

## NAME

linuxcnc – LinuxCNC (The Enhanced Machine Controller)

## SYNOPSIS

**linuxcnc** [-v] [-d] [*INIFILE*]

## DESCRIPTION

**linuxcnc** is used to start LinuxCNC (The Enhanced Machine Controller). It starts the realtime system and then initializes a number of LinuxCNC components (IO, Motion, GUI, HAL, etc). The most important parameter is *INIFILE*, which specifies the configuration name you would like to run. If *INIFILE* is not specified, the **linuxcnc** script presents a graphical wizard to let you choose one.

## OPTIONS

- v** Be a little bit verbose. This causes the script to print information as it works.
- d** Print lots of debug information. All executed commands are echoed to the screen. This mode is useful when something is not working as it should.

## INIFILE

The ini file is the main piece of an LinuxCNC configuration. It is not the entire configuration; there are various other files that go with it (NML files, HAL files, TBL files, VAR files). It is, however, the most important one, because it is the file that holds the configuration together. It can adjust a lot of parameters itself, but it also tells **linuxcnc** which other files to load and use.

There are several ways to specify which config to use:

Specify the absolute path to an ini, e.g.

**linuxcnc** /usr/local/linuxcnc/configs/sim/sim.ini

Specify a relative path from the current directory, e.g.

**linuxcnc** configs/sim/sim.ini

Otherwise, in the case where the **INIFILE** is not specified, the behavior will depend on whether you configured linuxcnc with **--enable-run-in-place**. If so, the linuxcnc config chooser will search only the configs directory in your source tree. If not (or if you are using a packaged version of linuxcnc), it may search several directories. The config chooser is currently set to search the path:

**~/linuxcnc/configs:/usr/local/etc/linuxcnc/configs:/usr/share/doc/linuxcnc/examples/sample-configs**

## EXAMPLES

**linuxcnc**

**linuxcnc** configs/sim/sim.ini

**linuxcnc** /etc/linuxcnc/sample-configs/stepper/stepper\_mm.ini

## SEE ALSO

**halcmd(1)**

Much more information about LinuxCNC and HAL is available in the LinuxCNC and HAL User Manuals, found at /usr/share/doc/linuxcnc/.

## HISTORY

**BUGS**

None known at this time.

**AUTHOR**

This man page written by Alex Joni, as part of the LinuxCNC Enhanced Machine Controller project.

**REPORTING BUGS**

Report bugs to alex\_joni AT users DOT sourceforge DOT net

**COPYRIGHT**

Copyright © 2006 Alex Joni.

This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

**NAME**

axis-remote – AXIS Remote Interface

**SYNOPSIS**

**axis-remote** *OPTIONS* *FILENAME*

**DESCRIPTION**

**axis-remote** is a small script that triggers commands in a running AXIS GUI. Use **axis-remote --help** for further information.

**OPTIONS**

**--ping, -p**

Check whether AXIS is running.

**--reload, -r**

Make AXIS reload the currently loaded file.

**--clear, -c**

Make AXIS clear the backplot.

**--quit, -q**

Make AXIS quit.

**--help, -h, -?**

Display a list of valid parameters for **axis-remote**.

**--mdi COMMAND, -m COMMAND**

Run the MDI command **COMMAND**.

**FILENAME**

Load the G-code file **FILENAME**.

**SEE ALSO**

**axis(1)**

Much more information about LinuxCNC and HAL is available in the LinuxCNC and HAL User Manuals, found at [usr/share/doc/linuxcnc/](http://usr/share/doc/linuxcnc/).

**HISTORY****BUGS**

None known at this time.

**AUTHOR**

This man page written by Alex Joni, as part of the LinuxCNC project.

**REPORTING BUGS**

Report bugs to alex\_joni AT users DOT sourceforge DOT net

**COPYRIGHT**

Copyright © 2007 Alex Joni.

This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

## NAME

`axis` – AXIS LinuxCNC Graphical User Interface

## SYNOPSIS

`axis -ini INIFILE`

## DESCRIPTION

`axis` is one of the Graphical User Interfaces (GUI) for LinuxCNC. It gets run by the runscrip usually.

## OPTIONS

### INIFILE

The ini file is the main piece of an LinuxCNC configuration. It is not the entire configuration; there are various other files that go with it (NML files, HAL files, TBL files, VAR files). It is, however, the most important one, because it is the file that holds the configuration together. It can adjust a lot of parameters itself, but it also tells **LinuxCNC** which other files to load and use.

## SEE ALSO

**LinuxCNC(1)**

Much more information about LinuxCNC and HAL is available in the LinuxCNC and HAL User Manuals, found at `/usr/share/doc/LinuxCNC/`.

## HISTORY

## BUGS

None known at this time.

## AUTHOR

This man page written by Alex Joni, as part of the LinuxCNC project.

## REPORTING BUGS

Report bugs to alex\_joni AT users DOT sourceforge DOT net

## COPYRIGHT

Copyright © 2007 Alex Joni.

This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

**NAME**

elbpcom – Communicate with Mesa ethernet cards

**SYNOPSIS**

Common options:

**elbpcom** [--ip=*IP*] [--port=*PORT*] [--timeout=*TIMEOUT*]

Reading data:

**elbpcom** [*common options*] --space=*SPACE* [--info] --address=*ADDRESS* --read=*LENGTH*

Writing data:

**elbpcom** [*common options*] --space=*SPACE* --address=*ADDRESS* --write=*HEXDATA*

Sending arbitrary packets:

**elbpcom** [*common options*] *HEXDATA*

**DESCRIPTION**

Read or write data from a Mesa ethernet card that uses the LBP16 protocol, such as the 7i80. This can be useful for performing certain low-level tasks.

For more information about the meaning of each address space, see the card documentation. Incorrect use of this utility can have negative effects such as changing the board's IP address or even corrupting the FPGA bitfile in the eeprom. For some tasks, such as updating FPGA bitfiles and setting IP addresses, **mesaflash**(1) is a more appropriate tool.

If not specified, the default values are

**--ip=192.168.1.1 --port=27181 --timeout=.2**

This example demonstrates reading the HOSTMOT2 identifying string from the IDROM in space 0:

```
$ elbpcom --space 0 --address 0x104 --read 8
> 82420401
< 484f53544d4f5432
HOSTMOT2
```

First the request is shown in hex. Then the response (if any) is shown in hex. Finally, the response is shown in ASCII, with "." replacing any non-ASCII characters. This is similar to the following invocations of mesaflash:

```
$ ./mesaflash --device 7i80 --rpo 0x104
54534F48
$ ./mesaflash --device 7i80 --rpo 0x108
32544F4D
```

but notice its different treatment of byte order.

**SEE ALSO**

**mesaflash**(1), **hostmot2**(9), **hm2\_eth**(9), Mesa's documentation for the Anything I/O boards (<http://www.mesanet.com>).

**NAME**

gladevcp – Virtual Control Panel for LinuxCNC based on Glade, Gtk and HAL widgets

**SYNOPSIS**

**gladevcp** [-g *WxH+X+Y*] [-c *component-name*] [-u *handler*] [-U *useroption*] [-H *halfile*] [-d] *myfile.ui*

**OPTIONS**

**-g** *WxH+X+Y*

This sets the initial geometry of the root window. Use 'WxH' for just size, '+X+Y' for just position, or 'WxH+X+Y' for both. Size / position use pixel units. Position is referenced from top left.

**-c** *component-name*

Use *component-name* as the HAL component name. If the component name is not specified, the basename of the ui file is used.

**-u** *handler*

Instructs gladevcp to inspect the Python script *handler* for event handlers, and connect them to signals in the ui file.

**-U** *useroption*

gladevcp collects all *useroption* strings and passes them to the handler *init()* method as a list of strings without further inspection.

**-x** *XID* Reparent gladevcp into an existing window *XID* instead of creating a new top level window.

**-H** *halfile*

gladevcp runs *halfile* - a list of HAL commands - by executing *halcmd -c halfile* after the HAL component is finalized.

**-d** enable debug output.

**-R** *gtkrcfile*

explicitly load a gtkrc file.

**-t** *THEME*

set gtk theme. Default is *system* theme. Different panels can have different themes.

**-m** *MAXIMUM*

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

**-R**

explicitly deactivate workaround for a gtk bug which makes matches of widget and widget\_class matches in gtk theme and gtkrc files fail. Normally not needed.

**SEE ALSO**

*GladeVCP* in the LinuxCNC documentation for a description of gladevcp's capabilities and the associated HAL widget set, along with examples

## NAME

**gs2\_vfd** - HAL userspace component for Automation Direct GS2 VFD's

## SYNOPSIS

**gs2\_vfd** [OPTIONS]

## DESCRIPTION

This manual page explains the **gs2\_vfd** component. This component reads and writes to the GS2 via a modbus connection.

**gs2\_vfd** is for use with LinuxCNC

## OPTIONS

**-b, --bits <n>**

(default 8) Set number of data bits to <n>, where n must be from 5 to 8 inclusive

**-d, --device <path>**

(default /dev/ttyS0) Set the name of the serial device node to use.

**-v, --verbose**

Turn on verbose mode.

**-g, --debug**

Turn on debug messages. Note that if there are serial errors, this may become annoying. Debug mode will cause all modbus messages to be printed in hex on the terminal.

**-n, --name <string>**

(default gs2\_vfd) Set the name of the HAL module. The HAL comp name will be set to <string>, and all pin and parameter names will begin with <string>.

**-p, --parity [even,odd,none]**

(default odd) Set serial parity to even, odd, or none.

**-r, --rate <n>**

(default 38400) Set baud rate to <n>. It is an error if the rate is not one of the following: 110, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200

**-s, --stopbits [1,2]**

(default 1) Set serial stop bits to 1 or 2

**-t, --target <n>**

(default 1) Set MODBUS target (slave) number. This must match the device number you set on the GS2.

**-A, --accel-seconds <n>**

(default 10.0) Seconds to accelerate the spindle from 0 to Max RPM.

**-D, --decel-seconds <n>**

(default 0.0) Seconds to decelerate the spindle from Max RPM to 0. If set to 0.0 the spindle will be allowed to coast to a stop without controlled deceleration.

**-R, --braking-resistor**

This argument should be used when a braking resistor is installed on the GS2 VFD (see Appendix A of the GS2 manual). It disables deceleration over-voltage stall prevention (see GS2 modbus Parameter 6.05), allowing the VFD to keep braking even in situations where the motor is regenerating high voltage. The regenerated voltage gets safely dumped into the braking resistor.

## PINS

- <name>.DC-bus-volts (float, out)**  
from the VFD
- <name>.at-speed (bit, out)**  
when drive is at commanded speed
- <name>.err-reset (bit, in)**  
reset errors sent to VFD
- <name>.firmware-revision (s32, out)**  
from the VFD
- <name>.frequency-command (float, out)**  
from the VFD
- <name>.frequency-out (float, out)**  
from the VFD
- <name>.is-stopped (bit, out)**  
when the VFD reports 0 Hz output
- <name>.load-percentage (float, out)**  
from the VFD
- <name>.motor-RPM (float, out)**  
from the VFD
- <name>.output-current (float, out)**  
from the VFD
- <name>.output-voltage (float, out)**  
from the VFD
- <name>.power-factor (float, out)**  
from the VFD
- <name>.scale-frequency (float, out)**  
from the VFD
- <name>.speed-command (float, in)**  
speed sent to VFD in RPM It is an error to send a speed faster than the Motor Max RPM as set in the VFD
- <name>.spindle-fwd (bit, in)**  
1 for FWD and 0 for REV sent to VFD
- <name>.spindle-on (bit, in)**  
1 for ON and 0 for OFF sent to VFD, only on when running
- <name>.spindle-rev (bit, in)**  
1 for ON and 0 for OFF, only on when running
- <name>.status-1 (s32, out)**  
Drive Status of the VFD (see the GS2 manual)
- <name>.status-2 (s32, out)**  
Drive Status of the VFD (see the GS2 manual) Note that the value is a sum of all the bits that are on. So a 163 which means the drive is in the run mode is the sum of 3 (run) + 32 (freq set by serial) + 128 (operation set by serial).

## PARAMETERS

- <name>.error-count (s32, RW)**

- <name>.loop-time (float, RW)**  
how often the modbus is polled (default 0.1)
- <name>.nameplate-HZ (float, RW)**  
Nameplate Hz of motor (default 60)
- <name>.nameplate-RPM (float, RW)**  
Nameplate RPM of motor (default 1730)
- <name>.retval (s32, RW)**  
the return value of an error in HAL
- <name>.tolerance (float, RW)**  
speed tolerance (default 0.01)
- <name>.ack-delay (s32, RW)**  
number of read/write cycles before checking at-speed (default 2)

**SEE ALSO**

*GS2 Driver* in the LinuxCNC documentation for a full description of the **GS2** syntax

*GS2 Examples* in the LinuxCNC documentation for examples using the **GS2** component

**BUGS****AUTHOR**

John Thornton

**LICENSE**

GPL

**NAME**

`hal_input` – control HAL pins with any Linux input device, including USB HID devices

**SYNOPSIS**

`loadusr hal_input [-KRAL] inputspec ...`

**DESCRIPTION**

`hal_input` is an interface between HAL and any Linux input device, including USB HID devices. For each device named, **hal\_input** creates pins corresponding to its keys, absolute axes, and LEDs. At a fixed rate of approximately 10ms, it synchronizes the device and the HAL pins.

**INPUT SPECIFICATION**

The *inputspec* may be in one of several forms:

A string *S*

A substring or shell-style pattern match will be tested against the "name" of the device, the "phys" (which gives information about how it is connected), and the "id", which is a string of the form "Bus=... Vendor=... Product=... Version=...". You can view the name, phys, and id of attached devices by executing **less /proc/bus/input/devices**. Examples:

```
SpaceBall
"Vendor=001f Product=0001"
serio*/input0
```

A number *N*

This opens `/dev/input/eventN`. Except for devices that are always attached to the system, this number may change over reboots or when the device is removed. For this reason, using an integer is not recommended.

When several devices are identified by the same string, add `":N"` where *N* is the index of the desired device. For example, if **Mouse** matches **input3** and **input10**, then **Mouse** and **Mouse:0** select **input3**. Specifying **mouse:1** selects **input10**.

For devices that appear as multiple entries in `/dev/input`, these indices are likely to stay the same every time. For multiple identical devices, these indices are likely to depend on the insertion order, but stay the same across reboots as long as the devices are not moved to different ports or unplugged while the machine is booted.

If the first character of the *inputspec* is a "+", then **hal\_input** requests exclusive access to the device. The first device matching an *inputspec* is used. Any number of *inputspecs* may be used.

A *subset option* may precede each *inputspec*. The subset option begins with a dash. Each letter in the subset option specifies a device feature to **include**. Features that are not specified are excluded. For instance, to export keyboard LEDs to HAL without exporting keys, use

```
hal_input -L keyboard ...
```

**DEVICE FEATURES SUPPORTED**

- EV\_KEY (buttons and keys). Subset -K
- EV\_ABS (absolute analog inputs). Subset -A
- EV\_REL (relative analog inputs). Subset -R
- EV\_LED (LED outputs). Subset -L

**HAL PINS AND PARAMETERS**

For buttons

**input.N.btn-name** bit out

**input.N.btn-name-not** bit out

Created for each button on the device.

**For keys****input.N.key-name****input.N.key-name-not**

Created for each key on the device.

**For absolute axes****input.N.abs-name-counts** s32 out**input.N.abs-name-position** float out**input.N.abs-name-scale** parameter float rw**input.N.abs-name-offset** parameter float rw**input.N.abs-name-fuzz** parameter s32 rw**input.N.abs-name-flat** parameter s32 rw**input.N.abs-name-min** parameter s32 r**input.N.abs-name-max** parameter s32 r

Created for each absolute axis on the device. Device positions closer than **flat** to **offset** are reported as **offset** in **counts**, and **counts** does not change until the device position changes by at least **fuzz**. The position is computed as **position** = (**counts** - **offset**) / **scale**. The default value of **scale** and **offset** map the range of the axis reported by the operating system to [-1,1]. The default values of **fuzz** and **flat** are those reported by the operating system. The values of **min** and **max** are those reported by the operating system.

**For relative axes****input.N.rel-name-counts** s32 out**input.N.rel-name-position** float out**input.N.rel-name-reset** bit in**input.N.rel-name-scale** parameter float rw**input.N.rel-name-absolute** parameter s32 rw**input.N.rel-name-precision** parameter s32 rw**input.N.rel-name-last** parameter s32 rw

Created for each relative axis on the device. As long as **reset** is true, **counts** is reset to zero regardless of any past or current axis movement. Otherwise, **counts** increases or decreases according to the motion of the axis. **counts** is divided by position-scale to give **position**. The default value of **position** is 1. There are some devices, notably scroll wheels, which return signed values with less resolution than 32 bits. The default value of **precision** is 32. **precision** can be set to 8 for a device that returns signed 8 bit values, or any other value from 1 to 32. **absolute**, when set true, ignores duplicate events with the same value. This allows for devices that repeat events without any user action to work correctly. **last** shows the most recent count value returned by the device, and is used in the implementation of **absolute**.

**For LEDs****input.N.led-name** bit out**input.N.led-name-invert** parameter bit rw

Created for each LED on the device.

**PERMISSIONS AND UDEV**

By default, the input devices may not be accessible to regular users--**hal\_input** requires read-write access, even if the device has no outputs. To change the default permission of a device, add a new file to `/etc/udev/rules.d` to set the device's GROUP to "plugdev". You can do this for all input devices with this rule:

```
SUBSYSTEM=="input", MODE="0660", GROUP="plugdev"
```

You can also make more specific rules for particular devices. For instance, a SpaceBall input device uses the 'spaceball' kernel module, so a udev entry for it would read:

```
DRIVER=="spaceball", MODE="0660", GROUP="plugdev"
```

the next time the device is attached to the system, it will be accessible to the "plugdev" group.

For USB devices, the udev line would refer to the device's Vendor and Product values, such as

```
SYSFS{idProduct}=="c00e", SYSFS{idVendor}=="046d", MODE="0660", GROUP="plugdev"
```

for a particular logitech-brand mouse.

For more information on writing udev rules, see **udev(8)**.

## **BUGS**

The initial state of keys, buttons, and absolute axes are erroneously reported as FALSE or 0 until an event is received for that key, button, or axis.

## **SEE ALSO**

**udev(8)**

**NAME**

halcmd – manipulate the LinuxCNC HAL from the command line

**SYNOPSIS**

**halcmd** [*OPTIONS*] [*COMMAND*] [*ARG*]

**DESCRIPTION**

**halcmd** is used to manipulate the HAL (Hardware Abstraction Layer) from the command line. **halcmd** can optionally read commands from a file, allowing complex HAL configurations to be set up with a single command.

If the **readline** library is available when LinuxCNC is compiled, then **halcmd** offers commandline editing and completion when running interactively. Use the up arrow to recall previous commands, and press tab to complete the names of items such as pins and signals.

**OPTIONS**

- I** Before tearing down the realtime environment, run an interactive halcmd. **halrun** only. If **-I** is used, it must precede all other commandline arguments.
- f** [*file*] Ignore commands on command line, take input from *file* instead. If *file* is not specified, take input from *stdin*.
- i** *infile* Use variables from *infile* for substitutions. See **SUBSTITUTION** below.
- k** Keep going after failed command(s). The default is to stop and return failure if any command fails.
- q** display errors only (default)
- Q** display nothing, execute commands silently
- s** Script-friendly mode. In this mode, *show* will not output titles for the items shown. Also, module names will be printed instead of ID codes in pin, param, and funct listings. Threads are printed on a single line, with the thread period, FP usage and name first, followed by all of the functions in the thread, in execution order. Signals are printed on a single line, with the type, value, and signal name first, followed by a list of pins connected to the signal, showing both the direction and the pin name.
- R** Release the HAL mutex. This is useful for recovering when a HAL component has crashed while holding the HAL mutex.
- v** display results of each command
- V** display lots of debugging junk
- h** [*command*] display a help screen and exit, displays extended help on *command* if specified

**COMMANDS**

Commands tell **halcmd** what to do. Normally **halcmd** reads a single command from the command line and executes it. If the **'-f'** option is used to read commands from a file, **halcmd** reads each line of the file as a new command. Anything following **'#'** on a line is a comment.

**loadrt** *modname*

(load realtime module) Loads a realtime HAL module called *modname*. **halcmd** looks for the module in a directory specified at compile time.

In systems with realtime, **halcmd** calls the **linuxcnc\_module\_helper** to load realtime modules. **linuxcnc\_module\_helper** is a setuid program and is compiled with a whitelist of modules it is allowed to load. This is currently just a list of **LinuxCNC**-related modules. The **linuxcnc\_module\_helper** execs insmod, so return codes and error messages are those from insmod. Administrators who wish to restrict which users can load these **LinuxCNC**-related kernel modules can do this

by setting the permissions and group on **linuxcnc\_module\_helper** appropriately.

In systems without realtime **halcmd** calls the **rtapi\_app** which creates the simulated realtime environment if it did not yet exist, and then loads the requested component with a call to **dlopen(3)**.

**unloadrt** *modname*

(*unload* realtime module) Unloads a realtime HAL module called *modname*. If *modname* is "all", it will unload all currently loaded realtime HAL modules. **unloadrt** also works by execing **linux-cnc\_module\_helper** or **rtapi\_app**, just like **loadrt**.

**loadusr** [*flags*] *unix-command*

(*load* Userspace component) Executes the given *unix-command*, usually to load a userspace component. [*flags*] may be one or more of:

- **-W** to wait for the component to become ready. The component is assumed to have the same name as the first argument of the command.
- **-Wn name** to wait for the component, which will have the given name.
- **-w** to wait for the program to exit
- **-i** to ignore the program return value (with -w)

**waitusr** *name*

(*wait* for Userspace component) Waits for user space component *name* to disconnect from HAL (usually on exit). The component must already be loaded. Usefull near the end of a HAL file to wait until the user closes some user interface component before cleaning up and exiting.

**unloadusr** *compname*

(*unload* Userspace component) Unloads a userspace component called *compname*. If *compname* is "all", it will unload all userspace components. **unloadusr** works by sending SIGTERM to all userspace components.

**unload** *compname*

Unloads a userspace component or realtime module. If *compname* is "all", it will unload all userspace components and realtime modules.

**newsig** *signame type*

(OBSOLETE - use **net** instead) (*new* signal) Creates a new HAL signal called *signame* that may later be used to connect two or more HAL component pins. *type* is the data type of the new signal, and must be one of "bit", "s32", "u32", or "float". Fails if a signal of the same name already exists.

**delsig** *signame*

(*delete* signal) Deletes HAL signal *signame*. Any pins currently linked to the signal will be unlinked. Fails if *signame* does not exist.

**sets** *signame value*

(*set* signal) Sets the value of signal *signame* to *value*. Fails if *signame* does not exist, if it already has a writer, or if *value* is not a legal value. Legal values depend on the signals's type.

**stype** *name*

(signal type) Gets the type of signal *name*. Fails if *name* does not exist as a signal.

**gets** *signame*

(*get* signal) Gets the value of signal *signame*. Fails if *signame* does not exist.

**linkps** *pinname* [*arrow*] *signame*

(OBSOLETE - use **net** instead) (*link* pin to signal) Establishes a link between a HAL component pin *pinname* and a HAL signal *signame*. Any previous link to *pinname* will be broken. *arrow* can be "=>", "<=", "<=>", or omitted. **halcmd** ignores arrows, but they can be useful in command files to document the direction of data flow. Arrows should not be used on the command line since

the shell might try to interpret them. Fails if either *pinname* or *signame* does not exist, or if they are not the same type type.

**linksp** *signame* [*arrow*] *pinname*

(OBSOLETE - use **net** instead) (*link signal to pin*) Works like **linkps** but reverses the order of the arguments. **halcmd** treats both link commands exactly the same. Use whichever you prefer.

**linkpp** *pinname1* [*arrow*] *pinname2*

(OBSOLETE - use **net** instead) (*link pin to pin*) Shortcut for **linkps** that creates the signal (named like the first pin), then links them both to that signal. **halcmd** treats this just as if it were:

**halcmd newsig** *pinname1*

**halcmd linksp** *pinname1* *pinname1*

**halcmd linksp** *pinname1* *pinname2*

**net** *signame* *pinname* ...

Create *signame* to match the type of *pinname* if it does not yet exist. Then, link *signame* to each *pinname* in turn. Arrows may be used as in **linkps**. When linking a pin to a signal for the first time, the signal value will inherit the pin's default value.

**unlinkp** *pinname*

(*unlink pin*) Breaks any previous link to *pinname*. Fails if *pinname* does not exist. An unlinked pin will retain the last value of the signal it was linked to.

**setp** *name value*

(*set parameter or pin*) Sets the value of parameter or pin *name* to *value*. Fails if *name* does not exist as a pin or parameter, if it is a parameter that is not writable, if it is a pin that is an output, if it is a pin that is already attached to a signal, or if *value* is not a legal value. Legal values depend on the type of the pin or parameter. If a pin and a parameter both exist with the given name, the parameter is acted on.

*paramname* = *value*

*pinname* = *value*

Identical to **setp**. This alternate form of the command may be more convenient and readable when used in a file.

**ptype** *name*

(*parameter or pin type*) Gets the type of parameter or pin *name*. Fails if *name* does not exist as a pin or parameter. If a pin and a parameter both exist with the given name, the parameter is acted on.

**getp** *name*

(*get parameter or pin*) Gets the value of parameter or pin *name*. Fails if *name* does not exist as a pin or parameter. If a pin and a parameter both exist with the given name, the parameter is acted on.

**addf** *funcname threadname*

(*add function*) Adds function *funcname* to realtime thread *threadname*. *funcname* will run after any functions that were previously added to the thread. Fails if either *funcname* or *threadname* does not exist, or if they are incompatible.

**delf** *funcname threadname*

(*delete function*) Removes function *funcname* from realtime thread *threadname*. Fails if either *funcname* or *threadname* does not exist, or if *funcname* is not currently part of *threadname*.

**start** Starts execution of realtime threads. Each thread periodically calls all of the functions that were added to it with the **addf** command, in the order in which they were added.

**stop** Stops execution of realtime threads. The threads will no longer call their functions.

**show** [*item*]

Prints HAL items to *stdout* in human readable format. *item* can be one of "**comp**" (components), "**pin**", "**sig**" (signals), "**param**" (parameters), "**funct**" (functions), "**thread**", or "**alias**". The type "**all**" can be used to show matching items of all the preceeding types. If *item* is omitted, **show** will print everything.

**item** This is equivalent to **show all** [*item*].

**save** [*item*]

Prints HAL items to *stdout* in the form of HAL commands. These commands can be redirected to a file and later executed using **halcmd -f** to restore the saved configuration. *item* can be one of the following: "**comp**" generates a **loadrt** command for realtime component. "**sig**" generates a **newsig** command for each signal, and "**sigu**" generates a **newsig** command for each unlinked signal (for use with **netl** and **netla**). "**link**" and "**linka**" both generate **linkps** commands for each link. (**linka** includes arrows, while **link** does not.) "**net**" and "**neta**" both generate one **newsig** command for each signal, followed by **linksp** commands for each pin linked to that signal. (**neta** includes arrows.) "**netl**" generates one **net** command for each linked signal, and "**netla**" generates a similar command using arrows. "**param**" generates one **setp** command for each parameter. "**thread**" generates one **addf** command for each function in each realtime thread. If *item* is omitted, **save** does the equivalent of **comp**, **sigu**, **link**, **param**, and **thread**.

**source** *filename.hal*

Execute the commands from *filename.hal*.

**alias** *type name alias*

Assigns "**alias**" as a second name for the pin or parameter "*name*". For most operations, an alias provides a second name that can be used to refer to a pin or parameter, both the original name and the alias will work.

"*type*" must be **pin** or **param**.

"*name*" must be an existing name or **alias** of the specified type.

**unalias** *type alias*

Removes any alias from the pin or parameter alias.

"*type*" must be **pin** or **param**

"*alias*" must be an existing name or **alias** of the specified type.

**list** *type* [*pattern*]

Prints the names of HAL items of the specified type.

'*type*' is '**comp**', '**pin**', '**sig**', '**param**', '**funct**', or

'**thread**'. If '*pattern*' is specified it prints only

those names that match the pattern, which may be a

'shell glob'.

For '**sig**', '**pin**' and '**param**', the first pattern may be

-**tdatatype** where datatype is the data type (e.g., 'float')

in this case, the listed pins, signals, or parameters

are restricted to the given data type

Names are printed on a single line, space separated.

**lock** [*all|tune|none*]

Locks HAL to some degree.

none - no locking done.

tune - some tuning is possible (**setp** & such).

all - HAL completely locked.

**unlock** [*all|tune*]

Unlocks HAL to some degree.

tune - some tuning is possible (**setp** & such).

all - HAL completely unlocked.

**status** [*type*]

Prints status info about HAL.

'type' is 'lock', 'mem', or 'all'.

If 'type' is omitted, it assumes 'all'.

**help** [*command*]

Give help information for command.

If 'command' is omitted, list command and brief description

**SUBSTITUTION**

After a command is read but before it is executed, several types of variable substitution take place.

**Environment Variables**

Environment variables have the following formats:

**\$ENVVAR** followed by end-of-line or whitespace

**\$(ENVVAR)**

**Infile Variables**

Infile variables are available only when an infile was specified with the halcmd **-i** flag. They have the following formats:

**[SECTION]VAR** followed by end-of-line or whitespace

**[SECTION](VAR)**

**EXAMPLES****HISTORY****BUGS**

None known at this time.

**AUTHOR**

Original version by John Kasunich, as part of the LinuxCNC project. Now includes major contributions by several members of the project.

**REPORTING BUGS**

Report bugs to the [LinuxCNC bug tracker](http://sf.net/p/emc/bugs/) (<http://sf.net/p/emc/bugs/>).

**COPYRIGHT**

Copyright © 2003 John Kasunich.

This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

**SEE ALSO**

**halrun(1)** -- a convenience script to start a realtime environment, process a .hal or a .tcl file, and optionally start an interactive command session using **halcmd** (described here) or **haltcl(1)**.

**NAME**

halcompile – Build, compile and install LinuxCNC HAL components

**SYNOPSIS**

```
halcompile [--compile|--preprocess|--document|--view-doc] compfile...  
sudo halcompile [--install|--install-doc] compfile...  
      halcompile --compile --userspace cfile...  
sudo halcompile --install --userspace cfile...  
sudo halcompile --install --userspace pyfile...
```

**DESCRIPTION**

**halcompile** performs many different functions:

- Compile **.comp** and **.c** files into **.so** or **.ko** HAL realtime components (the **--compile** flag)
- Compile **.comp** and **.c** files into HAL userspace components (the **--compile --userspace** flag)
- Preprocess **.comp** files into **.c** files (the **--preprocess** flag)
- Extract documentation from **.comp** files into **.9** manpage files (the **--document** flag)
- Display documentation from **.comp** files onscreen (the **--view-doc** flag)
- Compile and install **.comp** and **.c** files into the proper directory for HAL realtime components (the **--install** flag), which may require *sudo* to write to system directories.
- Install **.c** and **.py** files into the proper directory for HAL userspace components (the **--install --userspace** flag), which may require *sudo* to write to system directories.
- Extract documentation from **.comp** files into **.9** manpage files in the proper system directory (the **--install** flag), which may require *sudo* to write to system directories.
- Preprocess **.comp** files into **.c** files (the **--preprocess** flag)

**SEE ALSO**

*Halcompile HAL Component Generator* in the LinuxCNC documentation for a full description of the **.comp** syntax, along with examples

**pydoc hal** and *Creating Userspace Python Components* in the LinuxCNC documentation for documentation on the Python interface to HAL components

**NAME**

halmeter – observe HAL pins, signals, and parameters

**SYNOPSIS**

**halmeter** [-s] [**pin**|**sig**|**param** *name*] [-g *X-position Y-position [Width]*]

**DESCRIPTION**

**halmeter** is used to observe HAL (Hardware Abstraction Layer) pins, signals, or parameters. It serves the same purpose as a multimeter does when working on physical systems.

**OPTIONS**

**pin** *name*

display the HAL pin *name*.

**sig** *name*

display the HAL signal *name*.

**param** *name*

display the HAL parameter *name*.

If neither **pin**, **sig**, or **param** are specified, the

window starts out blank and the user must select an item to observe.

**-s** small window. Non-interactive, must be used with **pin**, **sig**, or **param** to select the item to display. The item name is displayed in the title bar instead of the window, and there are no "Select" or "Exit" buttons. Handy when you want a lot of meters in a small space.

**-g** geometry position. allows one to specify the initial starting position and optionally the width of the meter. Referenced from top left of screen in pixel units. Handy when you want to load a lot of meters in a script with out them displaying on top of each other.

**USAGE**

Unless **-s** is specified, there are two buttons, "Select" and "Exit". "Select" opens a dialog box to select the item (pin, signal, or parameter) to be observed. "Exit" does what you expect.

The selection dialog has "OK" "Apply", and "Cancel" buttons. OK displays the selected item and closes the dialog. "Apply" displays the selected item but keeps the selection dialog open. "Cancel" closes the dialog without changing the displayed item.

**EXAMPLES**

**halmeter**

Opens a meter window, with nothing initially displayed. Use the "Select" button to choose an item to observe. Does not return until the window is closed.

**halmeter &**

Open a meter window, with nothing initially displayed. Use the "Select" button to choose an item. Runs in the background leaving the shell free for other commands.

**halmeter pin parport.0.pin-03-out &**

Open a meter window, initially displaying HAL pin *parport.0.pin-03-out*. The "Select" button can be used to display other items. Runs in background.

**halmeter -s pin parport.0.pin-03-out &**

Open a small meter window, displaying HAL pin *parport.0.pin-03-out*. The displayed item cannot be changed. Runs in background.

**halmeter -s pin parport.0.pin-03-out -g 100 500 &**

Open a small meter window, displaying HAL pin *parport.0.pin-03-out*. places it 100 pixels to the left and 500 pixels down from top of screen. The displayed item cannot be changed. Runs in background.

**halmeter -s pin** *parport.0.pin-03-out -g 100 500 400 &*

Open a small meter window, displaying HAL pin *parport.0.pin-03-out*. places it 100 pixels to the left and 500 pixels down from top of screen. The width will be 400 pixels (270 is default) The displayed item cannot be changed. Runs in background.

## SEE ALSO

## HISTORY

## BUGS

## AUTHOR

Original version by John Kasunich, as part of the LinuxCNC project. Improvements by several other members of the LinuxCNC development team.

## REPORTING BUGS

Report bugs to [jmkasunich AT users DOT sourceforge DOT net](mailto:jmkasunich@users.sourceforge.net)

## COPYRIGHT

Copyright © 2003 John Kasunich.

This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

## NAME

**halrun** – manipulate the LinuxCNC HAL from the command line

## SYNOPSIS

**halrun** -h

**halrun** [-I] [halcmd\_opts] [filename[.hal|.tcl]]

**halrun** -T [halcmd\_opts] [filename[.hal|.tcl]]

**halrun** -U

## DESCRIPTION

**halrun** is a convenience script used to manipulate the HAL (Hardware Abstraction Layer) from the command line. When invoked, **halrun**:

- Sets up the realtime environment.
- Executes a command interpreter (**halcmd** or **haltcl**).
- (Optionally) runs an interactive session.
- Tears down the realtime environment.

If no filename is specified, an interactive session is started.  
The session will use **halcmd**(1) unless -T is specified in which case **haltcl**(1) will be used.

If a filename is specified and neither the -I nor the -T option is included, the filename will be processed by the command interpreter corresponding to the filename extension (**halcmd** or **haltcl**). After processing, the realtime environment will be torn down.

If a filename is specified and the -I or -T option is included, the file is processed by the appropriate command interpreter and then an interactive session is started for **halcmd** or **haltcl** according to the -I or -T option.

## OPTIONS

### halcmd\_opts

When a .hal file is specified, the **halcmd\_opts** are passed to **halcmd**. See the man page for **halcmd**(1). When a .tcl file is specified, the only valid options are:

- i infile
- f filename[.tcl|.hal] (alternate means of specifying a file)

**-I** Run an interactive **halcmd** session

**-T** Run an interactive **haltcl** session.

**-U** Forcibly cause the realtime environment to exit. It releases the HAL mutex, requests that all HAL components unload, and stops the realtime system. **-U** must be the only commandline argument.

**-h** display a brief help screen and exit

## EXAMPLES

## HISTORY

## BUGS

None known at this time.

## AUTHOR

Original version by John Kasunich, as part of the LinuxCNC Enhanced Machine Controller project. Now includes major contributions by several members of the project.

**REPORTING BUGS**

Report bugs to the LinuxCNC bug tracker (URL: <http://sf.net/p/emc/bugs/> ).

**COPYRIGHT**

Copyright © 2003 John Kasunich.

This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

**SEE ALSO**

**halcmd(1)**, **haltcl(1)**

**NAME**

**halsampler** – sample data from HAL in realtime

**SYNOPSIS**

**halsampler** [*options*]

**DESCRIPTION**

**sampler**(9) and **halsampler** are used together to sample HAL data in real time and store it in a file. **sampler** is a realtime HAL component that exports HAL pins and creates a FIFO in shared memory. It then begins sampling data from the HAL and storing it to the FIFO. **halsampler** is a user space program that copies data from the FIFO to stdout, where it can be redirected to a file or piped to some other program.

**OPTIONS**

**-c** *CHAN*

instructs **halsampler** to read from FIFO *CHAN*. FIFOs are numbered from zero, and the default value is zero, so this option is not needed unless multiple FIFOs have been created.

**-n** *COUNT*

instructs **halsampler** to read *COUNT* samples from the FIFO, then exit. If **-n** is not specified, **halsampler** will read continuously until it is killed.

**-t**

instructs **halsampler** to tag each line by printing the sample number in the first column.

**FILENAME**

instructs **halsampler** to write to **FILENAME** instead of to stdout.

**USAGE**

A FIFO must first be created by loading **sampler**(9) with **halcmd loadrt** or a **loadrt** command in a .hal file. Then **halsampler** can be invoked to begin printing data from the FIFO to stdout.

Data is printed one line per sample. If **-t** was specified, the sample number is printed first. The data follows, in the order that the pins were defined in the config string. For example, if the **sampler** config string was "ffbs" then a typical line of output (without **-t**) would look like:

```
123.55 33.4 0 -12
```

**halsampler** prints data as fast as possible until the FIFO is empty, then it retries at regular intervals, until it is either killed or has printed *COUNT* samples as requested by **-n**. Usually, but not always, data printed by **halsampler** will be redirected to a file or piped to some other program.

The FIFO size should be chosen to absorb samples captured during any momentary disruptions in the flow of data, such as disk seeks, terminal scrolling, or the processing limitations of subsequent program in a pipeline. If the FIFO gets full and **sampler** is forced to overwrite old data, **halsampler** will print 'overrun' on a line by itself to mark each gap in the sampled data. If **-t** was specified, gaps in the sequential sample numbers in the first column can be used to determine exactly how many samples were lost.

The data format for **halsampler** output is the same as for **halstreamer**(1) input, so 'waveforms' captured with **halsampler** can be replayed using **halstreamer**. The **-t** option should not be used in this case.

**EXIT STATUS**

If a problem is encountered during initialization, **halsampler** prints a message to stderr and returns failure.

Upon printing *COUNT* samples (if **-n** was specified) it will shut down and return success. If it is terminated before printing the specified number of samples, it returns failure. This means that when **-n** is not specified, it will always return failure when terminated.

**SEE ALSO**

**sampler**(9) **streamer**(9) **halstreamer**(1)

**HISTORY****BUGS****AUTHOR**

Original version by John Kasunich, as part of the LinuxCNC project. Improvements by several other members of the LinuxCNC development team.

**REPORTING BUGS**

Report bugs to [jmkasunich AT users DOT sourceforge DOT net](mailto:jmkasunich@users.sourceforge.net)

**COPYRIGHT**

Copyright © 2006 John Kasunich.

This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

**NAME**

**halstreamer** – stream file data into HAL in real time

**SYNOPSIS**

**halstreamer** [*options*]

**DESCRIPTION**

**streamer**(9) and **halstreamer** are used together to stream data from a file into the HAL in real time. **streamer** is a realtime HAL component that exports HAL pins and creates a FIFO in shared memory. **hal\_streamer** is a user space program that copies data from stdin into the FIFO, so that **streamer** can write it to the HAL pins.

**OPTIONS**

**-c** *CHAN*

instructs **halstreamer** to write to FIFO *CHAN*. FIFOs are numbered from zero, and the default value is zero, so this option is not needed unless multiple FIFOs have been created.

**FILENAME**

instructs **halsampler** to read from **FILENAME** instead of from stdin.

**USAGE**

A FIFO must first be created by loading **streamer**(9) with **halcmd loadrt** or a **loadrt** command in a .hal file. Then **halstreamer** can be invoked to begin writing data into the FIFO.

Data is read from stdin, and is almost always either redirected from a file or piped from some other program, since keyboard input would be unable to keep up with even slow streaming rates.

Each line of input must match the pins that are attached to the FIFO, for example, if the **streamer** config string was "ffbs" then each line of input must consist of two floats, a bit, and a signed integer, in that order and separated by whitespace. Floats must be formatted as required by **strtod**(3), signed and unsigned integers must be formatted as required by **strtol**(3) and **strtoul**(3), and bits must be either '0' or '1'.

**halstreamer** transfers data to the FIFO as fast as possible until the FIFO is full, then it retries at regular intervals, until it is either killed or reads **EOF** from stdin. Data can be redirected from a file or piped from some other program.

The FIFO size should be chosen to ride through any momentary disruptions in the flow of data, such as disk seeks. If the FIFO is big enough, **halstreamer** can be restarted with the same or a new file before the FIFO empties, resulting in a continuous stream of data.

The data format for **halstreamer** input is the same as for **halsampler**(1) output, so 'waveforms' captured with **halsampler** can be replayed using **halstreamer**.

**EXIT STATUS**

If a problem is encountered during initialization, **halstreamer** prints a message to stderr and returns failure.

If a badly formatted line is encountered while writing to the FIFO, it prints a message to stderr, skips the line, and continues (this behavior may be revised in the future).

Upon reading **EOF** from the input, it returns success. If it is terminated before the input ends, it returns failure.

**SEE ALSO**

**streamer**(9) **sampler**(9) **halsampler**(1)

**HISTORY**

**BUGS****AUTHOR**

Original version by John Kasunich, as part of the LinuxCNC project. Improvements by several other members of the LinuxCNC development team.

**REPORTING BUGS**

Report bugs to [jmkasunich AT users DOT sourceforge DOT net](mailto:jmkasunich@users.sourceforge.net)

**COPYRIGHT**

Copyright © 2006 John Kasunich.

This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

**NAME**

**haltcl** – manipulate the LinuxCNC HAL from the command line using a tcl interpreter.

**SYNOPSIS**

**haltcl** [*-i inifile*] [*filename*]

**DESCRIPTION**

**haltcl** is used to manipulate the HAL (Hardware Abstraction Layer) from the command line using a tcl interpreter. **haltcl** can optionally read commands from a file (*filename*), allowing complex HAL configurations to be set up with a single command.

**OPTIONS**

**-i inifile**

If specified, the inifile is read and used to create tcl global variable arrays. An array is created for each SECTION of the inifile with elements for each ITEM in the section.

For example, if the inifile contains:

```
[SECTION_A]ITEM_1 = 1
[SECTION_A]ITEM_2 = 2
[SECTION_B]ITEM_1 = 10
```

The corresponding tcl variables are:

```
SECTION_A(ITEM_1) = 1
SECTION_A(ITEM_2) = 2
SECTION_B(ITEM_1) = 10
```

**-ini inifile** -- declining usage, use **-i inifile**

**filename**

If specified, the tcl commands of **filename** are executed. If no filename is specified, **haltcl** opens an interactive session.

**COMMANDS**

**haltcl** includes the commands of a tcl interpreter augmented with commands for the hal language as described for **halcmd**(1). The augmented commands can be listed with the command:

```
haltcl: hal --commands
```

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

Two of the augmented commands, 'list' and 'gets', require special treatment to avoid conflict with tcl built-in commands having the same names. To use these commands, precede them with the keyword 'hal':

```
hal list
hal gets
```

**REPORTING BUGS**

Report bugs to the LinuxCNC bug tracker (URL: <http://sf.net/p/emc/bugs/> ).

**COPYRIGHT**

This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

**SEE ALSO****halcmd(1), halrun(1)**

**NAME**

halui – observe HAL pins and command LinuxCNC through NML

**SYNOPSIS**

**halui** [-ini <path-to-ini>]

**DESCRIPTION**

**halui** is used to build a User Interface using hardware knobs and switches. It exports a big number of pins, and acts accordingly when these change.

**OPTIONS****-ini name**

use the *name* as the configuration file. Note: halui must find the nml file specified in the ini, usually that file is in the same folder as the ini, so it makes sense to run halui from that folder.

**USAGE**

When run, **halui** will export a large number of pins. A user can connect those to his physical knobs & switches & leds, and when a change is noticed halui triggers an appropriate event.

**halui** expects the signals to be debounced, so if needed (bad knob contact) connect the physical button to a HAL debounce filter first.

**PINS****abort**

**halui.abort** bit in  
pin for clearing most errors

**tool**

**halui.tool.length-offset.a** float out  
current applied tool length offset for the A axis

**halui.tool.length-offset.b** float out  
current applied tool length offset for the B axis

**halui.tool.length-offset.c** float out  
current applied tool length offset for the C axis

**halui.tool.length-offset.u** float out  
current applied tool length offset for the U axis

**halui.tool.length-offset.v** float out  
current applied tool length offset for the V axis

**halui.tool.length-offset.w** float out  
current applied tool length offset for the W axis

**halui.tool.length-offset.x** float out  
current applied tool length offset for the X axis

**halui.tool.length-offset.y** float out  
current applied tool length offset for the Y axis

**halui.tool.length-offset.z** float out  
current applied tool length offset for the Z axis

**halui.tool.number** u32 out  
current selected tool

**spindle**

**halui.spindle.brake-is-on** bit out  
status pin that tells us if brake is on

**halui.spindle.brake-off** bit in  
pin for deactivating the spindle brake

**halui.spindle.brake-on** bit in  
pin for activating the spindle brake

**halui.spindle.decrease** bit in  
a rising edge on this pin decreases the current spindle speed by 100

**halui.spindle.forward** bit in  
a rising edge on this pin makes the spindle go forward

**halui.spindle.increase** bit in  
a rising edge on this pin increases the current spindle speed by 100

**halui.spindle.is-on** bit out  
status pin telling if the spindle is on

**halui.spindle.reverse** bit in  
a rising edge on this pin makes the spindle go reverse

**halui.spindle.runs-backward** bit out  
status pin telling if the spindle is running backward

**halui.spindle.runs-forward** bit out  
status pin telling if the spindle is running forward

**halui.spindle.start** bit in  
a rising edge on this pin starts the spindle

**halui.spindle.stop** bit in  
a rising edge on this pin stops the spindle

#### spindle override

**halui.spindle-override.count-enable** bit in (default: **TRUE**)  
When TRUE, modify spindle override when counts changes.

**halui.spindle-override.counts** s32 in  
counts X scale = spindle override percentage

**halui.spindle-override.decrease** bit in  
pin for decreasing the SO (==scale)

**halui.spindle-override.direct-value** bit in  
pin to enable direct spindle override value input

**halui.spindle-override.increase** bit in  
pin for increasing the SO (+=scale)

**halui.spindle-override.scale** float in  
pin for setting the scale of counts for SO

**halui.spindle-override.value** float out  
current FO value

#### program

**halui.program.block-delete.is-on** bit out  
status pin telling that block delete is on

**halui.program.block-delete.off** bit in  
pin for requesting that block delete is off

**halui.program.block-delete.on** bit in  
pin for requesting that block delete is on

**halui.program.is-idle** bit out  
status pin telling that no program is running

**halui.program.is-paused** bit out  
status pin telling that a program is paused

**halui.program.is-running** bit out  
status pin telling that a program is running

**halui.program.optional-stop.is-on** bit out  
status pin telling that the optional stop is on

**halui.program.optional-stop.off** bit in  
pin requesting that the optional stop is off

**halui.program.optional-stop.on** bit in  
pin requesting that the optional stop is on

**halui.program.pause** bit in  
pin for pausing a program

**halui.program.resume** bit in  
pin for resuming a program

**halui.program.run** bit in  
pin for running a program

**halui.program.step** bit in  
pin for stepping in a program

**halui.program.stop** bit in  
pin for stopping a program (note: this pin does the same thing as halui.abort)

#### mode

**halui.mode.auto** bit in  
pin for requesting auto mode

**halui.mode.is-auto** bit out  
pin for auto mode is on

**halui.mode.is-joint** bit out  
pin showing joint by joint jog mode is on

**halui.mode.is-manual** bit out  
pin for manual mode is on

**halui.mode.is-mdi** bit out  
pin for mdi mode is on

**halui.mode.is-teleop** bit out  
pin showing coordinated jog mode is on

**halui.mode.joint** bit in  
pin for requesting joint by joint jog mode

**halui.mode.manual** bit in  
pin for requesting manual mode

**halui.mode.mdi** bit in  
pin for requesting mdi mode

**halui.mode.teleop** bit in  
pin for requesting coordinated jog mode

#### **mdi** (optional)

**halui.mdi-command-XX** bit in  
**halui** looks for ini variables named [HALUI]MDI\_COMMAND, and exports a pin for each command it finds. When the pin is driven TRUE, **halui** runs the specified MDI command. XX is a two digit number starting at 00. If no [HALUI]MDI\_COMMAND variables are set in the ini file, no halui.mdi-command-XX pins will be exported by halui.

#### **mist**

**halui.mist.is-on** bit out  
pin for mist is on

**halui.mist.off** bit in  
pin for stopping mist

**halui.mist.on** bit in  
pin for starting mist

#### **max-velocity**

**halui.max-velocity.count-enable** bit in (default: **TRUE**)  
When TRUE, modify max velocity when counts changes.

**halui.max-velocity.counts** s32 in  
counts from an encoder for example to change maximum velocity

**halui.max-velocity.decrease** bit in  
pin for decreasing the maximum velocity (-=scale)

**halui.max-velocity.direct-value** bit in  
pin for using a direct value for max velocity

**halui.max-velocity.increase** bit in  
pin for increasing the maximum velocity (+=scale)

**halui.max-velocity.scale** float in  
pin for setting the scale on changing the maximum velocity

**halui.max-velocity.value** float out  
Current value for maximum velocity

#### **machine**

**halui.machine.is-on** bit out  
pin for machine is On/Off

**halui.machine.off** bit in  
pin for setting machine Off

**halui.machine.on** bit in  
pin for setting machine On

#### **lube**

**halui.lube.is-on** bit out  
pin for lube is on

**halui.lube.off** bit in  
pin for stopping lube

**halui.lube.on** bit in  
pin for starting lube

## joint

**halui.joint.N.has-fault** bit out  
status pin telling that joint N has a fault

**halui.joint.N.home** bit in  
pin for homing joint N

**halui.joint.N.is-homed** bit out  
status pin telling that joint N is homed

**halui.joint.N.is-selected** bit out  
status pin that joint N is selected

**halui.joint.N.on-hard-max-limit** bit out  
status pin telling that joint N is on the positive hardware limit

**halui.joint.N.on-hard-min-limit** bit out  
status pin telling that joint N is on the negative hardware limit

**halui.joint.N.on-soft-max-limit** bit out  
status pin telling that joint N is on the positive software limit

**halui.joint.N.on-soft-min-limit** bit out  
status pin telling that joint N is on the negative software limit

**halui.joint.N.select** bit in  
pin for selecting joint N

**halui.joint.N.unhome** bit in  
pin for unhoming joint N

**halui.joint.selected** u32 out  
selected joint

**halui.joint.selected.has-fault** bit out  
status pin selected joint is faulted

**halui.joint.selected.home** bit in  
pin for homing the selected joint

**halui.joint.selected.is-homed** bit out  
status pin telling that the selected joint is homed

**halui.joint.selected.on-hard-max-limit** bit out  
status pin telling that the selected joint is on the positive hardware limit

**halui.joint.selected.on-hard-min-limit** bit out  
status pin telling that the selected joint is on the negative hardware limit

**halui.joint.selected.on-soft-max-limit** bit out  
status pin telling that the selected joint is on the positive software limit

**halui.joint.selected.on-soft-min-limit** bit out  
status pin telling that the selected joint is on the negative software limit

**halui.joint.selected.unhome** bit in  
pin for unhoming the selected joint

**jog****halui.jog.deadband** float in

pin for setting jog analog deadband (jog analog inputs smaller/slower than this are ignored)

**halui.jog-speed** float in

pin for setting jog speed for plus/minus jogging.

**halui.jog.N.analog** float in

pin for jogging the axis N using an float value (e.g. joystick)

**halui.jog.N.increment** float in

pin for setting the jog increment for axis N when using increment-plus/minus

**halui.jog.N.increment-minus** bit in

a rising edge will will make axis N jog in the negative direction by the increment amount

**halui.jog.N.increment-plus** bit in

a rising edge will will make axis N jog in the positive direction by the increment amount

**halui.jog.N.minus** bit in

pin for jogging axis N in negative direction at the halui.jog-speed velocity

**halui.jog.N.plus** bit in

pin for jogging axis N in positive direction at the halui.jog-speed velocity

**halui.jog.selected.increment** float in

pin for setting the jog increment for the selected axis when using increment-plus/minus

**halui.jog.selected.increment-minus** bit in

a rising edge will will make the selected axis jog in the negative direction by the increment amount

**halui.jog.selected.increment-plus** bit in

a rising edge will will make the selected axis jog in the positive direction by the increment amount

**halui.jog.selected.minus** bit in

pin for jogging the selected axis in negative direction at the halui.jog-speed velocity

**halui.jog.selected.plus**

pin for jogging the selected axis bit in in positive direction at the halui.jog-speed velocity

**flood****halui.flood.is-on** bit out

pin for flood is on

**halui.flood.off** bit in

pin for stopping flood

**halui.flood.on** bit in

pin for starting flood

**feed override****halui.feed-override.count-enable** bit in (default: **TRUE**)

When TRUE, modify feed override when counts changes.

**halui.feed-override.counts** s32 in

counts X scale = feed override percentage

**halui.feed-override.decrease** bit in

pin for decreasing the FO (==scale)

**halui.feed-override.direct-value** bit in

pin to enable direct value feed override input

**halui.feed-override.increase** bit in  
pin for increasing the FO (+=scale)

**halui.feed-override.scale** float in  
pin for setting the scale on changing the FO

**halui.feed-override.value** float out  
current Feed Override value

#### rapid override

**halui.rapid-override.count-enable** bit in (default: **TRUE**)  
When TRUE, modify Rapid Override when counts changes.

**halui.rapid-override.counts** s32 in  
counts X scale = Rapid Override percentage

**halui.rapid-override.decrease** bit in  
pin for decreasing the Rapid Override (-=scale)

**halui.rapid-override.direct-value** bit in  
pin to enable direct value Rapid Override input

**halui.rapid-override.increase** bit in  
pin for increasing the Rapid Override (+=scale)

**halui.rapid-override.scale** float in  
pin for setting the scale on changing the Rapid Override

**halui.rapid-override.value** float out  
current Rapid Override value

#### estop

**halui.estop.activate** bit in  
pin for setting Estop (LinuxCNC internal) On

**halui.estop.is-activated** bit out  
pin for displaying Estop state (LinuxCNC internal) On/Off

**halui.estop.reset** bit in  
pin for resetting Estop (LinuxCNC internal) Off

#### axis

**halui.axis.N.pos-commanded** float out float out  
Commanded axis position in machine coordinates

**halui.axis.N.pos-feedback** float out float out  
Feedback axis position in machine coordinates

**halui.axis.N.pos-relative** float out float out  
Commanded axis position in relative coordinates

#### home

**halui.home-all** bit in  
pin for requesting home-all (only available when a valid homing sequence is specified)

#### SEE ALSO

#### HISTORY

#### BUGS

none known at this time.

**AUTHOR**

Written by Alex Joni, as part of the LinuxCNC project. Updated by John Thornton

**REPORTING BUGS**

Report bugs to alex\_joni AT users DOT sourceforge DOT net

**COPYRIGHT**

Copyright © 2006 Alex Joni.

This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

**NAME**

**hy\_vfd** - HAL userspace component for Huanyang VFDs

**SYNOPSIS**

**hy\_vfd** [OPTIONS]

**DESCRIPTION**

This component connects the Huanyang VFD to the LinuxCNC HAL via a serial (RS-485) connection.

The Huanyang VFD must be configured via the face plate user interface to accept serial communications:

**PD001 = 2**

Set register PD001 (source of run commands) to 2 (communication port).

**PD002 = 2**

Set register PD002 (source of operating frequency) to 2 (communication port).

**PD005**

Set register PD005 (max frequency) according to motor specs. This is the maximum frequency of the motor's power supply, in Hz.

**PD011**

Set register PD011 (min frequency) according to motor specs. This is the minimum frequency of the motor's power supply, in Hz.

**PD141**

Set register PD141 (rated motor voltage) according to motor name plate. This is the motor's maximum voltage, in Volts.

**PD142**

Set register PD142 (rated motor current) according to motor name plate. This is the motor's maximum current, in Amps.

**PD144**

Set register PD144 (rated motor revolutions) according to motor name plate. This is the motor's max speed in RPM.

**PD163 = 1**

Set register PD163 (communication address) to 1. This matches the default in the hy\_vfd driver, change this if your setup has special needs.

**PD164 = 2**

Set register PD164 (baud rate) to 2 (19200 bps). This matches the default in the hy\_vfd driver, change this if your setup has special needs.

**PD165 = 3**

Set register PD165 (communication data method) to 3 (8n1 RTU). This matches the default in the hy\_vfd driver, change this if your setup has special needs. Note that the hy\_vfd driver only supports RTU communication, not ASCII.

Consult the Huanyang instruction manual for details on using the face plate to program the VFDs registers, and alternative values for the above registers.

**OPTIONS****-d, --device <path>**

(default /dev/ttyS0) Set the name of the serial device node to use.

**-g, --debug**

Turn on debug messages. Note that if there are serial errors, this may become annoying. Debug mode will cause all serial communication messages to be printed in hex on the terminal.

- n, --name <string>**  
(default hy\_vfd) Set the name of the HAL module. The HAL comp name will be set to <string>, and all pin and parameter names will begin with <string>.
- b, --bits <n>**  
(default 8) Set number of data bits to <n>, where n must be from 5 to 8 inclusive. This must match the setting in register PD165 of the Huanyang VFD.
- p, --parity [even,odd,none]**  
(default odd) Set serial parity to even, odd, or none. This must match the setting in register PD165 of the Huanyang VFD.
- r, --rate <n>**  
(default 38400) Set baud rate to <n>. It is an error if the rate is not one of the following: 110, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200. This must match the setting in register PD164 of the Huanyang VFD.
- s, --stopbits [1,2]**  
(default 1) Set serial stop bits to 1 or 2. This must match the setting in register PD165 of the Huanyang VFD.
- t, --target <n>**  
(default 1) Set HYCOMM target (slave) number. This must match the device number you set on the Huanyang VFD in register PD163.
- F, --max-frequency <n>**  
(default: read from VFD) If specified, program register PD005 of the VFD with the specified max frequency of <n> Hz (and use the same max frequency in the hy\_vfd driver). If not specified, read the max frequency to use from register PD005 of the VFD.
- f, --min-frequency <n>**  
(default: read from VFD) If specified, program register PD011 of the VFD with the specified minimum frequency of <n> Hz (and use the same minimum frequency in the hy\_vfd driver). If not specified, read the minimum frequency to use from register PD011 of the VFD.
- V, --motor-voltage <n>**  
(default: read from VFD) If specified, program register PD141 of the VFD with the specified max motor voltage of <n> Volts. If not specified, read the max motor voltage from register PD141 of the VFD.
- I, --motor-current <n>**  
(default: read from VFD) If specified, program register PD142 of the VFD with the specified max motor current of <n> Amps. If not specified, read the max motor current from register PD142 of the VFD.
- V, --motor-speed <n>**  
(default: read from VFD) If specified, program register PD144 of the VFD with the specified max motor speed of <n> RPM (and use the same max motor speed in the hy\_vfd driver). If not specified, read the max motor speed from register PD144 of the VFD. **Note:** The Huanyang VFD can not store or report max motor speeds greater than 9999 RPM. If a max motor speed of greater than 9999 RPM is needed, you must supply it using this command-line argument. The VFD will be programmed with 1/10th of this max speed, which will cause the VFD display to show 1/10th of the actual speed of the motor.

## PINS

- <name>.enable**  
(bit, in) Enable communication from the hy\_vfd driver to the VFD.
- <name>.SetF**  
(float, out)

**<name>.OutF**  
(float, out)

**<name>.OutA**  
(float, out)

**<name>.Rott**  
(float, out)

**<name>.DCV**  
(float, out)

**<name>.ACV**  
(float, out)

**<name>.Cont**  
(float, out)

**<name>.Tmp**  
(float, out)

**<name>.spindle-forward**  
(bit, in)

**<name>.spindle-reverse**  
(bin, in)

**<name>.spindle-on**  
(bin, in)

**<name>.CNTR**  
(float, out)

**<name>.CNST**  
(float, out)

**<name>.CNST-run**  
(bit, out)

**<name>.CNST-jog**  
(bit, out)

**<name>.CNST-command-rf**  
(bit, out)

**<name>.CNST-running**  
(bit, out)

**<name>.CNST-jogging**  
(bit, out)

**<name>.CNST-running-rf**  
(bit, out)

**<name>.CNST-bracking**  
(bit, out)

**<name>.CNST-track-start**  
(bit, out)

**<name>.speed-command**  
(float, in)

**<name>.spindle-speed-fb**  
(float, out) Current spindle speed as reported by Huanyang VFD.

**<name>.spindle-at-speed-tolerance**

(float, in) Spindle speed error tolerance. If the actual spindle speed is within .spindle-at-speed-tolerance of the commanded speed, then the .spindle-at-speed pin will go True. The default .spindle-at-speed-tolerance is 0.02, which means the actual speed must be within 2% of the commanded spindle speed.

**<name>.spindle-at-speed**

(bit, out) True when the current spindle speed is within .spindle-at-speed-tolerance of the commanded speed.

**<name>.frequency-command**

(float, out)

**<name>.max-freq**

(float, out)

**<name>.freq-lower-limit**

(float, out)

**<name>.rated-motor-voltage**

(float, out)

**<name>.rated-motor-current**

(float, out)

**<name>.rated-motor-rev**

(float, out)

**<name>.hycomm-ok**

(bit, out)

**PARAMETERS****<name>.error-count**

(s32, RW)

**<name>.retval**

(float, RW)

**AUTHOR**

Sebastian Kuzminsky

**LICENSE**

GPL

**NAME**

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

**SYNOPSIS**

**loadusr io** [-ini *inifile*]

**DESCRIPTION**

These pins are created by the userspace IO controller, usually found in \$LINUXCNC\_HOME/bin/io

The signals are turned on and off in userspace - if you have strict timing requirements or simply need more i/o, consider using the realtime synchronized i/o provided by **motion**(9) instead.

The inifile is searched for in the directory from which halcmd was run, unless an absolute path is specified.

**PINS****iocontrol.0.coolant-flood**

(Bit, Out) TRUE when flood coolant is requested

**iocontrol.0.coolant-mist**

(Bit, Out) TRUE when mist coolant is requested

**iocontrol.0.emc-enable-in**

(Bit, In) Should be driven FALSE when an external estop condition exists.

**iocontrol.0.lube**

(Bit, Out) TRUE when lube is requested. This pin gets driven True when the controller comes out of E-stop, and when the "Lube On" command gets sent to the controller. It gets driven False when the controller goes into E-stop, and when the "Lube Off" command gets sent to the controller.

**iocontrol.0.lube\_level**

(Bit, In) Should be driven FALSE when lubrication tank is empty.

**iocontrol.0.tool-change**

(Bit, Out) TRUE when a tool change is requested

**iocontrol.0.tool-changed**

(Bit, In) Should be driven TRUE when a tool change is completed.

**iocontrol.0.tool-number**

(s32, Out) Current tool number

**iocontrol.0.tool-prep-number**

(s32, Out) The number of the next tool, from the RS274NGC T-word

**iocontrol.0.tool-prep-pocket**

(s32, Out) The pocket number (location in tool storage mechanism) of the next tool, as described in the tool table

**iocontrol.0.tool-prepare**

(Bit, Out) TRUE when a  $Tn$  tool prepare is requested

**iocontrol.0.tool-prepared**

(Bit, In) Should be driven TRUE when a tool prepare is completed.

**iocontrol.0.user-enable-out**

(Bit, Out) FALSE when an internal estop condition exists

**iocontrol.0.user-request-enable**

(Bit, Out) TRUE when the user has requested that estop be cleared

**SEE ALSO**

**motion(9)**

**NAME**

linuxcncrsh – text-mode interface for commanding LinuxCNC over the network

**SYNOPSIS**

**linuxcncrsh** [**OPTIONS**] [-- **LINUXCNC\_OPTIONS**]

**DESCRIPTION**

**linuxcncrsh** is a user interface for LinuxCNC. Instead of popping up a GUI window like axis(1) and touchy(1) do, it processes text-mode commands that it receives via the network. A human (or a program) can interface with **linuxcncrsh** using telnet(1) or nc(1) or similar programs.

All features of LinuxCNC are available via the **linuxcncrsh** interface.

**OPTIONS**

**-p,--port PORT\_NUMBER**

Specify the port for linuxcncrsh to listen on. Defaults to 5007 if omitted.

**-n,--name SERVER\_NAME**

Sets the server name that linuxcncrsh will use to identify itself during handshaking with a new client. Defaults to EMCNETSVR if omitted.

**-w,--connectpw PASSWORD**

Specify the connection password to use during handshaking with a new client. Note that the password is sent in the clear, so it can be read by anyone who can read packets on the network between the server and the client. Defaults to EMC if omitted.

**-e,--enablepw PASSWORD**

Specify the password required to enable LinuxCNC via linuxcncrsh. Note that the password is sent in the clear, so it can be read by anyone who can read packets on the network between the server and the client. Defaults to EMCTOO if omitted.

**-s,--sessions MAX\_SESSIONS**

Specify the maximum number of simultaneous connections. Defaults to -1 (no limit) if not specified.

In addition to the options listed above, linuxcncrsh accepts an optional special LINUXCNC\_OPTION at the end:

**-ini LINUXCNC\_INI\_FILE**

LinuxCNC .ini file to use. The -ini option **must** be preceeded by two dashes: "--". Defaults to emc.ini if omitted.

**Starting linuxcncrsh**

To use linuxcncrsh instead of a normal LinuxCNC GUI like axis or touch, specify it in your .ini file like this:

**[DISPLAY]**

**DISPLAY=linuxcncrsh**

To use linuxcncrsh in addition to a normal GUI, you can either start it at the end of your .hal file, or run it by hand in a terminal window.

To start it from hal, add a line like this to the end of your .hal file:

**loadusr linuxcncrsh** [**OPTIONS**] [-- **LINUXCNC\_OPTIONS**]

To start it from the terminal, run linuxcncrsh manually like this:

**linuxcncrsh** [**OPTIONS**] [-- **LINUXCNC\_OPTIONS**]

**Connecting**

Once LinuxCNC is up and linuxcncrsh is running, you can connect to it using **telnet** or **nc** or similar:

**telnet HOST PORT**

HOST is the hostname or IP address of the computer running linuxcncrsh, and PORT is

the port it's listening on (5007 if you did not give linuxcncrsh the --port option).

## Network protocol

linuxcncrsh accepts TCP connections on the port specified by the --port option, or 5007 if not specified.

The client sends requests, and the linuxcncrsh server returns replies. Requests consist of a command word followed by optional command-specific parameters. Requests and most request parameters are case insensitive. The exceptions are passwords, file paths and text strings.

Requests to linuxcncrsh are terminated with line endings, any combination of one or more '\r' and '\n' characters. Replies from linuxcncrsh are terminated with the sequence '\r\n'.

The supported commands are as follows:

### hello <password> <client> <version>

<password> must match linuxcncrsh's connect password, or "EMC" if no --connectpw was supplied. The three arguments may not contain whitespace. If a valid password was entered the server will respond with:

*HELLO ACK <ServerName> <ServerVersion>*

If an invalid password or any other syntax error occurs then the server responds with:

*HELLO NAK*

### get <subcommand> [<parameters>]

The get command takes one of the LinuxCNC sub-commands (described in the section **LinuxCNC Subcommands**, below) and zero or more additional subcommand-specific parameters.

### set <subcommand> <parameters>

The set command takes one of the LinuxCNC sub-commands (described in the section **LinuxCNC Subcommands**, below) and one or more additional parameters.

### quit

The quit command disconnects the associated socket connection.

### shutdown

The shutdown command tells LinuxCNC to shutdown and disconnect the session. This command may only be issued if the Hello has been successfully negotiated and the connection has control of the CNC (see **enable** subcommand in the **LinuxCNC Subcommands** section, below).

### help

The help command will return help information in text format over the connection. If no parameters are specified, it will itemize the available commands. If a command is specified, it will provide usage information for the specified command. Help will respond regardless of whether a "Hello" has been successfully negotiated.

## LinuxCNC Subcommands

Subcommands for **get** and **set** are:

### echo {on|off}

With get, any on/off parameter is ignored and the current echo state is returned. With set, sets the echo state as specified. Echo defaults to on when the connection is first established. When echo is on, all commands will be echoed upon receipt. This state is local to each connection.

### verbose {on|off}

With get, any on/off parameter is ignored and the current verbose state is returned. With set, sets the verbose state as specified. When verbose mode is on, all set commands return positive acknowledgement in the form SET <COMMAND> ACK, and text error messages will be issued (FIXME: I don't know what this means). The verbose state is local to each connection, and starts out OFF on new connections.

### enable {<passwd>|off}

The session's enable state indicates whether the current connection is enabled to perform control functions. With get, any parameter is ignored, and the current enable state is returned. With set

and a valid password matching linuxncrsh's --enablepw (EMCTOO if not specified), the current connection is enabled for control functions. "OFF" may not be used as a password and disables control functions for this connection.

#### **config [TBD]**

Unused, ignore for now.

#### **comm\_mode {ascii|binary}**

With get, any parameter is ignored and the current communications mode is returned. With set, will set the communications mode to the specified mode. The ascii mode is the text request/reply mode, the binary protocol is not currently designed or implemented.

#### **comm\_prot <version>**

With get, any parameter is ignored and the current protocol version used by the server is returned. With set, sets the server to use the specified protocol version, provided it is lower than or equal to the highest version number supported by the server implementation.

#### **inifile**

Not currently implemented! With get, returns the string "emc.ini". Should return the full path and file name of the current configuration inifile. Setting this does nothing.

#### **plat**

With get, returns the string "Linux".

#### **ini <var> <section>**

Not currently implemented, do not use! Should return the string value of <var> in section <section> of the ini file.

#### **debug <value>**

With get, any parameter is ignored and the current integer value of EMC\_DEBUG is returned. Note that the value of EMC\_DEBUG returned is the from the UI's ini file, which may be different than emc's ini file. With set, sends a command to the EMC to set the new debug level, and sets the EMC\_DEBUG global here to the same value. This will make the two values the same, since they really ought to be the same.

#### **set\_wait {received|done}**

The set\_wait setting controls the wait after receiving a command. It can be "received" (after the command was sent and received) or "done" (after the command was done). With get, any parameter is ignored and the current set\_wait setting is returned. With set, set the set\_wait setting to the specified value.

#### **wait {received|done}**

With set, force a wait for the previous command to be received, or done.

#### **set\_timeout <timeout>**

With set, set the timeout for commands to return to <timeout> seconds. Timeout is a real number. If it's <= 0.0, it means wait forever. Default is 0.0, wait forever.

#### **update {none|auto}**

The update mode controls whether to return fresh or stale values for "get" requests. When the update mode is "none" it returns stale values, when it's "auto" it returns fresh values. Defaults to "auto" for new connections. Set this to "none" if you like to be confused.

#### **error**

With get, returns the current error string, or "ok" if no error.

#### **operator\_display**

With get, returns the current operator display string, or "ok" if none.

#### **operator\_text**

With get, returns the current operator text string, or "ok" if none.

#### **time**

With get, returns the time, in seconds, from the start of the epoch. This starting time depends on the platform.

**estop {on|off}**

With get, ignores any parameters and returns the current estop setting as "on" or "off". With set, sets the estop as specified. Estop "on" means the machine is in the estop state and won't run.

**machine {on|off}**

With get, ignores any parameters and returns the current machine power setting as "on" or "off". With set, sets the machine on or off as specified.

**mode {manual|auto|mdi}**

With get, ignores any parameters and returns the current machine mode. With set, sets the machine mode as specified.

**mist {on|off}**

With get, ignores any parameters and returns the current mist coolant setting. With set, sets the mist setting as specified.

**flood {on|off}**

With get, ignores any parameters and returns the current flood coolant setting. With set, sets the flood setting as specified.

**lube {on|off}**

With get, ignores any parameters and returns the current lube pump setting. With set, sets the lube pump setting as specified.

**lube\_level**

With get, returns the lubricant level sensor reading as "ok" or "low". With set, mocks you for wishful thinking.

**spindle {forward|reverse|increase|decrease|constant|off}**

With get, any parameter is ignored and the current spindle state is returned as "forward", "reverse", "increase", "decrease", or "off". With set, sets the spindle as specified. Note that "increase" and "decrease" will cause a speed change in the corresponding direction until a "constant" command is sent.

**brake {on|off}**

With get, any parameter is ignored and the current brake setting is returned. With set, the brake is set as specified.

**tool**

With get, returns the id of the currently loaded tool.

**tool\_offset**

With get, returns the currently applied tool length offset.

**load\_tool\_table <file>**

With set, loads the tool table specified by <file>.

**home {0|1|2|...}**

With set, homes the indicated axis.

**jog\_stop {0|1|2|...}**

With set, stop any in-progress jog on the specified axis.

**jog {0|1|2|...} <speed>**

With set, jog the specified axis at <speed>; sign of speed is direction.

**jog\_incr {0|1|2|...} <speed> <incr>**

With set, jog the indicated axis by increment <incr> at the <speed>; sign of speed is direction.

**feed\_override <percent>**

With get, any parameter is ignored and the current feed override is returned (as a percentage of commanded feed). With set, sets the feed override as specified.

**spindle\_override <percent>**

With get, any parameter is ignored and the current spindle override is returned (as a percentage of commanded speed). With set, sets the spindle override as specified.

**abs\_cmd\_pos [{0|1|...}]**

With get, returns the specified axis' commanded position in absolute coordinates. If no axis is specified, returns all axes' commanded absolute position.

**abs\_act\_pos [{0|1|...}]**

With get, returns the specified axis' actual position in absolute coordinates. If no axis is specified, returns all axes' actual absolute position.

**rel\_cmd\_pos [{0|1|...}]**

With get, returns the specified axis' commanded position in relative coordinates, including tool length offset. If no axis is specified, returns all axes' commanded relative position.

**rel\_act\_pos [{0|1|...}]**

With get, returns the specified axis' actual position in relative coordinates, including tool length offset. If no axis is specified, returns all axes' actual relative position.

**joint\_pos [{0|1|...}]**

With get, returns the specified joint's actual position in absolute coordinates, excluding tool length offset. If no joint is specified, returns all joints' actual absolute position.

**pos\_offset [{X|Y|Z|R|P|W}]**

With get, returns the position offset associated with the world coordinate provided.

**joint\_limit [{0|1|...}]**

With get, returns limit status of the specified joint as "ok", "minsoft", "minhard", "maxsoft", or "maxhard". If no joint number is specified, returns the limit status of all joints.

**joint\_fault [{0|1|...}]**

With get, returns the fault status of the specified joint as "ok" or "fault". If no joint number is specified, returns the fault status of all joints.

**joint\_homed [{0|1|...}]**

With get, returns the homed status of the specified joint as "homed" or "not". If no joint number is specified, returns the homed status of all joints.

**mdi <string>**

With set, sends <string> as an MDI command.

**task\_plan\_init**

With set, initializes the program interpreter.

**open <filename>**

With set, opens the named file. The <filename> is opened by linuxcnc, so it should either be an absolute path or a relative path starting in the linuxcnc working directory (the directory of the active .ini file). Note that linuxcnc can only have one file open at a time, and it's up to the UI (linuxcncrsh or similar) to close any open file before opening a new file. linuxcncrsh currently does not support closing files, which rather limits the utility of this command.

**run [<StartLine>]**

With set, runs the opened program. If no StartLine is specified, runs from the beginning. If a StartLine is specified, start line, runs from that line. A start line of -1 runs in verify mode.

**pause**

With set, pause program execution.

**resume**

With set, resume program execution.

**abort**

With set, abort program or MDI execution.

**step**

With set, step the program one line.

**program**

With get, returns the name of the currently opened program, or "none".

**program\_line**

With get, returns the currently executing line of the program.

**program\_status**

With get, returns "idle", "running", or "paused".

**program\_codes**

With get, returns the string for the currently active program codes.

**joint\_type [<joint>]**

With get, returns "linear", "angular", or "custom" for the type of the specified joint (or for all joints if none is specified).

**joint\_units [<joint>]**

With get, returns "inch", "mm", "cm", or "deg", "rad", "grad", or "custom", for the corresponding native units of the specified joint (or for all joints if none is specified). The type of the axis (linear or angular) is used to resolve which type of units are returned. The units are obtained heuristically, based on the EMC\_AXIS\_STAT::units numerical value of user units per mm or deg. For linear joints, something close to 0.03937 is deemed "inch", 1.000 is "mm", 0.1 is "cm", otherwise it's "custom". For angular joints, something close to 1.000 is deemed "deg",  $\pi/180$  is "rad", 100/90 is "grad", otherwise it's "custom".

**program\_units**

Synonym for program\_linear\_units.

**program\_linear\_units**

With get, returns "inch", "mm", "cm", or "none", for the corresponding linear units that are active in the program interpreter.

**program\_angular\_units**

With get, returns "deg", "rad", "grad", or "none" for the corresponding angular units that are active in the program interpreter.

**user\_linear\_units**

With get, returns "inch", "mm", "cm", or "custom", for the corresponding native user linear units of the LinuxCNC trajectory level. This is obtained heuristically, based on the EMC\_TRAJ\_STAT::linearUnits numerical value of user units per mm. Something close to 0.03937 is deemed "inch", 1.000 is "mm", 0.1 is "cm", otherwise it's "custom".

**user\_angular\_units**

Returns "deg", "rad", "grad", or "custom" for the corresponding native user angular units of the LinuxCNC trajectory level. Like with linear units, this is obtained heuristically.

**display\_linear\_units**

With get, returns "inch", "mm", "cm", or "custom", for the linear units that are active in the display. This is effectively the value of linearUnitConversion.

**display\_angular\_units**

With get, returns "deg", "rad", "grad", or "custom", for the angular units that are active in the display. This is effectively the value of angularUnitConversion.

**linear\_unit\_conversion {inch|mm|cm|auto}**

With get, any parameter is ignored and the active unit conversion is returned. With set, sets the unit to be displayed. If it's "auto", the units to be displayed match the program units.

**angular\_unit\_conversion {deg|rad|grad|auto}**

With get, any parameter is ignored and the active unit conversion is returned. With set, sets the units to be displayed. If it's "auto", the units to be displayed match the program units.

**probe\_clear**

With set, clear the probe tripped flag.

**probe\_tripped**

With get, return the probe state - has the probe tripped since the last clear?

**probe\_value**

With get, return the current value of the probe signal.

**probe**

With set, move toward a certain location. If the probe is tripped on the way stop motion, record the position and raise the probe tripped flag.

**teleop\_enable [on|off]**

With get, any parameter is ignored and the current teleop mode is returned. With set, sets the teleop mode as specified.

**kinematics\_type**

With get, returns the type of kinematics functions used (identity=1, serial=2, parallel=3, custom=4).

**override\_limits {on|off}**

With get, any parameter is ignored and the override\_limits setting is returned. With set, the override\_limits parameter is set as specified. If override\_limits is on, disables end of travel hardware limits to allow jogging off of a limit. If parameters is off, then hardware limits are enabled.

**optional\_stop {0|1}**

With get, any parameter is ignored and the current "optional stop on M1" setting is returned. With set, the setting is set as specified.

**Example Session**

This section shows an example session. Bold items are typed by you, non-bold is machine output.

The user connects to linuxcncrsh, handshakes with the server (hello), enables machine commanding from this session (set enable), brings the machine out of estop (set estop off) and turns it on (set machine on), homes all the axes, switches the machine to mdi mode, sends an MDI g-code command, then disconnects and shuts down LinuxCNC.

**> telnet localhost 5007**

Trying 127.0.0.1...

Connected to 127.0.0.1

Escape character is '^]'.

**hello EMC user-typing-at-telnet 1.0**

HELLO ACK EMCNETSVR 1.1

**set enable EMCTOO**

set enable EMCTOO

**set mode manual**

set mode manual

**set estop off**

set estop off

**set machine on**

set machine on

**set home 0**

set home 0

**set home 1**

set home 1

**set home 2**

set home 2

**set mode mdi**

set mode mdi

**set mdi g0x1**

set mdi g0x1

**shutdown**

shutdown

Connection closed by foreign host.

**NAME**

milltask – Userspace task controller for LinuxCNC

**DESCRIPTION**

milltask is an internal process of LinuxCNC. It is generally not invoked directly. It creates the pins shown as owned by the "inihal" component, which allow runtime modification of certain values from the inifile.

**PINS****Per-axis pins****ini.#.backlash**

Allows adjustment of [AXIS\_#]BACKLASH

**ini.#.max\_acceleration**

Allows adjustment of [AXIS\_#]MAX\_ACCELERATION

**ini.#.max\_velocity**

Allows adjustment of [AXIS\_#]MAX\_VELOCITY

**ini.#.max\_limit**

Allows adjustment of [AXIS\_#]MAX\_LIMIT

**ini.#.min\_limit**

Allows adjustment of [AXIS\_#]MIN\_LIMIT

**ini.#.ferror**

Allows adjustment of [AXIS\_#]FERROR

**ini.#.min\_ferror**

Allows adjustment of [AXIS\_#]MIN\_FERROR

**Global pins****ini.traj\_default\_acceleration**

Allows adjustment of [TRAJ]DEFAULT\_ACCELERATION

**ini.traj\_default\_velocity**

Allows adjustment of [TRAJ]DEFAULT\_VELOCITY

**ini.traj\_max\_acceleration**

Allows adjustment of [TRAJ]MAX\_ACCELERATION

**ini.traj\_max\_velocity**

Allows adjustment of [TRAJ]MAX\_VELOCITY

**Global pins (arc\_blend trajectory planner)****ini.traj\_arc\_blend\_enable**

Allows adjustment of [TRAJ]ARC\_BLEND\_ENABLE

**ini.traj\_arc\_blend\_fallback\_enable**

Allows adjustment of [TRAJ]ARC\_BLEND\_FALLBACK\_ENABLE

**ini.traj\_arc\_blend\_gap\_cycles**

Allows adjustment of [TRAJ]ARC\_OPTIMIZATION\_DEPTH

**ini.traj\_arc\_blend\_optimization\_depth**

Allows adjustment of [TRAJ]ARC\_BLEND\_GAP\_CYCLES

**ini.traj\_arc\_blend\_ramp\_freq**

Allows adjustment of [TRAJ]ARC\_BLEND\_RAMP\_FREQ

**NOTES**

These pins cannot be linked or set in a halfile specified by [HAL]HALFILE. They can be altered by independent halcmd programs or by GUIs that support [HAL]POSTGUI\_HALFILE. The inifile is not automatically updated with these values but can be updated using the calibration program (emccalib.tcl) when

using a **[HAL]POSTGUI\_HALFILE**.

## NAME

moveoff\_gui – a gui for the moveoff component

## SYNOPSIS

**moveoff\_gui** [--help | -- -h | -?]

**moveoff\_gui** [options]

## DESCRIPTION

Moveoff\_gui is a sample graphical user interface (GUI) for controlling a Hal moveoff component to implement Hal-only offsets. See the manpage (man moveoff) for **IMPORTANT limitations and warnings**.

## OPTIONS

**--help | -? | -- -h**

Show options and exit

**-mode onpause** | always

onpause: popup gui to control offsets when program paused

always: show gui to control offsets always

Default: **onpause**

**-axes** axisnames

Letters from set of {x y z a b c u v w}

Examples: -axes x, -axes xyz, -axes xz (no spaces)

Default: **xyz**

**-inc** incrementvalue

Specify one increment value per -inc (up to 4)

Defaults: **0.001 0.01 0.10 1.0**

**-size** integer

Overall gui size is based on font size, typically 8 - 20

Default: **14**

**-loc** center | +x+y

Initial location on screen

Examples: -loc center, -loc +20+100

Default: **center**

**-autoresume**

Resume program when move-enable deasserted

Default: notused

**-delay** delay secs

Delay for autoresume (allow time to restore spindle speed etc) Default: **5**

## OTHER OPTIONS

These options are available for special cases:

**-noentry**

Disables creation of entry widgets

Default: notused

**-no\_resume\_inhibit**

Disable use of resume-inhibit to controlling gui

Default: notused

**-no\_pause\_requirement**

Disable check for halui.program.is-paused

Default: notused

**-no\_cancel\_autoresume**

Useful for retracting offsets with simple external controls

Default: notused

**-no\_display**

Use when both external controls and and external displays are in use

Default: notused

**NOTES**

LinuxCNC must be running.

Halui must be loaded, typical ini file setting:

**[HAL]HALUI = halui.**

The moveoff component must be loaded with the name 'mv' as:

**loadrt moveoff names=mv personality=number\_of\_axes**

If the pin mv.motion-enable is **not** connected when moveoff\_gui is started, **controls will be provided** to enable offsets and set offset values. If the pin **is** connected, **only a display** of offsets is shown and control must be made by **external** Hal connections.

If a pin named \*.resume-inhibit exists and is not connected, it will be set while offsets are applied. This pin may be provided by the controlling linuxcnc gui in use. Use of the pin may be disabled with the option -no\_resume\_inhibit.

The -autoresume option uses halui.program.resume to automatically resume program execution when the move-enable pin is deactivated and all offsets are removed. The resume pin is not activated until an additional interval (-delay delay\_secs) elapses. This delay interval may be useful for restarting related equipment (a spindle motor for example) While timing the delay, a popup is offered to cancel the automatic program resumption.

**USAGE**

The ini file in the configuration directory must provide HALFILES to loadrt the moveoff component, connect its pins, and addf its read and write functions in the proper order. These steps can be done at runtime using an existing configuration ini file and specifying a system library HALFILE **hookup\_moveoff.tcl** as illustrated below:

**[HAL]**

HALUI = halui

HALFILE = user\_halfile\_1

etc ...

HALFILE = user\_halfile\_n

**HALFILE = LIB:hookup\_moveoff.tcl**

The **hookup\_moveoff.tcl** halfile will use ini file settings for the moveoff component control pins:

```
[OFFSET]
EPSILON =
WAYPOINT_SAMPLE_SECS =
WAYPOINT_THRESHOLD =
BACKTRACK_ENABLE =
```

The **hookup\_moveoff.tcl** will use ini file settings for the moveoff per-axis limits:

```
[AXIS_m]
OFFSET_MAX_VELOCITY =
OFFSET_MAX_ACCELERATION =
OFFSET_MAX_LIMIT =
OFFSET_MIN_LIMIT =
```

The moveoff\_gui program should be specified in the APPLICATIONS stanza of the ini file, for example:

```
[APPLICATIONS]
DELAY = delay_in_secs_to_allow_hal_connections
APP = moveoff_gui -option1 -option2 ...
```

## SEE ALSO

Simulation configurations that demonstrate the moveoff\_gui and the moveoff component are located in:

```
configs/sim/axis/moveoff    (axis-ui)
configs/sim/touchy/ngcgui   (touchy-ui)
```

man page for the moveoff component:**moveoff(9)**

**NAME**

pyvcp – Virtual Control Panel for LinuxCNC

**SYNOPSIS**

**pyvcp** [-g *WxH+X+Y*] [-c *component-name*] *myfile.xml*

**OPTIONS**

**-g** *WxH+X+Y*

This sets the initial geometry of the root window. Use 'WxH' for just size, '+X+Y' for just position, or 'WxH+X+Y' for both. Size / position use pixel units. Position is referenced from top left.

**-c** *component-name*

Use *component-name* as the HAL component name. If the component name is not specified, the basename of the xml file is used.

**SEE ALSO**

*Python Virtual Control Panel* in the LinuxCNC documentation for a description of the xml syntax, along with examples

**NAME**

shuttlepress – control HAL pins with the ShuttleXpress device made by Contour Design

**SYNOPSIS**

*loadusr shuttlepress [DEVICE ...]*

**DESCRIPTION**

shuttlepress is a userspace HAL component that interfaces Contour Design's ShuttleXpress device with LinuxCNC's HAL. The ShuttleXpress has five momentary buttons, a 10 counts/revolution jog wheel with detents, and a 15-position spring-loaded outer wheel that returns to center when released.

If it is started without command-line arguments, it will probe all /dev/hidraw\* device files for ShuttleXpress devices, and use all devices found. If it is started with command-line arguments, only will only probe the devices specified.

**UDEV**

The shuttlepress module needs read permission on the /dev/hidraw\* device files. This can be accomplished by adding a file **/etc/udev/rules.d/99-shuttlepress.rules**, with the following contents:

```
SUBSYSTEM=="hidraw", ATTRS{idVendor}=="0b33", ATTRS{idProduct}=="0020", MODE="0444"
```

**A warning about the Jog Wheel**

The ShuttleXpress device has an internal 8-bit counter for the current jog-wheel position. The shuttlepress driver can not know this value until the ShuttleXpress device sends its first event. When the first event comes into the driver, the driver uses the device's reported jog-wheel position to initialize counts to 0.

This means that if the first event is generated by a jog-wheel move, that first move will be lost.

Any user interaction with the ShuttleXpress device will generate an event, informing the driver of the jog-wheel position. So if you (for example) push one of the buttons at startup, the jog-wheel will work fine and notice the first click.

**Pins**

(bit out) *shuttlepress.0.button-0*

(bit out) *shuttlepress.0.button-0-not*

(bit out) *shuttlepress.0.button-1*

(bit out) *shuttlepress.0.button-1-not*

(bit out) *shuttlepress.0.button-2*

(bit out) *shuttlepress.0.button-2-not*

(bit out) *shuttlepress.0.button-3*

(bit out) *shuttlepress.0.button-3-not*

(bit out) *shuttlepress.0.button-4*

(bit out) *shuttlepress.0.button-4-not*

The five buttons around the outside, starting with the counter-clockwise-most one.

(s32 out) *shuttlepress.0.counts*

Accumulated counts from the jog wheel (the inner wheel).

(s32 out) *shuttlepress.0.spring-wheel-s32*

The current deflection of the spring-wheel (the outer wheel).  
It's 0 at rest, and ranges from -7 at the counter-clockwise  
extreme to +7 at the clockwise extreme.

(float out) *shuttlepress.0.spring-wheel-f*

The current deflection of the spring-wheel (the outer wheel).  
It's 0 at rest, -1 at the counter-clockwise extreme, and  
+1 at the clockwise extreme. (The ShuttleXpress device reports the  
spring-wheel position quantized from -7 to +7, so this pin reports  
only 15 discrete values in its range.)

**NAME**

**sim\_pin** – gui for displaying and setting multiple Hal inputs

**SYNOPSIS**

**sim\_pin name1** [name2 [name3 ...]]

**DESCRIPTION**

If the named input is a **numerical** type, the gui displays:

**Entry** Entry widget for new value

**Set** Button to set new value from Entry (or use <RETURN>)

**Reset** Button to reset to the value present on initiation

If the input is a **bit** type, the gui shows a single pushbutton that is controlled by the radio-button selectors:

**OnePulse** Pulse input to 1 for each pushbutton press

**ToggleValue** Toggle input for each pushbutton press

**1 WhilePressed** Set input to 1 while pushbutton pressed

**NOTE**

LinuxCNC must be running

A named item can specify a **pin**, **param**, or **signal**. The named item must be writable:

**pin** **IN** or **I/O** (and not connected to a signal)

**param** **RW**

**signal** **connected to a writable pin**

**USAGE**

**sim\_pin** can be used interactively from a shell command line or started automatically from a configuration ini file.

**EXAMPLE**

Example for ini file usage:

[APPLICATIONS]

DELAY = 5

APP = **sim\_pin** \

halui.machine.off \

ini.traj\_arc\_blend\_enable \

motion-command-handler-tmax

**NAME**

**vfdb\_vfd** - HAL userspace component for Delta VFD-B Variable Frequency Drives

**SYNOPSIS**

**vfdb\_vfd** [OPTIONS]

**DESCRIPTION**

This manual page explains the **vfdb\_vfd** component. This component reads and writes to the VFD-B device via a Modbus connection.

**vfdb\_vfd** is for use with LinuxCNC.

**QUICK START**

The VFD-B ships in a configuration that can not talk to this driver. The VFD-B must be reconfigured via the face plate by the integrator before it will work. This section gives a brief description of what changes need to be made, consult your Delta VFD-B manual for more details.

Switch the VFD-B to Modbus RTU frame format:

Switch parameter 09-04 from the factory default of 0 (Ascii framing) to 3, 4, or 5 (RTU framing). The setting you choose will determine several serial parameters in addition to the Modbus framing protocol.

Set the frequency control source to be Modbus, not the keypad:

Switch parameter 02-00 from factory default of 00 (keypad control) to 5 (control from RS-485).

Set the run/stop control source to be Modbus, not the keypad:

Switch parameter 02-01 from the factory default of 0 (control from keypad) to 3 (control from Modbus, with Stop enabled on the keypad).

**OPTIONS**

**-n --name <halname>**

set the HAL component name

**-d --debug**

Turn on debugging messages. Also toggled by sending a USR1 signal to the vfdb\_vfd process.

**-m --modbus-debug**

Turn on Modbus debugging messages. This will cause all Modbus messages to be printed in hex on the terminal. Also toggled by sending a USR2 signal to the vfdb\_vfd process.

**-I --ini <inifilename>**

take configuration from this ini file. Defaults to environment variable INI\_FILE\_NAME. Most vfdb\_vfd configuration comes from the ini file, not from command-line arguments.

**-S --section <section name>**

take configuration from this section in the ini file. Defaults to 'VFD-B'.

**-r --report-device**

report device propertiers on console at startup

**INI CONFIG VARIABLES****DEBUG**

Set to a non-zero value to enable general debug output from the VFD-B driver. Optional.

**MODBUS\_DEBUG**

Set to a non-zero value to enable modbus debug output from the VFD-B driver. Optional.

**DEVICE**

Serial port device file to use for Modbus communication with the VFD-B. Defaults to `"/dev/ttyS0"`.

**BAUD** Modbus baud rate. Defaults to 19200.

**BITS** Modbus data bits. Defaults to 8.

**PARITY**

Modbus parity. Defaults to Even. Accepts 'Even', 'Odd', or 'None'.

**STOPBITS**

Modbus stop bits. Defaults to 1.

**TARGET**

Modbus target number of the VFD-B to speak to. Defaults to 1.

**POLLCYCLES**

Only read the less important variables from the VFD-B once in this many poll cycles. Defaults to 10.

**RECONNECT\_DELAY**

If the connection to the VFD-B is broken, wait this many seconds before reconnecting. Defaults to 1.

**MOTOR\_HZ, MOTOR\_RPM**

The frequency of the motor (in Hz) and the corresponding speed of the motor (in RPM). This information is provided by the motor manufacturer, and is generally printed on the motor's name plate.

**PINS****<name>.at-speed (bit, out)**

True when drive is at commanded speed (see *speed-tolerance* below)

**<name>.enable (bit, in)**

Enable the VFD. If False, all operating parameters are still read but control is released and panel control is enabled (subject to VFD setup).

**<name>.frequency-command (float, out)**

Current target frequency in HZ as set through speed-command (which is in RPM), from the VFD.

**<name>.frequency-out (float, out)**

Current output frequency of the VFD.

**<name>.inverter-load-percentage (float, out)**

Current load report from VFD.

**<name>.is-e-stopped (bit, out)**

The VFD is in emergency stop status (blinking "E" on panel).

**<name>.is-stopped (bit, out)**

True when the VFD reports 0 Hz output.

**<name>.jog-mode (bit, in)**

1 for ON and 0 for OFF, enables the VFD-B 'jog mode'. Speed control is disabled. This might be useful for spindle orientation.

**<name>.max-rpm (float, out)**

Actual RPM limit based on maximum frequency the VFD may generate, and the motors nameplate values. For instance, if *nameplate-HZ* is 50, and *nameplate-RPM* is 1410, but the VFD may generate up to 80Hz, then *max-rpm* would read as 2256 ( $80 \times 1410 / 50$ ). The frequency limit is read from the VFD at startup. To increase the upper frequency limit, the UL and FH parameters must be changed on the panel. See the VFD-B manual for instructions how to set the maximum frequency.

- <name>.modbus-ok (bit, out)**  
True when the Modbus session is successfully established and the last 10 transactions returned without error.
- <name>.motor-RPM (float, out)**  
Estimated current RPM value, from the VFD.
- <name>.motor-RPS (float, out)**  
Estimated current RPS value, from the VFD.
- <name>.output-voltage (float, out)**  
From the VFD.
- <name>.output-current (float, out)**  
From the VFD.
- <name>.speed-command (float, in)**  
Speed sent to VFD in RPM. It is an error to send a speed faster than the Motor Max RPM as set in the VFD.
- <name>.spindle-on (bit, in)**  
1 for ON and 0 for OFF sent to VFD, only on when running.
- <name>.max-speed (bit, in)**  
Ignore the loop-time parameter and run Modbus at maximum speed, at the expense of higher CPU usage. Suggested use during spindle positioning.
- <name>.status (s32, out)**  
Drive Status of the VFD (see the VFD manual). A bitmap.
- <name>.error-count (s32, out)**  
Total number of transactions returning a Modbus error.
- <name>.error-code (s32, out)**  
Most recent Error Code from VFD.
- <name>.frequency-limit (float, out)**  
Upper limit read from VFD setup.

## PARAMETERS

- <name>.loop-time (float, RW)**  
How often the Modbus is polled (default interval 0.1 seconds).
- <name>.nameplate-HZ (float, RW)**  
Nameplate Hz of motor (default 50). Used to calculate target frequency (together with *nameplate-RPM* ) for a target RPM value as given by speed-command.
- <name>.nameplate-RPM (float, RW)**  
Nameplate RPM of motor (default 1410)
- <name>.rpm-limit (float, RW)**  
Do-not-exceed soft limit for motor RPM (defaults to *nameplate-RPM* ).
- <name>.tolerance (float, RW)**  
Speed tolerance (default 0.01) for determining whether spindle is at speed (0.01 meaning: output frequency is within 1% of target frequency).

## USAGE

The vfdb\_vfd driver takes precedence over panel control while it is enabled (see *.enable* pin), effectively disabling the panel. Clearing the *.enable* pin re-enables the panel. Pins and parameters can still be set, but will not be written to the VFD until the *.enable* pin is set. Operating parameters are still read while bus control is disabled.

Exiting the vfdb\_vfd driver in a controlled way will release the VFD from the bus and restore panel control.

See the LinuxCNC Integrators Manual for more information. For a detailed register description of the Delta VFD-B, see the VFD manual.

**AUTHOR**

Yishin Li; based on vfd11\_vfd by Michael Haberler.

**LICENSE**

GPL

**NAME**

**vfs11\_vfd** - HAL userspace component for Toshiba-Schneider VF-S11 Variable Frequency Drives

**SYNOPSIS**

**vfs11\_vfd** [OPTIONS]

**DESCRIPTION**

This manual page explains the **vfs11\_vfd** component. This component reads and writes to the vfs11 via a Modbus connection.

**vfs11\_vfd** is for use with LinuxCNC.

**OPTIONS**

**-n --name <halname>**

set the HAL component name

**-d --debug**

Turn on debugging messages. Also toggled by sending a USR1 signal to the vfs11\_vfd process.

**-m --modbus-debug**

Turn on Modbus debugging messages. This will cause all Modbus messages to be printed in hex on the terminal. Also toggled by sending a USR2 signal to the vfs11\_vfd process.

**-I --ini <inifilename>**

take configuration from this ini file. Defaults to environment variable INI\_FILE\_NAME.

**-S --section <section name>**

take configuration from this section in the ini file. Defaults to 'VFS11'.

**-r --report-device**

report device properties on console at startup

**PINS**

**<name>.acceleration-pattern (bit, in)**

when true, set acceleration and deceleration times as defined in registers F500 and F501 respectively. Used in PID loops to choose shorter ramp times to avoid oscillation.

**<name>.alarm-code (s32, out)**

non-zero if drive is in alarmed state. Bitmap describing alarm information (see register FC91 description). Use *err-reset* (see below) to clear the alarm.

**<name>.at-speed (bit, out)**

when drive is at commanded speed (see *speed-tolerance* below)

**<name>.current-load-percentage (float, out)**

reported from the VFD

**<name>.dc-brake (bit, in)**

engage the DC brake. Also turns off spindle-on.

**<name>.enable (bit, in)**

enable the VFD. If false, all operating parameters are still read but control is released and panel control is enabled (subject to VFD setup).

**<name>.err-reset (bit, in)**

reset errors (alarms a.k.a Trip and e-stop status). Resetting the VFD may cause a 2-second delay until it's rebooted and Modbus is up again.

**<name>.estop (bit, in)**

put the VFD into emergency-stopped status. No operation possible until cleared with *err-reset* or powercycling.

- <name>.frequency-command (float, out)**  
current target frequency in HZ as set through speed-command (which is in RPM), from the VFD
- <name>.frequency-out (float, out)**  
current output frequency of the VFD
- <name>.inverter-load-percentage (float, out)**  
current load report from VFD
- <name>.is-e-stopped (bit, out)**  
the VFD is in emergency stop status (blinking "E" on panel). Use *err-reset* to reboot the VFD and clear the e-stop status.
- <name>.is-stopped (bit, out)**  
true when the VFD reports 0 Hz output
- <name>.jog-mode (bit, in)**  
1 for ON and 0 for OFF, enables the VF-S11 'jog mode'. Speed control is disabled, and the output frequency is determined by register F262 (preset to 5Hz). This might be useful for spindle orientation.
- <name>.max-rpm (float, R)**  
actual RPM limit based on maximum frequency the VFD may generate, and the motors nameplate values. For instance, if *nameplate-HZ* is 50, and *nameplate-RPM* is 1410, but the VFD may generate up to 80Hz, then *max-rpm* would read as 2256 ( $80 \times 1410 / 50$ ). The frequency limit is read from the VFD at startup. To increase the upper frequency limit, the UL and FH parameters must be changed on the panel. See the VF-S11 manual for instructions how to set the maximum frequency.
- <name>.modbus-ok (bit, out)**  
true when the Modbus session is successfully established and the last 10 transactions returned without error.
- <name>.motor-RPM (float, out)**  
estimated current RPM value, from the VFD
- <name>.output-current-percentage (float, out)**  
from the VFD
- <name>.output-voltage-percentage (float, out)**  
from the VFD
- <name>.output-voltage (float, out)**  
from the VFD
- <name>.speed-command (float, in)**  
speed sent to VFD in RPM. It is an error to send a speed faster than the Motor Max RPM as set in the VFD
- <name>.spindle-fwd (bit, in)**  
1 for FWD and 0 for REV, sent to VFD
- <name>.spindle-on (bit, in)**  
1 for ON and 0 for OFF sent to VFD, only on when running
- <name>.spindle-rev (bit, in)**  
1 for ON and 0 for OFF, only on when running
- <name>.max-speed (bit, in)**  
ignore the loop-time parameter and run Modbus at maximum speed, at the expense of higher CPU usage. Suggested use during spindle positioning.

**<name>.status (s32, out)**

Drive Status of the VFD (see the TOSVERT VF-S11 Communications Function Instruction Manual, register FD01). A bitmap.

**<name>.trip-code (s32, out)**

trip code if VF-S11 is in tripped state.

**<name>.error-count (s32, RW)**

total number of transactions returning a Modbus error

**PARAMETERS****<name>.frequency-limit (float, RO)**

upper limit read from VFD setup.

**<name>.loop-time (float, RW)**

how often the Modbus is polled (default interval 0.1 seconds)

**<name>.nameplate-HZ (float, RW)**

Nameplate Hz of motor (default 50). Used to calculate target frequency (together with *nameplate-RPM* ) for a target RPM value as given by speed-command.

**<name>.nameplate-RPM (float, RW)**

Nameplate RPM of motor (default 1410)

**<name>.rpm-limit (float, RW)**

do-not-exceed soft limit for motor RPM (defaults to *nameplate-RPM* ).

**<name>.tolerance (float, RW)**

speed tolerance (default 0.01) for determining whether spindle is at speed (0.01 meaning: output frequency is within 1% of target frequency)

**USAGE**

The vfs11\_vfd driver takes precedence over panel control while it is enabled (see *.enable* pin), effectively disabling the panel. Clearing the *.enable* pin re-enables the panel. Pins and parameters can still be set, but will not be written to the VFD until the *.enable* pin is set. Operating parameters are still read while bus control is disabled.

Exiting the vfs11\_vfd driver in a controlled will release the VFD from the bus and restore panel control.

See the LinuxCNC Integrators Manual for more information. For a detailed register description of the Toshiba VFD's, see the "TOSVERT VF-S11 Communications Function Instruction Manual" (Toshiba document number E6581222) and the "TOSVERT VF-S11 Instruction manual" (Toshiba document number E6581158).

**AUTHOR**

Michael Haberler; based on gs2\_vfd by Steve Padnos and John Thornton.

**LICENSE**

GPL

**NAME**

xhc-hb04 – User-space HAL component for the xhc-hb04 pendant.

**DESCRIPTION**

The xhc-hb04 component supports a common USB pendant that provides a number of pushbuttons, a manual pulse generator (mpg or jog wheel), and a selector switch for the wheel.

There are at least two hardware versions -- one with 16 buttons and a more common one with 18 buttons. The information herein is based on the 18 button device with a USB Vendor:Product code of 10CE:EB70.

In addition to buttons, the pendant provides an LCD display for the current stepsize multiplier (from a set of available integer values), position (absolute and relative, labeled MC and WC respectively), feedrate (override percent and value in units per minute), and spindle speed (override percent and value in revolutions per minute (RPM)). The display is managed by a rotary switch that selects one of four axes for wheel positioning, feed override, spindle override, or OFF.

The pendant display, its rotary selector switch, and the component pin names use designators x,y,z,a. While this arrangement presumes a machine configured as XYZA, the pins can be assigned independently as required in a HAL configuration.

**UDEV**

The xhc-hb04 executable needs permission for reading the pendant's USB device. Debian package installs (debs) handle this automatically but Run-In-Place (RIP) builds may need a udev rules file. This file should be created (using sudo and a text editor) as:

**/etc/udev/rules.d/99-xhc-hb04.rules** with the single line:

```
ATTR{idProduct}=="eb70", ATTR{idVendor}=="10ce", MODE="0666", OWNER="root", GROUP="plugdev"
```

**Standalone Usage**

The xhc-hb04 program can be run from the command line without LinuxCNC to test a pendant in a simulation mode. This standalone mode is used to identify the button codes produced for each button press and to verify proper counting of the jog wheel. The identified button codes can be used to create a **button-cfg-file**. When a **button-cfg-file** exists, pendant operation can be verified using the -I option to specify the file.

Usage:

```
$ xhc-hb04 [options]
```

**Options**

- h** list command line options and exit
- I button-cfg-file** (see below for file format)
- H** run in real-time HAL mode (simulation mode is default)
- x** wait for pendant detection before creating HAL pins.
- s n** n is one of the following stepsize sequences
  - 1: 1,10,100,1000 (default)
  - 2: 1,5,10,20
  - 3: 1,10,100
  - 4: 1,5,10,20,50,100
  - 5: 1,10,50,100,1000
 The stepsize selected is always multiplied by 0.001

**button-cfg-file format**

Standard configuration files are provided in the distribution for known button configurations:

`/usr/share/linuxcnc/hallib/xhc-hb04-layout1.cfg`

`/usr/share/linuxcnc/hallib/xhc-hb04-layout2.cfg`

or for a RIP build:

`rip_base_dir/lib/hallib/xhc-hb04-layout1.cfg`

`rip_base_dir/lib/hallib/xhc-hb04-layout2.cfg`

layout1 describes the 16 button pendant, layout2 describes the more common 18 button pendant.

The button configuration file follows the same format as ini files but should use a file suffix of .cfg.

File format:

`[XHC-HB04]`

`BUTTON=X1:button-thename1`

`BUTTON=X2:button-thename2`

`BUTTON=X3:button-thename3`

etc.

XN is the code reported for a button press and button-thenameN is the name to be assigned to the pin created for the button.

**Hal Usage**

Use the -H option to specify HAL mode and other options as required:

`loadusr -W xhc-hb04 -H [Options]`

Example: `loadusr -W xhc-hb04 -H -I path_to_cfg_file -s 2`

**Input Pins (Control)**

(bit in) `xhc-hb04.stepsize-up` A 1 pulse on this pin changes the stepsize to the next higher stepsize in the stepsize sequence specified in the xhc-hb04 (loadusr) command.

(bit in) `xhc-hb04.stepsize-down` A 1 pulse on this pin changes the stepsize to the next lower stepsize in the stepsize sequence specified in the xhc-hb04 (loadusr) command.

**Input Pins (to the pendant LCD display)**

(float in) `xhc-hb04.[xyza].pos-absolute` Absolute position display.  
(typically connect to: halui.axis.N.pos-feedback). The LCD display for pos-absolute is fixed format with a sign, 4 number digits and 3 fraction digits (+XXXX.XXX), require: -9999.999 <= value <= 9999.999.

(float in) `xhc-hb04.[xyza].pos-relative` Relative position display.  
(typically connect to: halui.axis.N.pos-relative). The LCD display for pos-relative is fixed format with a sign, 4 number digits and 3 fraction digits (+XXXX.XXX), require: -9999.999 <= value <= 9999.999.

(float in) `xhc-hb04.feed-override` Feed-override value.

The float value is converted to a 16 bit integer and multiplied by 100 in order to display as percent, require: 0 <= pinvalue <= 655 (typically connect to: halui.feed-override.value)

- (float in) *xhc-hb04.feed-value* Current Feed-value (units/sec).  
The float value is converted to a 16 bit integer and multiplied by 60 in order to display as units-per-minute, require:  $0 \leq \text{pinvalue} \leq 1092$  (65520 units-per-minute) (typically connect to: motion.current-vel)
- (float in) *xhc-hb04.spindle-override* Spindle-override value.  
The float value is converted to a 16 bit integer and multiplied by 100 in order to display as percent, require:  $0 \leq \text{pinvalue} \leq 655$  (typically connect to: halui.spindle-override.value)
- (float in) *xhc-hb04.spindle-rps* Spindle speed in rps.  
(revolutions per second). The float value is converted to a 16 bit integer and multiplied by 60 in order to display as RPMs, require:  $0 \leq \text{pinvalue} \leq 1092$  (65520 RPM) (typically connect to: motion.spindle-speed-out-rps-abs)
- (bit in) *xhc-hb04.inch-icon* Use inch icon (default is mm)

### Output Pins (Status)

- (bit out) *xhc-hb04.sleeping* True when the driver receives a pendant inactive (sleeping) message.
- (bit out) *xhc-hb04.jog.enable-off* True when the pendant rotary selector switch is in the OFF position or when the pendant is sleeping.
- (bit out) *xhc-hb04.enable-[xyza]* True when the pendant rotary selector switch is in the [xyza] position and not sleeping.
- (bit out) *xhc-hb04.enable-spindle-override* True when the pendant rotary selector switch is in the Spindle position and not sleeping. (typically connect to: halui.spindle-override-count-enable)
- (bit out) *xhc-hb04.enable-feed-override* True when the pendant rotary selector switch is in the Feed position and not sleeping. (typically connect to: halui.feed-override-count-enable)
- (bit out) *xhc-hb04.connected* True when connection to the pendant is established over the USB interface.
- (bit out) *xhc-hb04.require\_pendant* True if driver started with the -x option.
- (s32 out) *xhc-hb04.stepsize* Current stepsize in the stepsize sequence as controlled by the stepsize-up and/or stepsize-down pins.

### Output Pins (for jogging using axis.N.jog-counts)

- (s32 out) *xhc-hb04.jog.counts* Number of counts of the wheel since start-up (50 counts per wheel revolution). (typically connect to axis.N.jog-counts (lowpass filtering may be helpful))
- (s32 out) *xhc-hb04.jog.counts-neg* The value of the *xhc-hb04.jog.counts* multiplied by -1.
- (float out) *xhc-hb04.jog.scale* Value is the current stepsize multiplied by 0.001. (typically connect to axis.N.jog-scale)

### Experimental: Pins for halui plus/minus jogging

These pins provide some support for non-trivkins, world mode jogging.

- (float in) *xhc-hb04.jog.max-velocity* Connect to halui.max-velocity.value

(float out) *xhc-hb04.jog.velocity* Connect to halui.jog-speed  
 (bit out) *xhc-hb04.jog.plus-[xyza]* Connect to halui.jog.N.plus  
 (bit out) *xhc-hb04.jog.minus-[xyza]* Connect to halui.jog.N.minus  
 (float out) *xhc-hb04.jog.increment* Debug pin -- abs(delta\_pos)

### Button output pins (for the 18 button, layout2 pendant)

The output bit type pins are TRUE when the button is pressed.

#### ROW 1

(bit out) *xhc-hb04.button-reset*  
 (bit out) *xhc-hb04.button-stop*

#### ROW 2

(bit out) *xhc-hb04.button-goto-zero*  
 (bit out) *xhc-hb04.button-rewind*  
 (bit out) *xhc-hb04.button-start-pause*  
 (bit out) *xhc-hb04.button-probe-z*

#### ROW 3

(bit out) *xhc-hb04.button-spindle*  
 (bit out) *xhc-hb04.button-half*  
 (bit out) *xhc-hb04.button-zero*  
 (bit out) *xhc-hb04.button-safe-z*

#### ROW 4

(bit out) *xhc-hb04.button-home*  
 (bit out) *xhc-hb04.button-macro-1*  
 (bit out) *xhc-hb04.button-macro-2*  
 (bit out) *xhc-hb04.button-macro-3*

#### ROW 5

(bit out) *xhc-hb04.button-step*  
 (bit out) *xhc-hb04.button-mode*  
 (bit out) *xhc-hb04.button-macro-6*  
 (bit out) *xhc-hb04.button-macro-7*

### Synthesized button pins

Additional buttons are synthesized for buttons named **zero**, **goto-zero**, and **half**. These synthesized buttons are active when the button is pressed AND the selector-switch is set to the corresponding axis [xyza].

(bit out) *xhc-hb04.button-zero-[xyza]*  
 (bit out) *xhc-hb04.button-goto-zero-[xyza]*  
 (bit out) *xhc-hb04.button-half-[xyza]*

### DEBUGGING

For debugging USB activity, use environmental variable LIBUSB\_DEBUG:

```
export LIBUSB_DEBUG=[2 | 3 | 4]; xhc-hb04 [options]
2:warning, 3:info, 4:debug
```

## Sim Configs

The distribution includes several simulation configurations in the directory:

`/usr/share/doc/linuxcnc/examples/sample-configs/sim/axis/xhc-hb04/`

or for a RIP build:

`rip_base_dir/configs/sim/axis/xhc-hb04/`

These configurations use a distribution-provided script (`xhc-hb04.tcl`) to configure the pendant and make necessary HAL connections according to a number of ini file settings. The script uses an additional HAL component (`xhc_hb04_util`) to provide common functionality and includes support for a standard method for the start-pause button.

The settings available include:

- 1) specify button-cfg-file for standard layout1 or layout2
- 2) select axes (up to 4 axes from set of x y z a b c u v w)
- 3) implement per-axis filtering coefficients
- 4) implement per-axis acceleration for mpg jogging
- 5) implement per-axis scale settings
- 6) select normal or velocity based jog modes
- 7) select stepsize sequence
- 8) option to initialize pin for inch or mm display icon
- 9) option to require pendant on startup

The sim configs illustrate button connections that:

- 1) connect pendant stepsize-up button to the step input pin.
- 2) connect buttons to `halui.* pins`
- 3) connect buttons to `motion.* pins`

Another script is included to monitor the pendant and report loss of USB connectivity. See the README and .txt files in the above directory for usage.

**Note:** The sim configs use the axis gui but the scripts are available with any HAL configuration or gui. The same scripts can be used to adapt the xhc-hb04 to existing configurations provided that the `halui`, `motion`, and `axis.N` pins needed are not otherwise claimed. Instructions are included in README file in the directory named above.

Use `halcmd` to display the pins and signals used by the `xhc-hb04.tcl` script:

```
halcmd show pin xhc-hb04      (show all xhc-hb04 pins)
halcmd show pin pendant_util  (show all pendant_util pins)
halcmd show sig pendant:      (show all pendant signals)
```

## Author

Frederick Rible ([fribler@teaser.fr](mailto:fribler@teaser.fr))

## NAME

hal – Introduction to the HAL API

## DESCRIPTION

HAL stands for Hardware Abstraction Layer, and is used by LinuxCNC to transfer realtime data to and from I/O devices and other low-level modules.

**hal.h** defines the API and data structures used by the HAL. This file is included in both realtime and non-realtime HAL components. HAL uses the RTPAI real time interface, and the #define symbols RTAPI and ULAPI are used to distinguish between realtime and non-realtime code. The API defined in this file is implemented in `hal_lib.c` and can be compiled for linking to either realtime or user space HAL components.

The HAL is a very modular approach to the low level parts of a motion control system. The goal of the HAL is to allow a systems integrator to connect a group of software components together to meet whatever I/O requirements he (or she) needs. This includes realtime and non-realtime I/O, as well as basic motor control up to and including a PID position loop. What these functions have in common is that they all process signals. In general, a signal is a data item that is updated at regular intervals. For example, a PID loop gets position command and feedback signals, and produces a velocity command signal.

HAL is based on the approach used to design electronic circuits. In electronics, off-the-shelf components like integrated circuits are placed on a circuit board and their pins are interconnected to build whatever overall function is needed. The individual components may be as simple as an op-amp, or as complex as a digital signal processor. Each component can be individually tested, to make sure it works as designed. After the components are placed in a larger circuit, the signals connecting them can still be monitored for testing and troubleshooting.

Like electronic components, HAL components have pins, and the pins can be interconnected by signals.

In the HAL, a *signal* contains the actual data value that passes from one pin to another. When a signal is created, space is allocated for the data value. A *pin* on the other hand, is a pointer, not a data value. When a pin is connected to a signal, the pin's pointer is set to point at the signal's data value. This allows the component to access the signal with very little run-time overhead. (If a pin is not linked to any signal, the pointer points to a dummy location, so the realtime code doesn't have to deal with null pointers or treat unlinked variables as a special case in any way.)

There are three approaches to writing a HAL component. Those that do not require hard realtime performance can be written as a single user mode process. Components that need hard realtime performance but have simple configuration and init requirements can be done as a single kernel module, using either pre-defined init info, or insmod-time parameters. Finally, complex components may use both a kernel module for the realtime part, and a user space process to handle ini file access, user interface (possibly including GUI features), and other details.

HAL uses the RTAPI/ULAPI interface. If RTAPI is #defined `hal_lib.c` would generate a kernel module `hal_lib.o` that is insmoded and provides the functions for all kernel module based components. The same source file compiled with the ULAPI #define would make a user space `hal_lib.o` that is statically linked to user space code to make user space executables. The variable lists and link information are stored in a block of shared memory and protected with mutexes, so that kernel modules and any of several user mode programs can access the data.

## REALTIME CONSIDERATIONS

For an explanation of realtime considerations, see **intro(3rtapi)**.

**HAL STATUS CODES**

Except as noted in specific manual pages, HAL returns negative errno values for errors, and nonnegative values for success.

**SEE ALSO**

**intro(3rtapi)**

**NAME**

hal\_add\_funct\_to\_thread – cause a function to be executed at regular intervals

**SYNTAX**

```
int hal_add_funct_to_thread(const char *funct_name, const char *thread_name,
                           int position)
```

```
int hal_del_funct_from_thread(const char *funct_name, const char *thread_name)
```

**ARGUMENTS**

*funct\_name*

The name of the function

*thread\_name*

The name of the thread

*position*

The desired location within the thread. This determines when the function will run, in relation to other functions in the thread. A positive number indicates the desired location as measured from the beginning of the thread, and a negative is measured from the end. So +1 means this function will become the first one to run, +5 means it will be the fifth one to run, -2 means it will be next to last, and -1 means it will be last. Zero is illegal.

**DESCRIPTION**

**hal\_add\_funct\_to\_thread** adds a function exported by a realtime HAL component to a realtime thread. This determines how often and in what order functions are executed.

**hal\_del\_funct\_from\_thread** removes a function from a thread.

**RETURN VALUE**

Returns a HAL status code.

**REALTIME CONSIDERATIONS**

Call only from realtime init code, not from user space or realtime code.

**SEE ALSO**

**hal\_thread\_new(3hal), hal\_export\_funct(3hal)**

**NAME**

hal\_create\_thread – Create a HAL thread

**SYNTAX**

int hal\_create\_thread(const char \**name*, unsigned long *period*, int *uses\_fp*)

int hal\_thread\_delete(const char \**name*)

**ARGUMENTS**

*name*     The name of the thread

*period*   The interval, in nanoseconds, between iterations of the thread

*uses\_fp*   Must be nonzero if a function which uses floating-point will be attached to this thread.

**DESCRIPTION**

**hal\_create\_thread** establishes a realtime thread that will execute one or more HAL functions periodically.

All thread periods are rounded to integer multiples of the hardware timer period, and the timer period is based on the first thread created. Threads must be created in order, from the fastest to the slowest. HAL assigns decreasing priorities to threads that are created later, so creating them from fastest to slowest results in rate monotonic priority scheduling.

**hal\_delete\_thread** deletes a previously created thread.

**REALTIME CONSIDERATIONS**

Call only from realtime init code, not from user space or realtime code.

**RETURN VALUE**

Returns a HAL status code.

**SEE ALSO**

**hal\_export\_funct(3hal)**

**NAME**

hal\_exit – Shut down HAL

**SYNTAX**

int hal\_exit(int *comp\_id*)

**ARGUMENTS**

*comp\_id*

A HAL component identifier returned by an earlier call to **hal\_init**.

**DESCRIPTION**

**hal\_exit** shuts down and cleans up HAL and RTAPI. It must be called prior to exit by any module that called **hal\_init**.

**REALTIME CONSIDERATIONS**

Call only from within user or init/cleanup code, not from realtime tasks.

**RETURN VALUE**

Returns a HAL status code.

**NAME**

hal\_export\_funct – create a realtime function callable from a thread

**SYNTAX**

```
typedef void(*hal_funct_t)(void * arg, long period)
```

```
int hal_export_funct(const char *name, hal_funct_t funct, void *arg, int uses_fp, int reentrant, int comp_id)
```

**ARGUMENTS**

*name*     The name of the function.

*funct*    The pointer to the function

*arg*       The argument to be passed as the first parameter of *funct*

*uses\_fp*   Nonzero if the function uses floating-point operations, including assignment of floating point values with "=".

*reentrant*

If *reentrant* is non-zero, the function may be preempted and called again before the first call completes. Otherwise, it may only be added to one thread.

*comp\_id*

A HAL component identifier returned by an earlier call to **hal\_init**.

**DESCRIPTION**

**hal\_export\_funct** makes a realtime function provided by a component available to the system. A subsequent call to **hal\_add\_funct\_to\_thread** can be used to schedule the execution of the function as needed by the system.

When this function is placed on a HAL thread, and HAL threads are started, *funct* is called repeatedly with two arguments: *void \*arg* is the same value that was given to **hal\_export\_funct**, and *long period* is the interval between calls in nanoseconds.

Each call to the function should do a small amount of work and return.

**RETURN VALUE**

Returns a HAL status code.

**SEE ALSO**

**hal\_create\_thread(3hal)**, **hal\_add\_funct\_to\_thread(3hal)**

**NAME**

hal\_init – Sets up HAL and RTAPI

**SYNTAX**

```
int hal_init(const char *modname)
```

**ARGUMENTS**

*modname*

The name of this hal module

**DESCRIPTION**

**hal\_init** sets up HAL and RTAPI. It must be called by any module that intends to use the API, before any other RTAPI calls.

*modname* must point to a string that identifies the module. The string may be no longer than **HAL\_NAME\_LEN** characters.

**REALTIME CONSIDERATIONS**

Call only from within user or init/cleanup code, not from realtime tasks.

**RETURN VALUE**

On success, returns a positive integer module ID, which is used for subsequent calls to hal and rtapi APIs.

On failure, returns a HAL error code.

**NAME**

hal\_malloc – Allocate space in the HAL shared memory area

**SYNTAX**

```
void *hal_malloc(long int size)
```

**ARGUMENTS**

*size*      Gives the size, in bytes, of the block

**DESCRIPTION**

**hal\_malloc** allocates a block of memory from the main HAL shared memory area. It should be used by all components to allocate memory for HAL pins and parameters. It allocates ‘size’ bytes, and returns a pointer to the allocated space, or NULL (0) on error. The returned pointer will be properly aligned for any type HAL supports. A component should allocate during initialization all the memory it needs.

The allocator is very simple, and there is no ‘free’. The entire HAL shared memory area is freed when the last component calls **hal\_exit**. This means that if you continuously install and remove one component while other components are present, you eventually will fill up the shared memory and an install will fail. Removing all components completely clears memory and you start fresh.

**RETURN VALUE**

A pointer to the allocated space, which is properly aligned for any variable HAL supports. Returns NULL on error.

**NAME**

hal\_param\_new – Create a HAL parameter

**SYNTAX**

```
int hal_param_bit_new(const char *name, hal_param_dir_t dir, hal_bit_t * data_addr, int
comp_id)
```

```
int hal_param_float_new(const char *name, hal_param_dir_t dir, hal_float_t * data_addr, int
comp_id)
```

```
int hal_param_u32_new(const char *name, hal_param_dir_t dir, hal_u32_t * data_addr, int
comp_id)
```

```
int hal_param_s32_new(const char *name, hal_param_dir_t dir, hal_s32_t * data_addr, int
comp_id)
```

```
int hal_param_bit_newf(hal_param_dir_t dir, hal_bit_t * data_addr, int comp_id, const char *fmt,
...)
```

```
int hal_param_float_newf(hal_param_dir_t dir, hal_float_t * data_addr, int comp_id, const char
*fmt, ...)
```

```
int hal_param_u32_newf(hal_param_dir_t dir, hal_u32_t * data_addr, int comp_id, const char
*fmt, ...)
```

```
int hal_param_s32_newf(hal_param_dir_t dir, hal_s32_t * data_addr, int comp_id, const char
*fmt, ...)
```

```
int hal_param_new(const char *name, hal_type_t type, hal_param_dir_t dir, void *data_addr, int
comp_id)
```

**ARGUMENTS**

- name*     The name to give to the created parameter
- dir*       The direction of the parameter, from the viewpoint of the component. It may be one of **HAL\_RO**, or **HAL\_RW**. A component may assign a value to any parameter, but other programs (such as hal-cmd) may only assign a value to a parameter that is **HAL\_RW**.
- data\_addr*     The address of the data, which must lie within memory allocated by **hal\_malloc**.

*comp\_id*

A HAL component identifier returned by an earlier call to **hal\_init**.

*fmt, ...* A printf-style format string and arguments

*type* The type of the parameter, as specified in **hal\_type\_t(3hal)**.

## DESCRIPTION

The **hal\_param\_new** family of functions create a new *param* object.

There are functions for each of the data types that the HAL supports. Pins may only be linked to signals of the same type.

## RETURN VALUE

Returns a HAL status code.

## SEE ALSO

**hal\_type\_t(3hal)**

**NAME**

hal\_parport – portable access to PC-style parallel ports

**SYNTAX**

```
#include "hal_parport.h"
```

```
int hal_parport_get(int comp_id, hal_parport_t *port, unsigned short base, unsigned short base_hi,
    unsigned int modes)
```

```
void hal_parport_release(hal_parport_t *port)
```

**ARGUMENTS**

*comp\_id*

A HAL component identifier returned by an earlier call to **hal\_init**.

*port* A pointer to a hal\_parport\_t structure

*base* The base address of the port (if port >= 16) or the linux port number of the port (if port < 16)

*base\_hi*

The "high" address of the port (location of the ECP registers), 0 to use a probed high address, or -1 to disable the high address

*modes* Advise the driver of the desired port modes, from <linux/parport.h>. If a linux-detected port does not provide the requested modes, a warning is printed with rtapi\_print\_msg. This does not make the port request fail, because unfortunately, many systems that have working EPP parports are not detected as such by Linux.

**DESCRIPTION**

**hal\_parport\_get** allocates a parallel port for exclusive use of the named hal component. The port must be released with **hal\_parport\_release** before the component exits with **hal\_exit**.

**HIGH ADDRESS PROBING**

If the port is a parallel port known to Linux, and Linux detected a high I/O address, this value is used. Otherwise, if base+0x400 is not registered to any device, it is used. Otherwise, no address is used. If no high address is detected, port->base\_hi is 0.

**PARPORT STRUCTURE**

```
typedef struct
{
    unsigned short base;
    unsigned short base_hi;
    .... // and further unspecified fields
} hal_parport_t;
```

**RETURN VALUE**

**hal\_parport\_get** returns a HAL status code. On success, *port* is filled out with information about the allocated port. On failure, the contents of *port* are undefined except that it is safe (but not required) to pass this port to **hal\_parport\_release**.

**hal\_parport\_release** does not return a value. It always succeeds.

**NOTES**

In new code, prefer use of rtapi\_parport to hal\_parport.

**NAME**

hal\_pin\_new – Create a HAL pin

**SYNTAX**

```
int hal_pin_bit_new(const char *name, hal_pin_dir_t dir, hal_bit_t ** data_ptr_addr, int
comp_id)
```

```
int hal_pin_float_new(const char *name, hal_pin_dir_t dir, hal_float_t ** data_ptr_addr, int
comp_id)
```

```
int hal_pin_u32_new(const char *name, hal_pin_dir_t dir, hal_u32_t ** data_ptr_addr, int
comp_id)
```

```
int hal_pin_s32_new(const char *name, hal_pin_dir_t dir, hal_s32_t ** data_ptr_addr, int
comp_id)
```

```
int hal_pin_bit_newf(hal_pin_dir_t dir, hal_bit_t ** data_ptr_addr, int comp_id, const char *fmt,
...)
```

```
int hal_pin_float_newf(hal_pin_dir_t dir, hal_float_t ** data_ptr_addr, int comp_id, const char
*fmt, ...)
```

```
int hal_pin_u32_newf(hal_pin_dir_t dir, hal_u32_t ** data_ptr_addr, int comp_id, const char
*fmt, ...)
```

```
int hal_pin_s32_newf(hal_pin_dir_t dir, hal_s32_t ** data_ptr_addr, int comp_id, const char
*fmt, ...)
```

```
int hal_pin_new(const char *name, hal_type_t type, hal_pin_dir_t dir, void **data_ptr_addr, int
comp_id)
```

**ARGUMENTS**

*name*    The name of the pin

*dir*

The direction of the pin, from the viewpoint of the component. It may be one of **HAL\_IN**, **HAL\_OUT**, or **HAL\_IO**. Any number of **HAL\_IN** or **HAL\_IO** pins may be connected to the same signal, but at most one **HAL\_OUT** pin is permitted. A component may assign a value to a pin that is **HAL\_OUT** or **HAL\_IO**, but may not assign a value to a pin that is **HAL\_IN**.

*data\_ptr\_addr*

The address of the pointer-to-data, which must lie within memory allocated by **hal\_malloc**.

*comp\_id*

A HAL component identifier returned by an earlier call to **hal\_init**.

*fmt,*

A printf-style format string and arguments

*type*

The type of the param, as specified in **hal\_type\_t(3hal)**.

## DESCRIPTION

The **hal\_pin\_new** family of functions create a new *pin* object. Once a pin has been created, it can be linked to a signal object using **hal\_link**. A pin contains a pointer, and the component that owns the pin can dereference the pointer to access whatever signal is linked to the pin. (If no signal is linked, it points to a dummy signal.)

There are functions for each of the data types that the HAL supports. Pins may only be linked to signals of the same type.

## RETURN VALUE

Returns a HAL status code.

## SEE ALSO

**hal\_type\_t(3hal)**, **hal\_link(3hal)**

**NAME**

`hal_ready` – indicates that this component is ready

**SYNTAX**

```
hal_ready(int comp_id)
```

**ARGUMENTS**

*comp\_id*

A HAL component identifier returned by an earlier call to **hal\_init**.

**DESCRIPTION**

**hal\_ready** indicates that this component is ready (has created all its pins, parameters, and functions). This must be called in any realtime HAL component before its **rtapi\_app\_init** exits, and in any userspace component before it enters its main loop.

**RETURN VALUE**

Returns a HAL status code.

**NAME**

hal\_set\_constructor – Set the constructor function for this component

**SYNTAX**

```
typedef int (*hal_constructor_t)(const char *prefix, const char *arg); int hal_set_constructor(int comp_id,  
hal_constructor_t constructor)
```

**ARGUMENTS**

*comp\_id* A HAL component identifier returned by an earlier call to **hal\_init**.

*prefix* The prefix to be given to the pins, parameters, and functions in the new instance

*arg* An argument that may be used by the component to customize this instance.

**DESCRIPTION**

As an experimental feature in HAL 2.1, components may be *constructable*. Such a component may create pins and parameters not only at the time the module is loaded, but it may create additional pins and parameters, and functions on demand.

**RETURN VALUE**

Returns a HAL status code.

**SEE ALSO**

**halcmd(1)**

**NAME**

hal\_set\_lock, hal\_get\_lock – Set or get the HAL lock level

**SYNTAX**

```
int hal_set_lock(unsigned char lock_type)
```

```
int hal_get_lock()
```

**ARGUMENTS**

*lock\_type*

The desired lock type, which may be a bitwise combination of: **HAL\_LOCK\_LOAD**, **HAL\_LOCK\_CONFIG**, **HAL\_LOCK\_PARAMS**, or **HAL\_LOCK\_PARAMS**. **HAL\_LOCK\_NONE** or 0 locks nothing, and **HAL\_LOCK\_ALL** locks everything.

**DESCRIPTION****RETURN VALUE**

**hal\_set\_lock** Returns a HAL status code. **hal\_get\_lock** returns the current HAL lock level or a HAL status code.

**NAME**

hal\_signal\_new, hal\_signal\_delete, hal\_link, hal\_unlink – Manipulate HAL signals

**SYNTAX**

```
int hal_signal_new(const char *signal_name, hal_type_t type)
```

```
int hal_signal_delete(const char *signal_name)
```

```
int hal_link(const char *pin_name, const char *signal_name)
```

```
int hal_unlink(const char *pin_name)
```

**ARGUMENTS**

*signal\_name*

The name of the signal

*pin\_name*

The name of the pin

*type*

The type of the signal, as specified in **hal\_type\_t(3hal)**.

**DESCRIPTION**

**hal\_signal\_new** creates a new signal object. Once a signal has been created, pins can be linked to it with **hal\_link**. The signal object contains the actual storage for the signal data. Pin objects linked to the signal have pointers that point to the data. 'name' is the name of the new signal. It may be no longer than HAL\_NAME\_LEN characters. If there is already a signal with the same name the call will fail.

**hal\_link** links a pin to a signal. If the pin is already linked to the desired signal, the command succeeds. If the pin is already linked to some other signal, it is an error. In either case, the existing connection is not modified. (Use 'hal\_unlink' to break an existing connection.) If the signal already has other pins linked to it, they are unaffected - one signal can be linked to many pins, but a pin can be linked to only one signal.

**hal\_unlink** unlinks any signal from the specified pin.

**hal\_signal\_delete** deletes a signal object. Any pins linked to the object are unlinked.

**RETURN VALUE**

Returns a HAL status code.

**SEE ALSO**

**hal\_type\_t(3hal)**

**NAME**

hal\_start\_threads – Allow HAL threads to begin executing

**SYNTAX**

```
int hal_start_threads()
```

```
int hal_stop_threads()
```

**ARGUMENTS****DESCRIPTION**

**hal\_start\_threads** starts all threads that have been created. This is the point at which realtime functions start being called.

**hal\_stop\_threads** stops all threads that were previously started by **hal\_start\_threads**. It should be called before any component that is part of a system exits.

**RETURN VALUE**

Returns a HAL status code.

**SEE ALSO**

**hal\_export\_funct(3hal)**, **hal\_create\_thread(3hal)**, **hal\_add\_funct\_to\_thread(3hal)**

**NAME**

hal\_type\_t – typedefs for HAL datatypes

**DESCRIPTION**

typedef ... **hal\_bool**;

A type which may have a value of 0 or nonzero.

typedef ... **hal\_bit\_t**;

A volatile type which may have a value of 0 or nonzero.

typedef ... **hal\_s32\_t**;

A volatile type which may have a value from -2147483648 to 2147483647.

typedef ... **hal\_u32\_t**;

A volatile type which may have a value from 0 to 4294967295.

typedef ... **hal\_float\_t**;

A volatile floating-point type, which typically has the same precision and range as the C type **double**.

typedef ... **real\_t**;

A nonvolatile floating-point type with at least as much precision as **hal\_float\_t**.

typedef ... **ireal\_t**;

A nonvolatile unsigned integral type the same size as **hal\_float\_t**.

typedef enum **hal\_type\_t**;

**HAL\_BIT**

Corresponds to the type **hal\_bit\_t**.

**HAL\_FLOAT**

Corresponds to the type **hal\_float\_t**.

**HAL\_S32**

Corresponds to the type **hal\_s32\_t**.

**HAL\_U32**

Corresponds to the type **hal\_u32\_t**.

**NOTES**

**hal\_bit\_t** is typically a typedef to an integer type whose range is larger than just 0 and 1. When testing the value of a **hal\_bit\_t**, never compare it to 1. Prefer one of the following:

- if(b)
- if(b != 0)

It is often useful to refer to a type that can represent all the values as a hal type, but without the volatile qualifier. The following types correspond with the hal types:

hal_bit_t	int
hal_s32_t	__s32
hal_u32_t	__u32
hal_float_t	hal_real_t

Take care not to use the types **s32** and **u32**. These will compile in kernel modules but not in userspace, and not for "realtime components" when using simulated (userspace) realtime.

hal\_type\_t(3hal)

HAL

hal\_type\_t(3hal)

**SEE ALSO**

**hal\_pin\_new(3hal), hal\_param\_new(3hal)**

**NAME**

undocumented – undocumented functions in HAL

**SEE ALSO**

The header file *hal.h*. Most hal functions have documentation in that file.

## NAME

rtapi – Introduction to the RTAPI API

## DESCRIPTION

RTAPI is a library providing a uniform API for several real time operating systems. As of LinuxCNC 2.7, POSIX threads and RTAI are supported.

## HEADER FILES

### **rtapi.h**

The file **rtapi.h** defines the RTAPI for both realtime and non-realtime code. This is a change from Rev 2, where the non-realtime (user space) API was defined in `ulapi.h` and used different function names. The symbols RTAPI and ULAPI are used to determine which mode is being compiled, RTAPI for realtime and ULAPI for non-realtime.

### **rtapi\_math.h**

The file `rtapi_math.h` defines floating-point functions and constants. It should be used instead of `<math.h>` in `rtapi` real-time components.

### **rtapi\_string.h**

The file `rtapi_string.h` defines string-related functions. It should be used instead of `<string.h>` in `rtapi` real-time components.

### **rtapi\_byteorder.h**

This file defines the preprocessor macros `RTAPI_BIG_ENDIAN`, `RTAPI_LITTLE_ENDIAN`, and `RTAPI_FLOAT_BIG_ENDIAN` as true or false depending on the characteristics of the target system. It should be used instead of `<endian.h>` (userspace) or `<linux/byteorder.h>` (kernel space).

### **rtapi\_limits.h**

This file defines the minimum and maximum value of some fundamental integral types, such as `INT_MIN` and `INT_MAX`. This should be used instead of `<limits.h>` because that header file is not available to kernel modules.

## REALTIME CONSIDERATIONS

### **Userspace code**

Certain functions are not available in userspace code. This includes functions that perform direct device access such as **rtapi\_inb(3)**.

### **Init/cleanup code**

Certain functions may only be called from realtime init/cleanup code. This includes functions that perform memory allocation, such as **rtapi\_shmem\_new(3)**.

### **Realtime code**

Only a few functions may be called from realtime code. This includes functions that perform direct device access such as **rtapi\_inb(3)**. It excludes most Linux kernel APIs such as `do_gettimeofday(3)` and many `rtapi` APIs such as `rtapi_shmem_new(3)`.

### **Simulator**

For an RTAPI module to be buildable in the "sim" environment (fake realtime system without special privileges), it must not use **any** linux kernel APIs, and must not use the RTAPI APIs for direct device access

such as **rtapi\_inb(3)**. This automatically includes any hardware device drivers, and also devices which use Linux kernel APIs to do things like create special devices or entries in the **/proc** filesystem.

## **RTAPI STATUS CODES**

Except as noted in specific manual pages, RTAPI returns negative errno values for errors, and nonnegative values for success.

**NAME**

rtapi\_app\_exit – User-provided function to shut down a component

**SYNTAX**

```
#include <rtapi_app.h>
void rtapi_app_exit(void) {...}
```

**ARGUMENTS**

None

**DESCRIPTION**

The body of **rtapi\_app\_exit**, which is provided by the component author, generally consists of a call to **rtapi\_exit** or **hal\_exit**, preceded by other component-specific shutdown code.

This code is called when unloading a component which successfully initialized (i.e., returned zero from its **rtapi\_app\_main**). It is not called when the component did not successfully initialize.

**RETURN CODE**

None.

**REALTIME CONSIDERATIONS**

Called automatically by the rtapi infrastructure in an initialization (not realtime) context.

**SEE ALSO**

**rtapi\_app\_main(3rtapi)**, **rtapi\_exit(3rtapi)**, **hal\_exit(3hal)**

**NAME**

rtapi\_app\_main – User-provided function to initialize a component

**SYNTAX**

```
#include <rtapi_app.h>
int rtapi_app_main(void) {...}
```

**ARGUMENTS**

None

**DESCRIPTION**

The body of **rtapi\_app\_main**, which is provided by the component author, generally consists of a call to `rtapi_init` or `hal_init`, followed by other component-specific initialization code.

**RETURN VALUE**

Return 0 for success. Return a negative `errno` value (e.g., `-EINVAL`) on error. Existing code also returns RTAPI or HAL error values, but using negative `errno` values gives better diagnostics from `insmod`.

**REALTIME CONSIDERATIONS**

Called automatically by the `rtapi` infrastructure in an initialization (not realtime) context.

**SEE ALSO**

**rtapi\_app\_exit(3rtapi), rtapi\_init(3rtapi), hal\_init(3hal)**

**NAME**

rtapi\_atomic – subset of C11 <stdatomic.h>

**SYNTAX**

```
#include <rtapi_atomic.h>

enum memory_order { ... };

#define atomic_store(obj, desired)...

#define atomic_store_explicit(obj, desired, order)...

#define atomic_load(obj)...

#define atomic_load_explicit(obj, order)...
```

**ARGUMENTS****volatile A\* *obj***

A pointer to a volatile object that is the destination of the store or the source of the load. The pointer must have an appropriate type and alignment such that the underlying store or load operation itself is atomic; at a minimum, a properly aligned "int" may be assumed to be such a type. Improper size or alignment are undiagnosed errors.

**C *desired***

The value to be stored in the object. `"*obj = desired"` must be well-formed.

**memory\_order *order***

The required memory ordering semantic.

**DESCRIPTION**

This header provides at least the subset of C11's <stdatomic.h> given above. When there is an ordering requirement for multiple values read or written in RTAPI shared memory areas by other threads of execution, including the values of HAL pins and parameters, these functions (or function-like macros) are the only way to ensure the ordering requirement is obeyed. Otherwise, according to architecture-specific rules, loads and stores may be reordered from their normal source code order.

For example, to leave a message in a shared memory area from one thread and retrieve it from another, the writer must use an atomic store for the "message is complete" variable, and the reader must use an atomic load when checking that variable:

```
// producer
*message = 42;
atomic_store_explicit(message_ready, 1, memory_order_release);

// consumer
while(atomic_load_explicit(message_ready, memory_order_acquire) == 0) sched
printf("message was %d\n", *message); // must print 42
```

**REALTIME CONSIDERATIONS**

May be called from any code.

**RETURN VALUE**

**atomic\_load** and **atomic\_load\_explicit** return the value pointed to by the *obj* argument.

**atomic\_store** and **atomic\_store\_explicit** have no return value.

**SEE ALSO**

**<stdatomic.h>** (C11), **<rtapi\_bitops.h>** (for other atomic memory operations supported by rtapi)

**NAME**

rtapi\_bool.h – RTAPI wrappers for linux kernel functionality

**SYNTAX**

```
#include <rtapi_bool.h>
```

**DESCRIPTION**

Includes either <stdbool.h> or <linux/types.h> as appropriate, to obtain suitable declarations of "bool", "true" and "false".

**REALTIME CONSIDERATIONS**

None.

**NOTES**

Also permitted in C++ programs, where including it has no effect.

**NAME**

rtapi\_byteorder.h – RTAPI wrappers for linux kernel functionality

**SYNTAX**

```
#include <rtapi_byteorder.h>
```

RTAPI\_BIG\_ENDIAN

Defined to 1 if the platform is big-endian, 0 otherwise

RTAPI\_LITTLE\_ENDIAN

Defined to 1 if the platform is little-endian, 0 otherwise

RTAPI\_FLOAT\_BIG\_ENDIAN

Defined to 1 if the platform double-precision value is big-endian, 0 otherwise.

**DESCRIPTION**

In kernel space, each rtapi\_xxx or RTAPI\_XXX identifier is mapped to the underlying kernel functionality, if available.

In userspace, or in kernels where the underlying functionality is not provided by a kernel, generally another implementation--possibly with reduced functionality--is provided. (For example, the userspace implementation for rtapi\_byteorder\_register always succeeds)

**REALTIME CONSIDERATIONS**

May be used at any time.

**RETURN VALUE**

As in Linux.

**SEE ALSO**

**NAME**

rtapi\_clock\_set\_period – set the basic time interval for realtime tasks

**SYNTAX**

```
rtapi_clock_set_period(long int nsec)
```

**ARGUMENTS**

*nsec*     The desired basic time interval for realtime tasks.

**DESCRIPTION**

**rtapi\_clock\_set\_period** sets the basic time interval for realtime tasks. All periodic tasks will run at an integer multiple of this period. The first call to **rtapi\_clock\_set\_period** with *nsec* greater than zero will start the clock, using *nsec* as the clock period in nano-seconds. Due to hardware and RTOS limitations, the actual period may not be exactly what was requested. On success, the function will return the actual clock period if it is available, otherwise it returns the requested period. If the requested period is outside the limits imposed by the hardware or RTOS, it returns **-EINVAL** and does not start the clock. Once the clock is started, subsequent calls with non-zero *nsec* return **-EINVAL** and have no effect. Calling **rtapi\_clock\_set\_period** with *nsec* set to zero queries the clock, returning the current clock period, or zero if the clock has not yet been started.

**REALTIME CONSIDERATIONS**

Call only from within init/cleanup code, not from realtime tasks. This function is not available from user (non-realtime) code.

**RETURN VALUE**

The actual period provided by the RTOS, which may be different than the requested period, or a RTAPI status code.

**NAME**

rtapi\_delay – Busy-loop for short delays

**SYNTAX**

void rtapi\_delay(long int *nsec*)

void rtapi\_delay\_max()

**ARGUMENTS**

*nsec*     The desired delay length in nanoseconds

**DESCRIPTION**

**rtapi\_delay** is a simple delay. It is intended only for short delays, since it simply loops, wasting CPU cycles.

**rtapi\_delay\_max** returns the max delay permitted (usually approximately 1/4 of the clock period). Any call to **rtapi\_delay** requesting a delay longer than the max will delay for the max time only.

**rtapi\_delay\_max** should be called before using **rtapi\_delay** to make sure the required delays can be achieved. The actual resolution of the delay may be as good as one nano-second, or as bad as a several microseconds.

**REALTIME CONSIDERATIONS**

May be called from init/cleanup code, and from within realtime tasks.

**RETURN VALUE**

**rtapi\_delay\_max** returns the maximum delay permitted.

**SEE ALSO**

**rtapi\_clock\_set\_period(3rtapi)**

**NAME**

rtapi\_device.h – RTAPI wrappers for linux kernel functionality

**SYNTAX**

```
#include <rtapi_device.h>

struct rtapi_device;

int rtapi_dev_set_name(struct rtapi_device *dev, const char *name, ...);
int rtapi_device_register(struct rtapi_device *dev);
int rtapi_device_unregister(struct rtapi_device *dev);
```

**DESCRIPTION**

In kernel space, each `rtapi_xxx` or `RTAPI_XXX` identifier is mapped to the underlying kernel functionality, if available.

In userspace, or in kernels where the underlying functionality is not provided by a kernel, generally another implementation--possibly with reduced functionality--is provided. (For example, the userspace implementation for `rtapi_device_register` always succeeds)

**REALTIME CONSIDERATIONS**

Typically, these functions may be called from realtime init/cleanup code.

**RETURN VALUE**

As in Linux.

**SEE ALSO**

**NAME**

rtapi\_div\_u64 – unsigned division of a 64-bit number by a 32-bit number

**SYNTAX**

`__u64 rtapi_div_u64_rem(__u64 dividend, __u32 divisor, __u32 *remainder)`

`__u64 rtapi_div_u64(__u64 dividend, __u32 divisor)`

`__s64 rtapi_div_s64(__s64 dividend, __s32 divisor)`

`__s64 rtapi_div_s64_rem(__s64 dividend, __s32 divisor, __s32 *remainder)`

**ARGUMENTS**

*dividend*

The value to be divided

*divisor* The value to divide by

*remainder*

Pointer to the location to store the remainder. This may not be a NULL pointer. If the remainder is not desired, call **rtapi\_div\_u64** or **rtapi\_div\_s64**.

**DESCRIPTION**

Perform integer division (and optionally compute the remainder) with a 64-bit dividend and 32-bit divisor.

**RETURN VALUE**

The result of integer division of *dividend* / *divisor*. In versions with the *remainder* argument, the remainder is stored in the pointed-to location.

**NOTES**

If the result of the division does not fit in the return type, the result is undefined.

This function exists because in kernel space the use of the division operator on a 64-bit type can lead to an undefined symbol such as `__umoddi3` when the module is loaded.

**REALTIME CONSIDERATIONS**

May be called from init/cleanup code and from within realtime tasks. Available in userspace components.

**NAME**

rtapi\_exit – Shut down RTAPI

**SYNTAX**

```
int rtapi_exit(int module_id)
```

**ARGUMENTS**

*module\_id*

An rtapi module identifier returned by an earlier call to **rtapi\_init**.

**DESCRIPTION**

**rtapi\_exit** shuts down and cleans up the RTAPI. It must be called prior to exit by any module that called **rtapi\_init**.

**REALTIME CONSIDERATIONS**

Call only from within user or init/cleanup code, not from realtime tasks.

**RETURN VALUE**

Returns a RTAPI status code.

**NAME**

rtapi\_firmware.h – RTAPI wrappers for linux kernel functionality

**SYNTAX**

```
#include <rtapi_firmware.h>

struct rtapi_firmware;

int rtapi_request_firmware(const struct rtapi_firmware **fw,
                           const char *name, struct rtapi_device *device);

void rtapi_release_firmware(const struct rtapi_firmware *fw);
```

**DESCRIPTION**

In kernel space, each `rtapi_xxx` or `RTAPI_XXX` identifier is mapped to the underlying kernel functionality, if available.

In userspace, or in kernels where the underlying functionality is not provided by a kernel, generally another implementation--possibly with reduced functionality--is provided. (For example, the userspace implementation for `rtapi_device_register` always succeeds)

**REALTIME CONSIDERATIONS**

Typically, these functions may be called from realtime init/cleanup code.

**RETURN VALUE**

As in Linux.

**SEE ALSO**

**NAME**

rtapi\_get\_time – get the current time

**SYNTAX**

long long rtapi\_get\_time()

long long rtapi\_get\_clocks()

**DESCRIPTION**

**rtapi\_get\_time** returns the current time in nanoseconds. Depending on the RTOS, this may be time since boot, or time since the clock period was set, or some other time. Its absolute value means nothing, but it is monotonically increasing and can be used to schedule future events, or to time the duration of some activity. Returns a 64 bit value. The resolution of the returned value may be as good as one nano-second, or as poor as several microseconds. May be called from init/cleanup code, and from within realtime tasks.

**rtapi\_get\_clocks** returns the current time in CPU clocks. It is fast, since it just reads the TSC in the CPU instead of calling a kernel or RTOS function. Of course, times measured in CPU clocks are not as convenient, but for relative measurements this works fine. Its absolute value means nothing, but it is monotonically increasing and can be used to schedule future events, or to time the duration of some activity. (on SMP machines, the two TSC's may get out of sync, so if a task reads the TSC, gets swapped to the other CPU, and reads again, the value may decrease. RTAPI tries to force all RT tasks to run on one CPU.) Returns a 64 bit value. The resolution of the returned value is one CPU clock, which is usually a few nanoseconds to a fraction of a nanosecond.

Note that *long long* math may be poorly supported on some platforms, especially in kernel space. Also note that `rtapi_print()` will NOT print *long longs*. Most time measurements are relative, and should be done like this:

```
deltat = (long int)(end_time - start_time);
```

where `end_time` and `start_time` are `longlong` values returned from `rtapi_get_time`, and `deltat` is an ordinary `long int` (32 bits). This will work for times up to a second or so, depending on the CPU clock frequency. It is best used for millisecond and microsecond scale measurements though.

**RETURN VALUE**

Returns the current time in nanoseconds or CPU clocks.

**NOTES**

Certain versions of the Linux kernel provide a global variable `cpu_khz`. Computing

```
deltat = (end_clocks - start_clocks) / cpu_khz;
```

gives the duration measured in milliseconds. Computing

```
deltat = (end_clocks - start_clocks) * 1000000 / cpu_khz;
```

gives the duration measured in nanoseconds for deltas less than about 9 trillion clocks (e.g., 3000 seconds at 3GHz).

**REALTIME CONSIDERATIONS**

May be called from init/cleanup code and from within realtime tasks. Not available in userspace components.

**NAME**

rtapi\_gfp.h – RTAPI wrappers for linux kernel functionality

**SYNTAX**

```
#include <rtapi_gfp.h>
enum rtapi_gfp_e;
RTAPI_GFP_xxx
typedef ... rtapi_gfp_t;
```

**DESCRIPTION**

In kernel space, each `rtapi_xxx` or `RTAPI_XXX` identifier is mapped to the underlying kernel functionality, if available.

In userspace, or in kernels where the underlying functionality is not provided by a kernel, generally another implementation--possibly with reduced functionality--is provided. (For example, the userspace implementation for `rtapi_device_register` always succeeds)

**REALTIME CONSIDERATIONS**

Typically, these functions may be called from realtime init/cleanup code.

**RETURN VALUE**

As in Linux.

**SEE ALSO**

**NAME**

rtapi\_init – Sets up RTAPI

**SYNTAX**

```
int rtapi_init(const char *modname)
```

**ARGUMENTS**

*modname*

The name of this rtapi module

**DESCRIPTION**

**rtapi\_init** sets up the RTAPI. It must be called by any module that intends to use the API, before any other RTAPI calls.

*modname* can optionally point to a string that identifies the module. The string will be truncated at **RTAPI\_NAME\_LEN** characters. If *modname* is **NULL**, the system will assign a name.

**REALTIME CONSIDERATIONS**

Call only from within user or init/cleanup code, not from realtime tasks.

**RETURN VALUE**

On success, returns a positive integer module ID, which is used for subsequent calls to `rtapi_xxx_new`, `rtapi_xxx_delete`, and `rtapi_exit`. On failure, returns an RTAPI error code.

**NAME**

rtapi\_io.h – RTAPI wrappers for linux kernel functionality

**SYNTAX**

```
#include <rtapi_io.h>

unsigned char rtapi_inb(unsigned short int port);
unsigned short rtapi_inw(unsigned short int port);
unsigned int rtapi_inl(unsigned short int port);
unsigned void rtapi_inb(unsigned char value, unsigned short int port);
unsigned void rtapi_inw(unsigned short value, unsigned short int port);
unsigned void rtapi_inl(unsigned int value, unsigned short int port);
```

**DESCRIPTION**

In kernel space, each `rtapi_xxx` or `RTAPI_XXX` identifier is mapped to the underlying kernel functionality, if available.

In userspace, or in kernels where the underlying functionality is not provided by a kernel, generally another implementation--possibly with reduced functionality--is provided. (For example, the userspace implementation for `rtapi_device_register` always succeeds)

**REALTIME CONSIDERATIONS**

Call from init/cleanup code and from realtime tasks. These functions will cause illegal instruction exceptions in userspace components, as well as in uspace `rtapi_app` when it is not setuid root.

**RETURN VALUE**

As in Linux.

**SEE ALSO**

**NAME**

rtapi\_is – details of rtapi configuration

**SYNTAX**

int rtapi\_is\_kernelspace()

int rtapi\_is\_realtime()

**DESCRIPTION**

**rtapi\_is\_kernelspace()** returns nonzero when rtapi modules run in kernel space (e.g., under rtai) and zero when they run in userpace (e.g., under uspace).

**rtapi\_is\_realtime()** returns nonzero when capable of running with realtime guarantees. For rtai, this always returns nonzero (but actually loading realtime modules will fail if not running under the appropriate kernel). For uspace, this returns nonzero when the running kernel indicates it is capable of realtime performance. If **rtapi\_app** is not setuid root, this returns nonzero even though **rtapi\_app** will not be able to obtain realtime scheduling or hardware access, so e.g., attempting to **loadrt** a hardware driver will fail.

**REALTIME CONSIDERATIONS**

May be called from userspace or from realtime setup code. **rtapi\_is\_realtime()** may perform filesystem I/O.

**RETURN VALUE**

Zero for false, nonzero for true.

**NAME**

rtapi\_list.h – RTAPI wrappers for linux kernel functionality

**SYNTAX**

```
#include <rtapi_list.h>

struct rtapi_list_head;

void rtapi_list_add(struct rtapi_list_head *new_, struct rtapi_list_head *head);
void rtapi_list_add_tail(struct rtapi_list_head *new_, struct rtapi_list_head *head);
void rtapi_list_del(struct rtapi_list_head *entry);
void RTAPI_INIT_LIST_HEAD(struct rtapi_list_head *entry);
rtapi_list_for_each(pos, head) { ... }
rtapi_list_entry(ptr, type, member)
```

**DESCRIPTION**

In kernel space, each `rtapi_xxx` or `RTAPI_XXX` identifier is mapped to the underlying kernel functionality, if available.

In userspace, or in kernels where the underlying functionality is not provided by a kernel, generally another implementation--possibly with reduced functionality--is provided. (For example, the userspace implementation for `rtapi_device_register` always succeeds)

**REALTIME CONSIDERATIONS**

Call from init/cleanup code and from realtime tasks. These functions will cause illegal instruction exceptions in userspace components, as well as in `rtapi_app` when it is not setuid root.

**RETURN VALUE**

As in Linux.

**SEE ALSO**

**NAME**

rtapi\_module\_param – Specifying module parameters

**SYNTAX**

RTAPI\_MP\_INT(*var*, *description*)

RTAPI\_MP\_LONG(*var*, *description*)

RTAPI\_MP\_STRING(*var*, *description*)

RTAPI\_MP\_ARRAY\_INT(*var*, *num*, *description*)

RTAPI\_MP\_ARRAY\_LONG(*var*, *num*, *description*)

RTAPI\_MP\_ARRAY\_STRING(*var*, *num*, *description*)

MODULE\_LICENSE(*license*)

MODULE\_AUTHOR(*author*)

MODULE\_DESCRIPTION(*description*)

EXPORT\_FUNCTION(*function*)

**ARGUMENTS**

*var*        The variable where the parameter should be stored

*description*

A short description of the parameter or module

*num*        The maximum number of values for an array parameter

*license*    The license of the module, for instance "GPL"

*author*    The author of the module

*function*

The pointer to the function to be exported

**DESCRIPTION**

These macros are portable ways to declare kernel module parameters. They must be used in the global scope, and are not followed by a terminating semicolon. They must be used after the associated variable or function has been defined.

**NOTES**

EXPORT\_FUNCTION makes a symbol available for use by a subsequently loaded component. It is unrelated to hal functions, which are described in hal\_export\_func(3hal)

**Interpretation of license strings**

**MODULE\_LICENSE** follows the kernel's definition of license strings. Notably, "GPL" indicates "GNU Public License v2 or later". (emphasis ours).

**"GPL"**

GNU Public License v2 or later

**"GPL v2"**

GNU Public License v2

**"GPL and additional rights"**

GNU Public License v2 rights and more

**"Dual BSD/GPL"**

GNU Public License v2 or BSD license choice

**"Dual MIT/GPL"**

GNU Public License v2 or MIT license choice

**"Dual MPL/GPL"**

GNU Public License v2 or Mozilla license choice

**"Proprietary"**

Non-free products

It is still good practice to include a license block which indicates the author, copyright date, and disclaimer of warranty as recommended by the GNU GPL.

**REALTIME CONSIDERATIONS**

Not available in userspace code.

**NAME**

rtapi\_mutex – Mutex-related functions

**SYNTAX**

```
int rtapi_mutex_try(unsigned long *mutex)
```

```
void rtapi_mutex_get(unsigned long *mutex)
```

```
void rtapi_mutex_give(unsigned long *mutex)
```

**ARGUMENTS**

*mutex*    A pointer to the mutex.

**DESCRIPTION**

**rtapi\_mutex\_try** makes a non-blocking attempt to get the mutex. If the mutex is available, it returns 0, and the mutex is no longer available. Otherwise, it returns a nonzero value.

**rtapi\_mutex\_get** blocks until the mutex is available.

**rtapi\_mutex\_give** releases a mutex acquired by **rtapi\_mutex\_try** or **rtapi\_mutex\_get**.

**REALTIME CONSIDERATIONS**

**rtapi\_mutex\_give** and **rtapi\_mutex\_try** may be used from user, init/cleanup, and realtime code.

**rtapi\_mutex\_get** may not be used from realtime code.

**RETURN VALUE**

**rtapi\_mutex\_try** returns 0 for if the mutex was claimed, and nonzero otherwise.

**rtapi\_mutex\_get** and **rtapi\_mutex\_gif** have no return value.

**NAME**

rtapi\_outb, rtapi\_inb – Perform hardware I/O

**SYNTAX**

void rtapi\_outb(unsigned char *byte*, unsigned int *port*)

unsigned char rtapi\_inb(unsigned int *port*)

**ARGUMENTS**

*port*      The address of the I/O port

*byte*      The byte to be written to the port

**DESCRIPTION**

**rtapi\_outb** writes a byte to a hardware I/O port. **rtapi\_inb** reads a byte from a hardware I/O port.

**REALTIME CONSIDERATIONS**

May be called from init/cleanup code and from within realtime tasks. Not available in userspace components.

**RETURN VALUE**

**rtapi\_inb** returns the byte read from the given I/O port

**NOTES**

The I/O address should be within a region previously allocated by **rtapi\_request\_region**. Otherwise, another real-time module or the Linux kernel might attempt to access the I/O region at the same time.

**SEE ALSO**

**rtapi\_region(3rtapi)**

**NAME**

rtapi\_parport – portable access to PC-style parallel ports

**SYNTAX**

```
#include "rtapi_parport.h"
```

```
int rtapi_parport_get(const char *module_name, rtapi_parport_t *port, unsigned short base, unsigned short base_hi, unsigned int modes)
```

```
void rtapi_parport_release(rtapi_parport_t *port)
```

**ARGUMENTS**

*module\_name*

By convention, the name of the RTAPI module or HAL component using the parport.

*port* A pointer to a rtapi\_parport\_t structure

*base* The base address of the port (if port >= 16) or the linux port number of the port (if port < 16)

*base\_hi*

The "high" address of the port (location of the ECP registers), 0 to use a probed high address, or -1 to disable the high address

*modes* Advise the driver of the desired port modes, from <linux/parport.h>. If a linux-detected port does not provide the requested modes, a warning is printed with rtapi\_print\_msg. This does not make the port request fail, because unfortunately, many systems that have working EPP parports are not detected as such by Linux.

**DESCRIPTION**

**rtapi\_parport\_get** allocates a parallel port for exclusive use of the named hal component. If successful, access the port with I/O calls such as rtapi\_inb at address based at the **base** or **base\_hi** addresses. The port must be released with **rtapi\_parport\_release** before the component exits with **rtapi\_exit**.

**HIGH ADDRESS PROBING**

If the port is a parallel port known to Linux, and Linux detected a high I/O address, this value is used. Otherwise, if base+0x400 is not registered to any device, it is used. Otherwise, no address is used. If no high address is detected, port->base\_hi is 0.

**PARPORT STRUCTURE**

```
typedef struct
{
    unsigned short base;
    unsigned short base_hi;
    .... // and further unspecified fields
} rtapi_parport_t;
```

**RETURN VALUE**

**rtapi\_parport\_get** returns a HAL status code. On success, *port* is filled out with information about the allocated port. On failure, the contents of *port* are undefined except that it is safe (but not required) to pass this port to **rtapi\_parport\_release**.

**rtapi\_parport\_release** does not return a value. It always succeeds.

**NOTES**

In new code, prefer use of rtapi\_parport to rtapi\_parport.

**NAME**

rtapi\_pci.h – RTAPI wrappers for linux kernel functionality

**SYNTAX**

```
#include <rtapi_pci.h>

struct rtapi_pci_device_id { ... };
struct rtapi_pci_resource { ... };
struct rtapi_pci_dev { ... };
struct rtapi_pci_driver { ... };

const char *rtapi_pci_name(const struct rtapi_pci_dev *pdev);
int rtapi_pci_enable_device(struct rtapi_pci_dev *dev);
void rtapi__iomem *rtapi_pci_ioremap_bar(struct rtapi_pci_dev *pdev, int bar);
int rtapi_pci_register_driver(struct rtapi_pci_driver *driver);
void rtapi_pci_unregister_driver(struct rtapi_pci_driver *driver);
int rtapi_pci_enable_device(struct rtapi_pci_dev *dev);
int rtapi_pci_disable_device(struct rtapi_pci_dev *dev);
#define rtapi_pci_resource_start(dev, bar) ...
#define rtapi_pci_resource_end(dev, bar) ...
#define rtapi_pci_resource_flags(dev, bar) ...
#define rtapi_pci_resource_len(dev, bar) ...
void rtapi_pci_set_drvdata(struct rtapi_pci_dev *pdev, void *data)
void rtapi_pci_set_drvdata(struct rtapi_pci_dev *pdev, void *data)
void rtapi_iounmap(volatile void *addr);
struct rtapi_pci;
```

**DESCRIPTION**

In kernel space, each `rtapi_xxx` or `RTAPI_XXX` identifier is mapped to the underlying kernel functionality, if available.

In userspace, or in kernels where the underlying functionality is not provided by a kernel, generally another implementation--possibly with reduced functionality--is provided. (For example, the userspace implementation for `rtapi_pci_register` always succeeds)

**REALTIME CONSIDERATIONS**

Typically, these functions may be called from realtime init/cleanup code.

**RETURN VALUE**

As in Linux.

**SEE ALSO**

**NAME**

rtapi\_print, rtapi\_print\_msg – print diagnostic messages

**SYNTAX**

```
void rtapi_print(const char *fmt, ...)
```

```
void rtapi_print_msg(int level, const char *fmt, ...)
```

```
typedef void(*rtapi_msg_handler_t)(msg_level_t level, const char *msg);
```

```
void rtapi_set_msg_handler(rtapi_msg_handler_t handler);
```

```
rtapi_msg_handler_t rtapi_set_msg_handler(void);
```

**ARGUMENTS**

*level* A message level: One of **RTAPI\_MSG\_ERR**, **RTAPI\_MSG\_WARN**, **RTAPI\_MSG\_INFO**, or **RTAPI\_MSG\_DBG**.

*handler* A function to call from **rtapi\_print** or **rtapi\_print\_msg** to actually output the message.

*fmt*  
... Other arguments are as for *printf(3)*.

**DESCRIPTION**

**rtapi\_print** and **rtapi\_print\_msg** work like the standard C *printf* functions, except that a reduced set of formatting operations are supported.

Depending on the RTOS, the default may be to print the message to stdout, stderr, a kernel log, etc. In RTAPI code, the action may be changed by a call to **rtapi\_set\_msg\_handler**. A **NULL** argument to **rtapi\_set\_msg\_handler** restores the default handler. **rtapi\_msg\_get\_handler** returns the current handler. When the message came from **rtapi\_print**, *level* is **RTAPI\_MSG\_ALL**.

**rtapi\_print\_msg** works like **rtapi\_print** but only prints if *level* is less than or equal to the current message level.

**REALTIME CONSIDERATIONS**

**rtapi\_print** and **rtapi\_print\_msg** May be called from user, init/cleanup, and realtime code. **rtapi\_get\_msg\_handler** and **rtapi\_set\_msg\_handler** may be called from realtime init/cleanup code. A message handler passed to **rtapi\_set\_msg\_handler** may only call functions that can be called from real-time code.

**RETURN VALUE**

None.

**SEE ALSO**

**rtapi\_set\_msg\_level(3rtapi), rtapi\_get\_msg\_level(3rtapi), printf(3)**

**NAME**

rtapi\_prio – thread priority functions

**SYNTAX**

int rtapi\_prio\_highest()

int rtapi\_prio\_lowest()

int rtapi\_prio\_next\_higher(int *prio*)

int rtapi\_prio\_next\_lower(int *prio*)

**ARGUMENTS**

*prio*     A value returned by a prior **rtapi\_prio\_xxx** call

**DESCRIPTION**

The **rtapi\_prio\_xxxx** functions provide a portable way to set task priority. The mapping of actual priority to priority number depends on the RTOS. Priorities range from **rtapi\_prio\_lowest** to **rtapi\_prio\_highest**, inclusive. To use this API, use one of two methods:

- 1)     Set your lowest priority task to **rtapi\_prio\_lowest**, and for each task of the next lowest priority, set their priorities to **rtapi\_prio\_next\_higher(previous)**.
- 2)     Set your highest priority task to **rtapi\_prio\_highest**, and for each task of the next highest priority, set their priorities to **rtapi\_prio\_next\_lower(previous)**.

N.B. A high priority task will pre-empt or interrupt a lower priority task. Linux is always the lowest priority!

**REALTIME CONSIDERATIONS**

Call these functions only from within init/cleanup code, not from realtime tasks.

**RETURN VALUE**

Returns an opaque real-time priority number.

**SEE ALSO**

**rtapi\_task\_new(3rtapi)**

**NAME**

rtapi\_region – functions to manage I/O memory regions

**SYNTAX**

```
void *rtapi_request_region(unsigned long base, unsigned long int size, const char *name)
```

```
void rtapi_release_region(unsigned long base, unsigned long int size)
```

**ARGUMENTS**

*base*     The base address of the I/O region

*size*     The size of the I/O region

*name*     The name to be shown in /proc/ioprocs

**DESCRIPTION**

**rtapi\_request\_region** reserves I/O memory starting at *base* and going for *size* bytes.

**REALTIME CONSIDERATIONS**

May be called from realtime init/cleanup code only.

**BUGS**

On kernels before 2.4.0, **rtapi\_request\_region** always succeeds.

**RETURN VALUE**

**rtapi\_request\_region** returns NULL if the allocation fails, and a non-NULL value otherwise.

**rtapi\_release\_region** has no return value.

**NAME**

rtapi\_get\_msg\_level, rtapi\_set\_msg\_level – Get or set the logging level

**SYNTAX**

```
int rtapi_set_msg_level(int level)
```

```
int rtapi_get_msg_level()
```

**ARGUMENTS**

*level*     The desired logging level

**DESCRIPTION**

Get or set the RTAPI message level used by **rtapi\_print\_msg**. Depending on the RTOS, this level may apply to a single RTAPI module, or it may apply to a group of modules.

**REALTIME CONSIDERATIONS**

May be called from user, init/cleanup, and realtime code.

**RETURN VALUE**

**rtapi\_set\_msg\_level** returns a status code, and **rtapi\_get\_msg\_level** returns the current level.

**SEE ALSO**

**rtapi\_print\_msg(3rtapi)**

## NAME

rtapi\_shmem – Functions for managing shared memory blocks

## SYNTAX

```
int rtapi_shmem_new(int key, int module_id, unsigned long int size)
```

```
int rtapi_shmem_delete(int shmem_id, int module_id)
```

```
int rtapi_shmem_getptr(int shmem_id, void ** ptr)
```

## ARGUMENTS

*key*        Identifies the memory block. Key must be nonzero. All modules wishing to use the same memory must use the same key.

*module\_id*  
Module identifier returned by a prior call to **rtapi\_init**.

*size*        The desired size of the shared memory block, in bytes

*ptr*        The pointer to the shared memory block. Note that the block may be mapped at a different address for different modules.

## DESCRIPTION

**rtapi\_shmem\_new** allocates a block of shared memory. *key* identifies the memory block, and must be non-zero. All modules wishing to access the same memory must use the same key. *module\_id* is the ID of the module that is making the call (see **rtapi\_init**). The block will be at least *size* bytes, and may be rounded up. Allocating many small blocks may be very wasteful. When a particular block is allocated for the first time, the first 4 bytes are zeroed. Subsequent allocations of the same block by other modules or processes will not touch the contents of the block. Applications can use those bytes to see if they need to initialize the block, or if another module already did so. On success, it returns a positive integer ID, which is used for all subsequent calls dealing with the block. On failure it returns a negative error code.

**rtapi\_shmem\_delete** frees the shared memory block associated with *shmem\_id*. *module\_id* is the ID of the calling module. Returns a status code.

**rtapi\_shmem\_getptr** sets *\*ptr* to point to shared memory block associated with *shmem\_id*.

## REALTIME CONSIDERATIONS

**rtapi\_shmem\_getptr** may be called from user code, init/cleanup code, or realtime tasks.

**rtapi\_shmem\_new** and **rtapi\_shmem\_dete** may not be called from realtime tasks.

## RETURN VALUE

**NAME**

rtapi\_slab.h – RTAPI wrappers for linux kernel functionality

**SYNTAX**

```
#include <rtapi_slab.h>

void *rtapi_kmalloc(size_t size, gfp_t g);
void *rtapi_kzalloc(size_t size, gfp_t g);
void *rtapi_krealloc(size_t size, gfp_t g);
void rtapi_kfree(void *);
```

**DESCRIPTION**

In kernel space, each `rtapi_xxx` or `RTAPI_XXX` identifier is mapped to the underlying kernel functionality, if available.

In userspace, or in kernels where the underlying functionality is not provided by a kernel, generally another implementation--possibly with reduced functionality--is provided. (For example, the userspace implementation for `rtapi_device_register` always succeeds)

**REALTIME CONSIDERATIONS**

Call only from within init/cleanup code, not from realtime tasks. This function is not available from user (non-realtime) code.

**RETURN VALUE**

As in Linux.

**SEE ALSO**

**NAME**

rtapi\_snprintf, rtapi\_vsnprintf – Perform snprintf-like string formatting

**SYNTAX**

```
int rtapi_snprintf(char *buf, unsigned long int size, const char *fmt, ...)
```

```
int rtapi_vsnprintf(char *buf, unsigned long int size, const char *fmt, va_list apfB)
```

**ARGUMENTS**

As for *snprintf(3)* or *vsnprintf(3)*.

**DESCRIPTION**

These functions work like the standard C printf functions, except that a reduced set of formatting operations are supported.

In particular: formatting of long long values is not supported. Formatting of floating-point values is done as though with %A even when other formats like %f are specified.

**REALTIME CONSIDERATIONS**

May be called from user, init/cleanup, and realtime code.

**RETURN VALUE**

The number of characters written to *buf*.

**SEE ALSO**

**printf(3)**

**NAME**

rtapi\_stdint.h – RTAPI wrappers for linux kernel functionality

**SYNTAX**

```
#include <rtapi_stdint.h>

typedef ... rtapi_s8;
typedef ... rtapi_s16;
typedef ... rtapi_s32;
typedef ... rtapi_s64;
typedef ... rtapi_u8;
typedef ... rtapi_u16;
typedef ... rtapi_u32;
typedef ... rtapi_u64;
```

**DESCRIPTION**

In kernel space, each `rtapi_xxx` or `RTAPI_XXX` identifier is mapped to the underlying kernel functionality, if available.

In userspace, or in kernels where the underlying functionality is not provided by a kernel, generally another implementation--possibly with reduced functionality--is provided. (For example, the userspace implementation for `rtapi_device_register` always succeeds)

**REALTIME CONSIDERATIONS**

None.

**RETURN VALUE**

As in Linux.

**SEE ALSO**

**NAME**

rtapi\_string.h – RTAPI wrappers for linux kernel functionality

**SYNTAX**

```
#include <rtapi_string.h>

char **rtapi_argv_split(rtapi_gfp_t g, const char *argstr, int *argc);

void rtapi_argv_free(char **argv);

char *rtapi_kstrdup(const char *s, rtapi_gfp_t t);
```

**DESCRIPTION**

In kernel space, each `rtapi_xxx` or `RTAPI_XXX` identifier is mapped to the underlying kernel functionality, if available.

In userspace, or in kernels where the underlying functionality is not provided by a kernel, generally another implementation--possibly with reduced functionality--is provided. (For example, the userspace implementation for `rtapi_device_register` always succeeds)

**REALTIME CONSIDERATIONS**

Call only from within init/cleanup code, not from realtime tasks. This function is not available from user (non-realtime) code.

**RETURN VALUE**

As in Linux.

**SEE ALSO**

**NAME**

rtapi\_task\_new – create a realtime task

**SYNTAX**

```
int rtapi_task_new(void (*taskcode)(void*), void *arg, int prio, unsigned long stacksize, int
    uses_fp)
int rtapi_task_delete(int task_id)
```

**ARGUMENTS**

*taskcode*

A pointer to the function to be called when the task is started

*arg*

An argument to be passed to the *taskcode* function when the task is started

*prio*

A task priority value returned by **rtapi\_prio\_xxxx**

*uses\_fp*

A flag that tells the OS whether the task uses floating point or not.

*task\_id*

A task ID returned by a previous call to **rtapi\_task\_new**

**DESCRIPTION**

**rtapi\_task\_new** creates but does not start a realtime task. The task is created in the "paused" state. To start it, call either **rtapi\_task\_start** for periodic tasks, or **rtapi\_task\_resume** for free-running tasks.

**REALTIME CONSIDERATIONS**

Call only from within init/cleanup code, not from realtime tasks.

**RETURN VALUE**

On success, returns a positive integer task ID. This ID is used for all subsequent calls that need to act on the task. On failure, returns an RTAPI status code.

**SEE ALSO**

**rtapi\_prio(3rtapi)**, **rtapi\_task\_start(3rtapi)**, **rtapi\_task\_wait(3rtapi)**, **rtapi\_task\_resume(3rtapi)**

**NAME**

rtapi\_task\_pause, rtapi\_task\_resume – pause and resume real-time tasks

**SYNTAX**

```
void rtapi_task_pause(int task_id)
```

```
void rtapi_task_resume(int task_id)
```

**ARGUMENTS**

*task\_id* An RTAPI task identifier returned by an earlier call to **rtapi\_task\_new**.

**DESCRIPTION**

**rtapi\_task\_resume** starts a task in free-running mode. The task must be in the "paused" state.

A free running task runs continuously until either:

- 1) It is preempted by a higher priority task. It will resume as soon as the higher priority task releases the CPU.
- 2) It calls a blocking function, like **rtapi\_sem\_take**. It will resume when the function unblocks.
- 3) It is returned to the "paused" state by **rtapi\_task\_pause**. May be called from init/cleanup code, and from within realtime tasks.

**rtapi\_task\_pause** causes a task to stop execution and change to the "paused" state. The task can be free-running or periodic. Note that **rtapi\_task\_pause** may be called from any task, or from init or cleanup code, not just from the task that is to be paused. The task will resume execution when either **rtapi\_task\_resume** or **rtapi\_task\_start** (depending on whether this is a free-running or periodic task) is called.

**REALTIME CONSIDERATIONS**

May be called from init/cleanup code, and from within realtime tasks.

**RETURN VALUE**

An RTAPI status code.

**SEE ALSO**

**rtapi\_task\_new(3rtapi)**, **rtapi\_task\_start(3rtapi)**

**NAME**

rtapi\_task\_self – Retrieve ID of current task

**SYNTAX**

void rtapi\_task\_self()

**DESCRIPTION**

**rtapi\_task\_self** retrieves the current task, or -EINVAL if not in a realtime task (e.g., in startup or shutdown code).

**REALTIME CONSIDERATIONS**

May be called from init/cleanup code, and from within realtime tasks.

**RETURN VALUE**

The task number previously returned by **rtapi\_task\_new** or -EINVAL.

**SEE ALSO**

**rtapi\_task\_new(3rtapi)**

**NAME**

rtapi\_task\_start – start a realtime task in periodic mode

**SYNTAX**

int rtapi\_task\_start(int *task\_id*, unsigned long *period\_nsec*)

**ARGUMENTS**

*task\_id* A task ID returned by a previous call to **rtapi\_task\_new**

*period\_nsec*

The clock period in nanoseconds between iterations of a periodic task

**DESCRIPTION**

**rtapi\_task\_start** starts a task in periodic mode. The task must be in the *paused* state.

**REALTIME CONSIDERATIONS**

Call only from within init/cleanup code, not from realtime tasks.

**RETURN VALUE**

Returns an RTAPI status code.

**SEE ALSO**

**rtapi\_task\_new(3rtapi)**, **rtapi\_task\_pause(3rtapi)**, **rtapi\_task\_resume(3rtapi)**

**NAME**

rtapi\_task\_wait – suspend execution of this periodic task

**SYNTAX**

void rtapi\_task\_wait()

**DESCRIPTION**

**rtapi\_task\_wait** suspends execution of the current task until the next period. The task must be periodic. If not, the result is undefined.

**REALTIME CONSIDERATIONS**

Call only from within a periodic realtime task

**RETURN VALUE**

None

**SEE ALSO**

**rtapi\_task\_start(3rtapi)**, **rtapi\_task\_pause(3rtapi)**

**NAME**

undocumented – undocumented functions in RTAPI

**SEE ALSO**

The header file *rtapi.h*. Most rtapi functions have documentation in that file.

**NAME**

abs – Compute the absolute value and sign of the input signal

**SYNOPSIS**

**loadrt abs [count=*N* | names=*name1* [, *name2* ...]]**

**FUNCTIONS**

**abs.*N*** (requires a floating-point thread)

**PINS**

**abs.*N*.in** float in

Analog input value

**abs.*N*.out** float out

Analog output value, always positive

**abs.*N*.sign** bit out

Sign of input, false for positive, true for negative

**abs.*N*.is-positive** bit out

TRUE if input is positive, FALSE if input is 0 or negative

**abs.*N*.is-negative** bit out

TRUE if input is negative, FALSE if input is 0 or positive

**LICENSE**

GPL

**NAME**

abs\_s32 – Compute the absolute value and sign of the input signal

**SYNOPSIS**

**loadrt abs\_s32 [count=*N*]names=*name1*[,*name2*...]**

**FUNCTIONS**

**abs-s32.*N***

**PINS**

**abs-s32.*N*.in** s32 in  
input value

**abs-s32.*N*.out** s32 out  
output value, always non-negative

**abs-s32.*N*.sign** bit out  
Sign of input, false for positive, true for negative

**abs-s32.*N*.is-positive** bit out  
TRUE if input is positive, FALSE if input is 0 or negative

**abs-s32.*N*.is-negative** bit out  
TRUE if input is negative, FALSE if input is 0 or positive

**LICENSE**

GPL

**NAME**

and2 – Two-input AND gate

**SYNOPSIS**

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

**FUNCTIONS**

**and2.*N***

**PINS**

**and2.*N*.in0** bit in

**and2.*N*.in1** bit in

**and2.*N*.out** bit out

**out** is computed from the value of **in0** and **in1** according to the following rule:

**in0=TRUE in1=TRUE**

**out=TRUE**

Otherwise,

**out=FALSE**

**LICENSE**

GPL

**NAME**

`at_pid` – proportional/integral/derivative controller with auto tuning

**SYNOPSIS**

```
loadrt at_pid [num_chan=num | names=name1[,name2...]]
```

**DESCRIPTION**

**at\_pid** is a classic Proportional/Integral/Derivative controller, used to control position or speed feedback loops for servo motors and other closed-loop applications.

**at\_pid** supports a maximum of sixteen controllers. The number that are actually loaded is set by the **num\_chan** argument when the module is loaded. Alternatively, specify **names=** and unique names separated by commas.

The **num\_chan=** and **names=** specifiers are mutually exclusive. If neither **num\_chan=** nor **names=** are specified, the default value is three.

If **debug** is set to 1 (the default is 0), some additional HAL parameters will be exported, which might be useful for tuning, but are otherwise unnecessary.

**at\_pid** has a built in auto tune mode. It works by setting up a limit cycle to characterize the process. From this, **Pgain/Igain/Dgain** or **Pgain/Igain/FF1** can be determined using Ziegler-Nichols. When using **FF1**, scaling must be set so that **output** is in user units per second.

During auto tuning, the **command** input should not change. The limit cycle is setup around the commanded position. No initial tuning values are required to start auto tuning. Only **tune-cycles**, **tune-effort** and **tune-mode** need be set before starting auto tuning. When auto tuning completes, the tuning parameters will be set. If running from LinuxCNC, the **FERROR** setting for the axis being tuned may need to be loosened up as it must be larger than the limit cycle amplitude in order to avoid a following error.

To perform auto tuning, take the following steps. Move the axis to be tuned, to somewhere near the center of it's travel. Set **tune-cycles** (the default value should be fine in most cases) and **tune-mode**. Set **tune-effort** to a small value. Set **enable** to true. Set **tune-mode** to true. Set **tune-start** to true. If no oscillation occurs, or the oscillation is too small, slowly increase **tune-effort**. Auto tuning can be aborted at any time by setting **enable** or **tune-mode** to false.

**NAMING**

The names for pins, parameters, and functions are prefixed as:

**pid.N.** for N=0,1,...,num-1 when using **num\_chan=num**

**nameN.** for nameN=name1,name2,... when using **names=name1,name2,...**

The **pid.N.** format is shown in the following descriptions.

**FUNCTIONS**

**pid.N.do-pid-calcs** (uses floating-point)

Does the PID calculations for control loop *N*.

**PINS**

**pid.N.command** float in

The desired (commanded) value for the control loop.

**pid.N.feedback** float in

The actual (feedback) value, from some sensor such as an encoder.

**pid.N.error** float out

The difference between command and feedback.

**pid.N.output** float out

The output of the PID loop, which goes to some actuator such as a motor.

**pid.N.enable** bit in

When true, enables the PID calculations. When false, **output** is zero, and all internal integrators, etc, are reset.

**pid.N.tune-mode** bit in

When true, enables auto tune mode. When false, normal PID calculations are performed.

**pid.N.tune-start** bit io

When set to true, starts auto tuning. Cleared when the auto tuning completes.

## PARAMETERS

**pid.N.Pgain** float rw

Proportional gain. Results in a contribution to the output that is the error multiplied by **Pgain**.

**pid.N.Igain** float rw

Integral gain. Results in a contribution to the output that is the integral of the error multiplied by **Igain**. For example an error of 0.02 that lasted 10 seconds would result in an integrated error (**errorI**) of 0.2, and if **Igain** is 20, the integral term would add 4.0 to the output.

**pid.N.Dgain** float rw

Derivative gain. Results in a contribution to the output that is the rate of change (derivative) of the error multiplied by **Dgain**. For example an error that changed from 0.02 to 0.03 over 0.2 seconds would result in an error derivative (**errorD**) of 0.05, and if **Dgain** is 5, the derivative term would add 0.25 to the output.

**pid.N.bias** float rw

**bias** is a constant amount that is added to the output. In most cases it should be left at zero. However, it can sometimes be useful to compensate for offsets in servo amplifiers, or to balance the weight of an object that moves vertically. **bias** is turned off when the PID loop is disabled, just like all other components of the output. If a non-zero output is needed even when the PID loop is disabled, it should be added with an external HAL sum2 block.

**pid.N.FF0** float rw

Zero order feed-forward term. Produces a contribution to the output that is **FF0** multiplied by the commanded value. For position loops, it should usually be left at zero. For velocity loops, **FF0** can compensate for friction or motor counter-EMF and may permit better tuning if used properly.

**pid.N.FF1** float rw

First order feed-forward term. Produces a contribution to the output that **FF1** multiplied by the derivative of the commanded value. For position loops, the contribution is proportional to speed, and can be used to compensate for friction or motor CEMF. For velocity loops, it is proportional to acceleration and can compensate for inertia. In both cases, it can result in better tuning if used properly.

**pid.N.FF2** float rw

Second order feed-forward term. Produces a contribution to the output that is **FF2** multiplied by the second derivative of the commanded value. For position loops, the contribution is proportional to acceleration, and can be used to compensate for inertia. For velocity loops, it should usually be left at zero.

**pid.N.deadband** float rw

Defines a range of "acceptable" error. If the absolute value of **error** is less than **deadband**, it will be treated as if the error is zero. When using feedback devices such as encoders that are inherently quantized, the deadband should be set slightly more than one-half count, to prevent the control loop from hunting back and forth if the command is between two adjacent encoder values. When

the absolute value of the error is greater than the deadband, the deadband value is subtracted from the error before performing the loop calculations, to prevent a step in the transfer function at the edge of the deadband. (See **BUGS**.)

**pid.N.maxoutput** float rw

Output limit. The absolute value of the output will not be permitted to exceed **maxoutput**, unless **maxoutput** is zero. When the output is limited, the error integrator will hold instead of integrating, to prevent windup and overshoot.

**pid.N.maxerror** float rw

Limit on the internal error variable used for P, I, and D. Can be used to prevent high **Pgain** values from generating large outputs under conditions when the error is large (for example, when the command makes a step change). Not normally needed, but can be useful when tuning non-linear systems.

**pid.N.maxerrorD** float rw

Limit on the error derivative. The rate of change of error used by the **Dgain** term will be limited to this value, unless the value is zero. Can be used to limit the effect of **Dgain** and prevent large output spikes due to steps on the command and/or feedback. Not normally needed.

**pid.N.maxerrorI** float rw

Limit on error integrator. The error integrator used by the **Igain** term will be limited to this value, unless it is zero. Can be used to prevent integrator windup and the resulting overshoot during/after sustained errors. Not normally needed.

**pid.N.maxcmdD** float rw

Limit on command derivative. The command derivative used by **FF1** will be limited to this value, unless the value is zero. Can be used to prevent **FF1** from producing large output spikes if there is a step change on the command. Not normally needed.

**pid.N.maxcmdDD** float rw

Limit on command second derivative. The command second derivative used by **FF2** will be limited to this value, unless the value is zero. Can be used to prevent **FF2** from producing large output spikes if there is a step change on the command. Not normally needed.

**pid.N.tune-type** u32 rw

When set to 0, **Pgain/Igain/Dgain** are calculated. When set to 1, **Pgain/Igain/FF1** are calculated.

**pid.N.tune-cycles** u32 rw

Determines the number of cycles to run to characterize the process. **tune-cycles** actually sets the number of half cycles. More cycles results in a more accurate characterization as the average of all cycles is used.

**pid.N.tune-effort** float rw

Determines the effort used in setting up the limit cycle in the process. **tune-effort** should be set to a positive value less than **maxoutput**. Start with something small and work up to a value that results in a good portion of the maximum motor current being used. The smaller the value, the smaller the amplitude of the limit cycle.

**pid.N.errorI** float ro (only if debug=1)

Integral of error. This is the value that is multiplied by **Igain** to produce the Integral term of the output.

**pid.N.errorD** float ro (only if debug=1)

Derivative of error. This is the value that is multiplied by **Dgain** to produce the Derivative term of the output.

**pid.N.commandD** float ro (only if debug=1)

Derivative of command. This is the value that is multiplied by **FF1** to produce the first order feed-forward term of the output.

**pid.N.commandDD** float ro (only if debug=1)

Second derivative of command. This is the value that is multiplied by **FF2** to produce the second order feed-forward term of the output.

**pid.N.ultimate-gain** float ro (only if debug=1)

Determined from process characterization. **ultimate-gain** is the ratio of **tune-effort** to the limit cycle amplitude multiplied by 4.0 divided by Pi. **pid.N.ultimate-period** float ro (only if debug=1)

Determined from process characterization. **ultimate-period** is the period of the limit cycle.

## BUGS

Some people would argue that deadband should be implemented such that error is treated as zero if it is within the deadband, and be unmodified if it is outside the deadband. This was not done because it would cause a step in the transfer function equal to the size of the deadband. People who prefer that behavior are welcome to add a parameter that will change the behavior, or to write their own version of **at\_pid**. However, the default behavior should not be changed.

**NAME**

axistest – Used to allow testing of an axis. Used IN PNCconf

**SYNOPSIS**

**loadrt axistest** [**count**=*N* | **names**=*name1* [, *name2* ...]]

**FUNCTIONS**

**axistest.N.update** (requires a floating-point thread)

**PINS**

**axistest.N.jog-minus** bit in

Drive TRUE to jog the axis in its minus direction

**axistest.N.jog-plus** bit in

Drive TRUE to jog the axis in its positive direction

**axistest.N.run** bit in

Drive TRUE to run the axis near its current position\_fb with a trapezoidal velocity profile

**axistest.N.maxvel** float in

Maximum velocity

**axistest.N.amplitude** float in

Approximate amplitude of positions to command during 'run'

**axistest.N.dir** s32 in

Direction from central point to test: 0 = both, 1 = positive, 2 = negative

**axistest.N.position-cmd** float out

**axistest.N.position-fb** float in

**axistest.N.running** bit out

**axistest.N.run-target** float out

**axistest.N.run-start** float out

**axistest.N.run-low** float out

**axistest.N.run-high** float out

**axistest.N.pause** s32 in (default: 0)

pause time for each end of run in seconds

**PARAMETERS**

**axistest.N.epsilon** float rw (default: .001)

**axistest.N.elapsed** float r

Current value of the internal timer

**LICENSE**

GPL

**NAME**

bin2gray – convert a number to the gray-code representation

**SYNOPSIS**

**loadrt bin2gray** [**count**=*N* | **names**=*name1* [, *name2* ... ]]

**DESCRIPTION**

Converts a number into gray-code

**FUNCTIONS**

**bin2gray.N**

**PINS**

**bin2gray.N.in** u32 in  
binary code in

**bin2gray.N.out** u32 out  
gray code out

**AUTHOR**

andy pugh

**LICENSE**

GPL

**NAME**

biquad – Biquad IIR filter

**SYNOPSIS**

**loadrt biquad** [**count**=*N* | **names**=*name1* [, *name2* ...]]

**DESCRIPTION**

Biquad IIR filter. Implements the following transfer function:  $H(z) = (n_0 + n_1z^{-1} + n_2z^{-2}) / (1 + d_1z^{-1} + d_2z^{-2})$

**FUNCTIONS**

**biquad.N** (requires a floating-point thread)

**PINS**

**biquad.N.in** float in  
Filter input.

**biquad.N.out** float out  
Filter output.

**biquad.N.enable** bit in (default: 0)  
Filter enable. When false, the in is passed to out without any filtering. A transition from false to true causes filter coefficients to be calculated according to parameters

**biquad.N.valid** bit out (default: 0)  
When false, indicates an error occurred when calculating filter coefficients.

**PARAMETERS**

**biquad.N.type** u32 rw (default: 0)  
Filter type determines the type of filter coefficients calculated. When 0, coefficients must be loaded directly. When 1, a low pass filter is created. When 2, a notch filter is created.

**biquad.N.f0** float rw (default: 250.0)  
The corner frequency of the filter.

**biquad.N.Q** float rw (default: 0.7071)  
The Q of the filter.

**biquad.N.d1** float rw (default: 0.0)  
1st-delayed denominator coef

**biquad.N.d2** float rw (default: 0.0)  
2nd-delayed denominator coef

**biquad.N.n0** float rw (default: 1.0)  
non-delayed numerator coef

**biquad.N.n1** float rw (default: 0.0)  
1st-delayed numerator coef

**biquad.N.n2** float rw (default: 0.0)  
2nd-delayed numerator coef

**biquad.N.s1** float rw (default: 0.0)

**biquad.N.s2** float rw (default: 0.0)

**LICENSE**

GPL

**NAME**

bitslice – Converts an unsigned-32 input into individual bits

**SYNOPSIS**

**loadrt bitslice [count=*N* | names=*name1* [, *name2* ...]] [personality=*P*, *P*, ...]**

**DESCRIPTION**

This component creates individual bit-outputs for each bit of an unsigned-32 input. The number of bits can be limited by the "personality" modparam. The inverse process can be performed by the `weighted_sum` HAL component.

**FUNCTIONS**

**bitslice.*N***

**PINS**

**bitslice.*N*.in** u32 in

The input value

**bitslice.*N*.out-*MM*** bit out (*MM*=00..personality)

**AUTHOR**

Andy Pugh

**LICENSE**

GPL2+

**NAME**

bitwise – Computes various bitwise operations on the two input values

**SYNOPSIS**

**loadrt bitwise** [**count**=*N* | **names**=*name1* [, *name2* ... ]]

**FUNCTIONS**

**bitwise.N**

**PINS**

**bitwise.N.in0** u32 in

First input value

**bitwise.N.in1** u32 in

Second input value

**bitwise.N.out-and** u32 out

The bitwise AND of the two inputs

**bitwise.N.out-or** u32 out

The bitwise OR of the two inputs

**bitwise.N.out-xor** u32 out

The bitwise XOR of the two inputs

**bitwise.N.out-nand** u32 out

The inverse of the bitwise AND

**bitwise.N.out-nor** u32 out

The inverse of the bitwise OR

**bitwise.N.out-xnor** u32 out

The inverse of the bitwise XOR

**AUTHOR**

Andy Pugh

**LICENSE**

GPL 2+

**NAME**

bldc – BLDC and AC-servo control component

**SYNOPSIS**

**loadrt bldc personality=P**

**DESCRIPTION**

This component is designed as an interface between the most common forms of three-phase motor feedback devices and the corresponding types of drive. However there is no requirement that the motor and drive should necessarily be of inherently compatible types.

**SYNOPSIS**

(ignore the auto-generated SYNOPSIS above)

**loadrt bldc cfg=qi6,aH**

Each instance of the component is defined by a group of letters describing the input and output types. A comma separates individual instances of the component.

**Tags**

Input type definitions are all lower-case.

**n** No motor feedback. This mode could be used to drive AC induction motors, but is also potentially useful for creating free-running motor simulators for drive testing.

**h** Hall sensor input. Brushless DC motors (electronically commutated permanent magnet 3-phase motors) typically use a set of three Hall sensors to measure the angular position of the rotor. A lower-case **h** in the **cfg** string indicates that these should be used.

**a** Absolute encoder input. (Also possibly used by some forms of Resolver conversion hardware). The presence of this tag over-rides all other inputs. Note that the component still requires to be connected to the **rawcounts** encoder pin to prevent loss of commutation on index-reset.

**q** Incremental (quadrature) encoder input. If this input is used then the rotor will need to be homed before the motor can be run.

**i** Use the index of an incremental encoder as a home reference.

**f** Use a 4-bit Gray-scale pattern to determine rotor alignment. This scheme is only used on the Fanuc "Red Cap" motors. This mode could be used to control one of these motors using a non-Fanuc drive.

Output type descriptions are all upper-case.

**Defaults** The component will always calculate rotor angle, phase angle and the absolute value of the input **value** for interfacing with drives such as the Mesa 8i20. It will also default to three individual, bipolar phase output values if no other output type modifiers are used.

**B** Bit level outputs. Either 3 or 6 logic-level outputs indicating which high or low gate drivers on an external drive should be used.

**6** Create 6 rather than the default 3 outputs. In the case of numeric value outputs these are separate positive and negative drive amplitudes. Both have positive magnitude.

**H** Emulated Hall sensor output. This mode can be used to control a drive which expects 3x Hall signals, or to convert between a motor with one hall pattern and a drive which expects a different one.

**F** Emulated Fanuc Red Cap Gray-code encoder output. This mode might be used to drive a non-Fanuc

motor using a Fanuc drive intended for the "Red-Cap" motors.

**T** Force Trapezoidal mode.

## OPERATING MODES

The component can control a drive in either Trapezoidal or Sinusoidal mode, but will always default to sinusoidal if the input and output modes allow it. This can be over-ridden by the **T** tag. Sinusoidal commutation is significantly smoother (trapezoidal commutation induces 13% torque ripple).

## ROTOR HOMING.

To use an encoder for commutation a reference 0-degrees point must be found. The component uses the convention that motor zero is the point that an unloaded motor aligns to with a positive voltage on the A (or U) terminal and the B & C (or V and W) terminals connected together and to -ve voltage. There will be two such positions on a 4-pole motor, 3 on a 6-pole and so on. They are all functionally equivalent as far as driving the motor is concerned. If the motor has Hall sensors then the motor can be started in trapezoidal commutation mode, and will switch to sinusoidal commutation when an alignment is found. If the mode is **qh** then the first Hall state-transition will be used. If the mode is **qhi** then the encoder index will be used. This gives a more accurate homing position if the distance in encoder counts between motor zero and encoder index is known. To force homing to the Hall edges instead simply omit the **i**.

Motors without Hall sensors may be homed in synchronous/direct mode. The better of these options is to home to the encoder zero using the **iq** config parameter. When the **init** pin goes high the motor will rotate (in a direction determined by the **rev** pin) until the encoder indicates an index-latch (the servo thread runs too slowly to rely on detecting an encoder index directly). If there is no encoder index or its location relative to motor zero can not be found, then an alternative is to use *magnetic* homing using the **q** config. In this mode the motor will go through an alignment sequence ending at motor zero when the init pin goes high It will then set the final position as motor zero. Unfortunately the motor is rather *springy* in this mode and so alignment is likely to be fairly sensitive to load.

## FUNCTIONS

**bldc.N** (requires a floating-point thread)

## PINS

**bldc.N.hall1** bit in [if personality & 0x01]

Hall sensor signal 1

**bldc.N.hall2** bit in [if personality & 0x01]

Hall sensor signal 2

**bldc.N.hall3** bit in [if personality & 0x01]

Hall sensor signal 3

**bldc.N.hall-error** bit out [if personality & 0x01]

Indicates that the selected hall pattern gives inconsistent rotor position data. This can be due to the pattern being wrong for the motor, or one or more sensors being unconnected or broken. A consistent pattern is not necessarily valid, but an inconsistent one can never be valid.

**bldc.N.C1** bit in [if ( personality & 0x10 )]

Fanuc Gray-code bit 0 input

**bldc.N.C2** bit in [if ( personality & 0x10 )]

Fanuc Gray-code bit 1 input

**bldc.N.C4** bit in [if ( personality & 0x10 )]

Fanuc Gray-code bit 2 input

**bldc.N.C8** bit in [if ( personality & 0x10 )]  
Fanuc Gray-code bit 3 input

**bldc.N.value** float in  
PWM master amplitude input

**bldc.N.lead-angle** float in [if personality & 0x06] (default: 90)  
The phase lead between the electrical vector and the rotor position in degrees

**bldc.N.rev** bit in  
Set this pin true to reverse the motor. Negative PWM amplitudes will also reverse the motor and there will generally be a Hall pattern that runs the motor in each direction too.

**bldc.N.frequency** float in [if ( personality & 0x0F ) == 0]  
Frequency input for motors with no feedback at all, or those with only an index (which is ignored)

**bldc.N.initvalue** float in [if personality & 0x04] (default: 0.2)  
The current to be used for the homing sequence in applications where an incremental encoder is used with no hall-sensor feedback

**bldc.N.rawcounts** s32 in [if personality & 0x06] (default: 0)  
Encoder counts input. This must be linked to the encoder rawcounts pin or encoder index resets will cause the motor commutation to fail

**bldc.N.index-enable** bit io [if personality & 0x08]  
This pin should be connected to the associated encoder index-enable pin to zero the encoder when it passes index This is only used indicate to the bldc control component that an index has been seen

**bldc.N.init** bit in [if ( personality & 0x05 ) == 4]  
A rising edge on this pin starts the motor alignment sequence. This pin should be connected in such a way that the motors re-align any time that encoder monitoring has been interrupted. Typically this will only be at machine power-off. The alignment process involves powering the motor phases in such a way as to put the motor in a known position. The encoder counts are then stored in the **offset** parameter. The alignment process will tend to cause a following error if it is triggered while the axis is enabled, so should be set before the matching axis.N.enable pin. The complementary **init-done** pin can be used to handle the required sequencing.

Both pins can be ignored if the encoder offset is known explicitly, such as is the case with an absolute encoder. In that case the **offset** parameter can be set directly in the HAL file

**bldc.N.init-done** bit out [if ( personality & 0x05 ) == 4] (default: 0)  
Indicates homing sequence complete

**bldc.N.A-value** float out [if ( personality & 0xF00 ) == 0]  
Output amplitude for phase A

**bldc.N.B-value** float out [if ( personality & 0xF00 ) == 0]  
Output amplitude for phase B

**bldc.N.C-value** float out [if ( personality & 0xF00 ) == 0]  
Output amplitude for phase C

**bldc.N.A-on** bit out [if ( personality & 0xF00 ) == 0x100]  
Output bit for phase A

**bldc.N.B-on** bit out [if ( personality & 0xF00 ) == 0x100]  
Output bit for phase B

**bldc.N.C-on** bit out [if ( personality & 0xF00 ) == 0x100]  
Output bit for phase C

**bldc.N.A-high** float out [if ( personality & 0xF00 ) == 0x200]  
High-side driver for phase A

**bldc.N.B-high** float out [if ( personality & 0xF00 ) == 0x200]  
High-side driver for phase B

**bldc.N.C-high** float out [if ( personality & 0xF00 ) == 0x200]  
High-side driver for phase C

**bldc.N.A-low** float out [if ( personality & 0xF00 ) == 0x200]  
Low-side driver for phase A

**bldc.N.B-low** float out [if ( personality & 0xF00 ) == 0x200]  
Low-side driver for phase B

**bldc.N.C-low** float out [if ( personality & 0xF00 ) == 0x200]  
Low-side driver for phase C

**bldc.N.A-high-on** bit out [if ( personality & 0xF00 ) == 0x300]  
High-side driver for phase A

**bldc.N.B-high-on** bit out [if ( personality & 0xF00 ) == 0x300]  
High-side driver for phase B

**bldc.N.C-high-on** bit out [if ( personality & 0xF00 ) == 0x300]  
High-side driver for phase C

**bldc.N.A-low-on** bit out [if ( personality & 0xF00 ) == 0x300]  
Low-side driver for phase A

**bldc.N.B-low-on** bit out [if ( personality & 0xF00 ) == 0x300]  
Low-side driver for phase B

**bldc.N.C-low-on** bit out [if ( personality & 0xF00 ) == 0x300]  
Low-side driver for phase C

**bldc.N.hall1-out** bit out [if ( personality & 0x400 )]  
Hall 1 output

**bldc.N.hall2-out** bit out [if ( personality & 0x400 )]  
Hall 2 output

**bldc.N.hall3-out** bit out [if ( personality & 0x400 )]  
Hall 3 output

**bldc.N.C1-out** bit out [if ( personality & 0x800 )]  
Fanuc Gray-code bit 0 output

**bldc.N.C2-out** bit out [if ( personality & 0x800 )]  
Fanuc Gray-code bit 1 output

**bldc.N.C4-out** bit out [if ( personality & 0x800 )]  
Fanuc Gray-code bit 2 output

**bldc.N.C8-out** bit out [if ( personality & 0x800 )]  
Fanuc Gray-code bit 3 output

**bldc.N.phase-angle** float out (default: 0)  
Phase angle including lead/lag angle after encoder zeroing etc. Useful for angle/current drives. This value has a range of 0 to 1 and measures electrical revolutions. It will have two zeros for a 4 pole motor, three for a 6-pole etc

**bldc.N.rotor-angle** float out (default: 0)  
Rotor angle after encoder zeroing etc. Useful for angle/current drives which add their own phase offset such as the 8i20. This value has a range of 0 to 1 and measures electrical revolutions. It will have two zeros for a 4 pole motor, three for a 6-pole etc

**bldc.N.out** float out

Current output, including the effect of the dir pin and the alignment sequence

**bldc.N.out-dir** bit out

Direction output, high if /fBvalue/fR is negative XOR /fBrev/fR is true.

**bldc.N.out-abs** float out

Absolute value of the input value

## PARAMETERS

**bldc.N.in-type** s32 r (default: -1)

state machine output, will probably hide after debug

**bldc.N.out-type** s32 r (default: -1)

state machine output, will probably hide after debug

**bldc.N.scale** s32 rw [if personality & 0x06] (default: 512)

The number of encoder counts per rotor revolution.

**bldc.N.poles** s32 rw [if personality & 0x06] (default: 4)

The number of motor poles. The encoder scale will be divided by this value to determine the number of encoder counts per electrical revolution

**bldc.N.encoder-offset** s32 rw [if personality & 0x0A] (default: 0)

The offset, in encoder counts, between the motor electrical zero and the encoder zero modulo the number of counts per electrical revolution

**bldc.N.offset-measured** s32 r [if personality & 0x04] (default: 0)

The encoder offset measured by the homing sequence (in certain modes)

**bldc.N.drive-offset** float rw (default: 0)

The angle, in degrees, applied to the commanded angle by the drive in degrees. This value is only used during the homing sequence of drives with incremental encoder feedback. It is used to back-calculate from commanded angle to actual phase angle. It is only relevant to drives which expect rotor-angle input rather than phase-angle demand. Should be 0 for most drives.

**bldc.N.output-pattern** u32 rw [if personality & 0x400] (default: 25)

Commutation pattern to be output in Hall Signal translation mode. See the description of /fBpattern/fR for details

**bldc.N.pattern** u32 rw [if personality & 0x01] (default: 25)

Commutation pattern to use, from 0 to 47. Default is type 25. Every plausible combination is included. The table shows the excitation pattern along the top, and the pattern code on the left hand side. The table entries are the hall patterns in H1, H2, H3 order. Common patterns are: 0 (30 degree commutation) and 26, its reverse. 17 (120 degree). 18 (alternate 60 degree). 21 (300 degree, Bodine). 22 (240 degree). 25 (60 degree commutation).

Note that a number of incorrect commutations will have non-zero net torque which might look as if they work, but don't really.

If your motor lacks documentation it might be worth trying every pattern.

Phases, Source - Sink						
pat	B-A	C-A	C-B	A-B	A-C	B-C
0	000	001	011	111	110	100
1	001	000	010	110	111	101
2	000	010	011	111	101	100
3	001	011	010	110	100	101
4	010	011	001	101	100	110
5	011	010	000	100	101	111
6	010	000	001	101	111	110
7	011	001	000	100	110	111
8	000	001	101	111	110	010
9	001	000	100	110	111	011
10	000	010	110	111	101	001
11	001	011	111	110	100	000
12	010	011	111	101	100	000
13	011	010	110	100	101	001
14	010	000	100	101	111	011
15	011	001	101	100	110	010
16	000	100	101	111	011	010
17	001	101	100	110	010	011
18	000	100	110	111	011	001
19	001	101	111	110	010	000
20	010	110	111	101	001	000
21	011	111	110	100	000	001
22	010	110	100	101	001	011
23	011	111	101	100	000	010
24	100	101	111	011	010	000
25	101	100	110	010	011	001
26	100	110	111	011	001	000
27	101	111	110	010	000	001
28	110	111	101	001	000	010
29	111	110	100	000	001	011
30	110	100	101	001	011	010
31	111	101	100	000	010	011
32	100	101	001	011	010	110
33	101	100	000	010	011	111
34	100	110	010	011	001	101
35	101	111	011	010	000	100
36	110	111	011	001	000	100
37	111	110	010	000	001	101
38	110	100	000	001	011	111
39	111	101	001	000	010	110
40	100	000	001	011	111	110
41	101	001	000	010	110	111
42	100	000	010	011	111	101
43	101	001	011	010	110	100
44	110	010	011	001	101	100
45	111	011	010	000	100	101
46	110	010	000	001	101	111
47	111	011	001	000	100	110

**AUTHOR**

Andy Pugh

**LICENSE**

GPL

**NAME**

`bldc_hall3` – 3-wire BLDC motor driver using Hall sensors and trapezoidal commutation.

**SYNOPSIS**

The functionality of this component is now included in the generic "bldc" component. This component is likely to be removed in a future release

**DESCRIPTION**

This component produces a 3-wire bipolar output. This suits upstream drivers that interpret a negative input as a low-side drive and positive as a high-side drive. This includes the Hostmot2 3pwmgen function, which is likely to be the most common application of this component.

**FUNCTIONS**

**bldc-hall3.N** (requires a floating-point thread)  
Interpret Hall sensor patterns and set 3-phase amplitudes

**PINS**

**bldc-hall3.N.hall1** bit in  
Hall sensor signal 1

**bldc-hall3.N.hall2** bit in  
Hall sensor signal 2

**bldc-hall3.N.hall3** bit in  
Hall sensor signal 3

**bldc-hall3.N.value** float in  
PWM master amplitude input

**bldc-hall3.N.dir** bit in  
Forwards / reverse selection. Negative PWM amplitudes will also reverse the motor and there will generally be a pattern that runs the motor in each direction too.

**bldc-hall3.N.A-value** float out  
Output amplitude for phase A

**bldc-hall3.N.B-value** float out  
Output amplitude for phase B

**bldc-hall3.N.C-value** float out  
Output amplitude for phase C

**PARAMETERS**

**bldc-hall3.N.pattern** u32 rw (default: 25)  
Commutation pattern to use, from 0 to 47. Default is type 25. Every plausible combination is included. The table shows the excitation pattern along the top, and the pattern code on the left hand side. The table entries are the hall patterns in H1, H2, H3 order. Common patterns are: 0 (30 degree commutation) and 26, its reverse. 17 (120 degree). 18 (alternate 60 degree). 21 (300 degree, Bodine). 22 (240 degree). 25 (60 degree commutation).

Note that a number of incorrect commutations will have non-zero net torque which might look as if they work, but don't really.

If your motor lacks documentation it might be worth trying every pattern.

Phases, Source - Sink						
pat	B-A	C-A	C-B	A-B	A-C	B-C
0	000	001	011	111	110	100
1	001	000	010	110	111	101
2	000	010	011	111	101	100
3	001	011	010	110	100	101
4	010	011	001	101	100	110
5	011	010	000	100	101	111
6	010	000	001	101	111	110
7	011	001	000	100	110	111
8	000	001	101	111	110	010
9	001	000	100	110	111	011
10	000	010	110	111	101	001
11	001	011	111	110	100	000
12	010	011	111	101	100	000
13	011	010	110	100	101	001
14	010	000	100	101	111	011
15	011	001	101	100	110	010
16	000	100	101	111	011	010
17	001	101	100	110	010	011
18	000	100	110	111	011	001
19	001	101	111	110	010	000
20	010	110	111	101	001	000
21	011	111	110	100	000	001
22	010	110	100	101	001	011
23	011	111	101	100	000	010
24	100	101	111	011	010	000
25	101	100	110	010	011	001
26	100	110	111	011	001	000
27	101	111	110	010	000	001
28	110	111	101	001	000	010
29	111	110	100	000	001	011
30	110	100	101	001	011	010
31	111	101	100	000	010	011
32	100	101	001	011	010	110
33	101	100	000	010	011	111
34	100	110	010	011	001	101
35	101	111	011	010	000	100
36	110	111	011	001	000	100
37	111	110	010	000	001	101
38	110	100	000	001	011	111
39	111	101	001	000	010	110
40	100	000	001	011	111	110
41	101	001	000	010	110	111
42	100	000	010	011	111	101
43	101	001	011	010	110	100
44	110	010	011	001	101	100
45	111	011	010	000	100	101
46	110	010	000	001	101	111
47	111	011	001	000	100	110

**SEE ALSO**

bldc\_hall6 6-wire unipolar driver for BLDC motors.

**AUTHOR**

Andy Pugh

**LICENSE**

GPL

**NAME**

blend – Perform linear interpolation between two values

**SYNOPSIS**

**loadrt blend** [**count**=*N* | **names**=*name1* [, *name2*...]]

**FUNCTIONS**

**blend.N** (requires a floating-point thread)

**PINS**

**blend.N.in1** float in

First input. If select is equal to 1.0, the output is equal to in1

**blend.N.in2** float in

Second input. If select is equal to 0.0, the output is equal to in2

**blend.N.select** float in

Select input. For values between 0.0 and 1.0, the output changes linearly from in2 to in1

**blend.N.out** float out

Output value.

**PARAMETERS**

**blend.N.open** bit rw

If true, select values outside the range 0.0 to 1.0 give values outside the range in2 to in1. If false, outputs are clamped to the the range in2 to in1

**LICENSE**

GPL

**NAME**

carousel – Orient a toolchanger carousel using various encoding schemes

**SYNOPSIS**

**loadrt toolchange pockets=N[,N] encoding=ssss[,sss] num\_sense=N[,N] bidirectional=N[,N]**

**pockets** The number of pockets in each toolchanger.

Use up to 8 numbers separated by commas to create multiple carousel components.

**encoding** The position encoding.

gray, binary, index or single. Default = 'gray'

**num\_sense** The number of position sense pins.

Default = 4.

**dir** Set to 1 for unidirectional or 2 for bidirectional operation.

Default = bidirectional

**DESCRIPTION**

This component is intended to help operate various types of carousel-type toolchangers. The component can be configured to operate with binary or gray-coded position feedback, with an individual sensor for each tool position or with a sensor at each tool position and a separate index.

At the moment it is best-suited to Geneva-mechanism type systems where the motor is either on or off. Both unidirectional and bidirectional systems are supported.

The number of carousel component instances created depends on the number of entries in the 'pockets' modparam. For example

**loadrt carousel pockets=10,10,8**

Would create 3 carousel instances with 10, 10 and 8 pockets. The other parameters are optional. If absent then defaults will be used. Any missing entry will assume the previous value.

When the enable pin is set to true the component will immediately set the "active" pin to true and then (for a bidirectional instance) calculate the shortest path to the requested pocket number. The appropriate motor direction output pin will then be set.

The component will monitor the carousel position and, when the correct position is reached, set the motor-control pins to 0, set "active" to 0 and set "ready" to 1.

In index mode the behaviour is slightly different the first time that the "enable" pin is set; the carousel will rotate forwards when first enabled until both the index and pulse inputs are true. If there is no pulse line at the index position then a HAL "or2" function can be used to allow the index sensor to toggle both inputs. Setting "enable" low does not halt the homing move, so if homing on first tool change is not needed then the enable pin can be toggled by an axis homing pin or a script.

**FUNCTIONS**

**carousel.N**

**PINS**

**carousel.N.pocket-number** s32 in

The pocket to move to when the .enable pin goes high

**carousel.N.enable** bit in

Set this pin high to start movement. Setting it low will stop movement

**carousel.N.active** bit out

indicates that the component is active

**carousel.N.ready** bit out

This pin goes high when the carousel is in-position

**carousel.N.sense-M** bit in (M=0..personality)

Carousel position feedback pins. In 'index' mode there will be only 2 pins. sense-0 is the index and sense-1 is the pocket sensor.

**carousel.N.motor-fwd** bit out

Indicates the motor should run forwards (bigger numbers)

**carousel.N.motor-rev** bit out

Indicates the motor should run reverse.

**carousel.N.current-position** s32 out

This pin indicates the current position feedback

## PARAMETERS

**carousel.N.state** s32 r (default: 0)

Current component state

**carousel.N.homing** bit r (default: 0)

Shows that homing is in progress. Only used for index mode

**carousel.N.homed** bit r (default: 0)

Shows that homing is complete. Only used in index mode

## AUTHOR

andy pugh

## LICENSE

GPL

**NAME**

charge\_pump – Create a square-wave for the 'charge pump' input of some controller boards

**SYNOPSIS**

**loadrt charge\_pump**

**DESCRIPTION**

The 'Charge Pump' should be added to the base thread function. When enabled the output is on for one period and off for one period. To calculate the frequency of the output  $1/(\text{period time in seconds} \times 2) = \text{hz}$ . For example if you have a base period of 100,000ns that is 0.0001 seconds and the formula would be  $1/(0.0001 \times 2) = 5,000 \text{ hz}$  or 5 Khz. Two additional outputs are provided that run a factor of 2 and 4 slower for hardware that requires a lower frequency.

**FUNCTIONS****charge-pump**

Toggle the output bit (if enabled)

**PINS****charge-pump.out** bit out

Square wave if 'enable' is TRUE or unconnected, low if 'enable' is FALSE

**charge-pump.out-2** bit out

Square wave at half the frequency of 'out'

**charge-pump.out-4** bit out

Square wave at a quarter of the frequency of 'out'

**charge-pump.enable** bit in (default: *TRUE*)

If FALSE, forces all 'out' pins to be low

**LICENSE**

GPL

**NAME**

clarke2 – Two input version of Clarke transform

**SYNOPSIS**

**loadrt clarke2** [**count**=*N* | **names**=*name1* [, *name2* ... ]]

**DESCRIPTION**

The Clarke transform can be used to translate a vector quantity from a three phase system (three components 120 degrees apart) to a two phase Cartesian system.

**clarke2** implements a special case of the Clarke transform, which only needs two of the three input phases. In a three wire three phase system, the sum of the three phase currents or voltages must always be zero. As a result only two of the three are needed to completely define the current or voltage. **clarke2** assumes that the sum is zero, so it only uses phases A and B of the input. Since the H (homopolar) output will always be zero in this case, it is not generated.

**FUNCTIONS**

**clarke2.N** (requires a floating-point thread)

**PINS**

**clarke2.N.a** float in

**clarke2.N.b** float in

first two phases of three phase input

**clarke2.N.x** float out

**clarke2.N.y** float out

cartesian components of output

**SEE ALSO**

**clarke3** for the general case, **clarkeinv** for the inverse transform.

**LICENSE**

GPL

**NAME**

clarke3 – Clarke (3 phase to cartesian) transform

**SYNOPSIS**

**loadrt clarke3** [**count**=*N*][**names**=*name1*[,*name2*...]]

**DESCRIPTION**

The Clarke transform can be used to translate a vector quantity from a three phase system (three components 120 degrees apart) to a two phase Cartesian system (plus a homopolar component if the three phases don't sum to zero).

**clarke3** implements the general case of the transform, using all three phases. If the three phases are known to sum to zero, see **clarke2** for a simpler version.

**FUNCTIONS**

**clarke3.N** (requires a floating-point thread)

**PINS**

**clarke3.N.a** float in

**clarke3.N.b** float in

**clarke3.N.c** float in

three phase input vector

**clarke3.N.x** float out

**clarke3.N.y** float out

cartesian components of output

**clarke3.N.h** float out

homopolar component of output

**SEE ALSO**

**clarke2** for the 'a+b+c=0' case, **clarkeinv** for the inverse transform.

**LICENSE**

GPL

**NAME**

clarkeinv – Inverse Clarke transform

**SYNOPSIS**

**loadrt clarkeinv** [**count**=*N*][**names**=*name1*[,*name2*...]]

**DESCRIPTION**

The inverse Clarke transform can be used rotate a vector quantity and then translate it from Cartesian coordinate system to a three phase system (three components 120 degrees apart).

**FUNCTIONS**

**clarkeinv.N** (requires a floating-point thread)

**PINS**

**clarkeinv.N.x** float in

**clarkeinv.N.y** float in

cartesian components of input

**clarkeinv.N.h** float in

homopolar component of input (usually zero)

**clarkeinv.N.theta** float in

rotation angle: 0.00 to 1.00 = 0 to 360 degrees

**clarkeinv.N.a** float out

**clarkeinv.N.b** float out

**clarkeinv.N.c** float out

three phase output vector

**SEE ALSO**

**clarke2** and **clarke3** for the forward transform.

**LICENSE**

GPL

**NAME**

classicladder – realtime software plc based on ladder logic

**SYNOPSIS**

**loadrt classicladder\_rt** [**numRungs**=*N*] [**numBits**=*N*] [**numWords**=*N*] [**numTimers**=*N*] [**numMonostables**=*N*] [**numCounters**=*N*] [**numPhysInputs**=*N*] [**numPhysOutputs**=*N*] [**numArithmExpr**=*N*] [**numSections**=*N*] [**numSymbols**=*N*] [**numS32in**=*N*] [**numS32out**=*N*] [**numFloatIn**=*N*] [**numFloatOut**=*N*]

**DESCRIPTION**

These pins and parameters are created by the realtime **classicladder\_rt** module. Each period (minimum 1000000 ns), classicladder reads the inputs, evaluates the ladder logic defined in the GUI, and then writes the outputs.

**PINS**

**classicladder.0.in-*NN*** IN bit

These bit signal pins map to **%I*NN*** variables in classicladder

**classicladder.0.out-*NN*** OUT bit

These bit signal pins map to **%Q*NN*** variables in classicladder Output from classicladder

**classicladder.0.s32in-*NN*** IN s32

Integer input from classicladder These s32 signal pins map to **%IW*NN*** variables in classicladder

**classicladder.0.s32out-*NN*** OUT s32

Integer output from classicladder These s32 signal pins map to **%QW*NN*** variables in classicladder

**classicladder.0.floatin-*NN*** IN float

Integer input from classicladder These float signal pins map to **%IF*NN*** variables in classicladder These are truncated to S32 values internally. eg 7.5 will be 7

**classicladder.0.floatout-*NN*** OUT float

Float output from classicladder These float signal pins map to **%QF*NN*** variables in classicladder

**classicladder.0.hide\_gui** IN bit

This bit pin hides the classicladder window, while still having the userspace code run. This is usually desirable when modbus is used, as modbus requires the userspace code to run.

**PARAMETERS**

**classicladder.0.refresh.time** RO s32

Tells you how long the last refresh took

**classicladder.0.refresh.tmax** RW s32

Tells you how long the longest refresh took

**classicladder.0.ladder-state** RO s32

Tells you if the program is running or not

**FUNCTIONS**

**classicladder.0.refresh** FP

The rung update rate. Add this to the servo thread. You can added it to a faster thread but it Will update no faster than once every 1 millisecond (1000000 ns).

**BUGS**

See [http://wiki.linuxcnc.org/cgi-bin/wiki.pl?ClassicLadder\\_Ver\\_7.124](http://wiki.linuxcnc.org/cgi-bin/wiki.pl?ClassicLadder_Ver_7.124) for the latest.

**SEE ALSO**

*Classicladder* chapters in the LinuxCNC documentation for a full description of the **Classicladder** syntax and examples

[http://wiki.linuxcnc.org/cgi-bin/wiki.pl?ClassicLadder\\_Ver\\_7.124](http://wiki.linuxcnc.org/cgi-bin/wiki.pl?ClassicLadder_Ver_7.124)

**NAME**

comp – Two input comparator with hysteresis

**SYNOPSIS**

**loadrt comp** [**count**=*N* | **names**=*name1* [, *name2* ...]]

**FUNCTIONS**

**comp.N** (requires a floating-point thread)

Update the comparator

**PINS**

**comp.N.in0** float in

Inverting input to the comparator

**comp.N.in1** float in

Non-inverting input to the comparator

**comp.N.out** bit out

Normal output. True when **in1** > **in0** (see parameter **hyst** for details)

**comp.N.equal** bit out

Match output. True when difference between **in1** and **in0** is less than **hyst**/2

**PARAMETERS**

**comp.N.hyst** float rw (default: 0.0)

Hysteresis of the comparator (default 0.0)

With zero hysteresis, the output is true when **in1** > **in0**. With nonzero hysteresis, the output switches on and off at two different values, separated by distance **hyst** around the point where **in1** = **in0**. Keep in mind that floating point calculations are never absolute and it is wise to always set **hyst** if you intend to use equal

**LICENSE**

GPL

**NAME**

constant – Use a parameter to set the value of a pin

**SYNOPSIS**

**loadrt constant** [**count**=*N* | **names**=*name1* [, *name2* ...]]

**FUNCTIONS**

**constant.N** (requires a floating-point thread)

**PINS**

**constant.N.out** float out

**PARAMETERS**

**constant.N.value** float rw

**LICENSE**

GPL

**NAME**

conv\_bit\_s32 – Convert a value from bit to s32

**SYNOPSIS**

**loadrt conv\_bit\_s32** [**count**=*N*][**names**=*name1*[,*name2*...]]

**FUNCTIONS**

**conv-bit-s32.N**

Update 'out' based on 'in'

**PINS**

**conv-bit-s32.N.in** bit in

**conv-bit-s32.N.out** s32 out

**LICENSE**

GPL

**NAME**

conv\_bit\_u32 – Convert a value from bit to u32

**SYNOPSIS**

**loadrt conv\_bit\_u32** [**count**=*N*]**names=***name1*[,*name2*...]

**FUNCTIONS**

**conv-bit-u32.N**

Update 'out' based on 'in'

**PINS**

**conv-bit-u32.N.in** bit in

**conv-bit-u32.N.out** u32 out

**LICENSE**

GPL

**NAME**

conv\_float\_s32 – Convert a value from float to s32

**SYNOPSIS**

**loadrt conv\_float\_s32** [**count**=*N* | **names**=*name1* [, *name2* ... ]]

**FUNCTIONS**

**conv-float-s32.N** (requires a floating-point thread)

Update 'out' based on 'in'

**PINS**

**conv-float-s32.N.in** float in

**conv-float-s32.N.out** s32 out

**conv-float-s32.N.out-of-range** bit out

TRUE when 'in' is not in the range of s32

**PARAMETERS**

**conv-float-s32.N.clamp** bit rw

If TRUE, then clamp to the range of s32. If FALSE, then allow the value to "wrap around".

**LICENSE**

GPL

**NAME**

conv\_float\_u32 – Convert a value from float to u32

**SYNOPSIS**

**loadrt conv\_float\_u32** [**count**=*N* | **names**=*name1* [, *name2* ...]]

**FUNCTIONS**

**conv-float-u32.N** (requires a floating-point thread)

Update 'out' based on 'in'

**PINS**

**conv-float-u32.N.in** float in

**conv-float-u32.N.out** u32 out

**conv-float-u32.N.out-of-range** bit out

TRUE when 'in' is not in the range of u32

**PARAMETERS**

**conv-float-u32.N.clamp** bit rw

If TRUE, then clamp to the range of u32. If FALSE, then allow the value to "wrap around".

**LICENSE**

GPL

**NAME**

conv\_s32\_bit – Convert a value from s32 to bit

**SYNOPSIS**

**loadrt conv\_s32\_bit [count=*N* | names=*name1* [, *name2* ...]]**

**FUNCTIONS**

**conv-s32-bit.*N***

Update 'out' based on 'in'

**PINS**

**conv-s32-bit.*N*.in** s32 in

**conv-s32-bit.*N*.out** bit out

**conv-s32-bit.*N*.out-of-range** bit out

TRUE when 'in' is not in the range of bit

**PARAMETERS**

**conv-s32-bit.*N*.clamp** bit rw

If TRUE, then clamp to the range of bit. If FALSE, then allow the value to "wrap around".

**LICENSE**

GPL

**NAME**

conv\_s32\_float – Convert a value from s32 to float

**SYNOPSIS**

**loadrt conv\_s32\_float** [**count**=*N* | **names**=*name1* [, *name2* ... ]]

**FUNCTIONS**

**conv-s32-float.N** (requires a floating-point thread)

Update 'out' based on 'in'

**PINS**

**conv-s32-float.N.in** s32 in

**conv-s32-float.N.out** float out

**LICENSE**

GPL

**NAME**

conv\_s32\_u32 – Convert a value from s32 to u32

**SYNOPSIS**

**loadrt conv\_s32\_u32** [**count**=*N* | **names**=*name1* [, *name2* ... ]]

**FUNCTIONS**

**conv-s32-u32.N**

Update 'out' based on 'in'

**PINS**

**conv-s32-u32.N.in** s32 in

**conv-s32-u32.N.out** u32 out

**conv-s32-u32.N.out-of-range** bit out

TRUE when 'in' is not in the range of u32

**PARAMETERS**

**conv-s32-u32.N.clamp** bit rw

If TRUE, then clamp to the range of u32. If FALSE, then allow the value to "wrap around".

**LICENSE**

GPL

**NAME**

conv\_u32\_bit – Convert a value from u32 to bit

**SYNOPSIS**

**loadrt conv\_u32\_bit [count=*N* | names=*name1* [, *name2* ...]]**

**FUNCTIONS**

**conv-u32-bit.*N***

Update 'out' based on 'in'

**PINS**

**conv-u32-bit.*N*.in** u32 in

**conv-u32-bit.*N*.out** bit out

**conv-u32-bit.*N*.out-of-range** bit out

TRUE when 'in' is not in the range of bit

**PARAMETERS**

**conv-u32-bit.*N*.clamp** bit rw

If TRUE, then clamp to the range of bit. If FALSE, then allow the value to "wrap around".

**LICENSE**

GPL

**NAME**

conv\_u32\_float – Convert a value from u32 to float

**SYNOPSIS**

**loadrt conv\_u32\_float** [**count**=*N* | **names**=*name1* [, *name2* ... ]]

**FUNCTIONS**

**conv-u32-float.N** (requires a floating-point thread)

Update 'out' based on 'in'

**PINS**

**conv-u32-float.N.in** u32 in

**conv-u32-float.N.out** float out

**LICENSE**

GPL

**NAME**

conv\_u32\_s32 – Convert a value from u32 to s32

**SYNOPSIS**

**loadrt conv\_u32\_s32** [**count**=*N* | **names**=*name1* [, *name2* ... ]]

**FUNCTIONS**

**conv-u32-s32.N**

Update 'out' based on 'in'

**PINS**

**conv-u32-s32.N.in** u32 in

**conv-u32-s32.N.out** s32 out

**conv-u32-s32.N.out-of-range** bit out

TRUE when 'in' is not in the range of s32

**PARAMETERS**

**conv-u32-s32.N.clamp** bit rw

If TRUE, then clamp to the range of s32. If FALSE, then allow the value to "wrap around".

**LICENSE**

GPL

**NAME**

counter – counts input pulses (**DEPRECATED**)

**SYNOPSIS**

**loadrt counter [num\_chan=N]**

**DESCRIPTION**

**counter** is a deprecated HAL component and will be removed in a future release. Use the **encoder** component with `encoder.X.counter-mode` set to `TRUE`.

**counter** is a HAL component that provides software- based counting that is useful for spindle position sensing and maybe other things. Instead of using a real encoder that outputs quadrature, some lathes have a sensor that generates a simple pulse stream as the spindle turns and an index pulse once per revolution. This component simply counts up when a "count" pulse (phase-A) is received, and if reset is enabled, resets when the "index" (phase-Z) pulse is received.

This is of course only useful for a unidirectional spindle, as it is not possible to sense the direction of rotation.

**counter** conforms to the "canonical encoder" interface described in the HAL manual.

**FUNCTIONS**

**counter.capture-position** (uses floating-point)

Updates the counts, position and velocity outputs based on internal counters.

**counter.update-counters**

Samples the phase-A and phase-Z inputs and updates internal counters.

**PINS**

**counter.N.phase-A** bit in

The primary input signal. The internal counter is incremented on each rising edge.

**counter.N.phase-Z** bit in

The index input signal. When the **index-enable** pin is `TRUE` and a rising edge on **phase-Z** is seen, **index-enable** is set to `FALSE` and the internal counter is reset to zero.

**counter.N.index-enable** bit io

**counter.N.reset** bit io

**counter.N.counts** signed out

**counter.N.position** float out

**counter.N.velocity** float out

These pins function according to the canonical digital encoder interface.

**counter.N.position-scale** float rw

This parameter functions according to the canonical digital encoder interface.

**counter.N.rawcounts** signed ro

The internal counts value, updated from **update-counters** and reflected in the output pins at the next call to **capture-position**.

**SEE ALSO**

**encoder(9)**. in the LinuxCNC documentation.

**NAME**

ddt – Compute the derivative of the input function

**SYNOPSIS**

**loadrt ddt** [**count**=*N* | **names**=*name1* [, *name2* ... ]]

**FUNCTIONS**

**ddt.N** (requires a floating-point thread)

**PINS**

**ddt.N.in** float in

**ddt.N.out** float out

**LICENSE**

GPL

**NAME**

deadzone – Return the center if within the threshold

**SYNOPSIS**

**loadrt deadzone** [**count**=*N*]**names=***name1*[,*name2*...]

**FUNCTIONS**

**deadzone.N** (requires a floating-point thread)

Update **out** based on **in** and the parameters.

**PINS**

**deadzone.N.in** float in

**deadzone.N.out** float out

**PARAMETERS**

**deadzone.N.center** float rw (default: *0.0*)

The center of the dead zone

**deadzone.N.threshold** float rw (default: *1.0*)

The dead zone is **center**  $\pm$  (**threshold**/2)

**LICENSE**

GPL

**NAME**

debounce – filter noisy digital inputs

**SYNOPSIS**

**loadrt debounce cfg=size[,size,...]**

Creates debounce groups with the number of filters specified by (*size*). Every filter in the same group has the same sample rate and delay. For example `cfg=2,3` creates two filter groups with 2 filters in the first group and 3 filters in the second group.

**DESCRIPTION**

The debounce filter works by incrementing a counter whenever the input is true, and decrementing the counter when it is false. If the counter decrements to zero, the output is set false and the counter ignores further decrements. If the counter increments up to a threshold, the output is set true and the counter ignores further increments. If the counter is between zero and the threshold, the output retains its previous state. The threshold determines the amount of filtering: a threshold of 1 does no filtering at all, and a threshold of N requires a signal to be present for N samples before the output changes state.

**FUNCTIONS**

**debounce.G**

Sample all the input pins in group G and update the output pins.

**PINS**

**debounce.G.F.in** bit in

The F'th input pin in group G.

**debounce.G.F.out** bit out

The F'th output pin in group G. Reflects the last "stable" input seen on the corresponding input pin.

**debounce.G.delay** signed rw

Sets the amount of filtering for all pins in group G.

**NAME**

edge – Edge detector

**SYNOPSIS**

**loadrt edge** [**count**=*N*][**names**=*name1*[,*name2*...]]

**FUNCTIONS**

**edge.N** Produce output pulses from input edges

**PINS**

**edge.N.in** bit in

**edge.N.out** bit out

Goes high when the desired edge is seen on 'in'

**edge.N.out-invert** bit out

Goes low when the desired edge is seen on 'in'

**PARAMETERS**

**edge.N.both** bit rw (default: *FALSE*)

If TRUE, selects both edges. Otherwise, selects one edge according to in-edge

**edge.N.in-edge** bit rw (default: *TRUE*)

If both is FALSE, selects the one desired edge: TRUE means falling, FALSE means rising

**edge.N.out-width-ns** s32 rw (default: *0*)

Time in nanoseconds of the output pulse

**edge.N.time-left-ns** s32 r

Time left in this output pulse

**edge.N.last-in** bit r

Previous input value

**LICENSE**

GPL

**NAME**

encoder – software counting of quadrature encoder signals

**SYNOPSIS**

**loadrt encoder** [**num\_chan**=*num* | **names**=*name1*[,*name2*...]]

**DESCRIPTION**

**encoder** is used to measure position by counting the pulses generated by a quadrature encoder. As a software-based implementation it is much less expensive than hardware, but has a limited maximum count rate. The limit is in the range of 10KHz to 50KHz, depending on the computer speed and other factors. If better performance is needed, a hardware encoder counter is a better choice. Some hardware-based systems can count at MHz rates.

**encoder** supports a maximum of eight channels. The number of channels actually loaded is set by the **num\_chan** argument when the module is loaded. Alternatively, specify **names=** and unique names separated by commas.

The **num\_chan=** and **names=** specifiers are mutually exclusive. If neither **num\_chan=** nor **names=** are specified, or if **num\_chan=0** is specified, the default value is three.

**encoder** has a one-phase, unidirectional mode called *counter*. In this mode, the **phase-B** input is ignored; the counts increase on each rising edge of **phase-A**. This mode may be useful for counting a unidirectional spindle with a single input line, though the noise-resistant characteristics of quadrature are lost.

**FUNCTIONS**

**encoder.update-counters** (no floating-point)

Does the actual counting, by sampling the encoder signals and decoding the quadrature waveforms. Must be called as frequently as possible, preferably twice as fast as the maximum desired count rate. Operates on all channels at once.

**encoder.capture-position** (uses floating point)

Captures the raw counts from **update-counters** and performs scaling and other necessary conversion, handles counter rollover, etc. Can (and should) be called less frequently than **update-counters**. Operates on all channels at once.

**NAMING**

The names for pins and parameters are prefixed as:

**encoder.N.** for N=0,1,...,num-1 when using **num\_chan=num**

**nameN.** for nameN=name1,name2,... when using **names=name1,name2,...**

The **encoder.N.** format is shown in the following descriptions.

**PINS**

**encoder.N.counter-mode** bit i/o

Enables counter mode. When true, the counter counts each rising edge of the phase-A input, ignoring the value on phase-B. This is useful for counting the output of a single channel (non-quadrature) sensor. When false (the default), it counts in quadrature mode.

**encoder.N.counts** s32 out

Position in encoder counts.

**encoder.N.index-enable** bit i/o

When true, **counts** and **position** are reset to zero on the next rising edge of **Phase-Z**. At the same time, **index-enable** is reset to zero to indicate that the rising edge has occurred.

**encoder.N.min-speed-estimate** float in (default: 1.0)

Determine the minimum speed at which **velocity** will be estimated as nonzero and **position-interpolated** will be interpolated. The units of **min-speed-estimate** are the same as the units of **velocity**. Setting this parameter too low will cause it to take a long time for **velocity** to go to 0 after encoder pulses have stopped arriving.

**encoder.N.phase-A** bit in

Quadrature input for encoder channel *N*.

**encoder.N.phase-B** bit in

Quadrature input.

**encoder.N.phase-Z** bit in

Index pulse input.

**encoder.N.position** float out

Position in scaled units (see **position-scale**)

**encoder.N.position-interpolated** float out

Position in scaled units, interpolated between encoder counts. Only valid when velocity is approximately constant and above **min-speed-estimate**. Do not use for position control.

**encoder.N.position-scale** float i/o

Scale factor, in counts per length unit. For example, if **position-scale** is 500, then 1000 counts of the encoder will be reported as a position of 2.0 units.

**encoder.N.rawcounts** s32 out

The raw count, as determined by **update-counters**. This value is updated more frequently than **counts** and **position**. It is also unaffected by **reset** or the index pulse.

**encoder.N.reset** bit in

When true, **counts** and **position** are reset to zero immediately.

**encoder.N.velocity** float out

Velocity in scaled units per second. **encoder** uses an algorithm that greatly reduces quantization noise as compared to simply differentiating the **position** output. When the magnitude of the true velocity is below min-speed-estimate, the velocity output is 0.

**encoder.N.x4-mode** bit i/o

Enables times-4 mode. When true (the default), the counter counts each edge of the quadrature waveform (four counts per full cycle). When false, it only counts once per full cycle. In **counter-mode**, this parameter is ignored.

**encoder.N.latch-input** bit in

**encoder.N.latch-falling** bit in (default: **TRUE**)

**encoder.N.latch-rising** bit in (default: **TRUE**)

**encoder.N.counts-latched** s32 out

**encoder.N.position-latched** float out

Update **counts-latched** and **position-latched** on the rising and/or falling edges of **latch-input** as indicated by **latch-rising** and **latch-falling**.

**encoder.N.counter-mode** bit rw

Enables counter mode. When true, the counter counts each rising edge of the phase-A input, ignoring the value on phase-B. This is useful for counting the output of a single channel (non-quadrature) sensor. When false (the default), it counts in quadrature mode. **encoder.N.capture-position.tmax** s32 rw Maximum number of CPU cycles it took to execute this function.

## PARAMETERS

The encoder component has no HAL Parameters.



**NAME**

encoder\_ratio – an electronic gear to synchronize two axes

**SYNOPSIS**

```
loadrt encoder_ratio [num_chan=num | names=name1[,name2...]]
```

**DESCRIPTION**

**encoder\_ratio** can be used to synchronize two axes (like an "electronic gear"). It counts encoder pulses from both axes in software, and produces an error value that can be used with a PID loop to make the slave encoder track the master encoder with a specific ratio.

This module supports up to eight axis pairs. The number of pairs is set by the module parameter **num\_chan**. Alternatively, specify **names=** and unique names separated by commas.

The **num\_chan=** and **names=** specifiers are mutually exclusive. If neither **num\_chan=** nor **names=** are specified, the default value is one.

**FUNCTIONS****encoder\_ratio.sample**

Read all input pins. Must be called at twice the maximum desired count rate.

**encoder\_ratio.update (uses floating-point)**

Updates all output pins. May be called from a slower thread.

**NAMING**

The names for pins and parameters are prefixed as:

**encoder\_ratio.N.** for N=0,1,...,num-1 when using **num\_chan=num**

**nameN.** for nameN=name1,name2,... when using **names=name1,name2,...**

The **encoder\_ratio.N.** format is shown in the following descriptions.

**PINS**

**encoder\_ratio.N.master-A** bit in

**encoder\_ratio.N.master-B** bit in

**encoder\_ratio.N.slave-A** bit in

**encoder\_ratio.N.slave-B** bit in

The encoder channels of the master and slave axes

**encoder\_ratio.N.enable** bit in

When the enable pin is FALSE, the error pin simply reports the slave axis position, in revolutions. As such, it would normally be connected to the feedback pin of a PID block for closed loop control of the slave axis. Normally the command input of the PID block is left unconnected (zero), so the slave axis simply sits still. However when the enable input goes TRUE, the error pin becomes the slave position minus the scaled master position. The scale factor is the ratio of master teeth to slave teeth. As the master moves, error becomes non-zero, and the PID loop will drive the slave axis to track the master.

**encoder\_ratio.N.error** float out

The error in the position of the slave (in revolutions)

**PARAMETERS**

**encoder\_ratio.N.master-ppr** unsigned rw

**encoder\_ratio.N.slave-ppr** unsigned rw

The number of pulses per revolution of the master and slave axes

**encoder-ratio.N.master-teeth** unsigned rw

**encoder-ratio.N.slave-teeth** unsigned rw

The number of "teeth" on the master and slave gears.

## SEE ALSO

**encoder(9)**

**NAME**

estop\_latch – Software ESTOP latch

**SYNOPSIS**

**loadrt** **estop\_latch** [**count**=*N* | **names**=*name1* [, *name2*...]]

**DESCRIPTION**

This component can be used as a part of a simple software ESTOP chain.

It has two states: "OK" and "Faulted".

The initial state is "Faulted". When faulted, the **out-ok** output is false, the **fault-out** output is true, and the **watchdog** output is unchanging.

The state changes from "Faulted" to "OK" when **all** these conditions are true:

- **fault-in** is false
- **ok-in** is true
- **reset** changes from false to true

When "OK", the **out-ok** output is true, the **fault-out** output is false, and the **watchdog** output is toggling.

The state changes from "OK" to "Faulted" when **any** of the following are true:

- **fault-in** is true
- **ok-in** is false

To facilitate using only a single fault source, **ok-in** and **fault-en** are both set to the non-fault-causing value when no signal is connected. For estop-latch to ever be able to signal a fault, at least one of these inputs must be connected.

Typically, an external fault or estop input is connected to **fault-in**, **iocontrol.0.user-request-enable** is connected to **reset**, and **ok-out** is connected to **iocontrol.0.emc-enable-in**.

**In more complex systems, it may be more appropriate to use classicladder to manage the software portion of the estop chain.**

**FUNCTIONS**

**estop-latch.N**

**PINS**

**estop-latch.N.ok-in** bit in (default: *true*)  
**estop-latch.N.fault-in** bit in (default: *false*)  
**estop-latch.N.reset** bit in  
**estop-latch.N.ok-out** bit out (default: *false*)  
**estop-latch.N.fault-out** bit out (default: *true*)  
**estop-latch.N.watchdog** bit out

**LICENSE**

GPL

**NAME**

feedcomp – Multiply the input by the ratio of current velocity to the feed rate

**SYNOPSIS**

**loadrt feedcomp** [count=*N*][names=*name1*[,*name2*...]]

**FUNCTIONS**

**feedcomp.N** (requires a floating-point thread)

**PINS**

**feedcomp.N.out** float out  
Proportionate output value

**feedcomp.N.in** float in  
Reference value

**feedcomp.N.enable** bit in  
Turn compensation on or off

**feedcomp.N.vel** float in  
Current velocity

**PARAMETERS**

**feedcomp.N.feed** float rw  
Feed rate reference value

**NOTES**

Note that if enable is false, out = in

**LICENSE**

GPL

**NAME**

flipflop – D type flip-flop

**SYNOPSIS**

**loadrt flipflop** [**count**=*N* | **names**=*name1* [, *name2* ...]]

**FUNCTIONS**

**flipflop.N**

**PINS**

**flipflop.N.data** bit in  
data input

**flipflop.N.clk** bit in  
clock, rising edge writes data to out

**flipflop.N.set** bit in  
when true, force out true

**flipflop.N.reset** bit in  
when true, force out false; overrides set

**flipflop.N.out** bit io  
output

**LICENSE**

GPL

**NAME**

gantrykins – A kinematics module that maps one axis to multiple joints

**SYNOPSIS**

**loadrt gantrykins coordinates=axisletters**

**Specifying gantry joint mapping via loadrt**

The **coordinates=** parameter specifies the initial gantry joint mapping. Each axis letter is mapped to a joint, starting from 0. So **coordinates=XYZZ** maps the X axis to joint 0, the Y axis to joint 1 and 2, and the Z axis to joint 3. If not specified, the default mapping is **coordinates=XYZABC**. Coordinate letters may be specified in uppercase or lowercase.

**A note about joints and axes**

LinuxCNC makes a distinction between joints and axes: a joint is something controlled by a motor, and an axis is a coordinate you can move via G-code. You can also jog joints or jog axes.

A gantry has two joints controlling one axis, and this requires a bit of special care.

Homing always happens in joint mode (aka Free mode). The two joints of a gantry's axis must be homed together, so they must have the same [AXIS\_n]HOME\_SEQUENCE in the .ini file.

Jogging of a gantry must happen in world mode (aka Teleop mode). If you jog a gantry in joint mode (Free mode), you will move just one of the joints, and the gantry will rack. In contrast, if you jog a gantry in world mode (Teleop mode), it's the axis that jogs: linuxcnc will coordinate the motion of the two joints that make up the axis, both joints will move together, and the gantry will stay square.

The Axis GUI has provisions for jogging in joint mode (Free) and in world mode (Teleop). Use the "\$" hotkey, or the View menu to switch between them.

Joint-mode (aka Free mode) supports continuous and incremental jogging. World-mode (aka Teleop mode) only supports continuous jogging.

**KINEMATICS**

In the inverse kinematics, each joint gets the value of its corresponding axis. In the forward kinematics, each axis gets the value of the highest numbered corresponding joint. For example, with **coordinates=XYZZ** the Y axis position comes from joint 2, not joint 1.

**FUNCTIONS**

None.

**PINS**

None.

**PARAMETERS**

**gantrykins.joint-N** (s32)

Specifies the axis mapped to joint *N*. The values 0 through 8 correspond to the axes XYZ-ABCUVW. It is preferable to use the "coordinates=" parameter at loadrt-time rather than setting the joint-N parameters later, because the gantrykins module prints the joint-to-axis mapping at loadrt-time, and having that output correct is nice.

**NOTES**

**gantrykins** must be loaded before **motion**.

**SEE ALSO**

*Kinematics* section in the LinuxCNC documentation

**LICENSE**

GPL

**NAME**

gearchange – Select from one two speed ranges

**SYNOPSIS**

The output will be a value scaled for the selected gear, and clamped to the min/max values for that gear. The scale of gear 1 is assumed to be 1, so the output device scale should be chosen accordingly. The scale of gear 2 is relative to gear 1, so if gear 2 runs the spindle 2.5 times as fast as gear 1, scale2 should be set to 2.5.

**FUNCTIONS**

**gearchange.N** (requires a floating-point thread)

**PINS**

**gearchange.N.sel** bit in

Gear selection input

**gearchange.N.speed-in** float in

Speed command input

**gearchange.N.speed-out** float out

Speed command to DAC/PWM

**gearchange.N.dir-in** bit in

Direction command input

**gearchange.N.dir-out** bit out

Direction output - possibly inverted for second gear

**PARAMETERS**

**gearchange.N.min1** float rw (default: 0)

Minimum allowed speed in gear range 1

**gearchange.N.max1** float rw (default: 100000)

Maximum allowed speed in gear range 1

**gearchange.N.min2** float rw (default: 0)

Minimum allowed speed in gear range 2

**gearchange.N.max2** float rw (default: 100000)

Maximum allowed speed in gear range 2

**gearchange.N.scale2** float rw (default: 1.0)

Relative scale of gear 2 vs. gear 1. Since it is assumed that gear 2 is "high gear", **scale2** must be greater than 1, and will be reset to 1 if set lower.

**gearchange.N.reverse** bit rw (default: 0)

Set to 1 to reverse the spindle in second gear

**LICENSE**

GPL

**NAME**

gladevcp – displays Virtual control Panels built with GTK / GLADE

**SYNOPSIS**

**loadusr gladevcp [-c componentname0xN] [-g WxH+Xoffset+Yoffset0xN] [-H halcmdfile] [-x windowid] gladefile.glade**

**DESCRIPTION**

gladevcp parses a glade file and displays the widgets in a window. Then calls gladevcp\_makepins which again parses the glade file looking for specific HAL widgets then makes HAL pins and sets up updating for them. The HAL component name defaults to the basename of the glade file. The -x option directs gladevcp to reparent itself under this X window id instead of creating its own toplevel window. The -H option passes an input file for halcmd to be run after the gladevcp component is initialized. This is used in Axis when running gladevcp under a tab with the EMBED\_TAB\_NAME/EMBED\_TAB\_COMMAND ini file feature.

gladevcp supports gtkbuilder or libglade files though some widgets are not fully supported in gtkbuilder yet.

**ISSUES**

For now system links need to be added in the glade library folders to point to our new widgets and catalog files. look in lib/python/gladevcp/README for details

**NAME**

gray2bin – convert a gray-code input to binary

**SYNOPSIS**

**loadrt gray2bin** [**count**=*N* | **names**=*name1* [, *name2* ... ]]

**DESCRIPTION**

Converts a gray-coded number into the corresponding binary value

**FUNCTIONS**

**gray2bin.N**

**PINS**

**gray2bin.N.in** u32 in  
gray code in

**gray2bin.N.out** u32 out  
binary code out

**AUTHOR**

andy pugh

**LICENSE**

GPL

**NAME**

histobins – histogram bins utility for scripts/hal-histogram

**SYNOPSIS**

Usage:

Read availablebins pin for the number of bins available.

Set the minvalue, binsize, and nbins pins.

Ensure nbins <= availablebins

For nbins = N, the bins are numbered: 0 ... N-1

Iterate:

Set index pin to a bin number: 0 <= index < nbins.

Read check pin and verify that check pin == index pin.

Read outputs: binvalue, pextra, nextra pins.

(binvalue is count for the indexed bin)

(pextra is count for all inputs > maxvalue)

(nextra is count for all bins < minvalue)

If index is out of range (index < 0 or index > maxbinnumber)

then binvalue == -1.

The input-error pin is set when input rules are violated and updates cease.

The reset pin may be used to restart.

The input used is selected based on pintype:

pintype inputpin

-----

0 input

1 input-s32

2 input-u32

3 input-bit

Additional output statistics pins:

input-min

input-max

nsamples

variance

mean

The method input pin selects an alternate variance calculation.

Maintainers note: hardcoded for MAXBINNUMBER==200

**FUNCTIONS**

**histobins.N** (requires a floating-point thread)

**PINS**

**histobins.N.pintype** u32 in

**histobins.N.input** float in

**histobins.N.input-s32** s32 in

**histobins.N.input-u32** u32 in

**histobins.N.input-bit** bit in

**histobins.N.nbins** u32 in (default: 20)

**histobins.N.binsize** float in (default: 1)

**histobins.N.minvalue** float in (default: 0)

**histobins.N.index** s32 in

**histobins.N.check** s32 out  
**histobins.N.reset** bit in  
**histobins.N.method** bit in  
**histobins.N.input-error** bit out  
**histobins.N.binvalue** float out  
**histobins.N.pextra** float out  
**histobins.N.nextra** float out  
**histobins.N.input-min** float out  
**histobins.N.input-max** float out  
**histobins.N.nsamples** u32 out  
**histobins.N.variance** float out  
**histobins.N.mean** float out  
**histobins.N.availablebins** s32 out (default: 200)

## LICENSE

GPL

**NAME**

hm2\_7i43 – LinuxCNC HAL driver for the Mesa Electronics 7i43 EPP Anything IO board with HostMot2 firmware.

**SYNOPSIS**

```
loadrt hm2_7i43 [ioaddr=N[,N...]] [ioaddr_hi=N[,N...]] [epp_wide=N[,N...]] [config="str[,str...]"
[debug_epp=N[,N...]]
```

**ioaddr** [default: 0x378]

The base address of the parallel port.

**ioaddr\_hi** [default: 0]

The secondary address of the parallel port, used to set EPP mode. 0 means to use ioaddr + 0x400.

**epp\_wide** [default: 1]

Set to zero to disable the "wide EPP mode". "Wide" mode allows a 16- and 32-bit EPP transfers, which can reduce the time spent in the read and write functions. However, this may not work on all EPP parallel ports.

**config** [default: ""]

HostMot2 config strings, described in the hostmot2(9) manpage.

**debug\_epp** [default: 0]

Developer/debug use only! Enable debug logging of most EPP transfers.

**DESCRIPTION**

hm2\_7i43 is a device driver that interfaces the Mesa 7i43 board with the HostMot2 firmware to the LinuxCNC HAL. Both the 200K and the 400K FPGAs are supported.

The driver talks with the 7i43 over the parallel port, not over USB. USB can be used to power the 7i43, but not to talk to it. USB communication with the 7i43 will not be supported any time soon, since USB has poor real-time qualities.

The driver programs the board's FPGA with firmware when it registers the board with the hostmot2 driver. The old bload(1) firmware loading method is not used anymore. Instead the firmware to load is specified in the **config** modparam, as described in the hostmot2(9) manpage, in the *config modparam* section.

**Jumper settings**

To send the FPGA configuration from the PC, the board must be configured to get its firmware from the EPP port. To do this, jumpers W4 and W5 must both be down, ie toward the USB connector.

The board must be configured to power on whether or not the USB interface is active. This is done by setting jumper W7 up, ie away from the edge of the board.

**Communicating with the board**

The 7i43 communicates with the LinuxCNC computer over EPP, the Enhanced Parallel Port. This provides about 1 MBps of throughput, and the communication latency is very predictable and reasonably low.

The parallel port must support EPP 1.7 or EPP 1.9. EPP 1.9 is preferred, but EPP 1.7 will work too. The EPP mode of the parallel port is sometimes a setting in the BIOS.

Note that the popular "NetMOS" aka "MosChip 9805" PCI parport cards **do not work**. They do not meet the EPP spec, and cannot be reliably used with the 7i43. You have to find another card, sorry.

EPP is very reliable under normal circumstances, but bad cabling or excessively long cabling runs may cause communication timeouts. The driver exports a parameter named hm2\_7i43.<BoardNum>.io\_error to inform HAL of this condition. When the driver detects an EPP timeout, it sets io\_error to True and stops communicating with the 7i43 board. Setting io\_error back to False makes the driver start trying to

communicate with the 7i43 again.

Access to the EPP bus is not threadsafe: only one realtime thread may access the EPP bus.

**SEE ALSO**

hostmot2(9)

**LICENSE**

GPL

**NAME**

hm2\_7i90 – LinuxCNC HAL driver for the Mesa Electronics 7i90 EPP Anything IO board with HostMot2 firmware.

**SYNOPSIS**

**loadrt hm2\_7i90** [**ioaddr**=*N[,N...]*] [**ioaddr\_hi**=*N[,N...]*] [**epp\_wide**=*N[,N...]*] [**debug\_epp**=*N[,N...]*]

**ioaddr** [default: 0x378]

The base address of the parallel port.

**ioaddr\_hi** [default: 0]

The secondary address of the parallel port, used to set EPP mode. 0 means to use ioaddr + 0x400.

**epp\_wide** [default: 1]

Set to zero to disable the "wide EPP mode". "Wide" mode allows a 16- and 32-bit EPP transfers, which can reduce the time spent in the read and write functions. However, this may not work on all EPP parallel ports.

**debug\_epp** [default: 0]

Developer/debug use only! Enable debug logging of most EPP transfers.

**DESCRIPTION**

hm2\_7i90 is a device driver that interfaces the Mesa 7i90 board with the HostMot2 firmware to the LinuxCNC HAL.

The 7i90 firmware is fixed, it is not programmed by the driver at load time.

The driver talks with the 7i90 over the parallel port, via EPP.

The hm2\_7i90 driver requires an in-kernel EPP communications API. This is provided by the 'epp' driver. Load this driver before loading hm2\_7i90, by putting 'loadrt epp' in your .hal file.

**Communicating with the board**

The 7i90 communicates with the LinuxCNC computer over EPP, the Enhanced Parallel Port. This provides about 1 MBps of throughput, and the communication latency is very predictable and reasonably low.

The parallel port must support EPP 1.7 or EPP 1.9. EPP 1.9 is preferred, but EPP 1.7 will work too. The EPP mode of the parallel port is sometimes a setting in the BIOS.

Note that the popular "NetMOS" aka "MosChip 9805" PCI parport cards **do not work**. They do not meet the EPP spec, and cannot be reliably used with the 7i90. You have to find another card, sorry.

EPP is very reliable under normal circumstances, but bad cabling or excessively long cabling runs may cause communication timeouts. The driver exports a parameter named hm2\_7i90.<BoardNum>.io\_error to inform HAL of this condition. When the driver detects an EPP timeout, it sets io\_error to True and stops communicating with the 7i90 board. Setting io\_error back to False makes the driver start trying to communicate with the 7i90 again.

Access to the EPP bus is not threadsafe: only one realtime thread may access the EPP bus.

**SEE ALSO**

hostmot2(9)

**LICENSE**

GPL

**NAME**

hm2\_eth – LinuxCNC HAL driver for the Mesa Electronics Ethernet Anything IO boards, with HostMot2 firmware.

**SYNOPSIS**

```
loadrt hm2_eth [config="str[,str...]" [board_ip=ip[,ip...]] [board_mac=mac[,mac...]]fB]
```

**config** [default: ""]

HostMot2 config strings, described in the hostmot2(9) manpage.

**board\_ip** [default: ""]

The IP address of the board(s), separated by commas. As shipped, the board address is 192.168.1.121.

**DESCRIPTION**

hm2\_eth is a device driver that interfaces Mesa's ethernet based Anything I/O boards (with the HostMot2 firmware) to the LinuxCNC HAL.

The supported boards are: 7I80DB, 7I80HD.

The board must have its hardware loaded on the board by the mesaflash(1) program.

hm2\_eth is only available when linuxcnc is configured with "uspace" realtime.

**INTERFACE CONFIGURATION**

hm2\_eth should be used on a dedicated network interface, with only a cable between the PC and the board. Wireless and USB network interfaces are not suitable.

These instructions assume your dedicated network interface is "eth1", 192.168.1/24 is an unused private network, that the hostmot2 board is using the default address of 192.168.1.121, that you are using Debian 7 or similar, and that you do not otherwise use iptables. If any of these are false, you will need to modify the instructions accordingly. After following all the instructions, reboot so that the changes take effect.

It is particularly important to check that the network 192.168.1/24 is not already the private network used by your internet router, because this is a commonly-used value. If you use another network, you will also need to reconfigure the hostmot2 card to use an IP address on that network by using the mesaflash(1) utility and change jumper settings. Typically, you will choose one of the networks in the Private IPv4 address space. ([http://en.wikipedia.org/wiki/IPv4#Private\\_networks](http://en.wikipedia.org/wiki/IPv4#Private_networks)) One common alternative is PC address 10.10.10.1, hostmot2 address 10.10.10.10.

Use of the dedicated ethernet interface while linuxcnc is running can cause violation of realtime guarantees. hm2\_eth will automatically mitigate most accidental causes of interference.

**Configure network with static address**

Add these lines to the file /etc/network/interfaces to configure eth1 with a static address:

```
auto eth1
iface eth1 inet static
    address 192.168.1.1
```

**NOTES**

hm2\_eth uses an iptables chain called "hm2-eth-rules-output" to control access to the network interface while hal is running. The chain is created if it does not exist, and a jump to it is inserted at the beginning of the OUTPUT chain if it is not there already. If you have an existing iptables setup, you can insert a direct jump from OUTPUT to hm2-eth-rules-output in an order appropriate to your local network.

At (normal) exit, `hm2_eth` will remove the rules. After a crash, you can manually clear the rules with **`sudo iptables -F hm2-eth-rules-output`**; the rules are also removed by a reboot.

## BUGS

Some `hostmot2` functions such `uart` are coded in a way that causes additional latency when used with `hm2_eth`.

The HAL pins for the LEDs are called `CR01..CR04`, but the silkscreens are `CR3..CR6`. Depending on the FPGA firmware, the LEDs may initially be under control of the ethernet engine. This can be changed until power cycle with

```
elbpcom 01D914000000
```

## SEE ALSO

`hostmot2(9)`

## LICENSE

GPL

**NAME**

hm2\_pci – LinuxCNC HAL driver for the Mesa Electronics PCI-based Anything IO boards, with HostMot2 firmware.

**SYNOPSIS**

```
loadrt hm2_pci [config="str[,str...]"]
```

```
    config [default: ""]
```

HostMot2 config strings, described in the hostmot2(9) manpage.

**DESCRIPTION**

hm2\_pci is a device driver that interfaces Mesa's PCI and PC-104/Plus based Anything I/O boards (with the HostMot2 firmware) to the LinuxCNC HAL.

The supported boards are: the 5i20, 5i21, 5i22, 5i23, 5i24, and 5i25 (all on PCI); the 4i65, 4i68, and 4i69 (on PC-104/Plus), and the 3x20 (using a 6i68 or 7i68 carrier card) and 6i25 (on PCI Express).

The driver optionally programs the board's FPGA with firmware when it registers the board with the hostmot2 driver. The firmware to load is specified in the **config** modparam, as described in the hostmot2(9) manpage, in the *config modparam* section.

**SEE ALSO**

hostmot2(9)

**LICENSE**

GPL

**NAME**

hm2\_spi – LinuxCNC HAL driver for the Mesa Electronics SPI Anything IO boards, with HostMot2 firmware.

**SYNOPSIS**

```
loadrt hm2_spi [config="str[,str...]"] [spidev_path=path[,path...]] [spidev_rate=rate[,rate...]]
```

**config** [default: ""]

HostMot2 config strings, described in the hostmot2(9) manpage.

**spidev\_path** [default: "/dev/spidev1.0"]

The path to the spi device node, a character special device in /dev

**spidev\_rate** [default: 24000]

The desired rate of the SPI clock in kHz. If the exact specified clock is not available, a lower clock is used. Due to shortcomings in the spidev API, it is not possible for hal to report the actual clock used.

**DESCRIPTION**

hm2\_spi is a device driver that interfaces Mesa's SPI based Anything I/O boards (with the HostMot2 firmware) to the LinuxCNC HAL.

The supported boards are: 7I90HD.

The board must have a compatible firmware loaded on the board by the mesaflash(1) program.

hm2\_spi is only available when linuxcnc is configured with "uspace" realtime.

**INTERFACE CONFIGURATION**

It is possible for one SPI bus to connect several devices; in this configuration, a master device has several chip select lines. In order to meet realtime deadlines, hm2\_spi should be used on a dedicated SPI interface not shared with any other slaves.

**REALTIME PERFORMANCE OF LINUX SPIDEV DRIVERS**

As of kernel 3.8, most or all kernel SPI drivers do not achieve the high realtime response rate required for a typical linuxcnc configuration. The driver was tested with a modified version of the spi-s3c64xx SPI driver on the Odroid U3 platform. The patched kernel resides on github <<https://github.com/jepler/odroid-linux/tree/odroid-3.8.13-rt>>.

**SPI CLOCK RATES**

The maximum SPI clock of the 7i90 is documented as 50MHz. Other elements of the data path between HAL and the 7i90 may impose other limitations.

**SEE ALSO**

hostmot2(9)

**LICENSE**

GPL

**NAME**

hostmot2 – LinuxCNC HAL driver for the Mesa Electronics HostMot2 firmware.

**SYNOPSIS**

See the config modparam section below for Mesa card configuration. Typically hostmot2 is loaded with no parameters unless debugging is required.

**loadrt hostmot2** [**debug\_idrom**=*N*] [**debug\_module\_descriptors**=*N*] [**debug\_pin\_descriptors**=*N*] [**debug\_modules**=*N*]

**debug\_idrom** [default: 0]

Developer/debug use only! Enable debug logging of the HostMot2 IDROM header.

**debug\_module\_descriptors** [default: 0]

Developer/debug use only! Enable debug logging of the HostMot2 Module Descriptors.

**debug\_pin\_descriptors** [default: 0]

Developer/debug use only! Enable debug logging of the HostMot2 Pin Descriptors.

**debug\_modules** [default: 0]

Developer/debug use only! Enable debug logging of the HostMot2 Modules used.

**use\_serial\_numbers** [default: 0]

When creating HAL pins for smart-serial devices name the pins by the board serial number rather than which board and port they are connected to. With this option set to 1 pins will have names like **hm2\_8i20.1234.current** rather than **hm2\_5i23.0.8i20.0.1.current**. The identifier consists of the last 4 digits of the board serial number, which is normally on a sticker on the board. This will make configs less portable, but does mean that boards can be re-connected less carefully.

**DESCRIPTION**

hostmot2 is a device driver that interfaces the Mesa HostMot2 firmware to the LinuxCNC HAL. This driver by itself does nothing, the boards that actually run the firmware require their own drivers before anything can happen. Currently drivers are available for the 5i20, 5i22, 5i23, 5i25, 3x20, 4i65, and 4i68 (all using the hm2\_pci module) and the 7i43 (using the hm2\_7i43 module).

The HostMot2 firmware provides modules such as encoders, PWM generators, step/dir generators, and general purpose I/O pins (GPIOs). These things are called "Modules". The firmware is configured, at firmware compile time, to provide zero or more instances of each of these Modules.

**Board I/O Pins**

The HostMot2 firmware runs on an FPGA board. The board interfaces with the computer via PCI, PC-104/Plus, or EPP, and interfaces with motion control hardware such as servos and stepper motors via I/O pins on the board.

Each I/O pin can be configured, at board-driver load time, to serve one of two purposes: either as a particular I/O pin of a particular Module instance (encoder, pwmgen, stepgen etc), or as a general purpose digital I/O pin. By default all Module instances are enabled, and all the board's pins are used by the Module instances.

The user can disable Module instances at board-driver load time, by specifying a hostmot2 config string modparam. Any pins which belong to Module instances that have been disabled automatically become GPIOs.

All IO pins have some HAL presence, whether they belong to an active module instance or are full GPIOs. GPIOs can be changed (at run-time) between inputs, normal outputs, and open drains, and have a flexible HAL interface. IO pins that belong to active Module instances are constrained by the requirements of the owning Module, and have a more limited interface in HAL. This is described in the General Purpose I/O section below.

**config modparam**

All the board-driver modules (hm2\_pci and hm2\_7i43) accept a load-time modparam of type string array, named "config". This array has one config string for each board the driver should use. Each board's config string is passed to and parsed by the hostmot2 driver when the board-driver registers the board.

The config string can contain spaces, so it is usually a good idea to wrap the whole thing in double-quotes (the " character).

The comma character (,) separates members of the config array from each other.

For example, if your control computer has one 5i20 and one 5i23 you might load the hm2\_pci driver with a HAL command (in halcmd) something like this:

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

Note: this assumes that the hm2\_pci driver detects the 5i20 first and the 5i23 second. If the detection order does not match the order of the config strings, the hostmot2 driver will refuse to load the firmware and the board-driver (hm2\_pci or hm2\_7i43) will fail to load. To the best of my knowledge, there is no way to predict the order in which PCI boards will be detected by the driver, but the detection order will be consistent as long as PCI boards are not moved around. Best to try loading it and see what the detection order is.

The valid entries in the format string are:

```
[firmware=F]
[num_encoders=N]
[ssi_chan_N=abc%ng]
[biss_chan_N=abc%ng]
[fanuc_chan_N=abc%ng]
[num_resolvers=N]
[num_pwmgens=N]
[num_3pwmgens=N]
[num_stepgens=N]
[stepgen_width=N]
[sserial_port_0=00000000]
[num_leds=N]
[enable_raw]
```

**firmware** [optional]

Load the firmware specified by F into the FPGA on this board. If no "**firmware=F**" string is specified, the FPGA will not be re-programmed but may continue to run a previously downloaded firmware.

The requested firmware F is fetched by udev. udev searches for the firmware in the system's firmware search path, usually /lib/firmware. F typically has the form "hm2/<BoardType>/file.bit"; a typical value for F might be "hm2/5i20/SVST8\_4.BIT". The hostmot2 firmware files are supplied by the hostmot2-firmware packages, available from linuxcnc.org and can normally be installed by entering the command "sudo apt-get install hostmot2-firmware-5i23" to install the support files for the 5i23 for example.

The 5i25 / 6i25 come pre-programmed with firmware and no "firmware=" string should be used with these cards. To change the firmware on a 5i25 or 6i25 the "mesaflash" utility should be used (available from Mesa). It is perfectly valid and reasonable to load these cards with no config string at all.

**num\_dpils** [optional, default: -1]

The hm2dpil is a phase-locked loop timer module which may be used to trigger certain types of encoder. This parameter can be used to disable the hm2dpil by setting the number to 0. There is only ever one module of this type, with 4 timer channels, so the other valid numbers are -1 (enable all) and 1, both of which end up meaning the same thing.

**num\_encoders** [optional, default: -1]

Only enable the first N encoders. If N is -1, all encoders are enabled. If N is 0, no encoders are enabled. If N is greater than the number of encoders available in the firmware, the board will fail to register.

**ssi\_chan\_N** [optional, default: ""]

Specifies how the bit stream from a Synchronous Serial Interface device will be interpreted. There should be an entry for each device connected. Only channels with a format specifier will be enabled. (as the software can not guess data rates and bit lengths)

**biss\_chan\_N** [optional, default: ""]

As for ssi\_chan\_N, but for BiSS devices

**fanuc\_chan\_N** [optional, default: ""]

Specifies how the bit stream from a Fanuc absolute encoder will be interpreted. There should be an entry for each device connected. Only channels with a format specifier will be enabled. (as the software can not guess data rates and bit lengths)

**num\_resolvers** [optional, default: -1]

Only enable the first N resolvers. If N = -1 then all resolvers are enabled. This module does not work with generic resolvers (unlike the encoder module which works with any encoder). At the time of writing the Hostmot2 Resolver function only works with the Mesa 7i49 card.

**num\_pwmgens** [optional, default: -1]

Only enable the first N pwmgens. If N is -1, all pwmgens are enabled. If N is 0, no pwmgens are enabled. If N is greater than the number of pwmgens available in the firmware, the board will fail to register.

**num\_3pwmgens** [optional, default: -1]

Only enable the first N Three-phase pwmgens. If N is -1, all 3pwmgens are enabled. If N is 0, no pwmgens are enabled. If N is greater than the number of pwmgens available in the firmware, the board will fail to register.

**num\_stepgens** [optional, default: -1]

Only enable the first N stepgens. If N is -1, all stepgens are enabled. If N is 0, no stepgens are enabled. If N is greater than the number of stepgens available in the firmware, the board will fail to register.

**stepgen\_width** [optional, default: 2]

Used to mask extra, unwanted, stepgen pins. Stepper drives typically require only two pins (step and dir) but the Hostmot2 stepgen can drive up to 8 output pins for specialised applications (depending on firmware). This parameter applies to all stepgen instances. Unused, masked pins will be available as GPIO.

**sserial\_port\_N (N = 0 .. 3)** [optional, default: 00000000 for all ports]

Up to 32 Smart Serial devices can be connected to a Mesa Anything IO board depending on the firmware used and the number of physical connections on the board. These are arranged in 1-4 ports of 1 to 8 channels.

Some Smart Serial (SSLBP) cards offer more than one load-time configuration, for example all inputs, or all outputs, or offering additional analogue input on some digital pins.

To set the modes for port 0 use, for example **sserial\_port\_0=0120xxxx**

A '0' in the string sets the corresponding port to mode 0, 1 to mode 1, and so on up to mode 9. An "x" in any position disables that channel and makes the corresponding FPGA

pins available as GPIO.

The string can be up to 8 characters long, and if it defines more modes than there are channels on the port then the extras are ignored. Channel numbering is left to right so the example above would set sserial device 0.0 to mode 0, 0.2 to mode2 and disable channels 0.4 onwards.

The sserial driver will auto-detect connected devices, no further configuration should be needed. Unconnected channels will default to GPIO, but the pin values will vary semi-randomly during boot when card-detection runs, so it is best to actively disable any channel that is to be used for GPIO.

**num\_bspis** [optional, default: -1]

Only enable the first N Buffered SPI drivers. If N is -1 then all the drivers are enabled. Each BSPI driver can address 16 devices.

**num\_leds** [optional, default: -1]

Only enable the first N of the LEDs on the FPGA board. If N is -1, then HAL pins for all the LEDs will be created. If N=0 then no pins will be added.

**enable\_raw** [optional]

If specified, this turns on a raw access mode, whereby a user can peek and poke the firmware from HAL. See Raw Mode below.

## dpll

The hm2dpll module has pins like "hm2\_<BoardType>.<BoardNum>.dpll" It is likely that the pin-count will decrease in the future and that some pins will become parameters. This module is a phase-locked loop that will synchronise itself with the thread in which the hostmot2 "read" function is installed and will trigger other functions that are allocated to it at a specified time before or after the "read" function runs. This can currently only be applied to the three absolute encoder types and is intended to ensure that the data is ready when needed, and as fresh as possible.

Pins:

(float, in) hm2\_<BoardType>.<BoardNum>.dpll.NN.timer-us

This pin sets the triggering offset of the associated timer. There are 4 timers numbered 01 to 04, represented by the NN digits in the pin name. The units are micro-seconds. Generally the value will be negative, so that some action is undertaken by the fpga prior to the execution of the main hostmot2 read.

For stepgen and quadrature encoders, the value needs to be more than the maximum variation between read times. -100 will suffice for most systems, and -50 will work on systems with good performance and latency.

For serial encoders, the value also needs to include the time it takes to transfer the absolute encoder position. For instance, if 50 bits must be read at 500kHz then subtract an additional  $50/500\text{kHz} = 100\mu\text{s}$  to get a starting value of -200.

(float, in) hm2\_<BoardType>.<BoardNum>.dpll.base-freq-khz

This pin sets the base frequency of the phase-locked loop. by default it will be set to the nominal frequency of the thread in which the PLL is running and will not normally need to be changed.

(float, out) hm2\_<BoardType>.<BoardNum>.dpll.phase-error-us

Indicates the phase error of the DPLL. If the number cycles by a large amount it is likely that the PLL has failed to achieve lock and adjustments will need to be made.

- (u32, in) hm2\_<BoardType>.<BoardNum>.dpll.time-const"  
The filter time-constant for the PLL. Default 40960 (0xA000)
- (u32, in) hm2\_<BoardType>.<BoardNum>.dpll.plimit"  
Sets the phase adjustment limit of the PLL. If the value is zero then the PLL will free-run at the base frequency independent of the servo thread rate. This is probably not what you want. Default 4194304 (0x400000) Units not known...
- (u32, out) hm2\_<BoardType>.<BoardNum>.dpll.ddsiz  
Used internally by the driver, likely to disappear.
- (u32, in) hm2\_<BoardType>.<BoardNum>.dpll.prescale  
Prescale factor for the rate generator. Default 1.

## encoder

Encoders have names like `""hm2_<BoardType>.<BoardNum>.encoder.<Instance>".` "Instance" is a two-digit number that corresponds to the HostMot2 encoder instance number. There are "num\_encoders" instances, starting with 00.

So, for example, the HAL pin that has the current position of the second encoder of the first 5i20 board is: `hm2_5i20.0.encoder.01.position` (this assumes that the firmware in that board is configured so that this HAL object is available)

Each encoder uses three or four input IO pins, depending on how the firmware was compiled. Three-pin encoders use A, B, and Index (sometimes also known as Z). Four-pin encoders use A, B, Index, and Index-mask.

The hm2 encoder representation is similar to the one described by the Canonical Device Interface (in the HAL General Reference document), and to the software encoder component. Each encoder instance has the following pins and parameters:

Pins:

- (s32 out) count  
Number of encoder counts since the previous reset.
- (float out) position  
Encoder position in position units (count / scale).
- (float out) velocity  
Estimated encoder velocity in position units per second.
- (bit in) reset  
When this pin is TRUE, the count and position pins are set to 0. (The value of the velocity pin is not affected by this.) The driver does not reset this pin to FALSE after resetting the count to 0, that is the user's job.
- (bit in/out) index-enable  
When this pin is set to True, the count (and therefore also position) are reset to zero on the next Index (Phase-Z) pulse. At the same time, index-enable is reset to zero to indicate that the pulse has occurred.

(s32 out) rawcounts

Total number of encoder counts since the start, not adjusted for index or reset.

Parameters:

(float r/w) scale

Converts from 'count' units to 'position' units.

(bit r/w) index-invert

If set to True, the rising edge of the Index input pin triggers the Index event (if index-enable is True). If set to False, the falling edge triggers.

(bit r/w) index-mask

If set to True, the Index input pin only has an effect if the Index-Mask input pin is True (or False, depending on the index-mask-invert pin below).

(bit r/w) index-mask-invert

If set to True, Index-Mask must be False for Index to have an effect. If set to False, the Index-Mask pin must be True.

(bit r/w) counter-mode

Set to False (the default) for Quadrature. Set to True for Step/Dir (in which case Step is on the A pin and Dir is on the B pin).

(bit r/w) filter

If set to True (the default), the quadrature counter needs 15 clocks to register a change on any of the three input lines (any pulse shorter than this is rejected as noise). If set to False, the quadrature counter needs only 3 clocks to register a change. The encoder sample clock runs at 33 MHz on the PCI AnyIO cards and 50 MHz on the 7i43.

(float r/w) vel-timeout

When the encoder is moving slower than one pulse for each time that the driver reads the count from the FPGA (in the `hm2_read()` function), the velocity is harder to estimate. The driver can wait several iterations for the next pulse to arrive, all the while reporting the upper bound of the encoder velocity, which can be accurately guessed. This parameter specifies how long to wait for the next pulse, before reporting the encoder stopped. This parameter is in seconds.

(s32 r/w) `hm2_XiXX.N.encoder.MM.timer-num` (default: -1)

Sets the `hm2dp11` timer instance to be used to latch encoder counts. A setting of -1 does not latch encoder counts. A setting of 0 latches at the same time as the main `hostmot2` write. A setting of 1..4 uses a time offset from the main `hostmot2` write according to the `dp11`'s `timer-us` setting.

Typically, `timer-us` should be a negative number with a magnitude larger than the largest latency (e.g., -100 for a system with mediocre latency, -50 for a system with good latency).

If no DPLL module is present in the FPGA firmware, or if the encoder module does not support DPLL, then this pin is not created.

When available, this feature should typically be enabled. Doing so generally reduces following errors.

## Synchronous Serial Interface (SSI)

(Not to be confused with the Smart Serial Interface)

One pin is created for each SSI instance regardless of data format: (bit, in) hm2\_XiXX.NN.ssi.MM.data-incomplete This pin will be set "true" if the module was still transferring data when the value was read. When this problem exists there will also be a limited number of error messages printed to the UI. This pin should be used to monitor whether the problem has been addressed by config changes. Solutions to the problem depend on whether the encoder read is being triggered by the hm2dp11 phase-locked-loop timer (described above) or by the trigger-encoders function (described below).

The names of the pins created by the SSI module will depend entirely on the format string for each channel specified in the loadrt command line. A typical format string might be **ssi\_chan\_0=error%1bposition%24g**

This would interpret the LSB of the bit-stream as a bit-type pin named "error" and the next 24 bits as a Gray-coded encoder counter. The encoder-related HAL pins would all begin with "position".

There should be no spaces in the format string, as this is used as a delimiter by the low-level code.

The format consists of a string of alphanumeric characters that will form the HAL pin names, followed by a % symbol, a bit-count and a data type. All bits in the packet must be defined, even if they are not used. There is a limit of 64 bits in total.

The valid format characters and the pins they create are:

p: (Pad). Does not create any pins, used to ignore sections of the bit stream that are not required.

b: (Boolean).

(bit, out) hm2\_XiXX.N.ssi.MM.<name>. If any bits in the designated field width are non-zero then the HAL pin will be "true".

(bit, out) hm2\_XiXX.N.ssi.MM.<name>-not. An inverted version of the above, the HAL pin will be "true" if all bits in the field are zero.

u: (Unsigned)

(float, out) hm2\_XiXX.N.ssi.MM.<name>. The value of the bits interpreted as an unsigned integer then scaled such that the pin value will equal the scalmax parameter value when all bits are high. (for example if the field is 8 bits wide and the scalmax parameter was 20 then a value of 255 would return 20, and 0 would return 0.

s: (Signed)

(float, out) hm2\_XiXX.N.ssi.MM.<name>. The value of the bits interpreted as a 2s complement signed number then scaled similarly to the unsigned variant, except symmetrical around zero.

f: (bitField)

(bit, out) hm2\_XiXX.N.ssi.MM.<name>-NN. The value of each individual bit in the data field. NN starts at 00 up to the number of bits in the field.

(bit, out) hm2\_XiXX.N.ssi.MM.<name>-NN-not. An inverted version of the individual bit values.

e: (Encoder)

(s32, out) hm2\_XiXX.N.ssi.MM.<name>.count. The lower 32 bits of the total encoder counts. This value is reset both by the ...reset and the ...index- enable pins.

(s32, out) hm2\_XiXX.N.ssi.MM.<name>.rawcounts. The lower 32 bits of the total

encoder counts. The pin is not affected by reset and index.

(float, out) hm2\_XiXX.N.ssi.MM.<name>.position. The encoder position in machine units. This is calculated from the full 64-bit buffers so will show a true value even after the counts pins have wrapped. It is zeroed by reset and index enable.

(bit, IO) hm2\_XiXX.N.ssi.MM.<name>.index-enable. When this pin is set "true" the module will wait until the raw encoder counts next passes through an integer multiple of the number of counts specified by counts-per-rev parameter and then it will zero the counts and position pins, and set the index-enable pin back to "false" as a signal to the system that "index" has been passed. this pin is used for spindle-synchronised motion and index-homing.

(bit, in) (bit, out) hm2\_XiXX.N.ssi.MM.<name>.reset. When this pin is set high the counts and position pins are zeroed.

h: (Split encoder, high-order bits)

Some encoders (Including Fanuc) place the encoder part-turn counts and full-turn counts in separate, non-contiguous fields. This tag defines the high-order bits of such an encoder module. There can be only one h and one l tag per channel, the behaviour with multiple such channels will be undefined.

l: (Split encoder, low-order bits)

Low order bits (see "h")

g: (Gray-code). This is a modifier that indicates that the following format string is gray-code encoded. This is only valid for encoders (e, h l) and unsigned (u) data types.

Parameters:

Two parameters is universally created for all SSI instances

(float r/w) hm2\_XiXX.N.ssi.MM.frequency-khz

This parameter sets the SSI clock frequency. The units are kHz, so 500 will give a clock frequency of 500,000 Hz.

(s32 r/w) hm2\_XiXX.N.ssi.MM.timer-num

This parameter allocates the SSI module to a specific hm2dp11 timer instance. This pin is only of use in firmwares which contain a hm2dp11 function and will default to 1 in cases where there is such a function, and 0 if there is not. The pin can be used to disable reads of the encoder, by setting to a nonexistent timer number, or to 0.

Other parameters depend on the data types specified in the config string.

p: (Pad) No Parameters.

b: (Boolean) No Parameters.

u: (Unsigned)

(float, r/w) hm2\_XiXX.N.ssi.MM.<name>.scalemax. The scaling factor for the channel.

s: (Signed)

(float, r/w) hm2\_XiXX.N.ssi.MM.<name>.scalemax. The scaling factor for the channel.

f: (bitField): No parameters.

e: (Encoder):

(float, r/w) hm2\_XiXX.N.ssi.MM.<name>.scale: (float, r/w) The encoder scale in counts per machine unit.

(u32, r/w) hm2\_XiXX.N.ssi.MM.<name>.counts-per-rev (u32, r/w) Used to emulate the index behaviour of an incremental+index encoder. This would normally be set to the

actual counts per rev of the encoder, but can be any whole number of revs. Integer divisors or multipliers of the true PPR might be useful for index-homing. Non-integer factors might be appropriate where there is a synchronous drive ratio between the encoder and the spindle or ballscrew.

## BiSS

BiSS is a bidirectional variant of SSI. Currently only a single direction is supported by LinuxCNC (encoder to PC).

One pin is created for each BiSS instance regardless of data format:

(bit, in) hm2\_XiXX.NN.biss.MM.data-incomplete This pin will be set "true" if the module was still transferring data when the value was read. When this problem exists there will also be a limited number of error messages printed to the UI. This pin should be used to monitor whether the problem has been addressed by config changes. Solutions to the problem depend on whether the encoder read is being triggered by the hm2dpll phase-locked-loop timer (described above) or by the trigger-encoders function (described below)

The names of the pins created by the BiSS module will depend entirely on the format string for each channel specified in the loadrt command line and follow closely the format defined above for SSI. Currently data packets of up to 96 bits are supported by the LinuxCNC driver, although the Mesa Hostmot2 module can handle 512 bit packets. It should be possible to extend the number of packets supported by the driver if there is a requirement to do so.

## Fanuc encoder.

The pins and format specifier for this module are identical to the SSI module described above, except that at least one pre-configured format is provided. A modparam of fanuc\_chan\_N=AA64 (case sensitive) will configure the channel for a Fanuc Aa64 encoder. The pins created are:

hm2_XiXX.N.fanuc.MM.batt	indicates battery state
hm2_XiXX.N.fanuc.MM.batt-not	inverted version of above
hm2_XiXX.N.fanuc.MM.comm	The 0-1023 absolute output for motor commutation
hm2_XXiX.N.fanuc.MM.crc	The CRC checksum. Currently HAL has no way to use this
hm2_XiXX.N.fanuc.MM.encoder.count	Encoder counts
hm2_XiXX.N.fanuc.MM.encoder.index-enable	Simulated index. Set by counts-per-rev parameter
hm2_XiXX.N.fanuc.MM.encoder.position	Counts scaled by the ...scale paramter
hm2_XiXX.N.fanuc.MM.encoder.rawcounts	Raw counts, unaffected by reset or index
hm2_XiXX.N.fanuc.MM.encoder.reset	If high/true then counts and position = 0
hm2_XiXX.N.fanuc.MM.valid	Indicates that the absolute position is valid
hm2_XiXX.N.fanuc.MM.valid-not	Inverted version

## resolver

Resolvers have names like hm2\_<BoardType>.<BoardNum>.resolver.<Instance>. <Instance> is a 2-digit number, which for the 7i49 board will be between 00 and 05. This function only works with the Mesa Resolver interface boards (of which the 7i49 is the only example at the time of writing). This board uses an SPI interface to the FPGA card, and will only work with the correct firmware. The pins allocated will be listed in the dmesg output, but are unlikely to be usefully probed with HAL tools.

Pins:

(float, out) angle

This pin indicates the angular position of the resolver. It is a number between 0 and 1 for each electrical rotation.

(float, out) position

Calculated from the number of complete and partial revolutions since startup, reset, or index-reset multiplied by the scale parameter.

(float, out) velocity

Calculated from the rotational velocity and the velocity-scale parameter. The default scale is electrical rotations per second.

(s32, out) count

This pins outputs a simulated encoder count at  $2^{24}$  counts per rev (16777216 counts).

(s32, out) rawcounts

This is identical to the counts pin, except it is not reset by the 'index' or 'reset' pins. This is the pin which would be linked to the bldc HAL component if the resolver was being used to commutate a motor.

(bit, in) reset

Resets the position and counts pins to zero immediately.

(bit, in/out) index-enable

When this pin is set high the position and counts pins will be reset the next time the resolver passes through the zero position. At the same time the pin is driven low to indicate to connected modules that the index has been seen, and that the counters have been reset.

(bit, out) error

Indicates an error in the particular channel. If this value is "true" then the reported position and velocity are invalid.

Parameters:

(float, read/write) scale

The position scale, in machine units per resolver electrical revolution.

(float, read/write) velocity-scale

The conversion factor between resolver rotation speed and machine velocity. A value of 1 will typically give motor speed in rps, a value of 0.01666667 will give (approximate) RPM.

(u32, read/write) index-divisor (default 1)

The resolver component emulates an index at a fixed point in the sin/cos cycle. Some resolvers have multiple cycles per rev (often related to the number of pole-pairs on the attached motor). LinuxCNC requires an index once per revolution for proper threading etc. This parameter should be set to the number of cycles per rev of the resolver. CAUTION: Which pseudo-index is used will not necessarily be consistent between LinuxCNC runs. Do not expect to re-start a thread after restarting LinuxCNC. It is not appropriate to use this parameter for index-homing of axis drives.

(float, read/write) excitation-khz

This pin sets the excitation frequency for the resolver. This pin is module-level rather than instance-level as all resolvers share the same excitation frequency.

Valid values are 10 (~10kHz), 5 (~5kHz) and 2.5 (~2.5kHz). The actual frequency depends on the FPGA frequency, and they correspond to `CLOCK_LOW/5000`, `CLOCK_LOW/10000` and `CLOCK_LOW/20000` respectively. The parameter will be set to the closest available of the three frequencies.

A value of -1 (the default) indicates that the current setting should be retained.

## pwmgen

pwmgens have names like "hm2\_<BoardType>.<BoardNum>.pwmgen.<Instance>". "Instance" is a two-digit number that corresponds to the HostMot2 pwmgen instance number. There are 'num\_pwmgens' instances, starting with 00.

So, for example, the HAL pin that enables output from the fourth pwmgen of the first 7i43 board is: hm2\_7i43.0.pwmgen.03.enable (this assumes that the firmware in that board is configured so that this HAL object is available)

In HM2, each pwmgen uses three output IO pins: Not-Enable, Out0, and Out1.

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

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

Pins:

(bit input) enable

If true, the pwmgen will set its Not-Enable pin false and output its pulses. If 'enable' is false, pwmgen will set its Not-Enable pin true and not output any signals.

(float input) value

The current pwmgen command value, in arbitrary units.

Parameters:

(float rw) scale

Scaling factor to convert 'value' from arbitrary units to duty cycle:  $dc = value / scale$ . Duty cycle has an effective range of -1.0 to +1.0 inclusive, anything outside that range gets clipped. The default scale is 1.0.

(s32 rw) output-type

This emulates the `output_type` load-time argument to the software pwmgen component. This parameter may be changed at runtime, but most of the time you probably want to set it at startup and then leave it alone. Accepted values are 1 (PWM on Out0 and Direction on Out1), 2 (Up on Out0 and Down on Out1), 3 (PDM mode, PDM on Out0 and Dir on Out1), and 4 (Direction on Out0 and PWM on Out1, "for locked antiphase").

In addition to the per-instance HAL Parameters listed above, there are a couple of HAL Parameters that affect all the pwmgen instances:

**(u32 rw) pwm\_frequency**

This specifies the PWM frequency, in Hz, of all the pwmgen instances running in the PWM modes (modes 1 and 2). This is the frequency of the variable-duty-cycle wave. Its effective range is from 1 Hz up to 193 kHz. Note that the max frequency is determined by the ClockHigh frequency of the Anything IO board; the 5i20 and 7i43 both have a 100 MHz clock, resulting in a 193 kHz max PWM frequency. Other boards may have different clocks, resulting in different max PWM frequencies. If the user attempts to set the frequency too high, it will be clipped to the max supported frequency of the board. Frequencies below about 5 Hz are not terribly accurate, but above 5 Hz they're pretty close. The default pwm\_frequency is 20,000 Hz (20 kHz).

**(u32 rw) pdm\_frequency**

This specifies the PDM frequency, in Hz, of all the pwmgen instances running in PDM mode (mode 3). This is the "pulse slot frequency"; the frequency at which the pdm generator in the AnyIO board chooses whether to emit a pulse or a space. Each pulse (and space) in the PDM pulse train has a duration of 1/pdm\_frequency seconds. For example, setting the pdm\_frequency to 2e6 (2 MHz) and the duty cycle to 50% results in a 1 MHz square wave, identical to a 1 MHz PWM signal with 50% duty cycle. The effective range of this parameter is from about 1525 Hz up to just under 100 MHz. Note that the max frequency is determined by the ClockHigh frequency of the Anything IO board; the 5i20 and 7i43 both have a 100 MHz clock, resulting in a 100 Mhz max PDM frequency. Other boards may have different clocks, resulting in different max PDM frequencies. If the user attempts to set the frequency too high, it will be clipped to the max supported frequency of the board. The default pdm\_frequency is 20,000 Hz (20 kHz).

**3ppwmgen**

Three-Phase PWM generators (3pwmgens) are intended for controlling the high-side and low-side gates in a 3-phase motor driver. The function is included to support the Mesa motor controller daughter-cards but can be used to control an IGBT or similar driver directly. 3pwmgens have names like "hm2\_<BoardType>.<BoardNum>.3pwmgen.<Instance>" where <Instance> is a 2-digit number. There will be num\_3pwmgens instances, starting at 00. Each instance allocates 7 output and one input pins on the Mesa card connectors. Outputs are: PWM A, PWM B, PWM C, /PWM A, /PWM B, /PWM C, Enable. The first three pins are the high side drivers, the second three are their complementary low-side drivers. The enable bit is intended to control the servo amplifier. The input bit is a fault bit, typically wired to over-current detection. When set the PWM generator is disabled. The three phase duty-cycles are individually controllable from -Scale to +Scale. Note that 0 corresponds to a 50% duty cycle and this is the inialization value.

**Pins:**

(float input) A-value, B-value, C-value: The PWM command value for each phase, limited to +/- "scale". Defaults to zero which is 50% duty cycle on high-side and low-sidepins (but see the "deadtime" parameter)

**(bit input) enable**

When high the PWM is enabled as long as the fault bit is not set by the external fault input pin. When low the PWM is disabled, with both high- side and low-side drivers low. This is not the same as 0 output (50% duty cycle on both sets of pins) or negative full scale (where the low side drivers are "on" 100% of the time)

**(bit output) fault**

Indicates the status of the fault bit. This output latches high once set by the physical fault pin until the "enable" pin is set to high.

## Parameters:

(u32 rw) **deadtime**

Sets the dead-time between the high-side driver turning off and the low-side driver turning on and vice-versa. Deadtime is subtracted from on time and added to off time symmetrically. For example with 20 kHz PWM (50 uSec period), 50% duty cycle and zero dead time, the PWM and NPWM outputs would be square waves (NPWM being inverted from PWM) with high times of 25 uS. With the same settings but 1 uS of deadtime, the PWM and NPWM outputs would both have high times of 23 uS (25 - (2X 1 uS), 1 uS per edge). The value is specified in nS and defaults to a rather conservative 5000nS. Setting this parameter to too low a value could be both expensive and dangerous as if both gates are open at the same time there is effectively a short circuit across the supply.

(float rw) **scale**

Sets the half-scale of the specified 3-phase PWM generator. PWM values from -scale to +scale are valid. Default is +/- 1.0

(bit rw) **fault-invert**

Sets the polarity of the fault input pin. A value of 1 means that a fault is triggered with the pin high, and 0 means that a fault is triggered when the pin is pulled low. Default 0, fault = low so that the PWM works with the fault pin unconnected.

(u32 rw) **sample-time**

Sets the time during the cycle when an ADC pulse is generated. 0 = start of PWM cycle and 1 = end. Not currently useful to LinuxCNC. Default 0.5.

In addition the per-instance parameters above there is the following parameter that affects all instances

(u32 rw) **frequency**

Sets the master PWM frequency. Maximum is approx 48kHz, minimum is 1kHz. Defaults to 20kHz.

**stepgen**

stepgens have names like "hm2\_<BoardType>.<BoardNum>.stepgen.<Instance>". "Instance" is a two-digit number that corresponds to the HostMot2 stepgen instance number. There are 'num\_stepgens' instances, starting with 00.

So, for example, the HAL pin that has the current position feedback from the first stepgen of the second 5i22 board is: hm2\_5i22.1.stepgen.00.position-fb (this assumes that the firmware in that board is configured so that this HAL object is available)

Each stepgen uses between 2 and 6 IO pins. The signals on these pins depends on the step\_type parameter (described below).

The stepgen representation is modeled on the stepgen software component. Each stepgen instance has the following pins and parameters:

Pins:

(float input) position-cmd

Target position of stepper motion, in arbitrary position units. This pin is only used when the stepgen is in position control mode (control-type=0).

(float input) velocity-cmd

Target velocity of stepper motion, in arbitrary position units per second. This pin is only used when the stepgen is in velocity control mode (control-type=1).

(s32 output) counts

Feedback position in counts (number of steps).

(float output) position-fb

Feedback position in arbitrary position units. This is similar to "counts/position\_scale", but has finer than step resolution.

(float output) velocity-fb

Feedback velocity in arbitrary position units per second.

(bit input) enable

This pin enables the step generator instance. When True, the stepgen instance works as expected. When False, no steps are generated and velocity-fb goes immediately to 0. If the stepgen is moving when enable goes false it stops immediately, without obeying the maxaccel limit.

(bit input) control-type

Switches between position control mode (0) and velocity control mode (1). Defaults to position control (0).

Parameters:

(float r/w) position-scale

Converts from counts to position units.  $\text{position} = \text{counts} / \text{position\_scale}$

(float r/w) maxvel

Maximum speed, in position units per second. If set to 0, the driver will always use the maximum possible velocity based on the current step timings and position-scale. The max velocity will change if the step timings or position-scale changes. Defaults to 0.

(float r/w) maxaccel

Maximum acceleration, in position units per second per second. Defaults to 1.0. If set to 0, the driver will not limit its acceleration at all - this requires that the position-cmd or velocity-cmd pin is driven in a way that does not exceed the machine's capabilities. This is probably what you want if you're going to be using the LinuxCNC trajectory planner to jog or run G-code.

(u32 r/w) steplen

Duration of the step signal, in nanoseconds.

(u32 r/w) stepspace

Minimum interval between step signals, in nanoseconds.

(u32 r/w) dirsetup

Minimum duration of stable Direction signal before a step begins, in nanoseconds.

(u32 r/w) dirhold

Minimum duration of stable Direction signal after a step ends, in nanoseconds.

(u32 r/w) step\_type

Output format, like the step\_type modparam to the software stegen(9) component. 0 = Step/Dir, 1 = Up/Down, 2 = Quadrature, 3+ = table-lookup mode. In this mode the step\_type parameter determines how long the step sequence is. Additionally the stepgen\_width parameter in the loadrt config string must be set to suit the number of pins per stepgen required. Any stepgen pins above this number will be available for GPIO. This mask defaults to 2. The maximum length is 16. Note that Table mode is not enabled in all firmwares but if you see GPIO pins between the stepgen instances in the dmesg/log hardware pin list then the option may be available.

In Quadrature mode (step\_type=2), the stepgen outputs one complete Gray cycle (00 â 01 â 11 â 10 â 00) for each "step" it takes. In table mode up to 6 IO pins are individually controlled in an arbitrary sequence up to 16 phases long.

(u32 r/w) table-data-N

There are 4 table-data-N parameters, table-data-0 to table-data-3. These each contain 4 bytes corresponding to 4 stages in the step sequence. For example table-data-0 = 0x00000001 would set stepgen pin 0 (always called "Step" in the dmesg output) on the first phase of the step sequence, and table-data-4 = 0x20000000 would set stepgen pin 6 ("Table5Pin" in the dmesg output) on the 16th stage of the step sequence.

(s32 r/w) hm2\_XiXX.N.stepgen.MM.timer-num (default: -1)

Sets the hm2dpll timer instance to be used to latch stepgen counts. A setting of -1 does not latch encoder counts. A setting of 0 latches at the same time as the main hostmot2 write. A setting of 1..4 uses a time offset from the main hostmot2 write according to the dpll's timer-us setting.

Typically, timer-us should be a negative number with a magnitude larger than the largest latency (e.g., -100 for a system with mediocre latency, -50 for a system with good latency).

If no DPLL module is present in the FPGA firmware, or if the stepgen module does not support DPLL, then this pin is not created.

When available, this feature should typically be enabled. Doing so generally reduces following errors.

## Smart Serial Interface

The Smart Serial Interface allows up to 32 different devices such as the Mesa 8i20 2.2kW 3-phase drive or 7i64 48-way IO cards to be connected to a single FPGA card. The driver auto-detects the connected hardware port, channel and device type. Devices can be connected in any order to any active channel of an active port. (see the config modparam definition above).

For full details of the smart-serial devices see **man sserial**.

## BSPI

The BSPI (Buffered SPI) driver is unusual in that it does not create any HAL pins. Instead the driver exports a set of functions that can be used by a sub-driver for the attached hardware. Typically these would be written in the "comp" pre-processing language: see [http://linuxcnc.org/docs/html/hal\\_comp.html](http://linuxcnc.org/docs/html/hal_comp.html) or **man comp** for further details. See **man mesa\_7i65** and the source of **mesa\_7i65.comp** for details of a typical sub-driver. See **man hm2\_bsp\_i\_setup\_chan**, **man hm2\_bsp\_i\_write\_chan**, **man hm2\_tram\_add\_bsp\_i\_frame**, **man hm2\_allocate\_bsp\_i\_tram**, **man hm2\_bsp\_i\_set\_read\_funtion** and **man hm2\_bsp\_i\_set\_write\_function** for the exported functions.

The names of the available channels are printed to standard output during the driver loading process and take the form **hm2\_<board name>.<board index>.bsp\_i.<index>** For example **hm2\_5i23.0.bsp\_i.0**

## UART

The UART driver also does not create any HAL pins, instead it declares two simple read/write functions and a setup function to be utilised by user-written code. Typically this would be written in the "comp" pre-processing language: see [http://linuxcnc.org/docs/html/hal\\_comp.html](http://linuxcnc.org/docs/html/hal_comp.html) or **man comp** for further details. See **man mesa\_uart** and the source of **mesa\_uart.comp** for details of a typical sub-driver. See **man hm2\_uart\_setup\_chan**, **man hm2\_uart\_send**, **man hm2\_uart\_read** and **man hm2\_uart\_setup**.

The names of the available uart channels are printed to standard output during the driver loading process and take the form **hm2\_<board name>.<board index>.uart.<index>** For example **hm2\_5i23.0.uart.0**

## General Purpose I/O

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

GPIOs have names like **"hm2\_<BoardType>.<BoardNum>.gpio.<IONum>"**. IONum is a three-digit number. The mapping from IONum to connector and pin-on-that-connector is written to the syslog when the driver loads, and it's documented in Mesa's manual for the Anything I/O boards.

So, for example, the HAL pin that has the current inverted input value read from GPIO 012 of the second 7i43 board is: **hm2\_7i43.1.gpio.012.in-not** (this assumes that the firmware in that board is configured so that this HAL object is available)

The HAL parameter that controls whether the last GPIO of the first 5i22 is an input or an output is: **hm2\_5i22.0.gpio.095.is\_output** (this assumes that the firmware in that board is configured so that this HAL object is available)

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

(bit out) in & in\_not: State (normal and inverted) of the hardware input pin. Both full GPIO pins and IO pins used as inputs by active module instances have these pins.

(bit in) out

Value to be written (possibly inverted) to the hardware output pin. Only full GPIO pins have this pin.

Each GPIO can have the following Parameters:

(bit r/w) is\_output

If set to 0, the GPIO is an input. The IO pin is put in a high-impedance state (weakly pulled high), to be driven by other devices. The logic value on the IO pin is available in the "in" and "in\_not" HAL pins. Writes to the "out" HAL pin have no effect. If this parameter is set to 1, the GPIO is an output; its behavior then depends on the "is\_opendrain" parameter. Only full GPIO pins have this parameter.

(bit r/w) is\_opendrain

This parameter only has an effect if the "is\_output" parameter is true. If this parameter is false, the GPIO behaves as a normal output pin: the IO pin on the connector is driven to the value specified by the "out" HAL pin (possibly inverted), and the value of the "in" and "in\_not" HAL pins is undefined. If this parameter is true, the GPIO behaves as an open-drain pin. Writing 0 to the "out" HAL pin drives the IO pin low, writing 1 to the "out" HAL pin puts the IO pin in a high-impedance state. In this high-impedance state the IO pin floats (weakly pulled high), and other devices can drive the value; the resulting value on the IO pin is available on the "in" and "in\_not" pins. Only full GPIO pins and IO pins used as outputs by active module instances have this parameter.

(bit r/w) invert\_output

This parameter only has an effect if the "is\_output" parameter is true. If this parameter is true, the output value of the GPIO will be the inverse of the value on the "out" HAL pin. Only full GPIO pins and IO pins used as outputs by active module instances have this parameter.

## led

Creates HAL pins for the LEDs on the FPGA board.

Pins:

(bit in) CR<NN>

The pins are numbered from CR01 upwards with the name corresponding to the PCB silkscreen. Setting the bit to "true" or 1 lights the led.

## Watchdog

The HostMot2 firmware may include a watchdog Module; if it does, the hostmot2 driver will use it. The HAL representation of the watchdog is named "hm2\_<BoardType>.<Board-Num>.watchdog".

The watchdog starts out asleep and inactive. Once you access the board the first time by running the hm2 write() HAL function (see below), the watchdog wakes up. From then on it must be petted periodically or it will bite. Pet the watchdog by running the hm2 write() HAL function.

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

generating signals (which are *\*not\** relayed to the motors, because the IO Pins have become inputs).

Resetting the watchdog (by clearing the `has_bit` pin, see below) resumes communication and resets the I/O pins to the configuration chosen at load-time.

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

Pins:

(bit in/out) `has_bit`

True if the watchdog has bit, False if the watchdog has not bit. If the watchdog has bit and the `has_bit` bit is True, the user can reset it to False to resume operation.

Parameters:

(u32 read/write) `timeout_ns`

Watchdog timeout, in nanoseconds. This is initialized to 5,000,000 (5 milliseconds) at module load time. If more than this amount of time passes between calls to the `hm2_write()` function, the watchdog will bite.

## Raw Mode

If the "enable\_raw" config keyword is specified, some extra debugging pins are made available in HAL. The raw mode HAL pin names begin with "hm2\_<BoardType>.<BoardNum>.raw".

With Raw mode enabled, a user may peek and poke the firmware from HAL, and may dump the internal state of the hostmot2 driver to the syslog.

Pins:

(u32 in) `read_address`

The bottom 16 bits of this is used as the address to read from.

(u32 out) `read_data`

Each time the `hm2_read()` function is called, this pin is updated with the value at `.read_address`.

(u32 in) `write_address`

The bottom 16 bits of this is used as the address to write to.

(u32 in) `write_data`

This is the value to write to `.write_address`.

(bit in) `write_strobe`

Each time the `hm2_write()` function is called, this pin is examined. If it is True, then value in `.write_data` is written to the address in `.write_address`, and `.write_strobe` is set back to False.

(bit in/out) `dump_state`

This pin is normally False. If it gets set to True the hostmot2 driver will write its representation of the board's internal state to the syslog, and set the pin back to False.

## Setting up Smart Serial devices

See man setsserial for the current way to set smart-serial eeprom parameters.

## FUNCTIONS

### **hm2\_<BoardType>.<BoardNum>.read-request**

On boards with long turn around time for reads (at the time of writing, this applies only to ethernet boards), this function sends a read request. When multiple boards are used, this can reduce the servo thread execution time. In this case, the appropriate thread order would be

```
addf hm2_7i80.0.read-request
addf hm2_7i80.1.read-request
addf hm2_7i80.0.read
addf hm2_7i80.1.read
```

which causes the read request to be sent to board 1 before waiting for the response to the read request to arrive from board 0.

### **hm2\_<BoardType>.<BoardNum>.read**

This reads the encoder counters, stepgen feedbacks, and GPIO input pins from the FPGA.

### **hm2\_<BoardType>.<BoardNum>.write**

This updates the PWM duty cycles, stepgen rates, and GPIO outputs on the FPGA. Any changes to configuration pins such as stepgen timing, GPIO inversions, etc, are also effected by this function.

### **hm2\_<BoardType>.<BoardNum>.read\_gpio**

Read the GPIO input pins. Note that the effect of this function is a subset of the effect of the .read() function described above. Normally only .read() is used. The only reason to call this function is if you want to do GPIO things in a faster-than-servo thread. (This function is not available on the 7i43 due to limitations of the EPP bus.)

### **hm2\_<BoardType>.<BoardNum>.write\_gpio**

Write the GPIO control registers and output pins. Note that the effect of this function is a subset of the effect of the .write() function described above. Normally only .write() is used. The only reason to call this function is if you want to do GPIO things in a faster-than-servo thread. (This function is not available on the 7i43 due to limitations of the EPP bus.)

### **hm2\_<BoardType>.<BoardNum>.trigger-encoders**

This function will only appear if the firmware contains a BiSS, Fanuc or SSI encoder module and if the firmware does not contain a hm2dp11 module (qv) or if the modparam contains num\_dp11s=0. This function should be inserted first in the thread so that the encoder data is ready when the main **hm2\_XiXX.NN.read** function runs. An error message will be printed if the encoder read is not finished in time. It may be possible to avoid this by increasing the data rate. If the problem persists and if "stale" data is acceptable then the function may be placed later in the thread, allowing a full servo cycle for the data to be transferred from the devices. If available it is better to use the synchronous hm2dp11 triggering function.

## SEE ALSO

hm2\_7i43(9)

hm2\_pci(9)

Mesa's documentation for the Anything I/O boards, at <<http://www.mesanet.com>>

## LICENSE

GPL

**NAME**

hypot – Three-input hypotenuse (Euclidean distance) calculator

**SYNOPSIS**

**loadrt hypot** [**count**=*N* | **names**=*name1* [, *name2* ...]]

**FUNCTIONS**

**hypot.N** (requires a floating-point thread)

**PINS**

**hypot.N.in0** float in

**hypot.N.in1** float in

**hypot.N.in2** float in

**hypot.N.out** float out

$out = \sqrt{in0^2 + in1^2 + in2^2}$

**LICENSE**

GPL

**NAME**

ilowpass – Low-pass filter with integer inputs and outputs

**SYNOPSIS**

**loadrt ilowpass** [**count**=*N* | **names**=*name1* [, *name2*...]]

**DESCRIPTION**

While it may find other applications, this component was written to create smoother motion while jogging with an MPG.

In a machine with high acceleration, a short jog can behave almost like a step function. By putting the **ilowpass** component between the MPG encoder **counts** output and the axis jog-counts input, this can be smoothed.

Choose **scale** conservatively so that during a single session there will never be more than about  $2e9/\text{scale}$  pulses seen on the MPG. Choose **gain** according to the smoothing level desired. Divide the axis.*N*.jog-scale values by **scale**.

**FUNCTIONS**

**ilowpass.N** (requires a floating-point thread)

Update the output based on the input and parameters

**PINS**

**ilowpass.N.in** s32 in

**ilowpass.N.out** s32 out

**out** tracks **in**\***scale** through a low-pass filter of **gain** per period.

**PARAMETERS**

**ilowpass.N.scale** float rw (default: 1024)

A scale factor applied to the output value of the low-pass filter.

**ilowpass.N.gain** float rw (default: .5)

Together with the period, sets the rate at which the output changes. Useful range is between 0 and 1, with higher values causing the input value to be tracked more quickly. For instance, a setting of 0.9 causes the output value to go 90% of the way towards the input value in each period

**AUTHOR**

Jeff Epler <jepler@unpythonic.net>

**LICENSE**

GPL

**NAME**

integ – Integrator with gain pin and windup limits

**SYNOPSIS**

**loadrt** integ [**count**=*N* | **names**=*name1* [, *name2* ... ]]

**FUNCTIONS**

**integ.N** (*N* requires a floating-point thread)

**PINS**

**integ.N.in** float in

**integ.N.gain** float in (default: *1.0*)

**integ.N.out** float out

The discrete integral of 'gain \* in' since 'reset' was deasserted

**integ.N.reset** bit in

When asserted, set out to 0

**integ.N.max** float in (default: *1e20*)

**integ.N.min** float in (default: *-1e20*)

**LICENSE**

GPL

**NAME**

invert – Compute the inverse of the input signal

**SYNOPSIS**

The output will be the mathematical inverse of the input, ie **out** = 1/**in**. The parameter **deadband** can be used to control how close to 0 the denominator can be before the output is clamped to 0. **deadband** must be at least 1e-8, and must be positive.

**FUNCTIONS**

**invert.N** (requires a floating-point thread)

**PINS**

**invert.N.in** float in  
Analog input value

**invert.N.out** float out  
Analog output value

**PARAMETERS**

**invert.N.deadband** float rw  
The **out** will be zero if **in** is between **-deadband** and **+deadband**

**LICENSE**

GPL

**NAME**

joyhandle – sets nonlinear joypad movements, deadbands and scales

**SYNOPSIS**

**loadrt joyhandle** [**count**=*N* | **names**=*name1* [, *name2* ... ]]

**DESCRIPTION**

The component **joyhandle** uses the following formula for a non linear joypad movements:

$$y = (\text{scale} * (a * x^{\text{power}} + b * x)) + \text{offset}$$

The parameters *a* and *b* are adjusted in such a way, that the function starts at (deadband, offset) and ends at (1, scale+offset).

Negative values will be treated point symetrically to origin. Values  $-\text{deadband} < x < +\text{deadband}$  will be set to zero.

Values  $x > 1$  and  $x < -1$  will be skipped to  $\pm(\text{scale} + \text{offset})$ . Invert transforms the function to a progressive movement.

With power one can adjust the nonlinearity (default = 2). Default for deadband is 0.

Valid values are: power  $\geq 1.0$  (reasonable values are 1.x .. 4-5, take higher power-values for higher deadbands ( $>0.5$ ), if you want to start with a nearly horizontal slope),  $0 \leq \text{deadband} < 0.99$  (reasonable 0.1).

An additional offset component can be set in special cases (default = 0).

All values can be adjusted for each instance separately.

**FUNCTIONS**

**joyhandle.N** (requires a floating-point thread)

**PINS**

**joyhandle.N.in** float in

**joyhandle.N.out** float out

**PARAMETERS**

**joyhandle.N.power** float rw (default: 2.0)

**joyhandle.N.deadband** float rw (default: 0.)

**joyhandle.N.scale** float rw (default: 1.)

**joyhandle.N.offset** float rw (default: 0.)

**joyhandle.N.inverse** bit rw (default: 0)

**LICENSE**

GPL

**NAME**

kins – kinematics definitions for LinuxCNC

**SYNOPSIS**

**loadrt trivkins**

**loadrt rotatekins**

**loadrt tripodkins**

**loadrt genhexkins**

**loadrt maxkins**

**loadrt genserkins**

**loadrt pumakins**

**loadrt scarakins**

**DESCRIPTION**

Rather than exporting HAL pins and functions, these components provide the forward and inverse kinematics definitions for LinuxCNC.

**trivkins – Trivial Kinematics**

There is a 1:1 correspondence between joints and axes. Most standard milling machines and lathes use the trivial kinematics module.

**rotatekins – Rotated Kinematics**

The X and Y axes are rotated 45 degrees compared to the joints 0 and 1.

**tripodkins – Tripod Kinematics**

The joints represent the distance of the controlled point from three predefined locations (the motors), giving three degrees of freedom in position (XYZ)

**tripodkins.Bx**

**tripodkins.Cx**

**tripodkins.Cy**

The location of the three motors is (0,0), (Bx,0), and (Cx,Cy)

**genhexkins – Hexapod Kinematics**

Gives six degrees of freedom in position and orientation (XYZABC). The location of base and platform joints is defined by hal parameters. The forward kinematics iteration is controlled by hal pins.

**genhexkins.base.N.x**

**genhexkins.base.N.y**

**genhexkins.base.N.z**

**genhexkins.platform.N.x**

**genhexkins.platform.N.y**

**genhexkins.platform.N.z**

Parameters describing the Nth joint's coordinates.

**genhexkins.convergence-criterion**

Minimum error value that ends iterations with converged solution.

**genhexkins.limit-iterations**

Limit of iterations, if exceeded iterations stop with no convergence.

**genhexkins.max-error**

Maximum error value, if exceeded iterations stop with no convergence.

**genhexkins.last-iterations**

Number of iterations spent for the last forward kinematics solution.

**genhexkins.max-iterations**

Maximum number of iterations spent for a converged solution during current session.

**maxkins – 5-axis kinematics example**

Kinematics for Chris Radek's tabletop 5 axis mill named 'max' with tilting head (B axis) and horizontal rotary mounted to the table (C axis). Provides UVW motion in the rotated coordinate system. The source file, maxkins.c, may be a useful starting point for other 5-axis systems.

**genserkins – generalized serial kinematics**

Kinematics that can model a general serial-link manipulator with up to 6 angular joints.

The kinematics use Denavit-Hartenberg definition for the joint and links. The DH definitions are the ones used by John J Craig in "Introduction to Robotics: Mechanics and Control" The parameters for the manipulator are defined by hal pins.

**genserkins.A-*N***

**genserkins.ALPHA-*N***

**genserkins.D-*N***

Parameters describing the *N*th joint's geometry.

**pumakins – kinematics for puma typed robots**

Kinematics for a puma-style robot with 6 joints

**pumakins.A2**

**pumakins.A3**

**pumakins.D3**

**pumakins.D4**

Describe the geometry of the robot

**scarakins – kinematics for SCARA-type robots**

**scarakins.D1**

Vertical distance from the ground plane to the center of the inner arm.

**scarakins.D2**

Horizontal distance between joint[0] axis and joint[1] axis, ie. the length of the inner arm.

**scarakins.D3**

Vertical distance from the center of the inner arm to the center of the outer arm. May be positive or negative depending on the structure of the robot.

**scarakins.D4**

Horizontal distance between joint[1] axis and joint[2] axis, ie. the length of the outer arm.

**scarakins.D5**

Vertical distance from the end effector to the tooltip. Positive means the tooltip is lower than the end effector, and is the normal case.

**scarakins.D6**

Horizontal distance from the centerline of the end effector (and the joints 2 and 3 axis) and the tooltip. Zero means the tooltip is on the centerline. Non-zero values should be positive, if negative they introduce a 180 degree offset on the value of joint[3].

**SEE ALSO**

*Kinematics* section in the LinuxCNC documentation

**NAME**

knob2float – Convert counts (probably from an encoder) to a float value

**SYNOPSIS**

**loadrt knob2float** [**count**=*N*][**names**=*name1*[,*name2*...]]

**FUNCTIONS**

**knob2float.N** (requires a floating-point thread)

**PINS**

**knob2float.N.counts** s32 in  
Counts

**knob2float.N.enable** bit in  
When TRUE, output is controlled by count, when FALSE, output is fixed

**knob2float.N.scale** float in  
Amount of output change per count

**knob2float.N.out** float out  
Output value

**PARAMETERS**

**knob2float.N.max-out** float rw (default: *1.0*)  
Maximum output value, further increases in count will be ignored

**knob2float.N.min-out** float rw (default: *0.0*)  
Minimum output value, further decreases in count will be ignored

**LICENSE**

GPL

**NAME**

latencybins – comp utility for scripts/latency-histogram

**SYNOPSIS**

Usage:

Read availablebins pin for the number of bins available.

Set the maxbinnumber pin for the number of +/- bins.

Ensure maxbinnumber <= availablebins

For maxbinnumber = N, the bins are numbered:

-N ... 0 ... + N bins

(the -0 bin is not populated)

(total effective bins = 2\*maxbinnumber + 1)

Set nsbinsize pin for the binsize (ns)

Iterate:

Set index pin to a bin number: 0 <= index <= maxbinnumber.

Read check pin and verify that check pin == index pin.

Read output pins:

pbinvalue is count for bin = +index

nbinvalue is count for bin = -index

pextra is count for all bins > maxbinnumber

nextra is count for all bins < maxbinnumber

latency-min is max negative latency

latency-max is max positive latency

If index is out of range ( index < 0 or index > maxbinnumber)

then pbinvalue = nbinvalue = -1.

The reset pin may be used to restart.

The latency pin outputs the instantaneous latency.

Maintainers note: hardcoded for MAXBINNUMBER==1000

**FUNCTIONS**

latencybins.N

**PINS**

latencybins.N.maxbinnumber s32 in (default: 1000)

latencybins.N.index s32 in

latencybins.N.reset bit in

latencybins.N.nsbinsize s32 in

latencybins.N.check s32 out

latencybins.N.latency s32 out

latencybins.N.latency-max s32 out

latencybins.N.latency-min s32 out

latencybins.N.pbinvalue s32 out

latencybins.N.nbinvalue s32 out

latencybins.N.pextra s32 out

latencybins.N.nextra s32 out

latencybins.N.variance s32 out

latencybins.N.availablebins s32 out (default: 1000)

**LICENSE**

GPL

**NAME**

lcd – Stream HAL data to an LCD screen

**SYNOPSIS**

```
loadrt lcd fmt_strings=""Plain Text %4.4f\nAnd So on|Second Page, Next Inst""
```

**FUNCTIONS**

**lcd** (requires a floating-point thread). All LCD instances are updated by the same function.

**PINS**

**lcd.NN.out** (u32) out

The output byte stream is sent via this pin. One character is sent every thread invocation. There is no handshaking provided.

**lcd.NN.page.PP.arg.NN** (float/s32/u32/bit) in

The input pins have types matched to the format string specifiers.

**lcd.NN.page\_num** (u32) in

Selects the page number. Multiple layouts may be defined, and this pin switches between them.

**lcd.NN.contrast** (float) in

Attempts to set the contrast of the LCD screen using the byte sequence ESC C and then a value from 0x20 to 0xBF. (matching the Mesa 7i73). The value should be between 0 and 1.

**PARAMETERS**

**lcd.NN.decimal-separator** (u32) rw

Sets the decimal separator used for floating point numbers. The default value is 46 (0x2E) which corresponds to ".". If a comma is required then set this parameter to 44 (0x2C).

**DESCRIPTION**

**lcd** takes format strings much like those used in C and many other languages in the printf and scanf functions and their variants.

The component was written specifically to support the Mesa 7i73 pendant controller, however it may be of use streaming data to other character devices and, as the output format mimics the ADM3 terminal format, it could be used to stream data to a serial device. Perhaps even a genuine ADM3. The strings contain a mixture of text values which are displayed directly, "escaped" formatting codes and numerical format descriptors. For a detailed description of formatting codes see: <http://en.wikipedia.org/wiki/Printf>

The component can be configured to display an unlimited number of differently-formatted pages, which may be selected with a HAL pin.

**Escaped codes**

**\n** Inserts a clear-to-end, carriage return and line feed character. This will still linefeed and clear even if an automatic wrap has occurred (lcd has no knowledge of the width of the lcd display.) To print in the rightmost column it is necessary to allow the format to wrap and omit the **\n** code.

**\t** Inserts a tab (actually 4 spaces in the current version rather than a true tab.)

**\NN** inserts the character defined by the hexadecimal code NN.

**\\** Inserts a literal **\**.

**Numerical formats**

**lcd** differs slightly from the standard printf conventions. One significant difference is that width

limits are strictly enforced to prevent the LCD display wrapping and spoiling the layout. The field width includes the sign character so that negative numbers will often have a smaller valid range than positive. Numbers that do not fit in the specified width are displayed as a line of asterisks (\*\*\*\*\*).

Each format begins with a "%" symbol. (For a literal % use "% %"). Immediately after the % the following modifiers may be used:

" " (space) Pad the number to the specified width with spaces. This is the default and is not strictly necessary.

"0" Pad the number to the specified width with the numeral 0.

"+" Force display of a + symbol before positive numbers. This (like the - sign) will appear immediately to the left of the digits for a space-padded number and in the extreme left position for a 0-padded number.

"1234567890" A numerical entry (other than the leading 0 above) defines the total number of characters to display including the decimal separator and the sign. Whilst this number can be as many digits as required the maximum field width is 20 characters. The inherent precision of the "double" data type means that more than 14 digits will tend to show errors in the least significant digits. The integer data types will never fill more than 10 decimal digits.

Following the width specifier should be the decimal specifier. This can only be a full-stop character (.) as the comma (,) is used as the instance separator. Currently lcd does not access the locale information to determine the correct separator and the **decimal-separator** parameter should be used.

Following the decimal separator should be a number that determines how many places of decimals to display. This entry is ignored in the case of integer formats.

All the above modifiers are optional, but to specify a decimal precision the decimal point must precede the precision. For example %.3f.

The default decimal precision is 4.

The numerical formats supported are:

**%f %F** (for example, %+09.3f) These create a floating-point type HAL pin. The example would be displayed in a 9-character field, with 3 places of decimals, . as a decimal separator, padded to the left with 0s and with a sign displayed for both positive and negative. Conversely a plain %f would be 6 digits of decimal, variable format width, with a sign only shown for negative numbers. both %f and %F create exactly the same format.

**%i %d** (For example %+ 4d) Creates a signed (s32) HAL pin. The example would display the value at a fixed 4 characters, space padded, width including the + giving a range of +999 to -999. %i and %d create identical output.

**%u** (for example %08u) Creates an unsigned (u32) HAL pin. The example would be a fixed 8 characters wide, padded with zeros.

**%x, %X** Creates an unsigned (u32) HAL pin and displays the value in Hexadecimal. Both %x and %X display capital letters for digits ABCDEF. A width may be specified, though the u32 HAL type is only 8 hex digits wide.

**%o** Creates an unsigned (u32) pin and displays the value in Octal.

**%c** Creates a u32 HAL pin and displays the character corresponding to the value of the pin. Values less than 32 (space) are suppressed. A width specifier may be used, for example %20c might be used to create a complete line of one character.

**%b** This specifier has no equivalent in printf. It creates a bit (boolean) type HAL pin. The b should be followed by two characters and the display will show the first of these when the pin is true, and the second when false. Note that the characters follow, not precede the "b", unlike the case with other formats. The characters may be "escaped" Hex values. For example "%b\\xFF " will display a solid black block if true, and a space if false and "%b\\x7F\\x7E" would display right-arrow for false and left-arrow for true. An unexpected value of 'E' indicates a formatting error.

**Pages** The page separator is the "|" (pipe) character. (if the actual character is needed then \\7C may be used). A "Page" in this context refers to a separate format which may be displayed on the same display.

**Instances** The instance separator is the comma. This creates a completely separate lcd instance, for example to drive a second lcd display on the second 7i73. The use of comma to separate instances is built in to the modparam reading code so not even escaped commas "\\," can be used. A comma may be displayed by using the \\2C sequence.

## AUTHOR

Andy Pugh

## LICENSE

GPL

**NAME**

limit1 – Limit the output signal to fall between min and max

**SYNOPSIS**

**loadrt limit1** [**count**=*N*][**names**=*name1*[,*name2*...]]

**FUNCTIONS**

**limit1.N** (requires a floating-point thread)

**PINS**

**limit1.N.in** float in

**limit1.N.out** float out

**PARAMETERS**

**limit1.N.min** float rw (default: *-1e20*)

**limit1.N.max** float rw (default: *1e20*)

**LICENSE**

GPL

**NAME**

limit2 – Limit the output signal to fall between min and max and limit its slew rate to less than maxv per second. When the signal is a position, this means that position and velocity are limited.

**SYNOPSIS**

**loadrt limit2** [**count**=*N*][**names**=*name1*[,*name2*...]]

**FUNCTIONS**

**limit2.N** (requires a floating-point thread)

**PINS**

**limit2.N.in** float in

**limit2.N.out** float out

**limit2.N.load** bit in

When TRUE, immediately set **out to in**, ignoring maxv

**PARAMETERS**

**limit2.N.min** float rw (default: *-1e20*)

**limit2.N.max** float rw (default: *1e20*)

**limit2.N.maxv** float rw (default: *1e20*)

**LICENSE**

GPL

**NAME**

limit3 – Limit the output signal to fall between min and max, limit its slew rate to less than maxv per second, and limit its second derivative to less than maxa per second squared. When the signal is a position, this means that the position, velocity, and acceleration are limited.

**SYNOPSIS**

**loadrt limit3** [**count**=*N*][**names**=*name1*[,*name2*...]]

**FUNCTIONS**

**limit3.N** (requires a floating-point thread)

**PINS**

**limit3.N.in** float in

**limit3.N.out** float out

**limit3.N.load** bit in

When TRUE, immediately set **out to in**, ignoring maxv and maxa

**limit3.N.min** float in (default: *-1e20*)

**limit3.N.max** float in (default: *1e20*)

**limit3.N.maxv** float in (default: *1e20*)

**limit3.N.maxa** float in (default: *1e20*)

**LICENSE**

GPL

**NAME**

lincurve – one-dimensional lookup table

**SYNOPSIS**

**loadrt lincurve [count=*N*][names=*name1*[,*name2*...]] [personality=*P*,*P*,...]**

**DESCRIPTION**

This component can be used to map any floating-point input to a floating-point output. Typical uses would include linearisation of thermocouples, defining PID gains that vary with external factors or to substitute for any mathematical function where absolute accuracy is not required.

The component can be thought of as a 2-dimensional graph of points in (x,y) space joined by straight lines. The input value is located on the x axis, followed up until it touches the line, and the output of the component is set to the corresponding y-value.

The (x,y) points are defined by the x-val-NN and y-val-NN parameters which need to be set in the HAL file using "setp" commands.

The maximum number of (x,y) points supported is 16.

For input values less than the x-val-00 breakpoint the y-val-00 is returned. For x greater than the largest x-val-NN the yval corresponding to x-max is returned (ie, no extrapolation is performed.)

Sample usage: loadrt lincurve count=3 personality=4,4,4 for a set of three 4-element graphs.

**FUNCTIONS**

**lincurve.*N*** (requires a floating-point thread)

**PINS**

**lincurve.*N*.in** float in

The input value

**lincurve.*N*.out** float out

The output value

**lincurve.*N*.out-io** float io

The output value, compatible with PID gains

**PARAMETERS**

**lincurve.*N*.x-val-*MM*** float rw (*MM*=00..personality)  
axis breakpoints

**lincurve.*N*.y-val-*MM*** float rw (*MM*=00..personality)  
output values to be interpolated

**AUTHOR**

Andy Pugh

**LICENSE**

GPL

**NAME**

logic – LinuxCNC HAL component providing configurable logic functions

**SYNOPSIS**

**loadrt logic** [**count**=*N*][**names**=*name1*[,*name2*...]] [**personality**=*P,P*,...]

**DESCRIPTION**

General ‘logic function’ component. Can perform ‘and’, ‘or’ and ‘xor’ of up to 16 inputs.

Determine the proper value for ‘personality’ by adding the inputs and outputs then convert to hex:

- The number of input pins, usually from 2 to 16
- 256 (0x100) if the ‘and’ output is desired
- 512 (0x200) if the ‘or’ output is desired
- 1024 (0x400) if the ‘xor’ (exclusive or) output is desired

Outputs can be combined, for example 2 + 256 + 1024 = 1282 converted to hex would be 0x502 and would have two inputs and have both ‘xor’ and ‘and’ outputs.

**FUNCTIONS**

**logic.N**

**PINS**

**logic.N.in-MM** bit in (MM=00..personality & 0xff)

**logic.N.and** bit out [if personality & 0x100]

**logic.N.or** bit out [if personality & 0x200]

**logic.N.xor** bit out [if personality & 0x400]

**LICENSE**

GPL

**NAME**

lowpass – Low-pass filter

**SYNOPSIS**

**loadrt lowpass** [**count**=*N* | **names**=*name1* [, *name2* ... ]]

**FUNCTIONS**

**lowpass.N** (requires a floating-point thread)

**PINS**

**lowpass.N.in** float in

**lowpass.N.out** float out

out += (in - out) \* gain

**lowpass.N.load** bit in

When TRUE, copy **in** to **out** instead of applying the filter equation.

**PARAMETERS**

**lowpass.N.gain** float rw

**NOTES**

The effect of a specific **gain** value is dependent on the period of the function that **lowpass.N** is added to

**LICENSE**

GPL

**NAME**

lut5 – Arbitrary 5-input logic function based on a look-up table

**SYNOPSIS**

**loadrt lut5** [**count**=*N*][**names**=*name1*[,*name2*...]]

**DESCRIPTION**

**lut5** constructs a logic function with up to 5 inputs using a **look-up** table. The value for **function** can be determined by writing the truth table, and computing the sum of **all** the **weights** for which the output value would be TRUE. The weights are hexadecimal not decimal so hexadecimal math must be used to sum the weights. A wiki page has a calculator to assist in computing the proper value for function.

<http://wiki.linuxcnc.org/cgi-bin/wiki.pl?Lut5>

Note that LUT5 will generate any of the 4,294,967,296 logical functions of 5 inputs so **AND**, **OR**, **NAND**, **NOR**, **XOR** and every other combinatorial function is possible.

**Example Functions**

A 5-input *and* function is TRUE only when all the inputs are true, so the correct value for **function** is **0x80000000**.

A 2-input *or* function would be the sum of **0x2** + **0x4** + **0x8**, so the correct value for **function** is **0xe**.

A 5-input *or* function is TRUE whenever any of the inputs are true, so the correct value for **function** is **0xffffffff**. Because every weight except **0x1** is true the function is the sum of every line except the first one.

A 2-input *xor* function is TRUE whenever exactly one of the inputs is true, so the correct value for **function** is **0x6**. Only **in-0** and **in-1** should be connected to signals, because if any other bit is **TRUE** then the output will be **FALSE**.

Weights for each line of truth table					
Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Weight
0	0	0	0	0	0x1
0	0	0	0	1	0x2
0	0	0	1	0	0x4
0	0	0	1	1	0x8
0	0	1	0	0	0x10
0	0	1	0	1	0x20
0	0	1	1	0	0x40
0	0	1	1	1	0x80
0	1	0	0	0	0x100
0	1	0	0	1	0x200
0	1	0	1	0	0x400
0	1	0	1	1	0x800
0	1	1	0	0	0x1000
0	1	1	0	1	0x2000
0	1	1	1	0	0x4000
0	1	1	1	1	0x8000
1	0	0	0	0	0x10000
1	0	0	0	1	0x20000
1	0	0	1	0	0x40000
1	0	0	1	1	0x80000
1	0	1	0	0	0x100000
1	0	1	0	1	0x200000
1	0	1	1	0	0x400000
1	0	1	1	1	0x800000
1	1	0	0	0	0x1000000
1	1	0	0	1	0x2000000
1	1	0	1	0	0x4000000
1	1	0	1	1	0x8000000
1	1	1	0	0	0x10000000
1	1	1	0	1	0x20000000
1	1	1	1	0	0x40000000
1	1	1	1	1	0x80000000

**FUNCTIONS****lut5.N****PINS****lut5.N.in-0** bit in**lut5.N.in-1** bit in**lut5.N.in-2** bit in**lut5.N.in-3** bit in**lut5.N.in-4** bit in**lut5.N.out** bit out**PARAMETERS****lut5.N.function** u32 rw**LICENSE**

GPL

**NAME**

maj3 – Compute the majority of 3 inputs

**SYNOPSIS**

**loadrt maj3** [**count**=*N* | **names**=*name1* [, *name2* ... ]]

**FUNCTIONS**

**maj3.N**

**PINS**

**maj3.N.in1** bit in

**maj3.N.in2** bit in

**maj3.N.in3** bit in

**maj3.N.out** bit out

**PARAMETERS**

**maj3.N.invert** bit rw

**LICENSE**

GPL

**NAME**

match8 – 8-bit binary match detector

**SYNOPSIS**

**loadrt match8 [count=*N*|names=*name1*[,*name2*...]]**

**FUNCTIONS**

**match8.*N***

**PINS**

**match8.*N*.in** bit in (default: *TRUE*)

cascade input - if false, output is false regardless of other inputs

**match8.*N*.a0** bit in

**match8.*N*.a1** bit in

**match8.*N*.a2** bit in

**match8.*N*.a3** bit in

**match8.*N*.a4** bit in

**match8.*N*.a5** bit in

**match8.*N*.a6** bit in

**match8.*N*.a7** bit in

**match8.*N*.b0** bit in

**match8.*N*.b1** bit in

**match8.*N*.b2** bit in

**match8.*N*.b3** bit in

**match8.*N*.b4** bit in

**match8.*N*.b5** bit in

**match8.*N*.b6** bit in

**match8.*N*.b7** bit in

**match8.*N*.out** bit out

true only if in is true and a[m] matches b[m] for m = 0 thru 7

**LICENSE**

GPL

**NAME**

`matrix_kb` – Convert integers to HAL pins. Optionally scan a matrix of IO ports to create those integers.

**SYNOPSIS**

**loadrt matrix\_kb config=*RxCs,RxCs...* names=*name1,name2...***

Creates a component configured for R rows and N columns of matrix keyboard.

If the **s** option is specified then a set of output rows will be cyclically toggled, and a set of input columns will be scanned.

The **names** parameter is optional, but if used then the HAL pins and functions will use the specified names rather than the default ones. This can be useful for readability and 2-pass HAL parsing.

There must be no spaces in the parameter lists.

**DESCRIPTION**

This component was written to convert matrix keyboard scancodes into HAL pins. However, it might also find uses in converting integers from 0 to N into N HAL pins.

The component can work in two ways, and the HAL pins created vary according to mode.

In the default mode the component expects to be given a scan code from a separate driver but could be any integer from any source. Most typically this will be the keypad scancode from a Mesa 7i73. The default codes for keyup and keydown are based on the Mesa 7i73 specification with 0x40 indicating a keydown and 0x80 a keyup event.

If using the 7i73 it is important to match the keypad size jumpers with the HAL component. Valid configs for the 7i73 are 4x8 and 8x8. Note that the component will only work properly with the version 12 (0xC) 7i73 firmware. The firmware version is visible on the component parameters in HAL.

In the optional scan-generation mode the **matrix\_kb.N.keycode** pin changes to an output pin and a set of output row pins and input column pins are created. These need to be connected to physical inputs and outputs to scan the matrix and return values to HAL. Note the **negative-logic** parameter described below, this will need to be set on the most common forms of inputs which float high when unconnected.

In both modes a set of HAL output pins are created corresponding to each node of the matrix.

**FUNCTIONS**

**matrix\_kb.N**

Perform all requested functions. Should be run in a slow thread for effective debouncing.

**PINS**

**matrix\_kb.N.col-CC-in** bit in

The input pin corresponding to column C.

**matrix\_kb.N.key.rRcC** bit out

The pin corresponding to the key at row R column C of the matrix.

**matrix\_kb.N.keycode** unsigned in or out depending on mode.

This pin should be connected to the scancode generator if hardware such as a 7i73 is being used. In this mode it is an input pin. In the internally-generated scanning mode this pin is an output, but will not normally be connected. **matrix\_kb.N.row-RR-out bit out** The row scan drive pins. Should be connected to external hardware pins connected to the keypad.

## PARAMETERS

**matrix\_kb.N.key\_rollover** unsigned r/w (default 2)

With most matrix keyboards the scancodes are only unambiguous with 1 or 2 keys pressed. With more keys pressed phantom keystrokes can appear. Some keyboards are optimised to reduce this problem, and some have internal diodes so that any number of keys may be pressed simultaneously. Increase the value of this parameter if such a keyboard is connected, or if phantom keystrokes are more acceptable than only two keys being active at one time.

**matrix\_kb.N.negative-logic** bit r/w (default 1) only in scan mode

When no keys are pressed a typical digital input will float high. The input will then be pulled low by the keypad when the corresponding poll line is low. Set this parameter to 0 if the IO in use requires one row at a time to be high, and a high input corresponds to a button press.

**NAME**

mesa\_7i65 – Support for the Mesa 7i65 Octuple Servo Card

**SYNOPSIS**

**loadrt mesa\_7i65**

**DESCRIPTION**

The component takes parameters in the form of a comma-separated list of bspi (buffered SPI) instance names, for example:

**loadrt mesa\_7i65 bspi\_chans=hm2\_5i23.0.bspi.0, hm2\_5i23.0.bspi.1**

The BSPI instances are printed to the dmesg buffer during the Hostmot2 setup sequence, one for each bspi instance included in the bitfile loaded to each installed card during the Hostmot2 setup sequence. Type "dmesg" at the terminal prompt to view the output.

**PINS**

**mesa-7i65.N.analogue.M.out** float in (M=0..7)

Analogue output values. The value will be limited to a -1.0 to +1.0 range

**mesa-7i65.N.analogue.M.in** float out (M=0..7)

Analogue outputs read by the 7i65 (in Volts)

**mesa-7i65.N.digital.M.in** bit out (M=0..3)

Miscellaneous Digital Inputs

**mesa-7i65.N.enable.M.out** bit in (M=0..7)

Amplifier-enable control pins

**mesa-7i65.N.watchdog.has-bit** bit out

Indicates the status of the 7i65 Watchdog (which is separate from the FPGA card watchdog)

**PARAMETERS**

**mesa-7i65.N.scale-M** float rw (M=0..7) (default: 10)

Analogue output scale factor. For example if the scale is 7 then an input of 1.0 will give 7V on the output terminals

**mesa-7i65.N.is-bipolar-M** bit rw (M=0..7) (default: 1)

Set this value to TRUE for a plus/minus "scale" output. Set to 0 for a 0-"scale" output

**AUTHOR**

Andy Pugh / Cliff Blackburn

**LICENSE**

GPL

**NAME**

`mesa_uart` – An example component demonstrating how to access the Hostmot2 UART

**SYNOPSIS**

**loadrt mesa\_uart [count=*N* | names=*name1* [, *name2* ...]]**

**DESCRIPTION**

This component creates 16 input and 16 output pins. It transmits {name}.N.tx-bytes on the selected UART every thread cycle and reads up to 16 bytes each cycle out of the receive FIFO and writes the values to the associated output pins. {name}.rx-bytes indicates how many pins have been written to. (pins > rx-bytes simply hold their previous value)

This module uses the names= mode of loadrt declaration to specify which UART instances to enable. A check is included to ensure that the count= option is not used instead.

The component takes parameters in the form of a comma-separated list of UART instance names, for example:

**loadrt mesa\_uart names=hm2\_5i23.0.uart.0,hm2\_5i23.0.uart.7**

Note that no spaces are allowed in the string unless it is delimited by double quotes.

The UART instance names are printed to the dmesg buffer during the Hostmot2 setup sequence, one for each bspi instance included in the bitfile loaded to each installed card during the Hostmot2 setup sequence. Type "dmesg" at the terminal prompt to view the output.

The component exports two functions, send and receive, which need to be added to a realtime thread.

The above example will output data on UART channels 0 and 7 and the pins will have the names of the individual UARTS. (they need not be on the same card, or even the same bus).

Read the documents on "comp" for help with writing realtime components: <http://www.linux-cnc.org/docview/html/hal/comp.html>

**FUNCTIONS**

**mesa-uart.N.send** (requires a floating-point thread)

**mesa-uart.N.receive** (requires a floating-point thread)

**PINS**

**mesa-uart.N.tx-data-MM** u32 in (MM=00..15)

Data to be transmitted

**mesa-uart.N.rx-data-MM** u32 out (MM=00..15)

Data received

**mesa-uart.N.tx-bytes** s32 in

Number of bytes to transmit

**mesa-uart.N.rx-bytes** s32 out

Number of Bytes received

**AUTHOR**

Andy Pugh [andy@bodgesoc.org](mailto:andy@bodgesoc.org)

**LICENSE**

GPL

**NAME**

message – Display a message

**SYNOPSIS**

**loadrt message [count=*N* | names=*name1* [, *name2* ...]] [messages=*N*]**

**messages**

The messages to display. These should be listed, comma-delimited, inside a single set of quotes. See the "Description" section for an example. If there are more messages than "count" or "names" then the excess will be ignored. If there are fewer messages than "count" or "names" then an error will be raised and the component will not load.

**DESCRIPTION**

Allows HAL pins to trigger a message. Example hal commands:

```
loadrt message names=oillow,oilpressure,inverterfail messages="Slideway oil low,No oil pressure,Spindle inverter fault"
```

```
addf oillow servo-thread
```

```
addf oilpressure servo-thread
```

```
addf inverterfail servo-thread
```

```
setp oillow.edge 0 #this pin should be active low
```

```
net no-oil classicladder.0.out-21 oillow.trigger
```

```
net no-pressure classicladder.0.out-22 oilpressure.trigger
```

```
net no-inverter classicladder.0.out-23 inverterfail.trigger
```

When any pin goes active, the corresponding message will be displayed.

**FUNCTIONS**

**message.*N***

Display a message

**PINS**

**message.*N*.trigger** bit in (default: *FALSE*)

signal that triggers the message

**message.*N*.force** bit in (default: *FALSE*)

A FALSE->TRUE transition forces the message to be displayed again if the trigger is active

**PARAMETERS**

**message.*N*.edge** bit rw (default: *TRUE*)

Selects the desired edge: TRUE means falling, FALSE means rising

**LICENSE**

GPL v2

**NAME**

minmax – Track the minimum and maximum values of the input to the outputs

**SYNOPSIS**

**loadrt minmax** [**count**=*N* | **names**=*name1* [, *name2* ... ]]

**FUNCTIONS**

**minmax.N** (requires a floating-point thread)

**PINS**

**minmax.N.in** float in

**minmax.N.reset** bit in

When reset is asserted, 'in' is copied to the outputs

**minmax.N.max** float out

**minmax.N.min** float out

**LICENSE**

GPL

**NAME**

motion – accepts NML motion commands, interacts with HAL in realtime

**SYNOPSIS**

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

**DESCRIPTION**

By default, the base thread does not support floating point. Software stepping, software encoder counting, and software pwm do not use floating point. **base\_thread\_fp** can be used to enable floating point in the base thread (for example for brushless DC motor control).

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

Optionally the number of Digital I/O is set with num\_dio. The number of Analog I/O is set with num\_aio. The default is 4 each.

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

**PINS**

**axis.N.amp-enable-out** OUT BIT

TRUE if the amplifier for this joint should be enabled

**axis.N.amp-fault-in** IN BIT

Should be driven TRUE if an external fault is detected with the amplifier for this joint

**axis.N.home-sw-in** IN BIT

Should be driven TRUE if the home switch for this joint is closed

**axis.N.homing** OUT BIT

TRUE if the joint is currently homing

**axis.N.index-enable** IO BIT

Should be attached to the index-enable pin of the joint's encoder to enable homing to index pulse

**axis.N.is-unlocked** IN BIT

If the axis is a locked rotary the unlocked sensor should be connected to this pin

**axis.N.jog-counts** IN S32

Connect to the "counts" pin of an external encoder to use a physical jog wheel.

**axis.N.jog-enable** IN BIT

When TRUE (and in manual mode), any change to "jog-counts" will result in motion. When false, "jog-counts" is ignored.

**axis.N.jog-scale** IN FLOAT

Sets the distance moved for each count on "jog-counts", in machine units.

**axis.N.jog-vel-mode** IN BIT

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

**axis.N.joint-pos-cmd** OUT FLOAT

The joint (as opposed to motor) commanded position. There may be several offsets between the joint and motor coordinates: backlash compensation, screw error compensation, and home offsets.

**axis.N.joint-pos-fb** OUT FLOAT

The joint feedback position. This value is computed from the actual motor position minus joint offsets. Useful for machine visualization.

**axis.N.motor-pos-cmd** OUT FLOAT

The commanded position for this joint.

**axis.N.motor-pos-fb** IN FLOAT

The actual position for this joint.

**axis.N.neg-lim-sw-in** IN BIT

Should be driven TRUE if the negative limit switch for this joint is tripped.

**axis.N.pos-lim-sw-in** IN BIT

Should be driven TRUE if the positive limit switch for this joint is tripped.

**axis.N.unlock** OUT BIT

TRUE if the axis is a locked rotary and a move is commanded.

**motion.adaptive-feed** IN FLOAT

When adaptive feed is enabled with M52 P1, the commanded velocity is multiplied by this value. This effect is multiplicative with the NML-level feed override value and motion.feed-hold.

**motion.analog-in-NN** IN FLOAT

These pins are used by M66 Enn wait-for-input mode.

**motion.analog-out-NN** OUT FLOAT

These pins are used by M67-68.

**motion.coord-error** OUT BIT

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

**motion.coord-mode** OUT BIT

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

**motion.current-vel** OUT FLOAT  
Current cartesian velocity

**motion.digital-in-*NN*** IN BIT  
These pins are used by M66 Pnn wait-for-input mode.

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

**motion.distance-to-go** OUT FLOAT  
Distance remaining in the current move

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

**motion.feed-hold** IN BIT  
When Feed Stop Control is enabled with M53 P1, and this bit is TRUE, the feed rate is set to 0.

**motion.feed-inhibit** IN BIT  
When this pin is TRUE, machine motion is inhibited (this includes jogs, programmed feeds, and programmed rapids, aka traverse moves).

If the machine is performing a spindle synchronized move when this pin goes TRUE, the spindle synchronized motion will finish, and any following moves will be inhibited (this is to prevent damage to the machine, the tool, or the work piece).

If the machine is in the middle of a (non-spindle synchronized) move when this pin goes TRUE, the machine will decelerate to a stop at the maximum allowed acceleration rate.

Motion resumes when this pin goes FALSE.

**motion.in-position** OUT BIT  
TRUE if the machine is in position (ie, not currently moving towards the commanded position).

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

**motion.program-line** OUT S32

**motion.requested-vel** OUT FLOAT  
The requested velocity with no adjustments for feed override

**motion.spindle-at-speed** IN BIT  
Motion will pause until this pin is TRUE, under the following conditions: before the first feed move after each spindle start or speed change; before the start of every chain of spindle-synchronized moves; and if in CSS mode, at every rapid->feed transition.

**motion.spindle-brake** OUT BIT

TRUE when the spindle brake should be applied

**motion.spindle-forward** OUT BIT

TRUE when the spindle should rotate forward

**motion.spindle-index-enable** I/O BIT

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

**motion.spindle-inhibit** IN BIT

When TRUE, the spindle speed is set and held to 0.

**motion.spindle-on** OUT BIT

TRUE when spindle should rotate

**motion.spindle-reverse** OUT BIT

TRUE when the spindle should rotate backward

**motion.spindle-revs** IN FLOAT

For correct operation of spindle synchronized moves, this signal must be hooked to the position pin of the spindle encoder.

**motion.spindle-speed-in** IN FLOAT

Actual spindle speed feedback in revolutions per second; used for G96 (constant surface speed) and G95 (feed per revolution) modes.

**motion.spindle-speed-out** OUT FLOAT

Desired spindle speed in rotations per minute

**motion.spindle-speed-out-abs** OUT FLOAT

Desired spindle speed in rotations per minute, always positive regardless of spindle direction.

**motion.spindle-speed-out-rps** OUT float

Desired spindle speed in rotations per second

**motion.spindle-speed-out-rps-abs** OUT float

Desired spindle speed in rotations per second, always positive regardless of spindle direction.

**motion.spindle-orient-angle** OUT FLOAT

Desired spindle orientation for M19. Value of the M19 R word parameter plus the value of the [RS274NGC]ORIENT\_OFFSET ini parameter.

**motion.spindle-orient-mode** OUT BIT

Desired spindle rotation mode. Reflects M19 P parameter word.

**motion.spindle-orient** OUT BIT

Indicates start of spindle orient cycle. Set by M19. Cleared by any of M3,M4,M5. If spindle-orient-fault is not zero during spindle-orient true, the M19 command fails with an error message.

**motion.spindle-is-oriented** IN BIT

Acknowledge pin for spindle-orient. Completes orient cycle. If spindle-orient was true when spindle-is-oriented was asserted, the spindle-orient pin is cleared and the spindle-locked pin is asserted. Also, the spindle-brake pin is asserted.

**motion.spindle-orient-fault** IN S32

Fault code input for orient cycle. Any value other than zero will cause the orient cycle to abort.

**motion.spindle-locked** OUT BIT

Spindle orient complete pin. Cleared by any of M3,M4,M5.

**motion.teleop-mode** OUT bit**motion.tooloffset.x** OUT FLOAT**motion.tooloffset.y** OUT FLOAT**motion.tooloffset.z** OUT FLOAT**motion.tooloffset.a** OUT FLOAT**motion.tooloffset.b** OUT FLOAT**motion.tooloffset.c** OUT FLOAT**motion.tooloffset.u** OUT FLOAT**motion.tooloffset.v** OUT FLOAT**motion.tooloffset.w** OUT FLOAT

Current tool offset in all 9 axes.

**DEBUGGING PINS**

Many of the pins below serve as debugging aids, and are subject to change or removal at any time.

**axis.N.active** OUT BIT

TRUE when this joint is active

**axis.N.backlash-corr** OUT FLOAT

Backlash or screw compensation raw value

**axis.N.backlash-filt** OUT FLOAT

Backlash or screw compensation filtered value (respecting motion limits)

**axis.N.backlash-vel** OUT FLOAT

Backlash or screw compensation velocity

**axis.N.coarse-pos-cmd** OUT FLOAT

**axis.N.error** OUT BIT

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

**axis.N.f-error** OUT FLOAT

The actual following error

**axis.N.f-error-lim** OUT FLOAT

The following error limit

**axis.N.f-errored** OUT BIT

TRUE when this joint has exceeded the following error limit

**axis.N.faulted** OUT BIT

**axis.N.free-pos-cmd** OUT FLOAT

The "free planner" commanded position for this joint.

**axis.N.free-tp-enable** OUT BIT

TRUE when the "free planner" is enabled for this joint

**axis.N.free-vel-lim** OUT FLOAT

The velocity limit for the free planner

**axis.N.homed** OUT BIT

TRUE if the joint has been homed

**axis.N.in-position** OUT BIT

TRUE if the joint is using the "free planner" and has come to a stop

**axis.N.joint-vel-cmd** OUT FLOAT

The joint's commanded velocity

**axis.N.kb-jog-active** OUT BIT

**axis.N.neg-hard-limit** OUT BIT

The negative hard limit for the joint

**axis.N.pos-hard-limit** OUT BIT

The positive hard limit for the joint

**axis.N.wheel-jog-active** OUT BIT

**motion.motion-enabled** OUT BIT

**motion.motion-type** OUT S32

These values are from src/emc/nml\_intf/motion\_types.h

- 1: Traverse
- 2: Linear feed
- 3: Arc feed
- 4: Tool change
- 5: Probing
- 6: Rotary axis indexing

**motion.on-soft-limit** OUT BIT**motion.program-line** OUT S32**motion.teleop-mode** OUT BIT

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

**PARAMETERS**

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

**motion-command-handler.time****motion-command-handler.tmax****motion-controller.time****motion-controller.tmax**

Show information about the execution time of these HAL functions in CPU cycles

**motion.debug-\***

These values are used for debugging purposes.

**motion.servo.last-period**

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

**motion.servo.overruns**

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

**FUNCTIONS**

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

**motion-command-handler**

Processes motion commands coming from user space

**motion-controller**

Runs the LinuxCNC motion controller

**BUGS**

This manual page is horribly incomplete.

**SEE ALSO**

iocontrol(1)

**NAME**

moveoff – Component for Hal-only offsets

**SYNOPSIS**

**loadrt moveoff [count=N]names=name1[,name2...] [personality=P,P,...]**

**DESCRIPTION**

The moveoff component is used to offset joint positions using custom Hal connections. Implementing an offset-while-program-is-paused functionality is supported with appropriate connections for the input pins. Nine joints are supported.

The axis offset pin values (offset-in-M) are continuously applied (respecting limits on value, velocity, and acceleration) to the output pins (offset-current-M, pos-plusoffset-M, fb-minusoffset-M) when both enabling input pins (apply-offsets and move-enable) are TRUE. The two enabling inputs are anded internally. A **warning** pin is set and a message issued if the apply-offsets pin is deasserted while offsets are applied. The warning pin remains TRUE until the offsets are removed or the apply-offsets pin is set.

Typically, the move-enable pin is connected to external controls and the apply-offsets pin is connected to halui.program.is-paused (for offsets only while paused) or set to TRUE (for continuously applied offsets).

Applied offsets are **automatically returned** to zero (respecting limits) when either of the enabling inputs is deactivated. The zero value tolerance is specified by the epsilon input pin value.

Waypoints are recorded when the moveoff component is enabled. Waypoints are managed with the waypoint-sample-secs and waypoint-threshold pins. When the backtrack-enable pin is TRUE, the auto-return path follows the recorded waypoints. When the memory available for waypoints is exhausted, offsets are frozen and the waypoint-limit pin is asserted. This restriction applies regardless of the state of the backtrack-enable pin. An enabling pin must be deasserted to allow a return to the original (non-offset position).

Backtracking through waypoints results in **slower** movement rates as the moves are point-to-point respecting velocity and acceleration settings. The velocity and acceleration limit pins can be managed dynamically to control offsets at all times.

When backtrack-enable is FALSE, the auto-return move is **NOT** coordinated, each axis returns to zero at its own rate. If a controlled path is wanted in this condition, each axis should be manually returned to zero before deasserting an enabling pin.

The waypoint-sample-secs, waypoint-threshold, and epsilon pins are evaluated only when the component is idle.

The offsets-applied output pin is provided to indicate the current state to a GUI so that program resumption can be managed. If the offset(s) are non-zero when the apply-offsets pin is deasserted (for example when resuming a program when offsetting during a pause), offsets are returned to zero (respecting limits) and an **Error** message is issued.

**Caution:** If offsets are enabled and applied and the machine is turned off for any reason, any **external** Hal logic that manages the enabling pins and the offset-in-M inputs is responsible for their state when the machine is subsequently turned on again.

This Hal-only means of offsetting is typically not known to LinuxCNC nor available in GUI preview displays. **No protection is provided** for offset moves that exceed soft limits managed by LinuxCNC. Since soft limits are not honored, an offset move may encounter hard limits (or **CRASH** if there are no limit switches). Use of the offset-min-M and offset-max-M inputs to limit travel is recommended. Triggering a hard limit will turn off the machine -- see **Caution** above.

The offset-in-M values may be set with inifile settings, controlled by a GUI, or managed by other Hal

components and connections. Fixed values may be appropriate in simple cases where the direction and amount of offset is well-defined but a control method is required to deactivate an enabling pin in order to return offsets to zero. GUIs may provide means for users to set, increment, decrement, and accumulate offset values for each axis and may set offset-in-M values to zero before deasserting an enabling pin.

The default values for accel, vel, min, max, epsilon, waypoint-sample-secs, and waypoint-threshold may not be suitable for any particular application. This Hal component is unaware of limits enforced elsewhere by LinuxCNC. Users should test usage in a simulator application and understand all hazards **before** use on hardware.

The module personality item sets the number of joints supported (default==3, maximum is 9).

Use of the names= option for naming is **required** for compatibility with the gui provided as scripts/moveoff\_gui:

```
loadrt moveoff names=mv personality=number_of_joints
```

**man moveoff\_gui** for more information

## EXAMPLES

Example simulator configs that demonstrate the moveoff component and a simple gui (scripts/moveoff\_gui) are located in configs/sim/axis/moveoff. The axis gui is used for the demonstrations and the configs can be adapted for other guis like touchy and gscreen. An example with the touchy gui is provided in configs/sim/touchy/ngcgui/.

## FUNCTIONS

**moveoff.N.read-inputs** (requires a floating-point thread)

Read all inputs

**moveoff.N.write-outputs** (requires a floating-point thread)

Write computed offset outputs (offset-current-M, pos-plusoffset-M, fb-minusoffset-M). All other outputs are updated by read-inputs()

## PINS

**moveoff.N.power-on** bit in

Connect to motion.motion-enabled

**moveoff.N.move-enable** bit in

Enable offsets (Enabling requires apply-offset TRUE also)

**moveoff.N.apply-offsets** bit in

Enable offsets (Enabling requires move-enable TRUE also)

**moveoff.N.backtrack-enable** bit in (default: 1)

Enable backtrack on auto-return

**moveoff.N.epsilon** float in (default: 0.0005)

When enabling pins are deactivated, return to un-offsetted position within epsilon units. Warning: values that are too small in value may cause overshoot, A minimum value of 0.0001 is **silently enforced**

**moveoff.N.waypoint-threshold** float in (default: 0.02)

Minimum distance (in a single axis) for a new waypoint

**moveoff.N.waypoint-sample-secs** float in (default: 0.02)

Minimum sample interval (in seconds) for a new waypoint

**moveoff.N.warning** bit out

Set TRUE if apply-offsets is deasserted while offset-applied is TRUE

**moveoff.N.offset-applied** bit out

TRUE if one or more offsets are applied

**moveoff.N.waypoint-limit** bit out (default: 0)

Indicates waypoint limit reached (motion ceases), an enabling pin must be deasserted to initiate return to original position

**moveoff.N.waypoint-ct** s32 out

Waypoint count (for debugging)

**moveoff.N.waypoint-percent-used** s32 out

Percent of available waypoints used

**moveoff.N.offset-in-M** float in (M=0..personality)

Joint offset input value

**moveoff.N.pos-M** float in (M=0..personality)

Joint position (typ: axis.0.motor-pos-cmd)

**moveoff.N.fb-M** float in (M=0..personality)

Joint feedback (typ from encoder and input to pid controller (pid.feedback))

**moveoff.N.offset-current-M** float out (M=0..personality)

Joint offset current value

**moveoff.N.pos-plusoffset-M** float out (M=0..personality)

Computed joint position plus offset (typically connect to pid command input)

**moveoff.N.fb-minusoffset-M** float out (M=0..personality)

Computed Joint feedback minus offset (typically connected to axis.0.motor-pos-fb)

**moveoff.N.offset-vel-M** float in (M=0..personality) (default: 10)

Joint offset velocity limit

**moveoff.N.offset-accel-M** float in (M=0..personality) (default: 100)

Joint offset acceleration limit

**moveoff.N.offset-min-M** float in (M=0..personality) (default: -1e20)

Minimum limit for applied joint offset (typ negative)

**moveoff.N.offset-max-M** float in (M=0..personality) (default: 1e20)

Maximum limit for applied offset (typ positive)

**moveoff.N.dbg-waypoint-limit-test** bit in

Debug input to test with limited number of waypoints

**moveoff.N.dbg-state** s32 out

Debug output for current state of state machine

## LICENSE

GPL

**NAME**

mult2 – Product of two inputs

**SYNOPSIS**

**loadrt mult2** [**count**=*N*][**names**=*name1*[,*name2*...]]

**FUNCTIONS**

**mult2.N** (requires a floating-point thread)

**PINS**

**mult2.N.in0** float in

**mult2.N.in1** float in

**mult2.N.out** float out  
out = in0 \* in1

**LICENSE**

GPL

**NAME**

multiclick – Single-, double-, triple-, and quadruple-click detector

**SYNOPSIS**

**loadrt multiclick** [**count**=*N*][**names**=*name1*[,*name2*...]]

**DESCRIPTION**

A click is defined as a rising edge on the 'in' pin, followed by the 'in' pin being True for at most 'max-hold-ns' nanoseconds, followed by a falling edge.

A double-click is defined as two clicks, separated by at most 'max-space-ns' nanoseconds with the 'in' pin in the False state.

I bet you can guess the definition of triple- and quadruple-click.

You probably want to run the input signal through a debounce component before feeding it to the multiclick detector, if the input is at all noisy.

The '\*-click' pins go high as soon as the input detects the correct number of clicks.

The '\*-click-only' pins go high a short while after the click, after the click separator space timeout has expired to show that no further click is coming. This is useful for triggering halui MDI commands.

**FUNCTIONS**

**multiclick.N**

Detect single-, double-, triple-, and quadruple-clicks

**PINS**

**multiclick.N.in** bit in

The input line, this is where we look for clicks.

**multiclick.N.single-click** bit out

Goes high briefly when a single-click is detected on the 'in' pin.

**multiclick.N.single-click-only** bit out

Goes high briefly when a single-click is detected on the 'in' pin and no second click followed it.

**multiclick.N.double-click** bit out

Goes high briefly when a double-click is detected on the 'in' pin.

**multiclick.N.double-click-only** bit out

Goes high briefly when a double-click is detected on the 'in' pin and no third click followed it.

**multiclick.N.triple-click** bit out

Goes high briefly when a triple-click is detected on the 'in' pin.

**multiclick.N.triple-click-only** bit out

Goes high briefly when a triple-click is detected on the 'in' pin and no fourth click followed it.

**multiclick.N.quadruple-click** bit out

Goes high briefly when a quadruple-click is detected on the 'in' pin.

**multiclick.N.quadruple-click-only** bit out

Goes high briefly when a quadruple-click is detected on the 'in' pin and no fifth click followed it.

**multiclick.N.state** s32 out

**PARAMETERS**

**multiclick.N.invert-input** bit rw (default: *FALSE*)

If *FALSE* (the default), clicks start with rising edges. If *TRUE*, clicks start with falling edges.

**multiclick.N.max-hold-ns** u32 rw (default: *250000000*)

If the input is held down longer than this, it's not part of a multi-click. (Default 250,000,000 ns, 250 ms.)

**multiclick.N.max-space-ns** u32 rw (default: *250000000*)

If the input is released longer than this, it's not part of a multi-click. (Default 250,000,000 ns, 250 ms.)

**multiclick.N.output-hold-ns** u32 rw (default: *100000000*)

Positive pulses on the output pins last this long. (Default 100,000,000 ns, 100 ms.)

## LICENSE

GPL

**NAME**

multiswitch – This component toggles between a specified number of output bits

**SYNOPSIS**

**loadrt multiswitch personality=*P* [cfg=*N*]**

**cfg**      cfg should be a comma-separated list of sizes for example cfg=2,4,6 would create 3 instances of 2, 4 and 6 bits respectively.

Ignore the "personality" parameter, that is auto-generated

**FUNCTIONS**

**multiswitch.*N*** (requires a floating-point thread)

**PINS**

**multiswitch.*N*.up** bit in (default: *false*)

Receives signal to toggle up

**multiswitch.*N*.down** bit in (default: *false*)

Receives signal to toggle down

**multiswitch.*N*.bit-*MM*** bit out (*MM*=00..*personality*) (default: *false*)

Output bits

**PARAMETERS**

**multiswitch.*N*.top-position** u32 rw

Number of positions

**multiswitch.*N*.position** s32 rw

Current state (may be set in the HAL)

**AUTHOR**

ArcEye schooner30@tiscali.co.uk / Andy Pugh andy@bodgesoc.org

**LICENSE**

GPL

**NAME**

**mux16** – Select from one of sixteen input values

**SYNOPSIS**

**loadrt mux16** [**count**=*N* | **names**=*name1* [, *name2* ... ]]

**FUNCTIONS**

**mux16.N** (requires a floating-point thread)

**PINS**

**mux16.N.use-graycode** bit in

This signifies the input will use Gray code instead of binary. Gray code is a good choice when using physical switches because for each increment only one select input changes at a time.

**mux16.N.suppress-no-input** bit in

This suppresses changing the output if all select lines are false. This stops unwanted jumps in output between transitions of input. but make in00 unavaliable.

**mux16.N.debounce-time** float in

sets debouce time in seconds. eg. .10 = a tenth of a second input must be stable this long before outputs changes. This helps to ignore 'noisy' switches.

**mux16.N.selM** bit in (M=0..3)

Together, these determine which **inN** value is copied to **out**.

**mux16.N.out-f** float out

**mux16.N.out-s** s32 out

Follows the value of one of the **inN** values according to the four **sel** values and whether use-gray-code is active. The s32 value will be trunuated and limited to the max and min values of signed values.

**sel3=FALSE, sel2=FALSE, sel1=FALSE, sel0=FALSE**  
**out** follows **in0**

**sel3=FALSE, sel2=FALSE, sel1=FALSE, sel0=TRUE**  
**out** follows **in1**

etc.

**mux16.N.inMM** float in (MM=00..15)

array of selectable outputs

**PARAMETERS**

**mux16.N.elapsed** float r

Current value of the internal debounce timer for debugging.

**mux16.N.selected** s32 r

Current value of the internal selection variable after conversion for debugging

**LICENSE**

GPL

**NAME**

mux2 – Select from one of two input values

**SYNOPSIS**

**loadrt mux2** [**count**=*N* | **names**=*name1* [, *name2* ...]]

**FUNCTIONS**

**mux2.N** (requires a floating-point thread)

**PINS**

**mux2.N.sel** bit in

**mux2.N.out** float out

Follows the value of in0 if sel is FALSE, or in1 if sel is TRUE

**mux2.N.in1** float in

**mux2.N.in0** float in

**LICENSE**

GPL

**NAME**

**mux4** – Select from one of four input values

**SYNOPSIS**

**loadrt mux4 [count=*N*|names=*name1*[,*name2*...]]**

**FUNCTIONS**

**mux4.*N*** (requires a floating-point thread)

**PINS**

**mux4.*N*.sel0** bit in

**mux4.*N*.sel1** bit in

Together, these determine which **in<sub>*N*</sub>** value is copied to **out**.

**mux4.*N*.out** float out

Follows the value of one of the **in<sub>*N*</sub>** values according to the two **sel** values

**sel1=FALSE, sel0=FALSE**

**out** follows **in0**

**sel1=FALSE, sel0=TRUE**

**out** follows **in1**

**sel1=TRUE, sel0=FALSE**

**out** follows **in2**

**sel1=TRUE, sel0=TRUE**

**out** follows **in3**

**mux4.*N*.in0** float in

**mux4.*N*.in1** float in

**mux4.*N*.in2** float in

**mux4.*N*.in3** float in

**LICENSE**

GPL

**NAME**

**mux8** – Select from one of eight input values

**SYNOPSIS**

**loadrt mux8 [count=*N*|names=*name1*[,*name2*...]]**

**FUNCTIONS**

**mux8.*N*** (requires a floating-point thread)

**PINS**

**mux8.*N*.sel0** bit in

**mux8.*N*.sel1** bit in

**mux8.*N*.sel2** bit in

Together, these determine which **in<sub>*N*</sub>** value is copied to **out**.

**mux8.*N*.out** float out

Follows the value of one of the **in<sub>*N*</sub>** values according to the three **sel** values

**sel2=FALSE, sel1=FALSE, sel0=FALSE**

**out** follows **in0**

**sel2=FALSE, sel1=FALSE, sel0=TRUE**

**out** follows **in1**

**sel2=FALSE, sel1=TRUE, sel0=FALSE**

**out** follows **in2**

**sel2=FALSE, sel1=TRUE, sel0=TRUE**

**out** follows **in3**

**sel2=TRUE, sel1=FALSE, sel0=FALSE**

**out** follows **in4**

**sel2=TRUE, sel1=FALSE, sel0=TRUE**

**out** follows **in5**

**sel2=TRUE, sel1=TRUE, sel0=FALSE**

**out** follows **in6**

**sel2=TRUE, sel1=TRUE, sel0=TRUE**

**out** follows **in7**

**mux8.*N*.in0** float in

**mux8.*N*.in1** float in

**mux8.*N*.in2** float in

**mux8.*N*.in3** float in

**mux8.*N*.in4** float in

**mux8.*N*.in5** float in

**mux8.*N*.in6** float in

**mux8.*N*.in7** float in

**LICENSE**

GPL

**NAME**

`mux_generic` – choose one from several input values

**SYNOPSIS**

`loadrt mux_generic config="bb8,fu12...."`

**FUNCTIONS**

**mux-gen.NN** Depending on the data types can run in either a floating point or non-floating point thread.

**PINS**

**mux-gen.NN.suppress-no-input** bit in

This suppresses changing the output if all select lines are false. This stops unwanted jumps in output between transitions of input. but makes in00 unavailable.

**mux-gen.NN.debounce-us** unsigned in

sets debounce time in microseconds. eg. 100000 = a tenth of a second. The selection inputs must be stable this long before the output changes. This helps to ignore 'noisy' switches.

**mux-gen.NN.sel-bitMMM** bit in (M=0..N)

**mux-gen.NN.sel-intMMM** unsigned in

Together, these determine which **inN** value is copied to **output**. The bit pins are interpreted as binary bits, and the result is simply added on to the integer pin input. It is expected that either one or the other would normally be used. However, the possibility exists to use a higher-order bit to "shift" the values set by the integer pin. The sel-bit pins are only created when the size of the `mux_gen` component is an integer power of two. This component (unlike `mux16`) does not offer the option of decoding gray-code, however the same effect can be achieved by arranging the order of the input values to suit.

**mux-gen.NN.out-[bit/float/s32/u32]** variable-type out

Follows the value of one of the **inN** values according to the selection bits and/or the selection number. Values will be converted/truncated according to standard C rules. This means, for example that a float input greater than 2147483647 will give an S32 output of -2147483648.

**mux-gen.NN.in-[bit/float/s32/u32]-MM** variable-type in

The possible output values that are selected by the selection pins.

**PARAMETERS**

**mux-gen.N.elapsed** float r

Current value of the internal debounce timer for debugging.

**mux-gen.N.selected** s32 r

Current value of the internal selection variable after conversion for debugging. Possibly useful for setting up gray-code switches.

**DESCRIPTION**

This component is a more general version of the other multiplexing components. It allows the creation of arbitrary-size multiplexers (up to 1024 entries) and also supports differing data types on the input and output pins. The configuration string is a comma-separated list of code-letters and numbers, such as "bb4,fu12" This would create a 4-element bit-to-bit mux and a 12-element float-to-unsigned mux. The code letters are b = bit, f = float, s = signed integer, u = unsigned integer. The first letter code is the input type, the second is the output type. The codes are not case-sensitive. The order of the letters is significant but the position in the string is not. Do not insert any spaces in the config string. Any non-zero float value will be

converted to a "true" output in bit form. Be wary that float datatypes can be very, very, close to zero and not actually be equal to zero.

Each mux has its own HAL function and must be added to a thread separately. If neither input nor output is of type float then the function is base-thread (non floating-point) safe. Any mux\_generic with a floating point input or output can only be added to a floating-point thread.

**LICENSE**

GPL

**AUTHOR**

Andy Pugh

**NAME**

near – Determine whether two values are roughly equal.

**SYNOPSIS**

**loadrt near [count=*N*][names=*name1*[,*name2*...]]**

**FUNCTIONS**

**near.*N*** (requires a floating-point thread)

**PINS**

**near.*N*.in1** float in

**near.*N*.in2** float in

**near.*N*.out** bit out

**out** is true if **in1** and **in2** are within a factor of **scale** (i.e., for in1 positive,  $\text{in1}/\text{scale} \leq \text{in2} \leq \text{in1} * \text{scale}$ ), OR if their absolute difference is no greater than **difference** (i.e.,  $|\text{in1} - \text{in2}| \leq \text{difference}$ ). **out** is false otherwise.

**PARAMETERS**

**near.*N*.scale** float rw (default: *1*)

**near.*N*.difference** float rw (default: *0*)

**LICENSE**

GPL

**NAME**

not – Inverter

**SYNOPSIS**

**loadrt not** [**count**=*N* | **names**=*name1* [, *name2* ...]]

**FUNCTIONS**

**not.N**

**PINS**

**not.N.in** bit in

**not.N.out** bit out

**LICENSE**

GPL

**NAME**

offset – Adds an offset to an input, and subtracts it from the feedback value

**SYNOPSIS**

**loadrt offset** [**count**=*N* | **names**=*name1* [, *name2*...]]

**FUNCTIONS**

**offset.N.update-output** (requires a floating-point thread)

Updated the output value by adding the offset to the input

**offset.N.update-feedback** (requires a floating-point thread)

Update the feedback value by subtracting the offset from the feedback

**PINS**

**offset.N.offset** float in

The offset value

**offset.N.in** float in

The input value

**offset.N.out** float out

The output value

**offset.N.fb-in** float in

The feedback input value

**offset.N.fb-out** float out

The feedback output value

**LICENSE**

GPL

**NAME**

oneshot – one-shot pulse generator

**SYNOPSIS**

**loadrt oneshot** [**count**=*N* | **names**=*name1* [, *name2* ... ]]

**DESCRIPTION**

creates a variable-length output pulse when the input changes state. This function needs to run in a thread which supports floating point (typically the servo thread). This means that the pulse length has to be a multiple of that thread period, typically 1mS. For a similar function that can run in the base thread, and which offers higher resolution, see "edge".

**FUNCTIONS**

**oneshot.N** (requires a floating-point thread)  
Produce output pulses from input edges

**PINS**

**oneshot.N.in** bit in  
Trigger input

**oneshot.N.out** bit out  
Active high pulse

**oneshot.N.out-not** bit out  
Active low pulse

**oneshot.N.width** float in (default: 0)  
Pulse width in seconds

**oneshot.N.time-left** float out  
Time left in current output pulse

**PARAMETERS**

**oneshot.N.retriggerable** bit rw (default: *TRUE*)  
Allow additional edges to extend pulse

**oneshot.N.rising** bit rw (default: *TRUE*)  
Trigger on rising edge

**oneshot.N.falling** bit rw (default: *FALSE*)  
Trigger on falling edge

**LICENSE**

GPL

**NAME**

opto\_ac5 – Realtime driver for opto22 PCI-AC5 cards

**SYNOPSIS**

**loadrt opto\_ac5 [portconfig0=0xN] [portconfig1=0xN]**

**DESCRIPTION**

These pins and parameters are created by the realtime **opto\_ac5** module. The portconfig0 and portconfig1 variables are used to configure the two ports of each card. The first 24 bits of a 32 bit number represent the 24 i/o points of each port. The lowest (rightmost) bit would be HAL pin 0 which is header connector pin 47. Then next bit to the left would be HAL pin 1, header connector pin 45 and so on, until bit 24 would be HAL pin 23, header connector pin 1. "1" bits represent output points. So channel 0..11 as inputs and 12..23 as outputs would be represented by (in binary) 111111111110000000000000 which is 0xfff000 in hexadecimal. That is the number you would use Eg. loadrt opto\_ac5 portconfig0=0xfff000

If no portconfig variable is specified the default configuration is 12 inputs then 12 outputs.

Up to 4 boards are supported. Boards are numbered starting at 0.

Portnumber can be 0 or 1. Port 0 is closes to the card bracket.

**PINS**

**opto\_ac5.[BOARDNUMBER].port[PORTNUMBER].in-[PINNUMBER]** OUT bit

**opto\_ac5.[BOARDNUMBER].port[PORTNUMBER].in-[PINNUMBER]**-not OUT bit

Connect a hal bit signal to this pin to read an i/o point from the card. The PINNUMBER represents the position in the relay rack. Eg. PINNUMBER 0 is position 0 in a opto22 relay rack and would be pin 47 on the 50 pin header connector. The **-not** pin is inverted so that LOW gives TRUE and HIGH gives FALSE.

**opto\_ac5.[BOARDNUMBER].port[PORTNUMBER].out-[PINNUMBER]** IN bit

Connect a hal bit signal to this pin to write to an i/o point of the card. The PINNUMBER represents the position in the relay rack. Eg. PINNUMBER 23 is position 23 in a opto22 relay rack and would be pin 1 on the 50 pin header connector.

**opto\_ac5.[BOARDNUMBER].led[NUMBER]** OUT bit

Turns one of the on board LEDS on/off. LEDS are numbered 0 to 3.

**PARAMETERS**

**opto\_ac5.[BOARDNUMBER].port[PORTNUMBER].out-[PINNUMBER]**-invert W bit

When TRUE, invert the meaning of the corresponding **-out** pin so that TRUE gives LOW and FALSE gives HIGH.

**FUNCTIONS**

**opto\_ac5.0.digital-read**

Add this to a thread to read all the input points.

**opto\_ac5.0.digital-write**

Add this to a thread to write all the output points and LEDS.

**BUGS**

All boards are loaded with the same port configurations as the first board.

**SEE ALSO**

<http://wiki.linuxcnc.org/cgi-bin/wiki.pl?OptoPciAc5>

**NAME**

or2 – Two-input OR gate

**SYNOPSIS**

**loadrt or2** [**count**=*N* | **names**=*name1* [, *name2* ... ]]

**FUNCTIONS**

**or2.N**

**PINS**

**or2.N.in0** bit in

**or2.N.in1** bit in

**or2.N.out** bit out

**out** is computed from the value of **in0** and **in1** according to the following rule:

**in0=FALSE in1=FALSE**

**out=FALSE**

Otherwise,

**out=TRUE**

**LICENSE**

GPL

**NAME**

orient – Provide a PID command input for orientation mode based on current spindle position, target angle and orient mode

**SYNOPSIS**

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

**DESCRIPTION**

This component is designed to support a spindle orientation PID loop by providing a command value, and fit with the motion spindle-orient support pins to support the M19 code.

The spindle is assumed to have stopped in an arbitrary position. The spindle encoder position is linked to the **position** pin. The current value of the position pin is sampled on a positive edge on the **enable** pin, and **command** is computed and set as follows: floor(number of full spindle revolutions in the **position** sampled on positive edge) plus **angle**/360 (the fractional revolution) +1/-1/0 depending on **mode**.

The **mode** pin is interpreted as follows:

0: the spindle rotates in the direction with the lesser angle, which may be clockwise or counterclockwise.

1: the spindle rotates always rotates clockwise to the new angle.

2: the spindle rotates always rotates counterclockwise to the new angle.

**HAL USAGE**

On **motion.spindle-orient** disconnect the spindle control and connect to the orient-pid loop:

```
loadrt orient names=orient
loadrt pid names=orient-pid
net orient-angle motion.spindle-orient-angle orient.angle
net orient-mode motion.spindle-orient-mode orient.mode
net orient-enable motion.spindle-orient orient.enable orient-pid.enable
net spindle-pos ...encoder.position orient.position orient-pid.feedback
net orient-command orient.command orient-pid.command
```

**FUNCTIONS**

**orient.N** (requires a floating-point thread)

Update **command** based on **enable**, **position**, **mode** and **angle**.

**PINS**

**orient.N.enable** bit in

enable angular output for orientation mode

**orient.N.mode** s32 in

0: rotate - shortest move; 1: always rotate clockwise; 2: always rotate counterclockwise

**orient.N.position** float in

spindle position input, unit 1 rev

**orient.N.angle** float in

orient target position in degrees, 0 <= angle < 360

**orient.N.command** float out

target spindle position, input to PID command

**orient.N,poseerr** float out  
in degrees - aid for PID tuning

**AUTHOR**

Michael Haberler

**LICENSE**

GPL

**NAME**

pcl720 – Driver for the Advantech PCL 720 card.

**SYNOPSIS**

**loadrt pcl720 [ioaddr=*N*]**

**ioaddr** Base address of card. Separate each card base address with a comma but no space to load more than one card. eg loadrt pcl720 ioaddr=0x200,0x200. use 0xNNN to define addresses in Hex

**DESCRIPTION**

This driver supports the Advantech PCL720 ISA card. It might work with the PCI version too, but this is untested.

It creates hal pins corresponding to the digital inputs and outputs, but does not support the the counters/timers.

**FUNCTIONS****pcl720.N.read**

Reads each of the digital inputs and updates the HAL pins

**pcl720.N.write**

Writes the values of the output HAL pins to the digital IO

**pcl720.N.reset**

Waits for the length of time specified by the **reset-time** parameter and resets any pins for which the **reset** parameter has been set. This can be used to allow step generators to make a step every thread rather than every other thread. This function must be added to the thread after the "write" function.

Do not use this function if you do not wish to reset any pins.

the stepgen **step-space** parameter should be set to 0 to make use of this function.

**PINS**

**pcl720.N.pin-MM-out** bit in (MM=00..31)

Output pins

**pcl720.N.pin-MM-in** bit out (MM=00..31)

Input pins

**pcl720.N.pin-MM-in-not** bit out (MM=00..31)

Inverted version of each input pin

**pcl720.N.wait-clocks** u32 out

**PARAMETERS**

**pcl720.N.reset-time** u32 rw (default: 5000)

The time in nanoseconds after the write function has run to reset the pins for which the "reset" parameter is set.

**pcl720.N.pin-MM-reset** bit rw (MM=00..31)

specifies if the pin should be reset by the "reset" function

**pcl720.N.pin-MM-out-invert** bit rw (MM=00..31)

Set to true to invert the sense of the output pin

**AUTHOR**

Andy Pugh

**LICENSE**

GPL

**NAME**

**pid** – proportional/integral/derivative controller

**SYNOPSIS**

```
loadrt pid [num_chan=num | names=name1[,name2...]] [debug=dbg]
```

**DESCRIPTION**

**pid** is a classic Proportional/Integral/Derivative controller, used to control position or speed feedback loops for servo motors and other closed-loop applications.

**pid** supports a maximum of sixteen controllers. The number that are actually loaded is set by the **num\_chan** argument when the module is loaded. Alternatively, specify names= and unique names separated by commas.

The **num\_chan=** and **names=** specifiers are mutually exclusive. If neither **num\_chan=** nor **names=** are specified, the default value is three. If **debug** is set to 1 (the default is 0), some additional HAL parameters will be exported, which might be useful for tuning, but are otherwise unnecessary.

**NAMING**

The names for pins, parameters, and functions are prefixed as:

**pid.N.** for N=0,1,...,num-1 when using **num\_chan=num**

**nameN.** for nameN=name1,name2,... when using **names=name1,name2,...**

The **pid.N.** format is shown in the following descriptions.

**FUNCTIONS**

**pid.N.do-pid-calcs** (uses floating-point) Does the PID calculations for control loop *N*.

**PINS**

**pid.N.command** float in

The desired (commanded) value for the control loop.

**pid.N.Pgain** float in

Proportional gain. Results in a contribution to the output that is the error multiplied by **Pgain**.

**pid.N.Igain** float in

Integral gain. Results in a contribution to the output that is the integral of the error multiplied by **Igain**. For example an error of 0.02 that lasted 10 seconds would result in an integrated error (**errorI**) of 0.2, and if **Igain** is 20, the integral term would add 4.0 to the output.

**pid.N.Dgain** float in

Derivative gain. Results in a contribution to the output that is the rate of change (derivative) of the error multiplied by **Dgain**. For example an error that changed from 0.02 to 0.03 over 0.2 seconds would result in an error derivative (**errorD**) of 0.05, and if **Dgain** is 5, the derivative term would add 0.25 to the output.

**pid.N.feedback** float in

The actual (feedback) value, from some sensor such as an encoder.

**pid.N.output** float out

The output of the PID loop, which goes to some actuator such as a motor.

**pid.N.command-deriv** float in

The derivative of the desired (commanded) value for the control loop. If no signal is connected then the derivative will be estimated numerically.

**pid.N.feedback-deriv** float in

The derivative of the actual (feedback) value for the control loop. If no signal is connected then the derivative will be estimated numerically. When the feedback is from a quantized position

source (e.g., encoder feedback position), behavior of the D term can be improved by using a better velocity estimate here, such as the velocity output of encoder(9) or hostmot2(9).

**pid.N.error-previous-target** bit in

Use previous invocation's target vs. current position for error calculation, like the motion controller expects. This may make torque-mode position loops and loops requiring a large I gain easier to tune, by eliminating velocity-dependent following error.

**pid.N.error** float out

The difference between command and feedback.

**pid.N.enable** bit in

When true, enables the PID calculations. When false, **output** is zero, and all internal integrators, etc, are reset.

**pid.N.index-enable** bit in

On the falling edge of **index-enable**, pid does not update the internal command derivative estimate. On systems which use the encoder index pulse, this pin should be connected to the index-enable signal. When this is not done, and FF1 is nonzero, a step change in the input command causes a single-cycle spike in the PID output. On systems which use exactly one of the **-deriv** inputs, this affects the D term as well.

**pid.N.bias** float in

**bias** is a constant amount that is added to the output. In most cases it should be left at zero. However, it can sometimes be useful to compensate for offsets in servo amplifiers, or to balance the weight of an object that moves vertically. **bias** is turned off when the PID loop is disabled, just like all other components of the output. If a non-zero output is needed even when the PID loop is disabled, it should be added with an external HAL sum2 block.

**pid.N.FF0** float in

Zero order feed-forward term. Produces a contribution to the output that is **FF0** multiplied by the commanded value. For position loops, it should usually be left at zero. For velocity loops, **FF0** can compensate for friction or motor counter-EMF and may permit better tuning if used properly.

**pid.N.FF1** float in

First order feed-forward term. Produces a contribution to the output that **FF1** multiplied by the derivative of the commanded value. For position loops, the contribution is proportional to speed, and can be used to compensate for friction or motor CEMF. For velocity loops, it is proportional to acceleration and can compensate for inertia. In both cases, it can result in better tuning if used properly.

**pid.N.FF2** float in

Second order feed-forward term. Produces a contribution to the output that is **FF2** multiplied by the second derivative of the commanded value. For position loops, the contribution is proportional to acceleration, and can be used to compensate for inertia. For velocity loops, it should usually be left at zero.

**pid.N.deadband** float in

Defines a range of "acceptable" error. If the absolute value of **error** is less than **deadband**, it will be treated as if the error is zero. When using feedback devices such as encoders that are inherently quantized, the deadband should be set slightly more than one-half count, to prevent the control loop from hunting back and forth if the command is between two adjacent encoder values. When the absolute value of the error is greater than the deadband, the deadband value is subtracted from the error before performing the loop calculations, to prevent a step in the transfer function at the edge of the deadband. (See **BUGS**.)

**pid.N.maxoutput** float in

Output limit. The absolute value of the output will not be permitted to exceed **maxoutput**, unless **maxoutput** is zero. When the output is limited, the error integrator will hold instead of integrating, to prevent windup and overshoot.

**pid.N.maxerror** float in

Limit on the internal error variable used for P, I, and D. Can be used to prevent high **Pgain** values from generating large outputs under conditions when the error is large (for example, when the command makes a step change). Not normally needed, but can be useful when tuning non-linear systems.

**pid.N.maxerrorD** float in

Limit on the error derivative. The rate of change of error used by the **Dgain** term will be limited to this value, unless the value is zero. Can be used to limit the effect of **Dgain** and prevent large output spikes due to steps on the command and/or feedback. Not normally needed.

**pid.N.maxerrorI** float in

Limit on error integrator. The error integrator used by the **Igain** term will be limited to this value, unless it is zero. Can be used to prevent integrator windup and the resulting overshoot during/after sustained errors. Not normally needed.

**pid.N.maxcmdD** float in

Limit on command derivative. The command derivative used by **FF1** will be limited to this value, unless the value is zero. Can be used to prevent **FF1** from producing large output spikes if there is a step change on the command. Not normally needed.

**pid.N.maxcmdDD** float in

Limit on command second derivative. The command second derivative used by **FF2** will be limited to this value, unless the value is zero. Can be used to prevent **FF2** from producing large output spikes if there is a step change on the command. Not normally needed.

**pid.N.saturated** bit out

When true, the current PID output is saturated. That is,  
**output** =  $\pm$  **maxoutput**.

**pid.N.saturated-s** float out**pid.N.saturated-count** s32 out

When true, the output of PID was continually saturated for this many seconds (**saturated-s**) or periods (**saturated-count**).

**PARAMETERS****pid.N.errorI** float ro (only if debug=1)

Integral of error. This is the value that is multiplied by **Igain** to produce the Integral term of the output.

**pid.N.errorD** float ro (only if debug=1)

Derivative of error. This is the value that is multiplied by **Dgain** to produce the Derivative term of the output.

**pid.N.commandD** float ro (only if debug=1)

Derivative of command. This is the value that is multiplied by **FF1** to produce the first order feed-forward term of the output.

**pid.N.commandDD** float ro (only if debug=1)

Second derivative of command. This is the value that is multiplied by **FF2** to produce the second order feed-forward term of the output.

**BUGS**

Some people would argue that deadband should be implemented such that error is treated as zero if it is within the deadband, and be unmodified if it is outside the deadband. This was not done because it would cause a step in the transfer function equal to the size of the deadband. People who prefer that behavior are welcome to add a parameter that will change the behavior, or to write their own version of **pid**. However, the default behavior should not be changed.

Negative gains may lead to unwanted behavior. It is possible in some situations that negative FF gains

make sense, but in general all gains should be positive. If some output is in the wrong direction, negating gains to fix it is a mistake; set the scaling correctly elsewhere instead.

**NAME**

`pluto_servo` – Hardware driver and firmware for the Pluto-P parallel-port FPGA, for use with servo machines.

**SYNOPSIS**

**loadrt** `pluto_servo` [`ioaddr=N`] [`ioaddr_hi=N`] [`epp_wide=N`] [`watchdog=N`] [`test_encoder=N`]

**ioaddr** [default: 0x378]

The base address of the parallel port.

**ioaddr\_hi** [default: 0]

The secondary address of the parallel port, used to set EPP mode. 0 means to use `ioaddr + 0x400`. -1 means there is no secondary address. The secondary address is used to set the port to EPP mode.

**epp\_wide** [default: 1]

Set to zero to disable the "wide EPP mode". "Wide" mode allows a 16- and 32-bit EPP transfers, which can reduce the time spent in the read and write functions. However, this may not work on all EPP parallel ports.

**watchdog** [default: 1]

Set to zero to disable the "hardware watchdog". "Watchdog" will tristate all outputs approximately 6ms after the last execution of **pluto-servo.write**, which adds some protection in the case of LinuxCNC crashes.

**test\_encoder** [default: 0]

Internally connect `dout0..2` to QA0, QB0, QZ0 to test quadrature counting

**DESCRIPTION**

`Pluto_servo` is a LinuxCNC software driver and associated firmware that allow the Pluto-P board to be used to control a servo-based CNC machine.

The driver has 4 PWM channels, 4 quadrature channels with index pulse, 18 digital outputs (8 shared with PWM), and 20 digital inputs (12 shared with quadrature).

**Encoders**

The encoder pins and parameters conform to the 'canonical encoder' interface described in the HAL manual. It operates in 'x4 mode'.

The sample rate of the encoder is 40MHz. The maximum number quadrature rate is 8191 counts per LinuxCNC servo cycle. For correct handling of the index pulse, the number of encoder counts per revolution must be less than 8191.

**PWM**

The PWM pins and parameters conform to the 'canonical analog output' interface described in the HAL manual. The output pins are 'up/down' or 'pwm/dir' pins as described in the documentation of the 'pwm-gen' component.

Internally the PWM generator is based on a 12-bit, 40MHz counter, giving 4095 duty cycles from -100% to +100% and a frequency of approximately 19.5kHz. In PDM mode, the duty periods are approximately 100ns long.

**Digital I/O**

The digital output pins conform to the 'canonical digital output' interface described in the HAL manual.

The digital input pins conform to the 'canonical digital input' interface described in the HAL manual.

**FUNCTIONS**

- pluto-servo.read** (requires a floating-point thread)  
Read all the inputs from the pluto-servo board
- pluto-servo.write** (requires a floating-point thread)  
Write all the outputs on the pluto-servo board

**PINS**

- pluto-servo.encoder.M.count** s32 out (M=0..3)  
**pluto-servo.encoder.M.position** float out (M=0..3)  
**pluto-servo.encoder.M.velocity** float out (M=0..3)  
**pluto-servo.encoder.M.reset** bit in (M=0..3)  
**pluto-servo.encoder.M.index-enable** bit io (M=0..3)  
 encoder.M corresponds to the pins labeled QAM, QBM, and QZM on the pinout diagram
- pluto-servo.pwm.M.value** float in (M=0..3)  
**pluto-servo.pwm.M.enable** bit in (M=0..3)  
 pwm.M corresponds to the pins labeled UPM and DNM on the pinout diagram
- pluto-servo.dout.MM** bit in (MM=00..19)  
 dout.0M corresponds to the pin labeled OUTM on the pinout diagram. Other pins are shared with the PWM function, as follows:

<b>Pin</b>	<b>Shared</b>
<b>Label</b>	<b>with</b>
dout.10	UP0
dout.10	UP0
dout.12	UP1
dout.14	UP2
dout.18	UP3
dout.11	DOWN0
dout.13	DOWN1
dout.15	DOWN2
dout.19	DOWN3

- pluto-servo.din.MM** bit out (MM=00..19)  
**pluto-servo.din.MM-not** bit out (MM=00..19)  
 For M=0 through 7, din.0M corresponds to the pin labeled INM on the pinout diagram. Other pins are shared with the encoder function, as follows:

<b>Pin</b>	<b>Shared</b>
<b>Label</b>	<b>with</b>
din.8	QZ0
din.9	QZ1
din.10	QZ2
din.11	QZ3
din.12	QB0
din.13	QB1
din.14	QB2
din.15	QB3
din.16	QA0

din.17	QA1
din.18	QA2
din.19	QA3

## PARAMETERS

**pluto-servo.encoder.M.scale** float rw (M=0..3) (default: 1)

**pluto-servo.encoder.z-polarity** bit rw

Set to TRUE if the index pulse is active low, FALSE if it is active high. Affects all encoders.

**pluto-servo.pwm.M.offset** float rw (M=0..3)

**pluto-servo.pwm.M.scale** float rw (M=0..3) (default: 1)

**pluto-servo.pwm.M.max-dc** float rw (M=0..3) (default: 1)

**pluto-servo.pwm.M.min-dc** float rw (M=0..3) (default: 0)

**pluto-servo.pwm.M.pwmdir** bit rw (M=0..3) (default: 0)

Set to TRUE use PWM+direction mode. Set to FALSE to use Up/Down mode.

**pluto-servo.pwm.is-pdm** bit rw

Set to TRUE to use PDM (also called interleaved PWM) mode. Set to FALSE to use traditional PWM mode. Affects all PWM outputs.

**pluto-servo.dout.MM-invert** bit rw (MM=00..19)

If TRUE, the output on the corresponding **dout.MM** is inverted.

**pluto-servo.communication-error** u32 rw

Incremented each time `pluto-servo.read` detects an error code in the EPP status register. While this register is nonzero, new values are not being written to the Pluto-P board, and the status of digital outputs and the PWM duty cycle of the PWM outputs will remain unchanged. If the watchdog is enabled, it will activate soon after the communication error is detected. To continue after a communication error, set this parameter back to zero.

**pluto-servo.debug-0** s32 rw

**pluto-servo.debug-1** s32 rw

These parameters can display values which are useful to developers or for debugging the driver and firmware. They are not useful for integrators or users.

## SEE ALSO

The *pluto\_servo* section in the HAL User Manual, which shows the location of each physical pin on the pluto board.

## LICENSE

GPL

**NAME**

pluto\_step – Hardware driver and firmware for the Pluto-P parallel-port FPGA, for use with stepper machines.

**SYNOPSIS**

```
loadrt pluto_step ioaddr=addr ioaddr_hi=addr epp_wide=[0/1]
```

**ioaddr** [default: 0x378]

The base address of the parallel port.

**ioaddr\_hi** [default: 0]

The secondary address of the parallel port, used to set EPP mode. 0 means to use ioaddr + 0x400. -1 means there is no secondary address.

**epp\_wide** [default: 1]

Set to zero to disable "wide EPP mode". "Wide" mode allows 16- and 32-bit EPP transfers, which can reduce the time spent in the read and write functions. However, this mode may not work on all EPP parallel ports.

**watchdog** [default: 1]

Set to zero to disable the "hardware watchdog". "Watchdog" will tristate all outputs approximately 6ms after the last execution of **pluto\_step.write**, which adds some protection in the case of LinuxCNC crashes.

**speedrange** [default: 0]

Selects one of four speed ranges:

0: Top speed 312.5kHz; minimum speed 610Hz

1: Top speed 156.25kHz; minimum speed 305Hz

2: Top speed 78.125kHz; minimum speed 153Hz

3: Top speed 39.06kHz; minimum speed 76Hz

Choosing the smallest maximum speed that is above the maximum for any one axis may give improved step regularity at low step speeds.

**DESCRIPTION**

Pluto\_step is a LinuxCNC software driver and associated firmware that allow the Pluto-P board to be used to control a stepper-based CNC machine.

The driver has 4 step+direction channels, 14 dedicated digital outputs, and 16 dedicated digital inputs.

**Step generators**

The step generator takes a position input and output.

The step waveform includes step length/space and direction hold/setup time. Step length and direction setup/hold time is enforced in the FPGA. Step space is enforced by a velocity cap in the driver.

*(all the following numbers are subject to change)* In *speedrange=0*, the maximum step rate is 312.5kHz. For position feedback to be accurate, the maximum step rate is 512 pulses per servo cycle (so a 1kHz servo cycle does not impose any additional limitation). The maximum step rate may be lowered by the step length and space parameters, which are rounded up to the nearest multiple of 1600ns.

In successive speedranges the maximum step rate is divided in half, as is the maximum steps per servo cycle, and the minimum nonzero step rate.

**Digital I/O**

The digital output pins conform to the ‘canonical digital output’ interface described in the HAL manual.

The digital input pins conform to the ‘canonical digital input’ interface described in the HAL manual.

**FUNCTIONS**

**pluto-step.read** (requires a floating-point thread)

Read all the inputs from the pluto-step board

**pluto-step.write** (requires a floating-point thread)

Write all the outputs on the pluto-step board

**PINS**

**pluto-step.stepgen.M.position-cmd** float in (M=0..3)

**pluto-step.stepgen.M.velocity-fb** float out (M=0..3)

**pluto-step.stepgen.M.position-fb** float out (M=0..3)

**pluto-step.stepgen.M.counts** s32 out (M=0..3)

**pluto-step.stepgen.M.enable** bit in (M=0..3)

**pluto-step.stepgen.M.reset** bit in (M=0..3)

When TRUE, reset position-fb to 0

**pluto-step.dout.MM** bit in (MM=00..13)

dout.MM corresponds to the pin labeled OUTM on the pinout diagram.

**pluto-step.din.MM** bit out (MM=00..15)

**pluto-step.din.MM-not** bit out (MM=00..15)

din.MM corresponds to the pin labeled INM on the pinout diagram.

**PARAMETERS**

**pluto-step.stepgen.M.scale** float rw (M=0..3) (default: 1.0)

**pluto-step.stepgen.M.maxvel** float rw (M=0..3) (default: 0)

**pluto-step.stepgen.step-polarity** bit rw

**pluto-step.stepgen.steplen** u32 rw

Step length in ns.

**pluto-step.stepgen.stepspace** u32 rw

Step space in ns

**pluto-step.stepgen.dirtime** u32 rw

Dir hold/setup in ns. Refer to the pdf documentation for a diagram of what these timings mean.

**pluto-step.dout.MM-invert** bit rw (MM=00..13)

If TRUE, the output on the corresponding **dout.MM** is inverted.

**pluto-step.communication-error** u32 rw

Incremented each time **pluto-step.read** detects an error code in the EPP status register. While this register is nonzero, new values are not being written to the Pluto-P board, and the status of digital outputs and the PWM duty cycle of the PWM outputs will remain unchanged. If the hardware watchdog is enabled, it will activate shortly after the communication error is detected by Linux-CNC. To continue after a communication error, set this parameter back to zero.

**pluto-step.debug-0** s32 rw

**pluto-step.debug-1** s32 rw

**pluto-step.debug-2** float rw (default: .5)

**pluto-step.debug-3** float rw (default: 2.0)

Registers that hold debugging information of interest to developers

**SEE ALSO**

The *pluto\_step* section in the HAL User Manual, which shows the location of each physical pin on the pluto board.

**LICENSE**  
GPL

**NAME**

**pwmgen** – software PWM/PDM generation

**SYNOPSIS**

```
loadrt pwmgen output_type=type0[,type1...]
```

**DESCRIPTION**

**pwmgen** is used to generate PWM (pulse width modulation) or PDM (pulse density modulation) signals. The maximum PWM frequency and the resolution is quite limited compared to hardware-based approaches, but in many cases software PWM can be very useful. If better performance is needed, a hardware PWM generator is a better choice.

**pwmgen** supports a maximum of eight channels. The number of channels actually loaded depends on the number of *type* values given. The value of each *type* determines the outputs for that channel.

type 0: single output

A single output pin, **pwm**, whose duty cycle is determined by the input value for positive inputs, and which is off (or at **min-dc**) for negative inputs. Suitable for single ended circuits.

type 1: pwm/direction

Two output pins, **pwm** and **dir**. The duty cycle on **pwm** varies as a function of the input value. **dir** is low for positive inputs and high for negative inputs.

type 2: up/down

Two output pins, **up** and **down**. For positive inputs, the PWM/PDM waveform appears on **up**, while **down** is low. For negative inputs, the waveform appears on **down**, while **up** is low. Suitable for driving the two sides of an H-bridge to generate a bipolar output.

**FUNCTIONS**

**pwmgen.make-pulses** (no floating-point)

Generates the actual PWM waveforms, using information computed by **update**. Must be called as frequently as possible, to maximize the attainable PWM frequency and resolution, and minimize jitter. Operates on all channels at once.

**pwmgen.update** (uses floating point)

Accepts an input value, performs scaling and limit checks, and converts it into a form usable by **make-pulses** for PWM/PDM generation. Can (and should) be called less frequently than **make-pulses**. Operates on all channels at once.

**PINS**

**pwmgen.N.enable** bit in

Enables PWM generator *N* - when false, all **pwmgen.N** output pins are low.

**pwmgen.N.value** float in

Commanded value. When **value** = 0.0, duty cycle is 0%, and when **value** = +/-**scale**, duty cycle is +/- 100%. (Subject to **min-dc** and **max-dc** limitations.)

**pwmgen.N.pwm** bit out (output types 0 and 1 only)

PWM/PDM waveform.

**pwmgen.N.dir** bit out (output type 1 only)

Direction output: low for forward, high for reverse.

**pwmgen.N.up** bit out (output type 2 only)

PWM/PDM waveform for positive input values, low for negative inputs.

**pwmgen.N.down** bit out (output type 2 only)

PWM/PDM waveform for negative input values, low for positive inputs.

## PARAMETERS

**pwmgen.N.curr-dc** float ro

The current duty cycle, after all scaling and limits have been applied. Range is from -1.0 to +1.0.

**pwmgen.N.max-dc** float rw

The maximum duty cycle. A value of 1.0 corresponds to 100%. This can be useful when using transistor drivers with bootstrapped power supplies, since the supply requires some low time to recharge.

**pwmgen.N.min-dc** float rw

The minimum duty cycle. A value of 1.0 corresponds to 100%. Note that when the pwm generator is disabled, the outputs are constantly low, regardless of the setting of **min-dc**.

**pwmgen.N.scale** float rw

**pwmgen.N.offset** float rw

These parameters provide a scale and offset from the **value** pin to the actual duty cycle. The duty cycle is calculated according to  $dc = (value/scale) + offset$ , with 1.0 meaning 100%.

**pwmgen.N.pwm-freq** float rw

PWM frequency in Hz. The upper limit is half of the frequency at which **make-pulses** is invoked, and values above that limit will be changed to the limit. If **dither-pwm** is false, the value will be changed to the nearest integer submultiple of the **make-pulses** frequency. A value of zero produces Pulse Density Modulation instead of Pulse Width Modulation.

**pwmgen.N.dither-pwm** bit rw

Because software-generated PWM uses a fairly slow timebase (several to many microseconds), it has limited resolution. For example, if **make-pulses** is called at a 20KHz rate, and **pwm-freq** is 2KHz, there are only 10 possible duty cycles. If **dither-pwm** is false, the commanded duty cycle will be rounded to the nearest of those values. Assuming **value** remains constant, the same output will repeat every PWM cycle. If **dither-pwm** is true, the output duty cycle will be dithered between the two closest values, so that the long-term average is closer to the desired level. **dither-pwm** has no effect if **pwm-freq** is zero (PDM mode), since PDM is an inherently dithered process.

**NAME**

sample\_hold – Sample and Hold

**SYNOPSIS**

**loadrt sample\_hold [count=*N*|names=*name1*[,*name2*...]]**

**FUNCTIONS**

**sample-hold.*N***

**PINS**

**sample-hold.*N*.in** s32 in

**sample-hold.*N*.hold** bit in

**sample-hold.*N*.out** s32 out

**LICENSE**

GPL

**NAME**

sampler – sample data from HAL in real time

**SYNOPSIS**

**loadrt sampler depth=depth1[,depth2...] cfg=string1[,string2...]**

**DESCRIPTION**

**sampler** and **halsampler**(1) are used together to sample HAL data in real time and store it in a file. **sampler** is a realtime HAL component that exports HAL pins and creates a FIFO in shared memory. It then begins sampling data from the HAL and storing it to the FIFO. **halsampler** is a user space program that copies data from the FIFO to stdout, where it can be redirected to a file or piped to some other program.

**OPTIONS**

**depth=depth1[,depth2...]**

sets the depth of the realtime->user FIFO that **sampler** creates to buffer the realtime data. Multiple values of *depth* (separated by commas) can be specified if you need more than one FIFO (for example if you want to sample data from two different realtime threads).

**cfg=string1[,string2...]**

defines the set of HAL pins that **sampler** exports and later samples data from. One *string* must be supplied for each FIFO, separated by commas. **sampler** exports one pin for each character in *string*. Legal characters are:

**F, f** (float pin)

**B, b** (bit pin)

**S, s** (s32 pin)

**U, u** (u32 pin)

**FUNCTIONS**

**sampler.N**

One function is created per FIFO, numbered from zero.

**PINS**

**sampler.N.pin.M** input

Pin for the data that will wind up in column *M* of FIFO *N* (and in column *M* of the output file). The pin type depends on the config string.

**sampler.N.curr-depth** s32 output

Current number of samples in the FIFO. When this reaches *depth* new data will begin overwriting old data, and some samples will be lost.

**sampler.N.full** bit output

TRUE when the FIFO *N* is full, FALSE when there is room for another sample.

**sampler.N.enable** bit input

When TRUE, samples are captured and placed in FIFO *N*, when FALSE, no samples are acquired. Defaults to TRUE.

**PARAMETERS**

**sampler.N.overruns** s32 read/write

The number of times that **sampler** has tried to write data to the HAL pins but found no room in the FIFO. It increments whenever **full** is true, and can be reset by the **setp** command.

**sampler.N.sample-num** s32 read/write

A number that identifies the sample. It is automatically incremented for each sample, and can be reset using the **setp** command. The sample number can optionally be printed in the first column of the output from **halsampler**, using the **-t** option. (see **man 1 halsampler**)

## SEE ALSO

**halsampler(1)** **streamer(9)** **halstreamer(1)**

## HISTORY

## BUGS

## AUTHOR

Original version by John Kasunich, as part of the LinuxCNC project. Improvements by several other members of the LinuxCNC development team.

## REPORTING BUGS

Report bugs to [jmkasunich AT users DOT sourceforge DOT net](mailto:jmkasunich@users.sourceforge.net)

## COPYRIGHT

Copyright © 2006 John Kasunich.

This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

**NAME**

scale – LinuxCNC HAL component that applies a scale and offset to its input

**SYNOPSIS**

**loadrt scale** [**count**=*N* | **names**=*name1* [, *name2* ... ]]

**FUNCTIONS**

**scale.N** (requires a floating-point thread)

**PINS**

**scale.N.in** float in

**scale.N.gain** float in

**scale.N.offset** float in

**scale.N.out** float out

out = in \* gain + offset

**LICENSE**

GPL

**NAME**

select8 – 8-bit binary match detector

**SYNOPSIS**

**loadrt select8** [**count**=*N*][**names**=*name1*[,*name2*...]]

**FUNCTIONS**

**select8.N**

**PINS**

**select8.N.sel** s32 in

The number of the output to set TRUE. All other outputs will be set FALSE

**select8.N.out** *M* bit out (*M*=0..7)

Output bits. If enable is set and the sel input is between 0 and 7, then the corresponding output bit will be set true

**PARAMETERS**

**select8.N.enable** bit rw (default: *TRUE*)

Set enable to FALSE to cause all outputs to be set FALSE

**LICENSE**

GPL

**NAME**

serport – Hardware driver for the digital I/O bits of the 8250 and 16550 serial port.

**SYNOPSIS**

**loadrt serport io=addr[,addr...]**

The pin numbers refer to the 9-pin serial pinout. Keep in mind that these ports generally use rs232 voltages, not 0/5V signals.

Specify the I/O address of the serial ports using the module parameter **io=addr[,addr...]**. These ports must not be in use by the kernel. To free up the I/O ports after bootup, install setserial and execute a command like:

```
sudo setserial /dev/ttyS0 uart none
```

but it is best to ensure that the serial port is never used or configured by the Linux kernel by setting a kernel commandline parameter or not loading the serial kernel module if it is a modularized driver.

**FUNCTIONS**

**serport.N.read**

**serport.N.write**

**PINS**

**serport.N.pin-1-in** bit out

Also called DCD (data carrier detect); pin 8 on the 25-pin serial pinout

**serport.N.pin-6-in** bit out

Also called DSR (data set ready); pin 6 on the 25-pin serial pinout

**serport.N.pin-8-in** bit out

Also called CTS (clear to send); pin 5 on the 25-pin serial pinout

**serport.N.pin-9-in** bit out

Also called RI (ring indicator); pin 22 on the 25-pin serial pinout

**serport.N.pin-1-in-not** bit out

Inverted version of pin-1-in

**serport.N.pin-6-in-not** bit out

Inverted version of pin-6-in

**serport.N.pin-8-in-not** bit out

Inverted version of pin-8-in

**serport.N.pin-9-in-not** bit out

Inverted version of pin-9-in

**serport.N.pin-3-out** bit in

Also called TX (transmit data); pin 2 on the 25-pin serial pinout

**serport.N.pin-4-out** bit in

Also called DTR (data terminal ready); pin 20 on the 25-pin serial pinout

**serport.N.pin-7-out** bit in

Also called RTS (request to send); pin 4 on the 25-pin serial pinout

**PARAMETERS**

**serport.N.pin-3-out-invert** bit rw

**serport.N.pin-4-out-invert** bit rw

**serport.N.pin-7-out-invert** bit rw

**serport.N.ioaddr** u32 r

## **LICENSE**

GPL

**NAME**

setsserial - a utility for setting Smart Serial NVRAM parameters.

**SYNOPSIS**

```
loadrt setsserial cmd="set hm2_8i20.001f.nvmaxcurrent 750"
```

**FUNCTIONS**

None

**PINS**

None

**USAGE**

```
loadrt setsserial cmd="{command} {parameter/device} {value/filename}"
```

Commands available are **set** and **flash**.

This utility should be used under halcmd, without LinuxCNC running or any realtime threads running.

A typical command sequence would be:

```
halrun
loadrt hostmot2 use_serial_numbers=1
loadrt hm2_pci config="firmware=hm2/5i23/svss8_8.bit"
show param
loadrt setsserial cmd="set hm2_8i20.001f.nvmaxcurrent 750"
exit
```

This example uses the option to have the hal pins and parameters labelled by the serial number of the remote. This is not necessary but can reduce the scope for confusion. (The serial number is normally on a sticker on the device.)

The next line loads the hm2\_pci driver in the normal way. The hm2\_7i43 driver should work equally well, as should any future 7i80 driver. If the card has already been started up and a firmware has been loaded, then the config string may be omitted.

"show param" is optional, but provides a handy list of all the devices and parameters. It also shows the current values of the parameters which can be useful for determining scaling. u32 pin values are always shown in hex, but new values can be entered in decimal or hex. Use the Ox123ABC format to enter a hex value.

The next line invokes setsserial. This is run in a slightly strange way in order to have kernel-level access to a live Hostmot2 config. It is basically a HAL module that always fails to load. This may lead to error messages being printed to the halcmd prompt. These can often be ignored. All the real feedback is via the dmesg command. It is suggested to have a second terminal window open to run dmesg after each command.

On exiting there will typically be a further error message related to the driver failing to unload setsserial. This can be ignored.

The parameter changes will not show up until the drivers are reloaded. //TODO// Add a "get" command to avoid this problem.

**Flashing Firmware** To flash new firmware to an FPGA card such as the 5i25 or 5i20 the "mesaflash" utility should be used. Setsserial is only useful for changing/updating the firmware on smart-serial remote such as the 8i20. The firmware should be placed somewhere in the /lib/firmware/hm2 tree, where the Linux firmware loading macros can find it.

The flashing routine operates in a realtime thread, and can only send prompts to the user through the kernel

log (dmesg). It is most convenient to open two terminal windows, one for command entry and one to monitor progress.

In the first terminal enter

```
tail -f /var/log/kern.log
```

This terminal will now display status information.

The second window will be used to enter the commands. It is important that LinuxCNC and/or HAL are not already loaded when the process is started. To flash new firmware it is necessary to move a jumper on the smart-serial remote drive and to switch smart-serial communication to a slower baudrate.

A typical command sequence is then

```
halrun
loadrt hostmot2 sserial_baudrate=115200
loadrt hm2_pci config="firmware=hm2/5i23/svss8_8.bit"
loadrt setsserial cmd="flash hm2_5i23.0.8i20.0.1 hm2/8i20/8i20T.BIN"
exit
```

It is not necessary (or useful) to specify a config string in a system using the 5i25 or 6i25 cards.

Note that it is necessary to exit halrun and unload the realtime environment before flashing the next card (exit)

The correct sserial channel name to use can be seen in the dmesg output in the feedback terminal after the loadrt hm2\_pci step of the sequence.

## **LICENSE**

GPL

**NAME**

siggen – signal generator

**SYNOPSIS**

**loadrt siggen** [**num\_chan**=*num* | **names**=*name1*[,*name2*...]]

**DESCRIPTION**

**siggen** is a signal generator that can be used for testing and other applications that need simple waveforms. It produces sine, cosine, triangle, sawtooth, and square waves of variable frequency, amplitude, and offset, which can be used as inputs to other HAL components.

**siggen** supports a maximum of sixteen channels. The number of channels actually loaded is set by the **num\_chan** argument when the module is loaded. Alternatively, specify **names=** and unique names separated by commas.

The **num\_chan=** and **names=** specifiers are mutually exclusive. If neither **num\_chan=** nor **names=** are specified, the default value is one.

**NAMING**

The names for pins, parameters, and functions are prefixed as:

**siggen.N.** for N=0,1,...,num-1 when using **num\_chan=num**

**nameN.** for nameN=name1,name2,... when using **names=name1,name2,...**

The **siggen.N.** format is shown in the following descriptions.

**FUNCTIONS**

**siggen.N.update** (uses floating-point)

Updates output pins for signal generator *N*. Each time it is called it calculates a new sample. It should be called many times faster than the desired signal frequency, to avoid distortion and aliasing.

**PINS**

**siggen.N.frequency** float in

The output frequency for signal generator *N*, in Hertz. The default value is 1.0 Hertz.

**siggen.N.amplitude** float in

The output amplitude for signal generator *N*. If **offset** is zero, the outputs will swing from **-amplitude** to **+amplitude**. The default value is 1.00.

**siggen.N.offset** float in

The output offset for signal generator *N*. This value is added directly to the output signal. The default value is zero.

**siggen.N.clock** bit out

The clock output. Bit type clock signal output at the commanded frequency.

**siggen.N.square** float out

The square wave output. Positive while **triangle** and **cosine** are ramping upwards, and while **sine** is negative.

**siggen.N.sine** float out

The sine output. Lags **cosine** by 90 degrees.

**siggen.N.cosine** float out

The cosine output. Leads **sine** by 90 degrees.

**siggen.N.triangle** float out

The triangle wave output. Ramps up while **square** is positive, and down while **square** is negative. Reaches its positive and negative peaks at the same time as **cosine**.

**siggen.N.sawtooth** float out

The sawtooth output. Ramps upwards to its positive peak, then instantly drops to its negative peak and starts ramping again. The drop occurs when **triangle** and **cosine** are at their positive peaks, and coincides with the falling edge of **square**.

## PARAMETERS

None

**NAME**

`sim_axis_hardware` – A component to simulate home and limit switches

**SYNOPSIS**

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

**DESCRIPTION**

This component creates simulated home and limit switches based on the current position. It currently can supply simulation for X,Y,Z and A axes.

**FUNCTIONS**

`sim-axis-hardware.N.update` (requires a floating-point thread)

**PINS**

`sim-axis-hardware.N.Xcurrent-pos` float in

The current position on the axis - eg connect to axis.0.joint-pos-fb

`sim-axis-hardware.N.Ycurrent-pos` float in

`sim-axis-hardware.N.Zcurrent-pos` float in

`sim-axis-hardware.N.Acurrent-pos` float in

`sim-axis-hardware.N.Xhomesw-pos` float in (default: *I*)

The position of the home switch

`sim-axis-hardware.N.Yhomesw-pos` float in (default: *I*)

`sim-axis-hardware.N.Zhomesw-pos` float in (default: *I*)

`sim-axis-hardware.N.Ahomesw-pos` float in (default: *I*)

`sim-axis-hardware.N.Xmaxsw-upper` float in

The upper range of the maximum limit switch, above this is false.

`sim-axis-hardware.N.Ymaxsw-upper` float in

`sim-axis-hardware.N.Zmaxsw-upper` float in

`sim-axis-hardware.N.Amaxsw-upper` float in

`sim-axis-hardware.N.Xmaxsw-lower` float in

The lower range of the maximum limit switch, below this is false.

`sim-axis-hardware.N.Ymaxsw-lower` float in

`sim-axis-hardware.N.Zmaxsw-lower` float in

`sim-axis-hardware.N.Amaxsw-lower` float in

`sim-axis-hardware.N.Xminsw-upper` float in

The upper range of the minimum limit switch, above this is false.

`sim-axis-hardware.N.Yminsw-upper` float in

`sim-axis-hardware.N.Zminsw-upper` float in

`sim-axis-hardware.N.Aminsw-upper` float in

`sim-axis-hardware.N.Xminsw-lower` float in

The lower range of the minimum limit switch, below this is false.

`sim-axis-hardware.N.Yminsw-lower` float in

`sim-axis-hardware.N.Zminsw-lower` float in

`sim-axis-hardware.N.Aminsw-lower` float in

`sim-axis-hardware.N.Xhomesw-hyst` float in (default: *.02*)

range that home switch will be true +/- half this to the home position

`sim-axis-hardware.N.Yhomesw-hyst` float in (default: *.02*)

`sim-axis-hardware.N.Zhomesw-hyst` float in (default: *.02*)

`sim-axis-hardware.N.Ahomesw-hyst` float in (default: *.02*)

**sim-axis-hardware.N.Xhomesw-out** bit out  
Home switch for the X axis

**sim-axis-hardware.N.Yhomesw-out** bit out

**sim-axis-hardware.N.Zhomesw-out** bit out

**sim-axis-hardware.N.Ahomesw-out** bit out

**sim-axis-hardware.N.homesw-all** bit out

**sim-axis-hardware.N.Xmaxsw-out** bit out

Max limit switch

**sim-axis-hardware.N.Xminsw-out** bit out

min limit switch

**sim-axis-hardware.N.Xbothsw-out** bit out

True for both max and min limit switch

**sim-axis-hardware.N.Ymaxsw-out** bit out

**sim-axis-hardware.N.Yminsw-out** bit out

**sim-axis-hardware.N.Ybothsw-out** bit out

**sim-axis-hardware.N.Zmaxsw-out** bit out

**sim-axis-hardware.N.Zminsw-out** bit out

**sim-axis-hardware.N.Zbothsw-out** bit out

**sim-axis-hardware.N.Amaxsw-out** bit out

**sim-axis-hardware.N.Aminsw-out** bit out

**sim-axis-hardware.N.Abothsw-out** bit out

**sim-axis-hardware.N.limitsw-all** bit out

**sim-axis-hardware.N.limitsw-homesw-all** bit out

True for all limits and all home.

**sim-axis-hardware.N.Xmaxsw-homesw-out** bit out

**sim-axis-hardware.N.Xminsw-homesw-out** bit out

**sim-axis-hardware.N.Xbothsw-homesw-out** bit out

**sim-axis-hardware.N.Ymaxsw-homesw-out** bit out

**sim-axis-hardware.N.Yminsw-homesw-out** bit out

**sim-axis-hardware.N.Ybothsw-homesw-out** bit out

**sim-axis-hardware.N.Zmaxsw-homesw-out** bit out

**sim-axis-hardware.N.Zminsw-homesw-out** bit out

**sim-axis-hardware.N.Zbothsw-homesw-out** bit out

**sim-axis-hardware.N.Amaxsw-homesw-out** bit out

**sim-axis-hardware.N.Aminsw-homesw-out** bit out

**sim-axis-hardware.N.Abothsw-homesw-out** bit out

**sim-axis-hardware.N.limit-offset** float in (default: *.01*)

how much the limit switches are offset from inputed position. added to max, subtracted from min

## AUTHOR

Chris Morley

## LICENSE

GPL

**NAME**

`sim_encoder` – simulated quadrature encoder

**SYNOPSIS**

**loadrt** `sim_encoder` [**num\_chan**=*num* | **names**=*name1*[,*name2*...]]

**DESCRIPTION**

**sim\_encoder** can generate quadrature signals as if from an encoder. It also generates an index pulse once per revolution. It is mostly used for testing and simulation, to replace hardware that may not be available. It has a limited maximum frequency, as do all software based pulse generators.

**sim\_encoder** supports a maximum of eight channels. The number of channels actually loaded is set by the **num\_chan**= argument when the module is loaded. Alternatively, specify **names**= and unique names separated by commas.

The **num\_chan**= and **names**= specifiers are mutually exclusive. If neither **num\_chan**= nor **names**= are specified, the default value is one.

**FUNCTIONS**

**sim-encoder.make-pulses** (no floating-point)

Generates the actual quadrature and index pulses. Must be called as frequently as possible, to maximize the count rate and minimize jitter. Operates on all channels at once.

**sim-encoder.update-speed** (uses floating-point)

Reads the **speed** command and other parameters and converts the data into a form that can be used by **make-pulses**. Changes take effect only when **update-speed** runs. Can (and should) be called less frequently than **make-pulses**. Operates on all channels at once.

**NAMING**

The names for pins and parameters are prefixed as:

**sim-encoder.N.** for N=0,1,...,num-1 when using **num\_chan=num**

**nameN.** for nameN=name1,name2,... when using **names=name1,name2,...**

The **sim-encoder.N.** format is shown in the following descriptions.

**PINS**

**sim-encoder.N.phase-A** bit out

One of the quadrature outputs.

**sim-encoder.N.phase-B** bit out

The other quadrature output.

**sim-encoder.N.phase-Z** bit out

The index pulse.

**sim-encoder.N.speed** float in

The desired speed of the encoder, in user units per per second. This is divided by **scale**, and the result is used as the encoder speed in revolutions per second.

**PARAMETERS**

**sim-encoder.N.ppr** u32 rw

The pulses per revolution of the simulated encoder. Note that this is pulses, not counts, per revolution. Each pulse or cycle from the encoder results in four counts, because every edge is counted. Default value is 100 ppr, or 400 counts per revolution.

**sim-encoder.N.scale** float rw

Scale factor for the **speed** input. The **speed** value is divided by **scale** to get the actual encoder speed in revolutions per second. For example, if **scale** is set to 60, then **speed** is in revolutions per minute (RPM) instead of revolutions per second. The default value is 1.00.

**NAME**

sim\_home\_switch – Simple home switch simulator

**SYNOPSIS**

**loadrt sim\_home\_switch [count=*N* | names=*name1* [, *name2* ...]]**

**DESCRIPTION**

After tripping home switch, travel in opposite direction is required (amount set by the hysteresis pin)

**FUNCTIONS**

**sim-home-switch.*N*** (requires a floating-point thread)

**PINS**

**sim-home-switch.*N*.cur-pos** float in  
Current position (typically: axis.n.motor-pos-fb)

**sim-home-switch.*N*.home-pos** float in (default: *I*)  
Home switch position

**sim-home-switch.*N*.hysteresis** float in (default: *0.1*)  
Travel required to backoff (hysteresis)

**sim-home-switch.*N*.home-sw** bit out  
Home switch activated

**LICENSE**

GPL

**NAME**

`sim_parport` – A component to simulate the pins of the `hal_parport` component

**SYNOPSIS**

**loadrt `sim_parport` [`count=N` | `names=name1[,name2...]`]**

**DESCRIPTION**

`Sim_parport` is used to replace the pins of a real parport without changing any of the pins names in the rest of the config.

It has pass-through pins (ending in `-fake`) that allows connecting to other components.

eg `pin-02-in` will follow `pin-02-in-fake` 's logic.

`pin_01_out-fake` will follow `pin_01_out` (possibly modified by `pin_01_out-invert`)

It creates all possible pins of both 'in' and 'out' options of the `hal_parport` component.

This allows using other hardware I/O in place of the parport (without having to change the rest of the config)

or simulating hardware such as limit switches.

it's primary use is in Stepconf for building simulated configs.

You must use the `names=` option to have the right pin names.

eg. `names=parport.0,parport.1`

The read and write functions pass the logic from pins to fake pins or vice versa

The reset function is a no operation.

**FUNCTIONS**

**`sim-parport.N.read`**

**`sim-parport.N.write`**

**`sim-parport.N.reset`**

**PINS**

**`sim-parport.N.pin-01-out`** bit in

**`sim-parport.N.pin-02-out`** bit in

**`sim-parport.N.pin-03-out`** bit in

**`sim-parport.N.pin-04-out`** bit in

**`sim-parport.N.pin-05-out`** bit in

**`sim-parport.N.pin-06-out`** bit in

**`sim-parport.N.pin-07-out`** bit in

**`sim-parport.N.pin-08-out`** bit in

**`sim-parport.N.pin-09-out`** bit in

**`sim-parport.N.pin-14-out`** bit in

**`sim-parport.N.pin-16-out`** bit in

**`sim-parport.N.pin-17-out`** bit in

**`sim-parport.N.pin-01-out-fake`** bit out

**`sim-parport.N.pin-02-out-fake`** bit out

**`sim-parport.N.pin-03-out-fake`** bit out

**`sim-parport.N.pin-04-out-fake`** bit out

**`sim-parport.N.pin-05-out-fake`** bit out

**`sim-parport.N.pin-06-out-fake`** bit out

**`sim-parport.N.pin-07-out-fake`** bit out

**sim-parport.N.pin-08-out-fake** bit out  
**sim-parport.N.pin-09-out-fake** bit out  
**sim-parport.N.pin-14-out-fake** bit out  
**sim-parport.N.pin-16-out-fake** bit out  
**sim-parport.N.pin-17-out-fake** bit out  
**sim-parport.N.pin-02-in** bit out  
**sim-parport.N.pin-03-in** bit out  
**sim-parport.N.pin-04-in** bit out  
**sim-parport.N.pin-05-in** bit out  
**sim-parport.N.pin-06-in** bit out  
**sim-parport.N.pin-07-in** bit out  
**sim-parport.N.pin-08-in** bit out  
**sim-parport.N.pin-09-in** bit out  
**sim-parport.N.pin-10-in** bit out  
**sim-parport.N.pin-11-in** bit out  
**sim-parport.N.pin-12-in** bit out  
**sim-parport.N.pin-13-in** bit out  
**sim-parport.N.pin-15-in** bit out  
**sim-parport.N.pin-02-in-fake** bit in  
**sim-parport.N.pin-03-in-fake** bit in  
**sim-parport.N.pin-04-in-fake** bit in  
**sim-parport.N.pin-05-in-fake** bit in  
**sim-parport.N.pin-06-in-fake** bit in  
**sim-parport.N.pin-07-in-fake** bit in  
**sim-parport.N.pin-08-in-fake** bit in  
**sim-parport.N.pin-09-in-fake** bit in  
**sim-parport.N.pin-10-in-fake** bit in  
**sim-parport.N.pin-11-in-fake** bit in  
**sim-parport.N.pin-12-in-fake** bit in  
**sim-parport.N.pin-13-in-fake** bit in  
**sim-parport.N.pin-15-in-fake** bit in  
**sim-parport.N.pin-02-in-invert** bit out  
**sim-parport.N.pin-03-in-invert** bit out  
**sim-parport.N.pin-04-in-invert** bit out  
**sim-parport.N.pin-05-in-invert** bit out  
**sim-parport.N.pin-06-in-invert** bit out  
**sim-parport.N.pin-07-in-invert** bit out  
**sim-parport.N.pin-08-in-invert** bit out  
**sim-parport.N.pin-09-in-invert** bit out  
**sim-parport.N.pin-10-in-invert** bit out  
**sim-parport.N.pin-11-in-invert** bit out  
**sim-parport.N.pin-12-in-invert** bit out  
**sim-parport.N.pin-13-in-invert** bit out  
**sim-parport.N.pin-15-in-invert** bit out  
**sim-parport.N.reset-time** float in

## PARAMETERS

**sim-parport.N.pin-01-out-invert** bit rw  
**sim-parport.N.pin-02-out-invert** bit rw  
**sim-parport.N.pin-03-out-invert** bit rw  
**sim-parport.N.pin-04-out-invert** bit rw  
**sim-parport.N.pin-05-out-invert** bit rw  
**sim-parport.N.pin-06-out-invert** bit rw

**sim-parport.N.pin-07-out-invert** bit rw  
**sim-parport.N.pin-08-out-invert** bit rw  
**sim-parport.N.pin-09-out-invert** bit rw  
**sim-parport.N.pin-14-out-invert** bit rw  
**sim-parport.N.pin-16-out-invert** bit rw  
**sim-parport.N.pin-17-out-invert** bit rw  
**sim-parport.N.pin-01-out-reset** bit rw  
**sim-parport.N.pin-02-out-reset** bit rw  
**sim-parport.N.pin-03-out-reset** bit rw  
**sim-parport.N.pin-04-out-reset** bit rw  
**sim-parport.N.pin-05-out-reset** bit rw  
**sim-parport.N.pin-06-out-reset** bit rw  
**sim-parport.N.pin-07-out-reset** bit rw  
**sim-parport.N.pin-08-out-reset** bit rw  
**sim-parport.N.pin-09-out-reset** bit rw  
**sim-parport.N.pin-14-out-reset** bit rw  
**sim-parport.N.pin-16-out-reset** bit rw  
**sim-parport.N.pin-17-out-reset** bit rw

**AUTHOR**

Chris Morley

**LICENSE**

GPL

**NAME**

sim\_spindle – Simulated spindle with index pulse

**SYNOPSIS**

**loadrt sim\_spindle** [**count**=*N*][**names**=*name1*[,*name2*...]]

**FUNCTIONS**

**sim-spindle.N** (requires a floating-point thread)

**PINS**

**sim-spindle.N.velocity-cmd** float in  
Commanded speed

**sim-spindle.N.position-fb** float out  
Feedback position, in revolutions

**sim-spindle.N.index-enable** bit io  
Reset **position-fb** to 0 at the next full rotation

**PARAMETERS**

**sim-spindle.N.scale** float rw (default: *1.0*)  
factor applied to **velocity-cmd**.

The result of '**velocity-cmd** \* **scale**' be in revolutions per second. For example, if **velocity-cmd** is in revolutions/minute, **scale** should be set to 1/60 or 0.016666667.

**LICENSE**

GPL

**NAME**

**simple\_tp** – This component is a single axis simple trajectory planner, same as used for jogging in linux-cnc.

**SYNOPSIS**

Used by PNCconf to allow testing of acceleration and velocity values for an axis.

**FUNCTIONS**

**simple-tp.N.update** (requires a floating-point thread)

**PINS**

**simple-tp.N.target-pos** float in  
target position to plan for.

**simple-tp.N.maxvel** float in  
Maximum velocity

**simple-tp.N.maxaccel** float in  
Acceleration rate

**simple-tp.N.enable** bit in  
If disabled, planner sets velocity to zero immediately.

**simple-tp.N.current-pos** float out  
position commanded at this point in time.

**simple-tp.N.current-vel** float out  
velocity commanded at this moment in time.

**simple-tp.N.active** bit out  
if active is true, the planner is requesting movement.

**LICENSE**

GPL

**NAME**

sphereprobe – Probe a pretend hemisphere

**SYNOPSIS**

**loadrt sphereprobe** [**count**=*N* | **names**=*name1* [, *name2* ... ]]

**FUNCTIONS**

**sphereprobe.N**

update probe-out based on inputs

**PINS**

**sphereprobe.N.px** s32 in

**sphereprobe.N.py** s32 in

**sphereprobe.N.pz** s32 in

**rawcounts** position from software encoder

**sphereprobe.N.cx** s32 in

**sphereprobe.N.cy** s32 in

**sphereprobe.N.cz** s32 in

Center of sphere in counts

**sphereprobe.N.r** s32 in

Radius of hemisphere in counts

**sphereprobe.N.probe-out** bit out

**AUTHOR**

Jeff Epler

**LICENSE**

GPL

**NAME**

hostmot2 - Smart Serial LinuxCNC HAL driver for the Mesa Electronics HostMot2 Smart-Serial remote cards

**SYNOPSIS**

The Mesa Smart-Serial interface is a 2.5Mbps proprietary interface between the Mesa Anything-IO cards and a range of subsidiary devices termed "smart-serial remotes". The remote cards perform a variety of functions, but typically they combine various classes of IO. The remot cards are self-configuring, in that they tell the main LinuxCNC Hostmot2 driver what their pin functions are and what they should be named.

Many sserial remotes offer different pinouts depending on the mode they are started up in. This is set using the sserial\_port\_N= option in the hm2\_pci modparam. See the hostmot2 manpage for further details.

It is likely that this documentation will be permanently out of date.

Each Anything-IO board can attach up to 8 sserial remotes to each header (either the 5-pin headers on the 5i20/5i22/5i23/7i43 or the 25-pin connectors on the 5i25, 6i25 and 7i80). The remotes are grouped into "ports" of up to 8 "channels". Typically each header will be a single 8 channel port, but this is not necessarily always the case.

**PORTS**

In addition to the per-channel/device pins detailed below there are three per-port pins and three parameters.

Pins:

(bit, in) .sserial.port-N.run: Enables the specific Smart Serial module. Setting this pin low will disable all boards on the port and puts the port in a pass-through mode where device parameter setting is possible. This pin defaults to TRUE and can be left unconnected. However, toggling the pin low-to-high will re-enable a faulted drive so the pin could usefully be connected to the iocontrol.0.user-enable-out pin.

(u32, ro) .run\_state: Shows the state of the sserial communications state-machine. This pin will generally show a value of 0x03 in normal operation, 0x07 in setup mode and 0x00 when the "run" pin is false.

(u32, ro) .error-count: Indicates the state of the Smart Serial error handler, see the parameters sections for more details.

Parameters:

(u32 r/w) .fault-inc: Any over-run or handshaking error in the SmartSerial communications will increment the .fault-count pin by the amount specified by this parameter. Default = 10.

(u32 r/w) .fault-dec: Every successful read/write cycle decrements the fault counter by this amount. Default = 1.

(u32 r/w) .fault-lim: When the fault counter reaches this threshold the Smart Serial interface on the corresponding port will be stopped and an error printed in dmesg. Together these three pins allow for control over the degree of fault- tolerance allowed in the interface. The default values mean that if more than one transaction in ten fails, more than 20 times, then a hard error will be raised. If the increment were to be set to zero then no error would ever be raised, and the system would carry on regardless. Conversely setting decrement to zero, threshold to 1 and limit to 1 means that absolutely no errors will be tolerated. (This structure is copied directly from vehicle ECU practice)

## DEVICES

The other pins and parameters created in HAL depend on the devices detected. The following list of Smart Serial devices is by no means exhaustive.

**8i20** The 8i20 is a 2.2kW three-phase drive for brushless DC motors and AC servo motors. 8i20 pins and parameters have names like "hm2\_<BoardType>.<BoardNum>.8i20.<PortNum>.<Chan-Num>.<Pin>", for example "hm2\_5i23.0.8i20.1.3.current" would set the phase current for the drive connected to the fourth channel of the second sserial port of the first 5i23 board. Note that the sserial ports do not necessarily correlate in layout or number to the physical ports on the card.

Pins:

(float in) angle

The rotor angle of the motor in fractions of a full **phase** revolution. An angle of 0.5 indicates that the motor is half a turn / 180 degrees /  $\pi$  radians from the zero position. The zero position is taken to be the position that the motor adopts under no load with a positive voltage applied to the A (or U) phase and both B and C (or V and W) connected to -V or 0V. A 6 pole motor will have 3 zero positions per physical rotation. Note that the 8i20 drive automatically adds the phase lead/lag angle, and that this pin should see the raw rotor angle. There is a HAL module (bldc) which handles the complexity of differing motor and drive types.

(float, in) current

The phase current command to the drive. This is scaled from -1 to +1 for forwards and reverse maximum currents. The absolute value of the current is set by the max\_current parameter.

(float, ro) voltage

The drive bus voltage in V. This will tend to show 25.6V when the drive is unpowered and the drive will not operate below about 50V.

(float, ro) temp

The temperature of the driver in degrees C.

(u32, ro) comms

The communication status of the drive. See the manual for more details.

(bit, ro) status and fault.

The following fault/status bits are exported. For further details see the 8i20 manual. fault.U-current / fault.U-current-not / fault.V-current / fault.V-current-not / fault.W-current / fault.W-current-not / fault.bus-high / fault.bus-high-not / fault.bus-overv / fault.bus-overv-not / fault.bus-underv / fault.bus-underv-not / fault.framingr / fault.framingr-not / fault.module / fault.module-not / fault.no-enable / fault.no-enable-not / fault.overcurrent / fault.overcurrent-not / fault.overrun / fault.overrun-not / fault.overtemp / fault.overtemp-not / fault.watchdog / fault.watchdog-not

status.brake-old / status.brake-old-not / status.brake-on / status.brake-on-not / status.bus-underv / status.bus-underv-not / status.current-lim / status.current-lim-no / status.ext-reset / status.ext-reset-not / status.no-enable / status.no-enable-not / status.pid-on / status.pid-on-not / status.sw-reset / status.sw-reset-not / status.wd-reset / status.wd-reset-not

Parameters:

The following parameters are exported. See the pdf documentation downloadable from Mesa for further details

```

hm2_5i25.0.8i20.0.1.angle-maxlim
hm2_5i25.0.8i20.0.1.angle-minlim
hm2_5i25.0.8i20.0.1.angle-scalemax
hm2_5i25.0.8i20.0.1.current-maxlim
hm2_5i25.0.8i20.0.1.current-minlim
hm2_5i25.0.8i20.0.1.current-scalemax
hm2_5i25.0.8i20.0.1.nvbrakeoffv
hm2_5i25.0.8i20.0.1.nvbrakeonv
hm2_5i25.0.8i20.0.1.nvbusoverv
hm2_5i25.0.8i20.0.1.nvbusundervmax
hm2_5i25.0.8i20.0.1.nvbusundervmin
hm2_5i25.0.8i20.0.1.nvkdihi
hm2_5i25.0.8i20.0.1.nvkdil
hm2_5i25.0.8i20.0.1.nvkdilo
hm2_5i25.0.8i20.0.1.nvkdp
hm2_5i25.0.8i20.0.1.nvkqihi
hm2_5i25.0.8i20.0.1.nvkqil
hm2_5i25.0.8i20.0.1.nvkqilo
hm2_5i25.0.8i20.0.1.nvkqp
hm2_5i25.0.8i20.0.1.nvmaxcurrent
hm2_5i25.0.8i20.0.1.nvrembaudrate
hm2_5i25.0.8i20.0.1.swrevision
hm2_5i25.0.8i20.0.1.unitnumber

```

(float, rw) max\_current

Sets the maximum drive current in Amps. The default value is the maximum current programmed into the drive EEPROM. The value must be positive, and an error will be raised if a current in excess of the drive maximum is requested.

(u32, ro) serial\_number

The serial number of the connected drive. This is also shown on the label on the drive.

**7i64** The 7i64 is a 24-input 24-output IO card. 7i64 pins and parameters have names like "hm2\_<BoardType>.<BoardNum>.7i64.<PortNum>.<ChanNum>.<Pin>", for example hm2\_5i23.0.7i64.1.3.output-01

Pins: (bit, in) 7i64.0.0.output-NN: Writing a 1 or TRUE to this pin will enable output driver NN. Note that the outputs are drivers (switches) rather than voltage outputs. The LED adjacent to the connector on the board shows the status. The output can be inverted by setting a parameter.

(bit, out) 7i64.0.0.input-NN: The value of input NN. Note that the inputs are isolated and both pins of each input must be connected (typically to signal and the ground of the signal. This need not be the ground of the board.)

(bit, out) 7i64.0.0.input-NN-not: An inverted copy of the corresponding input.

(float, out) 7i64.0.0.analog0 & 7i64.0.0.analog1: The two analogue inputs (0 to 3.3V) on the board.

Parameters: (bit, rw) 7i64.0.0.output-NN-invert: Setting this parameter to 1 / TRUE will invert the output value, such that writing 0 to .gpio.NN.out will enable the output and vice-versa.

**7i76** The 7i76 is not only a smart-serial device. It also serves as a breakout for a number of other Hostmot2 functions. There are connections for 5 step generators (for which see the main hostmot2 manpage). The stepgen pins are associated with the 5i25 (hm2\_5i25.0.stepgen.00....) whereas the smart-serial pins are associated with the 7i76 (hm2\_5i25.0.7i76.0.0.output-00).

Pins:

(float out) .7i76.0.0.analogN (modes 1 and 2 only) Analogue input values.

(float out) .7i76.0.0.fieldvoltage (mode 2 only) Field voltage monitoring pin.

(bit in) .7i76.0.0.spindir: This pin provides a means to drive the spindle VFD direction terminals on the 7i76 board.

(bit in) .7i76.0.0.spinena: This pin drives the spindle-enable terminals on the 7i76 board.

(float in) .7i76.0.0.spinout: This controls the analogue output of the 7i76. This is intended as a speed control signal for a VFD.

(bit out) .7i76.0.0.output-NN: (NN = 0 to 15). 16 digital outputs. The sense of the signal can be set via a parameter

(bit out) .7i76.0.0.input-NN: (NN = 0 to 31) 32 digital inputs.

(bit in) .7i76.0.0.input-NN-not: (NN = 0 to 31) An inverted copy of the inputs provided for convenience. The two complementary pins may be connected to different signal nets.

Parameters:

(u32 ro) .7i76.0.0.nvbaudrate: Indicates the vbaud rate. This probably should not be altered, and special utils are needed to do so.

(u32 ro) .7i76.0.0.nvunitnumber: Indicates the serial number of the device and should match a siticker on the card. This can be useful for working out which card is which.

(u32 ro) .7i76.0.0.nvwatchdogtimeout: The sserial remote watchdog timeout. This is separate from the Anything-IO card timeout. This is unlikely to need to be changed.

(bit rw) .7i76.0.0.output-NN-invert: Invert the sense of the corresponding output pin.

(bit rw) .7i76.0.0.spindir-invert: Invert the sense of the spindle direction pin.

(bit rw) .7i76.0.0.spinena-invert: Invert the sense of the spindle-enable pin.

(float rw) .7i76.0.0.spinout-maxlim: The maximum speed request allowable

(float rw) .7i76.0.0.spinout-minlim: The minimum speed request.

(float rw) .7i76.0.0.spinout-scalemax: The spindle speed scaling. This is the speed request which would correspond to full-scale output from the spindle control pin. For example with a 10V drive voltage and a 10000rpm scalemax a value of 10,000 rpm on the spinout pin would produce 10V output. However, if spinout-maxlim were set to 5,000 rpm then no voltage above 5V would be output.

(u32 ro) .7i76.0.0.swrevision: The onboard firmware revision number. Utilities exist to update and change this firmware.

**7i77** The 7i77 is an 6-axis servo control card. The analogue outputs are smart-serial devices but the encoders are conventional hostmot2 encoders and further details of them may be found in the hostmot2 manpage.

Pins: (bit out) .7i77.0.0.input-NN: (NN = 0 to 31) 32 digital inputs.

(bit in) .7i77.0.0.input-NN-not: (NN = 0 to 31) An inverted copy of the inputs provided for convenience. The two complementary pins may be connected to different signal nets.

(bit out) .7i77.0.0.output-NN: (NN = 0 to 15). 16 digital outputs. The sense of the signal can be set via a parameter

(bit in) .7i77.0.0.spindir: This pin provides a means to drive the spindle VFD direction terminals on the 7i76 board.

(bit in) .7i77.0.0.spinena: This pin drives the spindle-enable terminals on the 7i76 board.

(float in) .7i77.0.0.spinout: This controls the analog output of the 7i77. This is intended as a speed control signal for a VFD.

(bit in) .7i77.0.1.analogena: This pin drives the analog enable terminals on the 7i77 board.

(float in) .7i77.0.1.analogoutN: (N = 0 to 5) This controls the analog output of the 7i77.

Parameters: (bit rw) .7i77.0.0.output-NN-invert: Invert the sense of the corresponding output pin.

(bit rw) .7i77.0.0.spindir-invert: Invert the sense of the spindle direction pin.

(bit rw) .7i77.0.0.spinena-invert: Invert the sense of the spindle-enable pin.

(float rw) .7i77.0.0.spinout-maxlim: The maximum speed request allowable

(float rw) .7i77.0.0.spinout-minlim: The minimum speed request.

(float rw) .7i77.0.0.spinout-scalemax: The spindle speed scaling. This is the speed request which would correspond to full-scale output from the spindle control pin. For example with a 10V drive voltage and a 10000rpm scalemax a value of 10,000 rpm on the spinout pin would produce 10V output. However, if spinout-maxlim were set to 5,000 rpm then no voltage above 5V would be output.

(float rw) .7i77.0.0.analogoutN-maxlim: (N = 0 to 5) The maximum speed request allowable

(float rw) .7i77.0.0.analogoutN-minlim: (N = 0 to 5) The minimum speed request.

//// \*\*\*\*\* CHECK ME \*\*\*\*\* I'm not sure about the description on analogoutN-scalemax ////

(float rw) .7i77.0.0.analogoutN-scalemax: (N = 0 to 5) The analog speed scaling. This is the speed request which would correspond to full-scale output from the spindle control pin. For example with a 10V drive voltage and a 10000rpm scalemax a value of 10,000 rpm on the spinout pin would produce 10V output. However, if spinout-maxlim were set to 5,000 rpm then no voltage above 5V would be output.

**7i69** The 7i69 is a 48 channel digital IO card. It can be configured in four different modes: Mode 0 B 48 pins bidirectional (all outputs can be set high then driven low to work as inputs)  
 Mode 1 48 pins, input only  
 Mode 2 48 pins, all outputs  
 Mode 3 24 inputs and 24 outputs.

Pins: (bit in) .7i69.0.0.output-NN: Digital output. Sense can be inverted with the corresponding Parameter

(bit out) .7i69.0.0.input-NN: Digital input

(bit out) .7i69.0.0.input-NN-not: Digital input, inverted.

Parameters:

(u32 ro) .7i69.0.0.nvbaudrate: Indicates the vbaud rate. This probably should not be altered, and special utils are needed to do so.

(u32 ro) .7i69.0.0.nvunitnumber: Indicates the serial number of the device and should match a siticker on the card. This can be useful for wokring out which card is which.

(u32 ro) .7i69.0.0.nvwatchdogtimeout: The sserial remote watchdog timeout. This is separate from the Anything-IO card timeout. This is unlikley to need to be changed.

(bit rw) .7i69.0.0.output-NN-invert: Invert the sense of the corresponding output pin.

(u32 ro) .7i69.0.0.swrevision: The onboard firmware revision number. Utilities exist to update and change this firmware.

## 7i70

The 7i70 is a remote isolated 48 input card. The 7i70 inputs sense positive inputs relative to a common field ground. Input impedance is 10K Ohms and input voltage can range from 5VDC to 32VDC. All inputs have LED status indicators. The input common field ground is galvanically isolated from the communications link.

The 7i70 has three software selectable modes. These different modes select different sets of 7i70 data to be transferred between the host and the 7i70 during real time process data exchanges. For high speed applications, choosing the correct mode can reduced the data transfer sizes, resulting in higher maximum update rates.

MODE 0 Input mode (48 bits input data only)

MODE 1 Input plus analog mode (48 bits input data plus 6 channels of analog data)

MODE 2 Input plus field voltage

Pins:

(float out) .7i70.0.0.analogN (modes 1 and 2 only) Analogue input values.

(float out) .7i70.0.0.fieldvoltage (mode 2 only) Field voltage monitoring pin.

(bit out) .7i70.0.0.input-NN: (NN = 0 to 47) 48 digital inputs.

(bit in) .7i70.0.0.input-NN-not: (NN = 0 to 47) An inverted copy of the inputs provided for

convenience. The two complementary pins may be connected to different signal nets.

#### Parameters:

(u32 ro) .7i70.0.0.nvbaudrate: Indicates the vbaud rate. This probably should not be altered, and special utils are needed to do so.

(u32 ro) .7i70.0.0.nvunitnumber: Indicates the serial number of the device and should match a sticker on the card. This can be useful for working out which card is which.

(u32 ro) .7i70.0.0.nvwatchdogtimeout: The sserial remote watchdog timeout. This is separate from the Anything-IO card timeout. This is unlikely to need to be changed.

(u32 ro) .7i69.0.0.swrevision: The onboard firmware revision number. Utilities exist to update and change this firmware.

## 7i71

The 7i71 is a remote isolated 48 output card. The 48 outputs are 8VDC to 28VDC sourcing drivers (common + field power) with 300 mA maximum current capability. All outputs have LED status indicators.

The 7i71 has two software selectable modes. For high speed applications, choosing the correct mode can reduced the data transfer sizes, resulting in higher maximum update rates

MODE 0 Output only mode (48 bits output data only)

MODE 1 Outputs plus read back field voltage

#### Pins:

(float out) .7i71.0.0.fieldvoltage (mode 2 only) Field voltage monitoring pin.

(bit out) .7i71.0.0.output-NN: (NN = 0 to 47) 48 digital outputs. The sense may be inverted by the invert parameter.

#### Parameters:

(bit rw) .7i71.0.0.output-NN-invert: Invert the sense of the corresponding output pin.

(u32 ro) .7i71.0.0.nvbaudrate: Indicates the vbaud rate. This probably should not be altered, and special utils are needed to do so.

(u32 ro) .7i71.0.0.nvunitnumber: Indicates the serial number of the device and should match a sticker on the card. This can be useful for working out which card is which.

(u32 ro) .7i71.0.0.nvwatchdogtimeout: The sserial remote watchdog timeout. This is separate from the Anything-IO card timeout. This is unlikely to need to be changed.

(u32 ro) .7i69.0.0.swrevision: The onboard firmware revision number. Utilities exist to update and change this firmware.

**7i73** The 7I73 is a remote real time pendant or control panel interface.

The 7I73 supports up to four 50KHz encoder inputs for MPGs, 8 digital inputs and 6 digital outputs and up to a 64 Key keypad. If a smaller keypad is used, more digital inputs and outputs become available. Up to eight 0.0V to 3.3V analog inputs are also provided. The 7I73 can drive a 4 line 20 character LCD for local DRO applications.

The 7I73 has 3 software selectable process data modes. These different modes select different sets of 7I73 data to be transferred between the host and the 7 I73 during real time process data exchanges. For high speed applications, choosing the correct mode can reduced the data transfer sizes, resulting in higher maximum update rates

MODE 0 I/O + ENCODER

MODE 1 I/O + ENCODER +ANALOG IN

MODE 2 I/O + ENCODER +ANALOG IN FAST DISPLAY

Pins:

(float out) .7i73.0.0.analoginN: Analogue inputs. Up to 8 channels may be available dependant on software and hardware configuration modes. (see the pdf manual downlaodable from [www.mesanet.com](http://www.mesanet.com))

(u32 in) .7i73.0.1.display (modes 1 and 2). Data for LCD display. This pin may be conveniently driven by the HAL "lcd" component which allows the formatted display of the values any number of HAL pins and textual content.

(u32 in) .7i73.0.1.display32 (mode 2 only). 4 bytes of data for LCD display. This mode is not supported by the HAL "lcd" component.

(s32 out) .7i73.0.1.encN: The position of the MPG encoder counters.

(bit out) .7i73.0.1.input-NN: Up to 24 digital inputs (dependent on config)

(bit out) .7i73.0.1.input-NN-not: Inverted copy of the digital inputs

(bit in) .7i73.0.1.output-NN: Up to 22 digital outputs (dependent on config)

Parameters:

(u32 ro) .7i73.0.1.nvanalogfilter:

(u32 ro) .7i73.0.1.nvbaudrate

(u32 ro) .7i73.0.1.nvcontrast

(u32 ro) .7i73.0.1.nvdispmode

(u32 ro) .7i73.0.1.nvencmode0

(u32 ro) .7i73.0.1.nvencmode1

(u32 ro) .7i73.0.1.nvencmode2

(u32 ro) .7i73.0.1.nvencmode3

(u32 ro) .7i73.0.1.nvkeytimer

(u32 ro) .7i73.0.1.nvunitnumber

(u32 ro) .7i73.0.1.nvwatchdogtimeout

(u32 ro) .7i73.0.1.output-00-invert

The above parameters are only settable with utility software, for further details of their use see the Mesa manual.

(bit rw) .7i73.0.1.output-01-invert: Invert the corresponding output bit.

(s32 ro) .7i73.0.1.swrevision: The version of firmware installed.

TODO: Add 7i77, 7i66, 7i72, 7i83, 7i84, 7i87.

**NAME**

stepgen – software step pulse generation

**SYNOPSIS**

**loadrt stepgen step\_type=type0[,type1...] [ctrl\_type=type0[,type1...]] [user\_step\_type=#,#...]**

**DESCRIPTION**

**stepgen** is used to control stepper motors. The maximum step rate depends on the CPU and other factors, and is usually in the range of 5KHz to 25KHz. If higher rates are needed, a hardware step generator is a better choice.

**stepgen** has two control modes, which can be selected on a channel by channel basis using **ctrl\_type**. Possible values are "p" for position control, and "v" for velocity control. The default is position control, which drives the motor to a commanded position, subject to acceleration and velocity limits. Velocity control drives the motor at a commanded speed, again subject to accel and velocity limits. Usually, position mode is used for machine axes. Velocity mode is reserved for unusual applications where continuous movement at some speed is desired, instead of movement to a specific position. (Note that velocity mode replaces the former component **freggen**.)

**stepgen** can control a maximum of 16 motors. The number of motors/channels actually loaded depends on the number of *type* values given. The value of each *type* determines the outputs for that channel. Position or velocity mode can be individually selected for each channel. Both control modes support the same 16 possible step types.

By far the most common step type is '0', standard step and direction. Others include up/down, quadrature, and a wide variety of three, four, and five phase patterns that can be used to directly control some types of motor windings. (When used with appropriate buffers of course.)

Some of the stepping types are described below, but for more details (including timing diagrams) see the **stepgen** section of the HAL reference manual.

type 0: step/dir

Two pins, one for step and one for direction. **make-pulses** must run at least twice for each step (once to set the step pin true, once to clear it). This limits the maximum step rate to half (or less) of the rate that can be reached by types 2-14. The parameters **steplen** and **stepspace** can further lower the maximum step rate. Parameters **dirsetup** and **dirhold** also apply to this step type.

type 1: up/down

Two pins, one for 'step up' and one for 'step down'. Like type 0, **make-pulses** must run twice per step, which limits the maximum speed.

type 2: quadrature

Two pins, phase-A and phase-B. For forward motion, A leads B. Can advance by one step every time **make-pulses** runs.

type 3: three phase, full step

Three pins, phase-A, phase-B, and phase-C. Three steps per full cycle, then repeats. Only one phase is high at a time - for forward motion the pattern is A, then B, then C, then A again.

type 4: three phase, half step

Three pins, phases A through C. Six steps per full cycle. First A is high alone, then A and B together, then B alone, then B and C together, etc.

types 5 through 8: four phase, full step

Four pins, phases A through D. Four steps per full cycle. Types 5 and 6 are suitable for use with unipolar steppers, where power is applied to the center tap of each winding, and four open-collector transistors drive the ends. Types 7 and 8 are suitable for bipolar steppers, driven by two H-bridges.

types 9 and 10: four phase, half step

Four pins, phases A through D. Eight steps per full cycle. Type 9 is suitable for unipolar drive, and type 10 for bipolar drive.

types 11 and 12: five phase, full step

Five pins, phases A through E. Five steps per full cycle. See HAL reference manual for the patterns.

types 13 and 14: five phase, half step

Five pins, phases A through E. Ten steps per full cycle. See HAL reference manual for the patterns.

type 15: user-specified

This uses the waveform specified by the **user\_step\_type** module parameter, which may have up to 10 steps and 5 phases.

## FUNCTIONS

**stepgen.make-pulses** (no floating-point)

Generates the step pulses, using information computed by **update-freq**. Must be called as frequently as possible, to maximize the attainable step rate and minimize jitter. Operates on all channels at once.

**stepgen.capture-position** (uses floating point)

Captures position feedback value from the high speed code and makes it available on a pin for use elsewhere in the system. Operates on all channels at once.

**stepgen.update-freq** (uses floating point)

Accepts a velocity or position command and converts it into a form usable by **make-pulses** for step generation. Operates on all channels at once.

## PINS

**stepgen.N.counts** s32 out

The current position, in counts, for channel *N*. Updated by **capture-position**.

**stepgen.N.position-fb** float out

The current position, in length units (see parameter **position-scale**). Updated by **capture-position**. The resolution of **position-fb** is much finer than a single step. If you need to see individual steps, use **counts**.

**stepgen.N.enable** bit in

Enables output steps - when false, no steps are generated.

**stepgen.N.velocity-cmd** float in (velocity mode only)

Commanded velocity, in length units per second (see parameter **position-scale**).

**stepgen.N.position-cmd** float in (position mode only)

Commanded position, in length units (see parameter **position-scale**).

**stepgen.N.step** bit out (step type 0 only)

Step pulse output.

**stepgen.N.dir** bit out (step type 0 only)

Direction output: low for forward, high for reverse.

**stepgen.N.up** bit out (step type 1 only)

Count up output, pulses for forward steps.

**stepgen.N.down** bit out (step type 1 only)

Count down output, pulses for reverse steps.

**stepgen.N.phase-A** thru **phase-E** bit out (step types 2-14 only)

Output bits. **phase-A** and **phase-B** are present for step types 2-14, **phase-C** for types 3-14, **phase-D** for types 5-14, and **phase-E** for types 11-14. Behavior depends on selected stepping type.

## PARAMETERS

**stepgen.N.frequency** float ro

The current step rate, in steps per second, for channel *N*.

**stepgen.N.maxaccel** float rw

The acceleration/deceleration limit, in length units per second squared.

**stepgen.N.maxvel** float rw

The maximum allowable velocity, in length units per second. If the requested maximum velocity cannot be reached with the current combination of scaling and **make-pulses** thread period, it will be reset to the highest attainable value.

**stepgen.N.position-scale** float rw

The scaling for position feedback, position command, and velocity command, in steps per length unit.

**stepgen.N.rawcounts** s32 ro

The position in counts, as updated by **make-pulses**. (Note: this is updated more frequently than the **counts** pin.)

**stepgen.N.steplen** u32 rw

The length of the step pulses, in nanoseconds. Measured from rising edge to falling edge.

**stepgen.N.stepspace** u32 rw (step types 0 and 1 only) The minimum

space between step pulses, in nanoseconds. Measured from falling edge to rising edge. The actual time depends on the step rate and can be much longer. If **stepspace** is 0, then **step** can be asserted every period. This can be used in conjunction with **hal\_parport**'s auto-resetting pins to output one step pulse per period. In this mode, **steplen** must be set for one period or less.

**stepgen.N.dirsetup** u32 rw (step type 0 only)

The minimum setup time from direction to step, in nanoseconds periods. Measured from change of direction to rising edge of step.

**stepgen.N.dirhold** u32 rw (step type 0 only)

The minimum hold time of direction after step, in nanoseconds. Measured from falling edge of step to change of direction.

**stepgen.N.dirdelay** u32 rw (step types 1 and higher only)

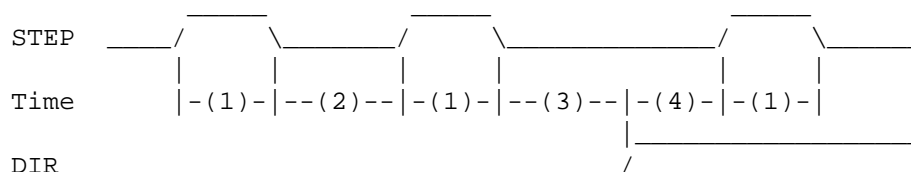
The minimum time between a forward step and a reverse step, in nanoseconds.

## TIMING

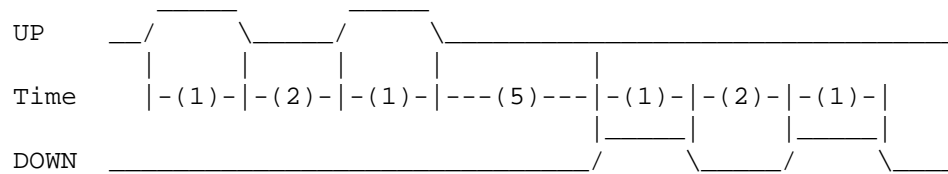
There are five timing parameters which control the output waveform. No step type uses all five, and only those which will be used are exported to HAL. The values of these parameters are in nano-seconds, so no recalculation is needed when changing thread periods. In the timing diagrams that follow, they are identified by the following numbers:

- (1) **stepgen.n.steplen**
- (2) **stepgen.n.stepspace**
- (3) **stepgen.n.dirhold**
- (4) **stepgen.n.dirsetup**
- (5) **stepgen.n.dirdelay**

For step type 0, timing parameters 1 thru 4 are used. The following timing diagram shows the output waveforms, and what each parameter adjusts.



For step type 1, timing parameters 1, 2, and 5 are used. The following timing diagram shows the output waveforms, and what each parameter adjusts.



For step types 2 and higher, the exact pattern of the outputs depends on the step type (see the HAL manual for a full listing). The outputs change from one state to another at a minimum interval of **steplen**. When a direction change occurs, the minimum time between the last step in one direction and the first in the other direction is the sum of **steplen** and **dirdelay**.

### SEE ALSO

The HAL User Manual.

**NAME**

steptest – Used by Stepconf to allow testing of acceleration and velocity values for an axis.

**SYNOPSIS**

**loadrt steptest [count=*N*|names=*name1*[,*name2*...]]**

**FUNCTIONS**

**steptest.*N*** (requires a floating-point thread)

**PINS**

**steptest.*N*.jog-minus** bit in

Drive TRUE to jog the axis in its minus direction

**steptest.*N*.jog-plus** bit in

Drive TRUE to jog the axis in its positive direction

**steptest.*N*.run** bit in

Drive TRUE to run the axis near its current position\_fb with a trapezoidal velocity profile

**steptest.*N*.maxvel** float in

Maximum velocity

**steptest.*N*.maxaccel** float in

Permitted Acceleration

**steptest.*N*.amplitude** float in

Approximate amplitude of positions to command during 'run'

**steptest.*N*.dir** s32 in

Direction from central point to test: 0 = both, 1 = positive, 2 = negative

**steptest.*N*.position-cmd** float out

**steptest.*N*.position-fb** float in

**steptest.*N*.running** bit out

**steptest.*N*.run-target** float out

**steptest.*N*.run-start** float out

**steptest.*N*.run-low** float out

**steptest.*N*.run-high** float out

**steptest.*N*.pause** s32 in (default: 0)

pause time for each end of run in seconds

**PARAMETERS**

**steptest.*N*.epsilon** float rw (default: .001)

**steptest.*N*.elapsed** float r

Current value of the internal timer

**LICENSE**

GPL

**NAME**

streamer – stream file data into HAL in real time

**SYNOPSIS**

**loadrt streamer depth=depth1[,depth2...] cfg=string1[,string2...]**

**DESCRIPTION**

**streamer** and **halstreamer**(1) are used together to stream data from a file into the HAL in real time. **streamer** is a realtime HAL component that exports HAL pins and creates a FIFO in shared memory. **hal\_streamer** is a user space program that copies data from stdin into the FIFO, so that **streamer** can write it to the HAL pins.

**OPTIONS**

**depth=depth1[,depth2...]**

sets the depth of the user->realtime FIFO that **streamer** creates to receive data from **halstreamer**. Multiple values of *depth* (separated by commas) can be specified if you need more than one FIFO (for example if you want to stream data from two different realtime threads).

**cfg=string1[,string2...]**

defines the set of HAL pins that **streamer** exports and later writes data to. One *string* must be supplied for each FIFO, separated by commas. **streamer** exports one pin for each character in *string*. Legal characters are:

**F, f** (float pin)

**B, b** (bit pin)

**S, s** (s32 pin)

**U, u** (u32 pin)

**FUNCTIONS**

**streamer.N**

One function is created per FIFO, numbered from zero.

**PINS**

**streamer.N.pin.M** output

Data from column *M* of the data in FIFO *N* appears on this pin. The pin type depends on the config string.

**streamer.N.curr-depth** s32 output

Current number of samples in the FIFO. When this reaches zero, new data will no longer be written to the pins.

**streamer.N.empty** bit output

TRUE when the FIFO *N* is empty, FALSE when valid data is available.

**streamer.N.enable** bit input

When TRUE, data from FIFO *N* is written to the HAL pins. When false, no data is transferred. Defaults to TRUE.

**streamer.N.underruns** s32 read/write

The number of times that **sampler** has tried to write data to the HAL pins but found no fresh data in the FIFO. It increments whenever **empty** is true, and can be reset by the **setp** command.

**streamer.N.clock** bit input

Clock for data as specified by the clock-mode pin

**streamer.N.clock-mode** s32 input

Defines behavior of clock pin:

- 0 (**default**) free run at every iteration
- 1 clock on falling edge of clock pin
- 2 clock on rising edge of clock pin
- 3 clock on any edge of clock pin

## SEE ALSO

**halstreamer(1)** **sampler(9)** **halsampler(1)**

## HISTORY

### BUGS

Should an **enable** HAL pin be added, to allow streaming to be turned on and off?

## AUTHOR

Original version by John Kasunich, as part of the LinuxCNC project. Improvements by several other members of the LinuxCNC development team.

## REPORTING BUGS

Report bugs to [jmkasunich AT users DOT sourceforge DOT net](mailto:jmkasunich@users.sourceforge.net)

## COPYRIGHT

Copyright © 2006 John Kasunich.

This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

**NAME**

sum2 – Sum of two inputs (each with a gain) and an offset

**SYNOPSIS**

**loadrt sum2** [**count**=*N* | **names**=*name1* [, *name2*...]]

**FUNCTIONS**

**sum2.N** (requires a floating-point thread)

**PINS**

**sum2.N.in0** float in

**sum2.N.in1** float in

**sum2.N.out** float out

$$\text{out} = \text{in0} * \text{gain0} + \text{in1} * \text{gain1} + \text{offset}$$

**PARAMETERS**

**sum2.N.gain0** float rw (default: *1.0*)

**sum2.N.gain1** float rw (default: *1.0*)

**sum2.N.offset** float rw

**LICENSE**

GPL

**NAME**

supply – set output pins with values from parameters (obsolete)

**SYNOPSIS**

**loadrt supply num\_chan=num**

**DESCRIPTION**

**supply** was used to allow the inputs of other HAL components to be manipulated for testing purposes. When it was written, the only way to set the value of an input pin was to connect it to a signal and connect that signal to an output pin of some other component, and then let that component write the pin value. **supply** was written to be that "other component". It reads values from parameters (set with the HAL command **setp**) and writes them to output pins.

Since **supply** was written, the **setp** command has been modified to allow it to set unconnected pins as well as parameters. In addition, the **sets** command was added, which can directly set HAL signals, as long as there are no output pins connected to them. Therefore, **supply** is obsolete.

**supply** supports a maximum of eight channels. The number of channels actually loaded is set by the **num\_chan** argument when the module is loaded. If **numchan** is not specified, the default value is one.

**FUNCTIONS**

**supply.N.update** (uses floating-point)  
Updates output pins for channel *N*.

**PINS**

**supply.N.q** bit out  
Output bit, copied from parameter **supply.N.d**.

**supply.N.\_q** bit out  
Output bit, inverted copy of parameter **supply.N.d**.

**supply.N.variable** float out  
Analog output, copied from parameter **supply.N.value**.

**supply.N.\_variable** float out  
Analog output, equal to -1.0 times parameter **supply.N.value**.

**supply.N.d** bit rw  
Data source for **q** and **\_q** output pins.

**supply.N.value** bit rw  
Data source for **variable** and **\_variable** output pins.

**NAME**

thc – Torch Height Control

**SYNOPSIS**

**loadrt thc**

**DESCRIPTION**

Torch Height Control Mesa THC > Encoder > LinuxCNC THC component

The Mesa THC sends a frequency based on the voltage detected to the encoder. The velocity from the encoder is converted to volts with the velocity scale parameter inside the THC component.

The THCAD card sends a frequency at 0 volts so the scale offset parameter is used to zero the calculated voltage.

Component Functions If enabled and torch is on and X + Y velocity is within tolerance of set speed allow the THC to offset the Z axis as needed to maintain voltage.

If enabled and torch is off and the Z axis is moving up remove any correction at a rate not to exceed the rate of movement of the Z axis.

If enabled and torch is off and there is no correction pass the Z position and feed back untouched.

If not enabled pass the Z position and feed back untouched.

**Physical Connections**

Plasma Torch Arc Voltage Signal => 6 x 487k 1% resistors => THC Arc Voltage In

THC Frequency Signal => Encoder #0, pin A (Input)

Plasma Torch Arc OK Signal => input pin

output pin => Plasma Torch Start Arc Contacts

**HAL Plasma Connections**

encoder.nn.velocity => thc.encoder-vel (tip voltage)

motion.spindle-on => output pin (start the arc)

thc.arc-ok <= motion.digital-in-00 <= input pin (arc ok signal)

**HAL Motion Connections**

thc.requested-vel <= motion.requested-vel

thc.current-vel <= motion.current-vel

**FUNCTIONS**

**thc** (requires a floating-point thread)

**PINS**

**thc.encoder-vel** float in

Connect to hm2\_5i20.0.encoder.00.velocity

**thc.current-vel** float in

Connect to motion.current-vel

**thc.requested-vel** float in

Connect to motion.requested-vel

**thc.volts-requested** float in

Tip Volts current\_vel >= min\_velocity requested

**thc.vel-tol** float in  
Velocity Tolerance (Corner Lock)

**thc.torch-on** bit in  
Connect to motion.spindle-on

**thc.arc-ok** bit in  
Arc OK from Plasma Torch

**thc.enable** bit in  
Enable the THC, if not enabled Z position is passed through

**thc.z-pos-in** float in  
Z Motor Position Command in from axis.n.motor-pos-cmd

**thc.z-pos-out** float out  
Z Motor Position Command Out

**thc.z-fb-out** float out  
Z Position Feedback to Axis

**thc.volts** float out  
The Calculated Volts

**thc.vel-status** bit out  
When the THC thinks we are at requested speed

## PARAMETERS

**thc.vel-scale** float rw  
The scale to convert the Velocity signal to Volts

**thc.scale-offset** float rw  
The offset of the velocity input at 0 volts

**thc.velocity-tol** float rw  
The deviation percent from planned velocity

**thc.voltage-tol** float rw  
The deviation of Tip Voltage before correction takes place

**thc.correction-vel** float rw  
The amount of change in user units per period to move Z to correct

## AUTHOR

John Thornton

## LICENSE

GPLv2 or greater

**NAME**

thcud – Torch Height Control Up/Down Input

**SYNOPSIS**

**loadrt thcud**

**DESCRIPTION**

Torch Height Control This THC takes either an up or a down input from a THC

If enabled and torch is on and X + Y velocity is within tolerance of set speed allow the THC to offset the Z axis as needed to maintain voltage.

If enabled and torch is off and the Z axis is moving up remove any correction at a rate not to exceed the rate of movement of the Z axis.

If enabled and torch is off and there is no correction pass the Z position and feed back untouched.

If not enabled pass the Z position and feed back untouched.

Physical Connections typical parport.0.pin-12-in <= THC controller Plasma Up parport.0.pin-13-in <= THC controller Plasma Down parport.0.pin-15-in <= Plasma Torch Arc Ok Signal parport.0.pin-16-out => Plasma Torch Start Arc Contacts

HAL Plasma Connections thc.torch-up <= parport.0.pin-12-in thc.torch-down <= parport.0.pin-13-in motion.spindle-on => parport.0.pin-16-out (start the arc) thc.arc-ok <= motion.digital-in-00 <= parport.0.pin-15-in (arc ok signal)

HAL Motion Connections thc.requested-vel <= motion.requested-vel thc.current-vel <= motion.current-vel

Pyvcp Connections In the xml file you need something like:

```
<checkboxbutton>
  <text>"THC Enable"</text>
  <halpin>"thc-enable"</halpin> </checkboxbutton> <spinbox>
  <width>"5"</width>
  <halpin>"vel-tol"</halpin>
  <min_>.01</min_>
  <max_>1</max_>
  <resolution>0.01</resolution>
  <initval>0.2</initval>
  <format>"1.2f"</format>
  <font>("Arial",10)</font> </spinbox>
```

Connect the Pyvcp pins in the postgui.hal file like this:

```
net thc-enable thcud.enable <= pyvcp.thc-enable
```

**FUNCTIONS**

**thcud** (requires a floating-point thread)

**PINS**

**thcud.torch-up** bit in

Connect to an input pin

**thcud.torch-down** bit in  
Connect to input pin

**thcud.current-vel** float in  
Connect to motion.current-vel

**thcud.requested-vel** float in  
Connect to motion.requested-vel

**thcud.torch-on** bit in  
Connect to motion.spindle-on

**thcud.arc-ok** bit in  
Arc Ok from Plasma Torch

**thcud.enable** bit in  
Enable the THC, if not enabled Z position is passed through

**thcud.z-pos-in** float in  
Z Motor Position Command in from axis.n.motor-pos-cmd

**thcud.z-pos-out** float out  
Z Motor Position Command Out

**thcud.z-fb-out** float out  
Z Position Feedback to Axis

**thcud.cur-offset** float out  
The Current Offset

**thcud.vel-status** bit out  
When the THC thinks we are at requested speed

**thcud.removing-offset** bit out  
Pin for testing

## PARAMETERS

**thcud.velocity-tol** float rw  
The deviation percent from planned velocity

**thcud.correction-vel** float rw  
The Velocity to move Z to correct

## AUTHOR

John Thornton

## LICENSE

GPLv2 or greater

**NAME**

**threads** – creates hard realtime HAL threads

**SYNOPSIS**

**loadrt threads name1=*name* period1=*period* [fp1=<0|1>] [<thread-2-info>] [<thread-3-info>]**

**DESCRIPTION**

**threads** is used to create hard realtime threads which can execute HAL functions at specific intervals. It is not a true HAL component, in that it does not export any functions, pins, or parameters of its own. Once it has created one or more threads, the threads stand alone, and the **threads** component can be unloaded without affecting them. In fact, it can be unloaded and then reloaded to create additional threads, as many times as needed.

**threads** can create up to three realtime threads. Threads must be created in order, from fastest to slowest. Each thread is specified by three arguments. **name1** is used to specify the name of the first thread (thread 1). **period1** is used to specify the period of thread 1 in nanoseconds. Both *name* and *period* are required. The third argument, **fp1** is optional, and is used to specify if thread 1 will be used to execute floating point code. If not specified, it defaults to **1**, which means that the thread will support floating point. Specify **0** to disable floating point support, which saves a small amount of execution time by not saving the FPU context. For additional threads, **name2**, **period2**, **fp2**, **name3**, **period3**, and **fp3** work exactly the same. If more than three threads are needed, unload threads, then reload it to create more threads.

**FUNCTIONS**

None

**PINS**

None

**PARAMETERS**

None

**BUGS**

The existence of **threads** might be considered a bug. Ideally, creation and deletion of threads would be done directly with **halcmd** commands, such as "**newthread** *name period*", "**delthread** *name*", or similar. However, limitations in the current HAL implementation require thread creation to take place in kernel space, and loading a component is the most straightforward way to do that.

**NAME**

threadtest – LinuxCNC HAL component for testing thread behavior

**SYNOPSIS**

**loadrt threadtest [count=*N* | names=*name1* [, *name2* ...]]**

**FUNCTIONS**

**threadtest.*N*.increment**

**threadtest.*N*.reset**

**PINS**

**threadtest.*N*.count** u32 out

**LICENSE**

GPL

**NAME**

time – Time on in Hours, Minutes, Seconds

**SYNOPSIS**

**loadrt time [count=N|names=name1[,name2...]]**

**DESCRIPTION**

Time

When the time.N.start bit goes true the cycle timer resets and starts to time until time.N.start goes false. If you connect time.N.start to halui.is-running as a cycle timer it will reset during a pause. See the example connections below to keep the timer timing during a pause.

Time returns the hours, minutes, and seconds that time.N.start is true.

Sample pyVCP code to display the hours:minutes:seconds.

```
<pyvcp>
<hbox>
<label>
  <text>"Cycle Time"</text>
  <font>("Helvetica",14)</font>
</label>
<u32>
  <halpin>"time-hours"</halpin>
  <font>("Helvetica",14)</font>
  <format>"2d"</format>
</u32>
<label>
  <text>":"</text>
  <font>("Helvetica",14)</font>
</label>
<u32>
  <halpin>"time-minutes"</halpin>
  <font>("Helvetica",14)</font>
  <format>"2d"</format>
</u32>
<label>
  <text>":"</text>
  <font>("Helvetica",14)</font>
</label>
<u32>
  <halpin>"time-seconds"</halpin>
  <font>("Helvetica",14)</font>
  <format>"2d"</format>
</u32>
</hbox> </pyvcp>
```

In your post-gui.hal file you might use the following to connect it up

```
loadrt time
loadrt not
addf time.0 servo-thread
addf not.0 servo-thread
net prog-running not.0.in <= halui.program.is-idle
net cycle-timer time.0.start <= not.0.out
```

```
net cycle-seconds pyvcp.time-seconds <= time.0.seconds
net cycle-minutes pyvcp.time-minutes <= time.0.minutes
net cycle-hours pyvcp.time-hours <= time.0.hours
```

## FUNCTIONS

**time.N** (requires a floating-point thread)

## PINS

**time.N.start** bit in  
Timer On

**time.N.seconds** u32 out  
Seconds

**time.N.minutes** u32 out  
Minutes

**time.N.hours** u32 out  
Hours

## AUTHOR

John Thornton

## LICENSE

GPL

**NAME**

timedelay – The equivalent of a time-delay relay

**SYNOPSIS**

**loadrt timedelay** [**count**=*N* | **names**=*name1* [, *name2* ... ]]

**FUNCTIONS**

**timedelay.N** (requires a floating-point thread)

**PINS**

**timedelay.N.in** bit in

**timedelay.N.out** bit out

Follows the value of **in** after applying the delays **on-delay** and **off-delay**.

**timedelay.N.on-delay** float in (default: 0.5)

The time, in seconds, for which **in** must be **true** before **out** becomes **true**

**timedelay.N.off-delay** float in (default: 0.5)

The time, in seconds, for which **in** must be **false** before **out** becomes **false**

**timedelay.N.elapsed** float out

Current value of the internal timer

**AUTHOR**

Jeff Epler, based on works by Stephen Wille Padnos and John Kasunich

**LICENSE**

GPL

**NAME**

timedelta – LinuxCNC HAL component that measures thread scheduling timing behavior

**SYNOPSIS**

**loadrt** **timedelta** [**count**=*N*]**names**=*name1*[,*name2*...]

**FUNCTIONS**

**timedelta.N**

**PINS**

**timedelta.N.out** s32 out

**timedelta.N.err** s32 out (default: 0)

**timedelta.N.min** s32 out (default: 0)

**timedelta.N.max** s32 out (default: 0)

**timedelta.N.jitter** s32 out (default: 0)

**timedelta.N.avg-err** float out (default: 0)

**timedelta.N.reset** bit in

**LICENSE**

GPL

**NAME**

toggle – ‘push-on, push-off’ from momentary pushbuttons

**SYNOPSIS**

**loadrt toggle [count=*N*|names=*name1*[,*name2*...]]**

**FUNCTIONS**

**toggle.*N***

**PINS**

**toggle.*N.in*** bit in  
button input

**toggle.*N.out*** bit io  
on/off output

**PARAMETERS**

**toggle.*N.debounce*** u32 rw (default: 2)  
debounce delay in periods

**LICENSE**

GPL

**NAME**

toggle2nist – toggle button to nist logic

**SYNOPSIS**

**loadrt toggle2nist** [**count**=*N* | **names**=*name1* [, *name2* ... ]]

**DESCRIPTION**

toggle2nist can be used with a momentary push button connected to a toggle component to control a device that has separate on and off inputs and has an is-on output. If in changes states via the toggle output

If is-on is true then on is false and off is true.

If is-on is false the on true and off is false.

**FUNCTIONS**

**toggle2nist.N** (requires a floating-point thread)

**PINS**

**toggle2nist.N.in** bit in

**toggle2nist.N.is-on** bit in

**toggle2nist.N.on** bit out

**toggle2nist.N.off** bit out

**LICENSE**

GPL

**NAME**

tristate\_bit – Place a signal on an I/O pin only when enabled, similar to a tristate buffer in electronics

**SYNOPSIS**

**loadrt** tristate\_bit [count=*N*|names=*name1*[,*name2*...]]

**FUNCTIONS**

**tristate-bit.***N*

If **enable** is TRUE, copy **in** to **out**.

**PINS**

**tristate-bit.***N.in* bit in

Input value

**tristate-bit.***N.out* bit io

Output value

**tristate-bit.***N.enable* bit in

When TRUE, copy in to out

**LICENSE**

GPL

**NAME**

`tristate_float` – Place a signal on an I/O pin only when enabled, similar to a tristate buffer in electronics

**SYNOPSIS**

`loadrt tristate_float [count=N]names=name1 [,name2...]`

**FUNCTIONS**

**tristate-float.*N*** (requires a floating-point thread)

If **enable** is TRUE, copy **in** to **out**.

**PINS**

**tristate-float.*N*.in** float in

Input value

**tristate-float.*N*.out** float io

Output value

**tristate-float.*N*.enable** bit in

When TRUE, copy in to out

**LICENSE**

GPL

**NAME**

updown – Counts up or down, with optional limits and wraparound behavior

**SYNOPSIS**

**loadrt updown [count=*N* | names=*name1* [, *name2* ... ]]**

**FUNCTIONS**

**updown.*N***

Process inputs and update count if necessary

**PINS**

**updown.*N*.countup** bit in

Increment count when this pin goes from 0 to 1

**updown.*N*.countdown** bit in

Decrement count when this pin goes from 1 to 0

**updown.*N*.reset** bit in

Reset count when this pin goes from 0 to 1

**updown.*N*.count** s32 out

The current count

**PARAMETERS**

**updown.*N*.clamp** bit rw

If TRUE, then clamp the output to the min and max parameters.

**updown.*N*.wrap** bit rw

If TRUE, then wrap around when the count goes above or below the min and max parameters.  
Note that wrap implies (and overrides) clamp.

**updown.*N*.max** s32 rw (default: *0x7FFFFFFF*)

If clamp or wrap is set, count will never exceed this number

**updown.*N*.min** s32 rw

If clamp or wrap is set, count will never be less than this number

**LICENSE**

GPL

**NAME**

watchdog – monitor multiple inputs for a "heartbeat"

**SYNOPSIS**

**loadrt watchdog num\_inputs=*N***

You must specify the number of inputs, from 1 to 32. Each input has a separate timeout value.

**FUNCTIONS****process**

Check all input pins for transitions, clear the **ok-out** pin if any input has no transition within its timeout period. This function does not use floating point, and should be added to a fast thread.

**set-timeouts**

Check for timeout changes, and convert the float timeout inputs to int values that can be used in **process**. This function also monitors **enable-in** for false to true transitions, and re-enables monitoring when such a transition is detected. This function does use floating point, and it is appropriate to add it to the servo thread.

**PINS****watchdog.input-n** bit in

Input number *n*. The inputs are numbered from 0 to **num\_inputs**-1.

**watchdog.enable-in** bit in (default: *FALSE*)

If TRUE, forces out-ok to be false. Additionally, if a timeout occurs on any input, this pin must be set FALSE and TRUE again to re-start the monitoring of input pins.

**watchdog.ok-out** bit out (default: *FALSE*)

OK output. This pin is true only if enable-in is TRUE and no timeout has been detected. This output can be connected to the enable input of a **charge\_pump** or **stepgen** (in v mode), to provide a heartbeat signal to external monitoring hardware.

**PARAMETERS****watchdog.timeout-n** float in

Timeout value for input number *n*. The inputs are numbered from 0 to **num\_inputs**-1. The timeout is in seconds, and may not be below zero. Note that a timeout of 0.0 will likely prevent **ok-out** from ever becoming true. Also note that excessively long timeouts are relatively useless for monitoring purposes.

**LICENSE**

GPL

**NAME**

wcomp – Window comparator

**SYNOPSIS**

**loadrt wcomp [count=*N*|names=*name1*[,*name2*...]]**

**FUNCTIONS**

**wcomp.*N*** (requires a floating-point thread)

**PINS**

**wcomp.*N.in*** float in

Value being compared

**wcomp.*N.min*** float in

Low boundary for comparison

**wcomp.*N.max*** float in

High boundary for comparison

**wcomp.*N.out*** bit out

True if **in** is strictly between **min** and **max**

**wcomp.*N.under*** bit out

True if **in** is less than or equal to **min**

**wcomp.*N.over*** bit out

True if **in** is greater than or equal to **max**

**NOTES**

If **max** <= **min** then the behavior is undefined.

**LICENSE**

GPL

**NAME**

`weighted_sum` – convert a group of bits to an integer

**SYNOPSIS**

**loadrt weighted\_sum wsum\_sizes=*size[,size,...]***

Creates weighted sum groups each with the given number of input bits (*size*).

**DESCRIPTION**

This component is a "weighted summer": Its output is the offset plus the sum of the weight of each TRUE input bit. The default value for each weight is  $2^n$  where  $n$  is the bit number. This results in a binary to unsigned conversion.

There is a limit of 8 weighted summers and each may have up to 16 input bits.

**FUNCTIONS**

**process\_wsums (requires a floating point thread)**

Read all input values and update all output values.

**PINS**

**wsum.*N*.bit.*M*.in** bit in

The  $m$ 'th input of weighted summer  $n$ .

**wsum.*N*.hold** bit in

When TRUE, the *sum* output does not change. When FALSE, the *sum* output tracks the *bit* inputs according to the weights and offset.

**wsum.*N*.sum** signed out

The output of the weighted summer

**wsum.*N*.bit.*M*.weight** signed rw

The weight of the  $m$ 'th input of weighted summer  $n$ . The default value is  $2^m$ .

**wsum.*N*.offset** signed rw

The offset is added to the weights corresponding to all TRUE inputs to give the final sum.

**NAME**

wj200\_vfd – Hitachi wj200 modbus driver

**SYNOPSIS**

**wj200\_vfd**

**PINS**

**wj200-vfd.N.commanded-frequency** float in

Frequency of vfd

**wj200-vfd.N.reverse** bit in

1 when reverse 0 when forward

**wj200-vfd.N.run** bit in

run the vfd

**wj200-vfd.N.enable** bit in

1 to enable the vfd. 0 will remote trip the vfd, thereby disabling it.

**wj200-vfd.N.is-running** bit out

1 when running

**wj200-vfd.N.is-at-speed** bit out

1 when running at assigned frequency

**wj200-vfd.N.is-ready** bit out

1 when vfd is ready to run

**wj200-vfd.N.is-alarm** bit out

1 when vfd alarm is set

**wj200-vfd.N.watchdog-out** bit out

Alternates between 1 and 0 after every update cycle. Feed into a watchdog component to ensure vfd driver is communicating with the vfd properly.

**PARAMETERS**

**wj200-vfd.N.mbslaveaddr** u32 rw

Modbus slave address

**LICENSE**

GPLv2 or greater

**NAME**

xhc\_hb04\_util – xhc-hb04 convenience utility

**SYNOPSIS**

**loadrt xhc\_hb04\_util [count=*N*]names=*name1*[,*name2*...]**

**DESCRIPTION**

Provides logic for a start/pause button and an interface to `halui.program.is_paused`, `is_idle`, `is_running` to generate outputs for `halui.program.pause`, `resume`, `run`.

Includes 4 simple lowpass filters with `coef` and `scale` pins. The `coef` value should be  $0 \leq \text{coef} \leq 1$ , smaller `coef` values slow response.

Includes 4 multiplexers to manage ini hal pins `ini.N.max_acceleration`. The `amux.N.in0` inputs are used for auto, mdi modes; the `amux.N.in1` inputs are selected when the `is_manual` pin is True and the `jogenable-off` pin is False. These pins are typically connected to `halui.mode.is-manual` and `xhc-hb04.jog.enable-off` respectively.

**FUNCTIONS**

**xhc-hb04-util.*N*** (requires a floating-point thread)

**PINS**

**xhc-hb04-util.*N*.start-or-pause** bit in  
**xhc-hb04-util.*N*.is-paused** bit in  
**xhc-hb04-util.*N*.is-idle** bit in  
**xhc-hb04-util.*N*.is-running** bit in  
**xhc-hb04-util.*N*.pause** bit out  
**xhc-hb04-util.*N*.resume** bit out  
**xhc-hb04-util.*N*.run** bit out  
**xhc-hb04-util.*N*.in0** s32 in  
**xhc-hb04-util.*N*.in1** s32 in  
**xhc-hb04-util.*N*.in2** s32 in  
**xhc-hb04-util.*N*.in3** s32 in  
**xhc-hb04-util.*N*.out0** s32 out  
**xhc-hb04-util.*N*.out1** s32 out  
**xhc-hb04-util.*N*.out2** s32 out  
**xhc-hb04-util.*N*.out3** s32 out  
**xhc-hb04-util.*N*.scale0** float in (default: *1.0*)  
**xhc-hb04-util.*N*.scale1** float in (default: *1.0*)  
**xhc-hb04-util.*N*.scale2** float in (default: *1.0*)  
**xhc-hb04-util.*N*.scale3** float in (default: *1.0*)  
**xhc-hb04-util.*N*.coef0** float in (default: *1.0*)  
**xhc-hb04-util.*N*.coef1** float in (default: *1.0*)  
**xhc-hb04-util.*N*.coef2** float in (default: *1.0*)  
**xhc-hb04-util.*N*.coef3** float in (default: *1.0*)  
**xhc-hb04-util.*N*.divide-by-k-in** float in  
**xhc-hb04-util.*N*.divide-by-k-out** float out  
**xhc-hb04-util.*N*.k** float in (default: *1.0*)  
**xhc-hb04-util.*N*.is-manual** bit in  
**xhc-hb04-util.*N*.jogenable-off** bit in  
**xhc-hb04-util.*N*.amux0-in0** float in  
**xhc-hb04-util.*N*.amux1-in0** float in  
**xhc-hb04-util.*N*.amux2-in0** float in

**xhc-hb04-util.N.amux3-in0** float in  
**xhc-hb04-util.N.amux0-in1** float in  
**xhc-hb04-util.N.amux1-in1** float in  
**xhc-hb04-util.N.amux2-in1** float in  
**xhc-hb04-util.N.amux3-in1** float in  
**xhc-hb04-util.N.amux0-out** float out  
**xhc-hb04-util.N.amux1-out** float out  
**xhc-hb04-util.N.amux2-out** float out  
**xhc-hb04-util.N.amux3-out** float out

**LICENSE**

GPL

**NAME**

xor2 – Two-input XOR (exclusive OR) gate

**SYNOPSIS**

**loadrt xor2 [count=*N* | names=*name1* [, *name2* ...]]**

**FUNCTIONS**

**xor2.*N***

**PINS**

**xor2.*N*.in0** bit in

**xor2.*N*.in1** bit in

**xor2.*N*.out** bit out

**out** is computed from the value of **in0** and **in1** according to the following rule:

**in0=TRUE in1=FALSE**

**in0=FALSE in1=TRUE**

**out=TRUE**

Otherwise,

**out=FALSE**

**LICENSE**

GPL