface card and two 7i37-COM isolated I/O cards. The LinuxCNC software communicates the control signals and receives feedback through these cards. The CNC machine is a 3-axis machine-that is it can move in the X, Y and Z directions. Each axis is moved by a servo-motor and each servo motor is driven by a driver which receives its control commands from the PC. The machine is functional, though the motors will require some tuning and limit switches need to be programmed in (they are physically installed on the machine but not included in the program). The MIG/Flux cored welder is rated at 180 Amp-DC, 240 Volt with a duty cycle of 20% at 140 amps. The welder has current and wire feed adjustment capabilities for controlling the weld. These two knobs will be controlled by two stepper motors which have been installed onto the welder already. The current sensor has the ability to measure up to 225A. It has been demonstrated to be functional and will be used to monitor the current of the weld. The infrared non-contact temperature sensor is rated to measure temperatures up to 1800 degrees Celsius, though no tests have been performed yet.

2.4 Requirements

- Must use a wire feed welder
- Welder must have a Control System
- Must measure weld temperature
- Must measure weld current
- Must use both previous parameters to estimate current quality of weld
- Must use “G code” as inputs
- And must control when material is being deposited
- Must have user interface
- Should allow for welder thermal shutdown
- Should Measure Wire Speed from welder

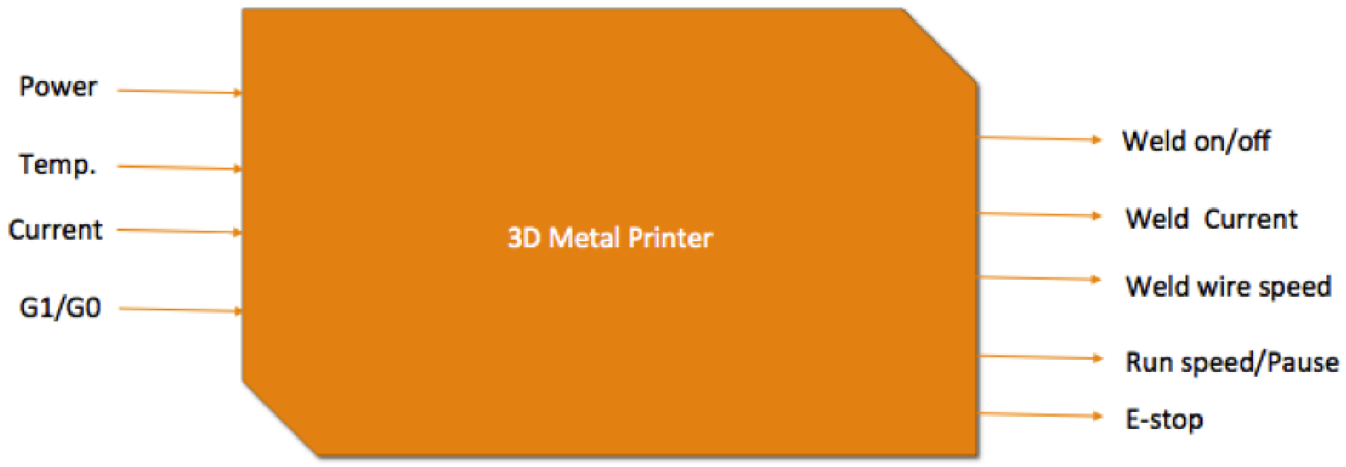


Figure 1: General Black Box Diagram of the 3D Metal Printer

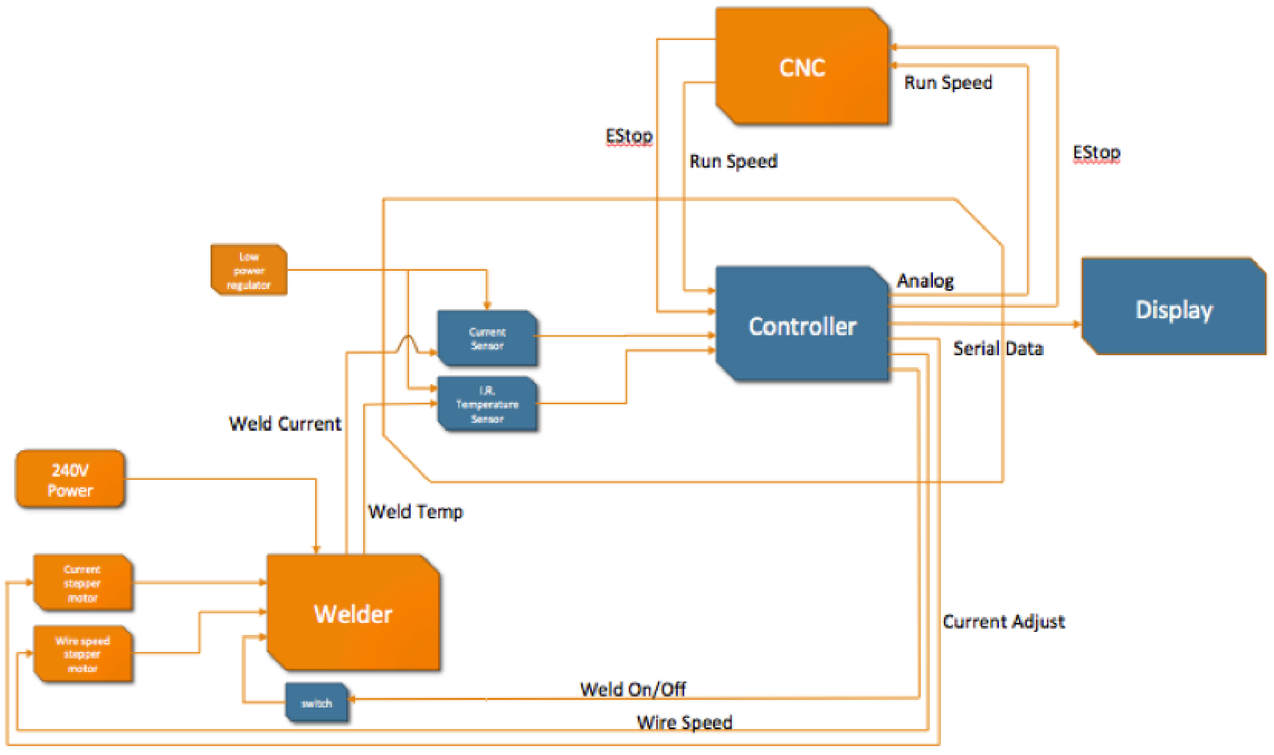


Figure 2: Detailed Black Box Diagram of the 3D Metal Printer

3 Hardware

3.1 Welder Control

To control the welder, a central control module will be used. This was a hot topic of debate for several weeks, as the number of choices available for this project are very high. The sponsor's requirements for the project was that all of the control work was done by a separate computer from the one used by Linux CNC, this only narrowed it down to a choice between a PCIe DAC board and a single board computer. Based on the need for both analog and digital control pins and the need for future expansion, we researched and came up with several options.

Single Board Computers	**DAC**
Raspberry-Pi	Sensoray 826
Intel Galileo	MCC DAS1602/16
SBC 8600B	
Wander Board Solo	
Beagle Bone Black	

In the end we chose the Sensoray 826 board because for the price it outperforms all other boards on the market by having 16 analog inputs, 8 analog outputs, and 48 digital I/O pins.

This board was chosen for the high level of future expandability that it has, and because it is a PCIe card which can be packaged into its own desktop as per request from the sponsor.

To control the current to the weld and the wire speed of the welder, two stepper motors have been fitted to the manual control knobs, and are connected to a motor driver module. The Sensoray board will be controlling the motor drivers using a sequence of rising and falling edges. To allow the controller board to control at what time the welder is depositing and when it is not depositing, a relay with a transistor driver will be used.

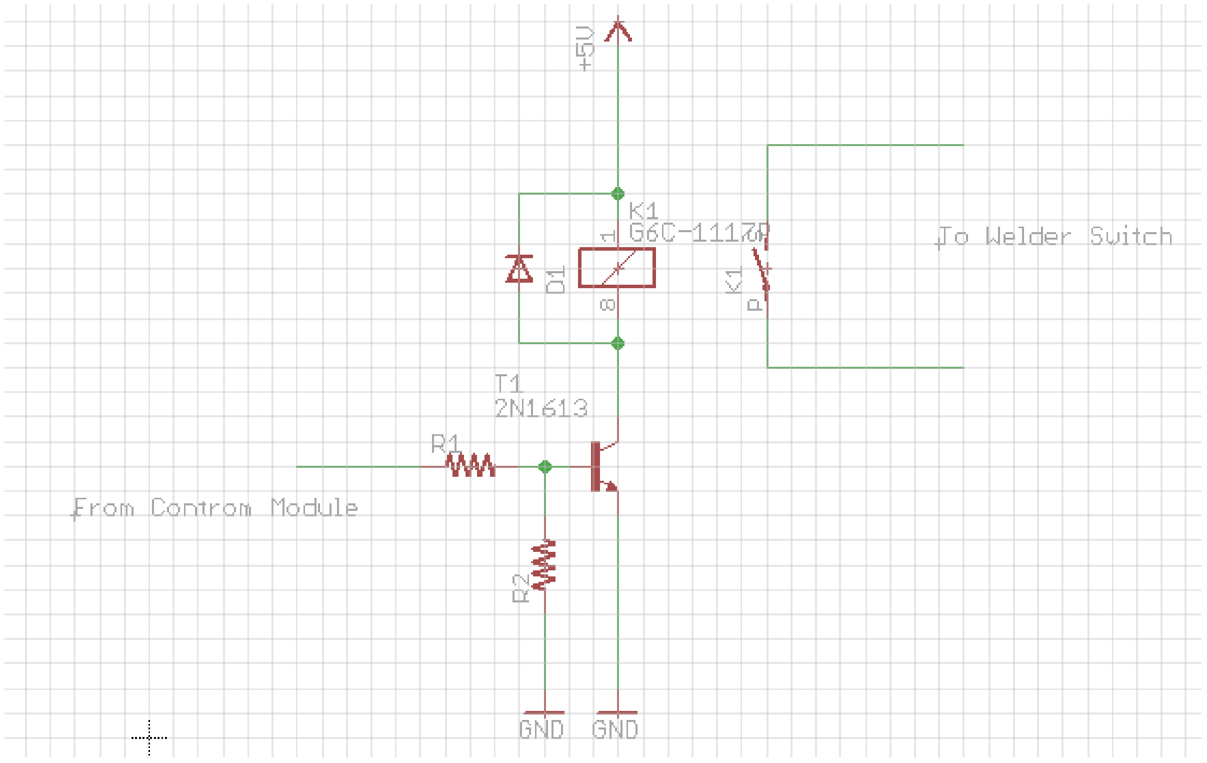


Figure 3: The Relay with the Transistor Driver

The signal that tells the controller will be coming from the CNC machine's I/O card. It is a switch type signal which means that when the signal is sent, an internal switch will be closed, causing what ever is on the input to be shown on the output. The CNC machine uses G-Code (described below), and Linux CNC allows outputs to be asserted when a particular G-Code instruction is executed. G1 and G0 are going to be used to tell the I/O card to close and open the switch, which will assert 5V DC to the input to the control module. The control module will assert an output high or low which will open or close the welder switch, turning the welder on and off.

The Sensoray 826 I/O card has three 50 pin connectors and two 26 pin connectors. To allow easy access to these pins, a breakout board with screw terminals has been made so that wires can easily be disconnected and switched.

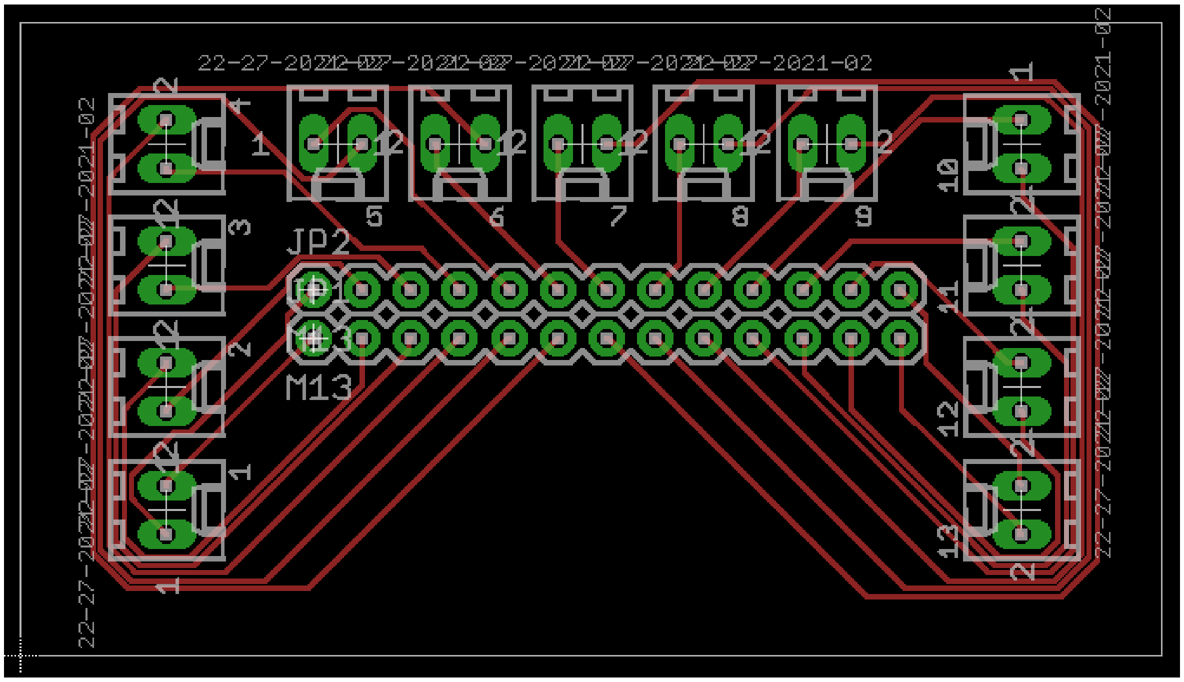


Figure 4: Schematic of the 26 pin Breakout Board for the Sensoray 826 I/O card

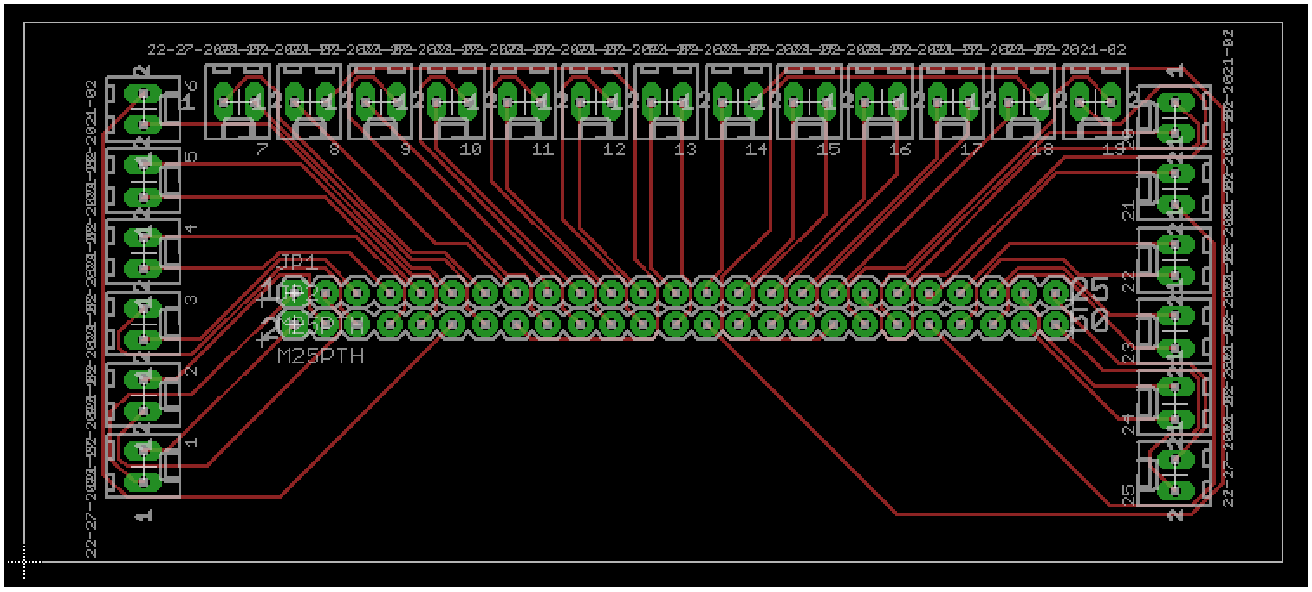


Figure 5: Schematic of the 50 pin Breakout Board for the Sensoray 826 I/O card

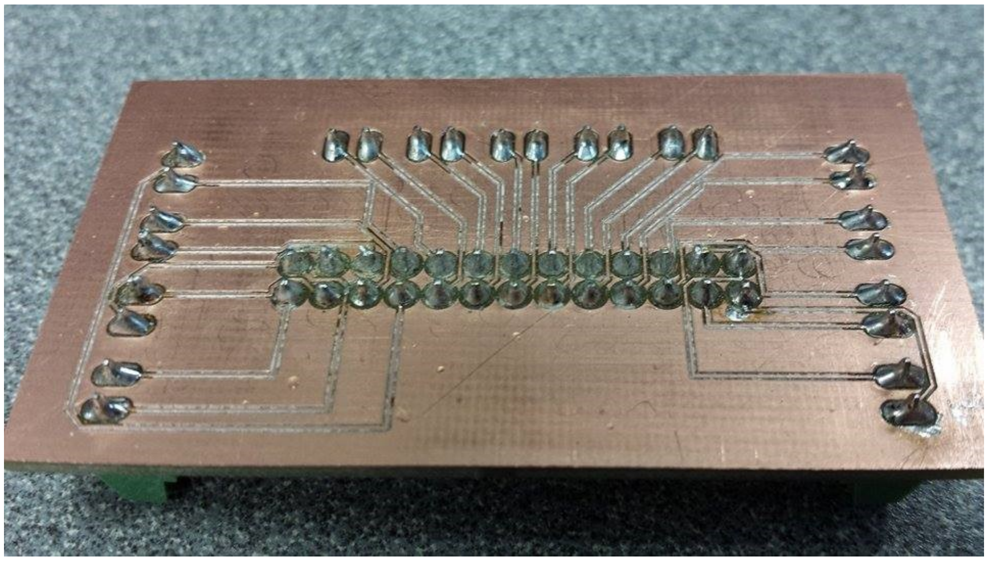


Figure 6: The 26 pin Breakout Board for The Sensoray 826 I/O card

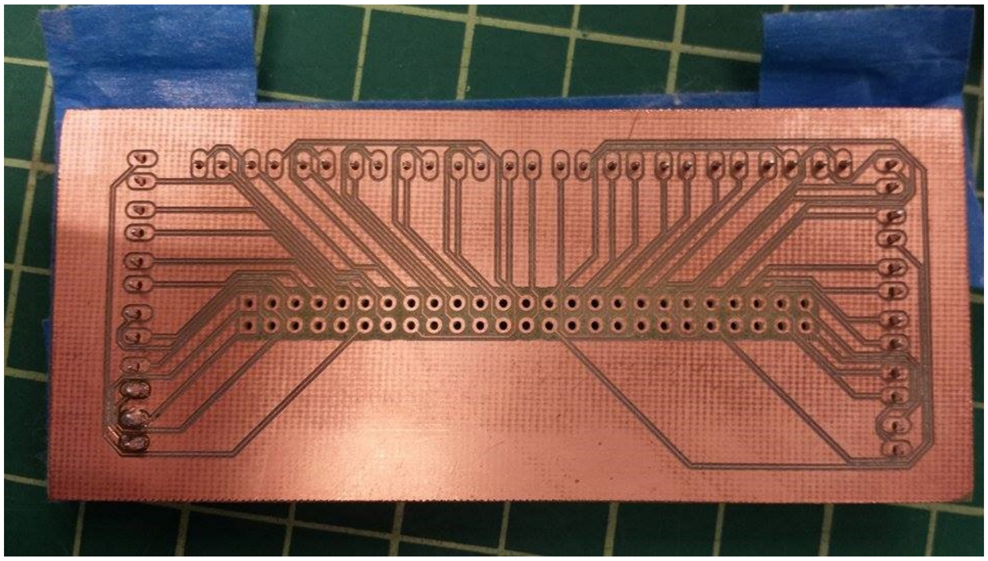


Figure 7: The 50 pin Breakout Board for the Sensoray 826 I/O card

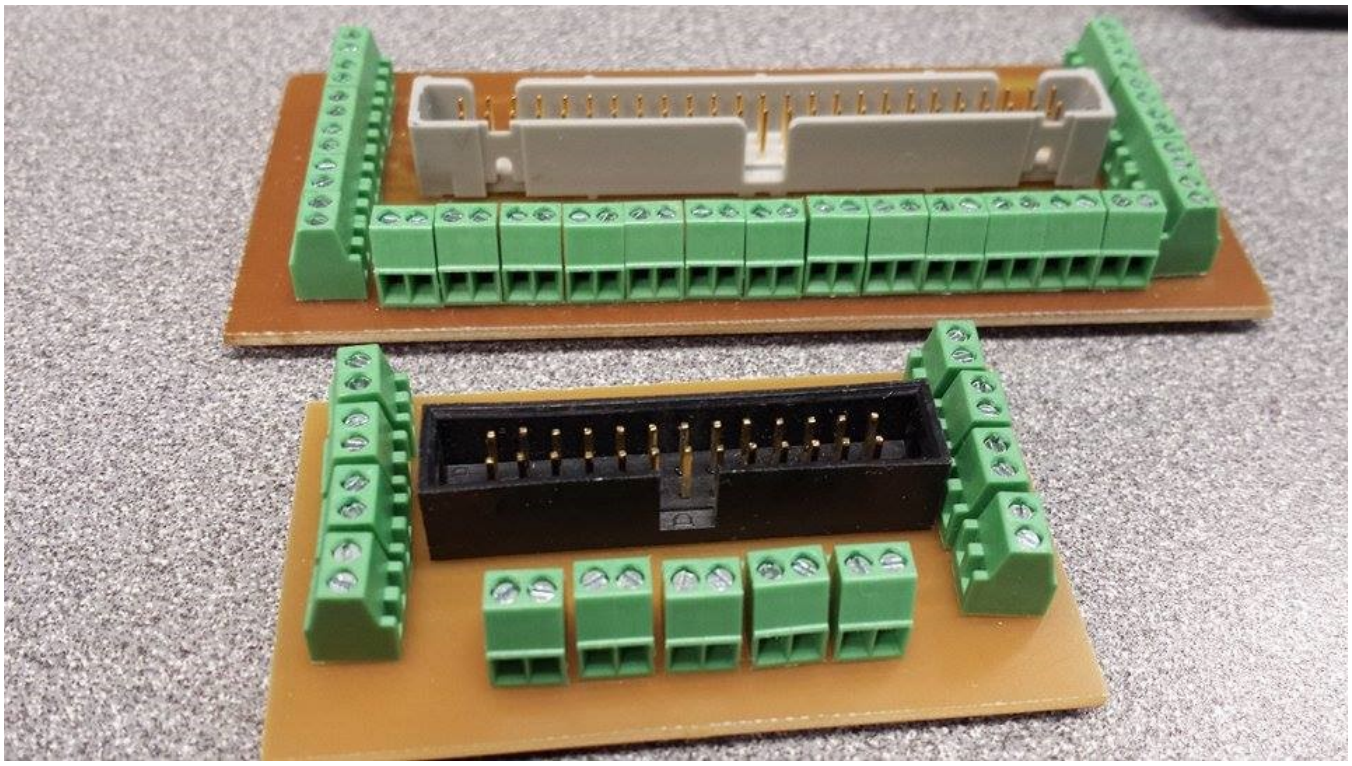
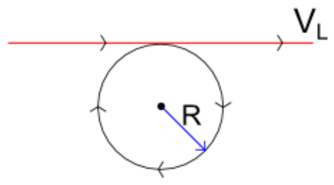


Figure 8: The 26 and 50 pin Breakout Board with connectors for The Sensoray 826 I/O card

3.2 Wire Speed

For calculating Wire Speed with Rotary Encoder



So Angular Velocity is,

$$\boldsymbol{f = \frac{n}{Nt}}$$

$n =$ number of up/down counts, $N =$ number of up/down counts/rev, $T =$ sampling time.

And Linear Velocity is,

$$v_L = \omega r$$
$$\omega = 2\pi f$$

So,

$$\boldsymbol{V = \frac{2\pi n r}{NT}}$$

**Note: For Sensoray 826, Max* $f = 25MHz$

Connections to 826

J_4	Pin 1	+A0
	Pin 2	-A0
	Pin 3	GND
	Pin 4	+B0
	Pin 5	-B0

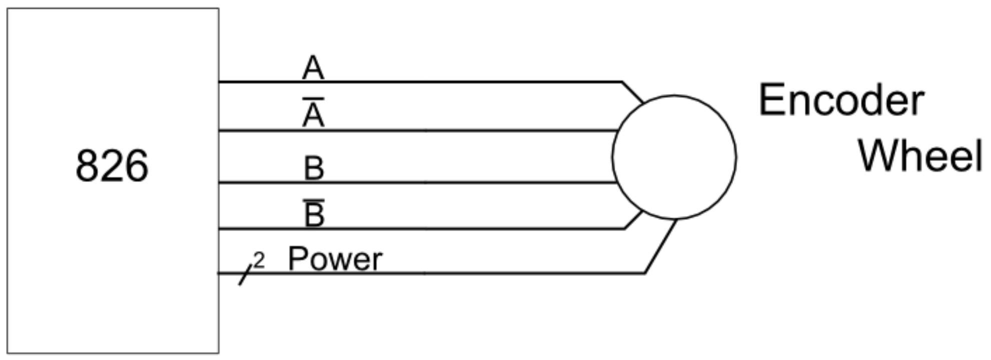


Figure 9: Encoder Connection

4 Software

4.1 G-Code

G-code is the commonly used name to refer to a numerical programming language. As is the case with all languages, G-code has its own syntax and semantics. A line of code is referred to as a block, and a program is defined as multiple blocks. All programs start and end with the percent symbol (%). While writing G-code, the backslash (/) can be used to comment out an entire line, whereas if you want to make comments about a block, parentheses are to be used. Anything inside of a set of parentheses will be ignored by the compiler. As is the case with most other languages, G-code ignores white space, so spacing is used to clarify the code for the writer as well as future users.

All points of G code are comprised of words and numbers. A word is simply a letter. For example, the block "X0" is simply the word X and the value 0. The words X, Y, and Z refer to the three axes, while the G words refer to the movement, motion, and location. When first started up, any blocks of code will use the point (0,0,0) as home, however G54 establishes a new temporary "home" point and G52 establishes the point where the temporary reference point is to be set. In other words, the block of code "G54 G52 X100 Y100 Z0" changes the reference point from (0,0,0) to (100,100,0) and the remainder of the code will run from this point, unless a new reference point is defined.

Similarly, other G-code words can be used to define how the space between two points should be interpolated. In some instances you may want two points to be connected via a straight line, while at other times it would be better to have them connected via a circular pattern. When dealing with circular interpolation, you can set it to be done in either a clockwise or counterclockwise manner. Beyond just linear and circular interpolation, there are dozens of G-code words that determine how the code is to be run (Appendix C).

4.2 GUI Description

Our software will have a GUI in which the user can see real-time graphed data coming from the current and temperature sensors as well as see the current wire speed. The system should use this feedback to automatically adjust the weld and keep it in a state that can be considered a "good weld". Also included here will be manual overrides for the user to adjust the current and wire speed to their own desired result. Fig. 10 shows an initial GUI layout that we will be aiming for.

4.3 The Graphical User Interface (GUI)

We are using GTK+ in C programming language to generate the Graphical User Interface in order to view the different data and also control different settings.

The GTK+ is a library for creating graphical user interfaces. The library is created in C programming language. The GTK+ library is also called the GIMP toolkit. Originally, the library was created while developing the GIMP image manipulation program. Since then, the GTK+ became one of the most popular toolkits under Linux and BSD Unix. Today, most of the GUI software in the open source world is created in Qt or in GTK+. The GTK+ is an object oriented application programming interface. The object oriented system is created with the Glib object system, which is a base for the GTK+ library. The GObject also enables to create language bindings for various other programming languages. Language bindings exist for C++, Python, Perl, Java, C#, and other programming languages.

The GTK+ itself depends on the following libraries.

- Glib
- Pango
- ATK
- GDK
- GdkPixbuf
- Cairo

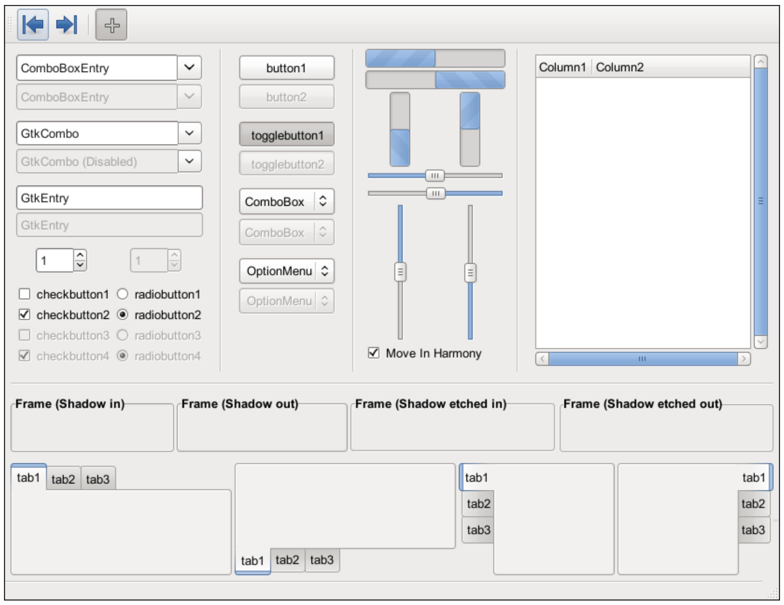
*Note: For detailed functions of each library refer to Appendix A

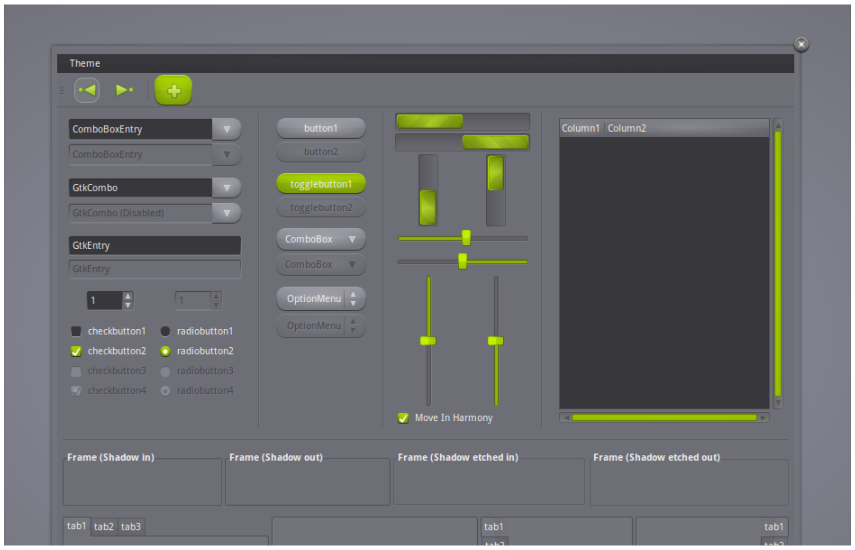
4.4 Reasons for using GTK+

- Language Bindings
  GTK+ is available in many other programming languages thanks to the language bindings available. This makes GTK+ quite an attractive toolkit for application development.

- Interfaces
  GTK+ has a comprehensive collection of core widgets and interfaces for use in your application.
  - Windows (normal window or dialog, about and assistant dialogs)
  - Displays (label, image, progress bar, status bar)
  - Buttons and toggles (check buttons, radio buttons, toggle buttons and link buttons)
  - Numerical (horizontal or vertical scales and spin buttons) and text data entry (with or without completion)
  - Multi-line text editor
  - Tree, list and icon grid viewer (with customizable renderers and model/view separation)
  - Combo box (with or without an entry)
  - Menus (with images, radio buttons and check items)
  - Toolbars (with radio buttons, toggle buttons and menu buttons)
  - GtkBuilder (creates your user interface from XML)
  - Selectors (color selection, file chooser, font selection)
  - Layouts (tabulated widget, table widget, expander widget, frames, separators and more)
  - Status icon (notification area on Linux, tray icon on Windows)
  - Printing widgets
  - Recently used documents (menu, dialog and manager)

4.5 Examples of GUIs created using GTK+





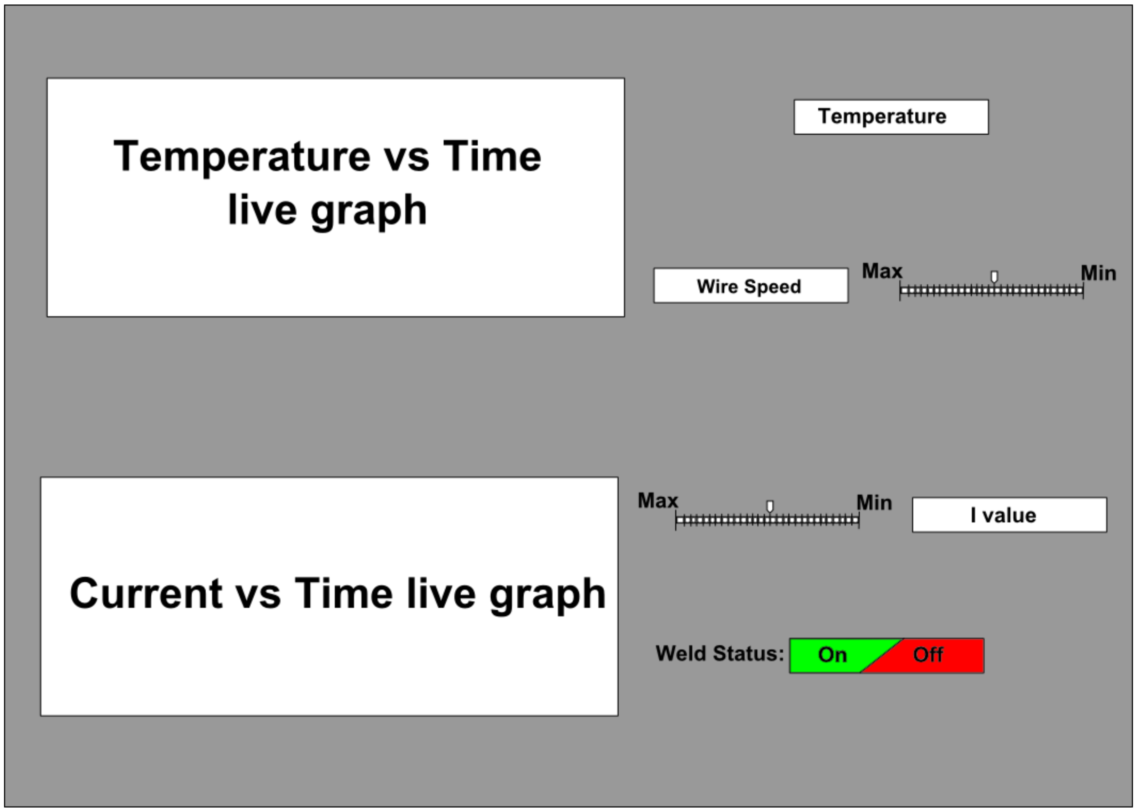
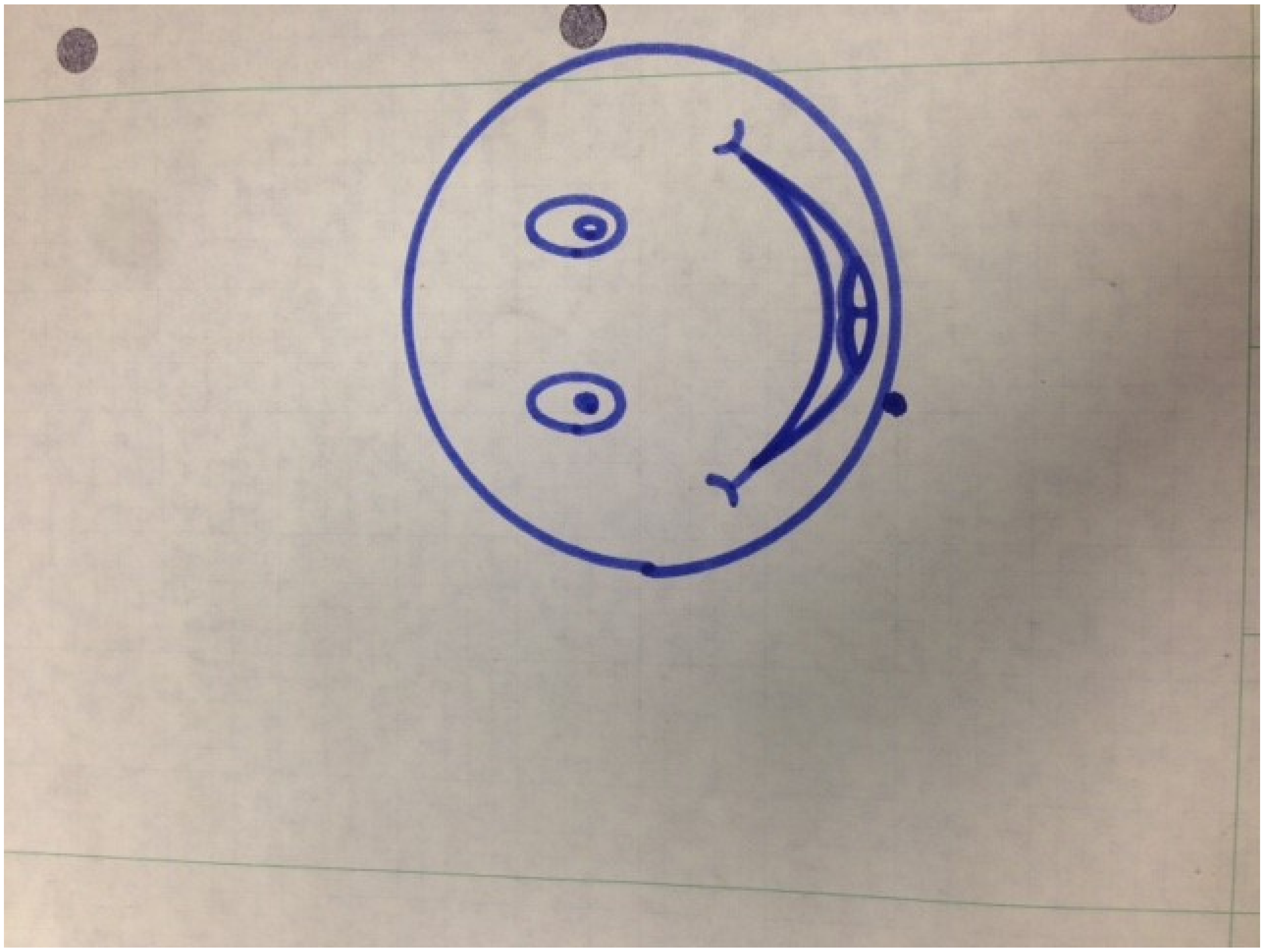


Figure 10: Proposed Rough GUI Layout

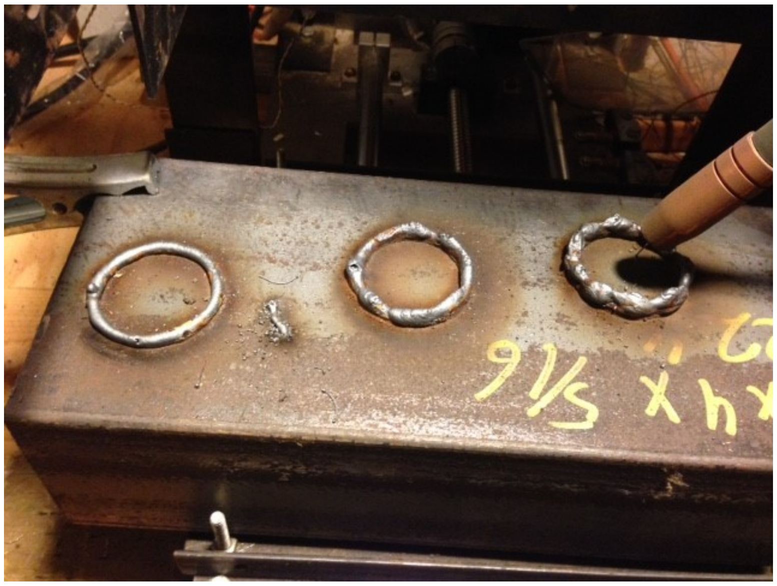
5 Photos of Progress

One of the fist things done in this project was to confirm the operation of the CNC machine. To do this, G-Code of a 2D image was uploaded to the machine. A felt marker was used to draw the image below.



Next, we fitted the welder to the machine, and placed a metal base plate to weld on. To control the welder we just used our hand to active the weld while the machine was moving.

After this, we connected the relay switch circuit in parallel with the manual welder switch, using G-Code to activate the switch. Shown Below is the result of letting the CNC Machine control the weld.



6 Bill of Materials (BOM)

Item	Quantity	Part Number	Mfg	Price	Description
CNC Machine					
X-Stepper Motor					
Y-Stepper Motor					
Z-Stepper Motor					
PCIe DAQ	1	826	Sensoray	$677	
Limit Switches	9				
PCI I/O Card		5I20	Mesa Electronics		
Servo Interface Card		7i33			
Isolated I/O Card	2	7i37			
Temperature Sensor					
Current Sensor					
MIG Welder			Chicago Electric		
Motor Controller					

*Note: Preliminary BOM: specific quantities and part numbers will be added as progress continues.

.

Appendix A

- **Glib**
  GLib provides the core application building blocks for libraries and applications written in C. It provides the core object system used in GNOME, the main loop implementation, and a large set of utility functions for strings and common data structures.
  Description

  - New types which are not part of standard C (but are defined in various C standard library header files) - gboolean, gsize, gssize,goffset, gintptr, guintptr.
  - Integer types which are guaranteed to be the same size across all platforms - gint8, guint8, gint16, guint16, gint32, guint32,gint64, guint64.
  - Types which are easier to use than their standard C counterparts - gpointer, gconstpointer, guchar, guint, gushort, gulong.

  GLib also defines macros for the limits of some of the standard integer and floating point types, as well as macros for suitableprintf() formats for these types.
  Standard Macros — commonly-used macros
  Type Conversion Macros — portably storing integers in pointer variables
  Byte Order Macros — a portable way to convert between different byte orders
  Numerical Definitions — mathematical constants, and floating point decomposition
  Miscellaneous Macros — specialized macros which are not used often
  Atomic Operations — basic atomic integer and pointer operations
  GLib Core Application Support

  - **The Main Event Loop** — manages all available sources of events
  - **Threads** — portable support for threads, mutexes, locks, conditions and thread private data
  - **Thread Pools** — pools of threads to execute work concurrently
  - **Asynchronous Queues** — asynchronous communication between threads
  - **Dynamic Loading of Modules** — portable method for dynamically loading 'plug-ins'
  - **Memory Allocation** — general memory-handling
  - **Memory Slices** — efficient way to allocate groups of equal-sized chunks of memory
  - **IO Channels** — portable support for using files, pipes and sockets
  - **Error Reporting** — a system for reporting errors
  - **Message Output and Debugging Functions** — functions to output messages and help debug applications
  - **Message Logging** — versatile support for logging messages with different levels of importance

GLib Utilities

- **String Utility Functions** — various string-related functions
- **Character Set Conversion** — convert strings between different character sets
- **Unicode Manipulation** — functions operating on Unicode characters and UTF-8 strings
- **Base64 Encoding** — encodes and decodes data in Base64 format
- **Data Checksums** — computes the checksum for data
- **Secure HMAC Digests** — computes the HMAC for data
- **Internationalization** — gettext support macros
- **Date and Time Functions** — calendrical calculations and miscellaneous time stuff
- **GTimeZone** — a structure representing a time zone
- **GDateTime** — a structure representing Date and Time
- **Random Numbers** — pseudo-random number generator
- **Hook Functions** — support for manipulating lists of hook functions
- **Miscellaneous Utility Functions** — a selection of portable utility functions
- **Lexical Scanner** — a general purpose lexical scanner
- **Timers** — keep track of elapsed time
- **Spawning Processes** — process launching
- **File Utilities** — various file-related functions
- **URI Functions** — manipulating URIs
- **Hostname Utilities** — Internet hostname utilities
- **Shell-related Utilities** — shell-like commandline handling
- **Commandline option parser** — parses commandline options
- **Glob-style pattern matching** — matches strings against patterns containing '*' (wildcard) and '?' (joker)
- **Perl-compatible regular expressions** — matches strings against regular expressions
- **Regular expression syntax** — syntax and semantics of regular expressions supported by GRegex
- **Simple XML Subset Parser** — parses a subset of XML
- **Key-value file parser** — parses .ini-like config files
- **Bookmark file parser** — parses files containing bookmarks
- **Testing** — a test framework
- **UNIX-specific utilities and integration** — pipes, signal handling

- **Windows Compatibility Functions** — UNIX emulation on Windows

GLib Data Types

- **Doubly-Linked Lists** — linked lists that can be iterated over in both directions
- **Singly-Linked Lists** — linked lists that can be iterated in one direction
- **Double-ended Queues** — double-ended queue data structure
- **Sequences** — scalable lists
- **Trash Stacks** — maintain a stack of unused allocated memory chunks
- **Hash Tables** — associations between keys and values so that given a key the value can be found quickly
- **Strings** — text buffers which grow automatically as text is added
- **String Chunks** — efficient storage of groups of strings
- **Arrays** — arrays of arbitrary elements which grow automatically as elements are added
- **Pointer Arrays** — arrays of pointers to any type of data, which grow automatically as new elements are added
- **Byte Arrays** — arrays of bytes
- **Balanced Binary Trees** — a sorted collection of key/value pairs optimized for searching and traversing in order
- **N-ary Trees**— trees of data with any number of branches
- **Quarks** — a 2-way association between a string and a unique integer identifier
- **Keyed Data Lists** — lists of data elements which are accessible by a string or GQuark identifier
- **Datasets** — associate groups of data elements with particular memory locations
- **GVariantType** — introduction to the GVariant type system
- **GVariant** — strongly typed value datatype
- **GVariant Format Strings** — varargs conversion of GVariants
- **GVariant Text Format** — textual representation of GVariants

Deprecated APIs

- **Deprecated thread API** — old thread APIs (for reference only)
- **Caches** — caches allow sharing of complex data structures to save resources
- **Relations and Tuples** — tables of data which can be indexed on any number of fields
- **Automatic String Completion** — support for automatic completion using a group of target strings

GLib Tools

- **glib-gettextize** — gettext internationalization utility
- **gtester** — test running utility
- **gtester-report** — test report formatting utility

- **Pango**

  Pango is a library for laying out and rendering of text, with an emphasis on internationalization. Pango can be used anywhere that text layout is needed, though most of the work on Pango so far has been done in the context of the GTK+ widget toolkit. Pango forms the core of text and font handling for GTK+-2.x.
  Pango is designed to be modular; the core Pango layout engine can be used with different font backends. There are three basic backends, with multiple options for rendering with each.

  - Client side fonts using the FreeType and fontconfig libraries, using HarfBuzz for complex-text handling. Rendering can be with with Cairo or Xft libraries, or directly to an in-memory buffer with no additional libraries.
  - Native fonts on Microsoft Windows using Uniscribe for complex-text handling. Rendering can be done via Cairo or directly using the native Win32 API.
  - Native fonts on MacOS X using CoreText for complex-text handling, rendering via Cairo.

  The integration of Pango with Cairo provides a complete solution with high quality text handling and graphics rendering.
  Dynamically loaded modules then handle text layout for particular combinations of script and font backend. Pango ships with a wide selection of modules, including modules for Hebrew, Arabic, Hangul, Thai, and a number of Indic scripts. Virtually all of the world's major scripts are supported. As well as the low level layout rendering routines, Pango includes PangoLayout, a high level driver for laying out entire blocks of text, and routines to assist in editing internationalized text. Pango depends on 2.x series of the GLib library.

- **ATK**

  ATK provides the set of accessibility interfaces that are implemented by other toolkits and applications. Using the ATK interfaces, accessibility tools have full access to view and control running applications.
  Base accessibility object

  - **AtkObject** — The base object class for the Accessibility Toolkit API.

  Event and Toolkit Support

  - **AtkUtil** — A set of ATK utility functions for event and toolkit support

ATK Interfaces

- **AtkAction** — The ATK interface provided by UI components which the user can activate/interact with.
- **AtkComponent** — The ATK interface provided by UI components which occupy a physical area on the screen. which the user can activate/interact with.
- **AtkDocument** — The ATK interface which represents the toplevel container for document content.
- **AtkEditableText** — The ATK interface implemented by components containing user-editable text content.
- **AtkHyperlinImpl** — An interface from which the AtkHyperlink associated with an AtkObject may be obtained.
- **AtkHypertext** — The ATK interface which provides standard mechanism for manipulating hyperlinks.
- **AtkImage** — The ATK Interface implemented by components which expose image or pixmap content on-screen.
- **AtkSelection** - The ATK interface implemented by container objects whose AtkObject children can be selected.
- **AtkStreamableContent** — The ATK interface which provides access to streamable content.
- **AtkTable** — The ATK interface implemented for UI components which contain tabular or row/column information.
- **AtkTableCell** - The ATK interface implemented for a cell inside a two-dimentional AtkTable
- **AtkText** — The ATK interface implemented by components with text content.
- **AtkValue** — The ATK interface implemented by valuators and components which display or select a value from a bounded range of values.
- **AtkWindow** — The ATK Interface provided by UI components that represent a top-level window.

Basic accessible data types

- **AtkRange** - A given range or subrange, to be used with AtkValue
- **AtkRelation** — An object used to describe a relation between a object and one or more other objects.
- **AtkRelationSet** — A set of AtkRelations, normally the set of AtkRelations which an AtkObject has.
- **AtkState** — An AtkState describes a single state of an object.
- **AtkStateSet** — An AtkStateSet contains the states of an object.

Custom accessible objects

- **AtkGObjectAccessible** — This object class is derived from AtkObject and can be used as a basis implementing accessible objects.
- **AtkHyperlink** — An ATK object which encapsulates a link or set of links in a hypertext document.
- **AtkNoOpObject** — An AtkObject which purports to implement all ATK interfaces.
- **AtkPlug** — Toplevel for embedding into other processes
- **AtkSocket** — Container for AtkPlug objects from other processes

Utilities

- **AtkNoOpObjectFactory** — The AtkObjectFactory which creates an AtkNoOpObject.
- **AtkObjectFactory** — The base object class for a factory used to create accessible objects for objects of a specific GType.
- **AtkRegistry** — An object used to store the GType of the factories used to create an accessible object for an object of a particular GType.
- **Versioning macros** — Variables and functions to check the ATK version

Deprecated Interfaces

- **AtkMisc** — A set of ATK utility functions for thread locking

• **GDK**

I **API Reference**

- **General** — Library initialization and miscellaneous functions
- **GdkDisplayManager** — Maintains a list of all open GdkDisplays
- **GdkDisplay** — Controls a set of GdkScreens and their associated input devices
- **GdkScreen** — Object representing a physical screen
- **GdkDeviceManager** — Functions for handling input devices
- **GdkDevice** — Object representing an input device
- **Points and Rectangles** — Simple graphical data types
- **Pixbufs** — Functions for obtaining pixbufs
- **RGBA Colors** — RGBA colors
- **Visuals** — Low-level display hardware information
- **Cursors** — Standard and pixmap cursors
- **Windows** — Onscreen display areas in the target window system
- **Frame clock** — Frame clock syncs painting to a window or display

  - **Frame timings** — Object holding timing information for a single frame
  - **GdkGLContext** — OpenGL context
  - **Events** — Functions for handling events from the window system
  - **Event Structures** — Data structures specific to each type of event
  - **Key Values** — Functions for manipulating keyboard codes
  - **Selections** — Functions for transfering data via the X selection mechanism
  - **Drag And Drop** — Functions for controlling drag and drop handling
  - **Properties and Atoms** — Functions to manipulate properties on windows
  - **Threads** — Functions for using GDK in multi-threaded programs
  - **Pango Interaction** — Using Pango in GDK
  - **Cairo Interaction** — Functions to support using cairo
  - **X Window System Interaction** — X backend-specific functions
  - **Wayland Interaction** — Wayland backend-specific functions
  - **Application launching** — Startup notification for applications
  - **Testing** — Test utilities

  II **Deprecated**

  - **Colors** — Manipulation of colors

- **GdkPixbuf**

  I **API Reference**

  - **Initialization and Versions** — Library version numbers.
  - **The GdkPixbuf Structure** — Information that describes an image.
  - **Reference Counting and Memory Management** — Functions for reference counting and memory management on pixbufs.
  - **File Loading** — Loading a pixbuf from a file.
  - **File saving** — Saving a pixbuf to a file.
  - **Image Data in Memory** — Creating a pixbuf from image data that is already in memory.
  - **Inline data** — Functions for inlined pixbuf handling.
  - **Scaling** — Scaling pixbufs and scaling and compositing pixbufs
  - **Rendering** — Rendering a pixbuf to a GDK drawable.
  - **Drawables to Pixbufs** — Getting parts of a GDK drawable's image data into a pixbuf.
  - **Utilities** — Utility and miscellaneous convenience functions.
  - **Animations** — Animated images.
  - **GdkPixbufLoader** — Application-driven progressive image loading.
  - **Module Interface** — Extending GdkPixBuf
  - **gdk-pixbuf Xlib initialization** — Initializing the gdk-pixbuf Xlib library.

- **Xlib Rendering** — Rendering a pixbuf to an X drawable.
- **X Drawables to Pixbufs** — Getting parts of an X drawable's image data into a pixbuf.
- **XlibRGB** — Rendering RGB buffers to X drawables.

II **Tools Reference**

- **gdk-pixbuf-csource** — C code generation utility for GdkPixbuf images
- **gdk-pixbuf-query-loaders** — GdkPixbuf loader registration utility

- **Cairo**

  A Vector Graphics Library.
  Drawing

  - **cairo_t** — The cairo drawing context
  - **Paths** — Creating paths and manipulating path data
  - **cairo_pattern_t** — Sources for drawing
  - **Regions** — Representing a pixel-aligned area
  - **Transformations** — Manipulating the current transformation matrix
  - **text** — Rendering text and glyphs
  - **Raster Sources** — Supplying arbitrary image data

  Fonts

  - **cairo_font_face_t** — Base class for font faces
  - **cairo_scaled_font_t** — Font face at particular size and options
  - **cairo_font_options_t** — How a font should be rendered
  - **FreeType Fonts** — Font support for FreeType
  - **Win32 Fonts** — Font support for Microsoft Windows
  - **Quartz (CGFont) Fonts** — Font support via CGFont on OS X
  - **User Fonts** — Font support with font data provided by the user

  Surfaces

  - **cairo_device_t** — interface to underlying rendering system
  - **cairo_surface_t** — Base class for surfaces
  - **Image Surfaces** — Rendering to memory buffers
  - **PDF Surfaces** — Rendering PDF documents
  - **PNG Support** — Reading and writing PNG images
  - **PostScript Surfaces** — Rendering PostScript documents

- **Recording Surfaces** — Records all drawing operations
- **Win32 Surfaces** — Microsoft Windows surface support
- **SVG Surfaces** — Rendering SVG documents
- **Quartz Surfaces** — Rendering to Quartz surfaces
- **XCB Surfaces** — X Window System rendering using the XCB library
- **XLib Surfaces** — X Window System rendering using XLib
- **XLib-XRender Backend** — X Window System rendering using XLib and the X Render extension
- **Script Surfaces** — Rendering to replayable scripts

Utilities

- **cairo_matrix_t** — Generic matrix operations
- **Error handling** — Decoding cairo's status
- **Version Information** — Compile-time and run-time version checks.
- **Types** — Generic data types

.

Appendix B

PCI Express Multifunction I/O Board Instruction Manual

Model 826 | Rev.3.0.5 | November 2013

SENSORAY | embedded electronics

Designed and manufactured in the U.S.A.

SENSORAY | p. 503.684.8005 | email: info@SENSORAY.com | www.SENSORAY.com

7313 SW Tech Center Drive | Portland, OR 97203

Table of Contents

Chapter 1: Preliminary

1.1 Limited Warranty

Sensoray Company, Incorporated (Sensoray) warrants the Model 826 hardware to be free from defects in material and workmanship and perform to applicable published Sensoray specifications for two years from the date of shipment to purchaser. Sensoray will, at its option, repair or replace equipment that proves to be defective during the warranty period. This warranty includes parts and labor.

The warranty provided herein does not cover equipment subjected to abuse, misuse, accident, alteration, neglect, or unauthorized repair or installation. Sensoray shall have the right of final determination as to the existence and cause of defect.

As for items repaired or replaced under warranty, the warranty shall continue in effect for the remainder of the original warranty period, or for ninety days following date of shipment by Sensoray of the repaired or replaced part, whichever period is longer.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. Sensoray will pay the shipping costs of returning to the owner parts that are covered by warranty. A restocking charge of 25% of the product purchase price will be charged for returning a product to stock.

Sensoray believes that the information in this manual is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, Sensoray reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition.

The reader should consult Sensoray if errors are suspected. In no event shall Sensoray be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, SENSORAY MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF SENSORAY SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. SENSORAY WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

Third party brands, names and trademarks are the property of their respective owners.

1.2 Handling Instructions

The Model 826 circuit board contains electronic circuitry that is sensitive to Electrostatic Discharge (ESD).

Special care should be taken in handling, transporting, and installing circuit board to prevent ESD damage to the board. In particular:

- Do not remove the circuit board from its protective packaging until you are ready to install it.
- Handle the circuit board only at grounded, ESD protected stations.
- Remove power from the equipment before installing or removing the circuit board. In particular, disconnect all field wiring from the board before installing or removing the board from the backplane.

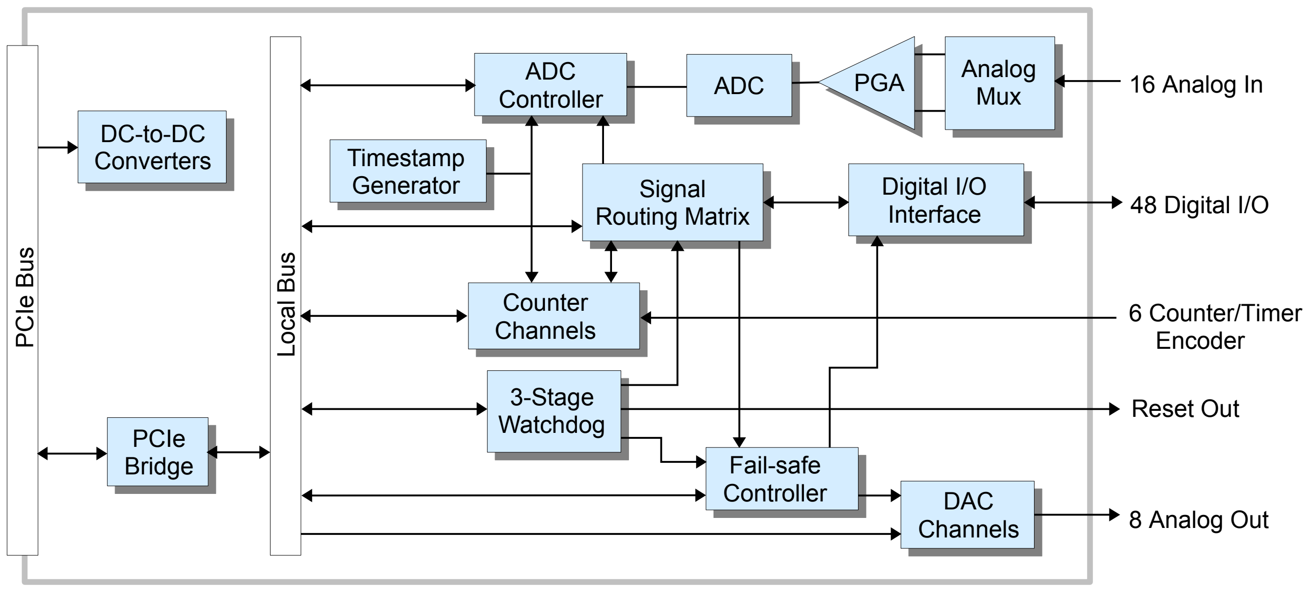
Chapter 2: Introduction

2.1 Overview

Model 826 is a PCI Express board that features an assortment of I/O interfaces commonly used in measurement and control applications:

- **48 bidirectional digital I/O channels** with edge detection and fail-safe output control.
- **Eight 16-bit analog outputs** (±10V, ±5V, 0-10V, 0-5V) with fail-safe output control.
- **Sixteen 16-bit differential analog inputs** (±10V, ±5V, ±2V, ±1V) with hardware and software triggering.
- **Six 32-bit counters**. Clock inputs may be driven from external quadrature-encoded (e.g., incremental encoder) or single-phase sources. Inputs accept differential RS-422 or standard TTL/CMOS single-ended signals. Auxiliary inputs and outputs can be internally routed to digital I/O channels.
- **Multistage watchdog timer** that can activate fail-safe outputs and generate service requests.
- **Fail-safe controller** switches outputs to safe states upon watchdog timeout or external trigger.
- **Signal routing matrix** allows software to interconnect interfaces without external wiring.

Figure 1: Model 826 block diagram



Sensoray provides a free, comprehensive API (application programming interface) to facilitate the rapid development of polled, event-driven, or mixed-mode programs for Model 826. The API works in concert with the board's advanced architecture to support complex, high performance applications.

Standard headers are provided for connecting on-board peripherals to external circuitry. The board's low-profile headers allow it to fit comfortably into high-density systems, and an integral cable clamp keeps cables secure and organized.

On-board LEDs provide visual indications of the board's condition. The PWR indicator is lit when the on-board power supplies are operating. Six LEDs are used to indicate counter clock activity as explained in Status LEDs.

2.1.1 Timestamp Generator

A timestamp generator is shared by the analog input system and counter channels. The timestamp generator is a free-running 32-bit counter that increments every microsecond. The generator starts at zero counts upon board reset and overflows every 2^{32} microseconds (approximately 71.6 minutes). The timestamp counter is not writable, but the current timestamp counts can be read by calling the S826_TimestampRead function.

2.1.2 Board Reset

Upon power-up, or in response to a hardware reset, all interfaces and outputs are forced to default initial states. The initial conditions of the interfaces are discussed in the interface chapters.

2.2 Hardware Configuration

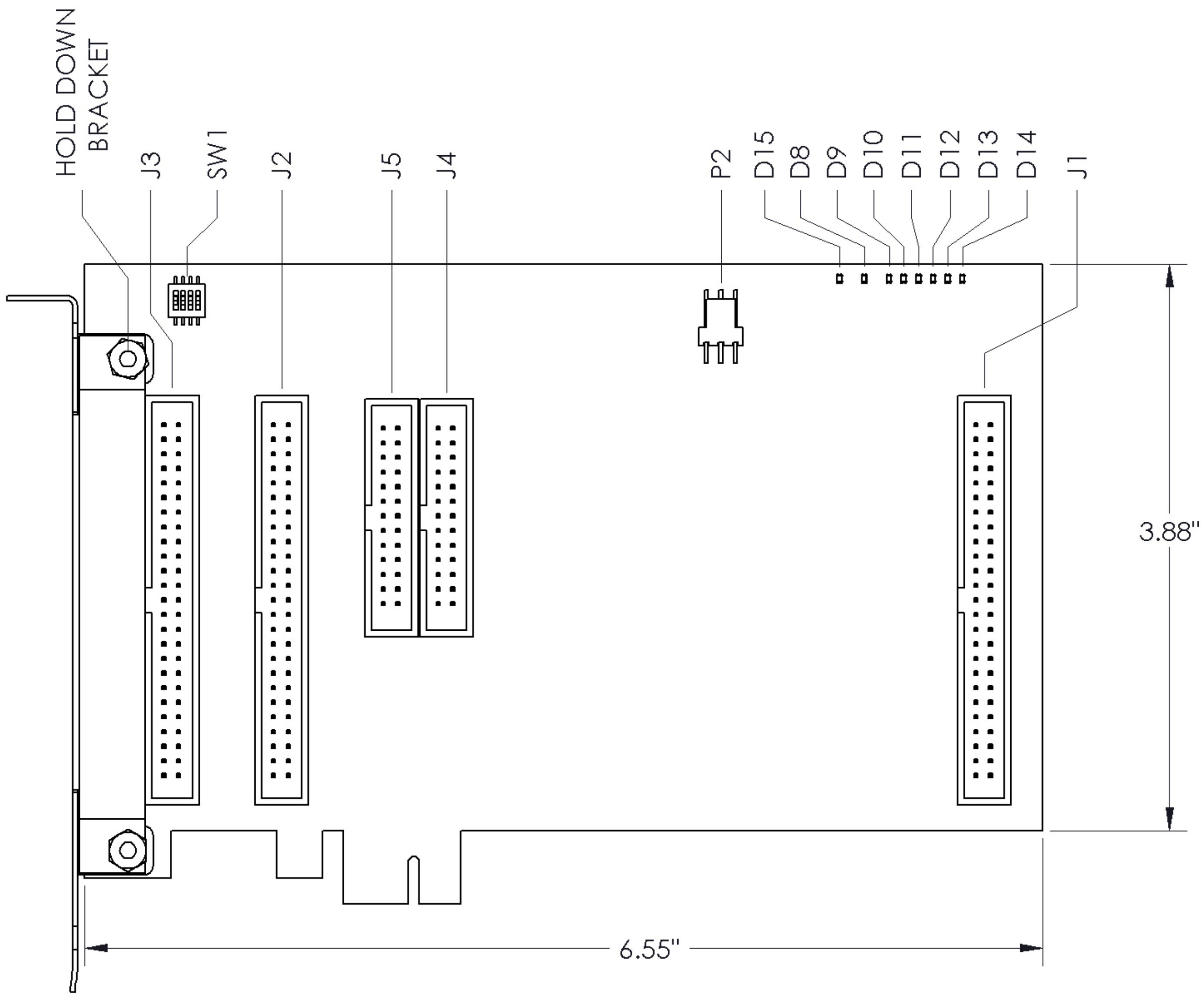
Up to sixteen model 826 boards can coexist in a single host computer. When more than one board is used in a system, each board must be configured before installation; this is done by setting quad dip switch SW1 to assign a unique identifier (board number) in the range 0 to 15 to the board.

By default, all model 826 boards are factory configured as board number 0. No reconfiguration is necessary if there is only one 826 board in the host computer; simply leave it configured at its default board number.

If more than one 826 board will be plugged into a common backplane, the boards must be configured so that each board has a unique board number. Board numbers are typically contiguous, though this is not required. For example, board numbers 0 and 3 could be assigned in a two-board system, though it would be more common to assign board numbers 0 and 1. This table shows the relationship between board number and switch settings:

Board Number	SW1-4	SW1-3	SW1-2	SW1-1	Notes
0	OFF	OFF	OFF	OFF	Factory default
1	OFF	OFF	OFF	ON	
2	OFF	OFF	ON	OFF	
3	OFF	OFF	ON	ON	
4	OFF	ON	OFF	OFF	
5	OFF	ON	OFF	ON	
6	OFF	ON	ON	OFF	
7	OFF	ON	ON	ON	
8	ON	OFF	OFF	OFF	
9	ON	OFF	OFF	ON	
10	ON	OFF	ON	OFF	
11	ON	OFF	ON	ON	
12	ON	ON	OFF	OFF	
13	ON	ON	OFF	ON	
14	ON	ON	ON	OFF	
15	ON	ON	ON	ON	

2.3 Board Layout



2.4 Cable Installation

The 826 board should be connected to external circuitry with Sensoray cables, model 826C1 (26 conductor, for counters) and 826C2 (50 conductor, for analog and digital I/O), which are specifically designed for this purpose. These cables feature thin, flat cable and low profile headers that allow the board to fit into high-density systems when loaded with a complete complement of five cables.

To install the cables (see above diagram):

- Loosen and remove the board's cable clamp (part of the hold-down bracket assembly).
- Pass each cable's low-profile end through the hold-down bracket (from left of bracket) and plug it into its connector.
- Install and tighten the cable clamp.

Chapter 3: Programming

A software developers kit (SDK) for Model 826 is available for download from Sensoray's web site. The SDK includes Linux and Windows device drivers, sample application programs, and an application programming interface (API) that is supplied as a both a Windows dynamic link library (DLL) and a Linux static library. The API core is available in source code form to OEMs who need to port the API to other operating systems.

This chapter provides an overview of the API and discusses functions that are used in all applications. Later chapters discuss API functions that are specific to the board's I/O systems.

3.1 Thread Safety

API functions are thread and process safe when they are not used to access shared hardware resources. For example, consider the case of an API function that is used to control general-purpose digital I/O (DIO) outputs where two threads or processes can call the function simultaneously. The function is guaranteed to be thread and process safe if those threads or processes interact with mutually exclusive DIOs. However, if the threads or processes attempt to manipulate the same DIO, that DIO will be a shared resource and the function is not guaranteed to be thread/process safe.

3.1.1 Atomic Read-Modify-Write

To facilitate high-performance thread-safe behavior, many of the API functions include a "mode" argument that specifies how a register write operation is to be performed. The mode argument works in conjunction with a bit-set and bit-clear hardware function that is implemented on many of the board's interface control registers.

mode	Operation
`0`	Write all data bits unconditionally. All register bits are programmed to explicit values.
`1`	Clear (program to '0') all register bits that have '1' in the corresponding data bit. All other register bits are unaffected.
`2`	Set (program to '1') all register bits that have '1' in the corresponding data bit. All other register bits are unaffected.

For example, if mode=2 and the data value is 0x00000003, register bits 0 and 1 will be set to '1' and all other register bits will retain their previous values.

3.2 Event-Driven Applications

Event-driven applications can be implemented by calling the API's blocking functions: S826_CounterSnapshotRead, S826_AdcRead, S826_DioCapRead, and S826_WatchdogEventWait. These functions can be used to block the calling thread while it waits for hardware signal events. Blocking functions will return immediately (without blocking) if the event occurs before the function is called.

All of the blocking functions include a "tmax" argument that specifies how long, in microseconds, to wait for events:

tmax	Blocking function behavior
`0`	Return immediately even if event has not occurred (never blocks).
`1 to 4294967294`	Block until event occurs or tmax microseconds elapse, whichever comes first. This corresponds to times ranging from 1 microsecond to approximately 71.6 minutes.
`S826_WAIT_INFINITE`	Block until event occurs, with no time limit.

Note that non-realtime operating systems (such as Windows) may not respond to events with deterministic timing. The responsiveness of such systems can depend on a number of factors including CPU loading, thread and process priorities, memory capacity and architecture, core architecture and count, and clock frequency.

3.3 Error Codes

Most of the API functions return an error code. These functions return zero if no errors are detected, otherwise a negative value will be returned that indicates the type of error that occurred.

Error Code	Value	Description
`S826_ERR_OK`	`0`	No errors.
`S826_ERR_BOARD`	`-1`	Invalid board number.
`S826_ERR_VALUE`	`-2`	Illegal argument value.
`S826_ERR_NOTREADY`	`-3`	Device was busy or unavailable, or wait timed out.
`S826_ERR_CANCELLED`	`-4`	Blocking function was canceled.
`S826_ERR_DRIVER`	`-5`	Driver call failed.
`S826_ERR_MISSEDTRIG`	`-6`	ADC trigger occurred while previous conversion burst was in progress.
`S826_ERR_DUPADDR`	`-9`	Two 826 boards are set to the same board number. Change DIP switch settings.
`S826_ERR_BOARDCLOSED`	`-10`	Addressed board is not open.
`S826_ERR_CREATEMUTEX`	`-11`	Failed to create internal mutex.
`S826_ERR_MEMORYMAP`	`-12`	Failed to map board into memory space.
`S826_ERR_MALLOC`	`-13`	Failed to allocate memory.
`S826_ERR_FIFOOVERFLOW`	`-15`	Counter channel's snapshot FIFO overflowed.
`S826_ERR_OSSPECIFIC`	`-1xx`	Error specific to the operating system. Contact Sensoray.

3.4 Open/Close Functions

3.4.1 S826_SystemOpen

The S826_SystemOpen function detects all Model 826 boards and enables communication with the boards.

```
int S826_SystemOpen(void);
```

Return Values

If the function succeeds, the return value is a set of bit flags indicating all detected 826 boards. Each bit position corresponds to the board number programmed onto a board's switches (see “Hardware Configuration“). For example, bit 0 will be set if an 826 with board number 0 is detected. The return value will be zero if no boards are detected, or positive if one or more boards are detected without error.

If the function fails, the return value is an error code (always a negative value). S826_ERR_DUPADDR will be returned if two boards are detected that have the same board number.

Remarks

This function must be called by a process before interacting with any 826 boards. The function allocates system resources that are used internally by other API functions. S826_SystemClose must be called once, for each call to S826_SystemOpen, to free the resources when they are no longer needed (e.g., when the 826 application program terminates). The board's circuitry is not reset by this function; all registers and I/O states are preserved.

S826_SystemOpen should be called once by every process that will use the 826 API. In a multi-threaded process, call the function once before any other API functions are called; all threads belonging to the process will then have access to all 826 boards in the system.

3.4.2 S826_SystemClose

The S826_SystemClose function frees all of the system resources that were allocated by S826_SystemOpen and disables communication with all 826 boards.

```
int S826_SystemClose();
```

Return Values

The return value is always zero.

Remarks

This function should be called when a process (e.g., an application program) has finished interacting with the board's I/O interfaces, to free system resources that are no longer needed. Any API functions that are blocking will return immediately, with return value S826_ERR_SYSCLOSED. The board's circuitry is not reset by this function; all registers and I/O states are preserved.

3.5 Status Functions

3.5.1 S826_VersionRead

The S826_VersionRead function returns API, driver, and board version information.

```
int S826_VersionRead(
  uint board,       // board identifier
  uint *api,        // API version
  uint *driver,     // driver version
  uint *bdrev,      // circuit board version
  uint *fpgarev     // FPGA version
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

api
Pointer to a buffer that will receive the API version info. The hexadecimal value is formatted as 0xXXYYZZZZ, where XX = major version, YY = minor version, and ZZZZ = build.

driver
Pointer to a buffer that will receive the driver version info. The hexadecimal info is formatted as 0xXXYYZZZZ, where XX = major version, YY = minor version, and ZZZZ = build.

boardrev
Pointer to a buffer that will receive the version number of the 826's circuit board.

fpgarev
Pointer to a buffer that will receive the board's FPGA version info. The hexadecimal info is formatted as 0xXXYYZZZZ, where XX = major version, YY = minor version, and ZZZZ = build.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

3.5.2 S826_TimestampRead

The S826_TimestampRead function returns the current value of the timestamp generator.

```
int S826_TimestampRead(
  uint board,       // board identifier
  uint *timestamp   // current timestamp
);
```

Parameters

board

826 board number. This must match the settings of the board's dip switches as described in section 2.2.

timestamp

Pointer to buffer that will receive the timestamp. The returned value is the contents of the timestamp generator at the moment the function executes.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function can be used to monitor elapsed times with microsecond level resolution, independent of the type of operating system being used.

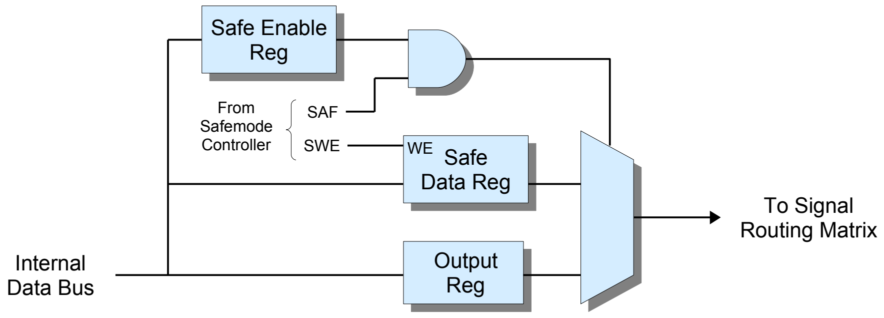
Chapter 4: Virtual Outputs

4.1 Introduction

The board has six virtual digital output channels, numbered 0 to 5, that can be used by software to signal various interfaces on the board. The virtual output channels are physical circuits that are architecturally similar to the board's general-purpose digital output channels, but they cannot be routed to the board's headers or connected to external circuitry.

Each virtual output channel is connected to the board's internal signal routing matrix, which in turn routes the channel's output signal to other on-board interfaces under program control. A virtual output channel can be routed to the ExtIn input of one or more counter channels, or to the ADC trigger input, or to combinations of these.

Figure 2: Virtual output channel (1 of 6)



4.1.1 Safemode

Safemode is activated when the SAF signal (see Figure 2) is asserted. When operating in safemode, the virtual output state is determined by the Safe Enable and Safe Data registers: when Safe Enable equals '1' the pin will be driven to the fail-safe value stored in Safe Data; when Safe Enable equals '0' the pin will output the normal runmode signal from its output register.

Upon board reset, the Safe Enable register is set to '1' so that the virtual output will exhibit fail-safe behavior by default (i.e., it will output the contents of the Safe Data register when SAF is asserted). Fail-safe operation can be disabled for a virtual output channel by programming its Safe Enable register to '0'.

The Safe Data register is typically programmed once (or left at its default value) by the application when it starts, though it may be changed at any time if required by the application. It can be written to only when the SWE bit is set to '1'. See “Safemode Controller” for more information about safemode.

4.2 Programming

The virtual output API functions employ individual bits to convey information about the virtual output channels, wherein each bit represents the information for one channel. The information is organized into a single quadlet as follows (e.g., “v5” = virtual output channel 5):

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIO	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	v5	v4	v3	v2	v1	v0

4.2.1 S826_VirtualWrite

The S826_VirtualWrite function programs the virtual output registers.

```
int S826_VirtualWrite(
  uint board,     // board identifier
  uint data,      // data to write
  uint mode       // 0=write, 1=clear bits, 2=set bits
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

data
Normal (runmode) output data for the six virtual channels (see Section 4.2).

mode
Write mode for data: 0 = write, 1 = clear bits, 2 = set bits (see “Atomic Read-Modify-Write”).

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

In mode zero, this function will unconditionally write new values to all virtual output registers. In modes one and two, the function will selectively clear or set any arbitrary combination of output registers; data bits that contain logic '1' indicate virtual output registers that are to be modified, while all other output registers will remain unchanged. Modes one and two can be used to ensure thread-safe operation as they atomically set or clear the specified bits.

4.2.2 S826_VirtualRead

The S826_VirtualRead function reads the programmed states of all virtual output registers.

```
int S826_VirtualRead(
  uint board,    // board identifier
  uint *data     // pointer to data buffer
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

data
Pointer to a buffer (see Section 4.2) that will receive the output register states.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function returns the output register states.

4.2.3 S826_VirtualSafeWrite

The S826_VirtualSafeWrite function programs the virtual output channel Safe registers.

```
int S826_VirtualSafeWrite(
  uint board,     // board identifier
  uint data,      // safemode data
  uint mode       // 0=write, 1=clear bits, 2=set bits
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

data
Pointer to data array (see Section 4.2) to be programmed into the Safe registers.

mode
Write mode for data: 0 = write, 1 = clear bits, 2 = set bits (see "Atomic Read-Modify-Write").

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function should only be called when the SWE bit is set (see S826_SafeWrenWrite). The function will fail without notification (return S826_ERR_OK) if SWE=0 (see Section 10.1.1).

4.2.4 S826_VirtualSafeRead

The S826_VirtualSafeRead function returns the contents of the virtual output channel Safe registers.

```
int S826_VirtualSafeRead(
  uint board,    // board identifier
  uint *data     // pointer to data buffer
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

data
Pointer to a buffer (see Section 4.2) that will receive the Safe register states.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

4.2.5 S826_VirtualSafeEnablesWrite

The S826_VirtualSafeEnablesWrite function programs the virtual output channel Safe Enable registers.

```
int S826_VirtualSafeEnablesWrite(
  uint board,       // board identifier
  uint enables      // safemode enables
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

enables
Pointer to array of values (see Section 4.2) to be programmed into the Safe Enable registers.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function should only be called when the SWE bit is set (see S826_SafeWrenWrite). The function will fail without notification (return S826_ERR_OK) if SWE=0 (see Section 10.1.1).

4.2.6 S826_VirtualSafeEnablesRead

The S826_VirtualSafeEnablesRead function returns the contents of the virtual output channel Safe Enable registers.

```
int S826_VirtualSafeEnablesRead(
  uint board,      // board identifier
  uint *enables    // pointer to data buffer
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

enables
Pointer to a buffer (see Section 4.2) that will receive the Safe Enable register contents.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

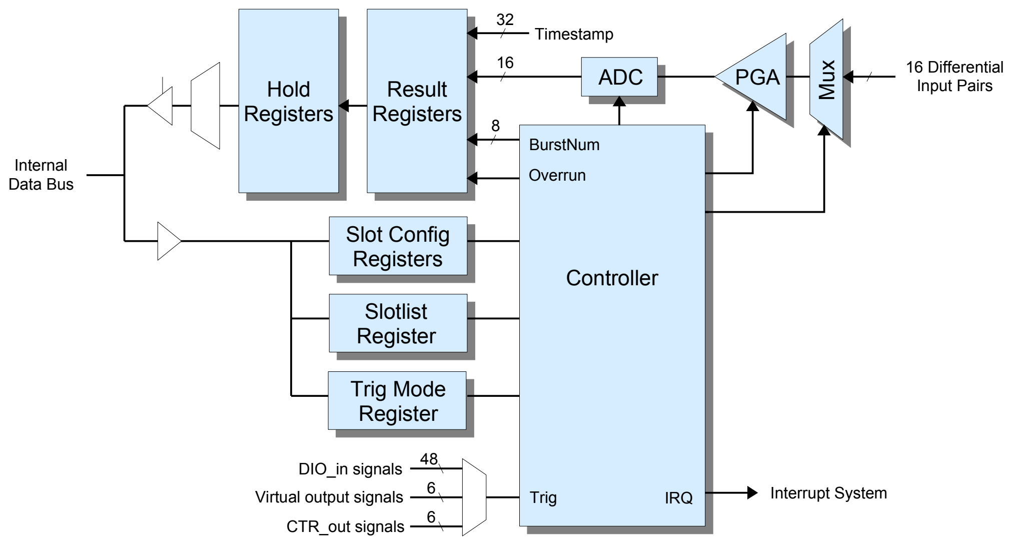
Chapter 5: Analog Inputs

5.1 Introduction

The board's analog input system consists of the following major elements:

- **Analog multiplexer (Mux)** - selects one of 16 differential input pairs (channels) for conversion.
- **Programmable gain amplifier (PGA)** - applies voltage gain of 1, 2, 5, or 10 to the selected input.
- **Analog-to-digital converter (ADC)** - performs an analog-to-digital conversion in three microseconds or less.
- **Controller** - manages operation of the analog input system.

Figure 3: Analog input system



Analog-to-digital (A/D) conversions are performed in bursts of up to 16 conversions. Each conversion takes place in a timeslot (called a *slot*), which is a reserved time interval within a burst. During a burst, slots are processed in numerical order beginning with slot 0 and ending with slot 15.

Every slot has three attributes that are programmed into its slot configuration register via the S826_AdcSlotConfigWrite function:

- **Analog channel number:** designates the analog input channel that will be digitized when the slot is processed. Any of the 16 channels may be assigned to any slot. A channel may be assigned to two or more slots if desired.
- **PGA gain:** specifies the analog gain to apply to the analog input signal.
- **Settling time:** specifies the delay from analog multiplexer switching to start of digitization.

The slotlist register designates each slot as either active or inactive; this is programmed via the S826_AdcSlotlistWrite function. When a burst occurs, each slot is processed according to the slotlist. An active slot is processed by performing one A/D conversion; its total time is the sum of its settling time plus a fixed conversion time. Inactive slots are skipped and consume no time during a burst (the slot configuration register is ignored for an inactive slot). The total processing time of each slot is configurable, from zero (inactive slot) to approximately 335 milliseconds, in one-microsecond increments.

ADC conversions are disabled upon board reset. Typically, an application program will configure the ADC system (by programming slotlist and slot configuration registers) before enabling conversions, though this is not required. The slotlist and slot configuration registers may be reprogrammed at any time, even when conversions are enabled.

5.1.1 Triggering

The trigger mode register determines which of two triggering modes, *triggered* or *continuous*, will be used to initiate conversion bursts. In triggered mode each burst must be initiated by a trigger signal, whereas in continuous mode the trigger signal is ignored and conversion bursts will automatically execute one after another with no idle time between bursts. The trigger mode and trigger signal source are programmed by calling S826_AdcTrigModeWrite.

When operating in triggered mode, S826_AdcTrigModeWrite selects one of the following signals to serve as the trigger:

- Any of the 48 general purpose digital I/O (DIO) channels. This enables an external digital signal to trigger bursts. When a DIO channel is used to trigger A/D conversions, the timing of the trigger signal is constrained by the DIO subsystem. See DIO “Pin Timing” for more information.
- Any of the six counter ExtOut signals. This enables a counter to periodically trigger ADC bursts. The selected counter output is internally routed to the ADC trigger input; no external wiring is required.
- Any of the six virtual digital outputs. This enables software to trigger ADC bursts by writing to a virtual output. See Virtual Outputs for more information. The selected virtual output is internally routed to the ADC trigger input; no external wiring is required.

5.1.2 Burst Counter

The controller maintains an 8-bit burst counter that indicates the number of conversion bursts since the ADC was enabled. The burst counter is zeroed upon board reset and whenever the ADC is disabled. The counter increments at the end of each burst and overflows to zero when incrementing from 255. The burst count is passed to the host along with every ADC sample so that the program can determine which burst a sample belongs to.

In triggered mode, the burst count can be used to keep track of the number of received triggers. If a trigger occurs while a burst is in progress, a MissedTrigger flag is set and the trigger will be ignored. After this happens, the burst count will no longer accurately indicate the number of received triggers (accuracy can be restored by disabling and then re-enabling ADC conversions).

5.1.3 Result Registers

Each slot has a result register that stores the slot's most recently acquired result, which is a set of four values: ADC output data, timestamp (which indicates the time the result was acquired), burst count, and overrun flag. Each slot also has a hold register that caches a result while it is being read by the host computer. The hold register ensures that the result's four component values will remain correlated if a new result is captured while the previous result is being read.

During a burst, the controller processes each slot by performing a sequence of operations. First it switches the analog multiplexer to the desired differential input pair and programs the gain and then, if the slot has a non-zero settling time, it waits for the settling time to elapse. When the settling time has elapsed, the controller starts an analog-to-digital conversion. At the moment the conversion completes, the four component values that comprise the result are simultaneously sampled and copied to the result register.

A new result will always overwrite the previous result, even if the previous result has not been read. If the previous result has not yet been read when a new result is written, the overrun flag will be set to '1'; otherwise it will be set to '0'.

Sixteen hardware status flags (one per slot) indicate unread results. A status flag is set when the controller writes a new result to the associated result register, and reset when the program reads the result. Results are read by calling the S826_AdcRead function. The S826_AdcStatusRead function can be called to check the status flags without returning results or altering status flags.

5.2 Connector J1

All analog input and output signals are available at connector J1.

J1 Pinout

Pin	Name	Function
1	GND	Power supply common
2	GND	Power supply common
3	-AIN0	Analog input (-) channel 0
4	+AIN0	Analog input (+) channel 0
5	-AIN1	Analog input (-) channel 1
6	+AIN1	Analog input (+) channel 1
7	-AIN2	Analog input (-) channel 2
8	+AIN2	Analog input (+) channel 2
9	-AIN3	Analog input (-) channel 3
10	+AIN3	Analog input (+) channel 3
11	-AIN4	Analog input (-) channel 4
12	+AIN4	Analog input (+) channel 4
13	-AIN5	Analog input (-) channel 5
14	+AIN5	Analog input (+) channel 5
15	-AIN6	Analog input (-) channel 6
16	+AIN6	Analog input (+) channel 6
17	-AIN7	Analog input (-) channel 7
18	+AIN7	Analog input (+) channel 7
19	GND	Power supply common
20	GND	Power supply common
21	-AIN8	Analog input (-) channel 8
22	+AIN8	Analog input (+) channel 8
23	-AIN9	Analog input (-) channel 9
24	+AIN9	Analog input (+) channel 9
25	-AIN10	Analog input (-) channel 10

Pin	Name	Function
26	+AIN10	Analog input (+) channel 10
27	-AIN11	Analog input (-) channel 11
28	+AIN11	Analog input (+) channel 11
29	-AIN12	Analog input (-) channel 12
30	+AIN12	Analog input (+) channel 12
31	-AIN13	Analog input (-) channel 13
32	+AIN13	Analog input (+) channel 13
33	-AIN14	Analog input (-) channel 14
34	+AIN14	Analog input (+) channel 14
35	-AIN15	Analog input (-) channel 15
36	SH15	Analog input (+) channel 15
37	GND	Power supply common
38	GND	Power supply common
39	GND	Power supply common
40	GND	Power supply common
41	AOUT4	Analog output channel 4
42	AOUT0	Analog output channel 0
43	AOUT5	Analog output channel 5
44	AOUT1	Analog output channel 1
45	AOUT6	Analog output channel 6
46	AOUT2	Analog output channel 2
47	AOUT7	Analog output channel 7
48	AOUT3	Analog output channel 3
49	GND	Power supply common
50	GND	Power supply common

5.3 Programming

5.3.1 S826_AdcSlotConfigWrite

The S826_AdcSlotConfigWrite function configures a timeslot.

```
int S826_AdcSlotConfigWrite(
  uint board,    // board identifier
  uint slot,     // timeslot number
  uint chan,     // analog input channel number
  uint tsettle,  // settling time in microseconds
  uint range     // input range code
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

slot
Timeslot number in the range 0 to 15.

chan
Analog input channel number in the range 0 to 15.

tsettle
Microseconds to allow the analog input to settle before conversion, in the range 0 to 335,544.

range
Enumerated value that specifies the analog input voltage range. This determines the analog gain that will be applied to the input signal before digitization.

range	PGA Gain	Analog Input Range	Notes
0	x1	-10V to +10V	Default upon reset
1	x2	-5V to +5V	
2	x5	-2V to +2V	
3	x10	-1V to +1V	

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

The settings established by this function will be used the next time the slot is converted and remain in effect until changed by another call to this function or a board reset.

The maximum total time for each slot is *tsettle*+3 µs. Sufficient settling time must be allowed when different channels are being digitized so that each input signal has time to stabilize before digitization. If a single channel is being digitized in multiple, consecutive slots, only the first of its slots must allow for settling time; subsequent slots may have zero settling time because the channel will already be settled. Similarly, if only one channel is being digitized (even if it is being digitized multiple times in different slots), all settling times may be set to zero because no channel switching will occur.

The ADC data latency is greater than the slot time because an additional 1µs is needed to fetch the digitized data after each conversion. The data latency is only incurred once per burst, because the ADC controller always fetches data from the previous conversion while the next conversion is in progress. Consequently, the elapsed time from hardware trigger to burst completion (in microseconds) is

$$t_{burst} \leq 1 + 3 \cdot NumTimeslots + \sum t_{settle}$$

5.3.2 S826_AdcSlotConfigRead

The S826_AdcSlotConfigRead function returns the configuration of a timeslot.

```
int S826_AdcSlotConfigRead(
  uint board,     // board identifier
  uint slot,      // timeslot number
  uint *chan,     // analog input channel number
  uint *tsettle,  // settling time in microseconds
  uint *range     // input range code
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

slot
Timeslot number in the range 0 to 15.

chan

Buffer that will receive the analog input channel number.

tsettle

Buffer that will receive the analog settling time in microseconds.

range

Buffer that will receive the analog input voltage range code, as specified in S826_AdcSlotConfigWrite.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function returns the settings established by a board reset or previous call to S826_AdcSlotConfigWrite.

5.3.3 S826_AdcSlotlistWrite

The S826_AdcSlotlistWrite function programs the conversion slot list.

```
int S826_AdcSlotlistWrite(
  uint board,      // board identifier
  uint slotlist,   // conversion slot list
  uint mode        // write mode
);
```

Parameters

board

826 board number. This must match the settings of the board's dip switches as described in section 2.2.

slotlist

List of slots to be digitized during subsequent conversion bursts, one bit per slot. Bits 0 to 15 correspond to slots 0 to 15, respectively. Each bit that is set to logic '1' will cause the corresponding slot to be digitized, while '0' will cause the slot to be skipped.

mode

Write mode for slotlist: 0 = write, 1 = clear bits, 2 = set bits (see “Atomic Read-Modify-Write”).

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

The settings established by this function will be used the next time a slot is digitized and remain in effect until changed by another call to this function or a board reset.

In mode zero, this function will unconditionally write a new slotlist. In modes one and two, the function will selectively set or clear any arbitrary combination of slots; slotlist bits that contain logic '1' indicate slots that are to be modified, while all other slots will remain unchanged. Modes one and two can be used to guarantee thread-safe operation as they atomically enable or disable the specified slots without affecting any other slots.

5.3.4 S826_AdcSlotlistRead

The S826_AdcSlotlistRead function returns the conversion slot list.

```
int S826_AdcSlotlistRead(
  uint board,       // board identifier
  uint *slotlist    // conversion slot list
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

slotlist
Pointer to a buffer that will receive the slotlist. In the received value, bits 0 to 15 correspond to slots 0 to 15. For each bit: '1' = active (slot will be digitized during bursts), '0' = inactive (slot will be skipped during bursts).

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function returns the current slotlist, which was previously programmed by S826_AdcSlotlistWrite or cleared by a board reset.

5.3.5 S826_AdcTrigModeWrite

The S826_AdcTrigModeWrite function programs the ADC triggering mode.

```
int S826_AdcTrigModeWrite(
  uint board,      // board identifier
  uint trigmode    // hardware trigger mode
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

trigmode
Hardware trigger configuration:

Bit	Function	Description
31-8	Reserved	Set all bits to 0.
7	Trigger enable	Set to 1 to enable hardware triggering (triggered mode), or 0 to disable hardware triggering (continuous mode).
6	Trigger polarity	1 selects rising edge; 0 selects falling edge. This is ignored if hardware triggering is disabled.
5-0	Trigger source	Hardware trigger signal source (ignored if hardware triggering is disabled): 0-47 = DIO channel 0-47. 48-53 = ExtOut signal from counter channel 0-5. 54-59 = Virtual digital output channel 0-5.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

The settings established by this function take effect upon the next conversion burst and remain in effect until changed by another call to this function or a board reset. Hardware triggering is disabled upon board reset.

5.3.6 S826_AdcTrigModeRead

The S826_AdcTrigModeRead function reads the ADC triggering mode.

```
int S826_AdcTrigModeRead(
  uint board,      // board identifier
  uint *trigmode   // hardware trigger mode
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

trigmode
Pointer to a buffer that will receive the hardware trigger configuration. See S826_AdcTrigModeWrite for the format of this value.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

5.3.7 S826_AdcEnableWrite

The S826_AdcEnableWrite function enables or disables ADC conversions.

```
int S826_AdcEnableWrite(
  uint board,    // board identifier
  uint enable    // enable conversions when true
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

enable
Set to 1 to enable, or 0 to disable ADC conversion bursts. Conversion bursts are disabled by default upon board reset.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

When enable is false, any conversion burst that is currently in progress will be terminated and all results and overrun status flags are reset. When enable is true, the resulting behavior depends on the trigger mode:

Continuous mode
The first conversion burst will begin immediately, followed by additional bursts. Each subsequent burst begins immediately after the preceding burst ends. Back-to-back bursts will continue until ADC conversions are disabled.

Triggered mode
A single conversion burst will begin in response to the next trigger and conversions will cease upon completion of that burst. Each subsequent trigger will cause another single burst. Triggers have no effect when ADC conversions are disabled.

5.3.8 S826_AdcEnableRead

The S826_AdcEnableRead function returns the enable status of the ADC system.

```
int S826_AdcEnableRead(
  uint board,    // board identifier
  uint *enable   // enable status
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

enable
Buffer that will receive the ADC system enable status: 1 = enabled, 0 = disabled.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function returns the ADC system's enable status that was previously configured by a board reset or a call to S826_AdcEnableWrite.

5.3.9 S826_AdcStatusRead

The S826_AdcStatusRead function reads the ADC conversion status.

```
int S826_AdcStatusRead(
  uint board,       // board identifier
  uint *status      // conversion status
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

status
Pointer to a buffer that will receive the conversion status for all slots. Each bit corresponds to one slot; bits 0 to 15 correspond to slots 0 to 15. A bit will be set to '1' if new (unread) data is available in the slot's result register, or '0' when the result register is empty (or has been read).

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function returns status information without altering the state of the ADC system.

5.3.10 S826_AdcRead

The S826_AdcRead function fetches ADC data from one or more timeslots.

```
int S826_AdcRead(
  uint board,       // board identifier
  int buf[16],      // pointer to adc result buffer
  uint tstamp[16],  // pointer to timestamps buffer
  uint *slotlist,   // pointer to slotlist
  uint tmax         // maximum time to wait
);
```

Parameters

board

826 board number. This must match the settings of the board's dip switches as described in section 2.2.

buf

Pointer to a buffer that will receive ADC results for the sixteen possible slots. The buffer must be large enough to accommodate sixteen values regardless of the number of active slots. Each slot is represented by a 32-bit value, which is stored at buf[SlotNumber]:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BSTNUM								V	0000000							ADCVAL															

Field	Description
BSTNUM	Burst number. This indicates the ADC burst number corresponding to the ADC data. It can be used to time-correlate the ADC data if V = '1'. The burst number is incremented at the end of each burst; it restarts at zero at the end of burst number 255.
V	Data Overwritten flag. When set to '1' this indicates the previous ADC result was overwritten by a new result before it was read.
ADCVAL	ADC data, expressed as 16-bit signed integer:

Analog Voltage	ADCVAL
-10V to +10V	0x8000 to 0x7FFF
-5V to +5V	0x8000 to 0x7FFF
-2V to +2V	0x8000 to 0x7FFF
-1V to +1V	0x8000 to 0x7FFF

tstamp

Pointer to a buffer that will receive the timestamps corresponding to ADC results. The buffer must be large enough to accommodate sixteen values regardless of the number of active slots. Each timestamp indicates the moment in time that its corresponding ADCVAL was acquired. For any given slot, the slot's timestamp will be stored in tstamp[SlotNumber]. The application may set this to NULL if timestamps are not needed.

slotlist

Pointer to a buffer containing bit flags, one bit per slot, that indicate slots of interest. Bits 0-15 correspond to slots 0-15. Before calling the function, set to '1' all bits that correspond to slots of interest. When the function returns, the buffer contents will have been changed so that '1' indicates a slot has new data in buf and '0' indicates no new data.

tmax

Maximum time, in microseconds, to wait for ADC data. See “Event-Driven Applications” for details.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function reads the result registers of an arbitrary set of slots and copies the results to buf. As many as sixteen results (one per slot) may be copied to buf each time the function is called. Over-range and under-range conditions are indicated by maximum or minimum data values for the selected input range, respectively; there are no special status flags that indicate these conditions.

Before calling the function, one or more slotlist bits must be set to indicate the slots of interest. Only new, unread results are copied to buf. If a slot is not of interest or has not been converted since it was last read, its buf value will not change. In all cases (even when the function returns an error), when the function returns, slotlist will indicate the slots of interest that have new buf values.

The function will operate in either blocking or non-blocking mode depending on the value of tmax. If tmax is zero, the function will return immediately. If tmax is greater than zero, the calling thread will block until all requested data is available or tmax elapses. In either case, the function will copy to buf all data from slots of interest that have a new result. The function will return S826_ERR_NOTREADY if any of the requested slot data is unavailable.

This function will return S826_ERR_CANCELLED if, while it is blocking, another thread calls S826_AdcWaitCancel to cancel waits on any of the slots of interest. It will return immediately if the wait criteria is completely satisfied due to the wait cancellations, otherwise it will continue to block and return when all remaining wait criteria is satisfied. S826_ERR_BOARDCLOSED will be returned immediately if S826_SystemClose executes while this function is blocking. In either case, no result data will be copied to buf.

In triggered mode, the board's internal MissedTrigger error flag will be set if a trigger occurs while an ADC burst is in progress. When the MissedTrigger flag is active, S826_AdcRead will wait for the requested slot data to arrive and then it will clear the MissedTrigger flag and return S826_ERR_MISSEDTRIG. If multiple threads are waiting in S826_AdcRead, only the first thread to return will receive S826_ERR_MISSEDTRIG; the other waiting threads will not receive this error.

Thread-safe operation is guaranteed only if the slots of interest for any given thread do not coincide with those of another thread. For example, thread safety would be assured if one thread designates slots 1 and 3-5 as slots of interest while another thread designates slots 2 and 9. Thread safety would be compromised, though, if both threads shared a common slot of interest.

5.3.11 S826_AdcWaitCancel

The S826_AdcWaitCancel function cancels a blocking wait on one or more slots.

```
int S826_AdcWaitCancel(
  uint board,      // board identifier
  uint slotlist    // slots to cancel
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

slotlist
Set of bit flags, one bit per slot, that indicate slots for which waiting is to be canceled. Bits 0-15 correspond to slots 0-15.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function cancels blocking for an arbitrary set of slots so that another thread, which is blocked by S826_AdcRead while waiting for adc data to arrive, will return immediately with S826_ERR_CANCELLED.

Chapter 6: Analog Outputs

6.1 Introduction

The 826 board has eight 16-bit, single-ended digital-to-analog converter (DAC) channels. Each channel can be independently configured to generate output voltages across one of four output voltage ranges: 0 to +5V (default upon reset), 0 to +10V, -5 to +5V, and -10 to +10V.

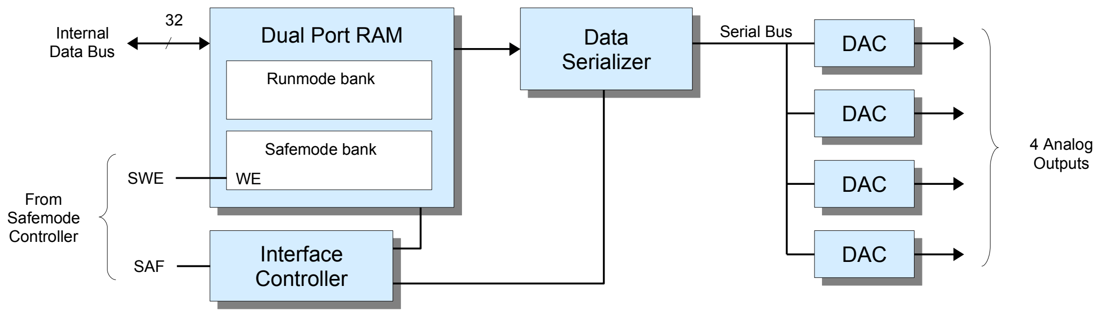
Each DAC channel has a setpoint and configuration register. A channel's output voltage is programmed by writing to its setpoint register. Before programming the setpoint, the DAC must be configured for the desired output range by writing to its configuration register.

Setpoint and configuration values are serially transmitted to the DAC devices over a high speed serial bus (Figure 4). The interface includes a dual-port RAM that allows the host to write setpoint and configuration values to the board while the serial bus is busy. If multiple untransmitted values are pending in the RAM, the controller will employ round-robin arbitration to ensure all pending values are transmitted in a fair and timely fashion. The data serializer requires 1.04 µs to transmit each setpoint or configuration value.

The board has two identical DAC interfaces as shown in Figure 4. Each interface manages a group of four DAC channels. One interface manages channels 0 to 3 and the other interface manages channels 4 to 7. The two interfaces operate concurrently, thus making it possible to simultaneously write to two DACs that reside in different four-channel groups. For example, DAC channels 0 and 4 can be written to simultaneously.

The host may write setpoint and configuration values to the board at any time. If a new value is written to the RAM for a particular channel before that channel's previously written value has been transmitted, the previous RAM value will be overwritten and only the new value will be transmitted. Consequently, the host is allowed to write setpoint data at a rate that exceeds the serial bus bandwidth, though doing so will result in dropped samples.

Figure 4: Analog output interface (1 of 2)



6.1.1 Safemode

The dual-port RAM has two memory banks, one for normal operation ("runmode") and another for "safemode" operation. The runmode bank stores the setpoint and configuration values that are used during normal operation; these values may be changed at any time as required by the application. The safemode bank contains alternate fail-safe settings that are typically programmed once during program initialization (or left at their default settings). Setpoint and configuration values can be written to the runmode bank at any time, but the safemode bank can only be written when SWE = '1'

The safemode signal (SAF) determines which RAM bank is being used by the interface controller ('1' = safemode, '0' = runmode). When SAF changes state, the appropriate RAM bank is selected and all of the DAC channels are reprogrammed to the settings stored in that bank. See "Safemode Controller" for more information about the fail-safe system.

6.1.2 Reset State

Upon reset, all DAC outputs and all channels in both RAM banks are programmed to 0V with the output range set to 0 to +5V.

6.2 Connector J1

DAC output signals are available on the analog I/O connector J1, which is shared with the ADC system. See Section 5.2 for the connector pinout.

Each DAC channel has a single-ended output signal that is referenced to the board's power supply common. The output and common signals are available on the analog I/O connector. DAC output signals are sensed at the connector; no external sense inputs are available on the connector.

6.3 Programming

6.3.1 S826_DacRangeWrite

The S826_DacRangeWrite function programs the voltage range of an analog output channel.

```
int S826_DacRangeWrite(
  uint board,    // board identifier
  uint chan,     // channel number
  uint range,    // output range
  uint safemode  // RAM bank select
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

chan
DAC channel number in the range 0 to 7.

range
Enumerated value that specifies the output voltage range:

range	Analog Output Range	Notes
0	0 to +5V	Default upon reset
1	0 to +10V	
2	-5 to +5V	
3	-10 to +10V	

safemode
Specifies the RAM bank to be written: '0' = runmode, '1' = safemode.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function configures a channel's output voltage range and programs the setpoint to zero volts. It can be called at any time, though it is typically called once per channel during program initialization for the runmode bank and, if necessary, once per channel for the safemode bank as well (if default safemode values are not suitable).

The SWE bit must be set (see S826_SafeWrenWrite) to allow writing to the safemode bank. This function will fail without notification (return S826_ERR_OK) if SWE=0 (see Section 10.1.1) when writing to the safemode bank.

6.3.2 S826_DacDataWrite

The S826_DacDataWrite function programs the output voltage of a DAC channel.

```
int S826_DacDataWrite(
  uint board,     // board identifier
  uint chan,      // channel number
  uint setpoint,  // output level
  uint safemode   // RAM bank select
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

chan
DAC channel number in the range 0 to 7.

setpoint
Analog output level. This is a value ranging from 0x0000 to 0xFFFF. The resulting output voltage depends on the channel's previously programmed output range.

Analog output range	setpoint range
0 to +5V	0x0000 to 0xFFFF
0 to +10V	0x0000 to 0xFFFF
-5V to +5V	0x0000 to 0xFFFF
-10V to +10V	0x0000 to 0xFFFF

safemode
Specifies the RAM bank to be written: '0' = runmode, '1' = safemode.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function programs a channel's runmode or safemode output voltage level. If the output range will also be changed, the range should be changed first, before calling this function. This function is frequently used to change a runmode setpoint whenever an output level change is required. It can also be used to change a safemode setpoint; this is typically done once per channel when the application starts, after programming the channel's safemode output range.

The SWE bit must be set (see S826_SafeWrenWrite) to allow writing to the safemode bank. This function will fail without notification (return S826_ERR_OK) if SWE=0 (see Section 10.1.1) when writing to the safemode bank.

6.3.3 S826_DacRead

The S826_DacRead function returns the output range and setpoint of an analog output channel.

```
int S826_DacRead(
  uint board,     // board identifier
  uint chan,      // channel number
  uint *range,    // output range
  uint *setpoint, // output level
  uint safemode   // RAM bank select
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

chan
DAC channel number in the range 0 to 7.

range
Pointer to a buffer that will receive the output range code as described in Section 5.3.1.

setpoint
Pointer to a buffer that will receive the output level as described in Section 5.3.2.

safemode
Specifies the RAM bank to be written: '0' = runmode, '1' = safemode.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function can be used to determine whether previously written configuration data has been sent to the DAC device. The value S826_ERR_NOTREADY will be returned if the range or setpoint is not yet active (i.e., one or both values have not yet been transmitted to the DAC device).

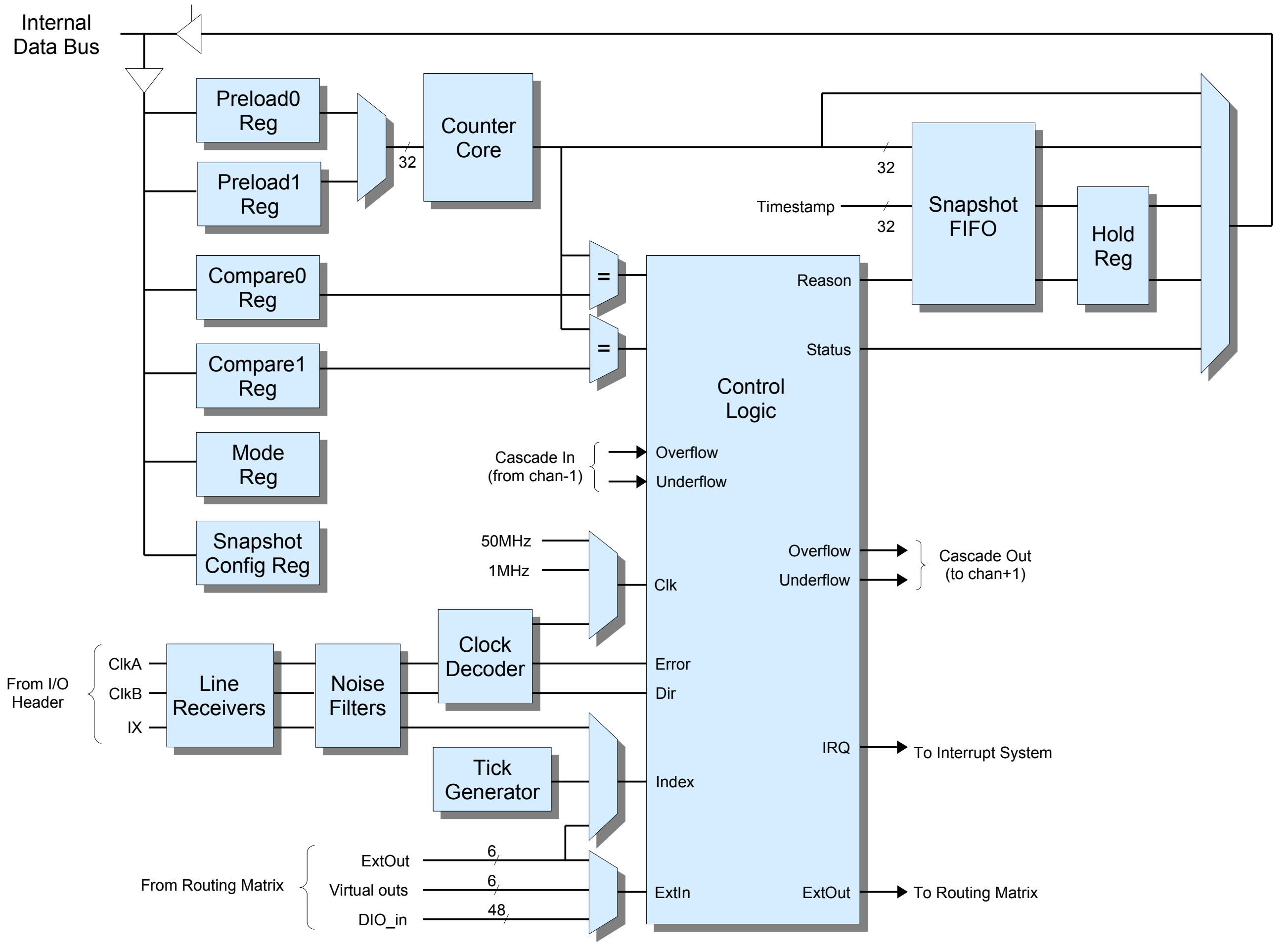
Chapter 7: Counters

7.1 Introduction

The model 826 board has six identical 32-bit programmable counter channels, numbered 0 to 5. Each counter channel can implement a complete solution for a variety of common applications, including incremental encoder interface, event counter, timer, pulse generator, PWM generator, pulse width measurement, period measurement, and frequency measurement. See “Application Connections” for tips on connecting external signals to counter channels, and “Common Applications” for programming strategies.

As shown in Figure 5, each channel can connect to as many as five external signals. Three input signals (ClkA, ClkB, and IX) are accessible through dedicated header pins. Two additional signals (input ExtIn and output ExtOut) can be routed to general-purpose digital I/O pins if physical access to these signals is needed.

Figure 5: Counter channel (1 of 6)



7.1.1 ClkA, ClkB and IX Signals

A channel can accept external signals on its ClkA and ClkB inputs consisting of either a single-phase or quadrature-encoded clocks. The maximum count rate is 25MHz regardless of external clock type. Single-phase clocks having exactly 50 percent duty rate can be counted at up to 25MHz; the maximum count frequency must be derated for other duty rates (see Specifications for details). Quadrature clocks up to 25 MHz (x1 multiplier), 12.5 MHz (x2), and 6.25 MHz (x4) are supported, with suitable derating for deviations from 90 degree clock phasing.

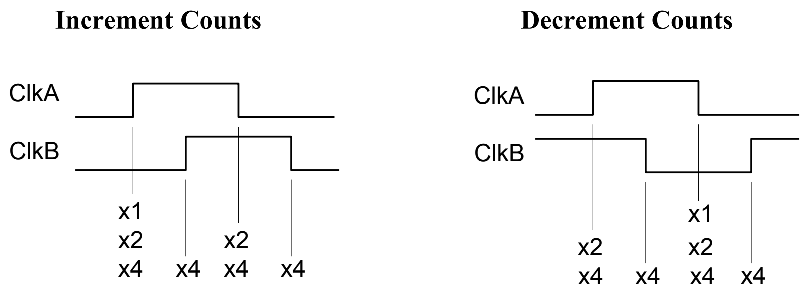
The ClkA, ClkB, and IX inputs employ differential RS-422 line receivers for buffering and noise immunity. All line receivers have built-in termination resistors. The clock and IX inputs are compatible with differential RS-422 signal pairs as well as single-ended TTL and 5V CMOS signals.

Each line receiver is followed by a noise filter that can be used to remove glitches. See S826_CounterFilterWrite for details.

7.1.2 Quadrature Decoder

When connected to quadrature-encoded clock signals, the ClkA and ClkB signals are processed by a quadrature decoder circuit to produce count direction, count enable, and quadrature error signals. Figure 6 shows the clock events that affect counting for each of the possible clock multipliers.

Figure 6: Quadrature Decoding



For example, with a x1 ("times 1") multiplier, the counts will only change when ClkB is low; the core will count up on the rising edge of ClkA and down on the falling edge of ClkA.

If the decoder detects an encoding error, it will generate a special error snapshot (see Snapshots below) and set an internal error flag. This will happen when ClkA and ClkB transition within 20 ns of each other (which should never occur in normal operation). This can be caused by various conditions, including signal noise on ClkA/ClkB or excessively high input clock frequency. The error snapshot serves as a warning that the counter core may no longer contain an accurate counts value.

7.1.3 ExtIn Signal

Some applications may require an additional external input (ExtIn) signal. If needed, this signal can be routed from any general-purpose digital I/O (DIO) channel, any virtual digital output channel, or the ExtOut signal of any counter channel (see Section 7.3.12 for details). When ExtIn is routed from a DIO channel, the signal appearing on the DIO channel's connector pin must conform to the timing constraints of the DIO subsystem as explained in "Pin Timing".

The ExtIn signal can be configured to behave as either a count or preload permissive by programming the IM field in the Mode register. See S826_CounterModeWrite for details.

7.1.4 ExtOut Signal

Some counter applications (e.g., pulse or PWM generator) may require a physical output from the counter. In such cases, the counter's ExtOut signal may be routed to a general-purpose DIO channel (see Section 8.3.12 for details). When so routed, the DIO channel will act as a dedicated counter output, while the DIO input and edge detection functions continue to operate normally.

When routed to a DIO, the ExtOut signal timing is constrained by the DIO subsystem as explained in "Pin Timing". In particular, the ExtOut signal may be delayed up to 20 ns before it appears on the DIO connector pin. Note also that short, positive output pulses may require the addition of external, supplemental pull-up resistance on the DIO pin to attain sufficiently fast rise time.

A channel's ExtOut signal will be asserted only when the channel is running. When the channel is halted, ExtOut is held at the inactive state, as defined by its configured polarity.

7.1.5 Snapshots

A "snapshot" consists of three values that are simultaneously sampled in response to a trigger: the counts contained in the counter core, the timestamp (which indicates the time the snapshot was captured), and a set of "reason" flags that indicate the type of event (or types of events, if multiple events occurred at the same time) that triggered the snapshot.

Snapshots are stored in the Snapshot FIFO. Snapshots are written to the FIFO as they occur, whereas the host may read them from the FIFO at any convenient time. The FIFO stores up to 16 snapshots. When the FIFO is full, a subsequent trigger will cause a new snapshot to be stored in the FIFO and the oldest snapshot in the FIFO will be deleted.

The Hold register caches a snapshot while it is being read by the host. The snapshot timestamp and reason code are copied to the Hold register when the snapshot counts are read. The host will then read the cached timestamp and reason code from the Hold register, thus ensuring the three snapshot values will remain correlated if a FIFO overflow occurs while the snapshot is being read.

A snapshot may be captured in response to various types of hardware and software triggers. All of the hardware trigger types can be individually enabled through the snapshot configuration register. Snapshots may be triggered when:

- The S826_CounterSnapshot function is called (i.e., a "soft" snapshot).
- The counter matches a compare register.
- Transitions occur on the Index input.
- Transitions occur on the ExtIn input.
- The counter reaches zero counts.
- A quadrature clock encoding error is detected. Once this happens, no further error-triggered snapshots can be captured until the error is cleared. The resulting error snapshot enables the application to determine the counts at the moment the error occurred. See S826_CounterSnapshotRead for further information.

Multiple counters can use the same snapshot trigger source or sources. For example, two or more counters could be configured to capture snapshots in response to a signal from a common DIO channel, thus enabling an external signal to simultaneously trigger snapshots on all affected counters. Similarly, a virtual digital output channel could be used as a common trigger, thereby allowing software to simultaneously trigger snapshots on the affected counters.

7.1.6 Preloading

The counter core can be "preloaded" (parallel-loaded) from the Preload0 and Preload1 registers in response to various hardware and software preload triggers. All of the hardware triggers can be individually enabled through the mode register. Preloads may be triggered:

- When the S826_CounterPreload function is called (i.e., a "soft" preload).
- When the channel state switches from halted to running.
- When the counter reaches zero counts.
- When the counter matches a compare register.
- When transitions occur on the Index input.
- While the Index input is held at its active level. This has the effect of holding the counter core at the preload value while Index is asserted.

The mode register's BP bit specifies whether both preload registers will be used or only Preload0. Preload0 is active (selected) by default when the channel state switches from halted to running. When a preload occurs, the core is first preloaded from the preload register and then the active register selector will change.

The preload mechanism behaves as shown in the following table. For example, if both preload registers are being used (BP=1) and a preload occurs because the counts reached zero, the core will be preloaded from the active preload register and then the other preload register will be activated.

Preload Behavior

Mode Register BP bit	Preload Trigger Type	Counter Core Loads From	Activated After Preload
0	Any	Preload0	Preload0
1	Zero Counts Reached	Active preload	Alternate preload
	Any except Zero Counts Reached	Preload0	Preload1

Preload triggers are prioritized. If two simultaneous preload triggers occur, the one with the highest priority is considered to be the cause of the preload and the preload mechanism will behave accordingly.

Some types of preload triggers are enabled and disabled by ExtIn when ExtIn is configured as a preload permissive (see “ExtIn Signal”). These triggers are disabled when ExtIn is negated and enabled with ExtIn is asserted.

Preload Trigger Type	Priority	Enabled/Disabled by ExtIn (when ExtIn is configured as preload permissive)
Channel switched to running state	7 (highest)	No
Zero counts reached	6	No
Compare1 match	5	Yes
Compare0 match	4	Yes
Index rising edge	3	Yes
Index falling edge	2	Yes
Index active level	1	Yes
Soft preload	0 (lowest)	Yes

7.1.7 Tick Generator

Each counter channel has an independent tick generator that can be used to create counting gates; this is useful for frequency measuring applications. The generator can produce periodic pulses at intervals ranging from one microsecond to ten seconds in decade steps. The channel's external IX input or the ExtOut output from any counter channel can be used in lieu of the internal tick generator if a custom gate time is needed.

7.1.8 Cascading

Limited cascading of counter channels is supported. Each channel receives overflow and underflow signals from the adjacent lower channel (channel 0 receives from channel 5). A channel may be cascaded onto the adjacent lower channel by setting K=4 in its mode register (see S826_CounterModeWrite). Note that when two channels are cascaded, only the count function is extended; the snapshot and preload mechanisms will continue to operate independently. Each cascade is limited to two counters.

7.1.9 Status LEDs

Each counter channel is associated with a status LED that indicates clock pulses are detected on that channel. The LED flashes at a constant rate while the clock signal is toggling. The LEDs are labeled E0-E5, corresponding to counter channels 0-5 respectively.

7.1.10 Reset State

Upon board reset, all counter channels are set to the “halted” state (see S826_CounterStateWrite for details). In addition, a board reset will also zero the Mode, Preload, and Compare registers.

7.2 Connectors J4/J5

Two 26-pin headers, J4 and J5, bring out connections from the board's six counter channels to external field wiring. J4 is used for counter channels 0-2 and J5 for channels 3-5.

J4 and J5 Pinouts

J4 - Encoder channels 0-2			
Pin	**Name**	**Function**	**Chan**
1	+A0	Differential A clock inputs	0
2	-A0		
3	GND	Power supply return	
4	+B0	Differential B clock inputs	
5	-B0		
6	+5V	+5V power output	
7	+I0	Differential IX inputs	
8	-I0		
9	GND	Power supply return	
10	+A1	Differential A clock inputs	1
11	-A1		
12	+5V	+5V power output	
13	+B1	Differential B clock inputs	
14	-B1		
15	GND	Power supply return	
16	+I1	Differential IX inputs	
17	-I1		
18	+5V	+5V power output	
19	+A2	Differential A clock inputs	2
20	-A2		
21	GND	Power supply return	
22	+B2	Differential B clock inputs	
23	-B2		
24	+5V	+5V power output	
25	+I2	Differential IX inputs	
26	-I2		

J5 - Encoder channels 3-5			
Pin	**Name**	**Function**	**Chan**
1	+A3	Differential A clock inputs	3
2	-A3		
3	GND	Power supply return	
4	+B3	Differential B clock inputs	
5	-B3		
6	+5V	+5V power output	
7	+I3	Differential IX inputs	
8	-I3		
9	GND	Power supply return	
10	+A4	Differential A clock inputs	4
11	-A4		
12	+5V	+5V power output	
13	+B4	Differential B clock inputs	
14	-B4		
15	GND	Power supply return	
16	+I4	Differential IX inputs	
17	-I4		
18	+5V	+5V power output	
19	+A5	Differential A clock inputs	5
20	-A5		
21	GND	Power supply return	
22	+B5	Differential B clock inputs	
23	-B5		
24	+5V	+5V power output	
25	+I5	Differential IX inputs	
26	-I5		

7.2.1 Counter Signals

Each counter channel has six input signals on its header connector: A+, A-, B+, B-, X+, and X-. The header also has power and ground pins that can be used to supply operating power to external devices such as incremental encoders. The following table details the header pins that are available for each channel and their recommended usage.

External signal connections to a counter channel

Function	Pin Name	Signal Type	Clock Source		
			Quadrature	Single-phase	Internal
ClkA	A+	RS-422	ClockA+	Clock+	NC
		TTL/CMOS	ClockA	Clock	NC
	A-	RS-422	ClockA-	Clock-	NC
		TTL/CMOS	NC	NC	NC
ClkB	B+	RS-422	ClockB+	NC	NC
		TTL/CMOS	ClockB	NC	NC
	B-	RS-422	ClockB-	NC	NC
		TTL/CMOS	NC	NC	NC
			NC		
			NC	Internal / None	
IX	X+	RS-422	NC	NC	
		TTL/CMOS	IX	NC	
	X-	RS-422	IX-	NC	
		TTL/CMOS	NC	NC	
ExtIn	Note 1		Auxiliary counter input. The behavior of this signal is configurable.		
ExtOut	Note 1		Counter output. The behavior of this signal is programmable.		
Device Power	GND	POWER	5V power supply return and ground reference for all logic signals.		
	+5V	POWER	+5VDC power. This can be used to power external devices such as incremental encoders. The total 5V current for all six counter channels is limited to 500mA.		

Notes

1. **If used, this signal must connect to a DIO channel through the board's DIO signal routing matrix. Refer to the DIO documentation for details of the routing matrix and electrical characteristics of the DIO channels.**

7.2.2 Application Connections

Typical counter connections for common applications

Application	Signals				
	ClkA	ClkB	IX	ExtIn	ExtOut
Encoder Interface	Phase A clock	Phase B clock	Snapshot trigger | Preload trigger | NC	Count enable | NC	NC
Event Counter	Event clock	NC	Snapshot trigger | Preload trigger | NC	Count enable | NC	NC
Pulse Generator	NC	NC	Pulse trigger | NC	Pulse trigger | NC	Pulse output
PWM Generator	NC	NC	Output enable | NC	NC	PWM output
Pulse Width Measurement	NC	NC	Signal to measure	NC	NC
Period Measurement	NC	NC	Signal to measure	NC	NC
Frequency Measurement	NC	NC	External time gate | NC	NC	NC

7.3 Programming

7.3.1 S826_CounterSnapshotRead

The S826_CounterSnapshotRead function reads a snapshot from a counter channel.

```
int S826_CounterSnapshotRead(
  uint board,     // board identifier
  uint chan,      // channel number
  uint *counts,   // pointer to counts buffer
  uint *tstamp,   // pointer to timestamp buffer
  uint *reason,   // pointer to reason buffer
  uint tmax       // maximum time to wait
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

chan
Counter channel number in the range 0 to 5.

counts
Pointer to a buffer that will receive the latched counts value. Set to NULL to ignore counts.

tstamp
Pointer to a buffer that will receive the timestamp. Set to NULL to ignore timestamp.

reason
Pointer to a buffer that will receive the reason flags, which indicate what caused the snapshot. When a flag bit = '1', the snapshot was caused by that event type. More than one event may have occurred at the same time, so multiple bits may be set.

bit	Description
8	Quadrature error
7	Soft snapshot (caused by calling S826_CounterSnapshot)
6	ExtIn rising edge
5	ExtIn falling edge
4	Index rising edge
3	Index falling edge
2	Zero counts reached
1	Compare1 match
0	Compare0 match

Set to NULL to ignore the reason.

tmax
Maximum time, in microseconds, to wait for data. See Event-Driven Applications for details.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function reads a previously acquired snapshot from the snapshot FIFO, consisting of the counts at a particular moment in time and the associated timestamp and reason flags. Each snapshot can be read only one time; when a snapshot is read, it is removed from the FIFO and cannot be read again.

Snapshots may be acquired in response to asynchronous events. In such cases, snapshots may occur at any time. However, if snapshots are not automatically acquired, or for some other reason it is necessary to invoke a snapshot under program control, the application program must call S826_CounterSnapshot to capture a new snapshot before calling this function to read the snapshot.

The reason flags indicate the type of event that caused the snapshot. If two or more simultaneous events would each cause a snapshot, all of the associated reason flags will be set.

Quadrature-encoded clocks are monitored for encoding errors as described in Quadrature Decoder. When an encoding error is detected, a snapshot is captured (with reason bit 8 set) and an internal error flag is set. While the error flag remains set, subsequent encoding errors will be ignored and will not trigger new snapshots (though other types of triggers will continue to cause snapshots). The error flag is automatically cleared when this function reads the error-triggered snapshot, or when the channel is halted, or when the snapshot FIFO becomes empty. The latter case ensures that the error flag will be cleared if the error-triggered snapshot is lost due to FIFO overflow.

This function can operate in either blocking or non-blocking mode, depending on the value of tmax. If tmax is zero, the function will return immediately. If tmax is greater than zero, the calling thread will block until a snapshot is available or tmax elapses. The function will return either S826_ERR_OK or S826_ERR_FIFOOVERFLOW if a snapshot was read, or S826_ERR_NOTREADY if no snapshot is available. S826_ERR_ FIFOOVERFLOW indicates that a snapshot was successfully read, but the channel's snapshot FIFO overflowed (one or more snapshots were lost); the FIFO overflow status will be automatically cleared when the function returns.

When this function is blocking, it will immediately return S826_ERR_CANCELLED if S826_CounterWaitCancel is called by another thread, or S826_ERR_BOARDCLOSED if S826_SystemClose is called. In either case, the counts, tstamp and reason values will be invalid.

7.3.2 S826_CounterWaitCancel

The S826_CounterWaitCancel function cancels a blocking wait on a counter channel.

```
int S826_CounterWaitCancel(
  uint board,    // board identifier
  uint chan      // channel number
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

chan
Counter channel number in the range 0 to 5, for which waiting is to be canceled.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function cancels blocking on a counter channel so that another thread, which is blocked by S826_CounterSnapshotRead while waiting for a snapshot, will return immediately with S826_ERR_CANCELLED.

7.3.3 S826_CounterCompareWrite

The S826_CounterCompareWrite function writes to a compare register.

```
int S826_CounterCompareWrite(
  uint board,    // board identifier
  uint chan,     // channel number
  uint regid,    // register identifier
  uint counts    // counts value
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

chan
Counter channel number in the range 0 to 5.

regid
Selects Compare register: 1 = Compare1 register, 0 = Compare0 register.

counts
Value to be written to the Compare register.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

7.3.4 S826_CounterCompareRead

The S826_CounterCompareRead function returns the contents of a compare register.

```
int S826_CounterCompareRead(
  uint board,    // board identifier
  uint chan,     // channel number
  uint regid,    // register identifier
  uint *counts   // counts value
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

chan
Counter channel number in the range 0 to 5.

regid
Selects Compare register: 1 = Compare1 register, 0 = Compare0 register.

counts
Pointer to a buffer that will receive the contents of the selected Compare register.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

7.3.5 S826_CounterSnapshot

The S826_CounterSnapshot function invokes an immediate snapshot.

```
int S826_CounterSnapshot(
  uint board,    // board identifier
  uint chan      // channel number
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

chan
Counter channel number in the range 0 to 5.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function forces a snapshot to be captured immediately. It is typically used to capture a snapshot under program control prior to reading a counter. This is one of two methods of reading the core's instantaneous counts; the other method is S826_CounterRead. This function may be called at any time when the counter channel is running.

7.3.6 S826_CounterRead

The S826_CounterRead function reads the counter core's instantaneous counts.

```
int S826_CounterRead(
  uint board,    // board identifier
  uint chan,     // channel number
  uint *counts   // counts value
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

chan
Counter channel number in the range 0 to 5.

counts
Pointer to a buffer that will receive the instantaneous counts from the counter core.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function directly reads and returns the value currently stored in the counter core. If a timestamp is also needed, the application can call this function and also call S826_TimestampRead, or it may trigger a soft snapshot and then read the snapshot.

7.3.7 S826_CounterPreloadWrite

The S826_CounterPreloadWrite function writes to a preload register.

```
int S826_CounterPreloadWrite(
  uint board,   // board identifier
  uint chan,    // channel number
  uint reg,     // register identifier
  uint counts   // counts value
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

chan
Counter channel number in the range 0 to 5.

reg
Selects the preload register to write to: 0 = Preload0 register, 1 = Preload1 register.

counts
The 32-bit value to be written to the selected Preload register.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

The counts value is immediately written to the selected preload register when this function executes. The counter core is not affected until the next core preload occurs.

7.3.8 S826_CounterPreloadRead

The S826_CounterPreloadRead function reads a preload register.

```
int S826_CounterPreloadRead(
  uint board,   // board identifier
  uint chan,    // channel number
  uint reg,     // register identifier
  uint *counts  // counts value
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

chan
Counter channel number in the range 0 to 5.

reg
Selects the preload register to write to: 0 = Preload0 register, 1 = Preload1 register.

counts
Pointer to a buffer that will receive the counts from the selected Preload register.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

7.3.9 S826_CounterPreload

The S826_CounterPreload function manually invokes a core preload from the Preload0 register.

```
int S826_CounterPreload(
  uint board,   // board identifier
  uint chan,    // channel number
  uint level,   // preload signal level
  uint sticky   // make level "sticky"
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

chan
Counter channel number in the range 0 to 5.

level
Effective preload signal level: 1 = invoke preload, 0 =don't invoke preload. This is ignored if sticky=0.

sticky
Persistence: 1 = maintain level until reprogrammed, 0 = strobe trigger (level is ignored).

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

If sticky is false, the level will be ignored. In this case, the channel's software preload command will be strobed active and then immediately return to its inactive state, thus causing Preload0 to be copied to the counter core.

If sticky is true, the specified level will be continuously applied to the channel's software preload command until changed by a subsequent call to this function. Typically, sticky will only be asserted if the counter is operating as a Pulse Generator, and only when output retriggering is enabled. When sticky and level are both '1', the counter will continuously preload from Preload0, thus causing the pulse output to go active and remain active until the function is called again to set the level to '0'; at that time, the counter will be allowed to resume counting.

7.3.10 S826_CounterStateWrite

The S826_CounterStateWrite function starts or stops channel operation.

```
int S826_CounterStateWrite(
  uint board,   // board identifier
  uint chan,    // channel number
  uint state    // channel state
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

chan
Counter channel number in the range 0 to 5.

state
Channel state: 0 = halted, 1 = running.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function is used to start and stop counter channel operation, thus placing the channel in either the “running” or “halted” state. A channel's operating mode should be configured before switching it to the running state; this is done by calling S826_CounterModeWrite. A channel will operate as specified by the mode register while it is in the running state.

In the halted state:

- Counting and preloading are not allowed.
- Counter core is reset to zero counts.
- Snapshot register is marked empty.
- Output is held inactive.
- Quadrature error is reset.

The mode, preload, compare, and snapshot configuration registers are not affected when a channel transitions between halted and running states. This function has no effect if used to start a channel that is already running, or to stop a channel that is already halted.

This function can be used to zero the counter core by calling it once to halt the channel (and zero the counts) and again to start the channel running. Another way to zero the counts is to zero to the Preload0 register (by calling S826_CounterPreloadWrite) and then transfer the preload value to the core (via S826_CounterPreload).

7.3.11 S826_CounterStatusRead

The S826_CounterStatusRead function returns information about a counter channel's status.

```
int S826_CounterStatusRead(
  uint board,    // board identifier
  uint chan,     // channel number
  uint *status   // channel status info
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

chan
Counter channel number in the range 0 to 5.

status
Pointer to a buffer that will receive the status:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	RUN	0	ST	0	0	0	0	0	0	0	0	0	0	PLS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Description
RUN	Channel enable: '1' = running, '0' = halted. This is controlled by S826_CounterStateWrite.
ST	Sticky preload command. This is controlled by S826_CounterPreload.
PLS	Preload selector. This indicates the active preload register: '1' = Preload1, '0' = Preload0. PLS can be '1' only if mode register bit BP=1 (see S826_CounterModeWrite).

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

7.3.12 S826_CounterExtInRoutingWrite

The S826_CounterExtInRoutingWrite function selects the signal source for a counter channel's ExtIn input.

```
int S826_CounterExtInRoutingWrite(
  uint board,    // board identifier
  uint chan,     // counter channel number
  uint route     // routing configuration
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

chan
Counter channel number in the range 0 to 5.

route
Signal to be connected to ExtIn (this is ignored if mode register field IM=0):
0 to 47 = DIO channel 0 to 47;
48 to 53 = counter channel 0 to 5 ExtOut;
54 to 59 = virtual digital output channel 0 to 5.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function will route the counter channel's ExtIn input to a general-purpose digital I/O (DIO) channel so that an external signal (applied to the DIO channel's I/O pin) can control the channel's ExtIn input. Alternatively, the ExtIn input may internally routed to any counter channel's ExtOut output, or to any virtual digital output channel.

7.3.13 S826_CounterExtInRoutingRead

The S826_CounterExtInRoutingRead function returns a counter channel's ExtIn signal routing configuration.

```
int S826_CounterExtInRoutingRead(
  uint board,    // board identifier
  uint chan,     // counter channel number
  uint *route    // routing configuration
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

chan
Counter channel number in the range 0 to 5.

route
Pointer to buffer that will receive the ExtIn routing configuration. The format of the returned value is identical to the route argument used in the S826_CounterExtInRoutingWrite function.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

7.3.14 S826_CounterSnapshotConfigWrite

The S826_CounterSnapshotConfigWrite function programs the snapshot configuration register.

```
int S826_CounterSnapshotConfigWrite(
  uint board,    // board identifier
  uint chan,     // counter channel number
  uint cfg,      // snapshot configuration flags
  uint mode      // 0=write, 1=clear bits, 2=set bits
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

chan
Counter channel number in the range 0 to 5.

cfg
Flags with 'E' prefix determine the types of events that will capture snapshots: '1' = enable capturing, '0' = disable capturing. Each flag with 'R' prefix determines whether the corresponding 'E' flag will be automatically cleared upon snapshot capture, thus preventing subsequent events of that type from invoking snapshots.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	RER	REF	RXR	RXF	RZ	RM1	RMO	0	0	0	0	0	0	0	0	0	EER	EEF	EXR	EXF	EZ	EM1	EM0

Flag	Snapshot trigger event
ER	ExtIn leading edge
EF	ExtIn falling edge
XR	Index leading edge
XF	Index falling edge
Z	Zero counts reached
M1	Compare1 register matches counter core
M0	Compare0 register matches counter core

mode
Write mode for cfg: 0 = write, 1 = clear bits, 2 = set bits (see “Atomic Read-Modify-Write”).

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function programs the snapshot configuration register. A snapshot will be captured when an 'E' flag is programmed to '1' and the corresponding event occurs. Upon snapshot capture, the associated 'E' flag will be automatically cleared if the corresponding 'R' flag is set.

Independent of these flags, snapshots may also be captured upon quadrature clock error or in response to calls to S826_CounterSnapshot.

7.3.15 S826_CounterSnapshotConfigRead

The S826_CounterSnapshotConfigRead function reads the snapshot configuration register.

```
int S826_CounterSnapshotConfigRead(
  uint board,    // board identifier
  uint chan,     // counter channel number
  uint *cfg      // configuration flags
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

chan
Counter channel number in the range 0 to 5.

cfg
Pointer to buffer that will receive the configuration flags. The format of the returned value is identical to the events value used in the S826_CounterSnapshotConfigWrite function. Note that 'E' flags (flags with 'E' name prefix) may be automatically reset in response to captured snapshots.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

7.3.16 S826_CounterFilterWrite

The S826_CounterFilterWrite function configures the IX, CLKA, and CLKB noise filters.

```
int S826_CounterFilterWrite(
  uint board,    // board identifier
  uint chan,     // counter channel number
  uint cfg       // filter configuration
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

chan
Counter channel number in the range 0 to 5.

cfg
Filter configuration:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IX	CK	0														T															

Field	Description
IX	Enable IX filter: '1' = enable, '0' = disable.
CK	Enable CLKA/CLKB filters: '1' = enable, '0' = disable.
T	Filter time interval specified as multiple of 20ns, common to IX, CLKA and CLKB input filters.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

Each of the IX, CLKA, and CLKB input signals has a dedicated noise filter. This function enables and disables the filters and programs the filter time interval. The CLKA and CLKB filters are enabled or disabled as a group, whereas the IX filter is independently enabled or disabled. The filter time interval, T, is common to all enabled filters.

A filter's output will not change state until its input has held a constant state for T * 20ns. The maximum T value is 65535, corresponding to a filter time of 1.3107ms. When a filter is enabled, it will delay its input signal by T * 20ns. Note that when the noise filter is enabled for the CLKA/CLKB inputs (CK = '1'), the counter's maximum clock frequency is reduced; the maximum frequency reduction is inversely proportional to T.

7.3.17 S826_CounterFilterRead

The S826_CounterFilterRead function reads the configuration of the IX, CLKA, and CLKB noise filters.

```
int S826_CounterFilterRead(
  uint board,    // board identifier
  uint chan,     // counter channel number
  uint *cfg      // filter configuration
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

chan
Counter channel number in the range 0 to 5.

cfg
Pointer to buffer that will receive the configuration. The format of the returned value is identical to the cfg value used in the S826_CounterFilterWrite function.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

7.3.18 S826_CounterModeWrite

The S826_**Counter**ModeWrite function configures a counter channel's operating mode.

```
int S826_CounterModeWrite(
  uint board,     // board identifier
  uint chan,      // channel number
  uint mode       // operating mode
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

chan
Counter channel number in the range 0 to 5.

mode
Channel operating mode (see "Common Applications" for examples):

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	IP	IM		0	0	0	TP	NR	UD	BP	OM			OP	TP						TE		TD		K			XS			

IP	ExtIn signal polarity. This is ignored if IM=0.
0	Normal polarity.
1	Inverted polarity.

IM	ExtIn mode. If IM>0 then ExtIn is routed to a DIO pin as specified by S826_CounterExtInRoutingWrite.
0	ExtIn input is not used. The counter's ExtIn input will effectively be hardwired to logic '1'.
1	ExtIn input is a count enable permissive. This is typically used with incremental encoders and event counting, with ExtIn acting as a count enable.
2	ExtIn input is a preload enable permissive. This is typically used when operating as a pulse generator, with ExtIn used as a pulse trigger.

TP bit	Preload enables. Determines event(s) that will cause a preload register to be copied to the counter. Each bit enables preloads for one type of event: '1' = enable, '0' = disable.
24	Preload upon start (when channel switches from halted to running)
16	Preload upon Index active level.
15	Preload upon Index leading edge.
14	Preload upon Index falling edge.
13	Preload upon zero counts reached.
12	Preload when Compare1 matches the counter core.
11	Preload when Compare0 matches the counter core.

NR	Preload permissive. Typically used for "one-shot" pulse generator applications.
0	Allow preloading any time. This can be used for applications such as a retriggerable one-shot.
1	Allow preloading only when counts equal zero. This can be used for applications such as a non-retriggerable one-shot. This applies to both hardware- and software-induced (via S826_CounterPreload function) preloads.

UD	Count direction
0	Normal count direction.
1	Reverse count direction.

BP	Preload toggle enable.
0	Use only Preload0. The Preload1 register will not be used.
1	Use both preload registers. This is typically used for PWM output.

OM	ExtOut mode. Determines when the channel's ExtOut output signal is active. The ExtOut signal may be routed to a DIO pin by calling S826_DioOutputSourceWrite.
0	Output always inactive.
1	Output pulses active upon Compare register snapshot.
2	Output active when Preload1 register is selected. This is typically used for PWM output.
3	Output active when the counts are not zero; useful for pulse generator applications.
4	Output active when the counts are zero; useful for watchdog timer applications.

OP	ExtOut signal polarity.
0	Normal polarity
1	Inverted

TE	Count enable trigger. Determines event(s) that will cause counting to be enabled.
0	Enable counting at start-up (i.e., when S826_CounterModeWrite is called to switch to running mode).
1	Enable counting upon Index leading edge.
2	Enable counting upon preload.
3	reserved -- do not use.

TD	Count disable trigger. Determines event(s) that will cause counting to be disabled.
0	Never disable counting.
1	Disable counting upon Index falling edge.
2	Disable counting upon zero counts reached.
3	reserved -- do not use

K	Clock mode.
0	External single-phase clock, increment on ClkA rising edge.
1	External single-phase clock, increment on ClkA falling edge.
2	Internal 1 MHz clock, increment every microsecond.
3	Internal 50 MHz clock, increment every 40 nanoseconds.
4	Link to adjacent channel's overflow/underflow outputs to this channel's overflow/underflow inputs (see "Cascading"). The adjacent channel's overflow/underflow output signals will cause this channel to increment/decrement. For channel 0, channel 5 is the adjacent channel; for all other channels N, the adjacent channel is N-1.
5	External quadrature clock, x1 multiplier.
6	External quadrature clock, x2 multiplier.
7	External quadrature clock, x4 multiplier.

XS	Index source.
0	External IX input, normal polarity.
1	External IX input, inverted.
2-7	ExtOut from counter channel 0-5 (e.g., 3 = ExtOut from channel 1).
8-15	Internal tick generator: 8 = 0.1 Hz, 9 = 1 Hz, 10 = 10 Hz, 11 = 100 Hz, 12 = 1 KHz, 13 = 10 KHz, 14 = 100 KHz, 15 = 1 MHz.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

7.3.19 S826_CounterModeRead

The S826_**Counter**ModeRead function returns a counter channel's operating mode.

```
int S826_CounterModeRead(
  uint board,    // board identifier
  uint chan,     // channel number
  uint *mode     // operating mode
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

chan
Counter channel number in the range 0 to 5.

mode
Buffer that will receive the channel operating mode, as defined in "S826_CounterModeWrite".

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

7.3.20 Common Applications

This section shows simple examples of how to configure and operate counter channels for common applications. In most cases there are other ways to configure the counters to achieve similar functionality, and numerous variations are possible that are not discussed here.

Encoder Interface

Monitor a quadrature-encoded device (e.g., incremental encoder) using x4 clock multiplier. Capture a snapshot at any time by calling S826_CounterSnapshot.

Register	Value
Mode	0x00000070

Event Counter

Count pulses applied to the ClkA input. Capture a snapshot at any time by calling S826_CounterSnapshot. Reset the counts to zero by calling S826_CounterPreload.

Register	Value
Mode	0x00000000
Preload0	0x00000000

Periodic Timer

Periodically capture snapshots.

Register	Value
Mode	0x00402020
Snapshot Configuration	0x00000004
Preload0	Period in µs

Delay Timer

Call S826_CounterPreload to start the timer, then S826_CounterSnapshotRead to wait for the delay time to elapse.

Register	Value
Mode	0x00400520
Snapshot Configuration	0x00000004
Preload0	Delay in µs

Frequency Measurement

Measure frequency by counting clocks applied to the ClkA input for one second intervals. At the end of each interval, a snapshot is captured and the next measurement begins automatically. The snapshot counts indicate measured frequency in Hz.

Register	Value
Mode	0x00008009
Snapshot Configuration	0x00000010
Preload0	0x00000000

Period Measurement

Measure the period of a signal applied to the IX input by counting internal clocks during one input cycle. At the end of each cycle, a snapshot is captured and the next measurement begins automatically. The snapshot counts indicate measured period in µs.

Register	Value
Mode	0x00008020
Snapshot Configuration	0x00000010
Preload0	0x00000000

Pulse Width Measurement

Measure the width of pulses applied to the IX input by counting internal clocks during one pulse. At the end of each pulse, a snapshot is captured and the next measurement begins automatically. The snapshot counts will indicate the measured pulse width in µs.

Register	Value
Mode	0x00008020
Snapshot Configuration	0x00000009
Preload0	0x00000000

<u>Pulse Generator</u>

Generate an output pulse on ExtOut in response to a trigger signal applied to the IX input. Also, a software trigger can be invoked by calling S826_CounterPreload. ExtOut must be routed to a DIO channel (see S826_DioOutputSourceWrite).

Register	Value
Mode	0x004C0520 (retriggerable), or 0x00CC0520 (non-retriggerable)
Preload0	Pulse duration in µs

<u>PWM Generator</u>

Output a pulse width modulation (PWM) signal on ExtOut, which must be routed to a DIO channel through the DIO_out signal routing matrix (see S826_DioOutputSourceWrite).

Register	Value
Mode	0x01682020
Preload0	PWM “on” time in µs
Preload1	PWM “off” time in µs

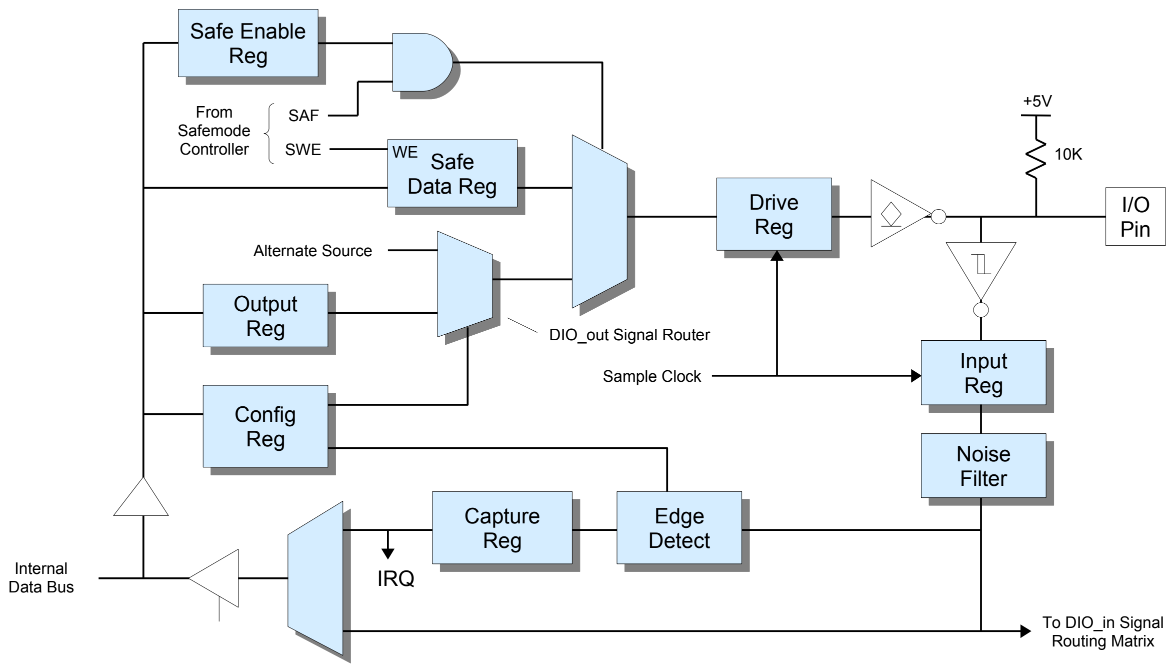
Synchronized PWM generators can be implemented by configuring the PWM channels as hardware triggered one-shots (pulse generators). Use an additional channel to generate a common trigger for the PWM channels; this channel should be configured to generate periodic output pulses at the desired PWM frequency. The duty cycle of each PWM channel is controlled by adjusting its output pulse width.

Chapter 8: Digital I/O

8.1 Introduction

The 826 board has 48 general-purpose digital I/O (DIO) channels. On the output side, each channel has a one-bit, writable output register, a synchronous drive register, and an inverting open-drain buffer that drives the channel's I/O pin. The open-drain buffer enables the pin to be driven low by the channel's output buffer or by an external signal. The pin, when not driven, is pulled up to +5V by a 10 kohm resistor. The output is high impedance when the board is unpowered so as to prevent unintended activation of an external solid state relay, if one is connected. The pin's physical state is sampled by an input register and then processed by a noise/debounce filter. The filter output is monitored by an edge detector and can also be directly read by the host computer.

Figure 7: DIO channel (1 of 48)



DIO signals are active-high on the local bus and active-low on the I/O pin. Writing a '1' to the output register causes the I/O pin to be driven low, whereas writing '0' allows the pin to be internally pulled up or driven high or low by an external circuit. Logic '0' must be written to the output register if the pin will be driven high by an external circuit; this will prevent high currents that could potentially damage the pin's output buffer.

The value read from a DIO channel indicates the filtered, sampled physical state of the I/O pin. If no external signals are driving the pin then the read value will equal the value stored in the drive register. The read value will differ from the drive register value if the drive register contains '0' while an external circuit drives the pin low.

Most of the host-accessible DIO registers support masked write operations so as to implement the atomic bit set and clear functions required for high performance, thread-safe operation.

8.1.1 Signal Routing Matrix

Each DIO channel connects to the board's internal signal routing matrix as shown in Figure 7. The matrix can be programmed to route the DIO connector pin to or from another interface (e.g., counter, watchdog, analog input system) so that the pin will act as a physical input or output for that interface.

The channel's DIO_out signal router consists of a data selector that can route either the DIO output register or an alternate source to the I/O pin. The pin will function as a general-purpose digital output when the DIO output register is selected. If the alternate source is selected, the pin state will be controlled by the alternate signal, but all DIO input functions (read, edge detection) will continue to operate normally. Each DIO is associated with a specific alternate source as explained in S826_DioOutputSourceWrite.

The DIO_in routing matrix connects the sampled DIO pin signal to other interfaces. The ADC trigger input and the six counter ExtIn inputs are connected to the DIO_in matrix so that any of these signals can be sourced from a DIO pin. When a DIO signal is routed to another interface via the DIO_in matrix, all of the DIO channel's input and output functions will continue to operate normally.

8.1.2 Safemode

Safemode is activated when the SAF signal (see Figure 7) is asserted. When operating in safemode, the DIO pin state is determined by the Safe Enable and Safe Data registers: when Safe Enable equals '1', the pin will be driven to the fail-safe value stored in Safe Data; when Safe Enable equals '0' the pin will output its normal runmode signal.

Upon board reset, the Safe Enable register is set to '1' so that the DIO pin will exhibit fail-safe behavior by default (i.e., it will output the contents of the Safe Data register when SAF is asserted). Fail-safe operation can be disabled for a DIO pin by programming its Safe Enable register to '0'.

The Safe Data register is typically programmed once (or left at its default value) by the application when it starts, though it may be changed at any time if required by the application. It can be written to only when the SWE bit is set to '1'. See “Safemode Controller” for more information about safemode.

8.1.3 Edge Capture

Every DIO channel includes an edge detection circuit and a capture flag register. When edge capturing is enabled, a channel's capture flag will be set when an edge is detected on its I/O pin. Each channel may be programmed to capture rising edges, falling edges, or both edges, or capturing may be disabled.

The API allows capture flags to be monitored by polling or, if the application is event driven, the calling thread can block while it waits for captured events. When blocking on edge capture events, the calling thread can specify a set of capture flags to wait for, and it can wait for either all of the events or any one event in the set.

When read by the host, capture flags are reset but remain enabled to capture future events. Edge events that occur on a channel while its capture flag is set will be lost. An input signal must hold for at least 20 ns after a transition for the transition to be reliably detected.

8.1.4 Pin Timing

The DIO subsystem is a fully synchronous system that is controlled by a 50 MHz sampling clock. The DIO pin drivers are updated and pin receivers are sampled once per cycle. As a result, outputs cannot change faster than the cycle time and inputs cannot be sampled faster than the cycle time.

Output registers are organized as two 24-channel groups. When these registers are written (via S826_DioOutputWrite), channels 0-23 will change simultaneously and channels 24-47 will also change simultaneously, but these two 24-channel groups are not guaranteed to change output states at the same time. Also, a DIO pin does not change output state immediately when its signal source (DIO output register or counter ExtOut) changes; it will be delayed for 20 ns due to the sampling clock.

When used as inputs, all 48 DIO channels are sampled simultaneously every 20 ns. As a result, the received signal on a DIO pin may be delayed up to 20 ns en route to its destination (e.g., DIO edge detector or read data, counter ExtIn input, or ADC trigger input), and input signal pulses shorter than 20 ns may not be recognized. The host reads pin states (via S826_DioInputRead) as two 24-channel groups (channels 0-23 and 24-47) in two separate read cycles. Consequently, channels within each group are guaranteed to be sampled simultaneously, but the two groups are not guaranteed to be associated with the same sample clock.

8.1.4.1 Noise Filter

Each DIO channel input circuit includes a noise filter that can be used to filter glitches (see S826_DioFilterWrite). A filter's output will change state only when its input has held constant for time T, the filter time interval. Consequently, when a DIO channel's filter is enabled, the sampled input signal will be delayed for an additional T * 20 ns en route to its destination, and input signal pulses shorter than T * 20 ns will not be recognized.

8.1.5 Reset State

DIO channels are forced to the following condition upon board reset:

- Output and Safe Data registers programmed to zero.
- Safe Enable registers programmed to all 1's.
- Output register is selected as I/O pin data source.
- Event capture disabled.

8.2 Connectors J2/J3

J2 Pinout - DIO channels 24-47

Pin	**1**	**3**	**5**	**7**	**9**	**11**	**13**	**15**	**17**	**19**	**21**	**23**	**25**	**27**	**29**	**31**	**33**	**35**	**37**	**39**	**41**	**43**	**45**	**47**	**49**	**Even**
DIO Channel	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	30	30	29	28	27	26	25	24	+5V	GND

J3 Pinout - DIO channels 0-23

Pin	**1**	**3**	**5**	**7**	**9**	**11**	**13**	**15**	**17**	**19**	**21**	**23**	**25**	**27**	**29**	**31**	**33**	**35**	**37**	**39**	**41**	**43**	**45**	**47**	**49**	**Even**
DIO Channel	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	+5V	GND

All even pin numbers on J2 and J3 connect to the power supply return. Odd pin numbers 1-47 are the active-low DIO channel I/O pins. Pin 49 is a +5V power output for low power loads such as solid state relay racks.

8.3 Programming

The DIO functions use individual bits to convey information about the DIO channels, wherein each bit represents the information for one channel. In such cases, the information for the 48 DIO channels is organized as an array of two bit quadlets (32-bit values), with each quadlet containing the information for 24 DIO channels:

The quadlet at array[0] is associated with DIO channels 0 to 23:

Bit	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
DIO	-	-	-	-	-	-	-	-	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

The quadlet at array[1] is associated with DIO channels 24 to 47:

Bit	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
DIO	-	-	-	-	-	-	-	-	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24

Typically, the DIO functions expect a pointer to a two-quadlet array, which must have been previously allocated by the program.

Although the DIO functions read or write values for all 48 DIO channels, the physical read or write operation is performed in steps; channels 0 to 23 first, as a group, and then channels 24 to 47 as another group. As a result, reads and writes do not sample or update all channels simultaneously. In general, channels 0 to 23 are sampled or updated concurrently as a group, and channels 24 to 47 are sampled or updated concurrently as a separate group.

8.3.1 S826_DioOutputWrite

The S826_DioOutputWrite function programs the DIO output registers.

```
int S826_DioOutputWrite(
  uint board,     // board identifier
  uint data[2],   // pointer to DIO data
  uint mode       // 0=write, 1=clear bits, 2=set bits
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

data
Pointer to data array (see Section 8.3).

mode
Write mode for data: 0 = write, 1 = clear bits, 2 = set bits (see “Atomic Read-Modify-Write”).

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

In mode zero, this function will unconditionally write new values to all DIO output registers. In modes one and two, the function will selectively clear or set any arbitrary combination of output registers; data bits that contain logic '1' indicate DIO output registers that are to be modified, while all other output registers will remain unchanged. Modes one and two can be used to ensure thread-safe operation as they atomically set or clear the specified bits.

8.3.2 S826_DioOutputRead

The S826_DioOutputRead function reads the programmed states of all DIO output registers.

```
int S826_DioOutputRead(
  uint board,    // board identifier
  uint data[2]   // pointer to data buffer
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

data
Pointer to a buffer (see Section 8.3) that will receive the output register states.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function returns the output register states. Note that the returned values may not be the same as the physical I/O pin states in the case of pins that are externally driven or routed to a counter channel's output signal.

8.3.3 S826_DioInputRead

The S826_DioInputRead function reads the physical states of all DIO channel I/O pins.

```
int S826_DioInputRead(
  uint board,    // board identifier
  uint data[2]   // pointer to data buffer
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

data
Pointer to a buffer (see Section 8.3) that will receive the physical I/O pin states.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function returns the sampled physical states of all DIO pins in the data buffer.

If this function is called immediately after calling S826_DioOutputWrite, the read data (sampled pin states) may not accurately reflect the previously written data. This happens because the DIO pins are driven by open-drain buffers with pull-up resistors. A pin will change state quickly when driven low, but when it is switched high, the state cannot change as quickly because additional time is required for the circuit capacitance to be charged through the pull-up resistor. The amount of time required for this depends on circuit capacitance; it can be shortened by decreasing the capacitance or by decreasing the pull-up resistance (by adding an external pull-up resistor).

In some applications, it may be desirable to have a DIO write function that will not return until all pins are stable. This can be implemented by calling S826_DioOutputWrite, and then polling with S826_DioInputRead until the expected states are read.

8.3.4 S826_DioSafeWrite

The S826_DioSafeWrite function programs the DIO Safe registers.

```
int S826_DioSafeWrite(
  uint board,     // board identifier
  uint data[2],   // pointer to safemode data
  uint mode       // 0=write, 1=clear bits, 2=set bits
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

data
Pointer to data array (see Section 8.3) to be programmed into the Safe registers.

mode
Write mode for data: 0 = write, 1 = clear bits, 2 = set bits (see "Atomic Read-Modify-Write").

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function should only be called when the SWE bit is set (see S826_SafeWrenWrite). The function will fail without notification (return S826_ERR_OK) if SWE=0 (see Section 10.1.1).

8.3.5 S826_DioSafeRead

The S826_DioSafeRead function returns the contents of the DIO Safe registers.

```
int S826_DioSafeRead(
  uint board,    // board identifier
  uint data[2]   // pointer to data buffer
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

data
Pointer to a buffer (see Section 8.3) that will receive the Safe register states.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

8.3.6 S826_DioSafeEnablesWrite

The S826_DioSafeEnablesWrite function programs the DIO Safe Enable registers.

```
int S826_DioSafeEnablesWrite(
  uint board,       // board identifier
  uint enables[2]   // pointer to safemode enables
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

enables
Pointer to array of values (see Section 8.3) to be programmed into the Safe Enable registers.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function should only be called when the SWE bit is set (see S826_SafeWrenWrite). The function will fail without notification (return S826_ERR_OK) if SWE=0 (see Section 10.1.1).

8.3.7 S826_DioSafeEnablesRead

The S826_DioSafeEnablesRead function returns the contents of the DIO Safe Enable registers.

```
int S826_DioSafeEnablesRead(
  uint board,      // board identifier
  uint enables[2]  // pointer to data buffer
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

enables
Pointer to a buffer (see Section 8.3) that will receive the Safe Enable register contents.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

8.3.8 S826_DioCapEnablesWrite

The S826_DioCapEnablessWrite function programs the edge sensitivity for DIO edge capturing.

```
int S826_DioCapEnablesWrite(
  uint board,       // board identifier
  uint rising[2],   // pointer to data buffer
  uint falling[2],  // pointer to data buffer
  uint mode         // write mode
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

rising
Pointer to a buffer (see Section 8.3) that specifies rising edge sensitivity: '1'=capture rising edges, '0'=don't capture rising edges.

falling
Pointer to a buffer (see Section 8.3) that specifies falling edge sensitivity: '1'=capture falling edges, '0'=don't capture falling edges.

mode
Write mode for rising/falling: 0 = write, 1 = clear bits, 2 = set bits (see “Atomic Read-Modify-Write”).

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function specifies the edges to be captured on DIO channels. It defines the edge sensitivity for each DIO channel in terms of the I/O pin voltage. For example, if falling edge sensitivity is enabled, edge capturing will occur when the I/O pin voltage transitions from 5V to 0V.

When mode is zero, all data flags are directly written to the capture enable registers. In modes one and two, the data flags determine which of the 48 DIO channels are to be affected; any flag that contains a logic '1' will cause the associated channel to be affected, while '0' will leave the channel unmodified.

After capturing has been enabled, it will remain enabled until disabled by this function or a board reset. Capturing is disabled on all channels following a board reset.

8.3.9 S826_DioCapEnablesRead

The S826_DioCapEnablesRead function returns the programmed edge sensitivity for DIO edge capturing.

```
int S826_DioCapEnablesRead(
  uint board,       // board identifier
  uint rising[2],   // pointer to data buffer
  uint falling[2]   // pointer to data buffer
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

rising
Pointer to a buffer (see Section 8.3) that receives rising edge sensitivity: '1'=capture rising edges, '0'=don't capture rising edges.

falling
Pointer to a buffer (see Section 8.3) that receives falling edge sensitivity: '1'=capture falling edges, '0'=don't capture falling edges.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function receives information about the types of edge events that will be captured, which were previously programmed by calling This function returns the sampled physical states of all DIO pins in the data buffer..

8.3.10 S826_DioCapRead

The S826_DioCapRead function waits for edge events on one or more DIO channels.

```
int S826_DioCapRead(
  uint board,         // board identifier
  uint chanlist[2],   // pointer to channel flags
  uint waitall,       // logic operator: 1=and (all), 0=or (any)
  uint tmax           // maximum time to wait
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

chanlist
Pointer to channel flag bits (see Section 8.3). Upon entry, set flags indicate channels of interest. Upon exit, set flags indicate channels with captured edges.

waitall
Logic operator to apply to channels of interest: 1 = AND (return when all channels have events), 0 = OR (return when any channel has an event). This is ignored if tmax = 0.

tmax
Maximum time, in microseconds, to wait for the events of interest. See “Event-Driven Applications“ for details.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function waits for edge events to be captured on an arbitrary set of DIO channels (the “channels of interest”) and then returns information about the events. Event capturing must have been previously enabled for channels of interest by calling S826_DioCapEnablesWrite.

Before calling the function, one or more chanlist bits must be set to identify the channels of interest. The function will modify chanlist to indicate channels of interest that have captured events, and it will reset the capture flag registers for all such indicated channels, thus re-enabling event capturing on those channels.

The function operates in either blocking or non-blocking mode depending on the value of tmax. If tmax is zero (non-blocking mode), the function will return immediately. If tmax is greater than zero (blocking mode), the calling thread will block until the capture criteria is satisfied or tmax elapses. In either case, some channels of interest may have captured events while others may not have; these will be indicated by chanlist when the function returns.

In blocking mode, the capture criteria is specified by waitall. Depending on waitall, the function will wait for events to be captured on either any, or all channels of interest. When waitall is true, the function will return when all channels of interest have captured events. When waitall is false, the function will return when any channel of interest has captured an event. The function will return S826_ERR_NOTREADY if tmax elapses before the capture criteria is satisfied.

In non-blocking mode, waitall is ignored and the function will never return S826_ERR_NOTREADY.

This function will return S826_ERR_CANCELLED if, while it is blocking, another thread calls S826_DioWaitCancel to cancel waits on any of the blocking channels. It will return immediately if the wait criteria is completely satisfied due to the wait cancellations, otherwise it will continue to block and return when all remaining wait criteria is satisfied. S826_ERR_BOARDCLOSED will be returned immediately if S826_SystemClose executes while this function is blocking. In either case, chanlist will be invalid when the function returns.

Thread-safe operation is guaranteed if the channels of interest for any given thread do not coincide with those of another thread. For example, thread safety is assured if a thread designates channels 1 and 3-5 as channels of interest while another thread designates channels 2 and 9.

8.3.11 S826_DioWaitCancel

The S826_DioWaitCancel function cancels a blocking wait on one or more DIO channels.

```
int S826_DioWaitCancel(
  uint board,         // board identifier
  uint chanlist[2]    // pointer to channel flags
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

chanlist
Pointer to DIO channel flag bits (see Section 8.3) that indicate channels for which waiting is to be cancelled.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function cancels blocking for an arbitrary set of DIO channels so that another thread, which is blocked by S826_DioCapRead while waiting for DIO edge events to be captured, will return immediately with S826_ERR_CANCELLED.

8.3.12 S826_DioOutputSourceWrite

The S826_DioOutputSourceWrite function assigns the signal sources for all DIO pins.

```
int S826_DioOutputSourceWrite(
  uint board,     // board identifier
  uint data[2]    // pointer to data buffer
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

data
Pointer to a buffer (see Section 8.3) that specifies signal sources: 0=DIO output register, 1=alternate source.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function specifies the signal source that will be used to drive each DIO pin. Each DIO pin may be driven by its output register or by its alternate signal source. Upon board reset, all DIO channels assume their default configuration so that all DIO pins are driven by the DIO output registers (vs. alternate sources).

A DIO pin's signal source is determined by the corresponding bit in the data buffer. When the bit is set to '1' the pin will be driven by the channel's alternate signal source; when set to '0' (default) the pin will behave as a standard digital output, driven by the channel's output register.

The data[0] quadlet selects the signal sources for DIO channels 0 to 23:

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIO	-	-	-	-	-	-	-	-	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Src	-	-	-	-	-	-	-	-	NMI	RST	C5	C4	C3	C2	C1	C0	NMI	RST	C5	C4	C3	C2	C1	C0	NMI	RST	C5	C4	C3	C2	C1	C0

The data[1] quadlet selects the signal sources for DIO channels 24 to 47:

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIO	-	-	-	-	-	-	-	-	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24
Src	-	-	-	-	-	-	-	-	NMI	RST	C5	C4	C3	C2	C1	C0	NMI	RST	C5	C4	C3	C2	C1	C0	NMI	RST	C5	C4	C3	C2	C1	C0

Every DIO channel is associated with one of eight alternate signal sources as shown in the above tables. The tables use the following abbreviations for alternate signal sources:

Symbol	Signal Source
C0	Counter 0 ExtOut
C1	Counter 1 ExtOut
C2	Counter 2 ExtOut
C3	Counter 3 ExtOut
C4	Counter 4 ExtOut
C5	Counter 5 ExtOut
RST	Watchdog RST (Timer2) output
NMI	Watchdog NMI (Timer1) output

Examples:

- When data[1] bit 2 is set, DIO26 will be driven by the ExtOut signal from counter channel 2.
- When data[0] bit 14 is set, DIO14 will be driven by the Watchdog RST output signal.

Each of the eight alternate signal sources is associated with six DIO channels. For example, the watchdog RST signal is associated with DIO channels, 6, 14, 22, 30, 38, and 46. Typically, an alternate source is either not used or it is routed to one of its associated DIO pins, though it may be simultaneously routed to any combination of its associated pins.

This function should only be called when the SWE bit is set (see S826_SafeWrenWrite). The function will fail without notification (return S826_ERR_OK) if SWE=0 (see Section 10.1.1).

8.3.13 S826_DioOutputSourceRead

The S826_DioOutputSourceRead function reads the signal sources assigned to all DIO channels.

```
int S826_DioOutputSourceRead(
  uint board,    // board identifier
  uint data[2]   // pointer to data buffer
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

data
Pointer to a buffer (see Section8.3) that will receive the signal source assignments for all DIO channels.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

8.3.14 S826_DioFilterWrite

The S826_DioFilterWrite function configures the DIO input noise filters.

```
int S826_DioFilterWrite(
  uint board,       // board identifier
  uint interval,    // filter interval (T)
  uint enables[2]   // filter enable flags
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

interval
Filter time interval in range 1 to 65535, specified as a multiple of 20ns. This is common to all DIO channels.

enables

Pointer to a buffer (see Section 8.3) that specifies filter enables for the DIO channels: '1'=enable, '0'=disable.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

Each DIO has an input noise filter that can be independently enabled or disabled. This function selectively enables and disables the individual filters and programs the filter time interval T, which is common to all DIO filters.

A filter's output will not change state until its input has held a constant state for T * 20ns. The maximum T value is 65535, corresponding to a filter time of 1.3107ms. When a DIO channel's noise filter is enabled, its input signal will be delayed by T * 20ns and input pulses shorter than T * 20ns will not be recognized.

8.3.15 S826_DioFilterRead

The S826_DioFilterRead function reads the configuration of the DIO input noise filters.

```
int S826_DioFilterRead(
  uint board,       // board identifier
  uint *interval,   // filter interval (T)
  uint enables[2]   // filter enable flags
);
```

Parameters

board

826 board number. This must match the settings of the board's dip switches as described in section 2.2.

interval

Pointer to a buffer that will receive the filter time interval as described in S826_DioFilterWrite.

enables

Pointer to a buffer (see Section 8.3) that will receive filter enable flags for the DIO channels: '1'=enable, '0'=disable.

Return Values

If the function succeeds, the return value is zero.

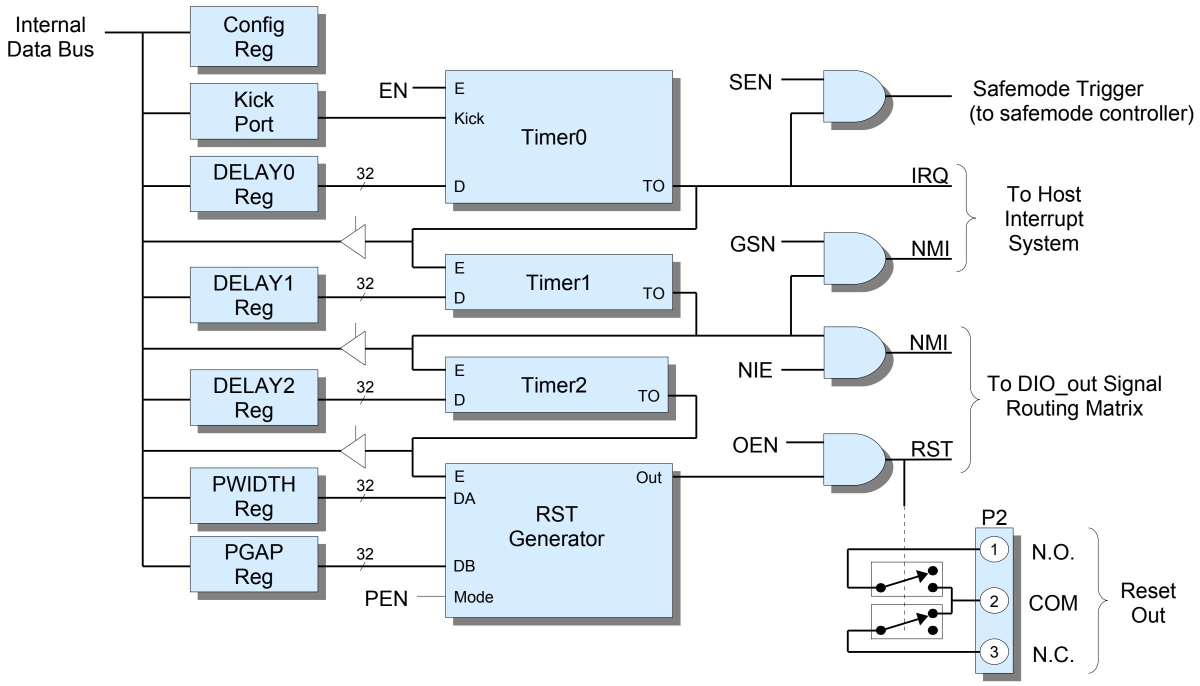
If the function fails, the return value is an error code.

Chapter 9: Watchdog Timer

9.1 Introduction

The model 826 board has a multistage watchdog timer that can activate the board's safemode system and generate service requests. The watchdog has three timer stages that generate timeout events in sequence according to user-defined timing. Each event is associated with an output signal. Typically, the event signals are utilized in such a way that successive events will generate service requests of progressively higher priority.

Figure 8: Watchdog system



9.1.1 Operation

When the watchdog system is disabled (default upon board reset), the three timers are halted and loaded from their associated DELAY registers. When the system becomes enabled (EN='1') by calling S826_WatchdogEnableWrite, initially only Timer0 is enabled so that it will count down towards zero while the other timers remain idle. During normal operation, the program regularly "kicks" the watchdog by writing to the Kick port, thus reloading Timer0 from DELAY0 before it can count down to zero.

If a fault condition prevents the program from kicking the watchdog, Timer0 will count down to zero and assert its timeout (TO) signal. After this event occurs, all subsequent kicks will be ignored. This event enables Timer1 and generates an interrupt request, and if SEN='1', it switches all control outputs to fail-safe states by triggering the safemode system. The program can wait for the interrupt by calling S826_WatchdogEventWait.

Timer1 asserts its TO signal upon counting down to zero. This event enables Timer2 and, if NIE='1', it asserts the NMI net of the DIO_out signal routing matrix, which in turn may route the net to a DIO pin (see S826_DioOutputSourceWrite). When GSN='1', a Timer1 event will cause the board to issue a PCI Express fatal error message; this can be used to generate a system non-maskable interrupt request if the system has been appropriately configured. Refer to your system documentation for information about generating NMI in response to a PCI Express fatal error message.

Timer2 asserts its TO signal upon counting down to zero, thus activating the RST signal generator. If OEN='1', the RST generator's output is routed to the RST net of the DIO_out signal routing matrix and to the board's Reset Out circuit. The RST generator can produce a continuous (non-pulsed) output or pulsed output. When generating a pulsed output, PWIDTH determines the pulse duration and PGAP determines the gap time between pulses. The RST signal is not internally connected to the host computer's system reset input; if desired, this must be implemented by externally routing the selected DIO pin to the computer's reset input.

9.1.2 Reset Out Circuit

Two solid state relays (SSRs) are provided for controlling external reset circuits. Both SSRs are energized when the watchdog RST generator is asserting its output signal. One SSR has normally open contacts and the other normally closed contacts. The SSRs are galvanically isolated from other board circuitry.

9.1.3 Initialization

Before enabling the watchdog, the program must initialize it by writing to the configuration register and the five timing control registers (DELAY0-DELAY2, PWIDTH, and PGAP). This is done by calling S826_WatchdogConfigWrite. The DELAY registers determine the time intervals of their three associated timers, whereas the PWIDTH and PGAP registers determine the timing of the RST output signal.

9.2 Connector P2

Connector P2 interfaces external circuitry to the Reset Out solid state relay. Refer to the block diagram in section 9.1 for connector pinout.

9.3 Programming

9.3.1 S826_WatchdogConfigWrite

The S826_WatchdogConfigWrite function configures the watchdog system.

```
int S826_WatchdogConfigWrite(
  uint board,     // board identifier
  uint cfg,       // configuration flags
  uint timing[5]  // time intervals
);
```

Parameters

board

826 board number. This must match the settings of the board's dip switches as described in section 2.2.

cfg

Configuration flags: '1' = enable feature, '0' = disable feature.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	GSN	0	SEN	NIE	PEN	0	OEN

Flag	Function
GSN	Generate host system NMI upon Timer1 event.
SEN	Activate safemode upon Timer0 event.
NIE	Connect Timer1 event signal to the DIO_out routing matrix NMI net.
PEN	Enable RST output to pulse (vs. continuous active level).
OEN	Connect RST generator to the DIO_out routing matrix RST net.

timing

Pointer to array of five quadlets that define the watchdog's time intervals. Each quadlet is written to one of the watchdog timing control registers as shown below. All times are specified as multiples of 20 nanoseconds. For example, use the value 50,000,000 for a one-second time interval.

Quadlet	Register	Function
timing[0]	DELAY0	Timer0 interval. The program must kick the watchdog within this interval to prevent a watchdog timeout. This must be set to a non-zero value; set to 1 for shortest possible delay.
timing[1]	DELAY1	Timer1 interval. This specifies the elapsed time from Timer1 timeout to Timer2 timeout. This must be set to a non-zero value; set to 1 for shortest possible delay.
timing[2]	DELAY2	Timer2 interval. This specifies the elapsed time from Timer2 timeout to RST generator enable. This must be set to a non-zero value; set to 1 for shortest possible delay.
timing[3]	PWIDTH	RST pulse width. This is ignored if PEN='0'.
timing[4]	PGAP	Time gap between RST pulses. This is ignored if PEN='0'.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function programs the watchdog configuration register and timing control registers. To ensure reliable operation, it should be called only when the watchdog is disabled.

The function should only be called when the SWE bit is set (see S826_SafeWrenWrite). The function will fail without notification (return S826_ERR_OK) if SWE=0 (see Section 10.1.1).

9.3.2 S826_WatchdogConfigRead

The S826_WatchdogConfigRead function reads the watchdog configuration.

```
int S826_WatchdogConfigRead(
  uint board,     // board identifier
  uint *cfg,      // configuration flags
  uint timing[5]  // time intervals
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

cfg
Pointer to buffer that will receive the watchdog configuration flags.

timing
Pointer to array of five quadlets that will receive the watchdog time intervals.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

See Section 9.3.1 for details about the returned cfg and timing values.

9.3.3 S826_WatchdogEnableWrite

The S826_WatchdogEnableWrite function enables or disables the watchdog system.

```
int S826_WatchdogEnableWrite(
  uint board,    // board identifier
  uint enable    // enable watchdog when true
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

enable
Set to '1' to enable, or '0' to disable the watchdog. The watchdog is disabled by default upon board reset.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function should only be called when the SWE bit is set (see S826_SafeWrenWrite). The function will fail without notification (return S826_ERR_OK) if SWE=0 (see Section 10.1.1).

9.3.4 S826_WatchdogEnableRead

The S826_WatchdogEnableRead function returns the enable status of the watchdog system.

```
int S826_WatchdogEnableRead(
  uint board,    // board identifier
  uint *enable   // enable status
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

enable
Buffer that will receive the watchdog system enable status: '1' = enabled, '0' = disabled.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

9.3.5 S826_WatchdogStatusRead

The S826_WatchdogStatusRead function reads the watchdog timeout status.

```
int S826_WatchdogStatusRead(
  uint board,       // board identifier
  uint *status      // watchdog status
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

status
Pointer to a quadlet buffer that will receive the watchdog status. Each status bit indicates the timeout status of one watchdog timer stage ('1' = timed out):

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TO2	TO1	TO0

Flag	Function
TO2	Timer2 timeout
TO1	Timer1 timeout
TO0	Timer0 timeout

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

9.3.6 S826_WatchdogKick

The S826_WatchdogKick function reads the watchdog timeout status.

```
int S826_WatchdogKick(
  uint board,       // board identifier
  uint data         // valid signature
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

data
Valid signature. This must be 0x5A55AA5A to kick the watchdog; any other value will fail to kick the watchdog, although no error will be returned.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

When the watchdog system is running, the program should call this function to prevent a watchdog timeout. The DELAY0 value determines how often this function must be called to prevent a timeout.

9.3.7 S826_WatchdogEventWait

The S826_WatchdogEventWait function waits for a watchdog event (timeout) on Timer1.

```
int S826_WatchdogEventWait(
  uint board,      // board identifier
  uint tmax        // maximum time to wait
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

tmax
Maximum time, in microseconds, to wait for data. See "Event-Driven Applications" for details.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function can operate in either blocking or non-blocking mode. If tmax is zero, the function will return immediately. If tmax is greater than zero, the calling thread will block until a watchdog event or tmax elapses. The function will return zero if a watchdog timeout event occurred, or S826_ERR_NOTREADY if the watchdog has not timed out.

When this function is blocking, it will return immediately with return code S826_ERR_CANCELLED if S826_WatchdogWaitCancel is called by another thread, or with S826_ERR_BOARDCLOSED if S826_SystemClose is called by another thread.

9.3.8 S826_WatchdogWaitCancel

The S826_WatchdogWaitCancel function cancels a blocking wait on watchdog Timer1.

```
int S826_WatchdogWaitCancel(
  uint board    // board identifier
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

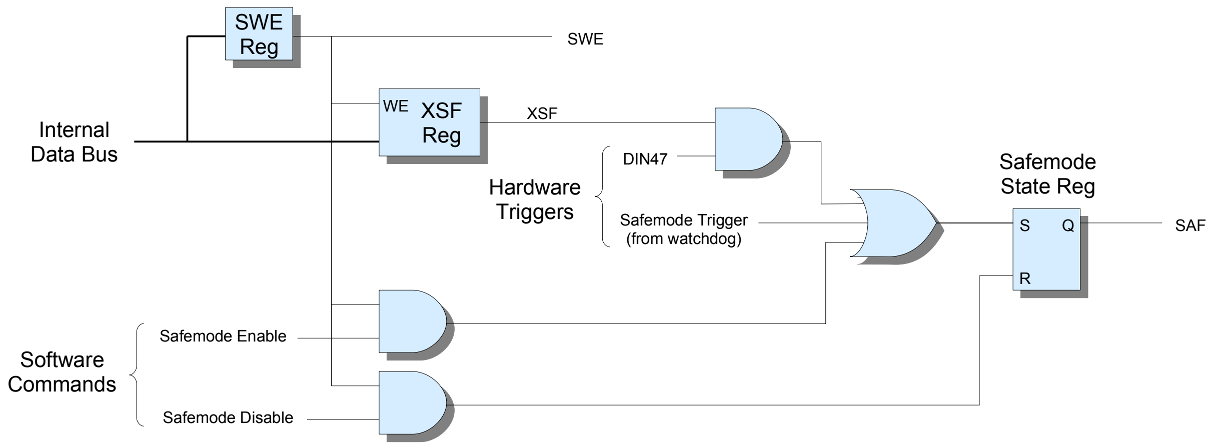
This function cancels blocking on the watchdog so that another thread, which is blocked by S826_WatchdogEventWait while waiting for a watchdog timeout event, will return immediately with S826_ERR_CANCELLED.

Chapter 10: Safemode Controller

10.1 Introduction

The 826 board features a fail-safe controller that forces analog and digital outputs to predetermined levels in response to hardware triggers. The controller works in concert with the watchdog timer and external devices such as emergency shutdown contacts to switch the board's outputs to fail-safe levels without software intervention.

Figure 9: Safemode Controller



The controller consists of configuration (XSF) and write protection control (SWE) registers, triggering logic, and a state register. When safemode is active (SAF = '1'), the board's analog and digital outputs are automatically switched to their fail-safe states. SAF can be set by the program and in response to hardware triggers, but only the program can reset SAF to turn off safemode. Upon power-up or board reset, SAF is reset.

If the watchdog is allowed to activate safemode (see S826_WatchdogConfigWrite), it will assert the Safemode Trigger signal upon Timer0 event, thus setting SAF. Once asserted, the trigger will remain asserted until the watchdog is disabled. Consequently, the program cannot reset SAF until the watchdog is disabled.

When XSF = '1', safemode can be triggered by an active-low signal applied to the DIO channel 47 connector pin (DIO47). When this happens, the program cannot reset SAF until DIO47 is negated or XSF is cleared.

Additional information about safemode can be found in Section 6.1.1 (analog outputs) and Section 8.1.2 (DIO outputs).

10.1.1 Write Protection

The SWE register controls write protection for registers associated with the watchdog and safemode controller. All affected registers are write-protected when SWE = '0'; this is the default state of SWE at power-up and upon system reset. The SWE register state does not change when the board is opened or closed.

Before writing to protected registers, the program must set SWE (by calling S826_SafeWrenWrite) to allow writes to the registers. During initialization, the program will typically disable write protection, write all fail-safe states as required by the application, and then re-enable write protection to prevent modification of the registers due to subsequent wayward software execution.

Several of the API functions write to SWE protected registers. These functions can fail without notification if called while SWE = '0' (they will return S826_ERR_OK if no other errors are detected, but the protected register will not be written). If it is necessary to detect a failed write to a write-protected register, the program should read the register after writing to it and compare the read and written values; a failed write is indicated when the read and written values are not equal. Each of the write functions has a corresponding read function that can be used to read back the programmed register state; these are not affected by the state of the SWE register.

These API functions write to SWE protected registers:

- S826_DacRangeWrite and S826_DacDataWrite (when safemode argument = '1')
- S826_DioOutputSourceWrite, S826_DioSafeWrite, and S826_DioSafeEnablesWrite
- S826_WatchdogConfigWrite, S826_WatchdogEnableWrite
- S826_SafeControlWrite
- S826_VirtualSafeWrite and S826_VirtualSafeEnablesWrite

10.2 Programming

10.2.1 S826_SafeControlWrite

The S826_SafeControlWrite function programs the board's fail-safe configuration and state.

```
int S826_SafeControlWrite(
  uint board,     // board identifier
  uint settings,  // safemode settings
  uint mode       // write mode
);
```

Parameters

board

826 board number. This must match the settings of the board's dip switches as described in section 2.2.

settings

Safemode configuration and state bits:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	XSF	0	SAF	0

Bit	Function
XSF	DIO47 trigger enable. Programmed to '0' upon reset. When '1', a low level (0 volts) on the DIO channel 47 header pin will set the SAF bit. When '0', DIO47 will not affect the SAF bit. When XSF=1, a thread can block on DIO47 falling edge events to receive notification when safemode is triggered. Alternatively,the program can poll SAF.
SAF	Safemode state. Programmed to '0' upon reset. '1' = safemode active, '0' = runmode active. This can be written by the application program. This bit can also be set by DIO47 when XSF=1, or by a timeout event on watchdog Timer0.

mode

Write mode: 0 = write, 1 = clear bits, 2 = set bits (see “Atomic Read-Modify-Write”).

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

This function should only be called when the SWE bit is set (see S826_SafeWrenWrite). The function will fail without notification (return S826_ERR_OK) if SWE=0 (see Section 10.1.1).

10.2.2 S826_SafeControlRead

The S826_SafeControlRead function returns the board's fail-safe configuration and state.

```
int S826_SafeControlRead(
  uint board,     // board identifier
  uint *settings  // safemode settings
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

settings
Pointer to buffer that will receive the safemode configuration and state as detailed in S826_SafeControlWrite.

mode
Write mode: 0 = write, 1 = clear bits, 2 = set bits (see “Atomic Read-Modify-Write”).

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

10.2.3 S826_SafeWrenWrite

The S826_SafeWrenWrite function enables or disables write protection for safemode-related registers.

```
int S826_SafeWrenWrite(
  uint board,     // board identifier
  uint wren       // write enable/disable
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

wren
1 = write protect (default upon board reset), 2 = write enable, other values have no effect. When writes are disabled (write protected), attempts to write to protected registers will without notification (return S826_ERR_OK).

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Remarks

See “Write Protection” for a list of registers that are write protected/enabled by this function.

10.2.4 S826_SafeWrenRead

The S826_SafeWrenRead function returns the board's fail-safe configuration and control settings.

```
int S826_SafeWrenRead(
  uint board,     // board identifier
  uint *wren      // write protection status
);
```

Parameters

board
826 board number. This must match the settings of the board's dip switches as described in section 2.2.

wren
Pointer to buffer that will receive the write protection status: 0 = write protected, 2 = write enabled.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is an error code.

Chapter 11: Specifications

Digital I/O		
Channels	Number/type	48 bi-directional
	Signal levels	5 V TTL/CMOS
	Sampling rate	50 Ms/s
Output	Driver type	Open drain, active-low sink driver
	Internal pull-up impedance	10 kΩ, 5%
	On-state sink current (maximum)	24 mA
	Sink current when board unpowered	< 20 µA
	Fail-safe mode	Yes
Input	Receiver type	Schmitt trigger
	Noise/debounce filter	Interval: 0 to 1.3107 ms in 20 ns steps, common to all channels. Enable/disable: per channel.

Counters		
Channels	Number/type	6 multifunction
	Resolution	32 bits
	Count rate (maximum)	25 MHz (external clock), 50 MHz (internal clock)
	Operating modes	Incremental encoder, event counter, frequency counter, pulse width measure, PWM generator, pulse generator, custom
Clock frequency	Internal	50 MHz, 50 ppm
	External (maximum)	Derate for deviations from 50% duty cycle: 6.25 MHz @ quadrature x4 12.5 MHz @ quadrature/mono x2 25 MHz @ quadrature/mono x1
Inputs (CLK, IX)	Receiver type	RS-422 differential
	Signal levels	Differential: RS-422, ±7 V CMV maximum Single ended: 5 V TTL/CMOS
	Noise/debounce filter	Interval: 0 to 1.3107 ms in 20 ns steps, common per channel. Enable/disable: independent IX, CLK pair.

Analog Inputs		
Channels	Number/type	16 differential
ADC	Resolution	16 bits
	Conversion time	≤ 3 µs
Input	Differential voltage measurement ranges	±1 V, ±2 V, ±5 V, ±10 V
	Absolute input voltage (signal + CMV, max)	±11 V off GND
	CMRR	> 80 dB @ 1 kHz, > 65 dB @ 10 kHz
	Input impedance	>10 MΩ in parallel with 100 pF
	Settling time for multichannel measurements	4µs max. @ < 1 kΩ input Z with no gain change
Triggering	Modes	Hardware: 48 external digital inputs, 6 internal counter outputs. Software: 6 virtual digital outputs. Untriggered (free-running).

Analog Outputs		
Channels	Number/type	8 single ended, with local (on-board) sense
DAC	Resolution	16 bits
	Conversion time (serializer transmission + analog conversion)	1.04 µs
Output	Voltage ranges	0 to +5 V, 0 to +10 V, ±5 V, ±10 V
	Load current (maximum)	2 mA
	Fail-safe mode	Yes

Watchdog Timer		
Timer stages	Number	3
	Interval (per stage)	Programmable from 1 to 2^{32}-1 * 20 ns (20 ns to ~85.9 s)
	Output events	Stage 0: Fail-safe trigger, IRQ Stage 1: NMI out via digital output, PCIe Fatal Error Stage 2: Reset via digital output or onboard solid state relay (SSR)
Solid state relay (Reset out)	On-state resistance (I_L = 10mA)	20 ohms typical, 30 Ω max.
	Off-state leakage current	0.03 µA typical, 1.0 µA max.
	Applied voltage (maximum)	200 V
	Load current (maximum)	100 mA

Power and Environmental		
Power	Input power	350 mA (+12V) and 450 mA (+3.3V), nominal, with no loads
	Encoder power out	5 VDC ±5%, 400 mA max. (total for all encoders)
Temperature	Operating	0 to 70^{o}C
Mating Connectors (not included)	Counters	Sullins SFH210-PPPC-D13-ID-BK or equivalent (qty. 2)
	Analog I/O	Sullins SFH210-PPPC-D25-ID-BK or equivalent (qty. 1)
	Digital I/O	Sullins SFH210-PPPC-D25-ID-BK or equivalent (qty. 2)

.

Appendix C

Getting Started V2.6.5-4-g7af4f1d, 2014-12-18

Contents

The LinuxCNC Team

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

Part I

Getting Started

Chapter 1

System Requirements

1.1 Minimum Requirements

The minimum system to run LinuxCNC and Ubuntu may vary depending on the exact usage. Stepper systems in general require faster threads to generate step pulses than servo systems. Using the Live-CD you can test the software before committing a computer. Keep in mind that the Latency Test numbers are more important than the processor speed for software step generation. More information on the Latency Test is here.

Additional information is on the LinuxCNC Wiki site:

Wiki.LinuxCNC.org, Hardware_Requirements

LinuxCNC and Ubuntu should run reasonably well on a computer with the following minimum hardware specification. These numbers are not the absolute minimum but will give reasonable performance for most stepper systems.

- 700 MHz x86 processor (1.2 GHz x86 processor recommended)
- 384 MB of RAM (512 MB up to 1 GB recommended)
- 8 GB hard disk
- Graphics card capable of at least 1024x768 resolution, which is not using the NVidia or ATI fglrx proprietary drivers, and which is not an onboard video chipset that shares main memory with the CPU
- A network or Internet connection (not strictly needed, but very useful for updates and for communicating with the LinuxCNC community)

Minimum hardware requirements change as Ubuntu evolves so check the Ubuntu web site for details on the LiveCD you're using. Older hardware may benefit from selecting an older version of the LiveCD when available.

1.2 Problematic Hardware

1.2.1 Laptops

Laptops are not generally suited to real time software step generation. Again a Latency Test run for an extended time will give you the info you need to determine suitability.

1.2.2 Video Cards

If your installation pops up with 800 x 600 screen resolution then most likely Ubuntu does not recognize your video card or monitor. Onboard video many times causes bad real time performance.

Chapter 2

About LinuxCNC

2.1 The Software

- LinuxCNC (the Enhanced Machine Control) is a software system for computer control of machine tools such as milling machines and lathes, robots such as puma and scara and other computer controlled machines up to 9 axes.
- LinuxCNC is free software with open source code. Current versions of LinuxCNC are entirely licensed under the GNU General Public License and Lesser GNU General Public License (GPL and LGPL)
- LinuxCNC provides:
  - a graphical user interface (actually several interfaces to choose from)
  - an interpreter for *G-code* (the RS-274 machine tool programming language)
  - a realtime motion planning system with look-ahead
  - operation of low-level machine electronics such as sensors and motor drives
  - an easy to use *breadboard* layer for quickly creating a unique configuration for your machine
  - a software PLC programmable with ladder diagrams
  - easy installation with a Live-CD
- It does not provide drawing (CAD - Computer Aided Design) or G-code generation from the drawing (CAM - Computer Automated Manufacturing) functions.
- It can simultaneously move up to 9 axes and supports a variety of interfaces.
- The control can operate true servos (analog or PWM) with the feedback loop closed by the LinuxCNC software at the computer, or open loop with step-servos or stepper motors.
- Motion control features include: cutter radius and length compensation, path deviation limited to a specified tolerance, lathe threading, synchronized axis motion, adaptive feedrate, operator feed override, and constant velocity control.
- Support for non-Cartesian motion systems is provided via custom kinematics modules. Available architectures include hexapods (Stewart platforms and similar concepts) and systems with rotary joints to provide motion such as PUMA or SCARA robots.
- LinuxCNC runs on Linux using real time extensions.

2.2 The Operating System

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

- Ubuntu will always be free of charge, and there is no extra fee for the *enterprise edition*, we make our very best work available to everyone on the same Free terms.
- LinuxCNC is paired with the LTS versions of Ubuntu which provide support and security fixes from the Ubuntu team for 3 - 5 years.
- Ubuntu uses the very best in translations and accessibility infrastructure that the Free Software community has to offer, to make Ubuntu usable for as many people as possible.
- The Ubuntu community is entirely committed to the principles of free software development; we encourage people to use open source software, improve it and pass it on.

2.3 Getting Help

2.3.1 IRC

IRC stands for Internet Relay Chat. It is a live connection to other LinuxCNC users. The LinuxCNC IRC channel is #linuxcnc on freenode.

The simplest way to get on the IRC is to use the embedded java client on this page.

Some IRC etiquette

- Ask specific questions... Avoid questions like "Can someone help me?".
- If you're really new to all this, think a bit about your question before typing it. Make sure you give enough information so someone can solve your question.
- Have some patience when waiting for an answer, sometimes it takes a while to formulate an answer or everyone might be busy working or something.
- Set up your IRC account with your unique name so people will know who you are. If you use the java client, use the same name every time you log in. This helps people remember who you are and if you have been on before many will remember the past discussions which saves time on both ends.

Sharing Files

The most common way to share files on the IRC is to upload the file to one of the following or a similar service and paste the link:

- *For text* - http://pastebin.com/ , http://pastie.org/, https://gist.github.com/
- *For pictures* - http://imagebin.org/ , http://imgur.com/ , http://bayimg.com/
- *For files* - https://filedropper.com/ , http://filefactory.com/ , http://1fichier.com/

2.3.2 Mailing List

An Internet Mailing List is a way to put questions out for everyone on that list to see and answer at their convenience. You get better exposure to your questions on a mailing list than on the IRC but answers take longer. In a nutshell you e-mail a message to the list and either get daily digests or individual replies back depending on how you set up your account.

The emc-users mailing list

2.3.3 LinuxCNC Wiki

A Wiki site is a user maintained web site that anyone can add to or edit.

The user maintained LinuxCNC Wiki site contains a wealth of information and tips at:

http://wiki.linuxcnc.org

2.4 Getting LinuxCNC

2.4.1 Normal Download

Download the Live CD from:

the LinuxCNC homepage at www.linuxcnc.org

and follow the Download link.

2.4.2 Multi-session Download

If the file is too large to download in one session because of a bad or slow Internet connection, use *wget* to allow resuming after an interrupted download.

Wget Linux

Open a terminal window. In Ubuntu it is Applications/Accessories/Terminal. Use *cd* to change to the directory where you would like to store the ISO. Use *mkdir* to create a new directory if needed.

Note that actual file names may change so you might have to go to http://www.linuxcnc.org/ and follow the Download link to get the actual file name. In most browsers you can right click on the link and select Copy Link Location or similar, then paste the link into the terminal window with a right mouse click and select Paste.

Debian Wheezy and LinuxCNC fresh install

The Debian image is a "hybrid" iso, which means you can use the same iso file for a USB stick or a DVD.

To get the Debian Wheezy LinuxCNC Live CD using wget copy one of this in a terminal window and press enter:

wget http://linuxcnc.org/binary.hybrid.iso

The md5sum of the above file is: *1d0025a8c1a61fbf32b99ab5797d95e3*

To continue a partial download that was interrupted add the -c option to wget:

wget -c http://linuxcnc.org/binary.hybrid.iso

To stop a download use Ctrl-C or close the terminal window.

Ubuntu 8.04 Hardy Heron and LinuxCNC (older)

For more information on other versions of LinuxCNC visit the linuxcnc.org download page.

http://linuxcnc.org/index.php/english/download

After the download is complete you will find the ISO file in the directory that you selected. Next we will burn the CD.

Wget Windows

The wget program is also available for Windows from:

http://gnuwin32.sourceforge.net/packages/wget.htm

Follow the instructions on the web page for downloading and installing the windows version of the wget program.

To run wget open a command prompt window.

In most Windows it is Programs/Accessories/Command Prompt

First you have to change to the directory where wget is installed in.

Typically it is in C:\Program Files\GnuWin32\bin so in the Command Prompt window type:

```
cd C:\Program Files\GnuWin32\bin
```

and the prompt should change to: *C:\Program Files\GnuWin32\bin>*

Type the wget command into the window and press enter as above.

2.4.3 Burning the CD

LinuxCNC is distributed as CD image files, called ISOs. To install LinuxCNC, you first need to burn the ISO file onto a CD. You need a working CD/DVD burner and an 80 minute (700 Mb) CD for this. If the CD writing fails, try writing at a slower burn speed.

Verify md5sum in Linux

Before burning a CD, it is highly recommended that you verify the md5sum (hash) of the .iso file.

Open a terminal window. In Ubuntu it is Applications/Accessories/Terminal.

Change to the directory where the ISO was downloaded to.

```
cd download_directory
```

Then run the md5sum command with the file name you saved.

```
md5sum -b binary.hybrid.iso
```

The md5sum should print out a single line after calculating the hash. On slower computers this might take a minute or two.

```
a9898e3397861c2cd0d707f9c27900de *binary.hybrid.iso
```

Now compare it to the md5sum value that it should be.

Burning the ISO in Linux

1. Insert a blank CD into your burner. A *CD/DVD Creator* or *Choose Disc Type* window will pop up. Close this, as we will not be using it.
2. Browse to the downloaded ISO image in the file browser.
3. Right click on the ISO image file and choose Write to Disc.
4. Select the write speed. If you are burning a Ubuntu Live CD, it is recommended that you write at the lowest possible speed.
5. Start the burning process.
6. If a *choose a file name for the disc image* window pops up, just pick OK.

Verify md5sum with Windows

Before burning a CD, it is highly recommended that you verify the md5 sum (hash) of the .iso file, to ensure that you got a good download.

Windows does not come with a md5sum program. You will have to download and install one to check the md5sum. More information can be found at:

https://help.ubuntu.com/community/HowToMD5SUM

Burning the ISO in Windows

1. Download and install Infra Recorder, a free and open source image burning program: http://infrarecorder.org/
2. Insert a blank CD in the drive and select Do nothing or Cancel if an auto-run dialog pops up.
3. Open Infra Recorder, and select the *Actions* menu, then *Burn image*.

2.4.4 Testing LinuxCNC

With the Live CD in the CD/DVD drive shut down the computer then turn the computer back on. This will boot the computer from the Live CD. Once the computer has booted up you can try out LinuxCNC without installing it. You can not create custom configurations or modify most system settings like screen resolution unless you install LinuxCNC.

To try out LinuxCNC from the Applications/CNC menu pick LinuxCNC. Then select a sim configuration to try out.

To see if your computer is suitable for software step pulse generation run the Latency Test as shown here.

2.4.5 Installing LinuxCNC

To install LinuxCNC from the LiveCD select 'Install (Graphical) at bootup.

2.4.6 Updates to LinuxCNC

With the normal install the Update Manager will notify you of updates to LinuxCNC when you go on line and allow you to easily upgrade with no Linux knowledge needed. If you want to upgrade to Debian Wheezy from 10.04 or 8.04 a clean install from the Live-CD is recommended. It is OK to upgrade everything except the operating system when asked to.

Warning: Do not upgrade the operating system if promped to do so.

2.4.7 Install Problems

In rare cases you might have to reset the BIOS to default settings if during the Live CD install it cannot recognize the hard drive during the boot up.

Chapter 3

Updating LinuxCNC

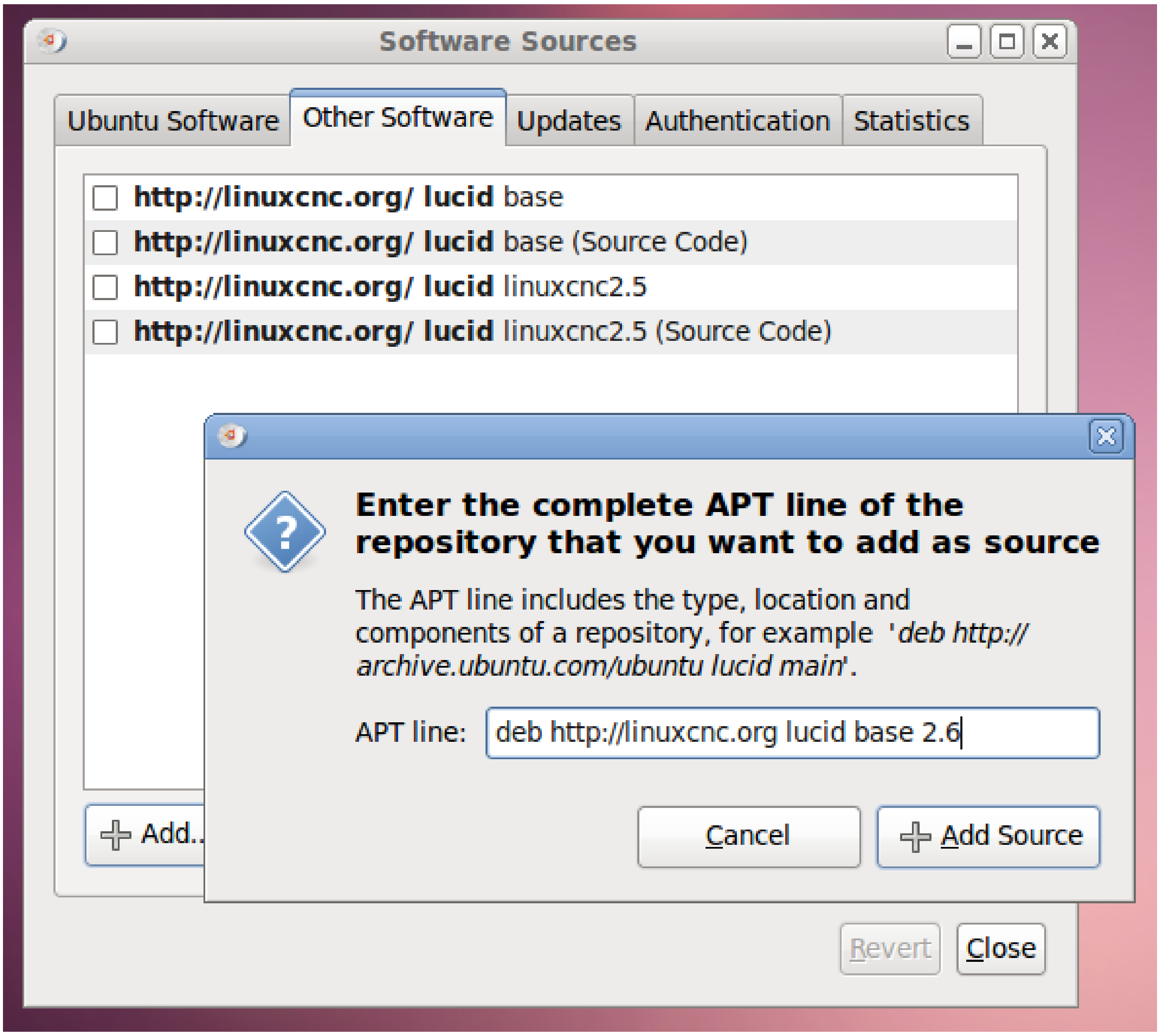
3.1 Updating from 2.5.x to 2.6.x

First you need to tell your computer where to find the new LinuxCNC software:

Click on the System menu in the top panel and select Administration->Software Sources.
Select the Other Software tab.
Disable or delete all the old linuxcnc.org entries.

Add a new Apt line that looks like this:

```
deb http://linuxcnc.org/ lucid base 2.6
```



http://linuxcnc.org/ lucid base

http://linuxcnc.org/ lucid base (Source Code)

http://linuxcnc.org/ lucid linuxcnc2.5

http://linuxcnc.org/ lucid linuxcnc2.5 (Source Code)

Enter the complete APT line of the repository that you want to add as source

The APT line includes the type, location and components of a repository, for example '*deb http://archive.ubuntu.com/ubuntu lucid main*'.

APT line: deb http://linuxcnc.org lucid base 2.6

When you add that line, Software Sources will pop up a window informing you that the information about available software is out-of-date. Click the Reload button.

Now your computer knows about the new software, next we need to tell it to install it:

- Click on the System menu in the top panel and select Administration->Synaptic Package Manager
- In the Quick Search bar at the top, type linuxcnc.
- Click the check box to mark the new linuxcnc package for installation.
- Click the Apply button, and let your computer install the new package. The old linuxcnc 2.5 package will be automatically upgraded to the new linuxcnc 2.6 package.

3.2 Updating from 2.4.x to 2.5.x

As of version 2.5.0, the name of the project has changed from EMC2 to LinuxCNC. All programs with "emc" in the name have been changed to "linuxcnc" instead. All documentation has been updated.

Additionally, the name of the debian package containing the software has changed. Unfortunately this breaks automatic upgrades. To upgrade from emc2 2.4.X to linuxcnc 2.5.X, do the following:

3.2.1 On Ubuntu Lucid 10.04

First you need to tell your computer where to find the new LinuxCNC software:

- Click on the System menu in the top panel and select Administration->Software Sources.
- Select the Other Software tab.
- Select the entry that says

  ```
  http://linuxcnc.org/lucid lucid base emc2.4

  or

  http://linuxcnc.org/lucid lucid base emc2.4-sim

  and click the Edit button.
  ```

- In the Components field, change `emc2.4` to `linuxcnc2.5`, or change `emc2.4-sim` to `linuxcnc2.5-sim`.
- Click the OK button.
- Back in the Software Sources window, Other Software tab, click the Close button.
- It will pop up a window informing you that the information about available software is out-of-date. Click the Reload button.

Now your computer knows about the new software, next we need to tell it to install it:

- Click on the System menu in the top panel and select Administration->Synaptic Package Manager
- In the Quick Search bar at the top, type `linuxcnc`.
- Click the check box to mark the new linuxcnc package for installation.
- Click the Apply button, and let your computer install the new package. The old emc 2.4 package will be automatically removed to make room for the new LinuxCNC 2.5 package.

3.2.2 On Ubuntu Hardy 8.04

First you need to tell your computer where to find the new LinuxCNC software:

- Click on the System menu in the top panel and select Administration->Synaptic Package Manager
- Go to Settings->Repositories.
- Select the "Third-Party Software" tab.
- Select the entry that says

  ```
  http://linuxcnc.org/hardy hard emc2.4

  or

  http://linuxcnc.org/hardy hardy emc2.4-sim
  ```

```
and click the Edit button.
```

- In the Components field, change `emc2.4` to `linuxcnc2.5` or `emc2.4-sim` to `linuxcnc2.5-sim`.
- Click the OK button.
- Back in the Software Sources window, click the Close button.
- Back in the Synaptic Package Manager window, click the Reload button.

Now your computer knows about the new software, next we need to tell it to install it:

- In the Synaptic Package Manager, click the Search button.
- In the Find dialog window that pops up, type `linuxcnc` and click the Search button.
- Click the check-box to mark the linuxcnc package for installation.
- Click the Apply button, and let your computer install the new package. The old emc 2.4 package will be automatically removed to make room for the new LinuxCNC 2.5 package.

3.3 Config changes

The user configs moved from $HOME/emc2 to $HOME/linuxcnc, so you will need to rename your directory, or move your files to the new place.

The hostmot2 watchdog in LinuxCNC 2.5 does not start running until the HAL threads start running. This means it now tolerates a timeout on the order of the servo thread period, instead of requiring a timeout that's on the order of the time between loading the driver and starting the HAL threads. This typically means a few milliseconds (a few times the servo thread period) instead of many hundreds of milliseconds. The default has been lowered from 1 second to 5 milliseconds. You generally don't need to set the hm2 watchdog timeout any more, unless you've changed your servo thread period.

The old driver for the Mesa 5i20, hal_m5i20, has been removed after being deprecated in favor of hostmot2 since early 2009 (version 2.3.) If you are still using this driver, you will need to build a new configuration using the hostmot2 driver. Pncconf may help you do this, and we have some sample configs (hm2-servo and hm2-stepper) that act as examples.

3.4 Upgrading from 2.3.x to 2.4.x

The following instructions only apply to Ubuntu 8.04 "Hardy Heron". LinuxCNC 2.4 is not available for older releases of Ubuntu.

Because there are several minor incompatibilities between 2.3.5 and 2.4.x, your existing install will not automatically be updated to 2.4.x. If you want to run 2.4.x, change to the LinuxCNC-2.4 repository by following these instructions:

run System/Administration/Synaptic Package Manager

go to Settings/Repositories

In the list of Third-Party software there should be at least two lines for linuxcnc.org.

For each of them:

- Select the line and click Edit
- On the Components line, change emc2.3 to emc2.4
- Click OK
- Close the *Software Preferences* window
- Click *Reload* as instructed

- Click *Mark All Upgrades*

Mesa card and hostmot2 users:

If you use a mesa card, find the proper hostmot2-firmware package for your card and mark it for installation. Hint: do a search for *hostmot2-firmware* in the synaptic package manager.

- Click *Apply*

3.5 Changes between 2.3.x and 2.4.x

Once you have done the upgrade, update any custom configurations by following these instructions:

3.5.1 emc.nml changes (2.3.x to 2.4.x)

For configurations that have not customized emc.nml, remove or comment out the inifile line NML_FILE = emc.nml. This will cause the most up to date version of emc.nml to be used.

For configurations that have customized emc.nml, a change similar to this one is required.

Failure to do this can cause an error like this one:

```
libnml/buffer/physmem.cc 143: PHYSMEM_HANDLE:
Can’t write 10748 bytes at offset 60 from buffer of size 10208.
```

3.5.2 tool table changes (2.3.x to 2.4.x)

The format of the tool table has been changed incompatibly. The documentation shows the new format. The tool table will automatically be converted to the new format.

3.5.3 hostmot2 firmware images (2.3.x to 2.4.x)

The hostmot2 firmware images are now a separate package. You can:

- Continue using an already-installed *emc2-firmware-mesa-** 2.3.x package
- Install the new packages from the synaptic package manager. The new packages are named *hostmot2-firmware-**
- Download the firmware images as tar files from http://emergent.unpy.net/01267622561 and install them manually

Chapter 4

Stepper Quickstart

This section assumes you have done a standard install from the Live CD. After installation it is recommended that you connect the computer to the Internet and wait for the update manager to pop up and get the latest updates for LinuxCNC and Ubuntu before continuing. For more complex installations see the Integrator Manual.

4.1 Latency Test

The Latency Test determines how late your computer processor is in responding to a request. Some hardware can interrupt the processing which could cause missed steps when running a CNC machine. This is the first thing you need to do. Follow the instructions here to run the latency test.

4.2 Sherline

If you have a Sherline several predefined configurations are provided. This is on the main menu CNC/EMC then pick the Sherline configuration that matches yours and save a copy.

4.3 Xylotex

If you have a Xylotex you can skip the following sections and go straight to the Stepper Config Wizard. LinuxCNC has provided quick setup for the Xylotex machines.

4.4 Machine Information

Gather the information about each axis of your machine.

Drive timing is in nano seconds. If you're unsure about the timing many popular drives are included in the stepper configuration wizard. Note some newer Gecko drives have different timing than the original one. A list is also on the user maintained LinuxCNC wiki site of more drives.

Axis	**Drive Type**	**Step Time ns**	**Step Space ns**	**Dir. Hold ns**	**Dir. Setup ns**
X					
Y					
Z					

4.5 Pinout Information

Gather the information about the connections from your machine to the PC parallel port.

Output Pin	Typ. Function	If Different	Input Pin	Typ. Function	If Different
1	E-Stop Out		10	X Limit/Home	
2	X Step		11	Y Limit/Home	
3	X Direction		12	Z Limit/Home	
4	Y Step		13	A Limit/Home	
5	Y Direction		15	Probe In	
6	Z Step				
7	Z Direction				
8	A Step				
9	A Direction				
14	Spindle CW				
16	Spindle PWM				
17	Amplifier Enable				

Note any pins not used should be set to Unused in the drop down box. These can always be changed later by running Stepconf again.

4.6 Mechanical Information

Gather information on steps and gearing. The result of this is steps per user unit which is used for SCALE in the .ini file.

Axis	Steps/Rev.	Micro Steps	Motor Teeth	Leadscrew Teeth	Leadscrew Pitch
X					
Y					
Z					

- *Steps per revolution* - is how many stepper-motor-steps it takes to turn the stepper motor one revolution. Typical is 200.
- *Micro Steps* - is how many steps the drive needs to move the stepper motor one full step. If microstepping is not used, this number will be 1. If microstepping is used the value will depend on the stepper drive hardware.
- *Motor Teeth and Leadscrew Teeth* - is if you have some reduction (gears, chain, timing belt, etc.) between the motor and the leadscrew. If not, then set these both to 1.
- *Leadscrew Pitch* - is how much movement occurs (in user units) in one leadscrew turn. If you're setting up in inches then it is inches per turn. If you're setting up in millimeters then it is millimeters per turn.

The net result you're looking for is how many CNC-output-steps it takes to move one user unit (inches or mm).

Example 4.1 Units inches

```
Stepper         = 200 steps per revolution
Drive           =  10 micro steps per step
Motor Teeth     =  20
Leadscrew Teeth =  40
Leadscrew Pitch =   0.2000 inches per turn
```

From the above information, the leadscrew moves 0.200 inches per turn. - The motor turns 2.000 times per 1 leadscrew turn. - The drive takes 10 microstep inputs to make the stepper step once. - The drive needs 2000 steps to turn the stepper one revolution. So the scale needed is:

$$\frac{200 motor\, steps}{1 motor rev} \times \frac{10 microsteps}{1 motor\, step} \times \frac{2 motor revs}{1 leadscrew rev} \times \frac{1 leadscrew revs}{0.2000 inch} = \frac{20{,}000 microsteps}{inch}$$

Example 4.2 Units mm

```
    Stepper         = 200 steps per revolution
    Drive           =   8 micro steps per step
    Motor Teeth     =  30
    Leadscrew Teeth =  90
    Leadscrew Pitch =   5.00 mm per turn
```

From the above information: - The leadscrew moves 5.00 mm per turn. - The motor turns 3.000 times per 1 leadscrew turn. - The drive takes 8 microstep inputs to make the stepper step once. - The drive needs 1600 steps to turn the stepper one revolution. So the scale needed is:

$$\frac{200\, full\, steps}{1\, rev} x \frac{8\, microsteps}{1\, step} x \frac{3\, revs}{1\, leadscrew\, rev} x \frac{1\, leadscrew\, rev}{5.00\mathrm{mm}} = \frac{960\, steps}{1\, mm}$$

Chapter 5

Stepper Configuration Wizard

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

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

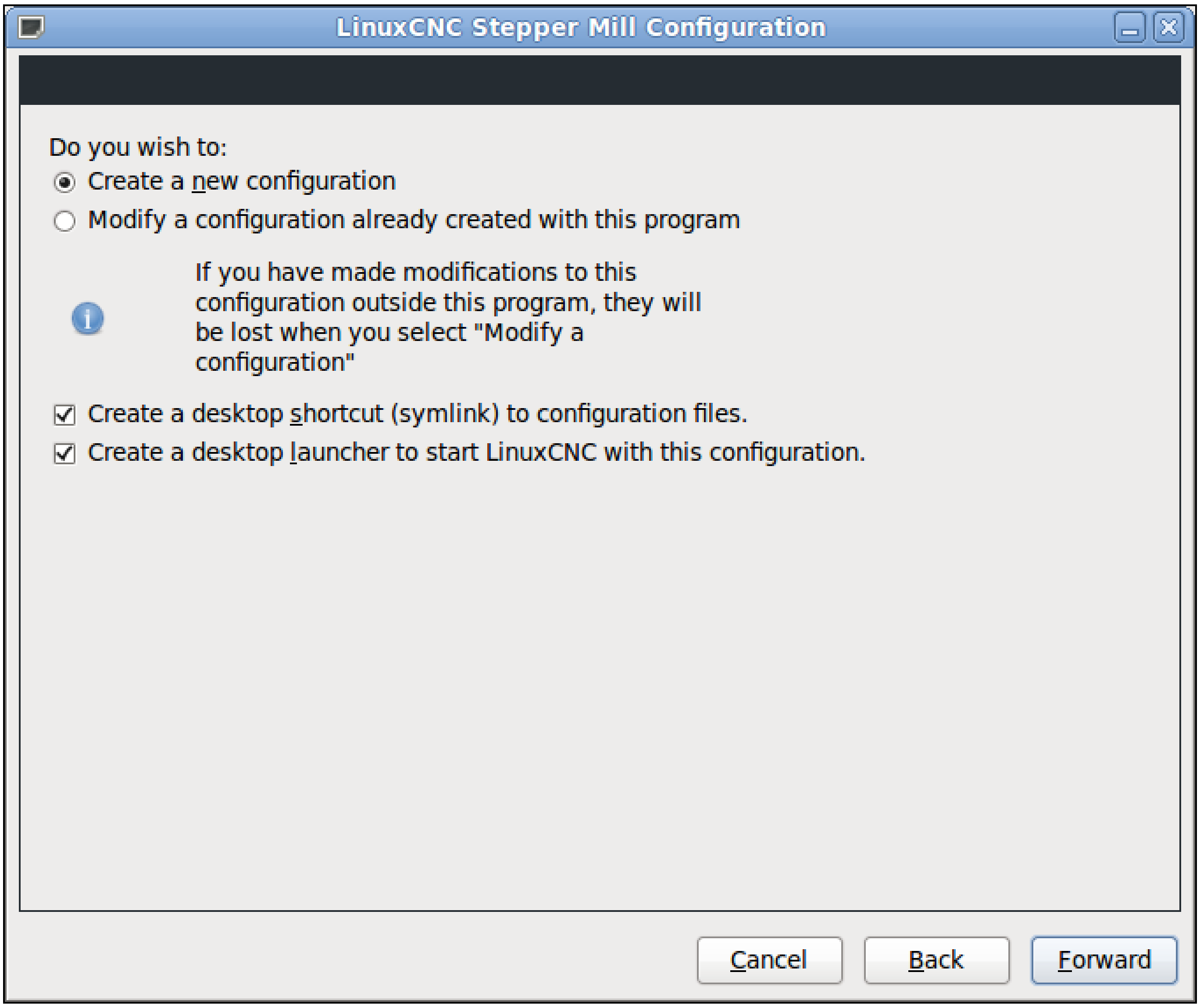
Stepconf is installed when you install LinuxCNC and is in the CNC menu.

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

The Stepconf Wizard needs at least 800 x 600 screen resolution to see the buttons on the bottom of the pages.

Step by Step Instructions

5.1 Entry Page


LinuxCNC Stepper Mill Configuration

Do you wish to:

● Create a new configuration

○ Modify a configuration already created with this program

If you have made modifications to this configuration outside this program, they will be lost when you select "Modify a configuration"

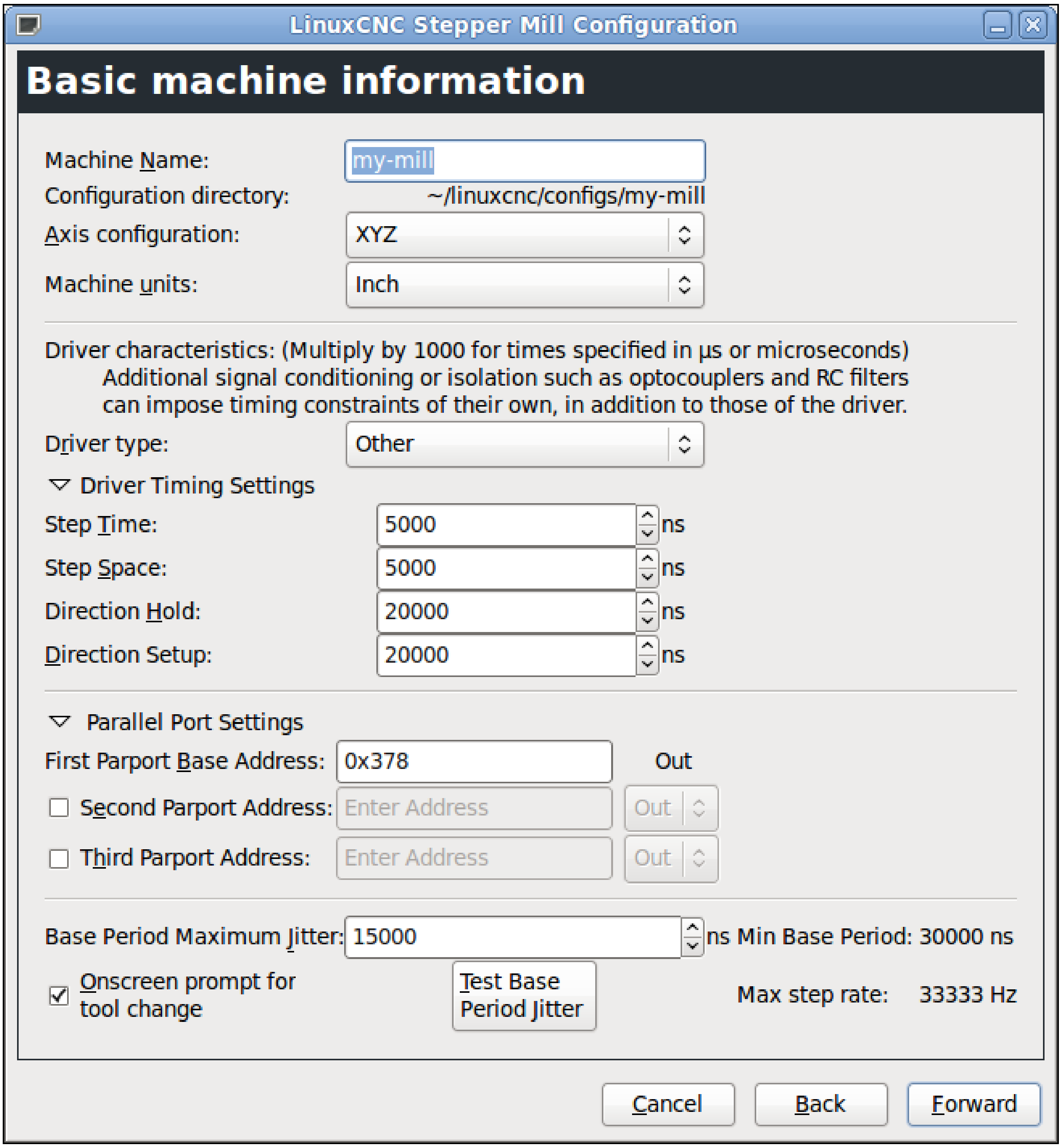
✓ Create a desktop shortcut (symlink) to configuration files.

✓ Create a desktop launcher to start LinuxCNC with this configuration.

Figure 5.1: Entry Page

- *Create New* - Creates a fresh configuration.
- *Modify* - Modify an existing configuration. After selecting this a file picker pops up so you can select the .stepconf file for modification. If you made any modifications to the main .hal or the .ini file these will be lost. Modifications to custom.hal and custom_postgui.hal will not be changed by the Stepconf Wizard.
- *Create Desktop Shortcut* - This will place a link on your desktop to the files.
- *Create Desktop Launcher* - This will place a launcher on your desktop to start your application.

5.2 Basic Information



Basic machine information

Machine Name: my-mill
Configuration directory: ~/linuxcnc/configs/my-mill
Axis configuration: XYZ
Machine units: Inch

Driver characteristics: (Multiply by 1000 for times specified in µs or microseconds)
Additional signal conditioning or isolation such as optocouplers and RC filters
can impose timing constraints of their own, in addition to those of the driver.

Driver type: Other

Driver Timing Settings
Step Time: 5000 ns
Step Space: 5000 ns
Direction Hold: 20000 ns
Direction Setup: 20000 ns

Parallel Port Settings
First Parport Base Address: 0x378 Out
☐ Second Parport Address: Enter Address Out
☐ Third Parport Address: Enter Address Out

Base Period Maximum Jitter: 15000 ns Min Base Period: 30000 ns
☑ Onscreen prompt for tool change
Max step rate: 33333 Hz

Figure 5.2: Basic Information Page

- *Machine Name* - Choose a name for your machine. Use only uppercase letters, lowercase letters, digits, - and _.
- *Axis Configuration* - Choose XYZ (Mill), XYZA (4-axis mill) or XZ (Lathe).

- *Machine Units* - Choose Inch or mm. All subsequent entries will be in the chosen units
- *Driver Type* - If you have one of the stepper drivers listed in the pull down box, choose it. Otherwise, select *Other* and find the timing values in your driver's data sheet and enter them as *nano seconds* in the *Driver Timing Settings*. If the data sheet gives a value in microseconds, multiply by 1000. For example, enter 4.5us as 4500ns.

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

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

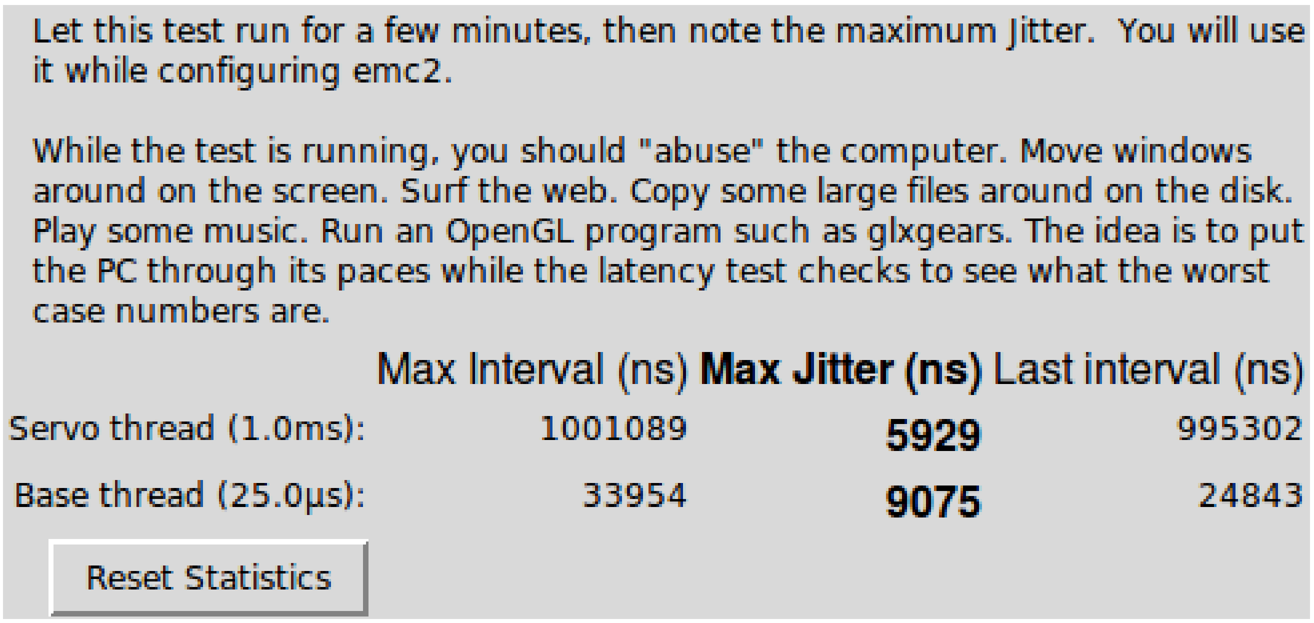
The LinuxCNC Configuration Selector has configs for Sherline already configured.

- *Step Time* - How long the step pulse is *on* in nano seconds. If your not sure about this setting a value of 20,000 will work with most drives.
- *Step Space* - Minimum time between step pulses in nano seconds. If your not sure about this setting a value of 20,000 will work with most drives.
- *Direction Hold* - How long the direction pin is held after a change of direction in nanoseconds. If your not sure about this setting a value of 20,000 will work with most drives.
- *Direction Setup* - How long before a direction change after the last step pulse in nanoseconds. If your not sure about this setting a value of 20,000 will work with most drives.
- *First Parport* - Usually the default of 0x378 is correct.
- *Second Parport* - If you need to specify additional parallel ports enter the address and the type. For information on finding the address of PCI parallel ports see the Port Address in the Integrator Manual. (Try 0x278 or 0x3BC first.)
- *Base Period Maximum Jitter* - Enter the result of the Latency Test here. To run a latency test press the *Test Base Period Jitter* button. See the Latency Test section for more details.
- *Max Step Rate* -Stepconf automatically calculates the Max Step Rate based on the driver characteristics entered and the latency test result.
- *Min Base Period* - Stepconf automatically determines the Min Base Period based on the driver characteristics entered and latency test result.
- *Onscreen Prompt For Tool Change* - If this box is checked, LinuxCNC will pause and prompt you to change the tool when *M6* is encountered. This feature is usually only useful if you have presettable tools.

5.3 Latency Test

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

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



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

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

	Max Interval (ns)	**Max Jitter (ns)**	Last interval (ns)
Servo thread (1.0ms):	1001089	**5929**	995302
Base thread (25.0µs):	33954	**9075**	24843

Figure 5.3: Latency Test

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

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

Troubleshooting SMI Issues (LinuxCNC.org Wiki)

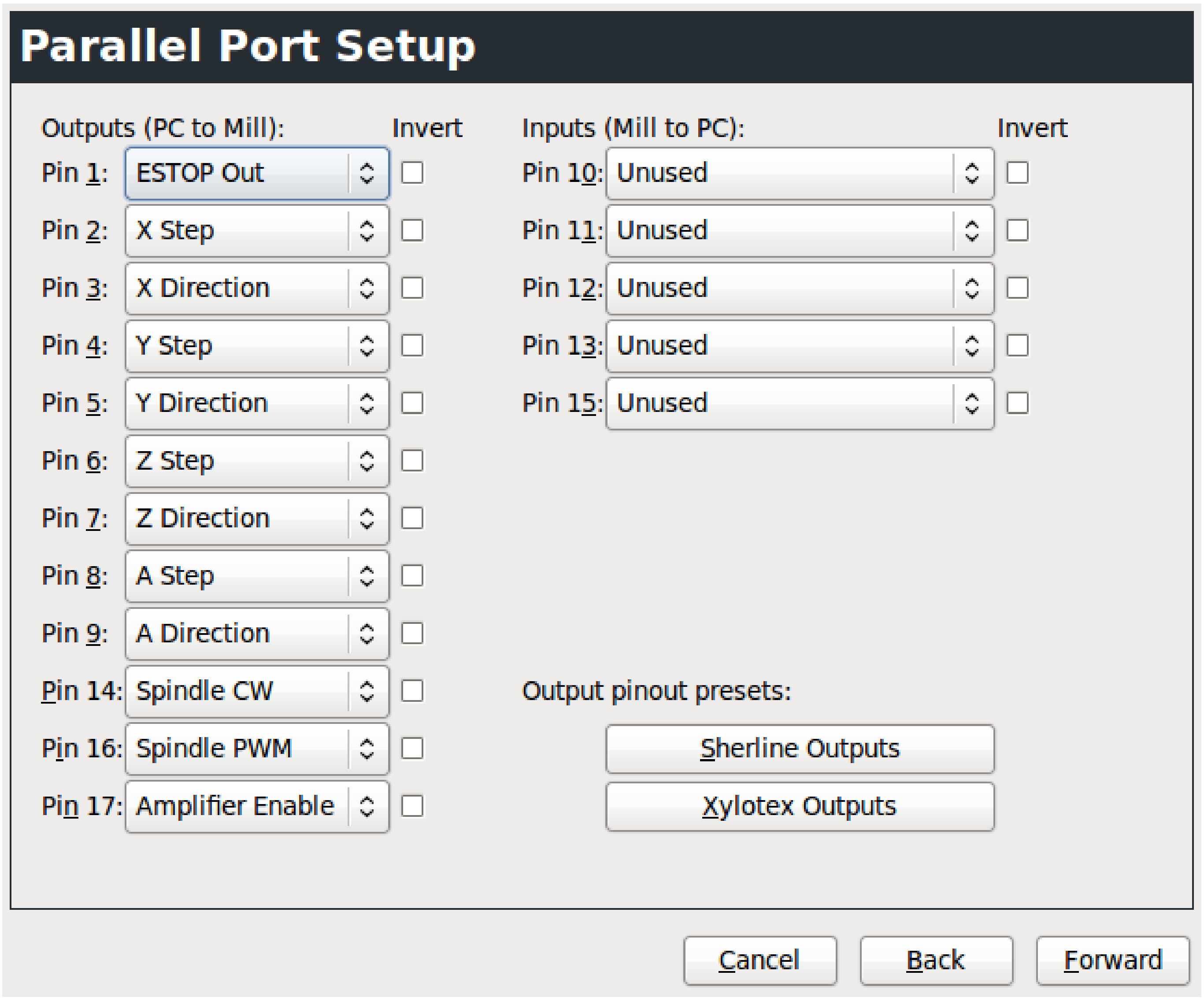
Fixing Realtime problems caused by SMI on Ubuntu

http://wiki.linuxcnc.org/cgi-bin/wiki.pl?FixingSMIIssues

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

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

5.4 Parallel Port Setup



Parallel Port Setup

Outputs (PC to Mill):		Invert
Pin 1:	ESTOP Out	☐
Pin 2:	X Step	☐
Pin 3:	X Direction	☐
Pin 4:	Y Step	☐
Pin 5:	Y Direction	☐
Pin 6:	Z Step	☐
Pin 7:	Z Direction	☐
Pin 8:	A Step	☐
Pin 9:	A Direction	☐
Pin 14:	Spindle CW	☐
Pin 16:	Spindle PWM	☐
Pin 17:	Amplifier Enable	☐

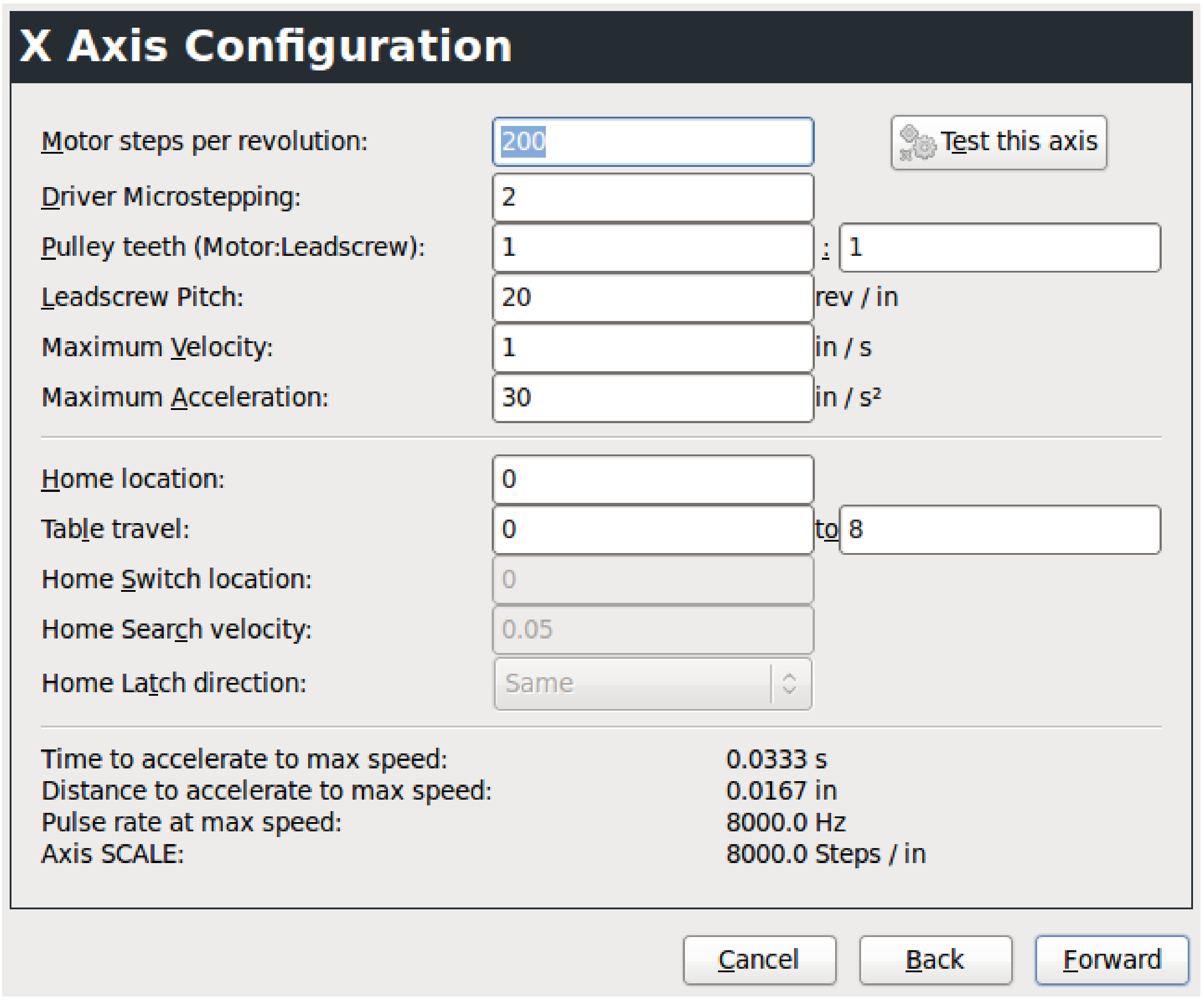
Inputs (Mill to PC):		Invert
Pin 10:	Unused	☐
Pin 11:	Unused	☐
Pin 12:	Unused	☐
Pin 13:	Unused	☐
Pin 15:	Unused	☐

Figure 5.4: Parallel Port Setup Page

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

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

5.5 Axis Configuration


X Axis Configuration

Motor steps per revolution:	200		Test this axis
Driver Microstepping:	2		
Pulley teeth (Motor:Leadscrew):	1	:	1
Leadscrew Pitch:	20	rev / in	
Maximum Velocity:	1	in / s	
Maximum Acceleration:	30	in / s²	
Home location:	0		
Table travel:	0	to	8
Home Switch location:	0		
Home Search velocity:	0.05		
Home Latch direction:	Same		

```
Time to accelerate to max speed:        0.0333 s
Distance to accelerate to max speed:    0.0167 in
Pulse rate at max speed:                8000.0 Hz
Axis SCALE:                             8000.0 Steps / in
```

Figure 5.5: Axis Configuration Page

- *Motor Steps Per Revolution* - The number of full steps per motor revolution. If you know how many degrees per step the motor is (e.g., 1.8 degree), then divide 360 by the degrees per step to find the number of steps per motor revolution.
- *Driver Microstepping* - The amount of microstepping performed by the driver. Enter *2* for half-stepping.
- *Pulley Ratio* - If your machine has pulleys between the motor and leadscrew, enter the ratio here. If not, enter *1:1*.
- *Leadscrew Pitch* - Enter the pitch of the leadscrew here. If you chose *Inch* units, enter the number of threads per inch If you chose *mm* units, enter the number of millimeters per revolution (e.g., enter 2 for 2mm/rev). If the machine travels in the wrong direction, enter a negative number here instead of a positive number, or invert the direction pin for the axis.
- *Maximum Velocity* -Enter the maximum velocity for the axis in units per second.
- *Maximum Acceleration* - The correct values for these items can only be determined through experimentation. See Finding Maximum Velocity to set the speed and Finding Maximum Acceleration to set the acceleration.

- *Home Location* - The position the machine moves to after completing the homing procedure for this axis. For machines without home switches, this is the location the operator manually moves the machine to before pressing the Home button. If you combine the home and limit switches you must move off of the switch to the home position or you will get a joint limit error.
- *Table Travel* - The range of travel for that axis based on the machine origin. The home location must be inside the *Table Travel* and not equal to one of the Table Travel values.
- *Home Switch Location* - The location at which the home switch trips or releases reletive to the machine origin. This item and the two below only appear when Home Switches were chosen in the Parallel Port Pinout. If you combine home and limit switches the home switch location can not be the same as the home position or you will get a joint limit error.
- *Home Search Velocity* - The velocity to use when searching for the home switch. If the switch is near the end of travel, this velocity must be chosen so that the axis can decelerate to a stop before hitting the end of travel. If the switch is only closed for a short range of travel (instead of being closed from its trip point to one end of travel), this velocity must be chosen so that the axis can decelerate to a stop before the switch opens again, and homing must always be started from the same side of the switch. If the machine moves the wrong direction at the beginning of the homing procedure, negate the value of *Home Search Velocity*.
- *Home Latch Direction* - Choose *Same* to have the axis back off the switch, then approach it again at a very low speed. The second time the switch closes, the home position is set. Choose *Opposite* to have the axis back off the switch and when the switch opens, the home position is set.
- *Time to accelerate to max speed* - Time to reach maximum speed calculated from *Max Acceleration* and *Max Velocity*.
- *Distance to accelerate to max speed* - Distance to reach maximum speed from a standstill.
- *Pulse rate at max speed* - Information computed based on the values entered above. The greatest *Pulse rate at max speed* determines the *BASE_PERIOD*. Values above 20000Hz may lead to slow response time or even lockups (the fastest usable pulse rate varies from computer to computer)
- *Axis SCALE* - The number that will be used in the ini file [SCALE] setting. This is how many steps per user unit.
- *Test this axis* - This will open a window to allow testing for each axis. This can be used after filling out all the information for this axis.

5.5.1 Test This Axis

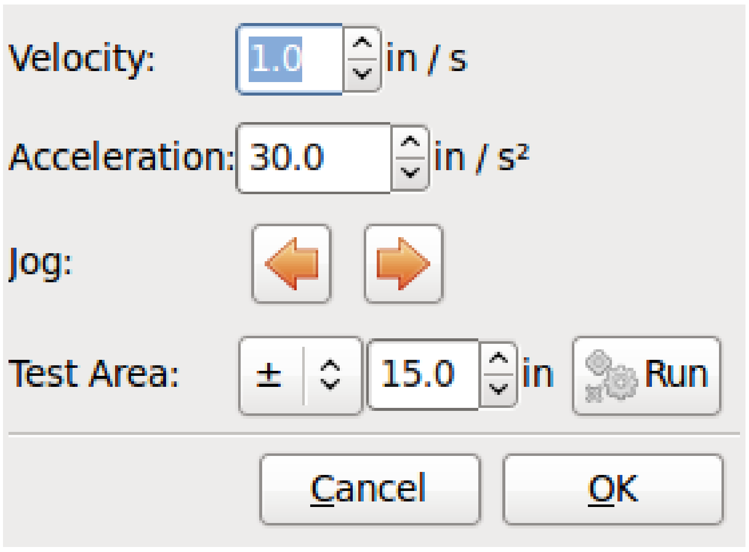

Velocity: 1.0 in / s
Acceleration: 30.0 in / s²
Jog:
Test Area: ± 15.0 in

Figure 5.6: Test This Axis

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

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

5.5.1.1 Finding Maximum Velocity

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

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

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

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

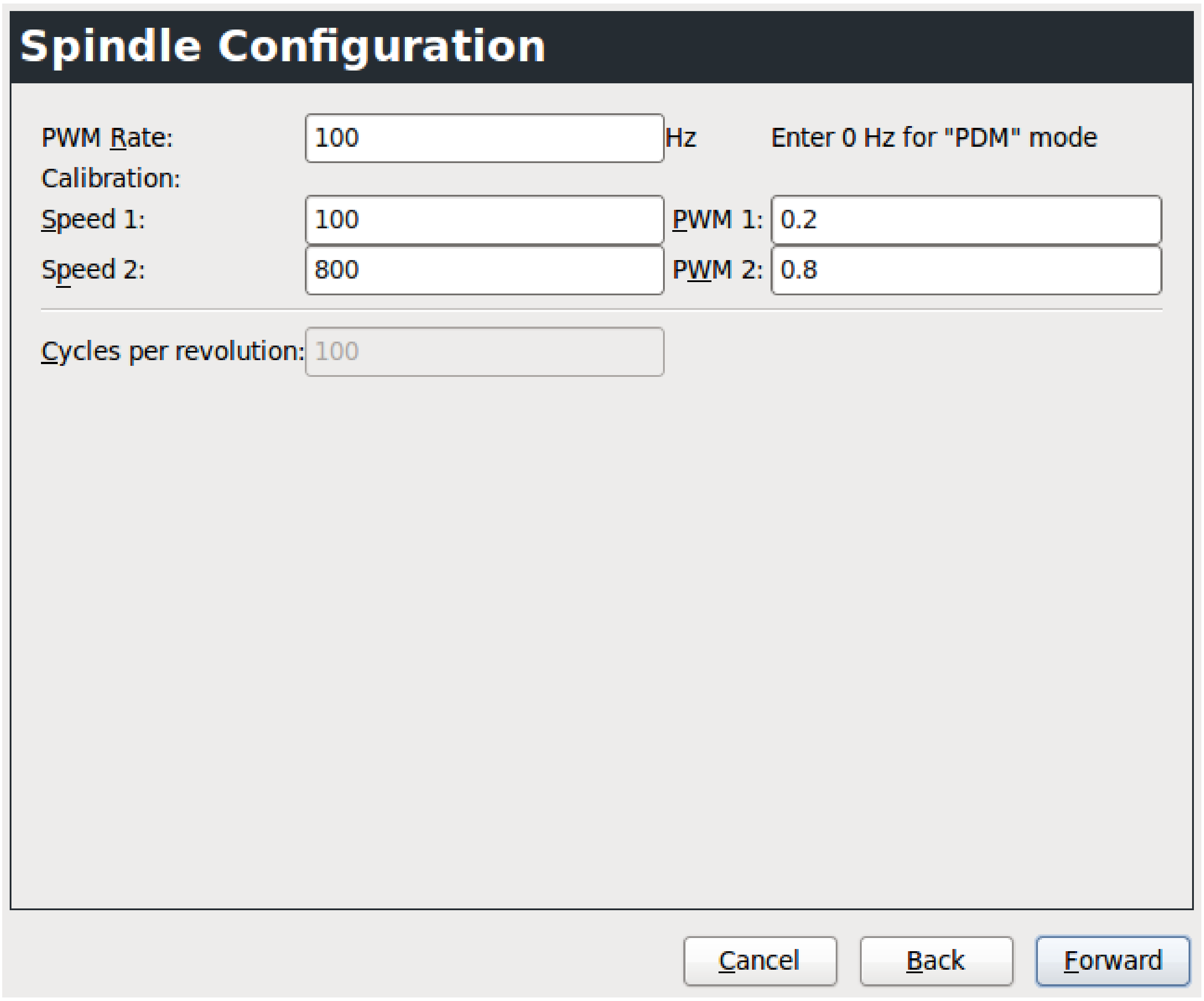
- Correct step waveform timings
- Correct pinout, including *Invert* on step pins
- Correct, well-shielded cabling
- Physical problems with the motor, motor coupling, leadscrew, etc.

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

5.5.1.2 Finding Maximum Acceleration

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

5.6 Spindle Configuration



PWM Rate: 100 Hz Enter 0 Hz for "PDM" mode

Calibration:

Speed 1:	100	PWM 1:	0.2
Speed 2:	800	PWM 2:	0.8

Cycles per revolution: 100

Figure 5.7: Spindle Configuration Page

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

5.6.1 Spindle Speed Control

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

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

5.6.2 Spindle-synchronized motion

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

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

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

- *Cycles per revolution* - The number of cycles of the *Spindle A* signal during one revolution of the spindle. This option is only enabled when an input has been set to *Spindle Phase A*
- *Maximum speed in thread* - The maximum spindle speed used in threading. For a high spindle RPM or a spindle encoder with high resolution, a low value of *BASE_PERIOD* is required.

5.6.3 Determining Spindle Calibration

Enter the following values in the Spindle Configuration page:

Speed 1:	0	PWM 1:	0
Speed 2:	1000	PWM 2:	1

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

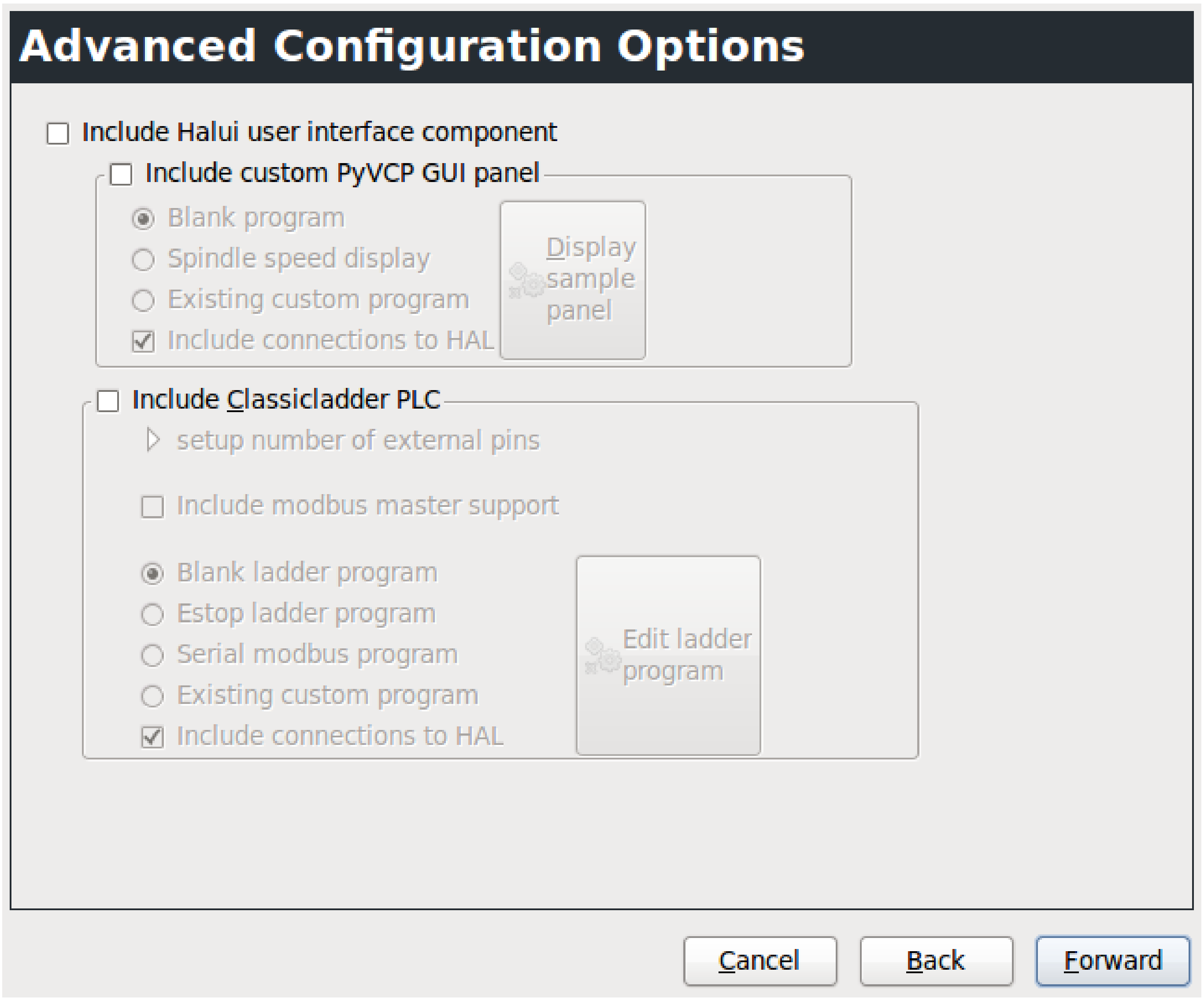
For two different S-numbers, measure the actual spindle speed in RPM. Record the S-numbers and actual spindle speeds. Run Stepconf again. For *Speed* enter the measured speed, and for *PWM* enter the S-number divided by 1000.

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

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

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

5.7 Advanced Configuration Options


Advanced Configuration Options

☐ Include Halui user interface component

☐ Include custom PyVCP GUI panel
- ● Blank program
- ○ Spindle speed display
- ○ Existing custom program
- ✓ Include connections to HAL

☐ Include Classicladder PLC
- setup number of external pins
- ☐ Include modbus master support
- ● Blank ladder program
- ○ Estop ladder program
- ○ Serial modbus program
- ○ Existing custom program
- ✓ Include connections to HAL

Figure 5.8: Advanced Configuration

- *Include Halui* - This will add the Halui user interface component. See the Integrator Manual for more information on Halui.
- *Include pyVCP* - This option adds the pyVCP panel base file or a sample file to work on. See the Integrator Manual for more information on pyVCP.
- *Include ClassicLadder PLC* - This option will add the ClassicLadder PLC (Programmable Logic Controller). See the Integrator Manual for more information on ClassicLadder.

5.8 Machine Configuration Complete

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

5.9 Axis Travel, Home Location, and Home Switch Location

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

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

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

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

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

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

5.9.1 Operating without Limit Switches

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

5.9.2 Operating without Home Switches

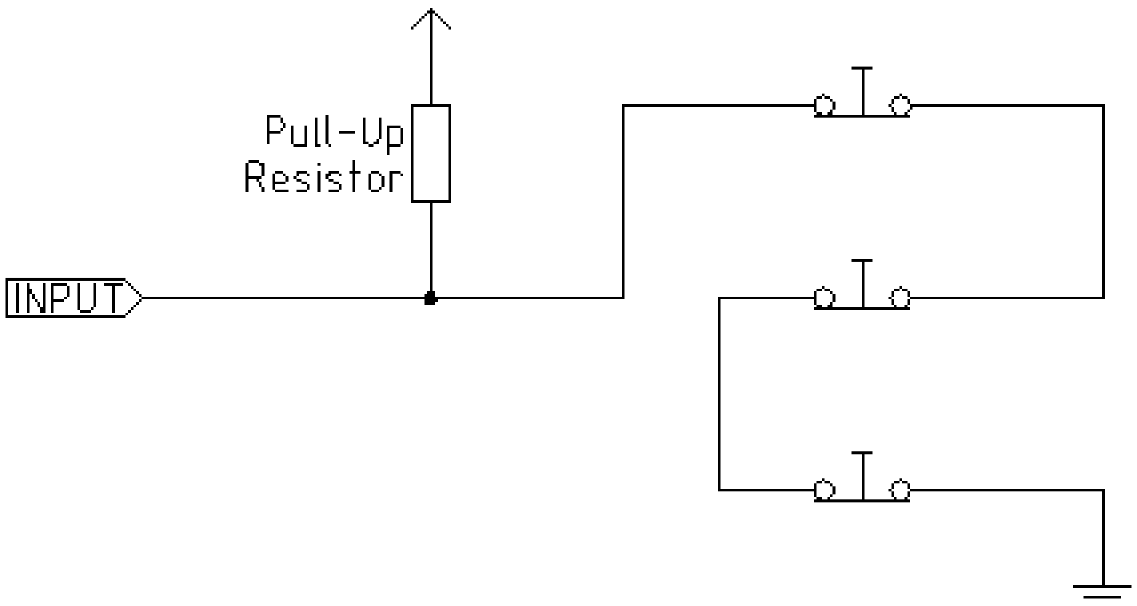
A machine can be operated without home switches. If the machine has limit switches, but no home switches, it is best to use a limit switch as the home switch (e.g., choose *Minimum Limit + Home X* in the pinout). If the machine has no switches at all, or the limit switches cannot be used as home switches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

5.9.3 Home and Limit Switch wiring options

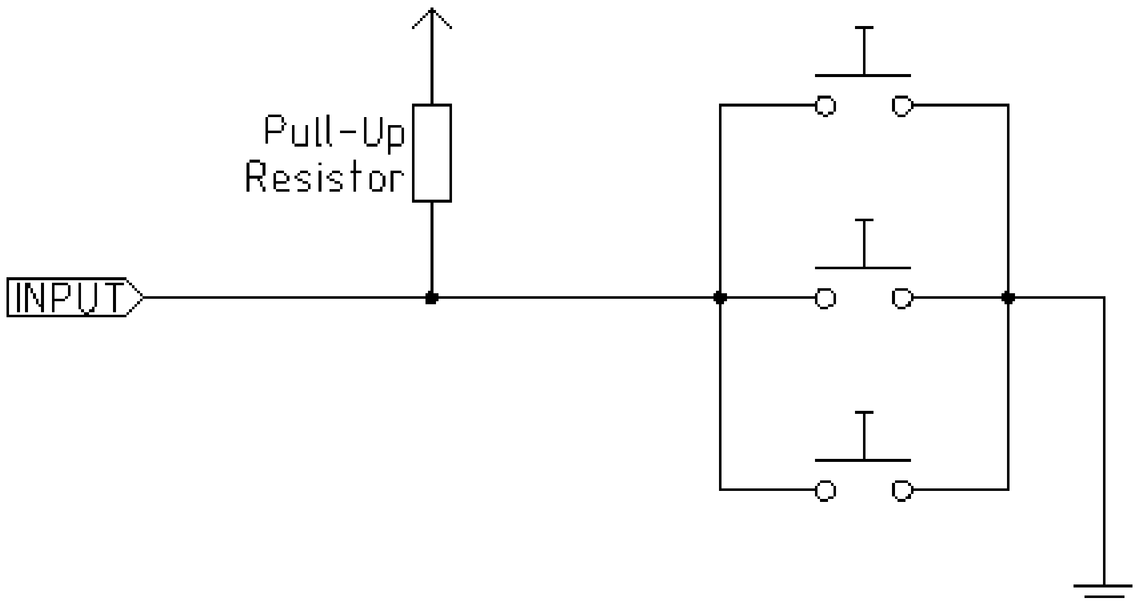
The ideal wiring for external switches would be one input per switch. However, the PC parallel port only offers a total of 5 inputs, while there are as many as 9 switches on a 3-axis machine. Instead, multiple switches are wired together in various ways so that a smaller number of inputs are required.

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

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



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



The following combinations of switches are permitted in Stepconf:

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

Chapter 6

Mesa Configuration Wizard

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

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

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

There are two trains of thought when using PNCconf:

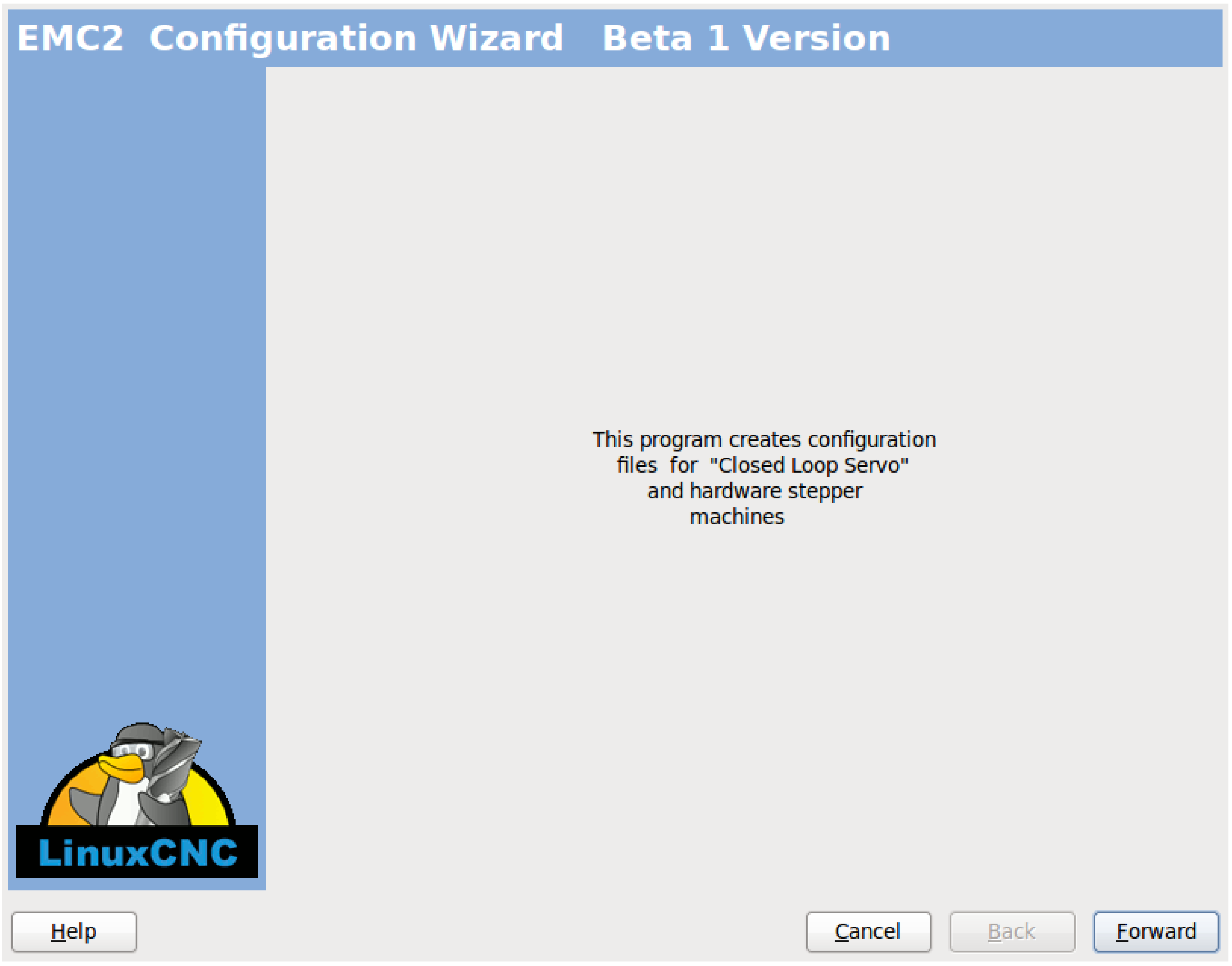
One is to use PNCconf to always configure your system - if you decide to change options, reload PNCconf and allow it to configure the new options. This will work well if your machine is fairly standard and you can use custom files to add non standard features. PNCconf tries to work with you in this regard.

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

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

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

Step by Step Instructions



This program creates configuration
files for "Closed Loop Servo"
and hardware stepper
machines

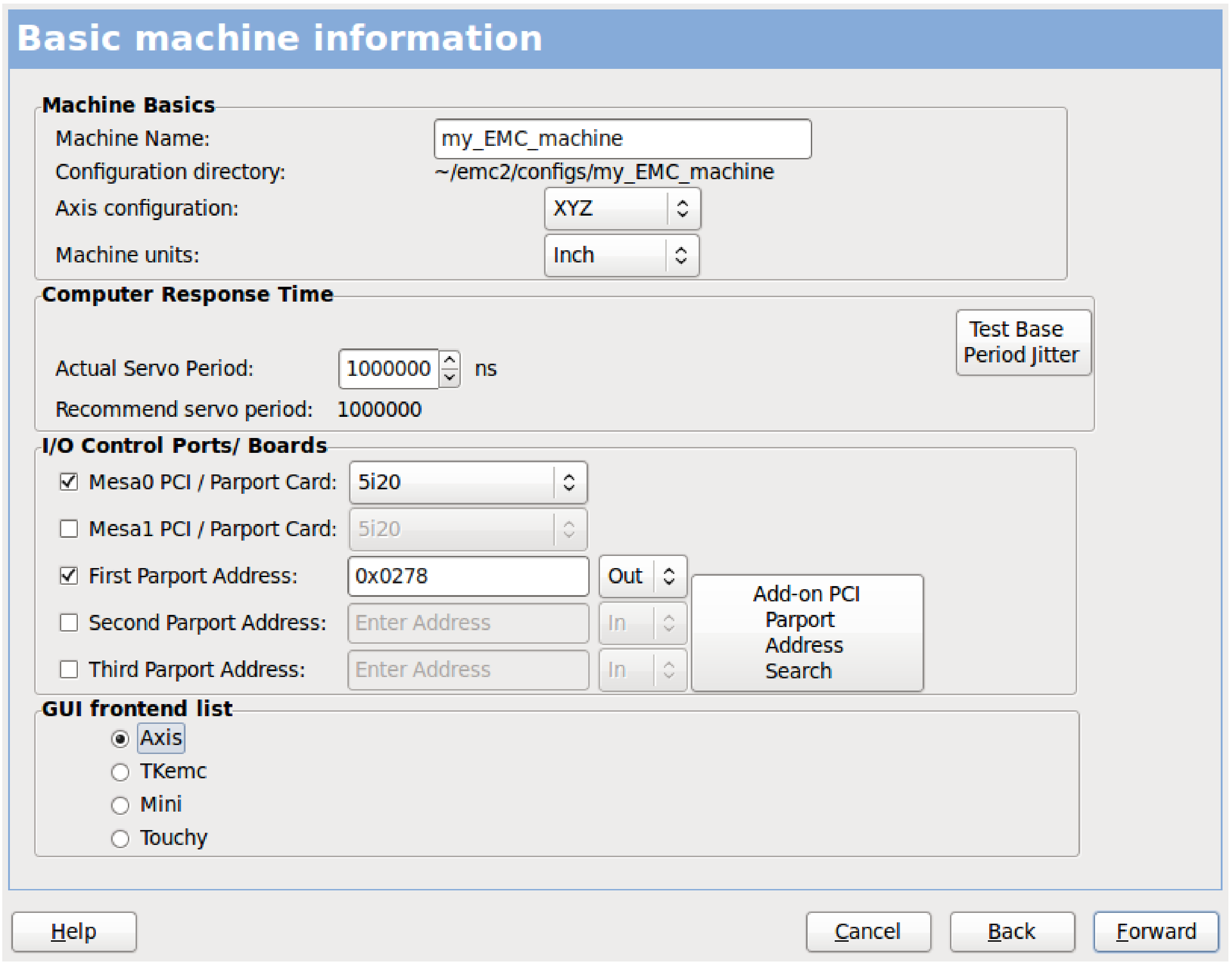
Figure 6.1: PnCConf Splash

6.1 Create or Edit

This allows you to select a previously saved configuration or create a new one. If you pick *Modify a configuration* and then press next a file selection box will show. Pncconf preselects your last saved file. Choose the the config you wish to edit. It also allows you to select desktop shortcut / launcher options. A desktop shortcut will place a folder icon on the desktop that points to your new configuration files. Otherwise you would have to look in your home folder under emc2/configs.

A Desktop launcher will add an icon to the desktop for starting your config directly. You can also launch it under Applications/cnc/emc2 and selecting your config name.

6.2 Basic Machine Information



Basic machine information

Machine Basics

Machine Name: my_EMC_machine

Configuration directory: ~/emc2/configs/my_EMC_machine

Axis configuration: XYZ

Machine units: Inch

Computer Response Time

Actual Servo Period: 1000000 ns

Recommend servo period: 1000000

I/O Control Ports/ Boards

☑ Mesa0 PCI / Parport Card: 5i20

☐ Mesa1 PCI / Parport Card: 5i20

☑ First Parport Address: 0x0278 Out

☐ Second Parport Address: Enter Address In

☐ Third Parport Address: Enter Address In

GUI frontend list

[✓ Axis] TKemc Mini Touchy

Figure 6.2: PnCConf Basic

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

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

Computer Response Time
The servo period sets the heart beat of the system. Latency refers to the amount of time the computer can be longer then that period. Just like a railroad, LinuxCNC requires everything on a very tight and consistent time line or bad things happen. LinuxCNC requires and uses a *real time* operating system, which just means it has a low latency (lateness)

response time when LinuxCNC requires its calculations and when doing LinuxCNCs calculations it cannot be interrupted by lower priority requests (such as user input to screen buttons or drawing etc).

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

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

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

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

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

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

GUI Frontend list
This specifies the graphical display screens LinuxCNC will use. Each one has different option.

AXIS

- fully supports lathes.

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

TOUCHY

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

MINI

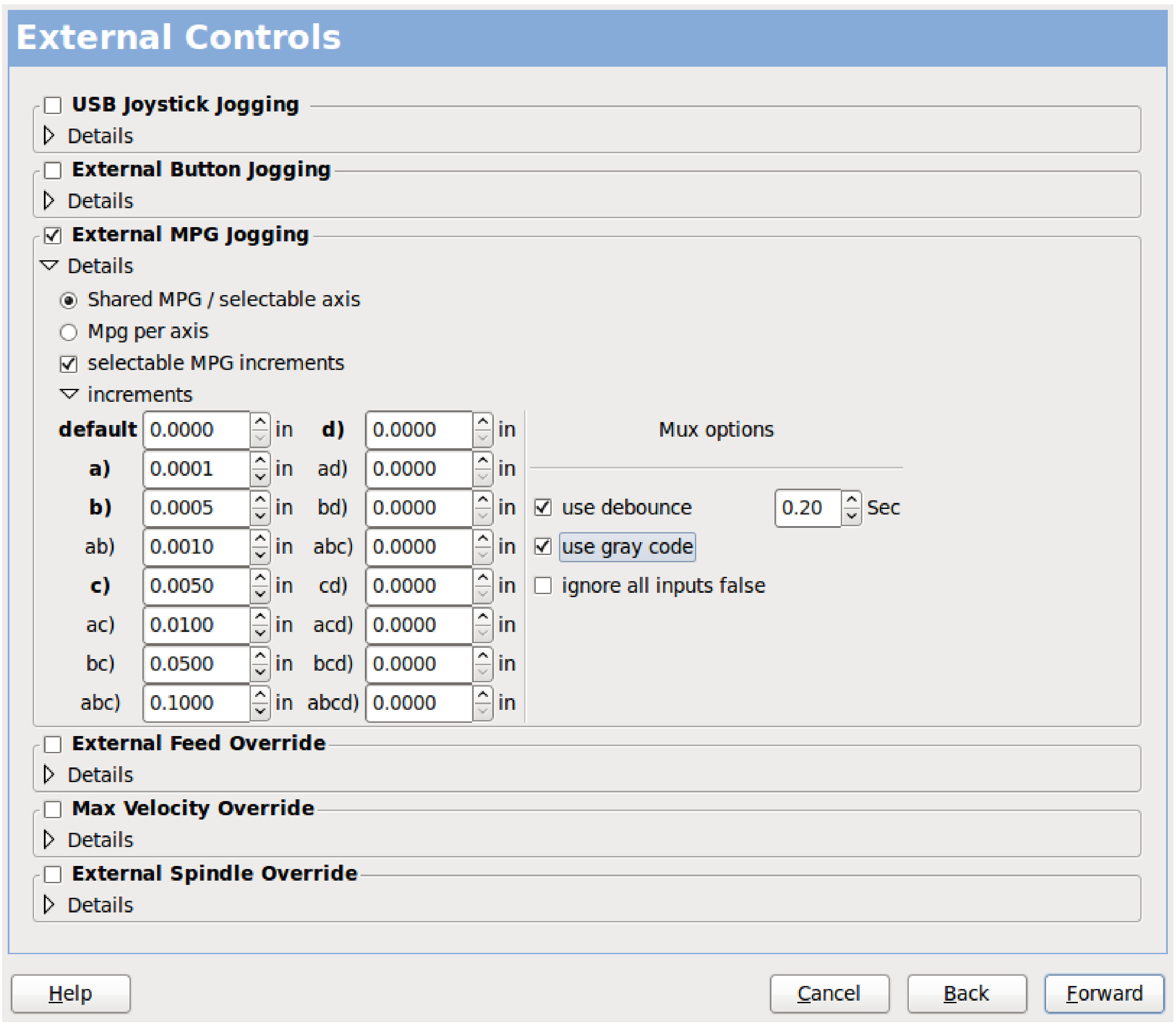
- standard on OEM Sherline machines
- does not use Estop
- no VCP integration

TkLinuxCNC

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

6.3 External Configuration

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



External Controls

- ☐ **USB Joystick Jogging**
  - ▷ Details
- ☐ **External Button Jogging**
  - ▷ Details
- ☑ **External MPG Jogging**
  - ▽ Details
    - ● Shared MPG / selectable axis
    - ○ Mpg per axis
    - ☑ selectable MPG increments
    - ▽ increments

default	0.0000	in	**d)**	0.0000	in
a)	0.0001	in	ad)	0.0000	in
b)	0.0005	in	bd)	0.0000	in
ab)	0.0010	in	abc)	0.0000	in
c)	0.0050	in	cd)	0.0000	in
ac)	0.0100	in	acd)	0.0000	in
bc)	0.0500	in	bcd)	0.0000	in
abc)	0.1000	in	abcd)	0.0000	in

Mux options

- ☑ use debounce 0.20 Sec
- ☑ use gray code
- ☐ ignore all inputs false

- ☐ **External Feed Override**
  - ▷ Details
- ☐ **Max Velocity Override**
  - ▷ Details
- ☐ **External Spindle Override**
  - ▷ Details

Figure 6.3: GUI External

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

Joystick jogging

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

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

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

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

joystick jogging uses HALUI and hal_input components.

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

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

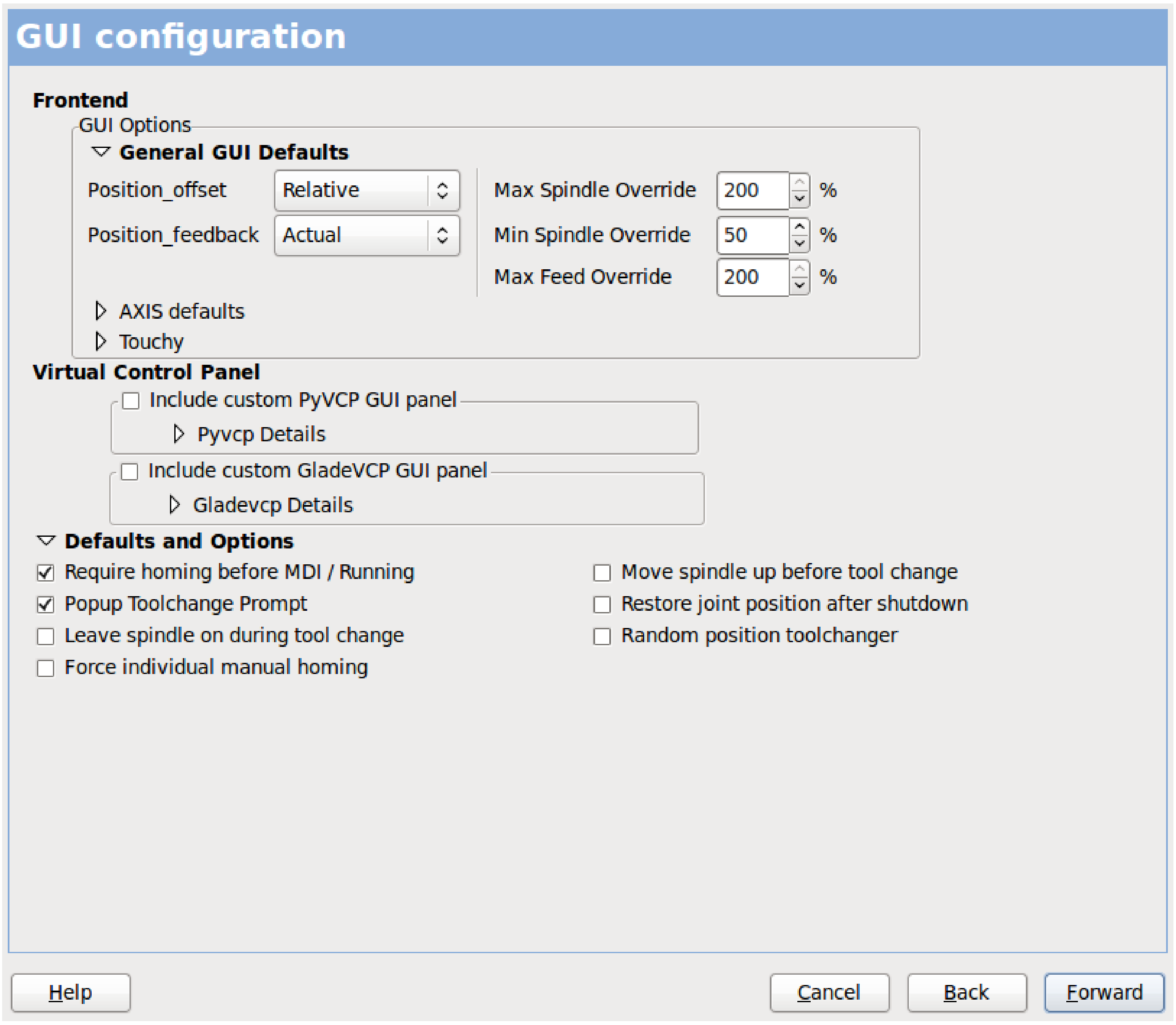
MPG's are often found on commercial grade machines. They output quadrature pulses that can be counted with a MESA encoder counter. PNCconf allows for an MPG per axis or one MPG shared with all axis. It allows for selection of jog speeds using switches or a single speed.

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

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

6.4 GUI Configuration

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


GUI configuration

Frontend

GUI Options

General GUI Defaults

Position_offset	Relative	Max Spindle Override	200	%
Position_feedback	Actual	Min Spindle Override	50	%
		Max Feed Override	200	%

AXIS defaults

Touchy

Virtual Control Panel

☐ Include custom PyVCP GUI panel

Pyvcp Details

☐ Include custom GladeVCP GUI panel

Gladevcp Details

Defaults and Options

- ☑ Require homing before MDI / Running
- ☑ Popup Toolchange Prompt
- ☐ Leave spindle on during tool change
- ☐ Force individual manual homing
- ☐ Move spindle up before tool change
- ☐ Restore joint position after shutdown
- ☐ Random position toolchanger

Figure 6.4: GUI Configuration

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

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

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

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

PyVCP

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

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

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

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

GLADE VCP

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

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

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

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

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

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

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

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

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

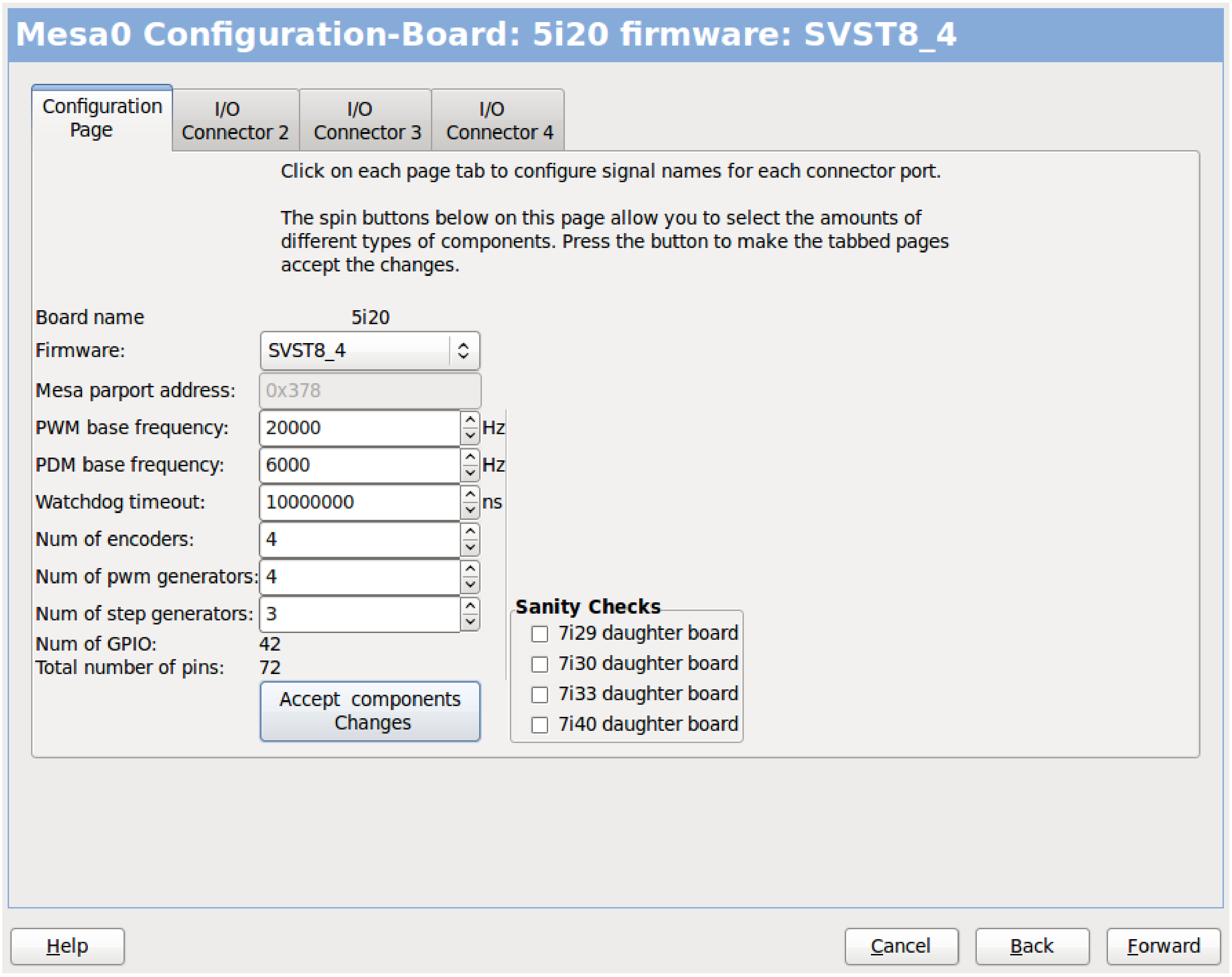
You can force it to be stand-alone or with AXIS it can be in the center or on the right side, with Touchy it can be in the center.

Defaults and Options

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

6.5 Mesa Configuration

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



Click on each page tab to configure signal names for each connector port.

The spin buttons below on this page allow you to select the amounts of different types of components. Press the button to make the tabbed pages accept the changes.

Board name	5i20
Firmware:	SVST8_4
Mesa parport address:	0x378
PWM base frequency:	20000 Hz
PDM base frequency:	6000 Hz
Watchdog timeout:	10000000 ns
Num of encoders:	4
Num of pwm generators:	4
Num of step generators:	3
Num of GPIO:	42
Total number of pins:	72

Sanity Checks
- ☐ 7i29 daughter board
- ☐ 7i30 daughter board
- ☐ 7i33 daughter board
- ☐ 7i40 daughter board

Figure 6.5: Mesa Configuration

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

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

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

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

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

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

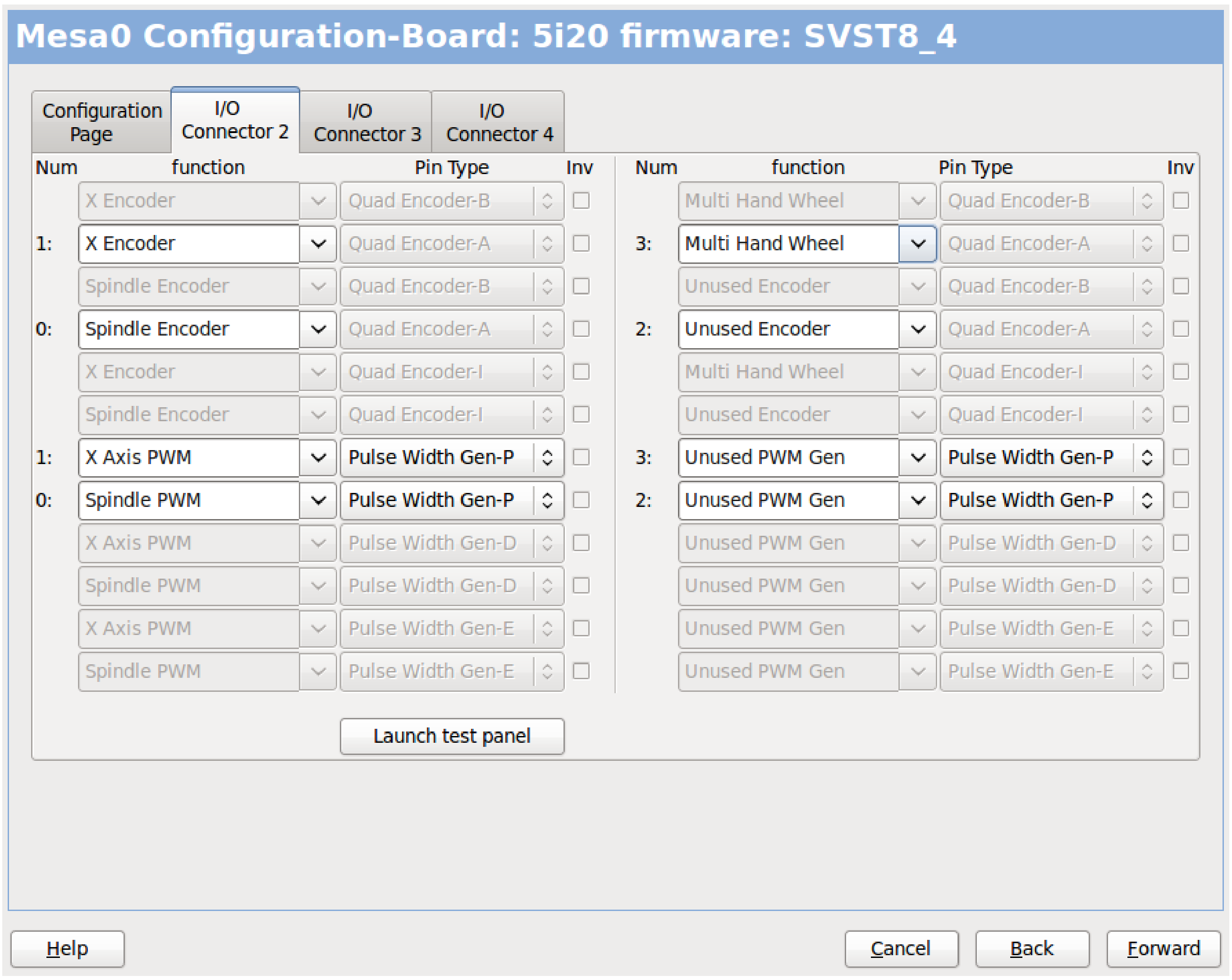
You may choose the number of available components by deselecting unused ones. Not all component types are available with all firmware.

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

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

6.6 Mesa I/O Setup

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

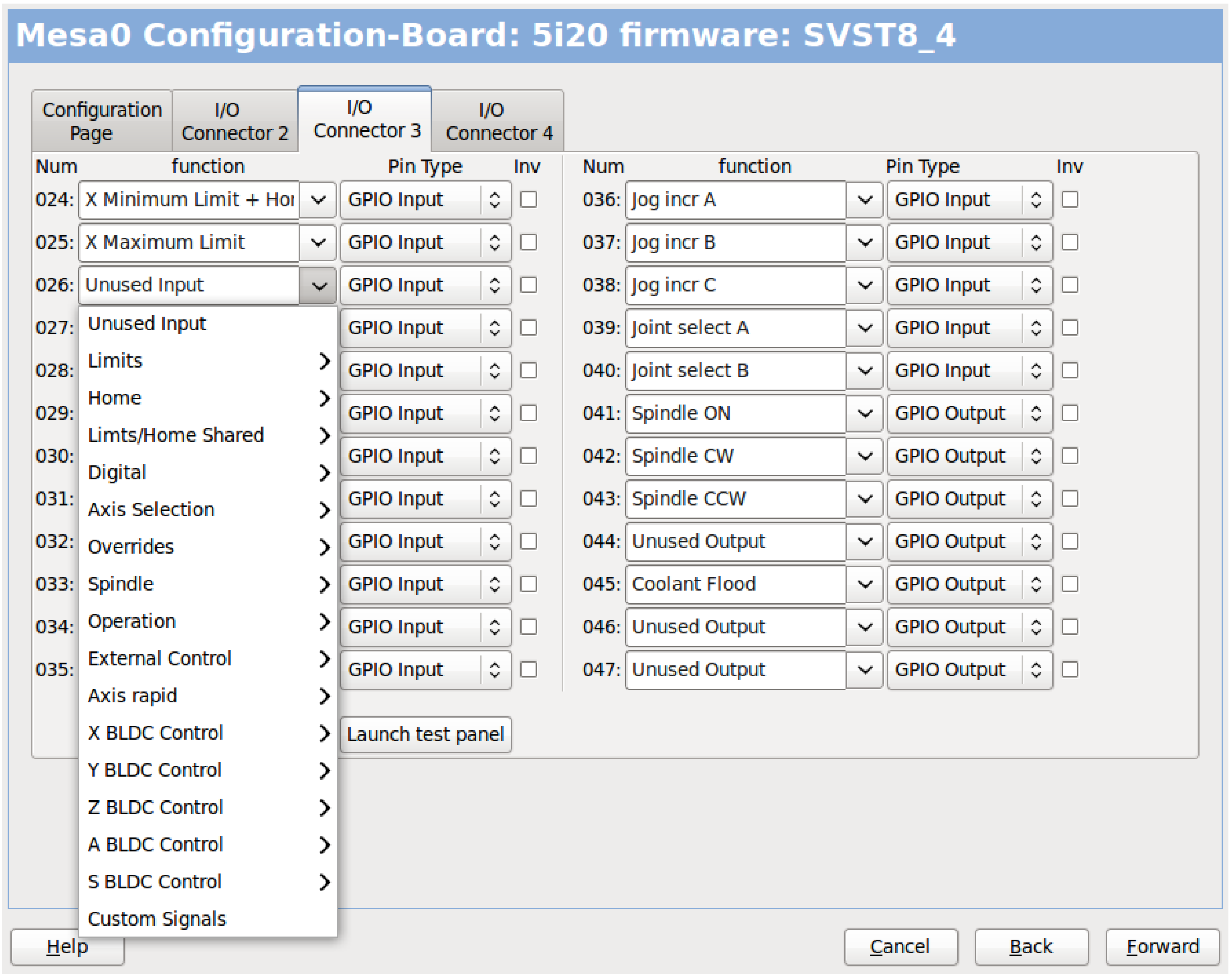


Mesa0 Configuration-Board: 5i20 firmware: SVST8_4

Num	function	Pin Type	Inv	Num	function	Pin Type	Inv
	X Encoder	Quad Encoder-B	☐		Multi Hand Wheel	Quad Encoder-B	☐
1:	X Encoder	Quad Encoder-A	☐	3:	Multi Hand Wheel	Quad Encoder-A	☐
	Spindle Encoder	Quad Encoder-B	☐		Unused Encoder	Quad Encoder-B	☐
0:	Spindle Encoder	Quad Encoder-A	☐	2:	Unused Encoder	Quad Encoder-A	☐
	X Encoder	Quad Encoder-I	☐		Multi Hand Wheel	Quad Encoder-I	☐
	Spindle Encoder	Quad Encoder-I	☐		Unused Encoder	Quad Encoder-I	☐
1:	X Axis PWM	Pulse Width Gen-P	☐	3:	Unused PWM Gen	Pulse Width Gen-P	☐
0:	Spindle PWM	Pulse Width Gen-P	☐	2:	Unused PWM Gen	Pulse Width Gen-P	☐
	X Axis PWM	Pulse Width Gen-D	☐		Unused PWM Gen	Pulse Width Gen-D	☐
	Spindle PWM	Pulse Width Gen-D	☐		Unused PWM Gen	Pulse Width Gen-D	☐
	X Axis PWM	Pulse Width Gen-E	☐		Unused PWM Gen	Pulse Width Gen-E	☐
	Spindle PWM	Pulse Width Gen-E	☐		Unused PWM Gen	Pulse Width Gen-E	☐

Figure 6.6: Mesa I/O C2

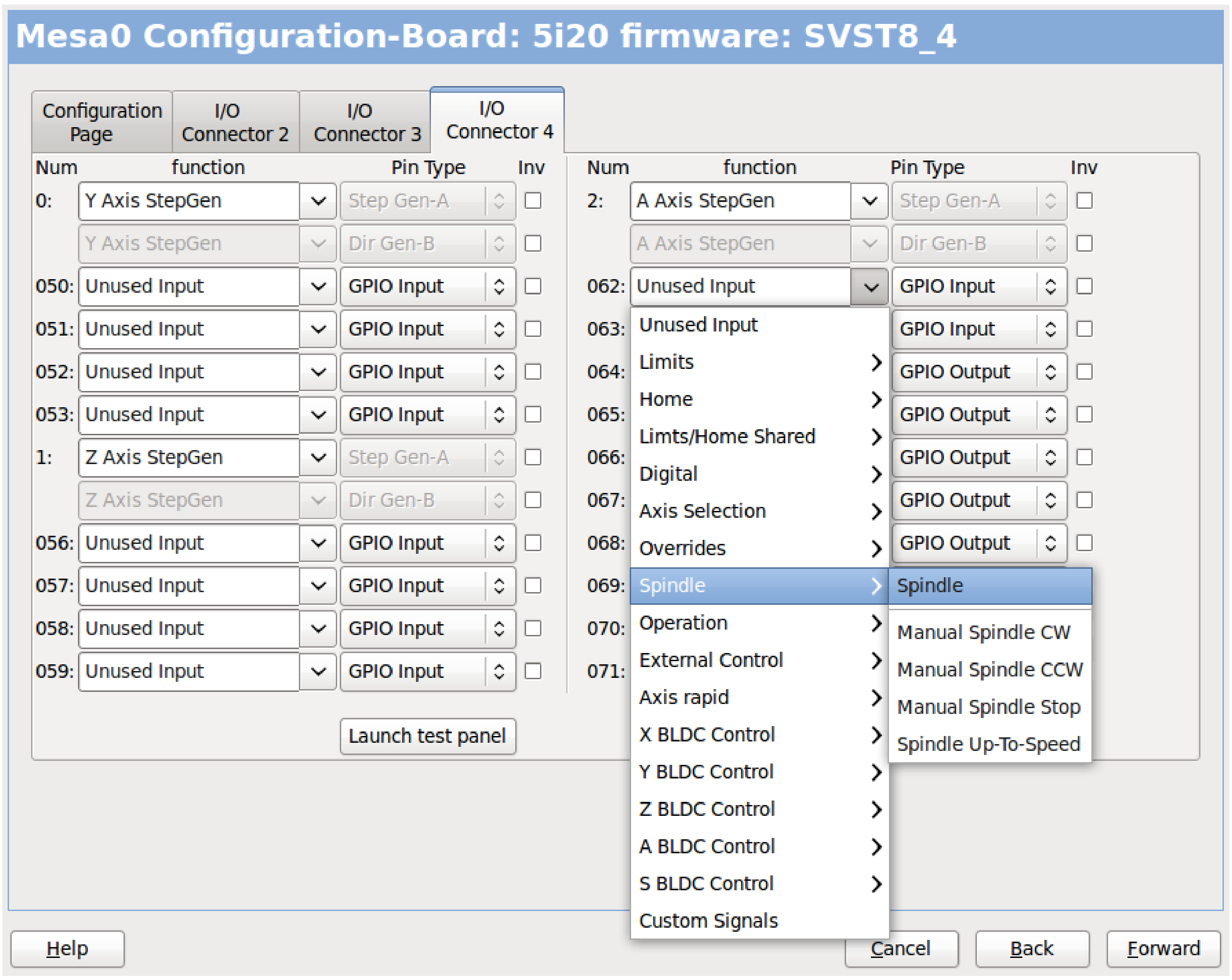
On this tab with this firmware the components are setup for a 7i33 daughter board, usually used with closed loop servos. Note the component numbers of the encoder counters and PWM drivers are not in numerical order. This follows the daughter board requirements.



Num	function	Pin Type	Inv	Num	function	Pin Type	Inv
024:	X Minimum Limit + Hoı	GPIO Input	☐	036:	Jog incr A	GPIO Input	☐
025:	X Maximum Limit	GPIO Input	☐	037:	Jog incr B	GPIO Input	☐
026:	Unused Input	GPIO Input	☐	038:	Jog incr C	GPIO Input	☐
027:		GPIO Input	☐	039:	Joint select A	GPIO Input	☐
028:		GPIO Input	☐	040:	Joint select B	GPIO Input	☐
029:		GPIO Input	☐	041:	Spindle ON	GPIO Output	☐
030:		GPIO Input	☐	042:	Spindle CW	GPIO Output	☐
031:		GPIO Input	☐	043:	Spindle CCW	GPIO Output	☐
032:		GPIO Input	☐	044:	Unused Output	GPIO Output	☐
033:		GPIO Input	☐	045:	Coolant Flood	GPIO Output	☐
034:		GPIO Input	☐	046:	Unused Output	GPIO Output	☐
035:		GPIO Input	☐	047:	Unused Output	GPIO Output	☐

Figure 6.7: Mesa I/O C3

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



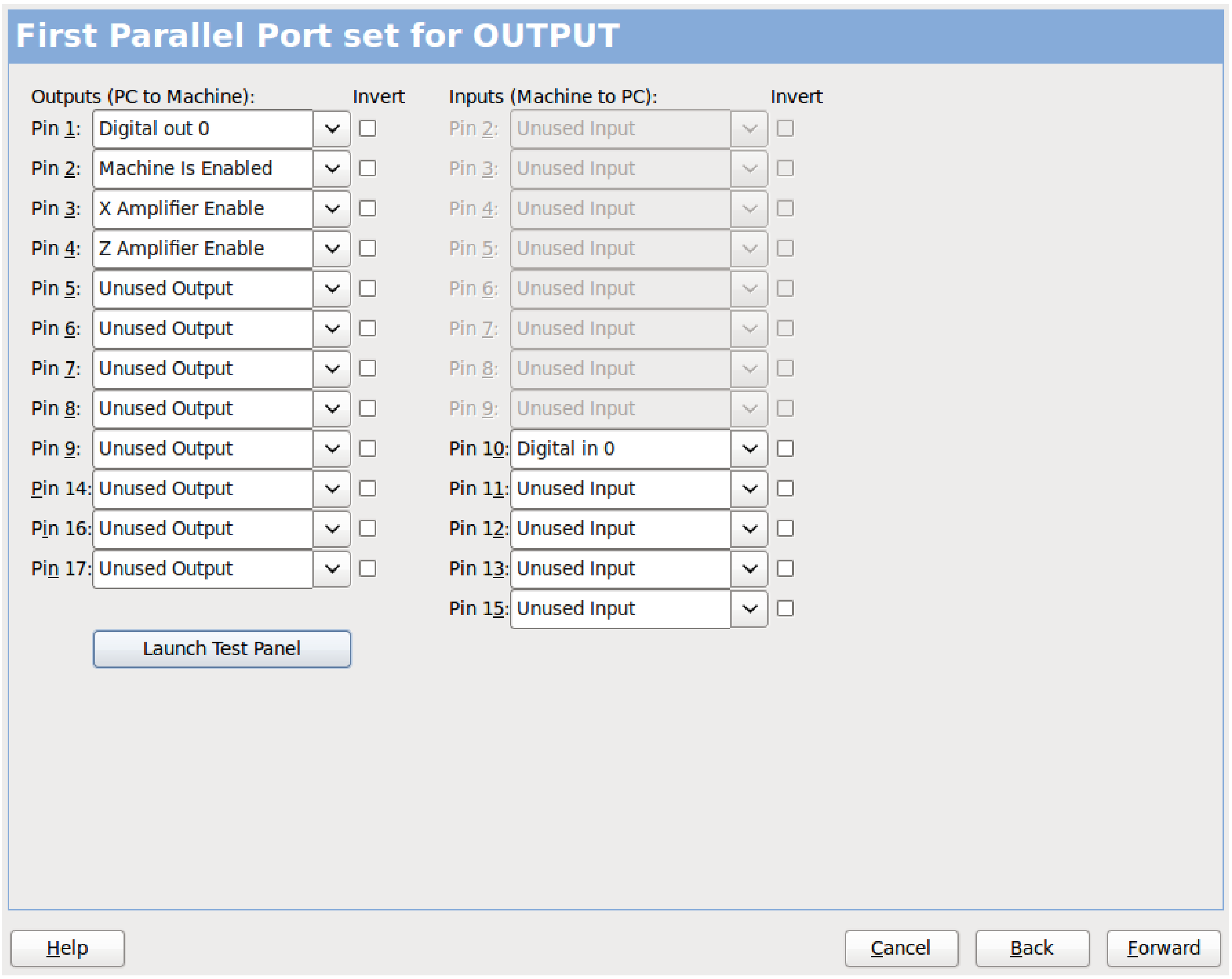
Num	function	Pin Type	Inv
0:	Y Axis StepGen	Step Gen-A	☐
	Y Axis StepGen	Dir Gen-B	☐
050:	Unused Input	GPIO Input	☐
051:	Unused Input	GPIO Input	☐
052:	Unused Input	GPIO Input	☐
053:	Unused Input	GPIO Input	☐
1:	Z Axis StepGen	Step Gen-A	☐
	Z Axis StepGen	Dir Gen-B	☐
056:	Unused Input	GPIO Input	☐
057:	Unused Input	GPIO Input	☐
058:	Unused Input	GPIO Input	☐
059:	Unused Input	GPIO Input	☐

Num	function	Pin Type	Inv
2:	A Axis StepGen	Step Gen-A	☐
	A Axis StepGen	Dir Gen-B	☐
062:	Unused Input	GPIO Input	☐
063:		GPIO Input	☐
064:		GPIO Output	☐
065:		GPIO Output	☐
066:		GPIO Output	☐
067:		GPIO Output	☐
068:		GPIO Output	☐
069:			
070:			
071:			

Figure 6.8: Mesa I/O C4

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

6.7 Parport configuration

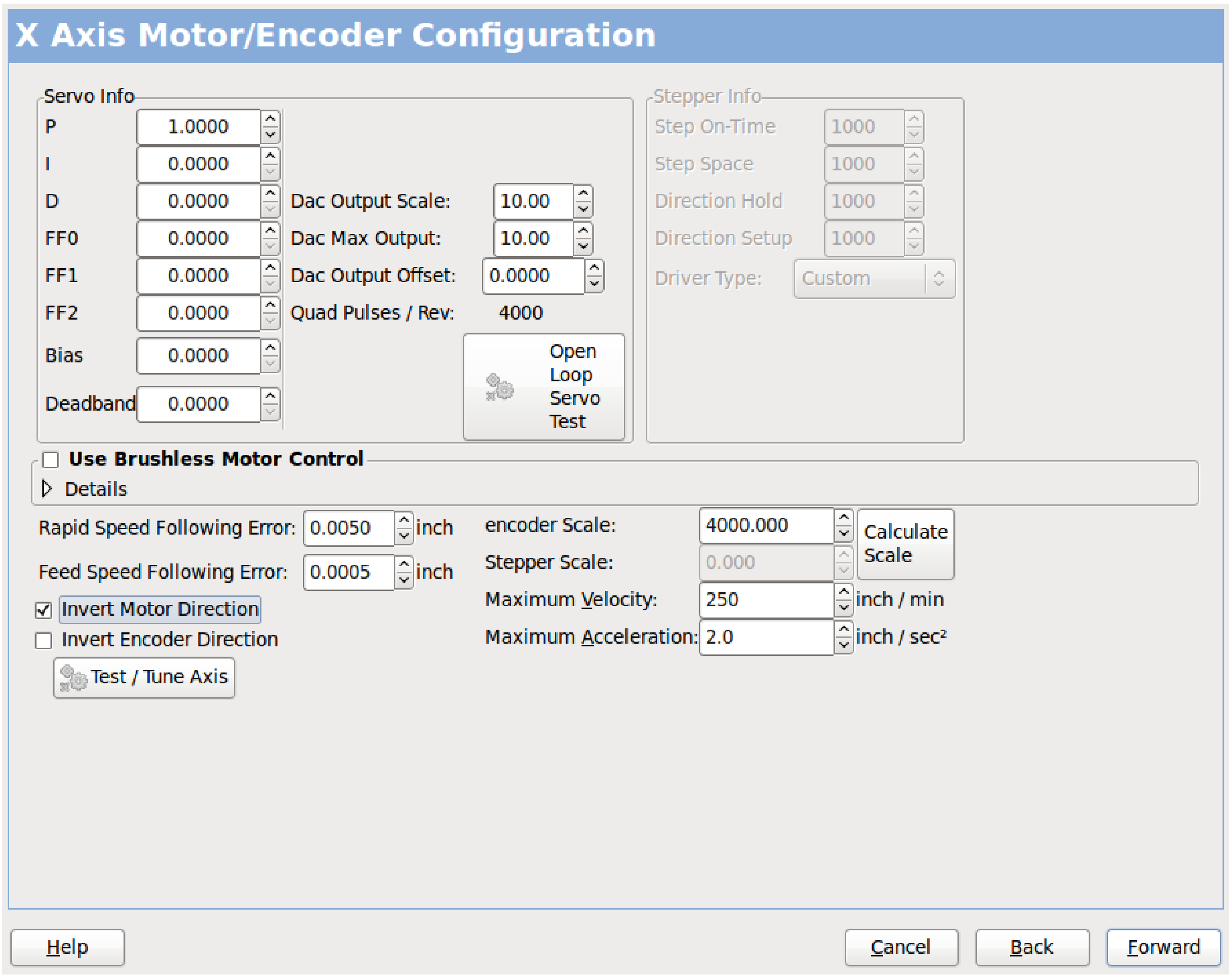


First Parallel Port set for OUTPUT

Outputs (PC to Machine):		Invert	Inputs (Machine to PC):		Invert
Pin 1:	Digital out 0	☐	Pin 2:	Unused Input	☐
Pin 2:	Machine Is Enabled	☐	Pin 3:	Unused Input	☐
Pin 3:	X Amplifier Enable	☐	Pin 4:	Unused Input	☐
Pin 4:	Z Amplifier Enable	☐	Pin 5:	Unused Input	☐
Pin 5:	Unused Output	☐	Pin 6:	Unused Input	☐
Pin 6:	Unused Output	☐	Pin 7:	Unused Input	☐
Pin 7:	Unused Output	☐	Pin 8:	Unused Input	☐
Pin 8:	Unused Output	☐	Pin 9:	Unused Input	☐
Pin 9:	Unused Output	☐	Pin 10:	Digital in 0	☐
Pin 14:	Unused Output	☐	Pin 11:	Unused Input	☐
Pin 16:	Unused Output	☐	Pin 12:	Unused Input	☐
Pin 17:	Unused Output	☐	Pin 13:	Unused Input	☐
			Pin 15:	Unused Input	☐

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

6.8 Axis Configuration


X Axis Motor/Encoder Configuration

Servo Info

P	1.0000
I	0.0000
D	0.0000
FF0	0.0000
FF1	0.0000
FF2	0.0000
Bias	0.0000
Deadband	0.0000

Dac Output Scale:	10.00
Dac Max Output:	10.00
Dac Output Offset:	0.0000
Quad Pulses / Rev:	4000

Stepper Info

Step On-Time	1000
Step Space	1000
Direction Hold	1000
Direction Setup	1000
Driver Type:	Custom

☐ Use Brushless Motor Control

Rapid Speed Following Error:	0.0050	inch
Feed Speed Following Error:	0.0005	inch
encoder Scale:	4000.000	
Stepper Scale:	0.000	
Maximum Velocity:	250	inch / min
Maximum Acceleration:	2.0	inch / sec²

☑ Invert Motor Direction
☐ Invert Encoder Direction

Figure 6.9: Axis Drive Configuration

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

Open Loop Test

An open loop test is important as it confirms the direction of the motor and encoder. The motor should move the axis in the positive direction when the positive button is pushed and also the encoder should count in the postie direction. The axis movement should follow the Machinery's Handbook [1] standards or AXIS graphical display will not make much sense. Hopefully the help page and diagrams can help figure this out. Note that axis directions are based on TOOL movement not table movement. There is no acceleration ramping with the open loop test so start with lower DAC numbers. By moving the axis a known distance one can confirm the encoder scaling. The encoder should count even without the amp enabled depending on how power is supplied to the encoder.

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

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

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

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

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

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

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

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

Table 6.1: Output Voltage Measurements

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

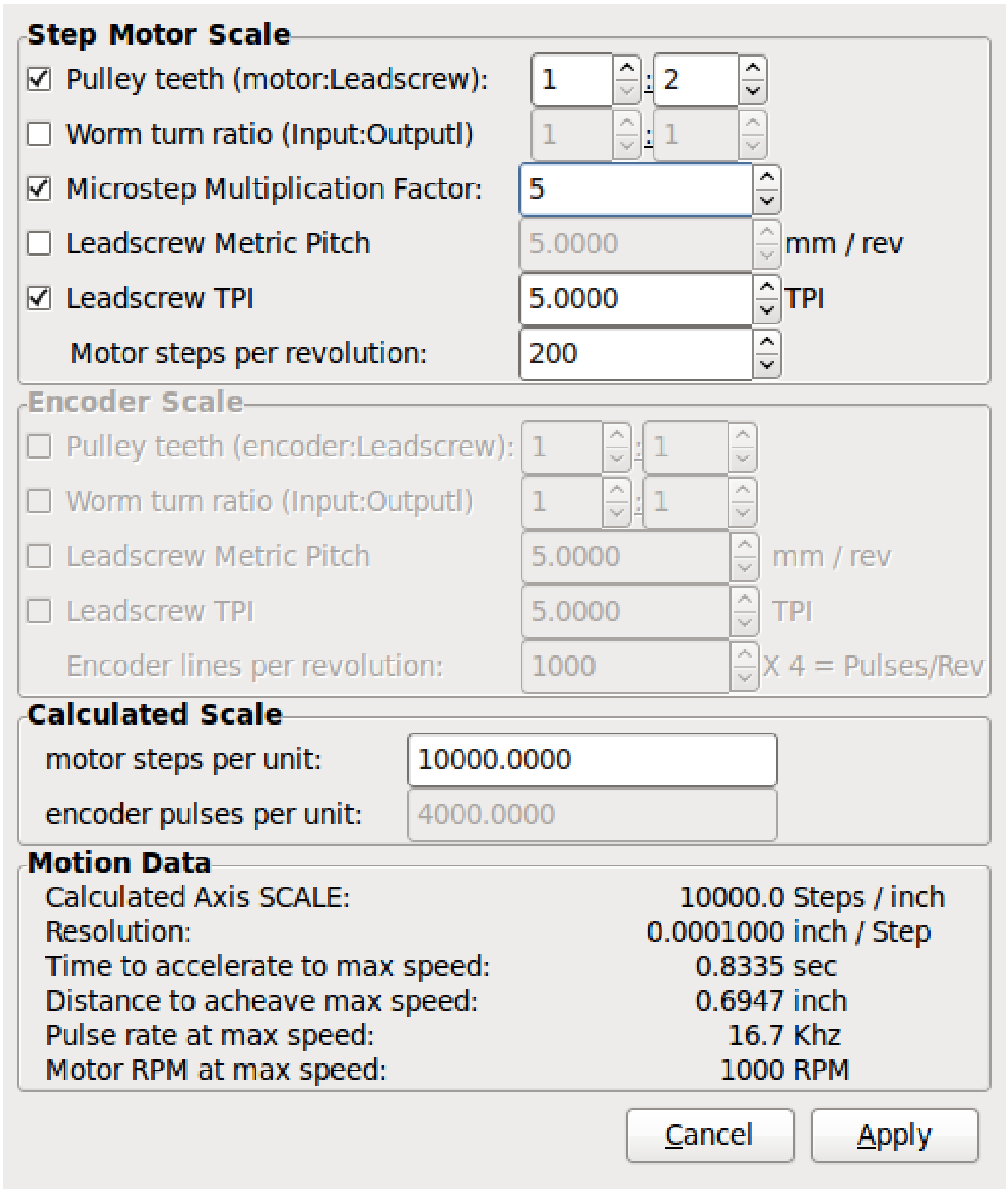
- Do a least-squares linear fit to get coefficients a, b such that meas=a*raw+b
- Note that we want raw output such that our measured result is identical to the commanded output. This means
  - cmd=a*raw+b
  - raw=(cmd-b)/a
- As a result, the a and b coefficients from the linear fit can be used as the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

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

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

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

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



Step Motor Scale

✓	Pulley teeth (motor:Leadscrew):	1 : 2	
☐	Worm turn ratio (Input:Outputl)	1 : 1	
✓	Microstep Multiplication Factor:	5	
☐	Leadscrew Metric Pitch	5.0000	mm / rev
✓	Leadscrew TPI	5.0000	TPI
	Motor steps per revolution:	200	

Encoder Scale

☐	Pulley teeth (encoder:Leadscrew):	1 : 1	
☐	Worm turn ratio (Input:Outputl)	1 : 1	
☐	Leadscrew Metric Pitch	5.0000	mm / rev
☐	Leadscrew TPI	5.0000	TPI
	Encoder lines per revolution:	1000	X 4 = Pulses/Rev

Calculated Scale

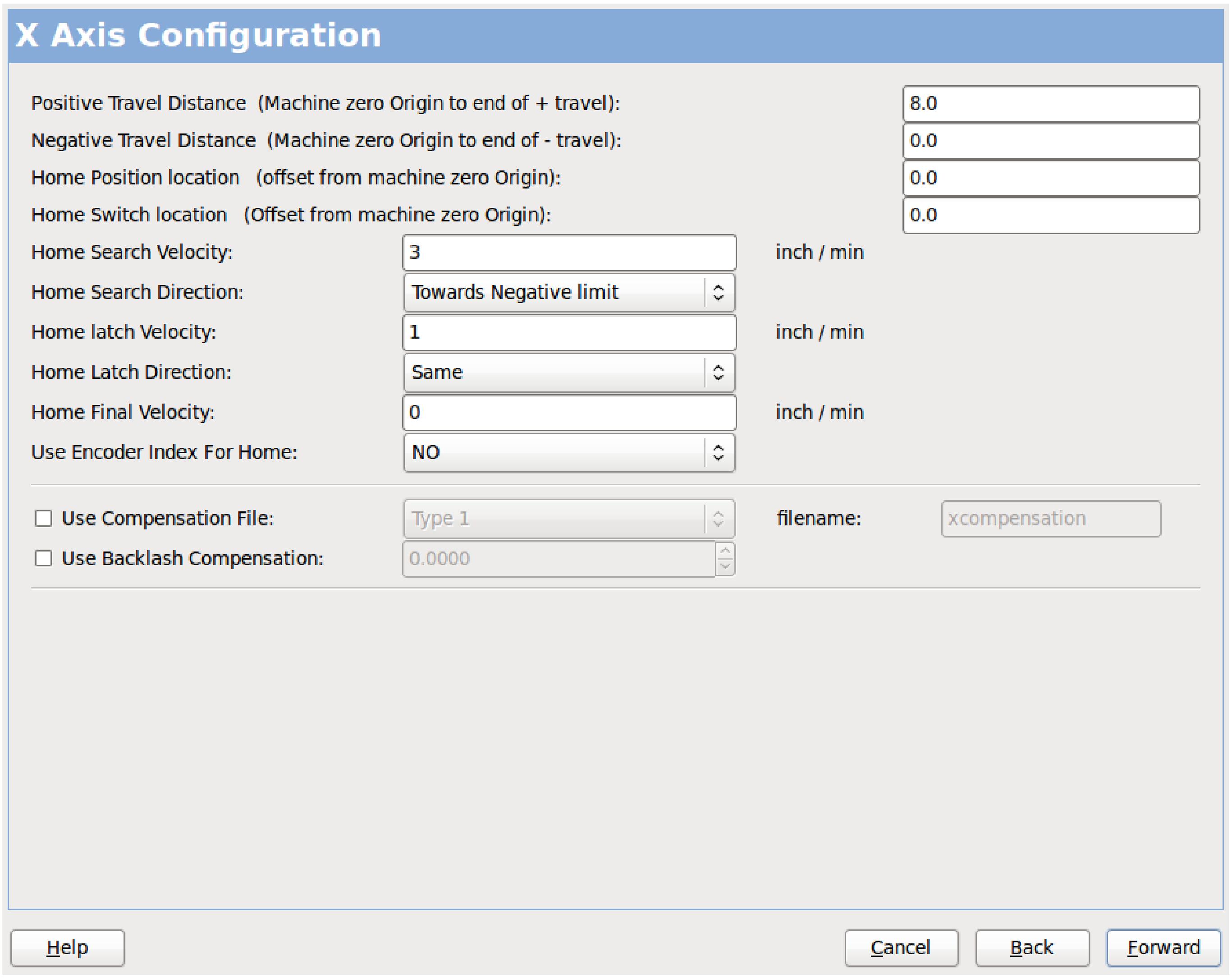
motor steps per unit:	10000.0000
encoder pulses per unit:	4000.0000

Motion Data

Calculated Axis SCALE:	10000.0 Steps / inch
Resolution:	0.0001000 inch / Step
Time to accelerate to max speed:	0.8335 sec
Distance to acheave max speed:	0.6947 inch
Pulse rate at max speed:	16.7 Khz
Motor RPM at max speed:	1000 RPM

Figure 6.10: Axis Scale Calculation

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


X Axis Configuration

Positive Travel Distance (Machine zero Origin to end of + travel):			8.0
Negative Travel Distance (Machine zero Origin to end of - travel):			0.0
Home Position location (offset from machine zero Origin):			0.0
Home Switch location (Offset from machine zero Origin):			0.0
Home Search Velocity:	3	inch / min	
Home Search Direction:	Towards Negative limit		
Home latch Velocity:	1	inch / min	
Home Latch Direction:	Same		
Home Final Velocity:	0	inch / min	
Use Encoder Index For Home:	NO		
☐ Use Compensation File:	Type 1	filename:	xcompensation
☐ Use Backlash Compensation:	0.0000		

Figure 6.11: Axis Configuration

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

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

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

ON A TYPICAL KNEE OR BED MILL

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

ON A TYPICAL LATHE

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

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

Decide on limit switch location.

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

Decide on the home switch location.

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

Decide on the MACHINE ORIGIN position.

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

Decide on the (final) HOME POSITION.

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

Measure / calculate the positive / negative axis travel distances.

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

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

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

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

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

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

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

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

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

Home Latch Velocity
Fine Home search velocity in units per minute

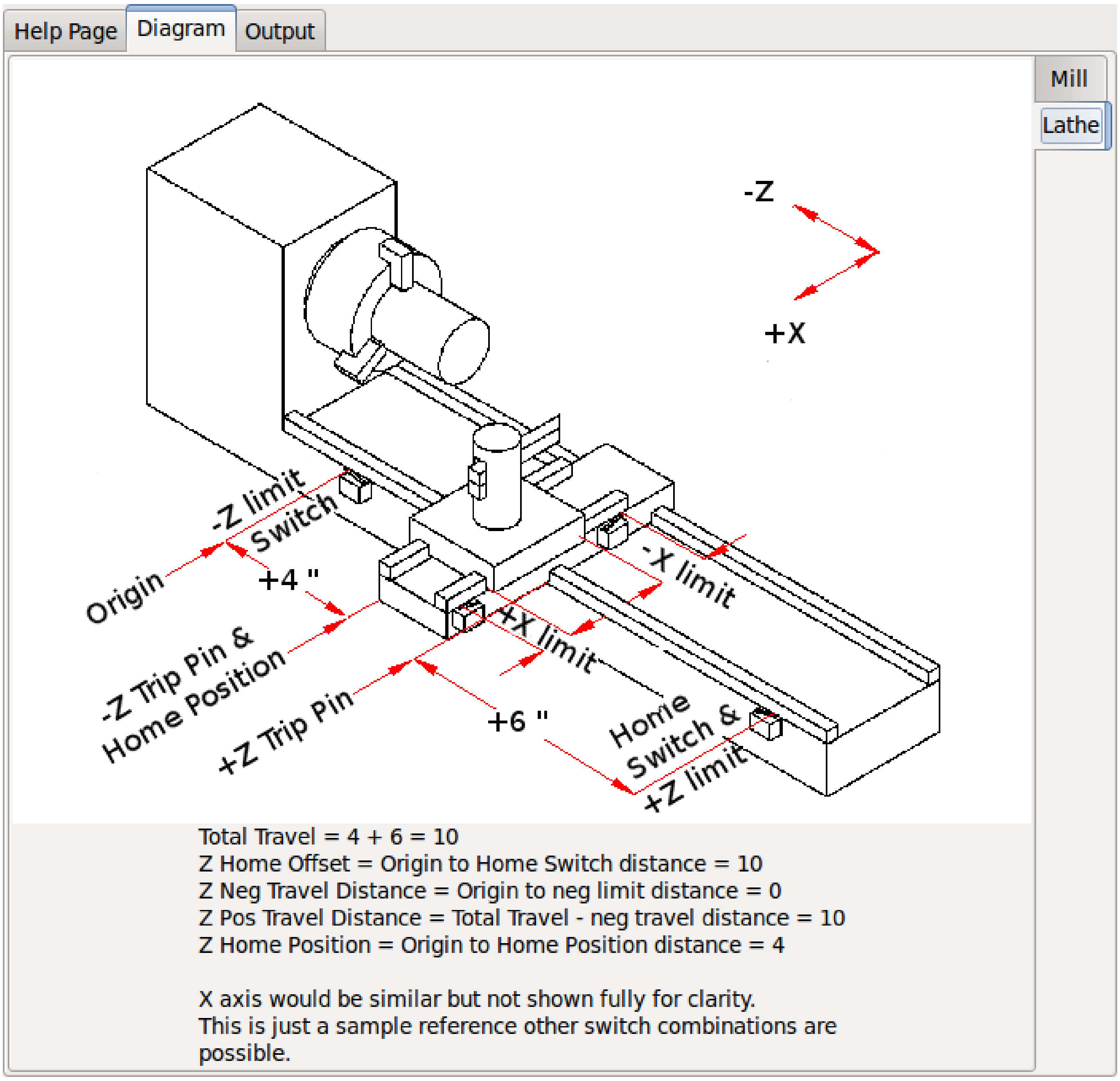
Home Final Velocity
Velocity used from latch position to (final) home position in units per minute. Set to 0 for max rapid speed

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

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

Use Compensation File
Allows specifying a Comp filename and type. Allows sophisticated compensation. See Manual.

Use Backlash Compensation
Allows setting of simple backlash compensation. Can not be used with Compensation File. See Manual.



Total Travel = 4 + 6 = 10
Z Home Offset = Origin to Home Switch distance = 10
Z Neg Travel Distance = Origin to neg limit distance = 0
Z Pos Travel Distance = Total Travel - neg travel distance = 10
Z Home Position = Origin to Home Position distance = 4

X axis would be similar but not shown fully for clarity.
This is just a sample reference other switch combinations are possible.

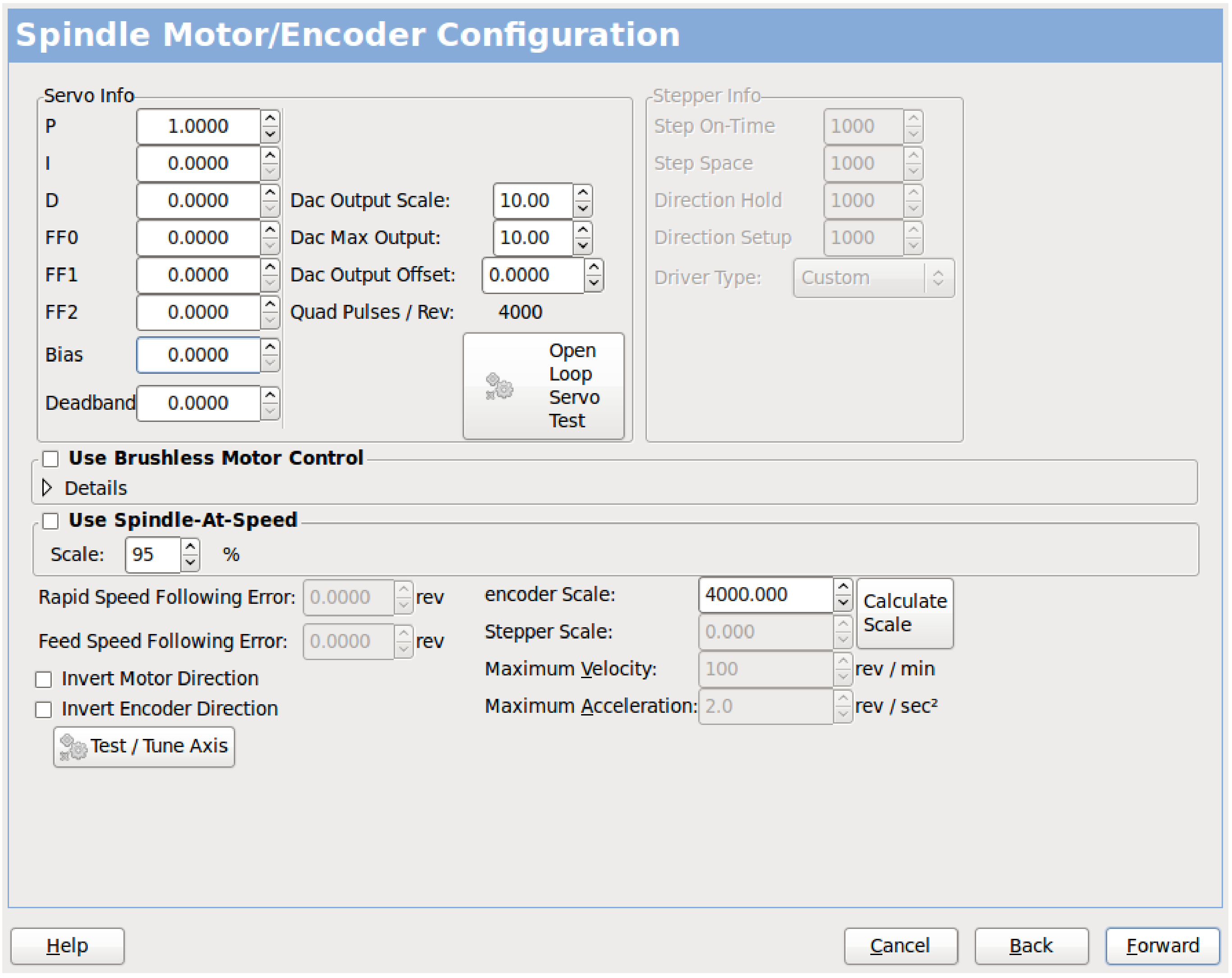
Figure 6.12: AXIS Help Diagram

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

6.9 Spindle Configuration

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

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



Spindle Motor/Encoder Configuration

Servo Info

P	1.0000		
I	0.0000		
D	0.0000	Dac Output Scale:	10.00
FF0	0.0000	Dac Max Output:	10.00
FF1	0.0000	Dac Output Offset:	0.0000
FF2	0.0000	Quad Pulses / Rev:	4000
Bias	0.0000		
Deadband	0.0000		

Stepper Info

Step On-Time	1000
Step Space	1000
Direction Hold	1000
Direction Setup	1000
Driver Type:	Custom

☐ Use Brushless Motor Control

☐ Use Spindle-At-Speed
Scale: 95 %

Rapid Speed Following Error:	0.0000 rev	encoder Scale:	4000.000
Feed Speed Following Error:	0.0000 rev	Stepper Scale:	0.000
		Maximum Velocity:	100 rev / min
		Maximum Acceleration:	2.0 rev / sec²

☐ Invert Motor Direction
☐ Invert Encoder Direction

Figure 6.13: Spindle Configuration

This page is similar to the axis motor configuration page.

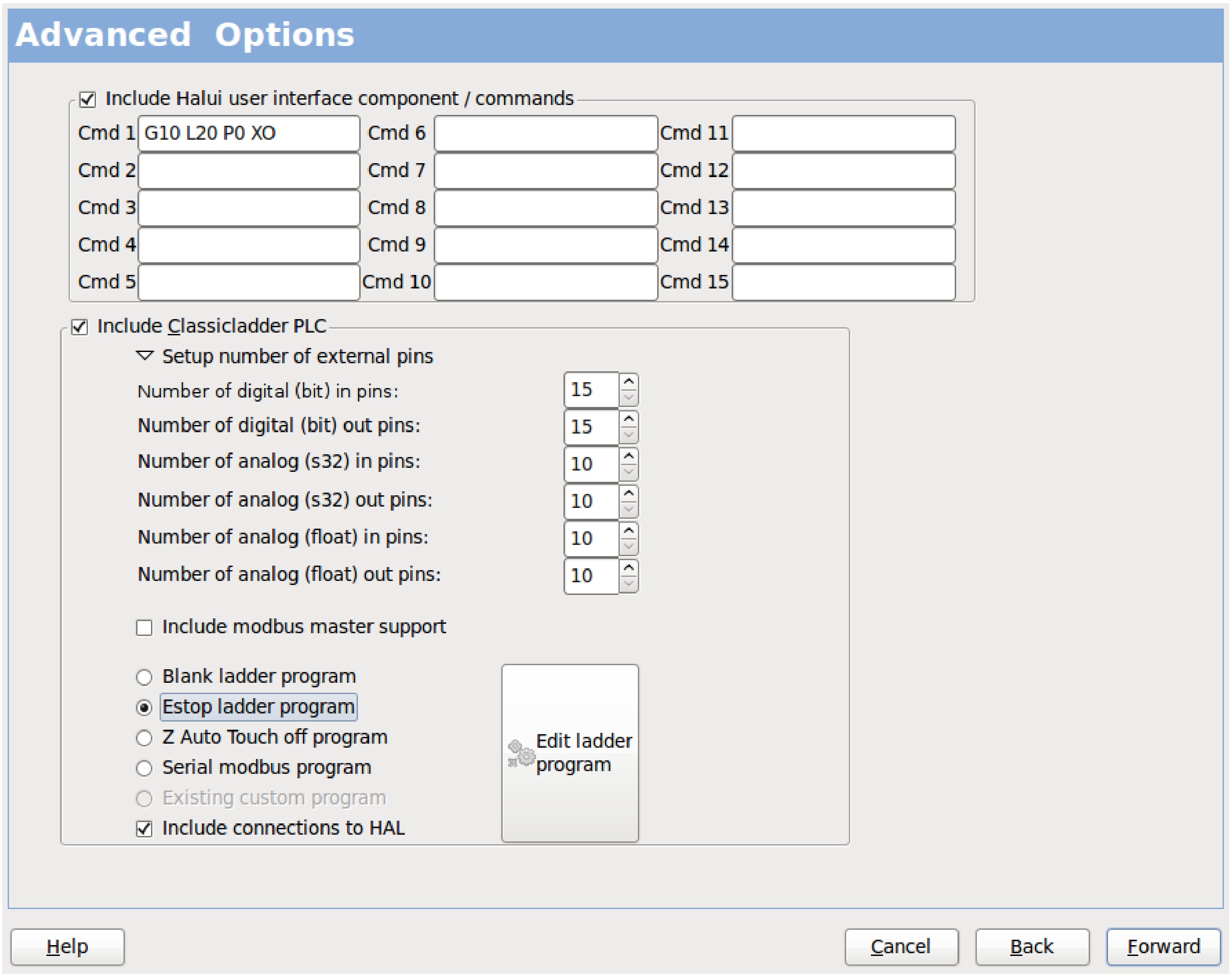
There are some differences:

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

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

6.10 Advanced Options

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


Advanced Options

☑ Include Halui user interface component / commands

Cmd 1	G10 L20 P0 XO	Cmd 6		Cmd 11	
Cmd 2		Cmd 7		Cmd 12	
Cmd 3		Cmd 8		Cmd 13	
Cmd 4		Cmd 9		Cmd 14	
Cmd 5		Cmd 10		Cmd 15	

☑ Include Classicladder PLC

Setup number of external pins

Number of digital (bit) in pins:	15
Number of digital (bit) out pins:	15
Number of analog (s32) in pins:	10
Number of analog (s32) out pins:	10
Number of analog (float) in pins:	10
Number of analog (float) out pins:	10

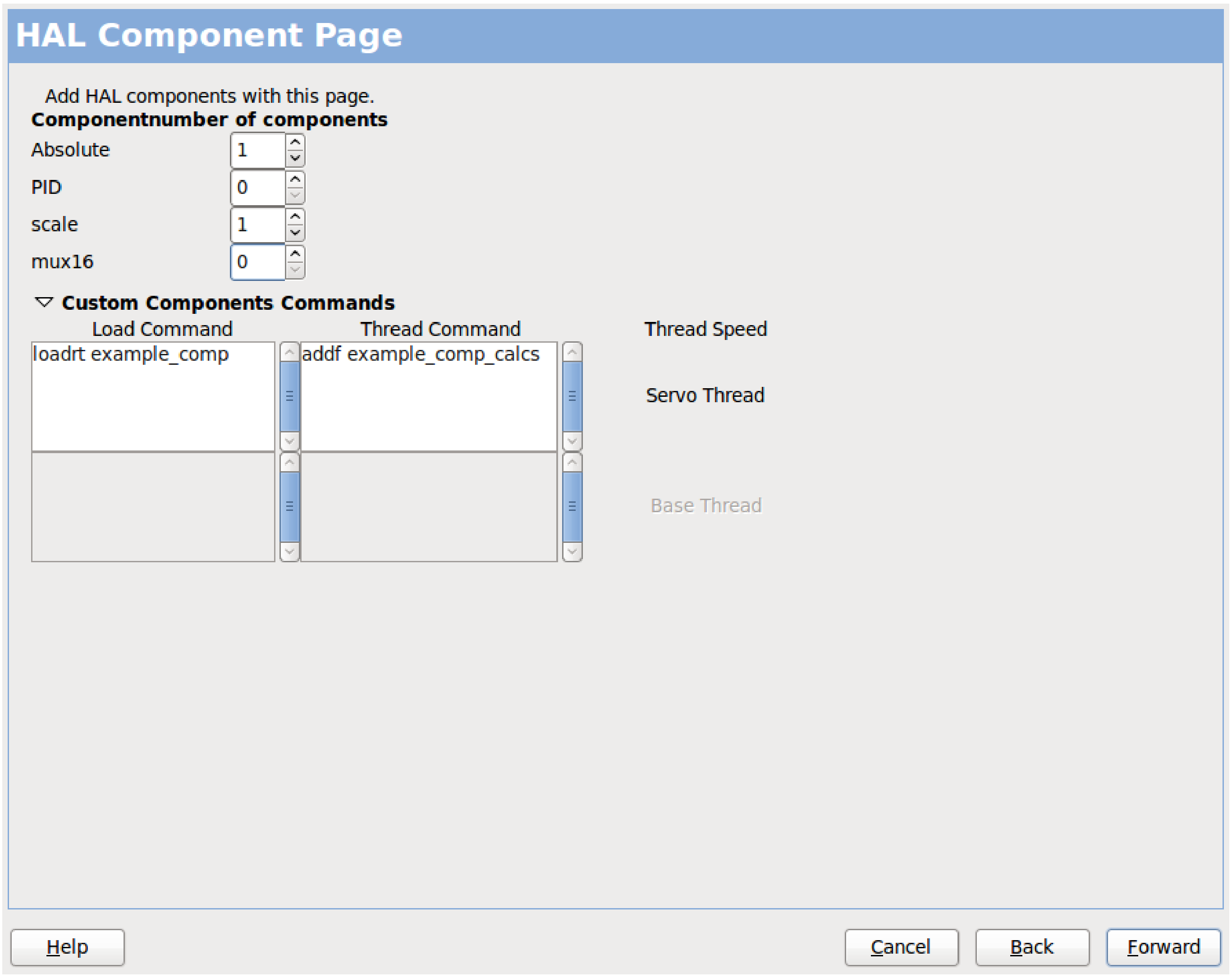
☐ Include modbus master support

○ Blank ladder program
● Estop ladder program
○ Z Auto Touch off program
○ Serial modbus program
○ Existing custom program
☑ Include connections to HAL

Figure 6.14: Advanced Options

6.11 HAL Components

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


HAL Component Page

Add HAL components with this page.

Componentnumber of components

Absolute	1
PID	0
scale	1
mux16	0

Custom Components Commands

Load Command	Thread Command	Thread Speed
loadrt example_comp	addf example_comp_calcs	Servo Thread
		Base Thread

Figure 6.15: HAL Components

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

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

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

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

6.12 Advanced Usage Of PNCconf

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

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

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

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

Encoders will add < customname > +:

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

Steppers add:

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

PWM add:

- enable
- value

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

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

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

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

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

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

Custom HAL Files

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

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

Chapter 7

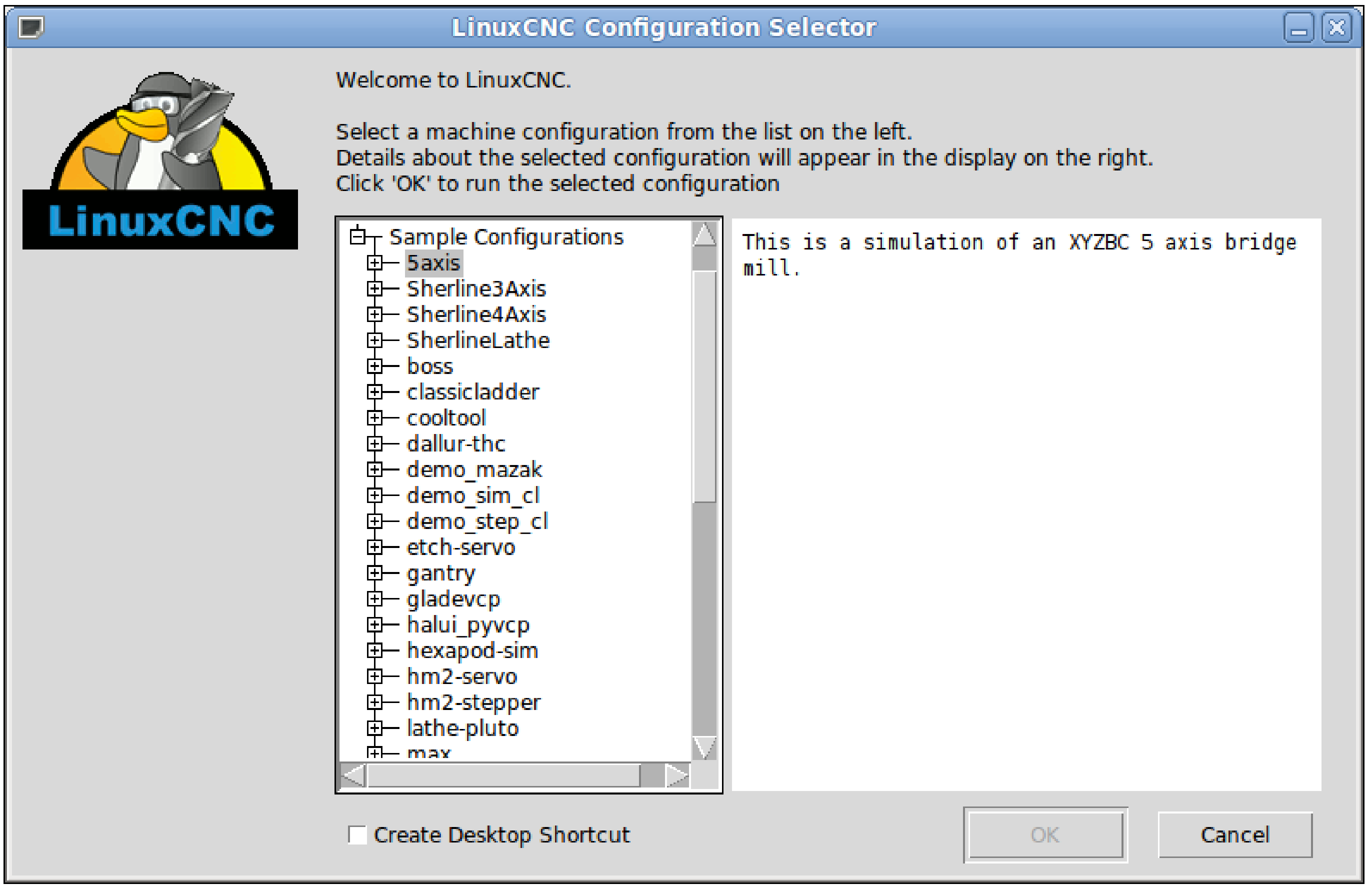
Running LinuxCNC

7.1 Invoking LinuxCNC

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

7.2 Configuration Selector

By default, the Configuration Selector dialog is shown when you first run LinuxCNC. Your own personalized configurations are shown at the top of the list, followed by sample configurations. Because each sample configuration is for a different type of hardware interface, almost all will not run without the hardware installed. The configurations listed under the category *sim* run entirely without attached hardware.



Welcome to LinuxCNC.

Select a machine configuration from the list on the left.
Details about the selected configuration will appear in the display on the right.
Click 'OK' to run the selected configuration

```
This is a simulation of an XYZBC 5 axis bridge
mill.
```

Figure 7.1: LinuxCNC Configuration Selector

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

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

7.3 Next steps in configuration

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

Chapter 8

Linux FAQ

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

8.1 Automatic Login

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

8.2 Automatic Startup

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

Example:

```
emc /home/mill/emc2/config/mill/mill.ini
```

8.3 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.

8.4 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.

8.5 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.

8.5.1 The Command Line Way

Open up *Applications* > *Accessories* > *Terminal*.

In the terminal window type

```
sudo gedit
```

Open the file with File > Open > Edit

8.5.2 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

8.5.3 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.

8.6 Terminal Commands

8.6.1 Working Directory

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

```
pwd
```

8.6.2 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
```

8.6.3 Listing files in a directory

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

```
dir
```

or

```
ls
```

8.6.4 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.

8.6.5 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.

8.6.6 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
```

8.7 Convenience Items

8.7.1 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.

8.8 Hardware Problems

8.8.1 Hardware Info

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

```
lspci -v
```

8.8.2 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

8.9 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 9

Legal Section

9.1 Copyright Terms

9.2 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

Chapter 10

Index