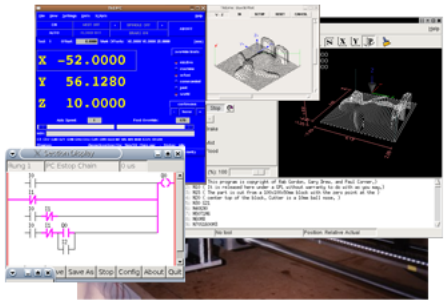
EMC²

The Enhanced Machine Controller



www.linuxcnc.org

Manuel de l'intégrateur V2.2

11 novembre 2008

The EMC Team

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

Table des matières

Première partie

Introduction

Chapter 1

The Enhanced Machine Control

1.1 Introduction

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

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

1.2 The Big CNC Picture

The term CNC has taken on a lot of different meanings over the years. In the early days CNC replaced the hands of a skilled machinist with motors that followed commands in much the same way that the machinist turned the hand wheels. From these early machines, a language of machine tool control has grown. This language is called RS274 and several standard variants of it have been put forward. It has also been expanded by machine tool and control builders in order to meet the needs of specific machines. If a machine changed tools during a program it needed to have tool change commands. If it changed pallets in order to load new castings, it had to have commands that allowed for these kinds of devices as well. Like any language, RS274 has evolved over time. Currently there are several dialects. In general each machine tool maker has been consistent within their product line but different dialects can have commands that cause quite different behavior from one machine to another.

More recently the language of CNC has been hidden behind or side-stepped by several programming schemes that are referred to as "Conversational[1] programming languages." One common feature of these kinds of programming schemes is the selection of a shape or geometry and the addition of values for the corners, limits, or features of that geometry.

The use of Computer Aided Drafting has also had an effect on the CNC programming languages. Because CAD drawings are saved as a list or database of geometries and variables associated with each, they are available to be interpreted into G-Code. These interpreters are called CAM (Computer Aided Machining) programs.

Like the CAD converters, the rise of drawing programs, like Corel™ and the whole bunch of paint programs, converters have been written that will take a bitmap or raster or vector image and turn it into G-Code that can be run with a CNC.

You're asking yourself, "Why did I want to know this?" The answer is that the EMC2 as it currently exists does not directly take in CAD or any image and run a machine using it. The EMC2 uses a

[1] One machine tool manufacturer, Hurco, claims to have a right to the use of these programming schemes and to the use of the term conversational when used in this context.

variant of the earlier CNC language named RS274NGC. (Next Generation Controller). All of the commands given to the EMC2 must be in a form that is recognized and have meaning to the RS274NGC interpreter. This means that if you want to carve parts that were drawn in some graphical or drafting program you will also have to find a converter that will transform the image or geometry list into commands that are acceptable to the EMC2 interpreter. Several commercial CAD/CAM programs are available to do this conversion. At least one converter (Ace) has been written that carries a copyright that makes it available to the public.

There has been recent talk about writing a "conversational" or geometric interface that would allow an operator to enter programs is much the same way that several modern proprietary controls enter programs but it isn't in there yet.

1.3 Computer Operating Systems

The EMC2 code can be compiled on almost any GNU-Linux Distribution (assuming it has been patched with a real time extension). In addition to the raw code, some binary distributions are available. The latest packages have been created around the Ubuntu GNU-Linux Distribution. Ubuntu is one of the distributions that is aimed at novice Linux users, and has been found to be very easy to use. Along with that, there are lots of places around the world that offer support for it. Installing EMC2 on it is trivial, see section **??**

The EMC2 will not run under a Microsoft (TM) operating system. The reason for this is that the EMC2 requires a real-time environment for the proper operation of its motion planning and stepper pulse outputs. Along with that, it also benefits from the much-needed stability and performance of the Linux OS.

1.4 History of the Software

The EMC code was started by the Intelligent Systems Division at the National Institute of Standards and Technology in the United States. The quotation below, taken from the NIST web presence some time back, should lend some understanding of the essential reasons for the existence of this software and of the NIST involvement in it.

> As part of our (NIST) collaboration with the OMAC User's Group, we have written software which implements real-time control of equipment such as machine tools, robots, and coordinate measuring machines. The goal of this software development is twofold: first, to provide complete software implementations of all OMAC modules for the purpose of validating application programming interfaces; and second, to provide a vehicle for the transfer of control technology to small- and medium-sized manufacturers via the NIST Manufacturing Extension Partnership. The EMC software is based on the NIST Real-time Control System (RCS) Methodology, and is programmed using the NIST RCS Library. The RCS Library eases the porting of controller code to a variety of Unix and Microsoft platforms, providing a neutral application programming interface (API) to operating system resources such as shared memory, semaphores, and timers. The RCS Library also implements a communication model, the Neutral Manufacturing Language, which allows control processes to read and write C++ data structures throughout a single homogeneous environment or a heterogeneous networked environment. The EMC software is written in C and C++, and has been ported to the PC Linux, Windows NT, and Sun Solaris operating systems. When running actual equipment, a real-time version of Linux is used to achieve the deterministic computation rates required (200 microseconds is typical). The software can also be run entirely in simulation, down to simulations of the machine motors. This enables entire factories of EMC machines to be set up and run in a computer integrated manufacturing environment.

EMC has been installed on many machines, both with servo motors and stepper motors. Here is a sampling of the earliest applications.

- 3-axis Bridgeport knee mill at Shaver Engineering. The machine uses DC brush servo motors and encoders for motion control, and OPTO-22 compatible I/O interfaced to the PC parallel port for digital I/O to the spindle, coolant, lube, and e-stop systems.
- 3-axis desktop milling machine used for prototype development. The machine uses DC brush servo motors and encoders. Spindle control is accomplished using the 4th motion control axis. The machine cuts wax parts.
- 4-axis Kearney & Trecker horizontal machining center at General Motors Powertrain in Pontiac, MI. This machine ran a precursor to the full-software EMC which used a hardware motion control board.

After these early tests, Jon Elson found the Shaver Engineering notes and replaced a refrigerator sized Allen Bradley 7300 control on his Bridgeport with the EMC running on a Red Hat 5.2 distribution of Linux. He was so pleased with the result that he advertised the software on several newsgroups. He continues to use that installation and has produced several boards that are supported by the software.

From these early applications news of the software spread around the world. It is now used to control many different kinds of machines. More recently the Sherline company http://www.sherline.com has released their first CNC mill. It uses a standard release of the EMC.

The source code files that make up the controller are kept in a repository on http://cvs.linuxcnc.org. They are available for anyone to inspect or download. The EMC2 source code (with a few exceptions[2]) is released under the GNU General Public License (GPL). The GPL controls the terms under which EMC2 can be changed and distributed. This is done in order to protect the rights of people like you to use, study, adapt, improve, and redistribute it freely, now and in the future. To read about your rights as a user of EMC2, and the terms under which you are allowed to distribute any modifications you may make, see the full GPL at http://www.gnu.org/copyleft/gpl.html.

1.5 How EMC2 Works

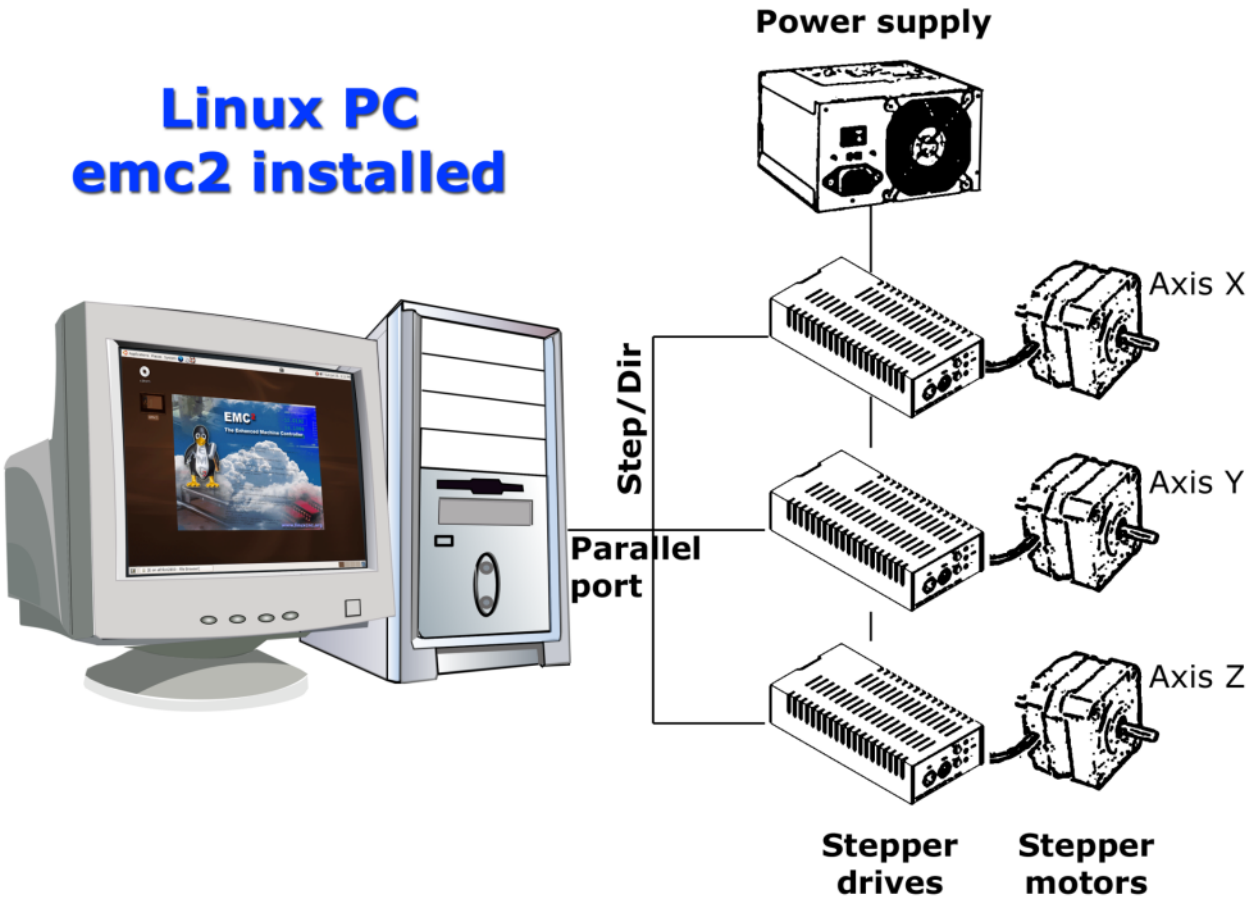
The Enhanced Machine Controller (EMC2) is a lot more than just another CNC mill program. It can control machine tools, robots, or other automated devices. It can control servo motors, stepper motors, relays, and other devices related to machine tools. In this handbook we focus on only a small part of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

Figure 1.1 shows a simple block diagram showing what a typical 3-axis EMC2 system might look like. This diagram shows a stepper motor system. The PC, running Linux as its operating system, is actually controlling the stepper motor drives by sending signals through the printer port. These signals (pulses) make the stepper drives move the stepper motors. The EMC2 can also run servo motors via servo interface cards or by using an extended parallel port to connect with external control boards. As we examine each of the components that make up an EMC2 system we will remind the reader of this typical machine.

There are four main components to the EMC2 software: a motion controller (EMCMOT), a discrete I/O controller (EMCIO), a task executor which coordinates them (EMCTASK), and a collection of text-based or graphical user interfaces. An EMC2 capable of running a mini mill must start some version of all four of these components in order to completely control it. Each component is briefly described below. In addition there is a layer called HAL (Hardware Abstraction Layer) which allows simple reconfiguration of EMC2 without the need of recompiling.

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

Figure 1.1: Simple EMC2 Controlled Machine



1.5.1 Graphical User Interfaces

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

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

Tkemc and Mini will run on Linux, Mac, and Microsoft Windows if the Tcl/Tk programming language has been installed. The Mac and Microsoft Windows version can connect to a real-time EMC2 running on a Linux machine via a network connection, allowing the monitoring of the machine from a remote location. Instructions for installing and configuring the connection between a Mac or Microsoft Machine and a PC running the EMC2 can be found in the Integrators Handbook.

1.5.2 Motion Controller EMCMOT

Motion control includes sampling the position of the axes to be controlled, computing the next point on the trajectory, interpolating between these trajectory points, and computing an output

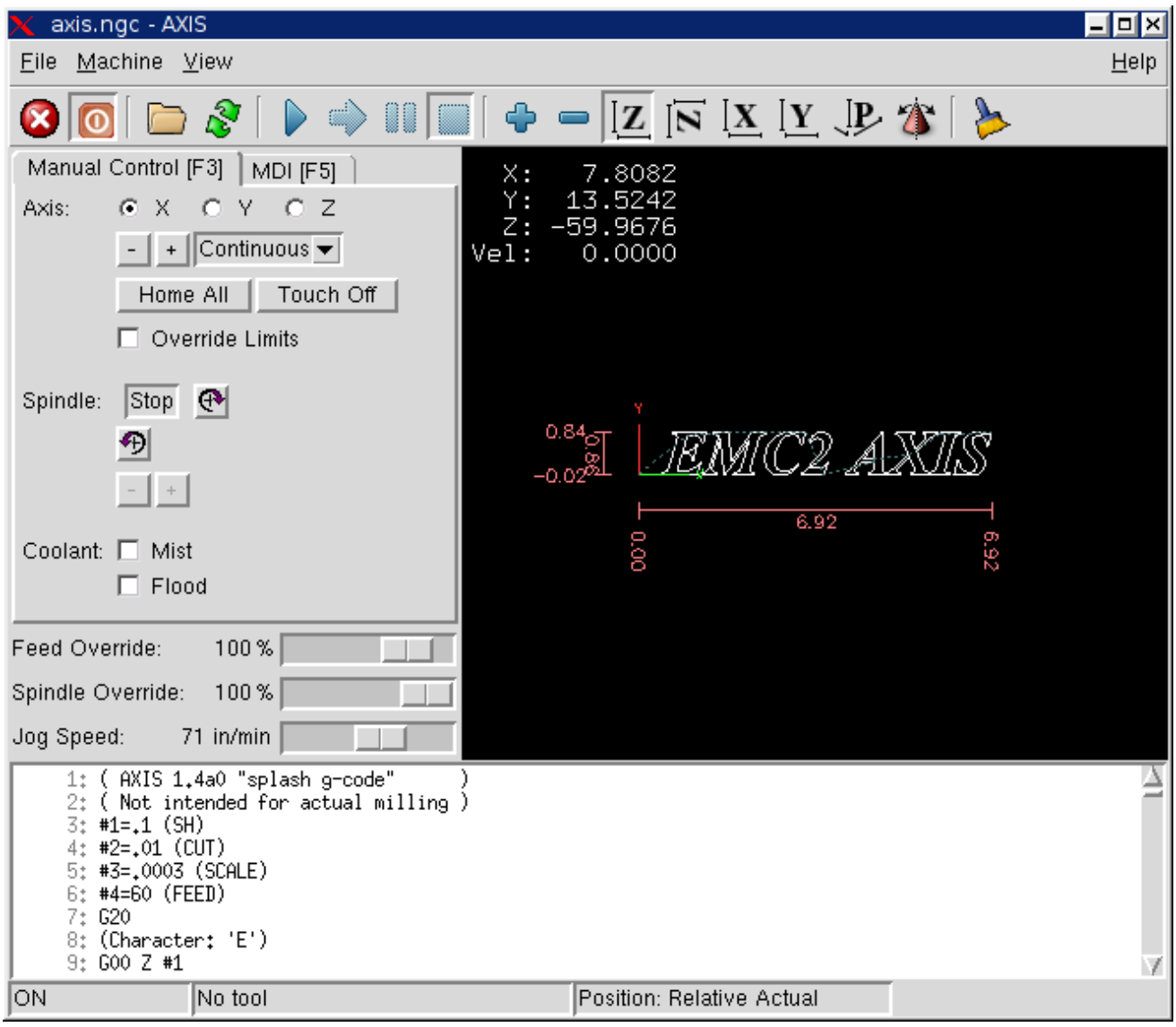


Figure 1.2: The AXIS Graphical Interface

to the motors. For servo systems, the output is based on a PID compensation algorithm. For stepper systems, the calculations run open-loop, and pulses are sent to the steppers based on whether their accumulated position is more than a pulse away from their commanded position. The motion controller includes programmable software limits, and interfaces to hardware limit and home switches.

The motion controller is written to be fairly generic. Initialization files (with the same syntax as Microsoft Windows INI files) are used to configure parameters such as number and type of axes (e.g., linear or rotary), scale factors between feedback devices (e.g., encoder counts) and axis units (e.g., millimeters), servo gains, servo and trajectory planning cycle times, and other system parameters. Complex kinematics for robots can be coded in C according to a prescribed interface to replace the default 3-axis Cartesian machine kinematics routines.

1.5.3 Discrete I/O Controller EMCIO

Discrete I/O controllers are highly machine-specific, and are not customizable in general using the INI file technique used to configure the more generic motion controller. However, since EMC2

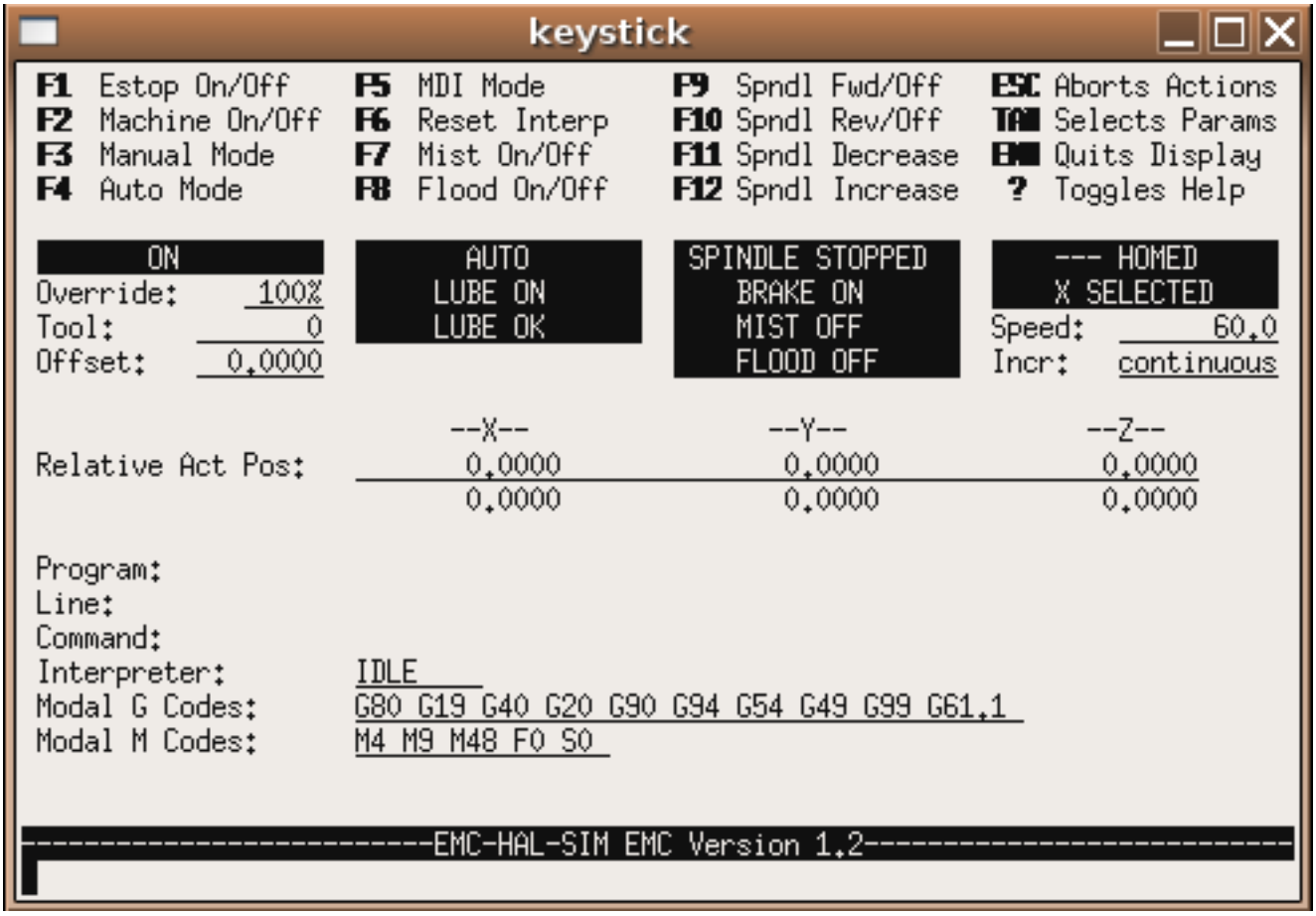


Figure 1.3: The Keystick interface

uses the HAL, reconfiguration of the I/O subsystem has become very powerful and flexible. EMC2 contains a Programmable Logic Controller module (behaves just like a hardware PLC) that can be used for very complex scenarios (tool changers, etc.).

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

1.5.4 Task Executor EMCTASK

The Task Executor is responsible for interpreting G and M code programs whose behavior does not vary appreciably between machines. G-code programming is designed to work like a machinist might work. The motion or turns of a hand wheel are coded into blocks. If a machinist wanted his mill to move an inch in the +X direction at some feed rate, he might slowly turn the hand wheel five turns clockwise in 20 seconds. The same machinist programming that same move for CNC might write the following block of code.

```
G1 F3 X1.000
```

G1 means that the machine is supposed to run at a programmed feed rate rather than at the fastest speed that it can (G0 is the way to command a rapid move like you would make above the work when not cutting). The F3 means that it should travel at 3 inches a minute or 3 millimeters a minute if it is working in metric mode. The X1.000 (assuming that the X axis started at zero) means

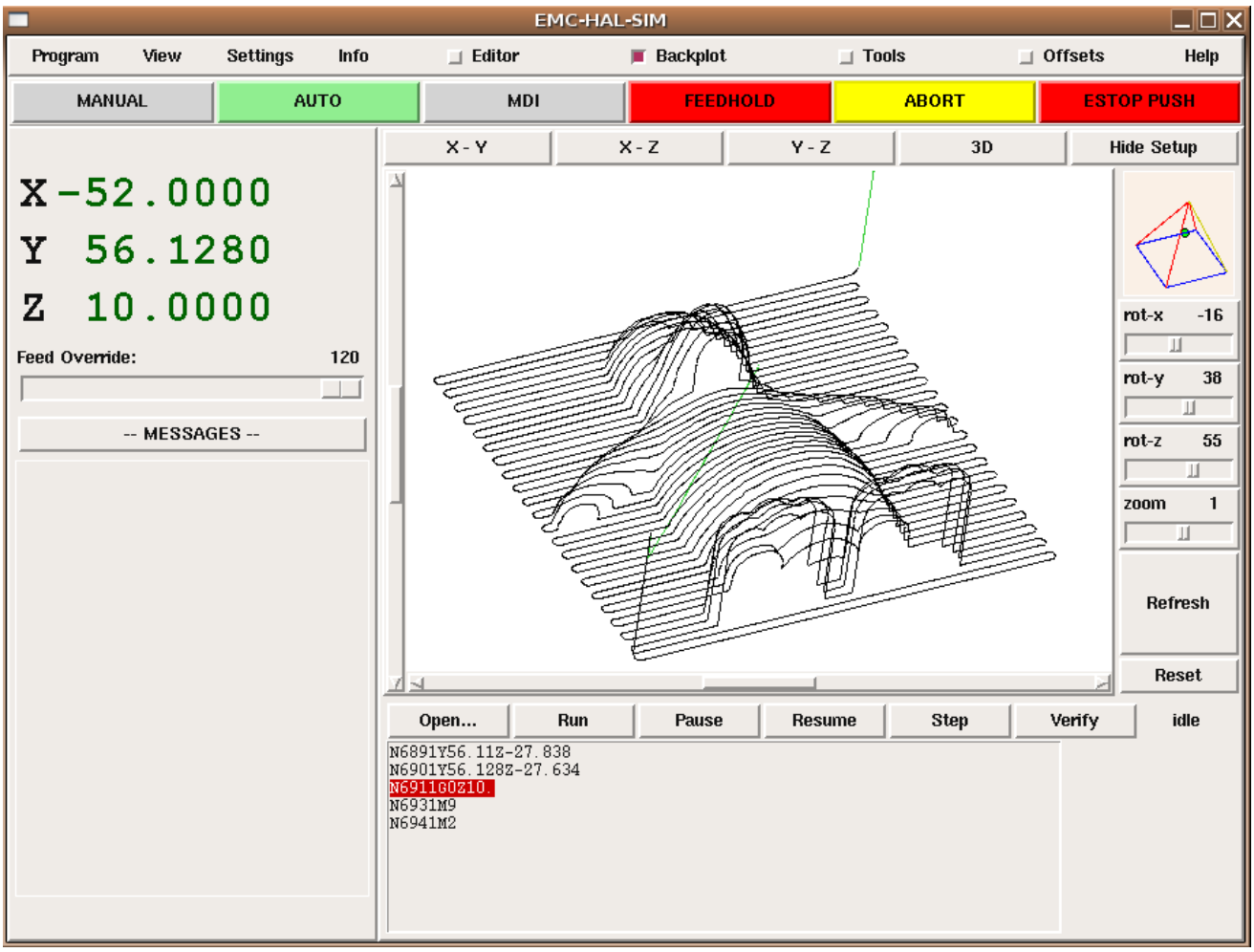


Figure 1.4: The Mini Graphical Interface

the machine should move one inch in the positive X direction. You will read quite a bit more about G-code in the programming chapters .

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

G-code is sent to the interpreter which compares the new block with what has already been sent to it. The interpreter then figures out what needs to be done for the motion and input or output systems and sends blocks of canonical commands to the task and motion planning programs.

1.5.5 Modes of Operation

When an EMC2 is running, there are three different major modes used for inputting commands. These are Manual, Auto, and MDI. Changing from one mode to another makes a big difference in the way that the EMC2 behaves. There are specific things that can be done in one mode that can not be done in another. An operator can home an axis in manual mode but not in auto or MDI modes. An operator can cause the machine to execute a whole file full of G-codes in the auto mode but not in manual or MDI.

In manual mode, each command is entered separately. In human terms a manual command might be “turn on coolant” or “jog X at 25 inches per minute.” These are roughly equivalent to flipping a

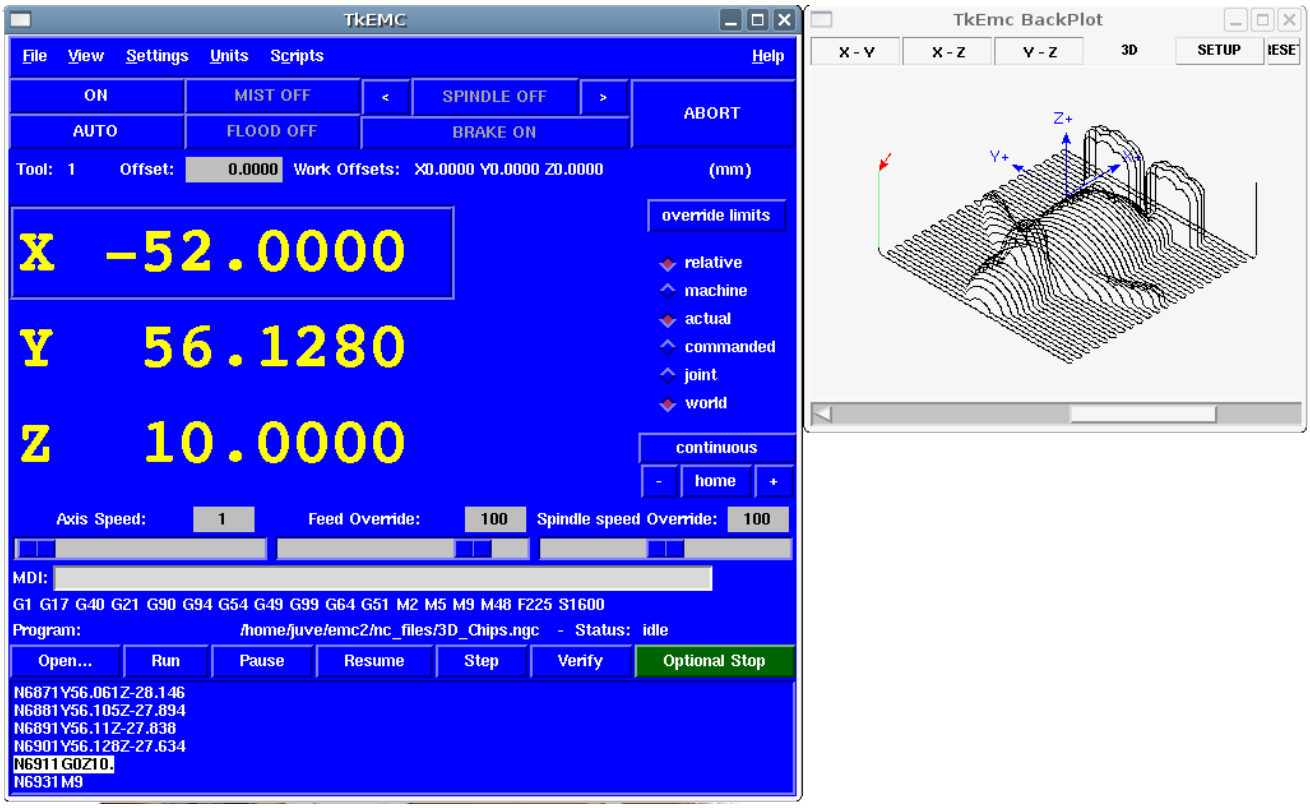


Figure 1.5: The TkEmc Graphical Interface

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

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

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

1.5.6 Information Display

While an EMC2 is running, each of the modules keeps up a conversation with the others and with the graphical display. It is up to the display to select from that stream of information what the operator needs to see, and to arrange it on the screen in a way that makes it easy for the operator to understand. Perhaps the most important display is the mode the EMC2 is running in. You will want to keep your eye on the mode display.

Right up there with knowing what mode is active is consistent display of the position of each axis. Most of the interfaces will allow the operator to read position based upon actual or commanded position as well as machine or relative position.

Machine This is the position of an axis relative to the place where it started or was homed.

Figure 1.6: The XEMC Graphical Interface

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

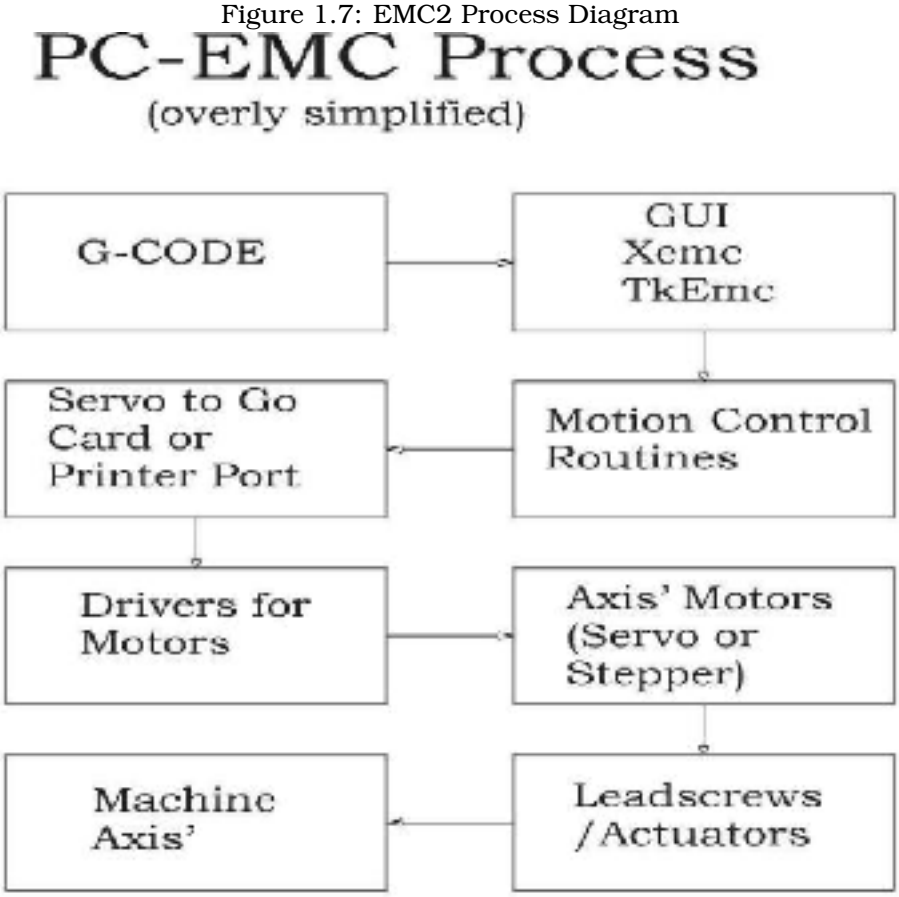
Actual This is the real position of the axis within the machine or relative system.

Commanded This is where the axis is commanded to be.

These may all be exactly the same if no offsets have been applied and there is no deadband set in the INI file. Deadband is a small distance which is assumed to be close enough – perhaps one stepper pulse or one encoder count.

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

Figure 1.7: EMC2 Process Diagram



stopped running.

As you work your way through this text, you will be learning, bit by bit, how to set up and run a machine with your copy of the EMC2 software. While you are learning about setting up and running a mini mill here, you will be thinking of other applications and other capabilities. These are the topics of the other linuxcnc.org handbooks.

1.6 Thinking Like An Integrator

The biggest task of a machine integrator is figuring out how to connect a PC running the EMC2 to a machine and configuring the software so that it runs the machine correctly.

1.6.1 Units

Units can be confusing. You might ask, "Does it work in inches, feet, centimeters, millimeters, or what?" There are several possible answers to this question but the best one is that it works in the units that you set it to work in.

At a machine level, we set each axis's units to some value using an INI variable that looks like this.

```
UNITS = inch
```

or

```
UNITS = mm
```

After we have decided upon a value for the units for an axis, we tell the EMC2 how may step pulses or encoder pulses it should send or read for each unit of distance to be traveled. Once we have done this, the EMC2 knows how to count units of distance. However it is very important to understand that this counting of distance is different from the commanding of distance. You can command distance in millimeters or inches without even thinking about the units that you defined. There are G-codes that allow you to switch easily between metric and imperial.

Part II

Installation

Chapter 2

Installer le logiciel EMC2

2.1 Introduction

Un des problèmes les plus souvent évoqués par les utilisateurs à propos d'EMC a été qu'il ne s'installait pas de lui-même. Il fallait qu'ils récupèrent les sources, les compilent eux-mêmes, essaient d'appliquer un noyau Linux patché RT, etc. Les développeurs d'EMC2 ont donc choisi une distribution standard appelée Ubuntu[1].

Ubuntu a été choisi parce-qu'il est parfaitement dans l'esprit Open Source d'EMC2:

- Ubuntu restera toujours gratuit, il n'y a pas de frais pour l'édition "enterprise edition", nous faisons de notre mieux pour rendre notre travail disponible à tous dans les mêmes termes de gratuité.
- Ubuntu fournit un support professionnel commercial à des centaines de sociétés dans le monde, vous aurez peut être besoin de ces services. Chaque nouvelle version d'Ubuntu reçoit des mises à jour de sécurité gratuites pendant 18 mois après sa publication, certaines versions sont supportées plus longtemps.
- Ubuntu utilise le meilleur en termes de traduction et d'accessibilité à ses infrastructures parmi ce que la communauté du logiciel libre peut offrir et pour faire qu'Ubuntu soit aprécié par autant d'utilisateurs que possible.
- Ubuntu est publié régulièrement selon un calendrier précis; Une nouvelle version est publiée tous les six mois. Vous pouvez utiliser la dernière version stable ou aider à stabiliser la version en cours de développement.
- La communauté Ubuntu est entièrement dévouée aux principes de développement du logiciel libre; elle encourage tout le monde à utiliser des logiciels libres et open source, les améliorer et les distribuer.

2.2 La page de téléchargement d'EMC

Vous pouvez trouver l'annonce des plus récentes versions publiées d'EMC2 sur le site www.linuxcnc.org. Les versions d'EMC2 sont fournies de deux manières (sources et paquets binaires). Les sources (décrites dans le manuel du développeur) consistent en un tarball (emc2-<version>.tar.gz), que vous devez charger et décompacter dans votre répertoire home.

[1] Le mot "Ubuntu" est un ancien mot Africain, signifiant "humanité aux autres". Ubuntu signifie aussi "Je suis ce que je suis à cause de ce que nous sommes tous". La distribution Ubuntu Linux amène l'esprit d'Ubuntu au monde du logiciel. Vous pouvez en lire plus à ce propos ici: http://www.ubuntu-fr.org/

Le présent document (plus orienté utilisateur final) expliquera seulement comment installer les paquets binaires sur une distribution Ubuntu[2].

2.3 Le live CD d'EMC2

Les développeurs d'EMC2 ont créé un Live-CD basé sur Ubuntu 6.06 qui vous permet d'essayer EMC2 avant de l'installer, c'est également une manière facile d'installer ensemble Ubuntu et EMC2.

Téléchargez l'image image ISO http://linuxcnc.org/iso/emc2-ubuntu6.06-desktop-i386.iso (Miroir EU http://dsplabs.utt.ro/~juve/emc/) et gravez la sur un CD. (la somme MD5 du CD est vérifiable)

Quand vous bootez avec ce CD dans le lecteur de votre machine, vous pouvez voir et expérimenter un environnement identique à celui d'EMC2 qui sera le vôtre si vous choisissez de l'installer.

Si cette démonstation vous a convaicu, cliquez sur l'icône Install du bureau, répondez à quelques questions (votre nom, fuseau horaire, mot de passe) et l'installation terminera en quelques minutes.

Cette installation vous apportera tout les avantages du support de la communauté Ubuntu avec la configuration automatique d'EMC2. Quand une mise à jour d'EMC2 sera publiée, le gestionnaire de paquets vous le fera savoir et vous permettra une mise à jour aisée.

2.4 Script d'installation d'EMC2

Il est également possible d'utiliser un simple script d'installation d'emc2 sur Ubuntu pour les utilisateurs ayant déjà une installation existante d'Ubuntu. Il lance la commande expliquée dans 2.5.

Pour l'utiliser vous devez:

- Charger le script depuis http://linuxcnc.org/dapper/emc2-install.sh (Pour Ubuntu 6.06)
- Le sauver sur votre bureau. Faire un clic droit sur son icône, sélectionner Propriétés. Choisir l'onglet Permissions et cocher Propriétaire: Exécuter. Fermer la fenêtre des propriétés.
- Maintenant double-cliquez sur l'icône emc2-install.sh, et choisissez "Run in Terminal". Une console va apparaître et vous demander votre mot de passe.
- Quand l'installation vous demande si vous voulez installer les paquets d'EMC2, pressez Entrée pour accepter. Laissez ensuite l'installation se poursuivre jusqu'à la fin.
- Quand elle est terminée, éjectez le CD puis vous devrez redémarrer votre machine (Système > Quitter > Redémarrer l'ordinateur). Quand vous aurez redémarré vous pourrez alors lancer EMC2 via le menu Applications > CNC.
- Si vous n'êtes pas prêt pour configurer votre machine, essayez la configuration sim-AXIS; elle démarre en mode "machine simulée" qui ne requiert le raccordement d'aucun matériel.
- Maintenant que l'installation est terminée, Ubuntu vous avertira quand des mises à jour d'EMC2 seront disponibles. Quand ça arrivera, vous pourrez mettre à jour facilement et automatiquement avec le gestionnaire de mises à jour.

[2]Pour plus d'informations sur les autres variantes Linux, lisez le Manuel du développeur ou demandez de l'aide sur la Liste de diffusion http://sourceforge.net/mail/?group_id=6744.

2.5 Installation manuelle par apt-get.

Cette petite section décrira comment installer EMC2 sur Ubuntu 6.06 “Dapper Dreake” en utilisant les commandes apt dans une console. Si vous connaissez un peu Linux et Debian, ça va être facile. Sinon, vous devriez peut être lire 2.4.

Premièrement, ajoutez le dépôt à /etc/apt/sources.list:

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

Puis faites les mises à jour et l'installation d'emc2 avec:

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

Ces commandes vont installer correctement les paquets emc2 avec toutes leurs dépendances[3].

Vous pourriez avoir une alarme indiquant que les paquets proviennent d'une source non vérifiée (ce qui voudrait dire que votre ordinateur ne reconnaît pas la signature GPG des paquets). Pour corriger cette situation, appliquez les commandes suivantes:

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

[3]Les dépendances sont un des atouts majeurs des distributions basées sur Debian. Elles assurent que vous avez la totalité de ce qui doit être installé. Même dans un cas comme emc2 qui nécessite un noyau de Linux patché pour travailler en temps réel, ainsi que toutes les librairies indispensables.

Chapter 3

Compiler EMC2 depuis les sources

3.1 Introduction

Quelques difficultés sont à surmonter quand vous commencez à installer EMC2, son téléchargement et l'installation du software proprement dit. L'ensemble des fichiers d'EMC2 sont placés dans le dépôt cvs.linuxcnc.org, c'est un dépôt avec gestion des versions (CVS). EMC2 est également disponible en paquets pré-compilés (pour différentes plateformes) pour téléchargement depuis ce site.

L'installation peut être une tâche compliquée pour quelqu'un de nouveau sous Linux. La partie la plus dure étant d'appliquer le patch temps réel (Real Time Linux) au noyau. Après ça, installer EMC2 est assez facile. Celà dit, il est dorénavant possible aux utilisateurs de profiter d'une possibilité totalement nouvelle, il leur suffit d'installer Ubuntu (une distribution Linux vraiment conviviale), puis d'exécuter un simple script d'installation, et ils auront alors un EMC2 directement en état de marche sur un noyau temps réel. Les informations pour accéder à cette solution sont diponibles sur www.linuxcnc.org à la page Download.

3.2 Page de téléchargement EMC

Vous pouvez trouver l'annonce des versions les plus récentes d'EMC2 sur www.linuxcnc.org. Les versions d'EMC2 sont fournies de deux manières, sources et paquets binaires. Les sources (described furtheron) sont sous forme de fichiers tarball (emc2-version.tar.gz), que vous devez télécharger et décompacter dans votre répertoire home.

3.3 Gestion des versions d'EMC2

EMC2 utilise un modèle de versions similaire (bien que simplifié) à celui utilisé par Debian. Il y a tout le temps trois versions d'EMC2. Debian utilise "stable", "testing" et "unstable". Nous utilisons "Released", "Testing" et "Head". Pour les dernières informations, cliquez sur la version qui vous intéresse.

Released est exactement ça, une version publiée d'EMC2 avec un numéro de version. Elle a été testée par beaucoup de développeurs et de béta testeurs avant d'être publiée, elle est utilisable par la moyenne des utilisateurs. Les développeurs et réguliers des IRC/mailing list sont prêts à aider ceux qui démarrent avec une version "released". **"Released"** est disponible sous plusieurs formes, incluant .debs pour Ubuntu et tarballs de sources pour une compilation locale. Il y a un dépôt Debian qui a toujours la dernière version "released" (elle permet donc de faciliter les mises à jour d'une version stable).

Testing est une version d'EMC2 qui est prête pour le "beta testing" mais pas pour une publication générale. Avant qu'une version soit labellisée **testing** elle doit d'abord être compilée et doit démarrer sur différentes plateformes, mais il y aura probablement des limitations et divers problèmes. La page **Testing** du wiki est prévue pour lister les problèmes connus et leurs solutions, mais il reste probablement aussi des bugs non découverts. Puisque la version **Testing** est un software "beta", il ne doit pas être utilisé pour tout ce qui est critique. Les utilisateurs de la version **Testing** doivent comprendre qu'il s'agit d'un software en beta et qu'ils doivent être disposés à donner des rapports de bugs détaillés si quelque chose ne va pas. **Testing** est disponible principalement comme une balise en CVS, toutefois pour la commodité des testeurs, un dépôt "testing" debian et/ou des tarballs peuvent aussi être disponibles. C'est le conseil d'administration d'EMC qui décide quand une version "Testing" est digne de devenir "Released". C'est une décision formelle, présentée par voix de motion aux votes du conseil d'administration ou votes par la mailing liste de l'IRC.

TRUNK est un terme CVS pour indiquer l'emplacement des versions en début de développement. Une version **TRUNK** peut souvent être non fonctionnelle. Lorsque la version **TRUNK** sera réputée digne par de nombreux testeurs soit un grand nombre de personnes, la balise **"Testing"** lui sera appliquée. C'est une décision informelle, prise par consensus à la tête des développeurs, habituellement sur l'IRC. Le développement continue immédiatement et un autre **TRUNK** diverge de cette nouvelle version **Testing**. **TRUNK** n'a pas de numéro de version, au cours d'un veek-end chargé il peut changer littéralement toutes les 10 minutes.

3.4 Téléchargement et compilation des sources.

Les quelques sections suivantes décriront comment se procurer les sources d'EMC2 et les compiler.

Pour les télécharger, allez simplement sur www.linuxcnc.org à la page “Download” et prenez les tarballs de la dernière version “release” ou “testing”.

Quand vous les avez dans votre répertoire home, il faut les extraire, ouvrez une console et faites:

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

Puis vous devez décider quel type d'installation vous voulez. Il y a deux possibilités pour essayer EMC2:

Installed Comme la plupart des autres logiciels sous Linux, les fichiers sont placés dans des répertoires système, ils sont automatiquement disponibles à tous les utilisateurs de l'ordinateur.[1]

Run-in-place Tous les dossiers sont conservés à l'intérieur du répertoire EMC2. Cette option est utile pour essayer EMC2, surtout quand il existe déjà une autre version d'EMC2 installée sur le système..

3.4.1 Télécharger une version CVS

Si vous souhaitez utiliser la version TRUNK d'EMC2, veuillez suivre les instructions de notre wiki pour obtenir le code source:: http://wiki.linuxcnc.org/cgi-bin/emcinfo.pl?CVS

3.5 Installed

EMC2 suit la manière standard de la compilation de logiciel sous linux. Pour compiler il suffit de se rendre dans le répertoire des sources:

[1] Le paquet pré-installé pour Ubuntu Linux utilise la méthode “installé”

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

et d'y lancer ces commandes:

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

Pour le lancer, tapez 'emc'.

3.6 Run-in-place

Si vous voulez seulement tester le logiciel avant de l'installer, ou si vous avez peur d'écraser une version déjà existante, vous pouvez essayer le mode Run-In-Place (RIP). Dans ce mode, il n'y a aucune installation et aucun fichier ne sera placé en dehors du répertoire `~/emc2`.

Faites juste:

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

puis tapez ces commandes:

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

Dans une console, où vous voulez utiliser EMC2, tapez:[2]

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

Jusqu'à ce que vous fermiez la console, il sera mis en place afin que les programmes et les pages de manuel soient disponibles sans avoir à se référer au chemin à chaque fois. Ensuite vous pouvez lancer EMC2 en faisant:

```
$ emc
```

3.7 Simulateur

Pour installer EMC2 sur un système sans noyau temps réel, ajoutez `--enable-simulator` à la ligne de commande `configure`. Dans ce mode, seule la partie purement programme d'EMC2 démarrera. Aucun matériel n'aura à être contrôlé, les timmings ne sont pas garantis, mais les autres fonctionnalités de HAL, EMC2 et ses diverses interfaces sont disponibles. Pour utiliser ce mode ajoutez `--enable-run-in-place` à la commande configure, l'étape du `sudo make setuid` n'est pas nécessaire.

[2] En tapant cette commande dans dans le script de démarrage de la console, comme `~/.bash_profile`, vous n'aurez plus à la taper manuellement dans la fenêtre de chaque console.

3.8 Editer et recompiler

Vous pouvez avoir besoin de recompiler le code d'EMC2 pour diverses raisons. Vous pouvez avoir à modifier le code source, ou vous pouvez avoir seulement téléchargé quelques nouveaux fichiers. Pour recompiler, tapez les commandes suivantes:

```
$ cd ~/emc2/src
$ make && sudo make install # pour le run-installed
$ make && sudo make setuid  # pour le run-in-place
$ make                      # pour le run-in-place en simulateur
```

Le processus de compilation est suffisamment performant pour ne recompiler que ce qui est affecté par vos changements.

Part III

EMC: Configuration

Chapter 4

Configuration, fichier ini

4.1 Fichiers utilisés pour la configuration

EMC est entièrement configuré avec des fichiers textes classiques. Tous ces fichiers peuvent être lus et modifiés dans n'importe quel éditeur de texte disponible dans toute distribution Linux[1] . Soyez prudent lorsque vous modifierez ces fichiers, certaines erreurs pourraient empêcher le démarrage d'EMC. Ces fichiers sont lus à chaque fois que le logiciel démarre. Certains d'entre eux sont lus de nombreuses fois pendant l'exécution d'CNC.

Les fichiers de configuration inclus:

INI Le fichier ini écrase les valeurs par défaut compilées dans le code d'EMC. Il contient également des sections qui sont lues directement par le HAL (Hardware Abstraction Layer, couche d'abstraction matérielle).

HAL Les fichiers hal installent les modules de process, ils créent les liens entre les signaux d'EMC et les broches spécifiques du matériel.

VAR Ce fichier contient une suite de numéros de variables. Ces variables contiennent les paramètres qui seront utilisés par l'interpréteur. Ces valeurs sont enregistrées d'une exécution à l'autre.

TBL Ce fichier contient les informations relatives aux outils.

NML Ce fichier configure les canaux de communication utilisés par EMC. Il est normalement réglé pour lancer toutes les communications avec un seul ordinateur, peut être modifié pour communiquer entre plusieurs ordinateurs.

.emcrc Ce fichier enregistre des informations spécifiques à l'utilisateur, il a été créé pour enregistrer le nom du répertoire lorsque l'utilisateur choisit sa première configuration d'EMC.[2]

Les éléments avec le repère **(HAL)** sont utilisés seulement pour les fichiers de HAL en exemples. C'est une bonne convention. D'autres éléments sont utilisés directement par EMC et doivent toujours avoir la section et le nom donné à l'item.

[1] Ne confondez pas un éditeur de texte et un traitement de texte. Un éditeur de texte comme gedit ou kwrite produisent des fichiers uniquement en texte. Les lignes de textes sont séparées les unes des autres. Un traitement de texte comme Open Office produit des fichiers avec des paragraphes, des mises en formes des mots. Ils ajoutent des codes de contrôles, des polices de formes et de tailles variées etc. Un éditeur de texte n'a rien de tout cela.

[2] Habituellement, ce fichier est dans le répertoire home de l'utilisateur (ex: /home/user/)

4.2 Organisation du fichier INI

Un fichier INI typique suit une organisation simple;

- commentaires.
- sections,
- variables.

Chacun de ces éléments est séparé, sur une seule ligne. Chaque fin de ligne ou retour chariot crée un nouvel élément.

4.2.1 Commentaires

Une ligne de commentaires débute avec un ; ou un #. Si le logiciel qui analyse le fichier ini rencontre l'un ou l'autre de ces caractères, le reste de la ligne est ignorée. Les commentaires peuvent être utilisés pour décrire ce que font les éléments du fichier INI.

```
; Ceci est le fichier de configuration de ma petite fraiseuse.
; Je l'ai ajusté le 12 janvier 2006
```

Des commentaires peuvent également être utilisés pour choisir entre plusieurs valeurs d'une seule variable.

```
# DISPLAY = tkemc
DISPLAY = axis
# DISPLAY = mini
# DISPLAY = keystick
```

Dans cette liste, la variable DISPLAY est positionnée sur axis puisque toutes les autres sont commentées. Si quelqu'un édite une liste comme celle-ci et par erreur, décommente deux lignes, c'est la première rencontrée qui sera utilisée.

Notez que dans une ligne de variables, les caractères "**#**" et " **;** " n'indiquent pas un commentaire.

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

4.2.2 Sections

Les différentes parties d'un fichier .ini sont regroupées dans des sections. Une section commence par son nom en majuscules entre crochets [UNE_SECTION]. L'ordre des sections est sans importance. Les sections suivantes sont utilisées par emc:

- [EMC] informations générales (4.3.1)
- [DISPLAY] sélection du type d'interface graphique (4.3.2)
- [RS274NGC] ajustements utilisés par l'interpréteur de g-code
- [EMCMOT] Réglages utilisés par le contrôleur de mouvements temps réel (4.3.3)
- [HAL] spécifications des fichiers .hal (4.3.5)
- [TASK] Réglages utilisés par le contrôleur de tâche (4.3.4)
- [TRAJ] Réglages additionnels utilisés par le contrôleur de mouvements temps réel (4.3.6)
- [AXIS_0] ... [AXIS_n] Groupes de variables pour AXIS (4.3.7)
- [EMCIO] Réglages utilisés par le contrôleur d'entrées/sorties (4.3.8)

4.2.3 Variables

Une ligne de variables est composée d'un nom de variable, du signe égal (=) et d'une valeur. Tout, du premier caractère non blanc qui suit le signe = jusqu'à la fin de la ligne, est passé comme valeur à la variable. Vous pouvez donc intercaler des espaces entre les symboles si besoin. Un nom de variable est souvent appelé un mot clé.

Les paragraphes suivants détaillent chaque section du fichier de configuration, en utilisant des exemples de variables dans les lignes de configuration.

Certaines de ces variables sont utilisées par EMC. Elles doivent toujours utiliser le nom de section et le nom de variable dans leur appellation. D'autres variables ne sont utilisées que par HAL. Les noms des sections et les noms des variables indiquées sont celles qui sont utilisées dans les exemples de fichiers de configuration.

4.3 Définition des variables du fichier INI

4.3.1 Section [EMC]

VERSION = $Revision: 1.3 $ Le numéro de version du fichier INI. La valeur indiquée ici semble étrange, car elle est automatiquement mise à jour lors de l'utilisation du système de contrôle de révision. C'est une bonne idée de changer ce numéro à chaque fois que vous modifiez votre fichier. Si vous voulez le modifier manuellement, il suffit de changer le numéro sans toucher au reste.

MACHINE = ma machine C'est le nom du contrôleur, qui est imprimé dans le haut de la plupart des fenêtres. Vous pouvez insérer ce que vous voulez ici tant que ça reste sur une seule ligne.

RS274NGC_STARTUP_CODE = G21 G90 Une chaine de codes NC qui sera utilisée pour initialiser l'interpréteur. Elle ne se substitue pas à la spécification des gcodes modaux du début de chaque fichier ngc. Les codes modaux des machines diffèrent, ils pourraient être modifiés par les gcodes interprétés plus tôt dans la session.

4.3.2 Section [DISPLAY]

Les différentes interfaces du programme utilisent différentes options. Toutes les options ne sont pas supportées par toutes les interfaces.

DISPLAY = tkemc Le nom de l'interface utilisateur à utiliser. Les options disponibles sont les suivantes:

- axis
- keystick
- mini
- tkemc
- xemc

POSITION_OFFSET = RELATIVE Le système de coordonnées (RELATIVE ou MACHINE) à utiliser au démarrage de l'interface utilisateur. Le système de coordonnées RELATIVE reflète le G92 et le décalage d'origine G5x actuellement actifs.

POSITION_FEEDBACK = ACTUAL Valeur de la position (COMMANDED ou ACTUAL) à afficher au démarrage de l'interface utilisateur. La position COMMANDED est la position exacte requise par emc. La position ACTUAL est la position retournée par l'électronique des moteurs.

MAX_FEED_OVERRIDE = 1.2 La correction de vitesse maximum que l'opérateur peut utiliser. 1.2 signifie 120% de la vitesse programmée.

MIN_SPINDLE_OVERRIDE = 0.5 Correction de vitesse minimum de broche que l'opérateur pourra utiliser. 0.5 signifie 50% de la vitesse de broche programmée. (utile si il est dangereux de démarrer un programme avec une vitesse de broche trop basse).

MAX_SPINDLE_OVERRIDE = 1.0 Correction de vitesse maximum de broche que l'opérateur pourra utiliser. 1.0 signifie 100% de la vitesse de broche programmée.

DEFAULT_LINEAR_VELOCITY = .25 Vitesse minimum par défaut pour les jogs linéaires, en unités machine par seconde. Seulement utilisé dans l'interface AXIS.

MAX_LINEAR_VELOCITY = 1.0 Vitesse maximum par défaut pour les jogs linéaires, en unités machine par seconde. Seulement utilisé dans l'interface AXIS.

DEFAULT_ANGULAR_VELOCITY = .25 Vitesse minimum par défaut pour les jogs angulaires, en unités machine par seconde. Seulement utilisé dans l'interface AXIS.

MAX_ANGULAR_VELOCITY = 1.0 Vitesse maximum par défaut pour les jogs angulaires, en unités machine par seconde. Seulement utilisé dans l'interface AXIS.

PROGRAM_PREFIX = ~/emc2/nc_files Répertoire par défaut des fichiers de g-codes et emplacement des M-codes définis par l'utilisateur.

INCREMENTS = 1 mm, .5 mm, . . . Définit les incréments disponibles pour le jog incremental. Voir la section 4.3.2 pour plus d'informations. Seulement utilisé dans l'interface AXIS.

INTRO_GRAPHIC = emc2.gif L'image affichée sur l'écran d'accueil.

INTRO_TIME = 5 Durée d'affichage de l'écran d'accueil.

OPEN_FILE = /full/path/to/file.ngc Le fichier NC à utiliser au démarrage d'AXIS.

4.3.3 Section [EMCMOT]

BASE_PERIOD = 50000 (HAL) "Période de base" des tâches, exprimée en nanosecondes. C'est la plus rapide des horloges de la machine.
Avec un système à servomoteurs, il n'y a généralement pas de raison pour que **BASE_PERIOD** soit plus petite que **SERVO_PERIOD**.
Sur une machine de type "step&direction" avec génération logicielle des impulsions de pas, c'est **BASE_PERIOD** qui détermine le nombre maximum de pas par seconde. Si de longues impulsions de pas ou de longs espaces entre les impulsions ne sont pas requis par l'électronique, la fréquence maximum absolue est de un pas par **BASE_PERIOD**. Ainsi, la **BASE_PERIOD** utilisée ici donnera une fréquence de pas maximum absolue de 20000 pas par seconde. 50000ns est une valeur assez large. La plus petite valeur utilisable est liée au résultat du test de latence (**??**), à la longueur des impulsions de pas nécessaire et à la vitesse du µP.
Choisir une BASE_PERIOD trop basse peut amener à des messages "Unexpected realtime delay", des bloquages ou des reboots spontanés.

SERVO_PERIOD = 1000000 (HAL) Période de la tâche "Servo", exprimée également en nanosecondes. Cette valeur sera arrondie à un multiple entier de **BASE_PERIOD**. Elle est utilisée aussi sur des systèmes basés sur des moteurs pas à pas
C'est la vitesse avec laquelle la nouvelle position des moteurs est traitée, les erreurs de suivi vérifiées, les valeurs des sorties PID sont rafraichies etc.
Sur la plupart des systèmes **cette** valeur n'est pas à modifier. Il s'agit du taux de mise à jour du planificateur de mouvement de bas niveau.

TRAJ_PERIOD = 1000000 (HAL) Période du planificateur de trajectoire, exprimée en nanosecondes. Cette valeur sera arrondie à un multiple entier de **SERVO_PERIOD**.
Excepté pour les machines avec une cinématique particulière (ex: hexapodes) Il n'y a aucune raison de rendre cette valeur supérieure à **SERVO_PERIOD**.

4.3.4 Section [TASK]

CYCLE_TIME = 0.001 Période exprimée en secondes, à laquelle EMCTASK va tourner. Ce paramètre affecte l'intervalle de polling lors de l'attente de la fin d'un mouvement, lors de l'exécution d'une pause d'instruction et quand une commande provenant d'une interface utilisateur est acceptée. Il n'est généralement pas nécessaire de modifier cette valeur.

4.3.5 Section [HAL]

HALFILE = example.hal Exécute le fichier 'example.hal' au démarrage. Si **HALFILE** est spécifié plusieurs fois, les fichiers sont exécutés dans l'ordre de leur apparition dans le fichier ini. Presque toutes les configurations auront au moins un **HALFILE**. Les systèmes à moteurs pas à pas ont généralement deux de ces fichiers, un qui spécifie la configuration générale des moteurs (`core_stepper.hal`) et un qui spécifie le brochage des sorties (`xxx_pinout.hal`)

HAL = command Exécute 'command' comme étant une simple commande hal. Si **HAL** est spécifié plusieurs fois, les commandes sont exécutées dans l'ordre où elles apparaissent dans le fichier ini. Les lignes **HAL** sont exécutées après toutes les lignes **HALFILE**.

SHUTDOWN = shutdown.hal Exécute le fichier 'shutdown.hal' quand emc s'arrête. Selon les pilotes de matériel utilisés, il est ainsi possible de positionner les sorties sur des valeurs définies quand emc s'arrête normalement. Cependant, parce qu'il n'y a aucune garantie que ce fichier sera exécuté (par exemple, dans le cas d'une panne de l'ordinateur), il ne remplace pas une véritable chaîne physique d'arrêt d'urgence ou d'autres logiciels de protection des défauts de fonctionnement.

POSTGUI_HALFILE = example2.hal *(Seulement avec l'interface AXIS)* Exécute 'example2.hal' après que l'interface graphique ait créé ses HAL pins.

4.3.6 Section [TRAJ]

La section [TRAJ] contient les paramètres généraux du module planificateur de trajectoires d' EMCMOT. Vous n'aurez pas à modifier ces valeurs si vous utilisez EMC avec une machine à trois axes en provenance des USA. Si vous êtes dans une zone métrique, utilisant des éléments matériels métriques, vous pourrez utiliser le fichier stepper_mm.ini dans lequel les valeurs sont déjà configurées dans cette unité.

COORDINATES = X Y Z Les noms des axes à contrôler. X, Y, Z, A, B, C, U, V, et W sont valides. Seuls les axes nommés dans **COORDINATES** seront acceptés dans le g-code. Cela n'a aucun effet sur l'ordonnancement des noms d'axes depuis le G-code (X- Y- Z-) jusqu'aux numéros d'articulations. Pour une "cinématique triviale", X est toujours l'articulation 0, A est toujours l'articulation 4, U est toujours l'articulation 7 et ainsi de suite. Il est permis d'écrire les noms d'axe par paire (ex: X Y Y Z pour une machine à portique) mais cela n'a aucun effet.

AXES = 3 Une unité de plus que le plus grand numéro d'articulation du système. Pour une machine XYZ, les articulations sont numérotées 0, 1 et 2. Dans ce cas, les AXES sont 3. Pour un système XYUV utilisant une "cinématique triviale", l'articulation V est numérotée 7 et donc les AXES devraient être 8. Pour une machine à cinématique non triviale (ex: scarakins) ce sera généralement le nombre d'articulations contrôlées.

HOME = 0 0 0 Coordonnées de l'origine machine de chaque axe. De nouveau, pour une machine 4 axes, vous devrez avoir 0 0 0 0. Cette valeur est utilisée uniquement pour les machines à cinématique non triviale. Sur les machines avec cinématique triviale, cette valeur est ignorée.

LINEAR_UNITS=<units> Le nom des unités utilisées dans le fichier INI. Les choix possibles sont 'in', 'inch', 'imperial', 'metric', 'mm'.
Cela n'affecte pas les unités linéaires du code NC (pour cela il y a les mots G20 et G21).

ANGULAR_UNITS=<units> Le nom des unités utilisées dans le fichier INI. Les choix possibles sont 'deg', 'degree' (360 pour un cercle), 'rad', 'radian' (2pi pour un cercle), 'grad', ou 'gon' (400 pour un cercle).
Cela n'affecte pas les unités angulaires du code NC. Dans le code RS274NGC, les mots A-, B- et C- sont toujours exprimés en degrés.

DEFAULT_VELOCITY = 0.0167 La vitesse initiale de jog des axes linéaires, en unités par seconde. La valeur indiquée ici correspond à une unité par minute.

DEFAULT_ACCELERATION = 2.0 Dans les machines à cinématique non triviale, l'accélération utilisée pour "teleop" jog (espace cartésien), en unités machine par seconde par seconde.

MAX_VELOCITY = 5.0 Vitesse maximale de déplacement pour les axes, exprimée en unités machine par seconde. La valeur indiquée est égale à 300 unités par minute.

MAX_ACCELERATION = 20.0 Accélération maximale pour les axes, exprimée en unités machine par seconde par seconde.

POSITION_FILE = position.txt Si réglée à une valeur non vide, les positions des articulations sont enregistrées dans ce fichier. Cela permet donc de redémarrer avec les mêmes coordonnées que lors de l'arrêt[3] . Si vide, les positions ne seront pas enregistrées et commenceront à 0 à chaque fois qu'EMC démarrera.

4.3.7 Section [AXIS_<num>]

Les sections [AXIS_0], [AXIS_1], etc. contiennent les paramètres généraux des composants individuels du module de contrôle. La numérotation des sections axis commencent à 0 et vont jusqu'au nombre d'axes spécifié dans la variable [TRAJ] AXES, moins 1.

TYPE = LINEAR Type des axes, soit LINEAR, soit ANGULAR.

UNITS = inch Ce réglage écrase celui des variables [TRAJ] UNITS si il est spécifié. (ex: [TRAJ]LINEAR_UNITS si le TYPE de cet axe est LINEAR, [TRAJ]ANGULAR_UNITS si le TYPE de cet axe est ANGULAR)

MAX_VELOCITY = 1.2 Vitesse maximum pour cet axe en unités machine par seconde.

MAX_ACCELERATION = 20.0 Accélération maximum pour cet axe en unités machine par seconde au carré.

BACKLASH = 0.000 Valeur de compensation du jeu en unités machine. Peut être utilisée pour atténuer de petites déficiences du matériel utilisé pour piloter cet axe.

COMP_FILE = file.extension Fichier dans lequel est enregistrée une structure de compensation spécifique à cet axe. Les valeurs internes sont des triplets représentant les positions suivantes:

1. Positions nominales
2. Positions en marche positive
3. Positions en marche négative.

La position nominale est celle où devrait être le mobile. La position en marche positive signifie, où se trouve le mobile pendant le déplacement dans le sens positif. La position en marche négative signifie, où se trouve le mobile pendant le déplacement dans le sens négatif. Un triplet par ligne. Actuellement la limite d'EMC2 est de 256 triplets par axe. Si COMP_FILE est spécifié, BACKLASH est ignoré. Les valeurs sont en unités machine.

[3]Cela suppose, que hors puissance, la machine ne fera aucun mouvement pendant tout son arrêt. C'est utile pour les petites machines sans contact d'origine machine.

COMP_FILE_TYPE = 1 En spécifiant une valeur non nulle, le format des triplets du fichier COMP_FILE sera différent. Pour COMP_FILE_TYPE = 0, les valeurs des triplets seront: position nominale, position en marche positive, position en marche négative. Pour COMP_FILE_TYPE différent de 0, les valeurs dans COMP_FILE seront: position nominale, écart sens positif, écart sens négatif. Comparées au valeurs définies au dessus elles correspondent à, nominale, nominale-position en marche positive, nominal-position en marche négative.

Exemple de triplet avec COMP_FILE_TYPE = 0: 1.00 1.01 0.99.

Le même exemple de triplet avec COMP_FILE_TYPE = 1: 1.00 -0.01 0.01

MIN_LIMIT = -1000 Limite minimum des mouvements de cet axe (limite soft), en unités machine. Quand cette limite tend à être dépassée, le contrôleur arrête le mouvement.

MAX_LIMIT = 1000 Limite maximum des mouvements de cet axe (limite soft), en unités machine. Quand cette limite tend à être dépassée, le contrôleur arrête le mouvement.

MIN_FERROR = 0.010 Valeur indiquant, en unités machine, de combien le mobile peut dévier à très petite vitesse de la position commandée. Si MIN_FERROR est plus petit que FERROR, les deux produisent une rampe de points de dérive. Vous pouvez imaginer un graphe sur lequel une dimension représente la vitesse et l'autre, l'erreur tolérée. Quand la vitesse augmente, la quantité d'erreurs de suivi augmente également et tend vers la valeur FERROR.

FERROR = 1.0 FERROR est le maximum d'erreurs de suivi tolérable, en unités machine. Si la différence entre la position commandée et la position retournée excède cette valeur, le contrôleur désactive les calculs des servomoteurs, positionne toutes les sorties à 0.0 et coupe les amplis des moteurs. Si MIN_FERROR est présent dans le fichier .ini, une vitesse proportionnelle aux erreurs de suivi est utilisée. Ici, le maximum d'erreur de suivi est proportionnel à la vitesse, quand FERROR est appliqué à la vitesse rapide définie dans [TRAJ]MAX_VELOCITY et proportionnel aux erreurs de suivi pour les petites vitesses. L'erreur maximale admissible sera toujours supérieure à MIN_FERROR. Cela permet d'éviter que de petites erreurs de suivi sur les axes stationnaires arrêtent les mouvements de manière impromptue. Des petites erreurs de suivi seront toujours présentes à cause des vibrations, etc. La polarité des valeurs de suivi détermine comment les entrées sont interprétées et comment les résultats sont appliqués aux sorties. Elles peuvent généralement être réglées par tâtonnement car il n'y a que deux possibilités. Le programme utilitaire USRMOT peut être utilisé pour les ajuster intéractivement et vérifier les résultats, de sorte que les valeurs puissent être mises dans le fichier INI avec un minimum de difficultés.

4.3.7.1 Variables relatives aux prises d'origine

Les cinq prochains paramètres sont relatifs aux prises d'origine.

HOME_OFFSET = 0.0 Position du contact d'origine machine de l'axe ou impulsion d'index, en unités machine.

HOME_SEARCH_VEL = 0.0 Vitesse du mouvement inital de prise d'origine, en unités machine par seconde. Une valeur de zéro suppose que la position courante est l'origine machine. Si votre machine n'a pas de contact d'origine, laissez cette valeur à zéro.

HOME_LATCH_VEL = 0.0 Vitesse du mouvement final de prise d'origine pour le dégagement du contact d'origine, en unités machine par seconde.

HOME_USE_INDEX = NO Si l'encodeur utilisé pour cet axe fournit une impulsion d'index et qu'elle est gérée par la carte contrôleur, vous pouvez mettre sur Yes. Quand il est sur yes, il aura une incidence sur le type de séquence de prise d'origine utilisé.

HOME_IGNORE_LIMITS = NO Certaines machines utilisent une limite d'axe comme contact d'origine machine. Cette variable devra être positionnée sur yes si c'est le cas de votre machine.

4.3.7.2 Variables relatives aux servo

Les entrées suivantes concernent les systèmes à servomoteurs, comme la carte du système univstep de Pico Systems.[4] Cette description suppose que les unités en sortie du composant PID sont des Volts.

P = 50 (HAL) La composante **p**roportionnelle du gain de l'ampli moteur de cet axe. Cette valeur multiplie l'erreur entre la position commandée et la position actuelle en unités machine, elle entre dans le calcul de la tension appliquée à l'ampli moteur. Les unités du gain **P** sont des Volts sur des unités machine, exemple: $\frac{volt}{mm}$ si l'unité machine est le millimètre.

I = 0 (HAL) La composante **i**ntégrale du gain de l'ampli moteur de cet axe. Cette valeur multiplie l'erreur cumulative entre la position commandée et la position actuelle en unités machine, elle entre dans le calcul de la tension appliquée à l'ampli moteur. Les unités du gain **I** sont des Volts sur des unités machine par seconde, exemple: $\frac{volt}{mm\,s}$ si l'unité machine est le millimètre.

D = 0 (HAL) La composante **d**érivée du gain de l'ampli moteur de cet axe. Cette valeur multiplie la différence entre l'erreur courante et les précédentes, elle entre dans le calcul de la tension appliquée à l'ampli moteur. Les unités du gain **D** sont des Volts sur des unités machine sur des secondes, exemple: $\frac{volt}{mm/s}$ si l'unité machine est le millimètre.

FF0 = 0 (HAL) Gain à priori (feedforward) d'orde 0. Cette valeur est multipliée par la position commandée, elle entre dans le calcul de la tension appliquée à l'ampli moteur. Les unités du gain FF0 sont des Volts sur des unités machine, exemple: $\frac{volt}{mm}$ si l'unité machine est le millimètre.

FF1 = 0 (HAL) Gain à priori (feedforward) de premier orde. Cette valeur est multipliée par l'écart de la position commandée par seconde, elle entre dans le calcul de la tension appliquée à l'ampli moteur. Les unités du gain FF1 sont des Volts sur des unités machine par seconde, exemple: $\frac{volt}{mm\,s}$ si l'unité machine est le millimètre.

FF2 = 0 (HAL) Gain à priori (feedforward) de second orde. Cette valeur est multipliée par l'écart de la position commandée par seconde au carré, elle entre dans le calcul de la tension appliquée à l'ampli moteur. Les unités du gain FF2 sont des Volts sur des unités machine par des secondes au carré, exemple: $\frac{volt}{mm\,s^2}$ si l'unité machine est le millimètre.

OUTPUT_SCALE = 1.000

OUTPUT_OFFSET = 0.000 (HAL) Ces deux valeurs sont, l'échelle et le facteur d'offset de l'ampli moteur de cet axe. La seconde valeur (offset) est soustraite de la valeur de sortie calculée (en Volts) puis divisée par la première valeur (facteur d'échelle), avant d'être écrite dans le convertisseur D/A. Les unités du facteur d'échelle sont des Volts réels par Volts en sortie de DAC. Les unités de la valeur d'offset sont en Volts. Ces valeurs peuvent être utilisées pour linéariser un DAC.
Plus précisément, quand les sorties sont écrites, EMC converti d'abord les unités quasi-SI des sorties concernées en valeurs brutes, exemple: Volts pour un amplificateur DAC. Cette mise à l'échelle ressemble à cela:

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

La valeur d'échelle peut être obtenue par analyse des unités, exemple: les unités sont [unités SI en sortie]/[unités de l'actuateur]. Par exemple, sur une machine sur laquelle une tension de consigne de l'ampli de 1 Volt donne une vitesse de 250 mm/sec :

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

Notez que les unités d'offset sont en unités machine, exemple: mm/sec et qu'elles sont déjà soustraites depuis la sonde de lecture. La valeur de cet offset est obtenue en prenant la valeur

[4] Référez vous au "Manuel de l'intégrateur d'EMC2" pour des informations complémentaires sur les systèmes à servomoteurs et leur contrôle en PID.

de votre sortie qui donne 0,0 sur la sortie de l'actuateur. Si le DAC est linéarisé, cet offset est normallement de 0.0.
L'échelle et l'offset peuvent être utilisés pour linéariser les DAC, d'où des valeurs qui reflètent les effets combinés du gain de l'ampli, de la non linéarité du DAC, des unités du DAC, etc. Pour ce faire, suivez cette procédure:

1. Construire un tableau de calibrage pour la sortie, piloter le DAC avec la tension souhaitée et mesurer le résultat. Voir le tableau 4.3.7.2 pour un exemple de mesures de tension.
2. Effectuer un "least squares" linéaire pour obtenir les coefficients a, b tels que:

$$meas = a * raw + b$$

3. Notez que nous voulons des sorties brutes de sorte que nos résultats mesurés soient identiques à la sortie commandée. Ce qui signifie:
   (a)
   $$cmd = a * raw + b$$
   (b)
   $$raw = (cmd - b)/a$$
4. En conséquence, les coefficients a et b d'ajustement linéaire peuvent être directement utilisés comme valeurs d'échelle et d'offset pour le contrôleur.

MAX_OUTPUT = 10 (HAL) La valeur maximale pour la sortie de la compensation PID pouvant être envoyée sur l'ampli moteur, en Volts. La valeur calculée de la sortie sera fixée à cette valeur limite. La limite est appliquée avant la mis à l'échelle de la sortie en unités brutes.

MIN_OUTPUT = -10 (HAL) La valeur minimale pour la sortie de la compensation PID pouvant être envoyée sur l'ampli moteur, en Volts. La valeur calculée de la sortie sera fixée à cette valeur limite. La limite est appliquée avant la mis à l'échelle de la sortie en unités brutes.

Mesure des tensions de sortie

Raw (brutes)	Mesurées
-10	-9.93
-9	-8.83
0	-0.03
1	0.96
9	9.87
10	10.87

INPUT_SCALE = 40000 (HAL) Spécifie le nombre d'impulsions qui correspond à un mouvement d'une unité machine. Un second chiffre, si spécifié, sera ignoré.
La valeur de l'échelle peut être obtenue par l'analyse des unités, exemple: les unités sont:

$$\frac{sensor\ units}{desired\ input\ SI\ units}$$

Par exemple, sur un codeur de 2000 top par tour, un réducteur de 10 tours/pouce et des unités demandées en mm, nous avons:

$$\begin{aligned} input\_scale &= 2000\frac{counts}{rev} * 10\frac{rev}{inch} * \frac{1\,inch}{25.4\,mm} \\ &= 787.40157\frac{counts}{mm} \end{aligned}$$

4.3.7.3 Variables relatives aux moteurs pas à pas

SCALE = 40000 (HAL) Spécifie le nombre d'impulsions qui correspond à un mouvement d'une unité machine. Pour les sytèmes à moteurs pas à pas, c'est le nombre d'impulsions de pas nécessaires pour avancer d'une unité machine. Pour les systèmes à servo, c'est le nombre d'impulsions de retour signifiant que le mobile a avancé d'une unité machine. Un second chiffre, si spécifié, sera ignoré.
La valeur de l'échelle peut être obtenue par l'analyse des unités, exemple: les unités sont:

$$\frac{step\,units}{desired\,input\,SI\,units}$$

Par exemple, un pas moteur de 1.8 degré, en mode demipas, avec une réduction de 10 tours/pouce et des unités souhaitées en mm, nous avons:

$$\begin{aligned} input\_scale &= \frac{2\,count}{1.8\,degree} * 10\frac{rev}{inch} * \frac{1\,inch}{25.4\,mm} \\ &= 157.48031\frac{counts}{mm} \end{aligned}$$

D'anciens fichiers de configuration .ini et .hal utilisaient INPUT_SCALE pour cette valeur.

STEPGEN_MAXACCEL = 21.0 (HAL) Limite d'accélération pour le générateur de pas. Elle doit être 1% à 10% supérieure à celle de l'axe MAX_ACCELERATION. Cette valeur améliore les réglages de la "boucle de position" de stepgen.

STEPGEN_MAXVEL = 1.4 (HAL) Les anciens fichiers de configuration avaient également une limite de vitesse du générateur de pas. Si spécifiée, elle doit auss être 1% à 10% supérieure à celle de l'axe MAX_VELOCITY. Des test ultérieurs ont montré que l'utilisation de STEPGEN_MAXVEL n'améliore pas le réglage de la boucle de position de stepgen.

4.3.8 Section [EMCIO]

CYCLE_TIME = 0.100 La période en secondes, à laquelle EMCIO va tourner. La mettre à 0.0 ou à une valeur négative fera qu'EMCIO tournera en permanence. Il est préférable de ne pas modifier cette valeur.

TOOL_TABLE = tool.tbl Ce fichier contient les informations des outils.

TOOL_CHANGE_POSITION = 0 0 2 Spécifies la position XYZ ou le mobile se déplacera lors d'un appel d'outil.

4.4 Prise d'origine

4.4.1 Vue d'ensemble

La prise d'origine semble assez simple, il suffit de déplacer chaque axe à un emplacement connu et de positionner l'ensemble des variables internes d'EMC en conséquence. Toutefois, les machines sont différentes les unes des autres et la prise d'origine est maintenant devenue assez complexe.

4.4.2 Séquence de prise d'origine

La figure 4.4.2 montre les quatre séquences de prise d'origine possibles, avec les variables de configuration associées. Une description détaillée de ces paramètres sera faite au chapitre suivant.

4.4.3 Configuration

Il y a six éléments d'information qui définissent le déroulement de la séquence de prise d'origine. Ils sont définis dans la section [AXIS] du fichier ini.

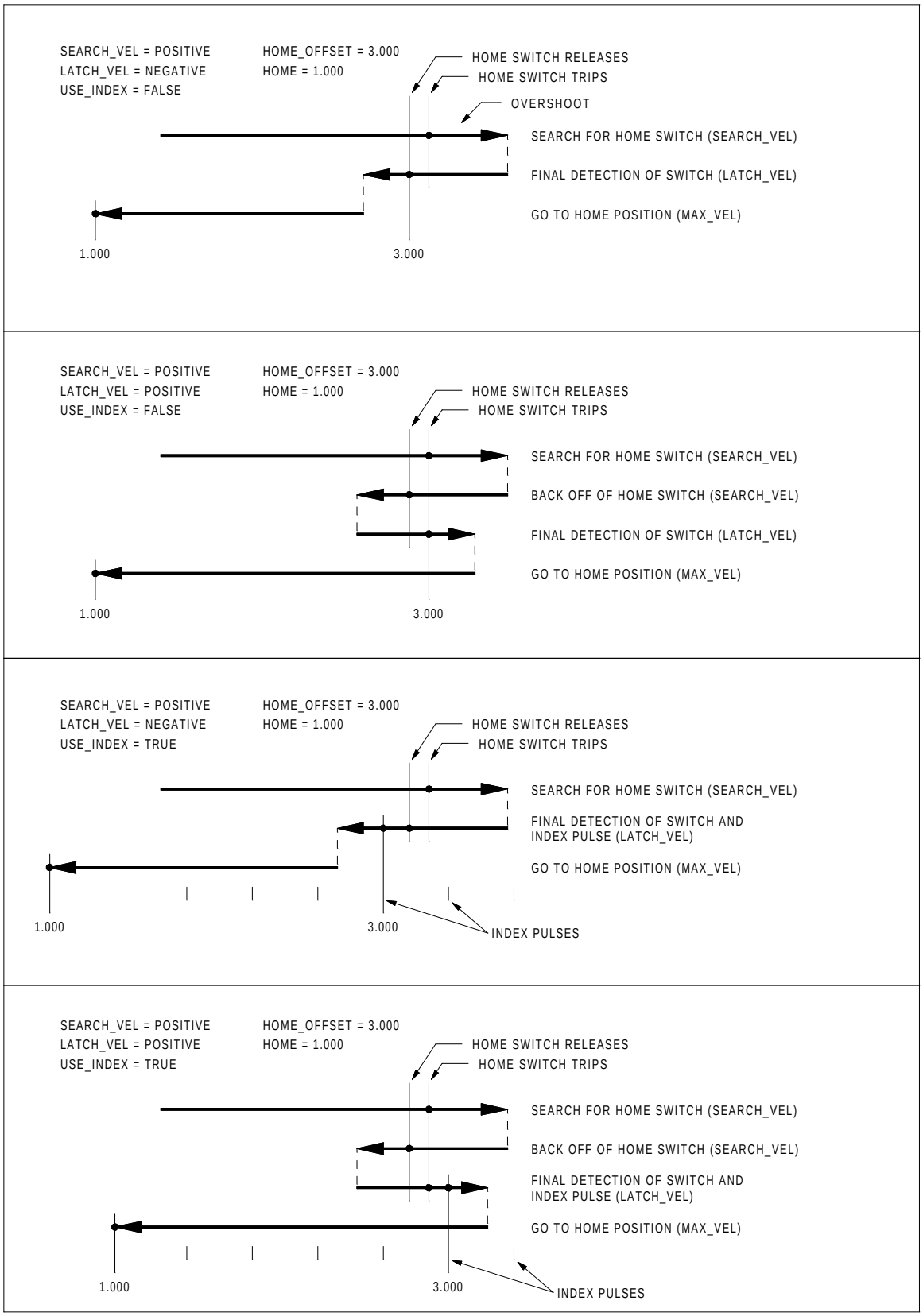
4.4.3.1 HOME_SEARCH_VEL = 0

Vitesse de la phase initiale de prise d'origine, c'est la recherche du contact d'origine machine. Une valeur différente de zéro indique à EMC la présence d'un contact d'origine machine. EMC va alors commencer par vérifier si ce contact est déjà pressé. Si oui, il le dégagera à la vitesse établie par "HOME_SEARCH_VEL", la direction du dégagement sera de signe opposé à celui de "HOME_SEARCH_VEL". Puis, il va revenir vers le contact en se déplaçant dans la direction spécifiée par le signe de "HOME_SEARCH_VEL" et à la vitesse déterminée par sa valeur absolue. Quand le contact d'origine machine est détecté, le mobile s'arrête aussi vite que possible, il y aura cependant toujours un certain dépassement dépendant de la vitesse. Si celle-ci est trop élevée, le mobile peut dépasser suffisamment le contact pour aller attaquer un fin de course de limite d'axe, voir même aller se crasher dans une butée mécanique. À l'opposé, si "HOME_SEARCH_VEL" est trop basse, la prise d'origine peut durer très longtemps.

Une valeur égale à zéro indique qu'il n'y a pas de contact d'origine machine, dans ce cas, les phases de recherche de ce contact seront occultées. La valeur par défaut est zéro.

4.4.3.2 HOME_LATCH_VEL=0

Spécifie la vitesse et la direction utilisée par le mobile pendant la dernière phase de la prise d'origine, c'est la recherche précise du contact d'origine machine, si il existe et de l'emplacement de l'impulsion d'index, si elle est présente. Cette vitesse est plus lente que celle de la phase initiale, afin d'améliorer la précision. Si "HOME_SEARCH_VEL" et "HOME_LATCH_VEL" sont de mêmes signes, la phase de recherche précise s'effectuera dans le même sens que la phase de recherche initiale. Dans ce cas, le mobile dégagera d'abord le contact en sens inverse avant de revenir vers lui à la vitesse définie ici. L'acquisition de la position d'origine se fera sur la première impulsion de changement d'état du contact. Si "HOME_SEARCH_VEL" et "HOME_LATCH_VEL" sont de signes opposés, la phase de recherche précise s'effectuera dans le sens opposé à celui de la recherche initiale. Dans ce cas, EMC dégagera le contact à la vitesse définie ici. L'acquisition de la position d'origine se fera sur la première impulsion de changement d'état du contact lors de son dégagement. Si "HOME_SEARCH_VEL" est à zéro, signifiant qu'il n'y a pas de contact et que "HOME_LATCH_VEL" et différente de zéro, le mobile continuera jusqu'à la prochaine impulsion d'index. Si "HOME_SEARCH_VEL" est différent de zéro et que "HOME_LATCH_VEL" est égal à zéro, c'est une cause d'erreur, l'opération de prise d'origine échouera. La valeur par défaut est zéro.



Séquences de prise d'origine possibles

4.4.3.3 HOME_IGNORE_LIMITS = YES/NO

Peut contenir les valeurs YES ou NO. Cette variable détermine si EMC doit ignorer les fins de course de limites d'axe. Certaines machines n'utilisent pas un contact d'origine séparé, à la place, elles utilisent un des interrupteurs de fin de course comme contact d'origine. Dans ce cas, EMC doit ignorer l'activation de cette limite de course pendant la séquence de prise d'origine. La valeur par défaut de ce paramètre est NO.

4.4.3.4 HOME_USE_INDEX = YES/NO

Spécifie si une impulsion d'index doit être prise en compte (cas de règles de mesure ou de codeurs de positions). Si cette variable est vraie (HOME_USE_INDEX = YES), EMC fera l'acquisition de l'origine machine sur le premier front de l'impulsion d'index. Si elle est fausse (=NO), EMC fera l'acquisition de l'origine sur le premier front produit par le contact d'origine (dépendra des signes de "HOME_SEARCH_VEL" et "HOME_LATCH_VEL"). La valeur par défaut est NO.

4.4.3.5 HOME_OFFSET

Contient l'emplacement du point d'origine ou de l'impulsion d'index, en coordonnées relatives. Il peut aussi être traité comme le décalage entre le point d'origine machine et le zéro de l'axe. A la détection de l'impulsion d'origine, EMC ajuste les coordonnées de l'axe à la valeur de "HOME_OFFSET". La valeur par défaut est zéro.

4.4.3.6 HOME

C'est la position sur laquelle ira le mobile à la fin de la séquence de prise d'origine. Après avoir détecté le contact d'origine, avoir ajusté les coordonnées de ce point à la valeur de "HOME_OFFSET", le mobile va se déplacer sur la valeur de "HOME", c'est le point final de la séquence de prise d'origine. La valeur par défaut est zéro. Notez que même si ce paramètre est égal à la valeur de "HOME_OFFSET", le mobile dépassera très légérement la position du point d'aquisition de l'origine machine avant de s'arrêter. Donc il y aura toujours un petit mouvement à ce moment là (sauf bien sûr si "HOME_SEARCH_VEL" est à zéro, et que toute la séquence de POM a été sautée). Ce mouvement final s'effectue en vitesse de déplacement rapide. Puisque l'axe est maintenant référencé, il n'y a plus de risque pour la machine, un mouvement rapide est donc la façon la plus rapide de finir la séquence de prise d'origine.[5]

4.4.3.7 HOME_IS_SHARED

Si cet axe n'a pas un contact d'origine séparé des autres, mais plusieurs contacts câblés sur la même broche, mettez cette valeur à 1 pour éviter de commencer la prise d'origine si un de ces contacts partagés est déjà activé. Mettez cette valeur à 0 pour permettre la prise d'origine même si un contact est déjà attaqué.

4.4.3.8 HOME_SEQUENCE

Utilisé pour définir l'ordre des séquences "HOME ALL" de prise d'origine des différents axes (exemple : la POM de l'axe X ne pourra se faire qu'après celle de Z). La POM d'un axe ne pourra se faire qu'après tous les autres en ayant la valeur la plus petite de "HOME_SEQUENCE" et après qu'ils soient déjà tous à "HOME_OFFSET". Si deux axes ont la même valeur de "HOME_SEQUENCE", leur POM s'effectueront simultanément. Si "HOME_SEQUENCE" est égale à -1 ou non spécifiée, l'axe ne

[5] La distinction entre l'origine machine et le décalage d'origine n'est pas aussi claire que je le voudrais. J'envisage de faire un petit dessin et un exemple pour la clarifier.

sera pas compris dans la séquence “HOME ALL”. Les valeurs de “HOME_SEQUENCE” débutent à 0, il ne peut pas y avoir de valeur inutilisée.

Chapitre 5

EMC2 et HAL

Voir également les pages **motion(9)** et **iocontrol(1)** du manuel.

5.1 motion (realtime)

Ces pins, paramètres et fonctions sont créés par le module temps réel "`motmod`".

5.1.1 Pins

motion.adaptive-feed IN float Quand la vitesse est placée en mode adaptatif avec `M52 P1` la vitesse commandée est multipliée par cette valeur. Cet effet est multiplicatif avec **motion.feed-hold** et la valeur du correcteur de vitesse du niveau NML.

motion.digital-out-NN OUT bit Ces pins sont controlées par les mots `M62` jusqu'à `M65`.

motion.enable IN bit Si ce bit devient FALSE, les mouvements s'arrêtent, la machine est placée dans l'état "machine arrêtée" et un message est affiché pour l'opérateur. Pour un fonctionnement normal, ce bit sera TRUE.

motion.feed-hold IN bit Quand la vitesse est placée en mode arrêt contrôlé avec `M53 P1` et que ce bit est TRUE, la vitesse est fixée à 0.

motion.motion-inpos OUT bit TRUE si la machine est en position.

motion.probe-input IN bit `G38.x` utilise la valeur de cette pin pour déterminer quand la sonde de mesure entre en contact. TRUE le contact de la sonde est fermé (touche), FALSE le contact de la sonde est ouvert.

motion.spindle-brake OUT bit TRUE quand le frein de broche doit être activé.

motion.spindle-forward OUT bit TRUE quand la broche doit tourner en sens horaire.

motion.spindle-reverse OUT bit TRUE quand la broche doit tourner en sens anti-horaire.

motion.spindle-on OUT bit TRUE quand la broche doit tourner.

motion.spindle-speed-out OUT float Consigne de vitesse de rotation de la broche, exprimée en tours par minute.

motion.spindle-index-enable I/O bit Pour les mouvements avec broche synchronisée, ce signal doit être raccordé à la broche "index-enable" du codeur de broche.

motion.spindle-revs IN float Pour les mouvements avec broche synchronisée, ce signal doit être raccordé à la broche "position" du codeur de broche.

5.1.2 Paramètres

Beaucoup de ces paramètres servent d'aide au déboguage et sont sujets aux changements ou au retrait à tout moment.

motion.coord-error TRUE quand le mouvement est en erreur, ex : dépasser une limite soft.

motion.coord-mode TRUE quand le mouvement est en "mode coordonnées" par opposistion au "mode téléopération".

motion.in-position Identique à la pin : **motion.motion-inpos.**

motion.motion-enabled TRUE quand le mouvement est activé.

motion.servo.last-period Le nombre de cycle du processeur entre les invoquations du thread servo. Typiquement, ce nombre divisé par la vitesse du processeur donne un temps en secondes. Il peut être utilisé pour determiner si le contrôleur de mouvement en temps réel respecte ses contraintes de timing.

motion.servo.overruns En voyant de grandes différences entre les valeurs successives de **motion.servo.last-period**, le contrôleur de mouvement peut déterminer qu'il a eu un échec pour respecter ses contraintes de timing. Chaque fois qu'une erreur est détectée, cette valeur est incrémentée.

motion.debug-bit-0

motion.debug-bit-1

motion.debug-float-0

motion.debug-float-1 Ces valeurs sont utilisées pour le déboguage.

5.1.3 Fonctions

Généralement, ces fonctions sont toutes les deux ajoutées à servo-thread dans l'ordre suivant :

motion-command-handler Processus des commandes de mouvement provenant de l'interface utilisateur.

motion-controller Lance le contrôleur de mouvement d'emc.

5.2 axis.N (temps réel)

Ces pins et paramètres sont créés par le module temps réel "`motmod`". Ce sont en fait des valeurs d'articulations, mais les pins et les paramètres sont toujours appelés "axis.N".[1] Ils sont lus et mis à jour par la fonction **motion-controller**.

5.2.1 Pins

axis.N.amp-enable-out OUT bit TRUE si l'ampli de cet axe doit être activé.

axis.N.amp-fault-in IN bit Doit être mis TRUE si une erreur externe est détectée sur l'ampli de cet axe.

axis.N.home-sw-in IN bit Doit être mis TRUE si le contact d'origine de cet axe est pressé.

axis.N.homing OUT bit TRUE si la prise d'origine de cette axe a été faite.

axis.N.pos-lim-sw-in IN bit Doit être mis TRUE si le fin de course de limite positive de cet axe est activé.

[1] Dans une machine à "cinématique triviale", il y a correspondance une pour une, entre les articulations et les axes.
NDT : nous utilisons dans cette traduction le terme "axe", dans le cas d'une cinématique non triviale il devra être remplacé par le terme "articulation" (joint).

axis.N.neg-lim-sw-in IN bit Doit être mis TRUE si le fin de course de limite négative de cet axe est activé.

axis.N.index-enable IO BIT Doit être reliée à la broche “index-enable” du codeur de cet axe pour activer la prise d’origine sur l’impulsion d’index.

axis.N.jog-counts IN s32 Connection à la broche “counts” d’un codeur externe utilisé comme manivelle.

axis.N.jog-enable IN bit Quand elle est TRUE (et en mode manuel), tout changement dans “jog-counts” se traduira par un mouvement. Quand elle est FALSE, “jog-counts” sera ignoré.

axis.N.jog-scale IN float Fixe la distance, en unités machine, du déplacement pour chaque évolution de “jog-counts”.

axis.N.jog-vel-mode IN bit Quand elle est FALSE (par défaut), la manivelle fonctionne en mode position. L’axe se déplace exactement selon l’incrément de jog sélectionné pour chaque impulsion, sans s’occuper du temps que prendra le mouvement. Quand elle est TRUE, la manivelle fonctionne en mode vitesse. Le mouvement s’arrête quand la manivelle s’arrête, même si le mouvement commandé n’est pas achevé.

axis.N.motor-pos-cmd OUT float La position commandée pour cet axe.

axis.N.motor-pos-fb IN float La position actuelle de cet axe.

axis.N.joint-pos-cmd Position commandée de l’axe (par opposition à celle du moteur). Il peut y avoir un décalage entre la position de l’axe et celle du moteur, par exemple, le processus de prise d’origine peut ajuster cet écart.

axis.N.joint-pos-fb Le retour de position (par opposition à celui du moteur).

5.2.2 Paramètres

Beaucoup de ces paramètres servent d’aide au déboguage et sont sujets aux changements ou au retrait à tout moment.

axis.N.active TRUE quand cet axe est actif.

axis.N.backlash-corr Valeur brute de ratrappage de jeu.

axis.N.backlash-filt Valeur filtrée de ratrappage de jeu (respect des limites de mouvement).

axis.N.backlash-vel Vitesse de ratrappage de jeu.

axis.N.coarse-pos-cmd

axis.N.error TRUE quand une erreur ce produit sur cet axe, ex : une limite de course est atteinte.

axis.N.f-error Erreur de suivi actuelle.

axis.N.f-error-lim Limite d’erreurs de suivi.

axis.N.f-errored TRUE quand cet axe à dépassé la limite d’erreurs de suivi.

axis.N.free-pos-cmd Position commandée en “free planner” pour cet axe.

axis.N.free-tp-enable TRUE quand le “free planner” est activé pour cet axe.

axis.N.free-vel-lim Vitesse limite en “free planner”.

axis.N.home-state Refète l’étape de la prise d’origine en cours actuellement.

axis.N.homed TRUE si la prise d’origine de cet axe a bien été réalisée.

axis.N.in-position TRUE si cet axe, utilisant le “free planner”, a atteint un arrêt.

axis.N.joint-vel-cmd Vitesse commandée des axes.

axis.N.neg-hard-limit Fin de course de limite d’axe négative.

axis.N.neg-soft-limit Limite soft négative de cet axe.

axis.N.pos-hard-limit Fin de course de limite d’axe positive.

axis.N.pos-soft-limit Limite soft positive de cet axe.

5.3 iocontrol (espace utilisateur)

Ces pins sont créées par le contrôleur d'entrées/sorties de l'espace utilisateur, habituellement appelé "`io`".

5.3.1 Pins

iocontrol.0.coolant-flood TRUE quand l'arrosage est demandé.

iocontrol.0.coolant-mist TRUE quand le brouillard est demandé.

iocontrol.0.emc-enable-in Doit être mise FALSE quand un arrêt d'urgence externe est enfoncé.

iocontrol.0.lube TRUE quand le graissage centralisé est commandé.

iocontrol.0.lube_level Doit être mise TRUE quand le niveau d'huile est assez haut.

iocontrol.0.tool-change TRUE quand un changement d'outil est demandé.

iocontrol.0.tool-changed Doit être mise TRUE quand le changement d'outil est terminé.

iocontrol.0.tool-prep-number Numéro du prochain outil, donné dans le mot T selon RS274NGC.

iocontrol.0.tool-prepare TRUE quand une préparation d'outil est demandée.

iocontrol.0.tool-prepared Doit être mise TRUE quand une préparation d'outil est terminée.

iocontrol.0.user-enable-out FALSE quand un arrêt d'urgence interne est enfoncé.

iocontrol.0.user-request-enable TRUE quand l'utilisateur relâche l'arrêt d'urgence.

Quatrième partie

HAL : Spécifications

Chapitre 6

Introduction

6.1 Qu'est-ce que HAL ?

HAL est le sigle de Hardware Abstraction Layer, le terme Anglais pour Couche d'Abstraction Matériel[1]. Au plus haut niveau, il s'agit simplement d'une méthode pour permettre à un grand nombre de "modules" d'être chargés et interconnectés pour assembler un système complexe. La partie "matériel" devient abstraite parce que HAL a été conçu à l'origine pour faciliter la configuration d'EMC pour une large gamme de matériels. Bon nombre de ces modules sont des pilotes de périphériques. Cependant, HAL peut faire beaucoup plus que configurer les pilotes du matériel.

6.1.1 HAL est basé sur le système traditionnel d'étude des projets techniques

HAL est basé sur le même principe que celui utilisé pour l'étude des circuits et des systèmes techniques, il va donc être utile d'examiner d'abord ces principes.

N'importe quel système, y compris les machines CNC, est fait de composants interconnectés. Pour les machines CNC, ces composants pourraient être le contrôleur principal, les amplis de servomoteurs, les amplis ou les commandes de puissance des moteurs pas à pas, les moteurs, les codeurs, les interrupteurs de fin de course, les panneaux de boutons de commande, les manivelles, peut être aussi un variateur de fréquence pour le moteur de broche, un automate programmable pour gérer le changeur d'outils, etc. Le constructeur de machine doit choisir les éléments, les monter et les câbler entre eux pour obtenir un système complet et fonctionnel.

6.1.1.1 Choix des organes

Il ne sera pas nécessaire au constructeur de machine de se soucier du fonctionnement de chacun des organes, il les traitera comme des boîtes noires. Durant la phase de conception, il décide des éléments qu'il va utiliser, par exemple, moteurs pas à pas ou servomoteurs, quelle marque pour les amplis de puissance, quels types d'interrupteurs de fin de course et combien il en faudra, etc. La décision d'intégrer tel ou tel élément spécifique plutôt qu'un autre, repose sur ce que doit faire cet élément et sur ses caractéristiques fournies par le fabricant. La taille des moteurs et la charge qu'ils doivent supporter affectera le choix des interfaces de puissance nécessaires pour les piloter. Le choix de l'ampli affectera le type des signaux de retour demandés ainsi que le type des signaux de vitesse et de position qui doivent lui être transmis.

Dans le monde de HAL, l'intégrateur doit décider quels composants de HAL sont nécessaires. Habituellement, chaque carte d'interface nécessite un pilote. Des composants supplémentaires peuvent être demandés, par exemple, pour la génération logicielle des impulsions d'avance, les fonctionnalités des automates programmables, ainsi qu'une grande variété d'autres tâches.

[1] Note du traducteur : nous garderons le sigle HAL dans toute la documentation.

6.1.1.2 Étude des interconnections

Le créateur d'un système matériel, ne sélectionnera pas seulement les éléments, il devra aussi étudier comment ils doivent être interconnectés. Chaque boîte noire dispose de bornes, deux seulement pour un simple contact, ou plusieurs douzaines pour un pilote de servomoteur ou un automate. Elles doivent être câblées entre elles. Les moteurs câblés à leurs interfaces de puissance, les fins de course câblés au contrôleur et ainsi de suite. Quand le constructeur de machine commence à travailler sur le câblage, il crée un grand plan de câblage représentant tous les éléments de la machine ainsi que les connections qui les relient entre eux.

En utilisant HAL, les *composants* sont interconnectés par des *signaux*. Le concepteur peut décider quels signaux sont nécessaires et à quoi ils doivent être connectés.

6.1.1.3 Implémentation

Une fois que le plan de câblage est complet, il est possible de construire la machine. Les pièces sont achetées et montées, elles peuvent alors être câblées et interconnectées selon le plan de câblage. Dans un système physique, chaque interconnection est un morceau de fil qui doit être coupé et raccordé aux bornes appropriées.

HAL fournit un bon nombre d'outils d'aide à la "construction" d'un système HAL. Certains de ces outils permettent de "connecter" (ou déconnecter) un simple "fil". D'autres permettent d'enregistrer une liste complète des organes, du câblage et d'autres informations à propos du système, de sorte qu'il puisse être "reconstruit" d'une simple commande.

6.1.1.4 Mise au point

Très peu de machines marchent bien dès la première fois. Lors des tests, le technicien peut utiliser un appareil de mesure pour voir si un fin de course fonctionne correctement ou pour mesurer la tension fournie aux servomoteurs. Il peut aussi brancher un oscilloscope pour examiner le réglage d'une interface ou pour rechercher des interférences électriques et déterminer leurs sources. En cas de problème, il peut s'avérer indispensable de modifier le plan de câblage, peut être que certaines pièces devront être recâblées différemment, voir même remplacées par quelque chose de totalement différent.

HAL fournit les équivalents logiciels du voltmètre, de l'oscilloscope, du générateur de signaux et les autres outils nécessaires à la mise au point et aux réglages d'un système. Les même commandes utilisées pour construire le système, seront utilisées pour faire les changements indispensables.

6.1.2 En résumé

Ce document est destiné aux personnes déjà capables de concevoir ce type de réalisation matérielle, mais qui ne savent pas comment connecter le matériel à EMC.

La conception de matériel, telle que décrite précédemment, s'arrête à l'interface de contrôle. Au delà, il y a un tas de boîtes noires, relativement simples, reliées entre elles pour faire ce qui est demandé. À l'intérieur, un grand mystère, c'est juste une grande boîte noire qui fonctionne, nous osons l'espérer.

HAL étend cette méthode traditionnelle de conception de matériel à l'intérieur de la grande boîte noire. Il transforme les pilotes de matériels et même certaines parties internes du matériel, en petites boîtes noires pouvant être interconnectées, elles peuvent alors remplacer le matériel externe. Il permet au "plan de câblage" de faire voir une partie du contrôleur interne et non plus, juste une grosse boîte noire. Plus important encore, il permet à l'intégrateur de tester et de modifier le contrôleur en utilisant les mêmes méthodes que celles utilisées pour le reste du matériel.

Les termes tels que moteurs, amplis et codeurs sont familiers aux intégrateurs de machines. Quand nous parlons d'utiliser un câble extra souple à huit conducteurs blindés pour raccorder un codeur de position à sa carte d'entrées placée dans l'ordinateur. Le lecteur comprend immédiatement de quoi il s'agit et se pose la question, "quel type de connecteurs vais-je devoir monter de chaque côté de ce câble ?" Le même genre de réflexion est indipensable pour HAL mais le cheminement de la pensée est différent. Au début les mots utilisés par HAL pourront sembler un peu étranges, mais ils sont identiques au concept de travail évoluant d'une connection à la suivante.

HAL repose sur une seule idée, l'idée d'étendre le plan de câblage à l'intérieur du contrôleur. Si vous êtes à l'aise avec l'idée d'interconnecter des boîtes noires matérielles, vous n'aurez sans doute aucune difficulté à utiliser HAL pour interconnecter des boites noires logicielles.

6.2 Concept de HAL

Cette section est un glossaire qui définit les termes clés de HAL mais il est différent d'un glossaire traditionnel en ce sens que les termes ne sont pas classés par ordre alphabétique. Ils sont classés par leur relation ou par le sens du flux à l'intérieur de HAL.

Component : (Composant) Lorsque nous avons parlé de la conception du matériel, nous avons évoqué les différents éléments individuels comme "pièces", "modules", "boîtes noires", etc. L'équivalent HAL est un "component" ou "HAL component". (ce document utilisera : "HAL component" quand la confusion avec un autre type de composant est possible, mais normalement, utilisez juste : "component".) Un HAL component est une pièce logicielle avec, bien définis, des entrées, des sorties, un comportement, qui peuvent éventuellement être interconnectés.

Parameter : (Paramètre) De nombreux composants matériels ont des réglages qui ne sont raccordés à aucun autre composant mais qui sont accessibles. Par exemple, un ampli de servomoteur a souvent des potentiomètres de réglage et des points tests sur lesquels on peut poser une pointe de touche de voltmètre ou une sonde d'oscilloscope pour visualiser le résultat des réglages. Les HAL components aussi peuvent avoir de tels éléments, ils sont appelés "parameters". Il y a deux types de paramètres : "Input parameters" qui sont des équivalents des potentiomètres. Ce sont des valeurs qui peuvent être réglées par l'utilisateur, elles gardent leur valeur jusqu'à un nouveau réglage. "Output parameters" qui ne sont pas ajustables. Ils sont équivalents aux points tests qui permettent de mesurer la valeur d'un signal interne.

Pin : (Broche) Les composants matériels ont des broches qui peuvent être interconnectées entre elles. L'équivalent HAL est une "pin" ou "HAL pin". ("HAL pin" est utilisé quand c'est nécessaire pour éviter la confusion.) Toutes les HAL pins sont nommées et les noms des pins sont utilisés lors des interconnexions entre elles. Les HAL pins sont des entités logicielles qui n'existent qu'à l'intérieur de l'ordinateur.

Physical_Pin : (Broche physique) La plupart des interfaces d'entrées/sorties ont des broches physiques réelles pour leur connection avec l'extérieur, par exemple, les broches du port parallèle. Pour éviter la confusion, elles sont appelées "physical pins". Ce sont des repères pour faire penser au monde physique réel.

Signal : Dans une machine physique réelle, les terminaisons des différents organes sont reliées par des fils. L'équivalent HAL d'un fil est un "signal" ou "HAL signal". Ces signaux connectent les "HAL pins" entre elles comme le requiert le concepteur de la machine. Les "HAL signals" peuvent être connectés et déconnectés à volonté (même avec la machine en marche).

Type : Quand on utilise un matériel réel, il ne viendrait pas à l'idée de connecter la sortie 24V d'un relais à l'entrée analogique +/-10V de l'ampli d'un servomoteur. Les "HAL pins" ont les même restrictions, qui sont fondées sur leur type. Les "pins" et les "signals" ont tous un type, un "signals" ne peux être connecté qu'à une "pins" de même type. Il y a actuellement les 4 types suivants :

- BIT - une simple valeur vraie ou fausse TRUE/FALSE ou ON/OFF
- FLOAT - un flottant de 32 bits, avec approximativement 24 bits de résolution et plus de 200 bits d'échelle dynamique.

– U32 - un entier non signé de 32 bits, les valeurs légales vont de 0 à +4294967295
– S32 - un entier signé de 32 bits, les valeurs légales vont de -2147483648 à +2147483647

Function : (Fonction) Les composants matériels réels ont tendance à réagir immédiatement à leurs signaux d'entrée. Par exemple, si la tension d'entrée d'un ampli de servo varie, la sortie varie aussi automatiquement. Les composants logiciels ne peuvent pas réagir immédiatement. Chaque composant a du code spécifique qui doit être exécuté pour faire ce que le composant est sensé faire. Dans certains cas, ce code tourne simplement comme une partie du composant. Cependant dans la plupart des cas, notamment dans les composants temps réel, le code doit être exécuté selon un ordre bien précis et à des intervalles trés précis. Par exemple, les données en entrée doivent d'abord être lues avant qu'un calcul ne puisse être effectué sur elles et les données en sortie ne peuvent pas être écrites tant que le calcul sur les données d'entrée n'est pas terminé. Dans ces cas, le code est confié au système sous forme de "functions". Chaque “function” est un bloc de code qui effectue une action spécifique. L'intégrateur peut utiliser des "threads" pour combiner des séries de “functions” qui seront exécutées dans un ordre particulier et selon des intervalles de temps spécifiques.

Thread : (Fil) Un "thread" est une liste de “functions” qui sont lancées à intervalles spécifiques par une tâche temps réel. Quand un “thread” est créé pour la première fois, il a son cadencement spécifique (période), mais pas de “functions”. Les “functions” seront ajoutées au “thread” et elle seront exécutées dans le même ordre, chaque fois que le “tread” tournera.

Prenons un exemple, supposons que nous avons un composant de port parallèle nommé “hal_parport”. Ce composant défini une ou plusieurs “HAL pins” pour chaque “physical pin”. Les “pins” sont décrites dans ce composant, comme expliqué dans la section “component“de cette doc, par : leurs noms, comment chaque “pin” est en relation avec la “physical pin”, est-elle inversée, peut-on changer sa polarité, etc. Mais ça ne permet pas d'obtenir les données des “HAL pins” aux “physical pins”. Le code est utilisé pour faire ça, et c'est la où les “functions” entrent en oeuvre. Le composant parport nécessite deux “functions” : une pour lire les broches d'entrée et mettre à jour les “HAL pins”, l'autre pour prendre les données des “HAL pins” et les écrire sur les broches de sortie “physical pins”. Ce deux fonctions font partie du pilote “hal_parport”.

6.3 Composants HAL

Chaque composant HAL est un morceau de logiciel avec, bien définis, des entrées, des sorties et un comportement. Ils peuvent être installés et interconnectés selon les besoins. Cette section liste certains des composants actuellement disponibles et décrit brièvement ce que chacun fait. Les détails complets sur chacun seront donnés plus loin dans ce document.

6.3.1 Programmes externes attachés à HAL

motion Un module temps réel qui accepte les commandes de mouvement en NML et inter-agit avec HAL

iocontrol Un module d'espace utilisateur qui accepte les commandes d'entrée/sortie (I/O) en NML et inter-agit avec HAL

classicladder Un automate programmable en langage à contacts utilisant HAL pour les entrées/sorties (I/O)

halui Un espace de utilisateur de programmation qui inter-agit avec HAL et envoie des commandes NML, Il est destiné à fonctionner comme une interface utilisateur en utilisant les boutons et interrupteurs externes.

6.3.2 Composants internes

stepgen Générateur d'impulsions de pas avec boucle de position.

encoder Codeur/compteur logiciel.

pid Boucle de contrôle Proportionnelle/Intégrale/Dérivée.

siggen Générateur d'ondes : sinusoïdale/cosinusoïdale/triangle/carrée, pour la mise au point.

supply Une simple alimentation, pour la mise au point

blocks Un assortiment de composants (mux, demux, or, and, integ, ddt, limit, wcomp, etc.)

6.3.3 Pilotes de matériels

hal_ax5214h Un pilote pour la carte d'entrées/sorties Axiom Measurement & Control AX5241H

hal_m5i20 Un pilote pour la carte Mesa Electronics 5i20

hal_motenc Un pilote pour la carte Vital Systems MOTENC-100

hal_parport Pilote pour le(ou les) port(s) parallèle(s).

hal_ppmc Un pilote pour la famille de contrôleurs Pico Systems (PPMC, USC et UPC)

hal_stg Un pilote pour la carte Servo To Go (versions 1 & 2)

hal_vti Un pilote pour le contrôleur Vigilant Technologies PCI ENCDAC-4

6.3.4 Outils-Utilitaires

halcmd Ligne de commande pour la configuration et les réglages.

halgui Outil graphique pour la configuration et les réglages. (pas encore implémenté).

halmeter Un multimètre pour les signaux HAL.

halscope Un oscilloscope digital à mémoire, complétement fonctionnel pour les signaux HAL.

Chacun de ces modules est décrit en détail dans les chapitres suivants.

6.4 Tinkertoys, Erector Sets, Legos et le HAL

Cette première introduction au concept de HAL peut être un peu déconcertante pour l'esprit. Construire quelque chose avec des blocs peut être un défi, pourtant certains jeux de construction avec lesquels nous avons joué étant enfants peuvent nous aider à construire un système HAL.

6.4.1 Une tour

> Je regardais mon fils et sa petite fille de six ans construire une tour à partir d'une boîte pleine de blocs de différentes tailles, de barres et de pièces rondes, des sortes de couvercles. L'objectif était de voir jusqu'où la tour pouvait monter. Plus la base était étroite plus il restait de pièces pour monter. Mais plus la base était étroite, moins la tour était stable. Je les voyais étudier combien de blocs ils pouvaient poser et où ils devaient les poser pour conserver l'équilibre avec le reste de la tour.

La notion d'empilage de cartes pour voir jusqu'où on peut monter est une très vieille et honorable manière de passer le temps. En première lecture, l'intégrateur pourra avoir l'impression que construire un HAL est un peu comme ça. C'est possible avec une bonne planification, mais l'intégrateur peut avoir à construire un système stable aussi complexe qu'une machine actuelle l'exige.

6.4.2 Erector Sets[2] (Meccano en France)

C'était une grande série de boites de construction en métal, des tôles perforées, plates ou en cornières, toutes avaient des trous régulierements espacés. Vous pouviez concevoir des tas de choses et les monter avec ces éléments maintenus entre eux par des petits boulons.

> J'ai eu ma première boîte Erector pour mon quatrième anniversaire. Je sais que la boîte était prévue pour des enfants beaucoup plus agés que moi. Peut être que mon père se faisait vraiment un cadeau à lui même. J'ai eu une période difficile avec les petites vis et les petits écrous. J'ai vraiment eu envie d'avoir quatre bras, un pour visser avec le tournevis, un pour tenir la vis, les pièces et l'écrou. En persévérant, de même qu'en agaçant mon père, j'ai fini par avoir fait tous les montages du livret. Bientôt, je lorgnais vers les plus grandes boîtes qui étaient imprimées sur ce livret. Travailler avec ces pièces de taille standard m'a ouvert le monde de la construction et j'ai bientôt été au delà des projets illustrés.

Les composants Hal ne sont pas tous de même taille ni de même forme mais ils permettent d'être regroupés en larges unités qui feront bien du travail. C'est dans ce sens qu'ils sont comme les pièces d'un jeu Erector. Certains composants sont longs et minces. Ils connectent essentiellement les commandes de niveau supérieur aux "physical pins". D'autres composants sont plus comme les plateformes rectangulaires sur lesquelles des machines entières pourraient être construites. Un intégrateur parviendra rapidement au delà des brefs exemples et commencera à assembler des composants entre eux d'une manière qui lui sera propre.

6.4.3 Tinkertoys[3]

> Le jouet en bois Tinkertoys est plus humain que l'acier froid de l'Erector. Le coeur de la construction avec TinkerToys est un connecteur rond avec huit trous équidistants sur la circonférence. Il a aussi un trou au centre, perpendiculaire aux autres trous répartis autour du moyeu.
>
> Les moyeux pouvaient être connectés avec des tiges rondes de différentes longueurs. Le constructeur pouvait faire une grosse roue à l'aide de rayons qui partaient du centre.
>
> Mon projet favori était une station spaciale rotative. De courtes tiges rayonnaient depuis les trous du moyeu central et étaient connectées avec d'autres moyeux aux extrémités des rayons. Ces moyeux extérieurs étaient raccordés entre eux avec d'autres rayons. Je passais des heures à rêver de vivre dans un tel dispositif, marchant de moyeu en moyeu et sur la passerelle extérieure qui tournait lentement à cause de la gravité dans l'espace en état d'apesanteur. Les provisions circulaient par les rayons et les ascenceurs qui les transféraient dans la fusée arrimée sur le rayon central pendant qu'on déchargeait sa précieuse cargaison.

L'idée qu'une "pin" ou qu'un "component" est la plaque centrale pour de nombreuses connections est aussi une notion facile avec le HAL. Les exemples deux à quatre (voir section 7) connectent le multimètre et l'oscilloscope aux signaux qui sont prévus pour aller ailleurs. Moins facile, la notion d'un moyeu pour plusieurs signaux entrants. Mais, c'est également possible avec l'utilisation appropriée des fonctions dans ce composant de moyeu qui manipulent les signaux quand ils arrivent, venant d'autres composants.

Une autre réflexion qui vient à partir de ce jouet mécanique est une représentation de "HAL threads". Un "thread" pourrait ressembler un peu à un chilopode, une chenille, ou un perce-oreille. Une épine dorsale, des "HAL components", raccordés entre eux par des tiges, les "HAL signals". Chaque composant prend dans ses propres paramètres et selon l'état de ses broches d'entrée, les passe sur ses broches de sortie à l'intention du composant suivant. Les signaux voyagent ainsi de bout en bout, le long de l'épine dorsale où ils sont ajoutés ou modifiés par chaque composant son tour venu.

[2] Le jeu Erector Set est une invention de AC Gilbert
[3] Tinkertoy est maintenant registered trademark of the Hasbro company.

Les “Threads” sont tous synchronisés et exécutent une série de tâches de bout en bout. Une représentation mécanique est possible avec Thinkertoys si on pense à la longueur du jouet comme étant la mesure du temps mis pour aller d'un bout à l'autre. Un thread, ou épine dorsale, très différent est créé en connectant le même ensemble de modules avec des tiges de longueur différente. La longueur totale de l'épine dorsale peut aussi être changée en jouant sur la longueur des tiges pour connecter les modules. L'ordre des opérations est le même mais le temps mis pour aller d'un bout à l'autre est très différent.

6.4.4 Un exemple en Lego[4]

Lorsque les blocs de Lego sont arrivés dans nos magasins, ils étaient à peu près tous de la même taille et de la même forme. Bien sûr il y avait les demi taille et quelques uns en quart de taille mais tous rectangulaires. Les blocs de Lego se relient ensembles en enfonçant les broches mâles d'une pièce dans les trous femelles de l'autre. En superposant les couches, les jonctions peuvent être rendues très solides, même aux coins et aux tés.

> J'ai vu mes enfants et mes petits-enfants construire avec des pièces Lego (les mêmes Lego). Il y en a encore quelques milliers dans une vieille et lourde boîte en carton qui dort dans un coin de la salle de jeux. Ils sont stockés dans cette boîte car c'était trop long de les ranger et de les ressortir à chacune de leur visite et ils étaient utilisés à chaque fois. Il doit bien y avoir les pièces de deux douzaines de boîtes différentes de Lego. Les petits livrets qui les accompagnaient ont été perdus depuis longtemps, mais la magie de la construction avec l'imbrication de ces pièces toutes de la même taille est quelque chose à observer.

6.5 Problèmes de timming dans HAL

Contrairement aux modèles physiques du câblage entre les boîtes noires sur lequel, nous l'avons dit, HAL est basé, il suffit de relier deux broches avec un signal hal, on est loin de l'action physique.

La vraie logique à relais consiste en relais connectés ensembles, quand un relais s'ouvre ou se ferme, le courant passe (ou s'arrête) immédiatement. D'autres bobines peuvent changer d'état etc. Dans le style langage à contacts d'automate comme le Ladder ça ne marche pas de cette façon. Habituellement dans un Ladder simple passe, chaque barreau de l'échelle est évalué dans l'ordre où il se présente et seulement une fois par passe. Un exemple parfait est un simple Ladder avec un contact en série avec une bobine. Le contact et la bobine actionnent le même relais.

Si c'était un relais conventionnel, dès que la bobine est sous tension, le contact s'ouvre et coupe la bobine, le relais retombe etc. Le relais devient un buzzer.

Avec un automate programmable, si la bobine est OFF et que le contact est fermé quand l'automate commence à évaluer le programme, alors à la fin de la passe, la bobine sera ON. Le fait que la bobine ouvre le contact qui la prive de courant est ignoré jusqu'à la prochaine passe. À la passe suivante, l'automate voit que le contact est ouvert et désactive la bobine. Donc, le relais va battre rapidement entre on et off à la vitesse à laquelle l'automate évalue le programme.

Dans HAL, c'est le code qui évalue. En fait, la version Ladder HAL temps réel de ClassicLadder exporte une fonction pour faire exactement cela. Pendant ce temps, un thread exécute les fonctions spécifiques à intervalle régulier. Juste comme on peut choisir de régler la durée de la boucle de programme d'un automate programmable à 10ms, ou à 1 seconde, on peut définir des “HAL threads” avec des périodes différentes.

Ce qui distingue un thread d'un autre n'est pas ce qu'il fait mais quelles fonctions lui sont attachées. La vraie distinction est simplement combien de fois un thread tourne.

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

Dans EMC on peut avoir un thread à 50μs et un thread à 1ms. En se basant sur les valeurs de BASE_PERIOD et de SERVO_PERIOD. Valeurs fixées dans le fichier ini.

La prochaine étape consiste à décider de ce que chaque thread doit faire. Certaines de ces décisions sont les mêmes dans (presque) tous les systèmes emc. Par exemple, le gestionnaire de mouvement est toujours ajouté au servo-thread.

D'autres connections seront faites par l'intégrateur. Il pourrait s'agir de brancher la lecture d'un codeur par une carte STG à un DAC pour écrire les valeurs dans le servo thread, ou de brancher une fonction stepgen au base-thread avec la fonction parport pour écrire les valeurs sur le port.

Chapitre 7

Tutoriel de HAL

7.1 Introduction

La configuration passe de la théorie à la pratique de HAL. Pour ceux qui ont juste un peu de pratique avec la programmation des ordinateurs, cette section est le “Hello World” de HAL. Comme indiqué précédemment halrun peut être utilisé pour créer un système qui fonctionne. Il s’agit d’un outil de configuration et de mise au point en ligne de commande ou en fichier texte. Les exemples suivants illustrent son installation et son fonctionnement.

7.1.1 Notation

Les exemples en ligne de commande sont représentés en police **`bold typewriter`**. Les réponses de l’ordinateur sont en police `typewriter`. Le texte optionnel est entre crochets `[comme ça]`. Le texte `<comme ça>` représente un champ qui peut prendre différentes valeurs, le paragraphe adjacent explique quelles sont les valeurs appropriées. Les élements textuels séparés par des barres verticales indiquent qu’une valeur ou l’autre mais pas les deux, doit être présente. Toutes les lignes de commandes considèrent que vous êtes dans le répertoire `emc2/` , les chemins sont affichés en accord avec ce principe.

7.1.2 L’environnement RTAPI

RTAPI est le sigle de Real Time Application Programming Interface. De nombreux composants HAL travaillent en temps réel et tous les composants de HAL stockent leurs données dans la mémoire partagée, de sorte que les composants temps réel puissent y accéder. Normalement, Linux ne prend pas en charge les programmes temps réel ni le type de mémoire partagée dont HAL a besoin. Heureusement, il existe des systèmes d’exploitation temps réel RTOS qui fournissent les extensions nécessaires à Linux. Malheureusement, chaque RTOS fait les choses différemment des autres.

Pour remédier à ces différences, l’équipe d’EMC a proposé RTAPI, qui fournit une manière cohérente aux programmes de parler au RTOS. Si vous êtes un programmeur qui veut travailler à l’intérieur d’EMC, vous pouvez étudier `emc2/src/rtapi/rtapi.h` pour comprendre l’API. Mais si vous êtes une personne normale, tout ce que vous avez besoin de savoir à propos de RTAPI est qu’il doit être (avec le RTOS) chargé dans la mémoire de votre ordinateur avant de pouvoir faire n’importe quoi avec HAL.

Pour ce tutoriel, nous allons supposer que vous avez compilé avec succès l’arborescence emc2/src et, si nécessaire, invoqué le script `emc-environment` pour préparer votre shell. Dans ce cas, tout ce que vous avez à faire est de charger le RTOS requis et les modules RTAPI dans la mémoire. Tapez juste les commandes suivantes dans une console :

```
emc2$ halrun
halcmd :
```

Une fois l'OS temps réel et RTAPI chargés, vous pouvez aller au premier exemple. Notez que le prompt a changé, il est passé de "$" à "halcmd". La raison en est que les commandes ultérieures seront interprétées comme des commandes HAL et non plus comme des commandes shell. `halrun` est un simple script shell, il est plus ou moins équivalent de lancer :

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

Pour quitter `halcmd` et que halrun arrête le système temps réel, tapez :

```
emc2$ realtime stop
```

Vous pouvez également passer des arguments à halrun, il seront répercutés sur halcmd, ou passer le nom d'un fichier .hal. Parce que `halrun` arrête le système temps réel quand il se termine, le fichier hal fonctionnera de cette façon et s'arrêtera généralement avec une commande comme : `loadrt -w halscope`.

7.2 Tab-complétion

Votre version de halcmd peut inclure la complétion avec la touche tab. Au lieu de compléter les noms de fichiers comme le fait un shell, il complète les commandes avec les identifiants HAL. Essayez de presser la touche tab après le début d'une commande HAL :

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

7.3 Un exemple simple

7.3.1 Chargement d'un composant temps réel

Pour le premier exemple, nous allons utiliser un composant HAL appelé `siggen`, qui est un simple générateur de signal. Une description complète du composant `siggen` reste disponible à la section 14.7 de ce document. Il s'agit d'un composant en temps réel, mis en oeuvre comme un module du noyau Linux. Pour charger `siggen` utiliser la commande `halcmd loadrt` :

```
halcmd : loadrt siggen
```

7.3.2 Examiner HAL

Maintenant que le module est chargé, il faut introduire `halcmd`, l'outil en ligne de commande utilisé pour configurer le HAL. Pour une description plus complète essayez : `man halcmd`, ou consultez la section `halcmd` à la section 10.1 de ce document. La première commande de halcmd et show, qui affichera les informations concernant l'état actuel du HAL. Pour afficher tout ce qui est installé tapez :

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

Puisque `halcmd` lui même est un composant HAL, il sera toujours présent dans la liste[1]. La liste montre aussi le composant `siggen` que nous avions installé à l'étape précédente. Le "RT" sous "Type" indique que `siggen` est un composant temps réel.

Ensuite, voyons quelles pins `siggen` rend disponibles :

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

Cette commande affiche toutes les pins dans le HAL. Un système complexe peut avoir plusieurs dizaines ou centaines de pins. Mais pour le moment il y a seulement cinq pins. Toutes ces cinq pins sont des flottants, elles transportent toutes des données en provenance du composant `siggen`. Puisque nous n'avons pas encore exécuté le code contenu dans le composant, toutes les pins ont une valeur de zéro.

L'étape suivante consiste à examiner les paramètres :

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

La commande "show param" affiche tous les paramètres de HAL. Pour le moment chaque paramètre à la valeur par défaut attribuée quand le composant a été chargé. Notez dans la colonne `Dir`. Les valeurs marquées `-W` écriture possible, pour ceux qui ne sont jamais modifié par le composant lui-même, mais qui sont modifiable par l'utilisateur pour contrôler le composant. Nous verrons comment plus tard. Les paramètres marqués `R-` sont en lecture seule. Il ne peuvent être modifiés que par le composant. Finalement, les paramètres marqués `RW` sont en lecture/écriture. Ils peuvent être modifiés par le composant et aussi par l'utilisateur. Nota : les paramètres `siggen.0.update.time` et `siggen.0.update.tmax` existent dans un but de déboguage, ils ne sont pas couverts par cette documentation.

La plupart des composants temps réel exportent une ou plusieurs fonctions pour que le code qu'elles contiennent soit exécuté en temps réel. Voyons ce que la fonction `siggen` exporte :

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

Le composant siggen exporte une seule fonction. Il nécessite un flottant (Floating Point). Il n'est lié à aucun thread, puisque "users" est à zero[2].

[1] Le nombre après halcmd dans la liste des composants est le "process ID". Il est toujours possible de lancer plus d'une instance de halcmd en même temps (dans différentes fenêtres par exemple), Le numéro PID est ajouté à la fin du nom pour rendre celui-ci unique.

[2] Les champs codeaddr et arg ont été utilisés pendant le développement et devraient probablement disparaître.

7.3.3 Exécuter le code temps réel

Pour faire tourner le code actuellement contenu dans la fonction `siggen.0.update`, nous avons besoin d'un thread temps réel. C'est le composant appelé `threads` qui est utilisé pour créer le nouveau thread. Créons un thread appelé `test-thread` avec une période de 1ms (1000000ns) :

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

Voyons si il fonctionne :

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

Il fonctionne. La période n'est pas exactement de 1000000ns à cause des limitations dues au matériel, mais nous avons bien un thread qui tourne à une période approximativement correcte et qui peut manipuler des fonctions en virgule flottante. La prochaine étape sera de connecter la fonction au thread :

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

Pour le moment nous avions utilisé `halcmd` seulement pour regarder le HAL. Mais cette fois-ci, nous avons utilisé la commande `addf` (add function) pour changer quelque chose dans le HAL. Nous avons dit à `halcmd` d'ajouter la fonction `siggen.0.update` au thread `test-thread` et la commande suivante indique qu'il a réussi :

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

Il y a une étape de plus avant que le composant `siggen` ne commence à générer des signaux. Quand le HAL est démarré pour la première fois, les threads ne sont pas en marche. C'est pour vous permettre de compléter la configuration du système avant que le code temps réel ne démarre. Une fois que vous êtes satisfait de la configuration, vous pouvez lancer le code temps réel comme ceci :

```
halcmd : start
```

Maintenant le générateur de signal est en marche. Regardons ses pins de sortie :

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

Nous avons fait, très rapidement, deux commandes `show pin` et vous pouvez voir que les sorties ne sont plus à zéro. Les sorties sinus, cosinus, dents de scie et triangle changent constamment. La sortie carrée fonctionne également, mais elle passe simplement de +1.0 à -1.0 à chaque cycle.

7.3.4 Modifier des paramètres

La réelle puissance de HAL est de permettre de modifier les choses. Par exemple, on peut utiliser la commande `setp` pour ajuster la valeur d'un paramètre. Modifions l'amplitude du signal de sortie du générateur de 1.0 à 5.0 :

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

Voyons encore une fois les paramètres et les pins :

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

Notez que la valeur du paramètre `siggen.0.amplitude` est bien passée à 5.000 et que les pins ont maintenant des valeurs plus grandes.

7.3.5 Enregistrer la configuration de HAL

La plupart de ce que nous avons fait jusqu'ici avec `halcmd` a été de simplement regarder les choses avec la commande `show`. Toutefois, deux commandes ont rééllement modifié des valeurs. Au fur et à mesure que nous concevons des systèmes plus complexes avec HAL, nous allons utiliser de nombreuses commandes pour le configurer comme nous le souhaitons. HAL a une mémoire d'éléphant et peut retenir sa configuration jusqu'à ce qu'il s'arrête. Mais qu'en est-il de la prochaine fois ? Nous ne voulons pas entrer une série de commande à chaque fois que l'on veut utiliser le système. Nous pouvons enregistrer la configuration de l'ensemble de HAL en une seule commande :

```
halcmd : save
# components
loadrt threads name1=test-thread period1=1000000
loadrt siggen
# signals
# links
# parameter values
setp siggen.0.amplitude  5.00000e+00
setp siggen.0.frequency  1.00000e+00
```

```
setp siggen.0.offset  0.00000e+00
# realtime thread/function links
addf siggen.0.update test-thread
```

La sortie de la commande `save` est une séquence de commandes HAL. Si vous commencez par un HAL "vide" et que vous tapez toute la séquence de commandes HAL, vous aurez la configuration qui existait lors de l'exécution de la commande `save`. Pour sauver ces commandes pour une utilisation ultérieure, nous allons simplement rediriger la sortie vers un fichier :

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

7.3.6 Restaurer la configuration de HAL

Pour restaurer la configuration de HAL stockée dans `saved.hal`, nous avons besoin d'exécuter toutes ces commandes HAL. Pour ce faire, nous utiliserons la commande `-f <filename>` qui lit les commandes à partir d'un fichier, le `-I` qui affichera le prompt halcmd après l'exécution des commandes :

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

Notez qu'il n'y a pas de commande 'start' dans le fichier saved.hal. Il est nécessaire de la retaper (ou d'éditer saved.hal pour l'ajouter) :

```
halcmd : start
```

7.4 Visualiser le HAL avec halmeter

Vous pouvez construire des systèmes HAL vraiment complexes sans utiliser d'interface graphique. Mais il y a quelque chose de rassurant à visualiser le résultat du travail. Le premier, et le plus simple des outils graphiques pour le HAL, est "halmeter". C'est un programme très simple qui s'utilise comme un multimètre.

Nous allons utiliser de nouveaux éléments du composant siggen pour vérifier halmeter. Si vous avez fini l'exemple précédent, alors siggen est déjà chargé. Sinon, on peut charger tout comme nous l'avons fait précédemment :

```
emc2$ halrun
halcmd : loadrt siggen
halcmd : loadrt threads name1=test-thread period1=1000000
halcmd : addf siggen.0.update test-thread
halcmd : start
halcmd : setp siggen.0.amplitude 5
```

7.4.1 Lancement de halmeter

À ce stade, nous avons chargé le composant siggen qui est en cours d'exécution. Nous pouvons lancer halmeter. Puisque halmeter est une application graphique, X doit être actif.

```
halcmd : loadusr halmeter
```

Dans le même temps, une fenêtre s'ouvre sur votre écran, ressemblant à la figure 7.1.

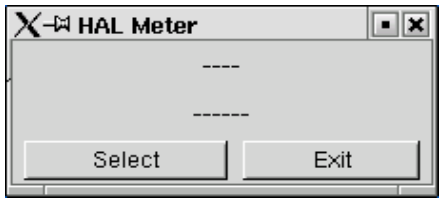


FIG. 7.1 – Lancement de Halmeter, rien n'est sélectionné

FIG. 7.2 – Dialogue de sélection de Halmeter

7.4.2 Utilisation de halmeter

La fenêtre de la figure 7.1 n'est pas d'une grande utilité, parce qu'elle n'affiche rien. Pour afficher des valeurs, cliquez sur le bouton 'Select', le dialogue de sélection des éléments à mesurer (figure 7.2).

Ce dialogue contient trois onglets. Le premier onglet affiche toutes les HAL pins du système. La seconde affiche tous les signaux et le troisième affiche tous les paramètres. Si nous voulons analyser la pin `siggen.0.triangle`, il suffit de cliquer sur elle puis sur le bouton 'OK'. Le dialogue de sélection se ferme et la mesure s'affiche, quelque chose comme sur la figure 7.3.

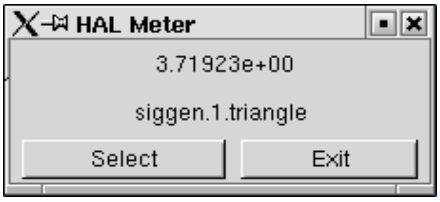


FIG. 7.3 – Affichage dans Halmeter de la valeur d'une pin

Vous devriez voir la valeur évoluer à mesure que siggen génère son signal triangulaire. Halmeter est raffraîchi environ 5 fois par seconde.

Si vous voulez visualiser rapidement des pins successives, vous pouvez cliquer le bouton 'Accept' du dialogue de sélection. Cliquez 'Select' pour ouvrir le dialoque une nouvelle fois. Mais cette fois, cliquez une autre pin, comme siggen.0.cosine, puis cliquez 'Accept'. La nouvelle valeur s'affiche alors immédiatement, mais le dialogue de sélection reste ouvert. Essayez d'afficher un paramètre au lieu d'une pin. Cliquez sur l'onglet 'Parameters', puis sélectionnez un paramètre et cliquez encore sur 'Accept'. Vous pouvez ainsi afficher très rapidement la valeur d'un élément puis d'un autre en quelques clics.

Pour arrêter halmeter, cliquez simplement sur le bouton "quitter".

Si vous voulez regarder plus d'une pin, plus d'un signal, ou plusieurs paramètres à la fois, il vous suffit de lancer plusieurs instances de halmeters. La fenêtre de halmeter a été conçue petite, pour permettre d'en avoir beaucoup à la fois sur l'écran. [3]

[3] Halmeter sera réécrit. Cette réécriture le rendra plus agréable à utiliser. La notation scientifique sera éliminée, elle est trop difficile à lire. Certaines formes d'échelles seront ajoutées (y compris, une échelle automatique) pour afficher des échelles plus grandes sans notation scientifique. Un affichage par "bargraph analogique" sera également ajouté pour donner une idée des évolutions rapides. Quand cette réécriture sera terminée, ces captures d'écran et les textes les accompagnant seront révisés pour refléter la nouvelle version.

7.5 Un exemple un peu plus complexe.

Jusqu'à maintenant, nous avons chargé un composant HAL. Mais l'idée générale de HAL est de vous permettre de charger et de relier un grand nombre de composants pour en faire un système complexe. Le prochain exemple va utiliser deux composants.

Avant de mettre en place ce nouvel exemple, nous allons commencer par un petit nettoyage. Si vous avez fini l'un des exemples précédents, il faut supprimer tous les composants et ensuite recharger la RTAPI et les librairies de HAL en faisant :

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

7.5.1 Installation des composants

Maintenant, nous allons charger le composant générateur d'impulsions. Pour l'instant, nous pouvons nous passer des détails et exécuter les commandes suivantes :[4]

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

La première commande charge deux générateurs d'impulsions, configurés pour générer des impulsions de type 0. La seconde commande charge notre vieil ami siggen et la troisième crée deux threads, un rapide (fast) avec une période de 50µs et un lent avec une période de 1ms. Le thread rapide ne prend pas en charge les fonctions à virgule flottante (fp1=0).

Comme précédemment, on peut utiliser `halcmd show` pour jeter un coup d'oeil au HAL. Cette fois, nous aurons beaucoup plus de pins et de paramètres que précédemment :

```
halcmd : show pin
Component Pins :
Owner  Type  Dir    Value      Name
 03    float -W   0.00000e+00  siggen.0.cosine
 03    float -W   0.00000e+00  siggen.0.sawtooth
 03    float -W   0.00000e+00  siggen.0.sine
 03    float -W   0.00000e+00  siggen.0.square
 03    float -W   0.00000e+00  siggen.0.triangle
 02    s32   -W            0   freqgen.0.counts
 02    bit   -W     FALSE      freqgen.0.dir
 02    float -W   0.00000e+00  freqgen.0.position
 02    bit   -W     FALSE      freqgen.0.step
 02    float R-   0.00000e+00  freqgen.0.velocity
 02    s32   -W            0   freqgen.1.counts
 02    bit   -W     FALSE      freqgen.1.dir
 02    float -W   0.00000e+00  freqgen.1.position
 02    bit   -W     FALSE      freqgen.1.step
 02    float R-   0.00000e+00  freqgen.1.velocity
halcmd : show param
Parameters :
Owner  Type  Dir    Value      Name
 03    float  -W   1.00000e+00  siggen.0.amplitude
 03    float  -W   1.00000e+00  siggen.0.frequency
```

[4] Le signe "\" à la fin d'une longue ligne indique que la ligne est tronquée (c'est nécessaire pour formater ce document). Quand vous entrez la commande en ligne dans la console, sautez simplement le "\" (ne pressez pas Entrée) et continuez à taper la ligne suivante.

```
03    float  -W   0.00000e+00  siggen.0.offset
02    u32    -W     000000001  freqgen.0.dirhold
02    u32    -W     000000001  freqgen.0.dirsetup
02    float  R-   0.00000e+00  freqgen.0.frequency
02    float  -W   0.00000e+00  freqgen.0.maxaccel
02    float  -W   1.00000e+15  freqgen.0.maxfreq
02    float  -W   1.00000e+00  freqgen.0.position-scale
02    s32    R-             0  freqgen.0.rawcounts
02    u32    -W     000000001  freqgen.0.steplen
02    u32    -W     000000001  freqgen.0.stepspace
02    float  -W   1.00000e+00  freqgen.0.velocity-scale
02    u32    -W     000000001  freqgen.1.dirhold
02    u32    -W     000000001  freqgen.1.dirsetup
02    float  R-   0.00000e+00  freqgen.1.frequency
02    float  -W   0.00000e+00  freqgen.1.maxaccel
02    float  -W   1.00000e+15  freqgen.1.maxfreq
02    float  -W   1.00000e+00  freqgen.1.position-scale
02    s32    R-             0  freqgen.1.rawcounts
02    u32    -W     000000001  freqgen.1.steplen
02    u32    -W     000000001  freqgen.1.stepspace
02    float  -W   1.00000e+00  freqgen.1.velocity-scale
```

7.5.2 Connecter les pins avec des signaux

Nous avons donc deux générateurs d'impulsions carrées et un générateur de signaux. Maintenant, nous allons créer des signaux HAL pour connecter ces trois composants. Nous allons faire comme si nous pilotions les axes X et Y d'une machine avec nos générateurs d'impulsions. Nous voulons déplacer la table en ronds. Pour ce faire, nous allons envoyer un signal cosinusoïdal à l'axe des X et un signal sinusoïdal à l'axe des Y. Le module siggen créera le sinus et le cosinus, mais nous aurons besoin de "fils" pour connecter les modules ensemble. Dans le HAL, les "fils" sont appelés signaux. Nous devons en créer deux. Nous pouvons les appeler comme on veut, pour cet exemple il y aura `X_vel` et `Y_vel`. Le signal `X_vel` partira de la sortie cosinus du générateur de signal et arrivera sur l'entrée "velocity" du premier générateur d'impulsions. La première étape consiste à connecter le signal à la sortie du générateur de signaux. Pour connecter un signal à une pin, nous utilisons la commande `net` :

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

Pour voir l'effet de la commande `net`, regardons les signaux :

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

Quand un signal est connecté à une ou plusieurs pins, la commande `show` liste les pins immédiatement suivies par nom du signal. Les flèches montrent la direction du flux de données, dans ce cas, les flux vont de la pin `siggen.0.cosine` pour le signal `X_vel`. Maintenant, connectons `X_vel` à l'entrée "velocity" du générateur d'impulsions carrées :

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

On peut aussi connecter l'axe Y au signal `Y_vel`. Il doit partir de la sortie sinus du générateur de signal pour arriver sur l'entrée du second générateur d'impulsions carrées. La commande suivante fait en une ligne, la même chose que les deux commandes `net` précédentes ont fait pour `X_vel` :

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

Regardons une dernière fois les signaux et les pins connectés ensembles :

```
halcmd : show sig
Signals :
Type      Value      Name
float   0.00000e+00  X_vel
                         <== siggen.0.cosine
                         ==> freqgen.0.velocity
float   0.00000e+00  Y_vel
                         <== siggen.0.sine
                         ==> freqgen.1.velocity
```

Avec la commande `show sig` on voit clairement comment les flux de données traversent le HAL. Par exemple, le signal `X_vel` part de la pin `siggen.0.cosine` et arrive sur la pin `freqgen.0.velocity`.

7.5.3 Exécuter les réglages du temps réel - threads et functions

Penser à ce qui circule dans les “fils” rend les pins et les signaux assez faciles à comprendre. Les threads et les fonctions sont un peu plus difficiles. Les fonctions contiennent des instructions pour l'ordinateur. Les threads sont les méthodes utilisées pour faire exécuter ces instructions quand c'est nécessaire. Premièrement, regardons les fonctions dont nous disposons :

```
halcmd : show funct
Exported Functions :
Owner CodeAddr   Arg    FP  Users  Name
 03   D89051C4 D88F10FC YES    0   siggen.0.update
 02   D8902868 D88F1054 YES    0   freqgen.capture_position
 02   D8902498 D88F1054 NO     0   freqgen.make_pulses
 02   D89026F0 D88F1054 YES    0   freqgen.update_freq
```

En règle générale, vous devez vous référer à la documentation de chaque composant pour voir ce que font ses fonctions. Dans notre exemple, la fonction `siggen.0.update` est utilisée pour mettre à jour les sorties du générateur de signaux. Chaque fois qu'elle est exécutée, le générateur recalcule les valeurs de ses sorties sinus, cosinus, triangle, carrée. Pour générer un signal régulier, il doit fonctionner à des intervalles bien spécifiques.

Les trois autres fonctions sont liées au générateur d'impulsions :

La première, `freqgen.capture_position`, est utilisée pour un retour de position. Elle capture la valeur d'un compteur interne qui compte les impulsions qui sont générées. S'il n'y a pas de perte de pas, ce compteur indique la position du moteur.

La fonction principale du générateur d'impulsions est `freqgen.make_pulses`. Chaque fois que `make_pulses` démarre elle décide qu'il est temps de faire un pas, si oui il fixe les sorties en conséquence. Pour des pas plus doux, il doit fonctionner le plus souvent possible. Parce qu'il a besoin de fonctionner de manière rapide, `make_pulses` est hautement optimisé et n'effectue que quelques calculs. Contrairement aux autres, il n'a pas besoin de virgule flottante pour ses calculs.

La dernière fonction, `freqgen.update_freq`, est responsable de l'échelle et de quelques autres calculs qui ne doivent être effectués que lors d'une commande de changement de fréquence.

Pour notre exemple nous voulons faire tourner `siggen.0.update` avec une vitesse de calcul des valeurs sinus et cosinus modérée. Immédiatement après, nous lançons `siggen.0.update`. Nous voulons lancer `freqgen.update_freq` pour charger les nouvelles valeurs dans le générateur d'impulsions. Finalement nous lancerons `freqgen.make_pulses` aussi vite que possible pour des pas plus doux. Comme nous n'utilisons pas de retour de position, nous n'avons pas besoin de lancer `freqgen.capture_position`.

Nous lançons les fonctions en les ajoutant aux threads. Chaque thread va à une vitesse spécifique. Regardons de quels threads nous disposons :

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

Les deux `threads` ont été créés lorsque nous les avons chargés. Le premier, `slow`, tourne toutes les millisecondes, il est capable d'exécuter des fonctions en virgule flottante (FP). Nous l'utilisons pour `siggen.0.update` et `freqgen.update_freq`. Le deuxième thread est `fast`, il tourne toutes les 50 microsecondes, il ne prend pas en charge les calculs en virgule flottante. Nous l'utilisons pour `freqgen.make_pulses`. Pour connecter des fonctions au bon thread, nous utilisons la commande `addf`. Nous spécifions la fonction en premier suivie par le thread :

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

Après avoir lancé ces commandes, nous pouvons lancer la commande `show thread` une nouvelle fois pour voir ce qui ce passe :

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

Maintenant, chaque thread est suivi par les noms des fonctions, dans l'ordre dans lequel les fonctions seront exécutées.

7.5.4 Réglage des paramètres

Nous sommes presque prêts à démarrer notre système HAL. Mais il faut auparavant régler quelques paramètres. Par défaut le composant siggen génère des signaux qui varient entre +1 et -1. Pour notre exemple, c'est très bien, nous voulons que la vitesse de la table varie de +1 à -1 pouce par seconde. Toutefois, l'échelle du générateur d'impulsions de pas n'est pas bonne. Par défaut, il génère une fréquence de sortie de 1 pas par seconde avec une capacité de 1000. Il est fort improbable qu'un pas par seconde nous donne une vitesse de déplacement de la table d'un pouce par seconde. Supposons que notre vis fasse 5 tours par pouce, couplée à un moteur pas à pas de 200 pas par tour et une interface qui fournit 10x micropas par pas. Il faut donc 2000 pas pour faire un tour de vis et 5 tours pour faire un pouce. Ce qui signifie que notre montage utilisera 10000 pas par pouce. Nous avons besoin de multiplier la vitesse d'entrée à l'étape générateur d'impulsions par 10000 pour obtenir la bonne valeur. C'est exactement pour cela qu'existe le paramètre `freqgen.n.velocity-scale`. Dans notre cas, les axes X et Y ont la même échelle et nous pouvons passer les deux paramètres à 10000 :

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

Cela signifie que, avec la pin `freqgen.0.velocity` à 1.000 et le générateur réglé pour 10000 impulsions par seconde (10kHz), avec le moteur et la vis décrits précédemment, nos axes auront une vitesse de déplacement de exactement 1.000 pouce par seconde. Cela illustre une notion clé du concept de HAL, des éléments comme les échelles étant au plus bas niveau possible, dans notre exemple le générateur d'impulsions de pas, le signal interne `X_vel` est celui de la vitesse de déplacement de la table en pouces par seconde. Les autres composants comme `siggen` ne savent rien du tout à propos de l'échelle des autres. Si on change de vis, ou de moteur, il n'y a qu'un seul paramètre à changer, l'échelle du générateur d'impulsions de pas.

7.5.5 Lançons le !

Nous avons maintenant tout configuré et sommes prêts à démarrer. Tout comme dans le premier exemple, nous utilisons la commande `start` :

```
halcmd : start
```

Bien que rien ne semble se produire, à l'intérieur de l'ordinateur les impulsions de pas sont présentes sur la sortie du générateur, variant entre 10kHz dans un sens et 10kHz dans l'autre à chaque seconde. Dans la suite de ce tutoriel, nous allons voir comment convertir ces signaux internes des moteurs dans le monde réel, mais nous allons d'abord les examiner pour voir ce qui se passe.

7.6 Voyons-y de plus près avec halscope.

L'exemple précédent génère certains signaux très intéressants. Mais beaucoup de ce qui se passe est beaucoup trop rapide pour être vu avec halmeter. Pour examiner de plus près ce qui se passe à l'intérieur de HAL, il faudrait un oscilloscope. Heureusement HAL en a un, appelé halscope.

7.6.1 Démarrer Halscope

Halscope comporte deux parties, une partie en temps réel qui est chargée comme un module de noyau et une partie utilisateur qui fournit l'interface graphique et l'affichage. Cependant, vous n'avez pas à vous inquiéter à ce sujet car l'interface demandera automatiquement que la partie temps réel soit chargée :

```
halcmd : loadusr halscope
```

La fenêtre graphique du scope s'ouvre, immédiatement suivie par un dialogue "Realtime function not linked" visible sur la figure 7.4[5].

Ce dialogue est l'endroit où vous définissez le taux d'échantillonnage de l'oscilloscope. Pour le moment nous voulons un échantillon par milliseconde, alors cliquez sur le thread "slow" 1.03mSec et laissez le multiplicateur à 1. Nous allons aussi passer la longueur d'enregistrement à 4047 samples (échantillons), de sorte que nous puissions utiliser jusqu'à 4 canaux simultanément. Quand vous sélectionnez un thread puis que vous cliquez sur le bouton "OK", le dialogue disparaît et la fenêtre du scope affiche quelque chose comme sur la figure 7.5.

[5]Plusieurs de ces captures d'écran font référence à des threads nommés "siggen.thread" et "stepgen.thread" au lieu de "slow" et "fast". Lorsque les captures d'écran ont été faites, les composants thread n'existaient pas et différentes méthodes étaient utilisées pour créer des threads en leur donnant des noms différents. Aussi, les captures d'écran montrent des , etc, comme "stepgen.xxx" au lieux de "freqgen.xxx". Le nom original du module freqgen était stepgen et je n'ai pas eux le temps depuis pour refaire toutes les captures avec les nouveaux noms. Le nom "stepgen" actuel fait référence à un générateur d'impulsions de pas différent, un qui accepte les commandes de position au lieu de celles de vitesse. Les deux sont décrit en détail plus loin dans ce document.

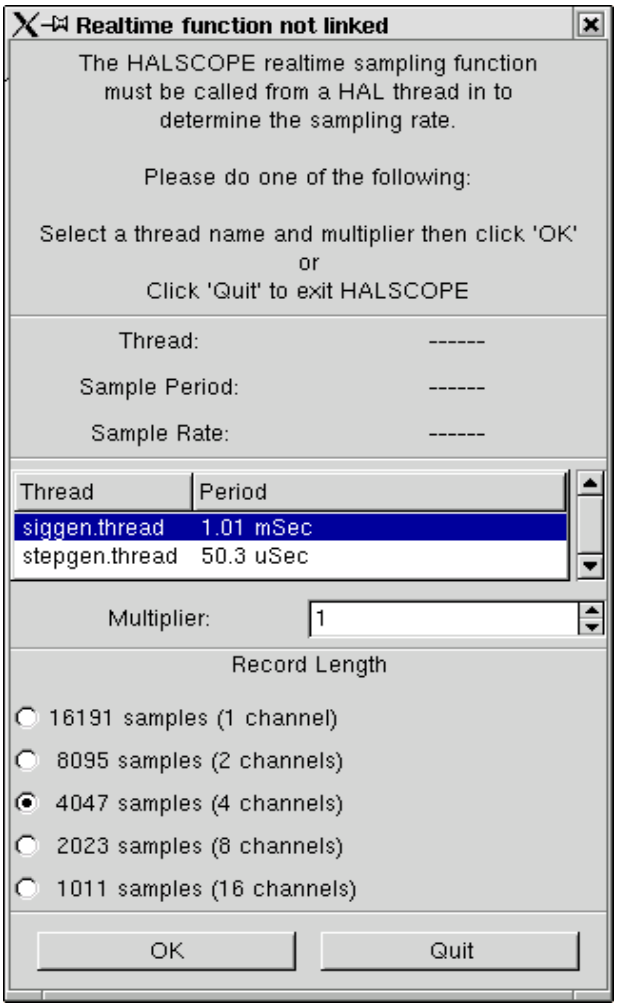


FIG. 7.4 – Dialogue“Realtime function not linked”

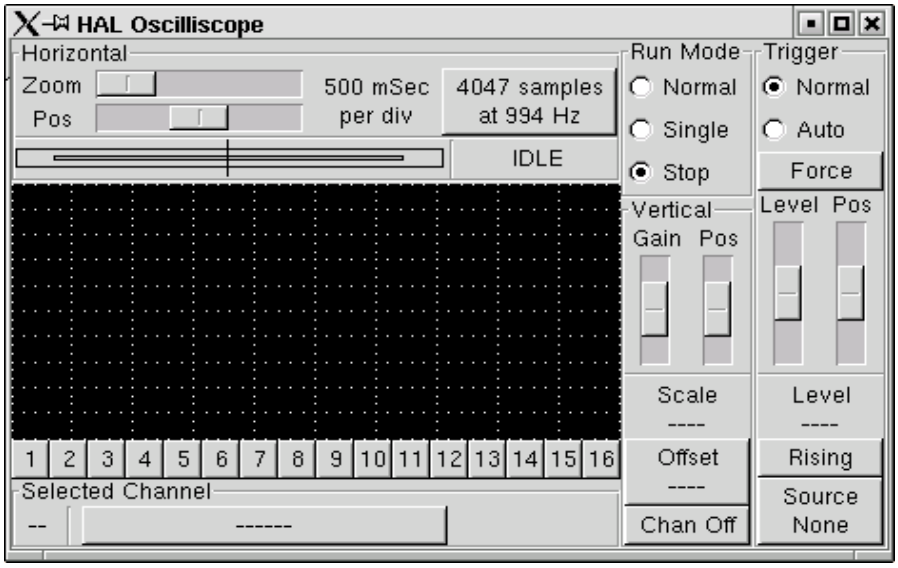


FIG. 7.5 – Fenêtre initiale du scope

7.6.2 Branchement des “sondes du scope”

À ce stade, Halscope est prêt à l’emploi. Nous avons déjà choisi le taux d’échantillonnage et la longueur d’enregistrement, de sorte que la prochaine étape consiste à décider de ce qu’il faut mesurer. C’est équivalent à brancher les “sondes virtuelles du scope” à HAL. Halscope dispose de 16 canaux, mais le nombre que vous pouvez utiliser à un moment donné dépend de la longueur d’enregistrement, plus il y a de canaux, plus les enregistrements doivent être courts, car la mémoire disponible pour l’enregistrement est fixée à environ 16000 échantillons.

Les boutons des canaux se situent en dessous de l’écran du scope. Cliquez le boutton “1” et vous verrez apparaître le dialogue de sélection “Select Channel Source”, comme sur la figure 7.6. Ce dialogue est très similaire à celui utilisé par Halmeter. Nous aimerions bien regarder les signaux que nous avons défini précédemment, pour cela, cliquons sur l’onglet “Signaux” et le dialogue affichera tous les signaux existants dans le HAL (seulement deux dans notre exemple).

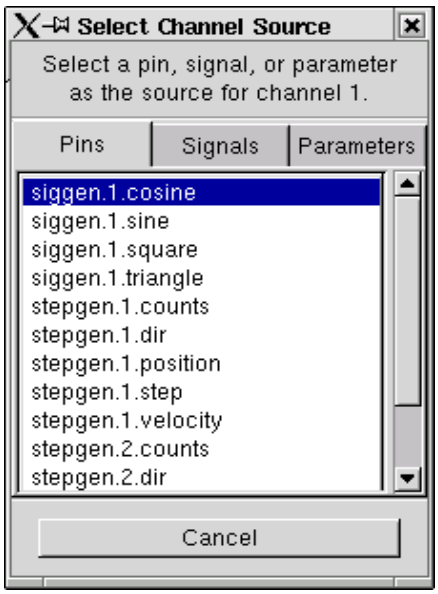


FIG. 7.6 – Dialogue de sélection du scope

Pour choisir un signal, il suffit de cliquer dessus. Dans notre cas, nous voulons utiliser le canal 1 pour afficher le signal “ X_vel”. Lorsque l’on clique sur “ X_vel”, la fenêtre se ferme et le canal a été sélectionné. Le bouton du canal 1 est pressé et le numéro du canal 1 et le nom “ X_vel” apparaissent sous la rangée de boutons. L’affichage indique toujours le canal sélectionné, vous pouvez avoir beaucoup de canaux sur l’écran, mais celui qui est actif sera en surbrillance. Les différents contrôles comme la position verticale et l’amplitude sont toujours relatifs au canal 1. Pour ajouter un signal au canal 2, cliquez sur le bouton "2”. Dans la fenêtre de dialogue, cliquez sur l’onglet "Signals”, puis cliquez sur “ Y_vel”.

Nous voulons aussi voir les signaux carrés et triangles produits. Il n’existe pas de signaux connectés à ces pins, nous utilisons donc l’onglet “Pins”. Pour le canal 3, sélectionnez “ siggen.0.triangle” et pour le canal 4, choisissez “ siggen.0.square”.

7.6.3 Capturer notre première forme d'onde

Maintenant que nous avons plusieurs sondes branchées sur HAL, il est temps de capturer quelques formes d'ondes. Pour démarrer le scope, cochez la case "Normal" du groupe "run mode" (en haut à droite). Puisque nous avons une longueur d'enregistrement de 4000 échantillons et une acquisition de 1000 échantillons par seconde, il faudra à halscope environ 2 secondes pour remplir la moitié de son tampon. Pendant ce temps, une barre de progression juste au-dessus de l'écran principal affichera le remplissage du tampon. Une fois que le tampon est à moitié plein, scope attend un déclencheur. Puisque nous n'en avons pas encore configuré, il attendra toujours. Pour déclencher manuellement, cliquez sur le bouton "Force" du groupe " Trigger" en haut à droite. Vous devriez voir le reste de la zone tampon se remplir, puis l'écran afficher les ondes capturées. Le résultat ressemble à la figure 7.7.

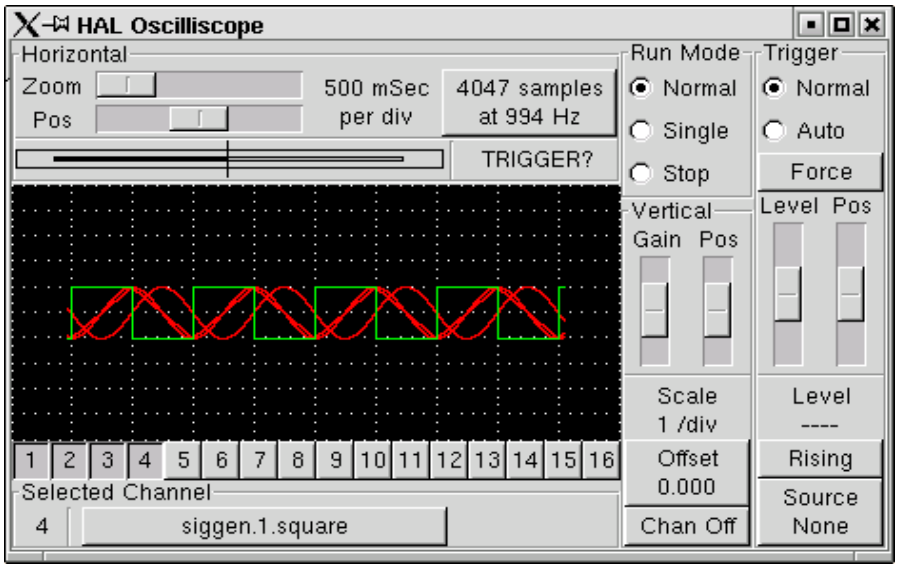


FIG. 7.7 – Capture d'ondes

Le grand bouton "Selected Channel" en bas, indique que le canal 4 est actuellement sélectionné, donc, qu'il correspond à l'onde verte, celle de la mesurée sur la pin "siggen.1.square". Essayez de cliquer sur les autres canaux pour mettre leurs traces en évidence.

7.6.4 Ajustement vertical

Les traces sont assez difficiles à distinguer car toutes les quatre sont les unes sur les autres. Pour résoudre ce problème, nous utilisons le curseur “Vertical” situé à droite de l’écran. Ces contrôles agissent sur le canal actuellement sélectionné. En ajustant le gain, notez qu’il couvre une large échelle (contrairement aux scopes réels), celle-ci permet d’afficher des signaux très petits (pico unités) à très grands (Tera - unités). Le curseur “position” déplace la trace affichée de haut en bas sur toute la hauteur de l’écran. Pour de plus grands ajustements le bouton Offset doit être utilisé (voir les références halscope dans la section 10.3 pour plus de détails).

7.6.5 Triggering

L’utilisation du bouton “Force” n’est parfois pas satisfaisante pour déclencher le scope. Pour régler un déclenchement (triggering) réel, cliquez sur le bouton “Source” situé en bas à droite. Il ouvre alors le dialogue “Trigger Source”, qui est simplement la liste de toutes les sondes actuellement branchées (Figure 7.8). Sélectionnez la sonde à utiliser pour déclencher en cliquant dessus. Pour notre exemple nous utilisons 3 canaux, essayez l’onde triangle.

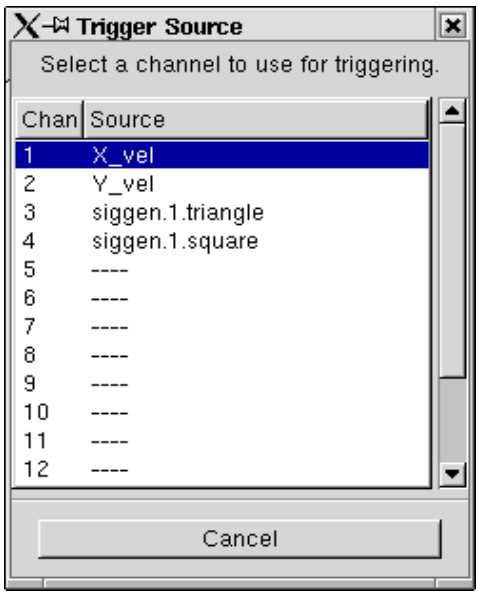


FIG. 7.8 – Dialogue des sources de déclenchement

Après avoir défini les sources de déclenchement, vous pouvez ajuster le niveau de déclenchement avec les curseurs dans la boîte “ Trigger” le long du bord droit. Le niveau peut être modifié à partir du haut vers le bas de l’écran, et est affiché sous les curseurs. La position est l’emplacement du point de déclenchement dans l’enregistrement complet. Avec le curseur tout en bas, le point de déclenchement est à la fin de l’enregistrement, et halscope affiche ce qui s’est passé avant le point de déclenchement. Lorsque le curseur est tout en haut, le point de déclenchement est au début de l’enregistrement, l’affichage représente ce qui s’est passé après le point de déclenchement. Le point de déclenchement est visible comme une ligne verticale dans la barre de progression située juste au dessus de l’écran. La polarité du signal de déclenchement peut être inversée en cliquant sur le bouton situé juste sous l’affichage du niveau de déclenchement. Notez que la modification de la position de déclenchement arrête le scope une fois la position ajustée, vous relancez le scope en cliquant sur le bouton “Normal” du groupe “Run Mode”.

Maintenant que nous avons réglé la position verticale et le déclenchement, l’écran doit ressembler à la figure 7.9.

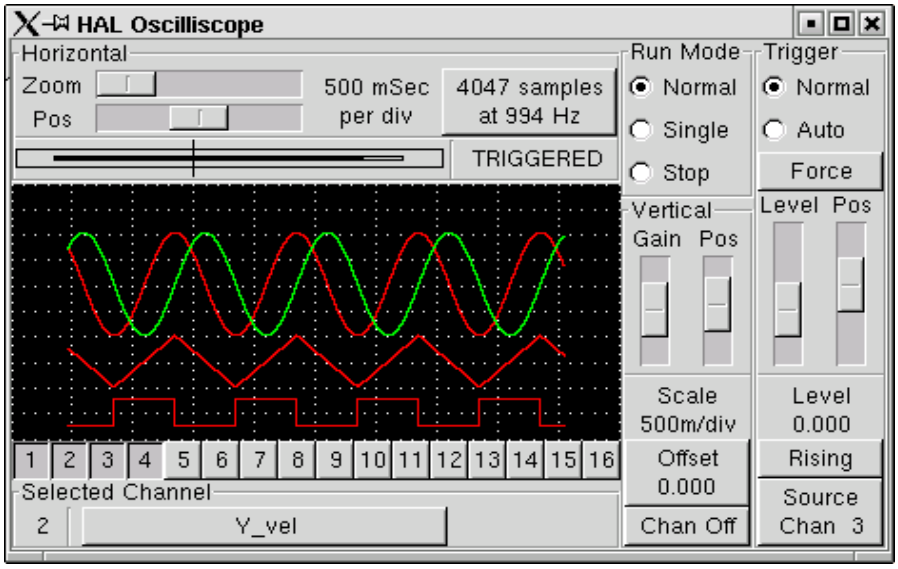


FIG. 7.9 – Formes d'ondes avec déclenchement

7.6.6 Ajustement horizontal

Pour examiner de près une partie d'une forme d'onde, vous pouvez utiliser le zoom au dessus de l'écran pour étendre la trace horizontalement et le curseur de position pour déterminer quelle partie de l'onde zoomée est visible. Parfois simplement élargir l'onde n'est pas suffisant et il faut augmenter la fréquence d'échantillonnage. Par exemple, nous aimerions voir les impulsions de pas qui sont générés dans notre exemple. Mais les impulsions de pas font seulement 50µs de long, l'échantillonnage à 1kHz n'est pas assez rapide. Pour changer le taux d'échantillonnage, cliquez sur le bouton qui affiche la longueur de l'enregistrement et l'échantillonnage pour avoir le dialogue "Select Sample Rate", figure 7.10. Pour notre exemple, nous cliquerons sur le thread à 50uS, "fast", qui fournira un échantillonnage à environ 20KHz. Maintenant au lieu d'afficher environ 4 secondes de données, un enregistrement est de 4000 échantillons à 20KHz, soit environ 0.20 seconde.

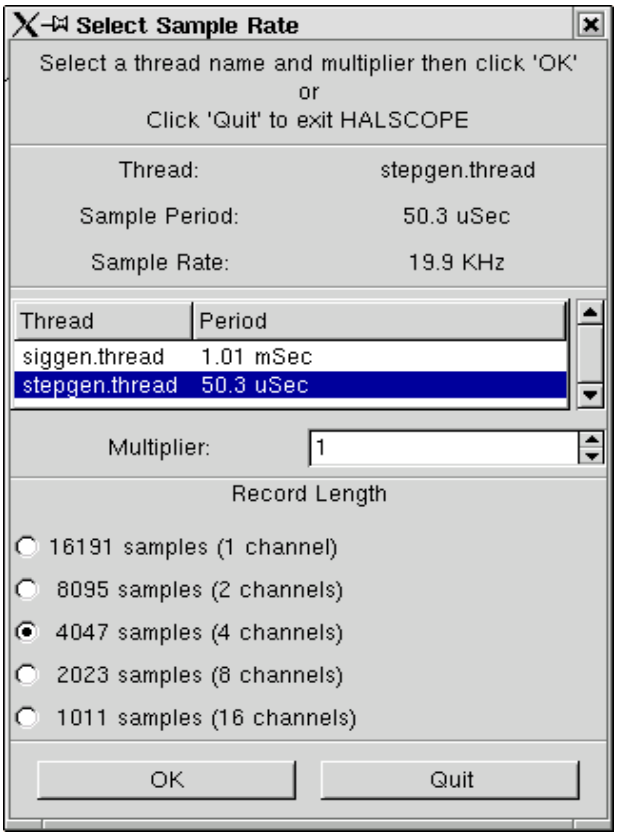


FIG. 7.10 – Dialogue de choix d'échantillonnage

7.6.7 Plus de canaux

Maintenant regardons les impulsions de pas. Halscope dispose de 16 canaux, mais pour cet exemple, nous en utilisons seulement 4 à la fois. Avant de sélectionner tout autre canal, nous avons besoin d'en éteindre certains. Cliquez sur le canal 2, puis sur le bouton "Off" sous le groupe " vertical". Ensuite, cliquez sur le canal 3, mettez le off et faites de même pour le canal 4. Même si les circuits sont éteints, ils ont encore en mémoire ce à quoi ils sont connectés et en fait, nous continuerons d'utiliser le canal 3 comme source de déclenchement. Pour ajouter de nouveaux canaux, sélectionnez le canal 5, choisissez la pin " stepgen.1.dir", puis canal 6 et sélectionnez "stepgen.1.step". Ensuite, cliquez sur "mode Normal" pour lancer le scope, ajustez le zoom horizontal à 5ms par division. Vous devriez voir les impulsions de pas ralentir à la vitesse commandée (channel 1) approcher de zéro, puis la pin de direction changer d'état et les impulsions de pas reprendre de nouveau de la vitesse. Vous aurez peut être besoin d'augmenter le gain sur le canal 1 à environ 20ms par division afin de mieux voir l'évolution de la vitesse de commande. Le résultat devrait être proche de celui de la figure 7.11.

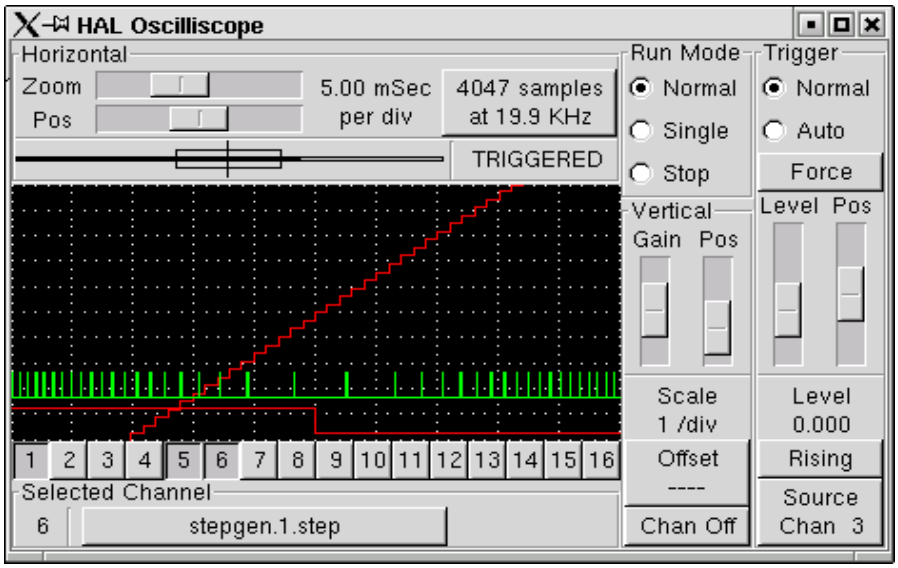


FIG. 7.11 – Observer les impulsions de pas

7.6.8 Plus d'échantillons

Si vous souhaitez enregistrer plus d'échantillons à la fois, redémarrez le temps réel et chargez halscope avec un argument numérique qui indique le nombre d'échantillons que vous voulez capturer, comme :

```
halcmd : loadusr halscope 80000
```

Si le composant `scope_rt` n'est pas déjà chargé, halscope va le charger et lui demander un total de 80000 échantillons, de sorte que lorsque l'échantillonnage se fera sur 4 canaux à la fois, il y aura 20000 échantillons par canal. (Si `scope_rt` est déjà chargé, l'argument numérique passé à halscope sera sans effet)

Chapitre 8

Informations générales

8.1 Notation

8.1.1 Conventions typographiques

Les lignes de commandes sont représentées en police **`bold typewriter`**. Les réponses de l'ordinateur sont en police `typewriter`. Depuis début 2006, plus aucune commande ne nécessite les privilèges du root, de sorte que tous les exemples seront précédés par le prompt utilisateur normal, `$`. Le texte entre crochets est un texte optionnel `[comme-cela]`. Le texte entre crochets `<comme-cela>` représente un champ qui peut prendre différentes valeurs, le paragraphe suivant explique les valeurs appropriées. Les items de texte séparés par une barre verticale signifie que l'un ou l'autre, mais pas plus, doit être présent. Toutes les lignes de commandes des exemples supposent que vous êtes dans le répertoire `d'emc2/` ou vous avez configuré ou compilé emc2 avec l'option –run-in-place. Les chemins seront par conséquent, affichés en accord avec cet emplacement.

8.1.2 Noms

Toutes les entités de HAL sont accessibles et manipulées par leurs noms, donc, documenter les noms des pins, signaux, paramètres, etc, est très important. Les noms dans HAL ont un maximum de 41 caractères de long (comme défini par HAL_NAME_LEN dans hal.h). De nombreux noms seront présentés dans la forme générale, avec un texte entre crochets `<comme-cela>` représentant les champs de valeurs diverses.

Quand les pins, signaux, ou paramètres sont décrits pour la première fois, leur nom sera précédé par leur type en (PETITES CAPITALES) et suivi par une brève description. Les définitions typiques de pins ressemblent à ces exemples :

– `(BIT) parport.<portnum>.pin-<pinnum>-in` – La HAL pin associée avec la broche physique d'entrée `<pinnum>` du connecteur db25.
– `(FLOAT) pid.<loopnum>.output` – La sortie de la boucle PID.

De temps en temps, une version abrégée du nom peut être utilisée, par exemple la deuxième pin ci-dessus pourrait être appelée simplement `avec .output` quand cela peut être fait sans prêter à confusion.

8.2 Conventions générales de nommage

Le but des conventions de nommage est de rendre l'utilisation de HAL plus facile. Par exemple, si plusieurs interfaces de codeur fournissent le même jeu de pins et qu'elles sont nommées de la même

façon, il serait facile de changer l'interface d'un codeur à un autre. Malheureusement, comme tout projet open-source, HAL est la combinaison de choses diversement conçues et comme les choses simples évoluent. Il en résulte de nombreuses incohérences. Cette section vise à remédier à ce problème en définissant certaines conventions, mais il faudra certainement un certain temps avant que tous les modules soient convertis pour les suivre.

Halcmd et d'autres utilitaires HAL de bas niveau, traitent les noms HAL comme de simples entités, sans structure. Toutefois, la plupart des modules ont une certaine structure implicite. Par exemple, une carte fournit plusieurs blocs fonctionnels, chaque bloc peut avoir plusieurs canaux et chaque canal, une ou plusieurs broches. La structure qui en résulte ressemble à une arborescence de répertoires. Même si halcmd ne reconnait pas la structure arborescente, la convention de nommage est un bon choix, elles lui permettra de regrouper ensemble, les items du même groupe, car il trie les noms. En outre, les outils de haut niveau peuvent être conçus pour reconnaitre de telles structures si les noms fournissent les informations nécessaires. Pour cela, tous les modules de HAL devraient suivrent les règles suivantes :

– Les points (".") séparent les niveaux hiérarchiques. C'est analogue à la barre de fraction ("/") dans les noms de fichiers.
– Le tiret ("-") sépare les mots ou les champs dans la même hiérarchie.
– Les modules HAL ne doivent pas utiliser le caractère souligné ou les casses mélangées. [1]
– Utiliser seulement des caractères minuscules, lettres et chiffres.

8.3 Conventions de nommage des pilotes de matériels[2]

8.3.1 Noms de pin/paramètre

Les pilotes matériels devraient utiliser cinq champs (sur trois niveaux) pour obtenir un nom de pin ou de paramètre, comme le suivant :

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

Les champs individuels sont :

<device-name> Le matériel avec lequel le pilote est sensé travailler. Il s'agit le plus souvent d'une carte d'interface d'un certain type, mais il existe d'autres possibilités.

<device-num> Il est possible d'installer plusieurs cartes servo, ports parallèles ou autre périphérique matériel dans un ordinateur. Le numéro du périphérique identifie un périphérique spécifique. Les numéros de périphériques commencent à 0 et s'incrémentent.[3]

<io-type> La plupart des périphériques fournissent plus d'un type d'I/O. Même le simple port parallèle a, à la fois plusieurs entrées et plusieurs sorties numériques. Les cartes commerciales plus complexes peuvent avoir des entrées et des sorties numériques, des compteurs de codeurs, des générateurs d'impulsions de pas ou de PWM, des convertisseurs numérique/analogique, analogique/numérique et d'autres possibilités plus spécifiques. Le "I/O type" est utilisé pour identifier le type d'I/O avec lequel la pin ou le paramètre est associé. Idéalement, les pilotes qui implémentent les mêmes type d'I/O, même sur des dispositifs très différents, devraient fournir un jeu de pins et de paramètres cohérents et de comportements identiques. Par exemple, toutes les entrées numériques doivent se comporter de la même manière quand elles sont vues de l'intérieur de HAL, indépendamment du périphérique.

[1] Les caractères souslignés ont été enlevés, mais il reste quelques cas de mélange de casses, par exemple "pid.0.Pgain" au lieux de "pid.0.p-gain".

[2] La plupart des pilotes ne suivent pas ces conventions dans la version 2.0. Ce chapitre est réellement un guide pour les développements futurs.

[3] Certains matériels utilisent des cavaliers ou d'autres dispositifs pour définir une identification spécifique à chacun. Idéalement, le pilote fournit une manière à l'utilisateur de dire, le "device-num 0 est spécifique au périphérique qui a l'ID XXX", ses sous-ensembles porterons tous un numéro commençant par 0. Mais à l'heure actuelle, certains pilotes utilisent l'ID directement comme numéro de périphérique. Ce qui signifie qu'il est possible d'avoir un périphérique Numéro 2, sans en avoir en Numéro 0. C'est un bug qui devrait disparaître en version 2.1.

<chan-num> Quasiment tous les périphériques d'I/O ont plusieurs canaux, le numéro de canal "chan-num" identifie un de ceux ci. Comme les numéros de périphériques "device-num", les numéros de canaux, "chan-num", commencent à zéro et s'incrémentent.[4] Si plusieurs périphériques sont installés, les numéro de canaux des périphériques supplémentaires recommencent à zéro. Comme il est possible d'avoir un numéro de canal supérieur à 9, les numéros de canaux doivent avoir deux chiffres, avec un zéro en tête pour les nombres inférieur à 10 pour préserver l'ordre des tris. Certains modules ont des pins et/ou des paramètres qui affectent plusieurs canaux. Par exemple un générateur de PWM peut avoir quatre canaux avec quatre entrées "duty-cycle" indépendantes, mais un seul paramètre "frequency" qui contrôle les quatres canaux (à cause de limitations matérielles). Le paramètre "frequency" doit utiliser les numéros de canaux de "00-03".

<specific-name> Un canal individuel d'I/O peut avoir une seule HAL pin associée avec lui, mais la plupart en ont plus. Par exemple, une entrée numérique a deux pins, une qui est l'état de la broche physique, l'autre qui est la même chose mais inversée. Cela permet au configurateur de choisir entre les deux états de l'entrée, active haute ou active basse. Pour la plupart des types d' entrée/sortie, il existe un jeu standard de broches et de paramètres, (appelé l'"interface canonique") que le pilote doit implémenter. Les interfaces canoniques sont décrites au chapitre 9.

8.3.1.1 Exemples

motenc.0.encoder.2.position – la sortie position du troisième canal codeur sur la première carte Motenc.

stg.0.din.03.in – l'état de la quatrième entrée numérique sur la première carte Servo-to-Go.

ppmc.0.pwm.00-03.frequency – la fréquence porteuse utilisée sur les canaux PWM de 0 à 3.

8.3.2 Noms des fonctions

Les pilotes matériels ont généralement seulement deux types de fonctions HAL, une qui lit l'état du matériel et met à jour les pins HAL, l'autre qui écrit sur le matériel en utilisant les données fournies sur les pins HAL. Ce qui devrait être nommé de la façon suivante :

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

<device-name> Le même que celui utilisé pour les pins et les paramètres.

<device-num> Le périphérique spécifique auquel la fonction aura accès.

<io-type> Optionnel. Une fonction peut accéder à toutes les d'entrées/sorties d'une carte ou, elle peut accéder seulement à un certain type. Par exemple, il peut y avoir des fonctions indépendantes pour lire les compteurs de codeurs et lire les entrées/sorties numériques. Si de telles fonctions indépendantes existent, le champ <io-type> identifie le type d'I/O auquelles elles auront accès. Si une simple fonction lit toutes les entrés/sorties fournies par la carte, <io-type> n'est pas utilisé.[5]

<chan-num-range> Optionnel. Utilisé seulement si l'entrée/sortie <io-type> est cassée dans des groupes et est accédée par différentes fonctions.

read|write Indique si la fonction lit le matériel ou lui écrit.

[4]Une exception à la règle du "numéro de canal commençant à zéro" est le port parallèle. Ses "HAL pins" sont numérotées avec le numéro de la broche correspondante du connecteur DB-25. C'est plus pratique pour le câblage, mais non cohérent avec les autres pilotes. Il y a débat pour savoir si c'est un bogue ou une fonctionnalité.

[5]Note aux programmeurs de pilotes : ne PAS implémenter des fonctions séparées pour différents types d'I/O à moins qu'elles ne soient interruptibles et puissent marcher dans des threads indépendants. Si l'interruption de la lecture d'un codeur pour lire des entrées numériques, puis reprendre la lecture du codeur peut poser problème, alors implémentez une fonction unique qui fera tout.

8.3.2.1 Exemples

motenc.0.encoder.read – lit tous les codeurs sur la première carte motenc.

generic8255.0.din.09-15.read – lit le deuxième port 8 bits sur la première carte d'entrées/sorties à base de 8255.

ppmc.0.write – écrit toutes les sorties (générateur de pas, pwm, DAC et ADC) sur la première carte ppmc.

Chapitre 9

Périphériques d'interfaces canoniques[1]

Les sections qui suivent expliquent les pins, paramètres et functions qui sont fournies par les "périphériques canoniques". Tous les pilotes de périphériques HAL devraient fournir les mêmes pins et paramètres et implémenter les mêmes comportements.

Noter que seuls les champs `<io-type>` et `<specific-name>` sont définis pour un périphérique canonique. Les champs `<device-name>`, `<device-num>` et `<chan-num>` sont définis en fonction des caractéristiques du périphérique réel.

9.1 Entrée numérique (Digital Input)

L'entrée numérique canonique (I/O type : **digin**) est assez simple.

9.1.1 Pins

– (BIT) **in** – état de l'entrée matérielle.
– (BIT) **in-not** – état inversé de l'entrée matérielle.

9.1.2 Paramètres

– Aucun

9.1.3 Fonctions

– (FUNCT) **read** – lire le matériel et ajuster les HAL pins **in** et **in-not**.

9.2 Sortie numérique (Digital Output)

La sortie numérique canonique est également très simple (I/O type : **digout**).

[1] Ã partir de la version 2.0, la plupart des pilotes de HAL ne correspondent plus tout à fait à l'interface canonique décrite ici. Dans le version 2.1, les pilotes seront modifiés pour correspondre à ces spécifications.

9.2.1 Pins

– (BIT) **out** – Valeur à écrire (éventuellement inversée) sur une sortie matérielle.

9.2.2 Paramètres

– (BIT) **invert** – Si TRUE, **out** est inversée avant écriture sur la matériel.

9.2.3 Fonctions

– (FUNCT) **write** – Lit **out** et **invert** et ajuste la sortie en conséquence.

9.3 Entrée analogique (Analog Input)

L'entrée analogique canonique (I/O type : **adcin**). Devrait être utilisée pour les convertisseurs analogiques/numériques, qui convertissent par exemple, les tensions en une échelle continue de valeurs.

9.3.1 Pins

– (FLOAT) **value** – Lecture du matériel, avec mise à l'échelle ajustée par les paramètres **scale** et **offset**. **Value =** ((lecture entrée, en unités dépendantes du matériel) * **scale**) - **offset**

9.3.2 Paramètres

– (FLOAT) **scale** – La tension d'entrée (ou l'intensité) sera multipliée par **scale** avant d'être placée dans **value**.
– (FLOAT) **offset** – Sera soustrait à la tension d'entrée (ou l'intensité) après que la mise à l'échelle par scale ait été appliquée.
– (FLOAT) **bit_weight** – Valeur du bit le moins significatif (LSB). C'est effectivement, la granularité de lecture en entrée.
– (FLOAT) **hw_offset** – Valeur présente sur l'entrée quand 0 volts sont appliqués sur la pin d'entrée.

9.3.3 Fonctions

– (FUNCT) **read** – Lit les valeurs de ce canal d'entrée analogique. Peut être utilisé pour lire un canal individuellement, ou pour lire tous les canaux à la fois.

9.4 Sortie analogique (Analog Output)

La sortie analogique canonique (I/O Type : **adcout**). Elle est destinée à tout type de matériel capable de sortir une échelle plus ou moins étendue de valeurs. Comme par exemple les convertisseurs numérique/analogique ou les générateurs de PWM.

Pins

– (FLOAT) **value** – La valeur à écrire. La valeur réelle sur la sortie matérielle dépends de la mise à l'échelle des paramètres d'offset.
– (BIT) **enable** – Si fausse, la sortie matérielle passera à 0, indépendamment de la pin **value**.

9.4.1 Paramètres

– (FLOAT) **offset** – Sera ajouté à **value** avant l'actualisation du matériel.
– (FLOAT) **scale** – Doit être défini de sorte qu'une entrée avec 1 dans **value** produira 1V
– (FLOAT) **high_limit** (optionnel) – Quand la valeur en sortie matérielle est calculée, si **value** + **offset** est plus grande que **high_limit**, alors **high_limit** lui sera substitué.
– (FLOAT) **low_limit** (optionnel) – Quand la valeur en sortie matérielle est calculée, si **value** + **offset** est plus petite que **low_limit**, alors **low_limit** lui sera substitué.
– (FLOAT) **bit_weight** (optionnel) – La valeur du bit le moins significatif (LSB), en Volts (ou mA, pour les sorties courant)
– (FLOAT) **hw_offset** (optionnel) – La tension actuelle (ou l'intensité) présente sur la sortie quand 0 est écrit sur le matériel.

9.4.2 Fonctions

(FUNCT) **write** – Ecrit la valeur calculée sur la sortie matérielle. Si enable est fausse, la sortie passera à 0, indépendamment des valeurs de **value**, **scale** et **offset**. La signification de "0" dépend du matériel. Par exemple, un convertisseur A/D 12 bits peut vouloir écrire 0x1FF (milieu d'échelle) alors que le convertisseur D/A reçoit 0 Volt de la broche matérielle. Si enable est vraie, l'échelle, l'offset et la valeur sont traités et (**scale** * **value**) + **offset** sont envoyés en sortie de l'adc . Si enable est faux, la sortie passe à 0.

9.5 Codeur

L'interface de codeur canonique(I/O type : **`encoder`**) fournit les fonctionnalités nécessaires pour une prise d'origine sur une impulsion d'index et pour la synchronisation avec la vitesse de broche, ainsi que de base pour le positionnement et/ou le contrôle de vitesse. Cette interface devrait être implémentable quel que soit le matériel sous-jacent, même si certains matériels donnent de "meilleurs" résultats que d'autres. (Par exemple, pour capturer un index de position à +/- 1 impulsion lors d'un mouvement rapide, ou avoir moins de fluctuation sur la pin de vitesse).

9.5.1 Pins

– (S32) **`count`** – Valeur de comptage du codeur.
– (FLOAT) **`position`** – Valeur de position en unités de longueur (voir paramètre "scale").
– (FLOAT) **`velocity`** – Vitesse en unités de longueur par seconde.
– (BIT) **`reset`** – Quand il est vrai, force le compteur à zéro.
– (BIT) **`index-enable`** – (bidirectionnel) Quand il est vrai, remise à zéro à la prochaine impulsion d'index et passe les pins sur faux.

La pin "index-enable" est bi-directionnelle, elle exige un peu plus d'explications. Si "index-enable" est faux, le canal d'index du codeur sera ignoré et le compteur comptera normalement. Le pilote du codeur ne passera jamais "index-enable" sur vrai. Cependant, un autre composant peut le faire. Si "index-enable" est vrai, alors quand la prochaine impulsion d'index arrivera, le compteur du codeur sera remis à zéro et le pilote passera "index-enable" sur faux. Ce qui permettra à l'autre composant de savoir qu'une impulsion d'index est arrivée. C'est une forme de poignée de main, l'autre composant passe "index-enable" sur vrai pour requérir une remise à zéro du comptage d'impulsion d'index et le pilote le repasse sur faux quand la requête à été satisfaite.

9.5.2 Paramètres

– (FLOAT) **`scale`** – Le facteur d'échelle à utiliser pour convertir la valeur de comptage (count) en unités de longueur. Il se trouve dans "counts par unité de longueur". Par exemple, si vous avez

un codeur qui fournit 512 impulsions par tour de codeur sur une vis qui fait 5 tours par pouce, l'échelle (scale) devra être de 512*5 = 2560 counts par pouce, ce qui se traduira par la "position" en pouces et la "vitesse" en pouces par seconde.

– (FLOAT) **max-index-vel** – (optionnel) La vitesse maximale (en unités de longueur par seconde) à laquelle le codeur peut remettre le comptage à zéro avec une précision de +/- 1 impulsion. Il s'agit d'une sortie du pilote du codeur, elle est destinée à informer l'utilisateur des capacités du codeur. Certains codeurs peuvent remettre le comptage à zéro exactement à l'apparition de l'impulsion d'index. D'autres peuvent seulement dire qu'une impulsion d'index s'est produite depuis la dernière fois que la fonction de lecture à été appelée. Pour ces derniers, une précision de +/- 1 impulsion ne peut être atteinte que si le codeur avance d'une impulsion ou moins entre deux appels à la fonction de lecture.

– (FLOAT) **velocity-resolution** – (optionnel) La résolution de la sortie vitesse, en unités de longueur par seconde. Il s'agit d'une sortie du pilote du codeur, elle est destinée à informer l'utilisateur des capacités du codeur. L'implémentation la plus simple de la sortie vitesse est le changement de position entre deux appels à la fonction de lecture, divisé par le temps entre ces appels. Cela permet d'obtenir un signal de vitesse grossier avec des fluctuations évaluées entre deux valeurs aussi éloignées que possible (erreur de quantification). Cependant, certains matériels capturent le comptage et le temps exact quand une impulsion arrive (éventuellement avec une haute résolution d'horloge). Ces données permettent au pilote de calculer une vitesse avec une résolution plus fine et moins de fluctuations.

9.5.3 Fonctions

Il n'y a qu'une fonction pour lire les codeurs.

– (FUNCT) **read** – Capture le comptage (counts) et mets à jour la position et la vitesse.

Chapitre 10

Outils et utilitaires pour HAL

10.1 Halcmd

Halcmd est un outil en ligne de commande pour manipuler HAL. Il existe une man page plus complète pour halcmd, elle sera installée en même temps qu' EMC2 depuis ses sources ou depuis un paquet. Si EMC2 a été compilé en "run-in-place", la man page n'est pas installée, mais elle est accessible, dans le répertoire principal d'EMC2, taper :

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

Le chapitre 7 montre de nombreux exemples d'utilisation de halcmd, c'est un bon tutoriel pour halcmd.

10.2 Halmeter

Halmeter est un "voltmètre" pour HAL. Il permet de regarder les pins, signaux, ou paramètres en affichant la valeur courante de ces items. Il est très simple à utiliser. Dans une console taper "`halmeter`". Halmeter est une application pour environnement graphique. Deux fenêtres vont apparaîtrent, la fenêtre de sélection est la plus grande. Elle comprend trois onglets. Un onglet liste toutes les pins actuellement définies dans HAL. Le suivant, liste tous les signaux et le dernier onglet, liste tous les paramètres. Cliquer sur un onglet, puis cliquer sur un des pin/signal/paramètre pour le sélectionner. La petite fenêtre affichera le nom et la valeur de l'item sélectionné. L'affichage est mis à jour environ 10 fois par seconde. Pour libérer de la place sur l'écran, la fenêtre de sélection peut être fermée avec le bouton "Close". Sur la petite fenêtre, cachée sous la grande à l'ouverture, le bouton "Select", réouvre la fenêtre de sélection et le bouton Exit arrête le programme et ferme les fenêtres.

Il est possible d'ouvrir et de faire fonctionner simultanément plusieurs halmeters, ce qui permet de visualiser plusiers items en même temps. Pour ouvrir un halmeter en libérant la console, taper "`halmeter &`" pour le lancer en tâche de fond. Il est possible de lancer halmeter en lui faisant afficher immédiatement un item, pour cela, ajouter les arguments sur la ligne de commande "`pin|sig|par[am] <nom>`". Il affichera le signal, pin, ou paramètre <nom> dès qu'il démarrera. (Si l'item indiqué n'existe pas, il démarrera normalement. Finalement, si un item est spécifié pour l'affichage, il est possible d'ajouter "-s" devant pin|sig|param pour indiquer à halmeter d'utiliser une fenêtre encore plus réduite. Le nom de l'item sera affiché dans la barre de titre au lieu de sous la valeur et il n'y aura pas de bouton. Utile pour afficher beaucoup de halmeter dans un petit espace de l'écran.

10.3 Halscope

Halscope est un “oscilloscope” pour HAL. Il permet de capturer la valeur des pins, signaux et paramètres en fonction du temps. Des instructions plus complètes seront ajoutées ici, éventuellement. Pour l’instant, se référer à la section 7.6 dans le chapitre du tutoriel, qui explique les bases de son utilisation.

10.4 Halshow

Le script halshow peut vous aider à retrouver votre chemin dans un HAL en fonctionnement. Il s'agit d'un système très spécialisé qui doit se connecter à un HAL en marche. Il ne peut pas fonctionner seul car il repose sur la capacité de HAL de rapporter ce qu'il connaît de lui même par la librairie d'interface de halcmd. Chaque fois que halshow fonctionne avec une configuration d'EMC différente, il sera différent.

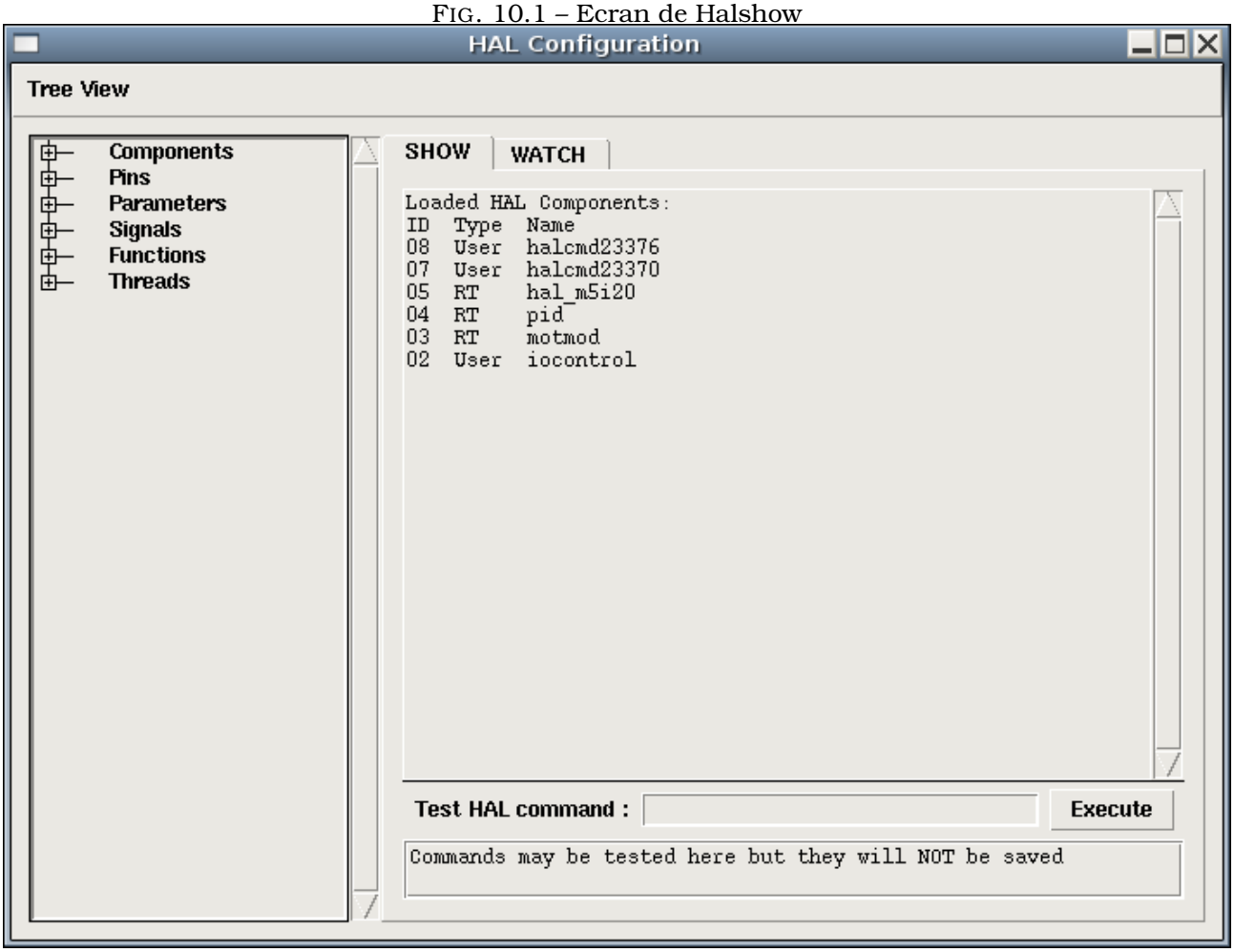
Comme nous le verrons bientôt, cette capacité de HAL de se documenter lui même est un des facteurs clés pour arriver à un système CNC optimum.

On peut accéder à Halshow depuis Axis, pour cela, aller dans le menu "Machine" puis choisir "Afficher la configuration de HAL".

10.4.1 Zone de l'arborescence de Hal

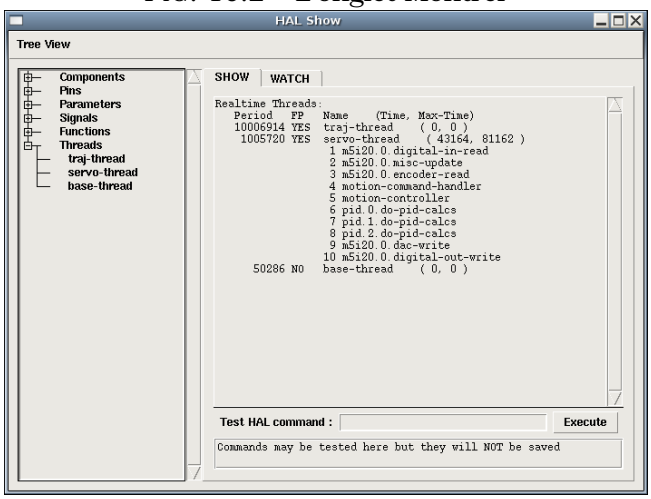
La gauche de l'écran que montre la figure 10.1 est une arborescence, un peu comme vous pouvez le voir avec certains navigateurs de fichiers. Sur la droite, une zone avec deux onglets : MONTRER et WATCH.

FIG. 10.1 – Ecran de Halshow



L'arborescence montre toutes les parties principales de HAL. En face de chacune d'entre elles, se trouve un petit signe + ou - dans une case. Cliquer sur le signe plus pour déployer cette partie de

FIG. 10.2 – L'onglet Montrer



l'arborescence et affichera son contenu. Si cette case affiche un signe moins, cliquer dessus repliera cette section de l'arborescence.

Il est également possible de déployer et de replier l'arborescence complète depuis le menu "Arborescence".

10.4.2 Zone de l'onglet MONTRER

En cliquant sur un nom dans l'arborescence plutôt que sur son signe plus ou moins, par exemple le mot "Components", HAL affichera tout ce qu'il connait du contenu de celui-ci. La figure 10.1 montre une liste comme celle que vous verrez si vous cliquez sur "Components" avec une carte servo standard m5i20 en fonctionnement. L'affichage des informations est exactement le même que celui des traditionnels outils d'analyse de HAL en mode texte. L'avantage ici, c'est que nous y avons accès d'un clic de souris. Accès qui peut être aussi large ou aussi focalisé que vous le voulez.

Si nous examinons de plus près l'affichage de l'arborescence, nous pouvons voir que les six éléments principaux peuvent tous être déployés d'au moins un niveau. Quand ces niveaux sont à leur tour déployés vous obtenez une information de plus en plus focalisée en cliquant sur le nom des éléments dans l'arborescence. Vous trouverez que certaines hal pins et certains paramètres affichent plusieurs réponses. C'est dû à la nature des routines de recherche dans halcmd lui même. Si vous cherchez une pin, vous pouvez en trouver deux comme cela :

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

Le deuxième nom de pin contient le nom complémenté du premier.

Dans le bas de l'onglet Montrer, un champ de saisie permet de jouer sur le fonctionnement de HAL. Les commandes que vous entrez ici et leur effet sur HAL, ne sont pas enregistrés. Elles persisteront tant qu'EMC tournera, mais disparaîtront dès son arrêt.

Le champ de saisie marqué "Tester une commande HAL :" acceptera n'importe quelle commande valide pour halcmd. Elles incluent :

– loadrt, unloadrt
– addf, delf
– newsig, delsig
– linkpp, linksp, linkps, unlinkp

– setp, sets

Ce petit éditeur entrera une commande à chaque fois que vous presserez <Entrée> ou que vous cliquerez sur le bouton “Exécuter”. Si une commande y est mal formée, un dialogue d’erreur s’affichera. Si vous n’êtes pas sûr de savoir comment formuler une commande, vous trouverez la réponse dans la documentation de halcmd et des modules spécifiques avec lesquels vous travaillez.

Nous allons utiliser cet éditeur pour ajouter un module différentiel à HAL et le connecter à la position d’un axe pour voir le ratio de changement de position, par exemple, l’accélération. Il faut d’abord charger un module de HAL nommé blocks, l’ajouter au thread servo et le connectet à la pin position d’un axe. Une fois cela fait, nous pourrons retrouver la sortie du différentiateur dans halscope. Alors allons-y. (oui j’ai vérifié).

```
loadrt blocks ddt=1
```

Maintenant, regardez dans components, vous devriez y voir blocks.

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

Effectivement, il est là. Dans notre cas l’id est 08. Ensuite nous devons savoir quelles fonctions sont disponibles avec lui, nous regardons dans Functions.

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

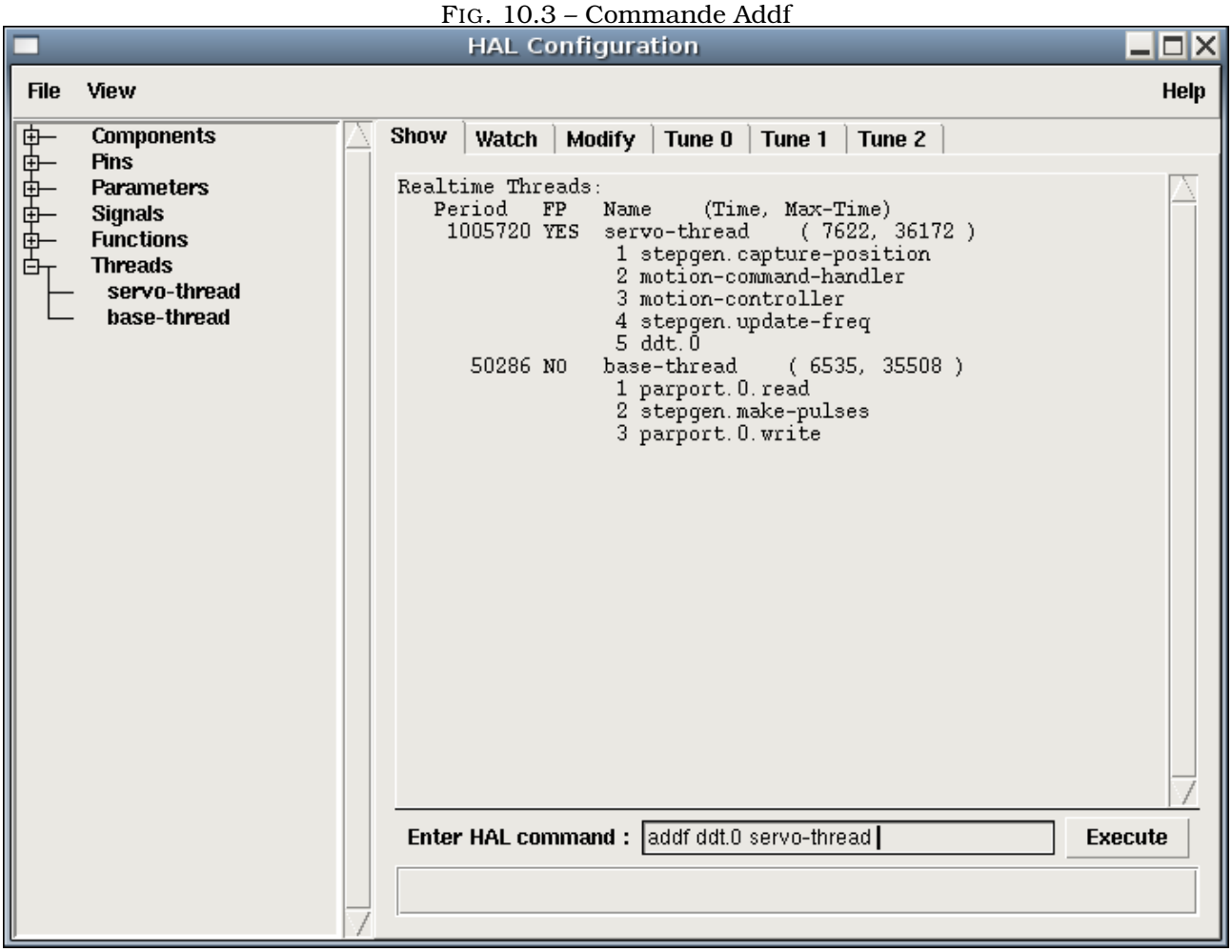
Ici, nous cherchons owner #08 et voyons que blocks a exporté une fonction nommée ddt.0. Nous devrions être en mesure d’ajouter ddt.0 au thread servo et il fera ses calculs chaque fois que le thread sera mis à jour. Encore une fois recherchons la commande addf et on voit qu’elle utilise trois arguments comme cela :

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

Nous connaissons déjà functname=ddt.0, pour trouver le nom du thread, déployons l’arborescence des Threads. Nous y trouvons deux threads, servo-thread et base-thread. La position de ddt.0 dans le thread n’est pas critique. Passons la commande :

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

FIG. 10.3 – Commande Addf



Comme c'est juste pour visualiser, nous laissons la position en blanc pour obtenir la dernière position dans le thread. La figure 10.3 montre l'état de halshow après que cette commande a été exécutée.

Ensuite, nous devons connecter ce block à quelque chose. Mais comment savoir quelles pins sont disponibles ? La réponse se trouve dans l'arbre, en regardant sous Pins. On y trouve ddt et on voit :

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

Cela semble assez facile à comprendre, mais à quel signal ou pin voulons-nous nous connecter, ça pourrait être une pin d'axe, une pin de stepgen, ou un signal. On vois cela en regardant dans axis.0.

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

Donc, il semble que Xpos-cmd devrait être un bon signal à utiliser. Retour à l'éditeur et entrons la commande suivante :

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

Maintenant si on regarde le signal Xpos-cmd dans l'arbre, on voit ce qu'on a fait.

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

Nous voyons que ce signal provient de axis.0.motor-pos-cmd et va, à la fois, sur ddt.0.in et sur stepgen.0.position-cmd. En connectant notre block au signal nous avons évité les complications avec le flux normal de cette commande de mouvement.

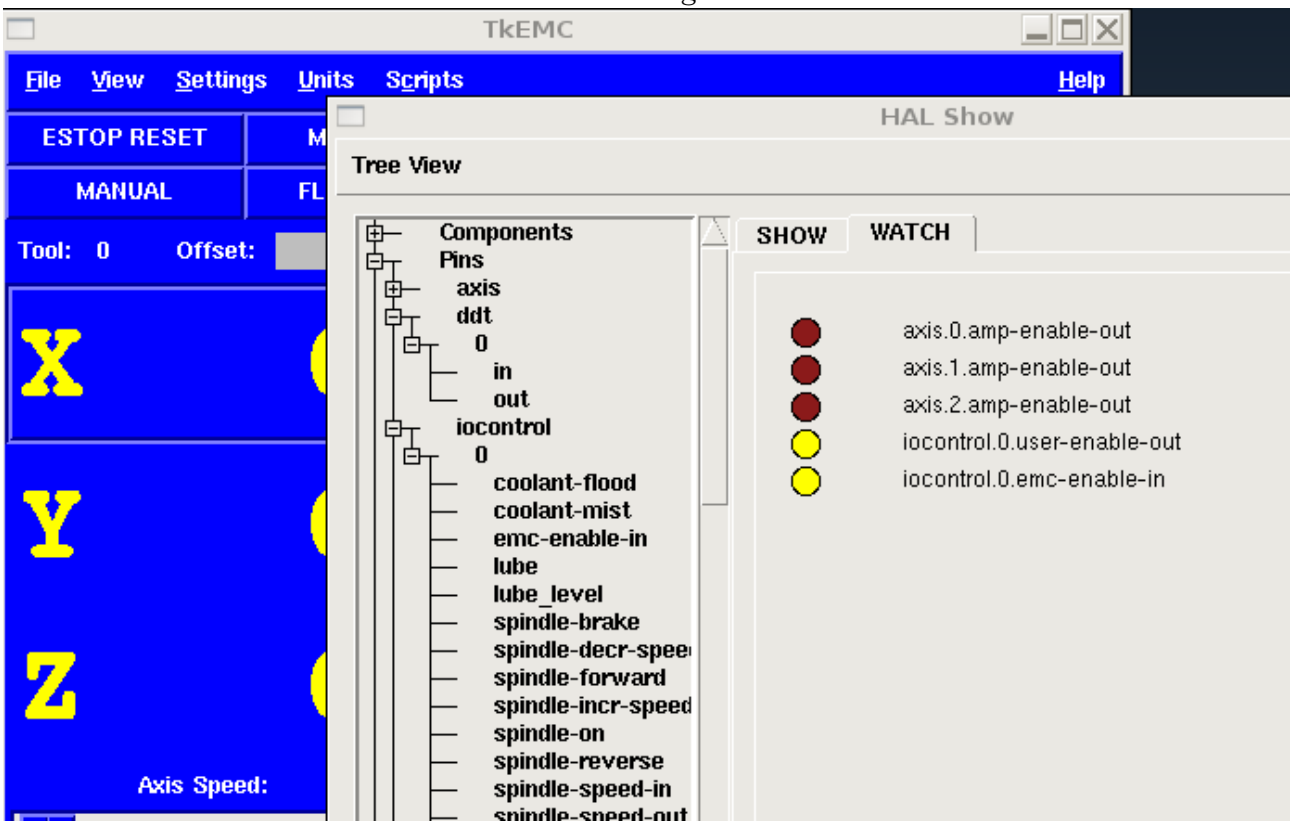
La zone de l'onglet "Montrer" utilise halcmd pour découvrir ce qui se passe à l'intérieur de HAL pendant son fonctionnement. Il vous donne une information complète de ce qu'il découvre. Il met aussi à jour dès qu'une commande est envoyée depuis le petit éditeur pour modifier ce HAL. Il arrive un temps ou vous voulez autre chose d'affiché, sans la totalité des informations disponibles dans cette zone. C'est la grande valeur de l'onglet WATCH d'offrir cela.

10.4.3 Zone de l'onglet WATCH

En cliquant sur l'onglet WATCH une zone vide s'affichera. Vous pouvez ajouter des signaux et des pins dans cette zone et visualiser leurs valeurs.[1] Vous pouvez ajouter des pins ou des signaux quand l'onglet Watch est ouvert, en cliquant sur leurs noms. La figure 10.4 Montre cette zone avec plusieurs signaux de type "bit". Parmis ces signaux, les enable-out pour les trois premiers axes et deux de la branche iocontrol, les signaux "estop". Notez que les axes ne sont pas activés même si les signaux estop disent qu'EMC n'est pas en estop. Un bref regard sur themc en arrière plan, montre que l'état d'EMC est ESTOP RESET. L'activation des amplis ne deviendra pas vraie tant que la machine ne sera pas mise en marche.

[1] Le taux de rafraîchissement de la zone Watch est plus lent que celui de Halmeter ou de Halscope. Si vous avez besoin d'une bonne résolution dans le timming des signaux, ces outils sont plus efficaces.

FIG. 10.4 – L'onglet Watch



Les cercles de deux couleurs, simili leds, sont toujours bruns foncé quand un signal est faux. Elle sont jaunes quand le signal est vrai. Quand une pin ou un signal est sélectionné mais n'est pas de type bit, sa valeur numérique s'affiche.

Watch permet de visualiser rapidement le résultat de tests sur des contacts ou de voir l'effet d'un changement que vous faites dans EMC en utilisant l'interface graphique. Le taux de rafraîchissement de Watch est un peu trop lent pour visualiser les impulsions de pas d'un moteur mais vous pouvez l'utiliser si vous déplacez un axe très lentement ou par très petits incréments de distance. Si vous avez déjà utilisé IO_Show dans EMC, la page de Watch de halshow peut être réglée pour afficher ce que fait le port parallèle.

Chapter 11

comp: a tool for creating HAL modules

11.1 Introduction

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

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

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

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

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

11.2 Definitions

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

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

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

11.3 Instance creation

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

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

11.4 Syntax

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

Declarations include:

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

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

HALNAME An identifier.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

fp Indicates that the function performs floating-point calculations.

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

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

option singleton yes (default: no)

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

option default_count *number* (default: 1)

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

option count_function yes (default: no)

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

option rtapi_app no (default: yes)

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

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

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

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

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

option extra_setup yes (default: no)

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

option extra_cleanup yes (default: no)

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

option userspace yes (default: no)

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

option userinit yes (default: no)

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

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

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

11.5 Per-instance data storage

variable *CTYPE NAME*;

variable *CTYPE NAME*[*SIZE*];

variable *CTYPE NAME* = *DEFAULT*;

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

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

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

11.6 Other restrictions on comp files

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

11.7 Convenience Macros

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

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

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

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

pin_name

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

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

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

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

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

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

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

11.8 Components with one function

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

11.9 Component "Personality"

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

11.10 Compiling `.comp` files in the source tree

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

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

11.11 Compiling realtime components outside the source tree

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

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

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

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

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

```
comp rtexample.comp
```

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

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

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

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

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

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

or copied to a standard location for manual pages.

11.12 Compiling userspace components outside the source tree

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

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

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

11.13 Examples

11.13.1 constant

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

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

11.13.2 sincos

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

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

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

11.13.3 out8

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

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

function _;

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

;;
#include <asm/io.h>

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

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

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

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

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

11.13.4 hal_loop

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

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

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

11.13.5 arraydemo

This realtime component illustrates use of fixed-size arrays:

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

11.13.6 rand

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

```
component rand;
option userspace;

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

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

11.13.6.1 logic

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

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

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

A typical load line for this component might be

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

which creates the following pins:

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

Chapter 12

Creating Userspace Python Components with the 'hal' module

12.1 Basic usage

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

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

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

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

12.2 Userspace components and delays

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

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

12.3 Create pins and parameters

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

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

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

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

Table 12.1: HAL Option Names

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

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

```
h.ready()
```

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

12.3.1 Changing the prefix

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

12.4 Reading and writing pins and parameters

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

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

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

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

12.4.1 Driving output (HAL_OUT) pins

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

12.4.2 Driving bidirectional (HAL_IO) pins

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

12.5 Exiting

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

12.6 Project ideas

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

Part V

EMC en relation avec HAL

Chapter 13

Configuration simple pour système “dir/step”

13.1 Introduction

This chapter describes some of the more common settings that users want to change when setting up EMC2. Because of the various possibilities of configuring EMC2, it is very hard to document them all, and keep this document relatively short. This chapter describes some of the more common settings that users want to change when setting up EMC2. Because of the various possibilities of configuring EMC2, it is very hard to document them all, and keep this document relatively short.

The most common EMC2 usage (as reported by our users) is for stepper based systems. These systems are using stepper motors with drives that accept step & direction signals.

It is one of the simpler setups, because the motors run open-loop (no feedback comes back from the motors), yet the system needs to be configured properly so the motors don't stall or lose steps.

Most of this chapter is based on the sample config released along with EMC2. The config is called stepper, and usually it is found in `/etc/emc2/sample-configs/stepper`.

13.2 Maximum step rate

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

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

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

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

13.3 Pinout

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

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

The ones relevant for our pinout are[1]:

```
signals: Xstep, Xdir & Xen
pins: parport.0.pin-XX-out & parport.0.pin-XX-in [2]
```

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

13.3.1 standard_pinout.hal

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

13.3.2 Overview of the standard_pinout.hal

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

1. The Parport driver gets loaded (see 15.1 for details)
2. The read & write functions of the parport driver get assigned to the Base thread [3]
3. The step & direction signals for axes X,Y,Z get linked to pins on the parport
4. Further IO signals get connected (estop loopback, toolchanger loopback)
5. A spindle On signal gets defined and linked to a parport pin

13.3.3 Changing the standard_pinout.hal

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

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

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

can be changed to:

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

or basicly any other numbers you like.

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

13.3.4 Changing the polarity of a signal

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

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

13.3.5 Adding PWM Spindle Speed Control

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

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

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

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

13.3.6 Adding an enable signal

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

To connect them use the following example:

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

You can either have one single pin that enables all drives, or several, depending on the setup you have. Note however that usually when one axis faults, all the other ones will be disabled aswell, so having only one signal / pin is perfectly safe.

13.3.7 Adding an external ESTOP button

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

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

```
linkpp iocontrol.0.user-enable-out iocontrol.0.emc-enable-in
```

with

```
linkpp parport.0.pin-01-in iocontrol.0.emc-enable-in
```

This assumes an ESTOP switch connected to pin 01 on the parport. As long as the switch will stay pushed[5], EMC2 will be in the ESTOP state. When the external button gets released EMC2 will imediately switch to the ESTOP-RESET state, and all you need to do is switch to Machine On and you'll be able to continue your work with EMC2.

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

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

Chapter 14

Composants internes

La plupart des composants ont leurs pages de manuel, style *nix. Pour afficher ces “man pages” pour les composants temps réel, taper dans un terminal “man 9 *nomducomposant*”.

Le présent document se concentre sur les composants les plus complexes qui demandent des figures difficiles à reproduire dans une man page.

14.1 Stepgen

Ce composant fournit un générateur logiciel d’impulsions de pas répondant aux commandes de position ou de vitesse. En mode position, il contient une boucle de position pré-réglée, de sorte que les réglages de PID ne sont pas nécessaires. En mode vitesse, il pilote un moteur à la vitesse commandée, tout en obéissant aux limites de vitesse et d’accélération. C’est un composant uniquement temps réel, dépendant de plusieurs facteurs comme la vitesse du CPU, etc, il est capable de fournir des fréquences de pas maximum comprisent entre 10kHz et 50kHz. La figure 14.1 montre trois schémas fonctionnels, chacun est un simple générateur d’impulsions de pas. Le premier diagramme est pour le type ’0’, (pas et direction). Le second est pour le type ’1’ (up/down, ou pseudo-PWM) et le troisième est pour les types 2 jusqu’à 14 (les différentes séquences de pas). Les deux premiers diagrammes montrent le mode de commande position et le troisième montre le mode vitesse. Le mode de commande et le type de pas, se réglent indépendamment et n’importe quelle combinaison peut être choisie.

14.1.1 L’installer

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

<type-array> est une série d’entiers décimaux séparés par des virgules. Chaque chiffre provoquera le chargement d’un simple générateur d’impulsions de pas, la valeur de ce chiffre déterminera le type de pas. <ctrl_array> est une série de lettres “**p**” ou “**v**” séparées par des virgules, qui spécifient le mode pas ou le mode vitesse. **ctrl_type** est optionnel, si il est omis, tous les générateurs de pas seront en mode position. Par exemple, la commande:

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

va installer trois générateurs de pas. Les deux premiers utilisent le type de pas ’0’ (pas et direction) et fonctionnent en mode position. Le dernier utilise le type de pas ’2’ (quadrature) et fonctionne en mode vitesse. La valeur par défaut de <config-array> est “0,0,0” qui va installer trois générateurs de type ’0’ (step/dir). Le nombre maximum de générateurs de pas est de 8 (comme définit par

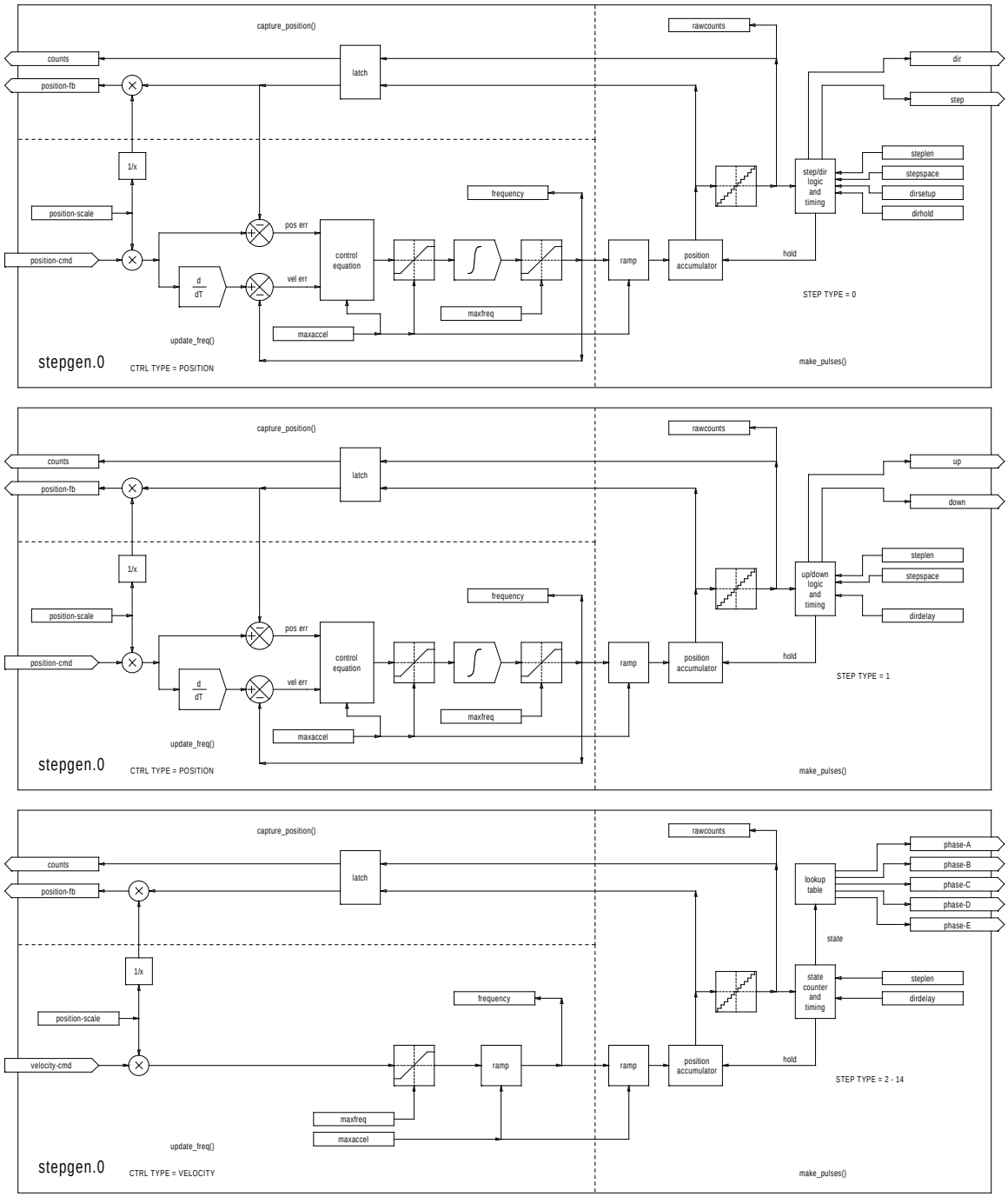


Figure 14.1: Diagramme bloc du générateur de pas (en mode position)

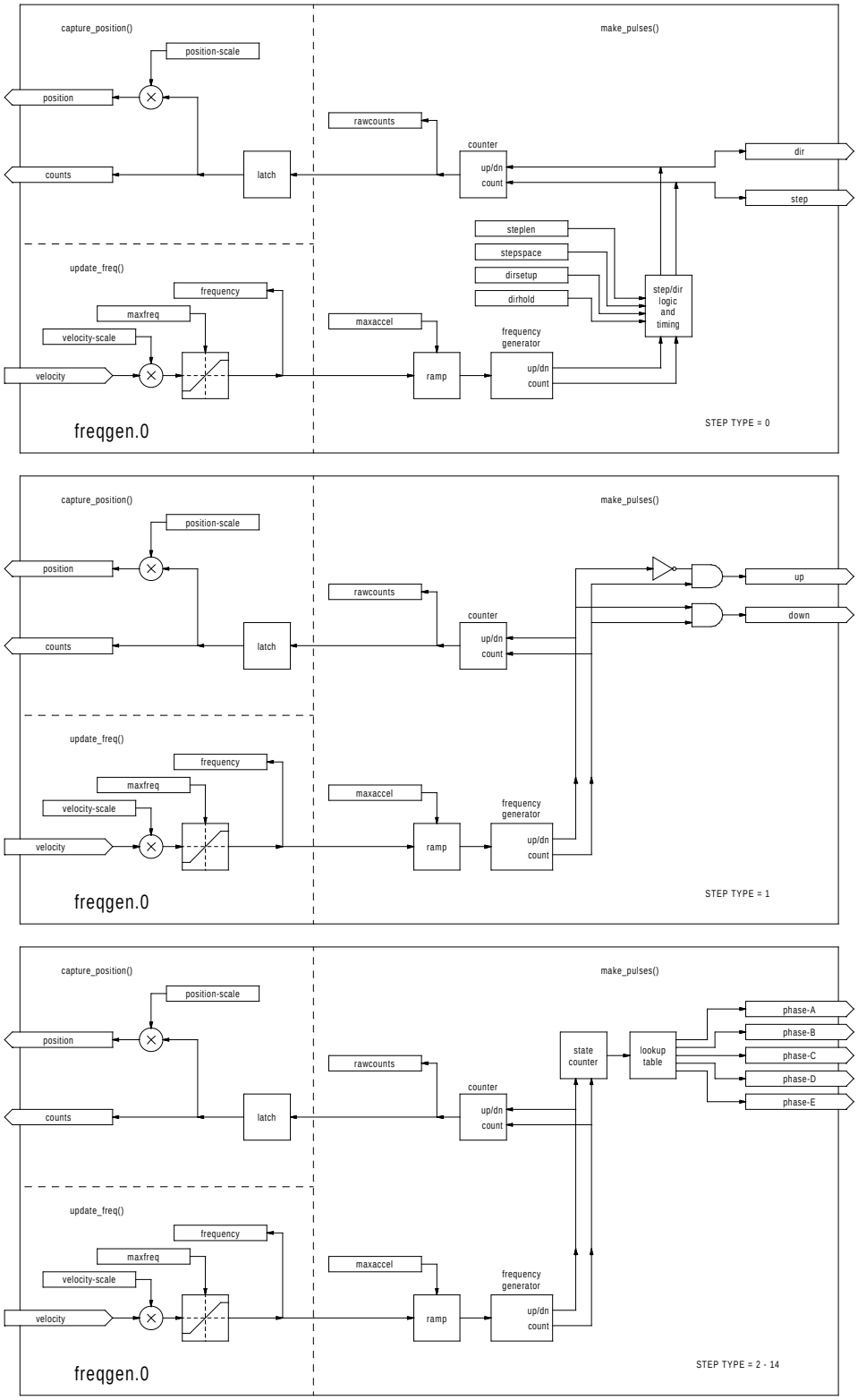


Figure 14.2: Diagramme bloc du générateur de pas (en mode vitesse)

MAX_CHAN dans stepgen.c). Chaque générateur est indépendant, mais tous sont actualisés par la même fonction(s), au même instant. Dans les descriptions qui suivent, <chan> est le nombre de générateurs spécifiques. La numérotation des générateurs commence à 0.

14.1.2 Le désinstaller

```
emc2$ halcmd unloadrt stepgen
```

14.1.3 Pins

Chaque générateur d'impulsions de pas n'aura que certaines de ces pins, selon le type de pas et le mode de contrôle sélectionné.

- (FLOAT) stepgen.<chan>.position-cmd – Position désirée du moteur, en unités de longueur (mode position seulement).
- (FLOAT) stepgen.<chan>.velocity-cmd – Vitesse désirée du moteur, en unités de longueur par seconde (mode vitesse seulement).
- (S32) stepgen.<chan>.counts – Rétroaction de la position en unités de comptage, actualisée par la fonction capture_position().
- (FLOAT) stepgen.<chan>.position-fb – Rétroaction de la position en unités de longueur, actualisée par la fonction capture_position().
- (BIT) stepgen.<chan>.step – Sortie des impulsions de pas (type de pas 0 seulement).
- (BIT) stepgen.<chan>.dir – Sortie direction (type de pas 0 seulement).
- (BIT) stepgen.<chan>.up – Sortie UP en pseudo-PWM (type de pas 1 seulement).
- (BIT) stepgen.<chan>.down – Sortie DOWN en pseudo-PWM (type de pas 1 seulement).
- (BIT) stepgen.<chan>.phase-A – Sortie phase A (séquences de pas 2 à 14 seulement).
- (BIT) stepgen.<chan>.phase-B – Sortie phase B (séquences de pas 2 à 14 seulement).
- (BIT) stepgen.<chan>.phase-C – Sortie phase C (séquences de pas 3 à 14 seulement).
- (BIT) stepgen.<chan>.phase-D – Sortie phase D (séquences de pas 5 à 14 seulement).
- (BIT) stepgen.<chan>.phase-E – Sortie phase E (séquences de pas 11 à 14 seulement).

14.1.4 Paramètres

- (FLOAT) stepgen.<chan>.position-scale – Pas par unité de longueur. Ce paramètre est utilisé pour les sorties et les rétroactions.
- (FLOAT) stepgen.<chan>.maxvel – Vitesse maximale, en unités de longueur par seconde. Si égal à 0.0, n'a aucun effet.
- (FLOAT) stepgen.<chan>.maxaccel – Valeur maximale d'accélération, en unités de longueur par seconde au carré. Si égal à 0.0, n'a aucun effet.
- (FLOAT) stepgen.<chan>.frequency – Fréquence des pas, en pas par seconde.
- (FLOAT) stepgen.<chan>.steplen – Durée de l'impulsion de pas (types de pas 0 et 1) ou durée minimum dans un état donné (séquences de pas 2 à 14), en nanosecondes.

- (FLOAT) stepgen.<chan>.stepspace – Espace minimum entre deux impulsions de pas (types de pas 0 et 1 seulement), en nanosecondes.
- (FLOAT) stepgen.<chan>.dirsetup – Durée minimale entre un changement de direction et le début de la prochaine impulsion de pas (type de pas 0 seulement), en nanosecondes.
- (FLOAT) stepgen.<chan>.dirhold – Durée minimale entre la fin d'une impulsion de pas et un changement de direction (type de pas 0 seulement), en nanosecondes.
- (FLOAT) stepgen.<chan>.dirdelay – Durée minimale entre un pas dans une direction et un pas dans la direction opposée (séquences de pas 1 à 14 seulement), en nanosecondes.
- (S32) stepgen.<chan>.rawcounts – Valeur de comptage brute (count) de la rétroaction, réactualisée par la fonction make_pulses().

En mode position, les valeurs de maxvel et de maxaccel sont utilisées par la boucle de position interne pour éviter de générer des trains d'impulsions de pas que le moteur ne peut pas suivre. Lorsqu'elles sont réglées sur des valeurs appropriées pour le moteur, même un grand changement instantané dans la position commandée produira un mouvement trapézoïdal en douceur vers la nouvelle position. L'algorithme fonctionne en mesurant à la fois, l'erreur de position et l'erreur de vitesse, puis en calculant une accélération qui tende à réduire vers zéro, les deux en même temps. Pour plus de détails, y compris les contenus de la boîte "d'équation de contrôle", consulter le code source.

En mode vitesse, maxvel est une simple limite qui est appliquée à la vitesse commandée, maxaccel est utilisé pour créer une rampe avec la fréquence actuelle, si la vitesse commandée change brutalement. Comme dans le mode position, des valeurs appropriées de ces paramètres assurent que le moteur pourra suivre le train d'impulsions généré.

14.1.5 Séquences de pas

Le générateur de pas supporte 15 différentes "séquences de pas". Le type de pas 0 est le plus familier, c'est le standard pas et direction (step/dir). Quand stepgen est configuré pour le type 0, il y a quatre paramètres supplémentaires qui déterminent le timing exact des signaux de pas et de direction. Voir la figure 14.3 pour la signification de ces paramètres. Les paramètres sont en nanosecondes, mais ils doivent être arrondis à un entier, multiple de la période du thread qui appelle make_pulses(). Par exemple, si make_pulses() est appelée toutes les 16s et que "steplen" est à 20000, alors l'impulsion de pas aura une durée de 2 x 16 = 32s. La valeur par défaut de ces quatre paramètres est de 1ns, mais l'arrondi automatique prendra effet au premier lancement du code. Puisqu'un pas exige d'être haut pendant "steplen"ns et bas pendant "stepspace"ns, la fréquence maximale est 1.000.000.000 divisé par (steplen+stepspace). Si "maxfreq" est réglé plus haut que cette limite, il sera abaissé automatiquement. Si "maxfreq" est à zéro, il restera à zéro, mais la fréquence de sortie sera toujours limitée.

Le type de pas 1 a deux sorties, up et down. Les impulsions apparaissent sur l'une ou l'autre, selon la direction du déplacement. Chaque impulsion a une durée de "steplen"ns et les impulsions sont séparées de "stepspace"ns. La fréquence maximale est la même que pour le type 0. Si "maxfreq" est réglé plus haut que cette limite il sera abaissé automatiquement. Si "maxfreq" est à zéro, il restera à zéro, mais la fréquence de sortie sera toujours limitée.

Les séquences 2 jusqu'à 14 sont basées sur les états et ont entre deux et cinq sorties. Pour chaque pas, un compteur d"état est incrémenté ou décrémenté. Les figures 14.4, 14.5 et 14.6 montrent les différentes séquences des sorties en fonction de l'état du compteur. La fréquence maximale est 1.000.000.000 divisé par "steplen" et comme dans les autres séquences, "maxfreq" sera abaissé si il est au dessus de cette limite.

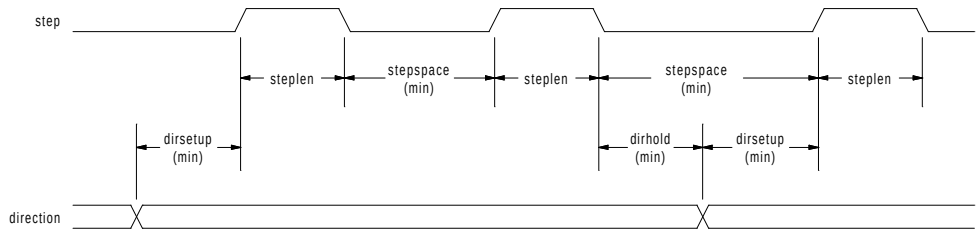


Figure 14.3: Timing pas et direction

14.1.6 Fonctions

Le composant exporte trois fonctions. Chaque fonction agit sur tous les générateurs d'impulsions de pas. Lancer différents générateurs dans différents threads n'est pas supporté.

- `(FUNCT) stepgen.make-pulses` – Fonction haute vitesse de génération et de comptage des impulsions (non flottant).
- `(FUNCT) stepgen.update-freq` – Fonction basse vitesse de conversion de position en vitesse, mise à l'échelle et traitement des limitations.
- `(FUNCT) stepgen.capture-position` – Fonction basse vitesse pour la rétroaction, met à jour les latches et les mesures de position.

La fonction à grande vitesse "`stepgen.make-pulses`" devrait être lancée dans un thread très rapide, entre 10 et 50s selon les capacités de l'ordinateur. C'est la période de ce thread qui détermine la fréquence maximale des pas, de `steplen`, `stepspace`, `dirsetup`, `dirhold` et `dirdelay`, `tous sont arrondis` au multiple entier de la période du thread en nanosecondes. Les deux autres fonctions peuvent être appelées beaucoup plus lentement.

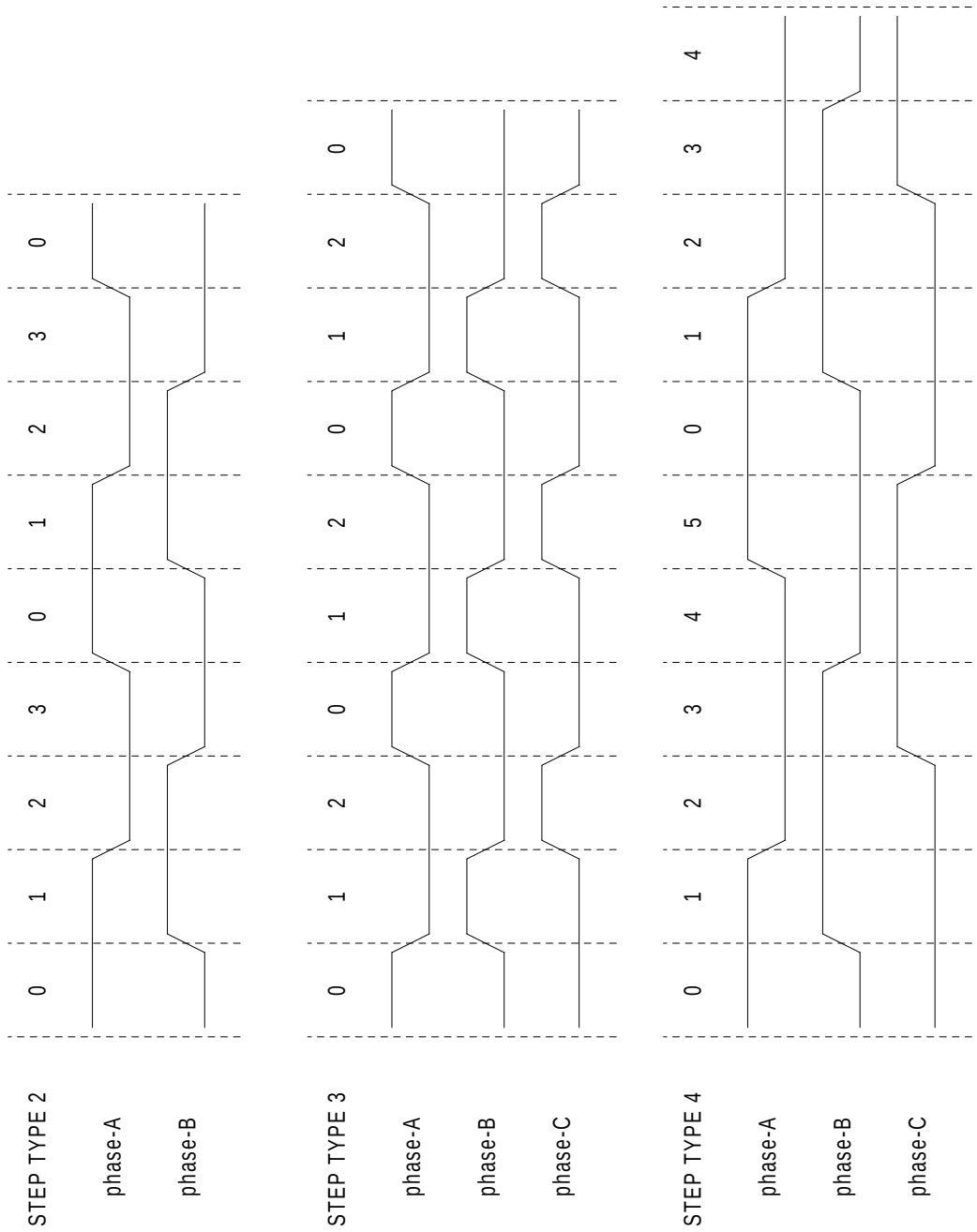


Figure 14.4: Séquences de pas à trois phases

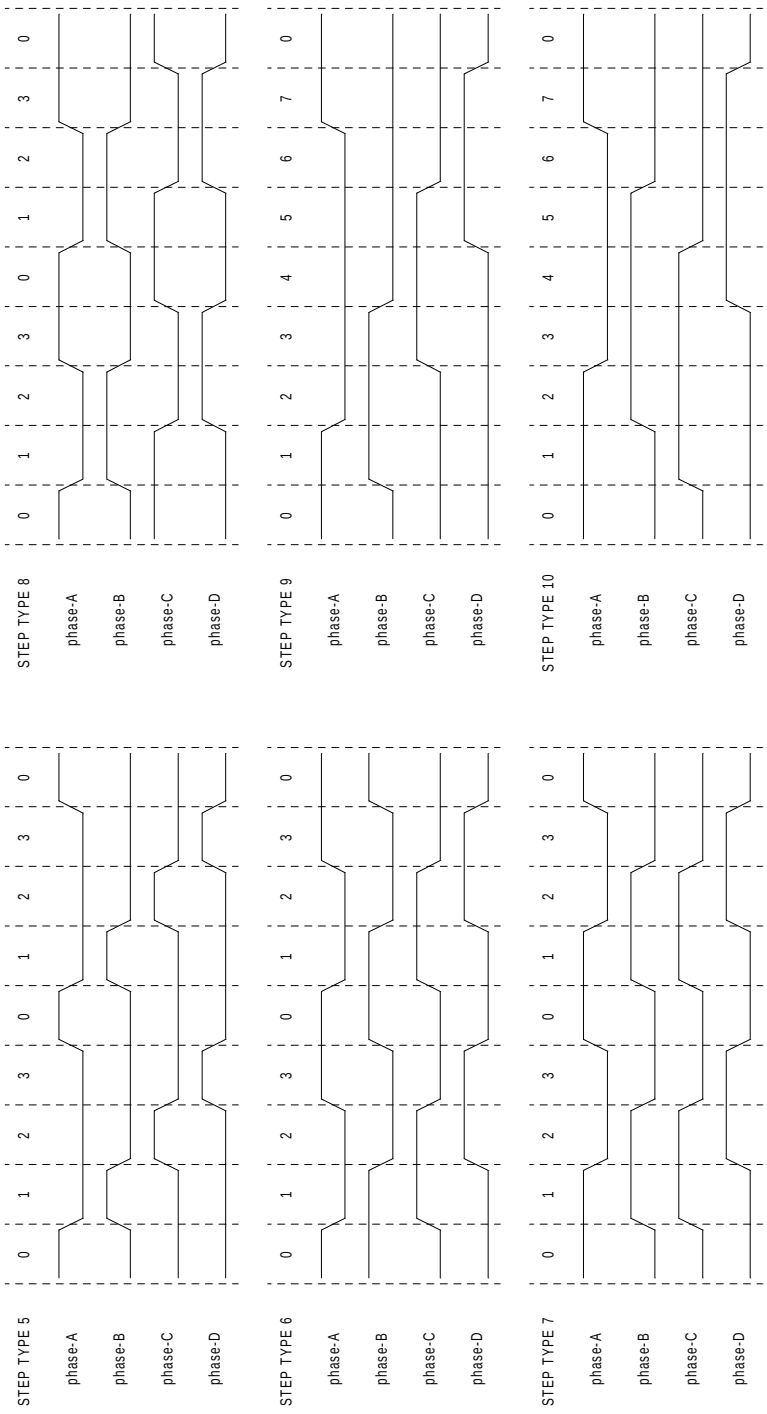


Figure 14.5: Séquences de pas à quatre phases

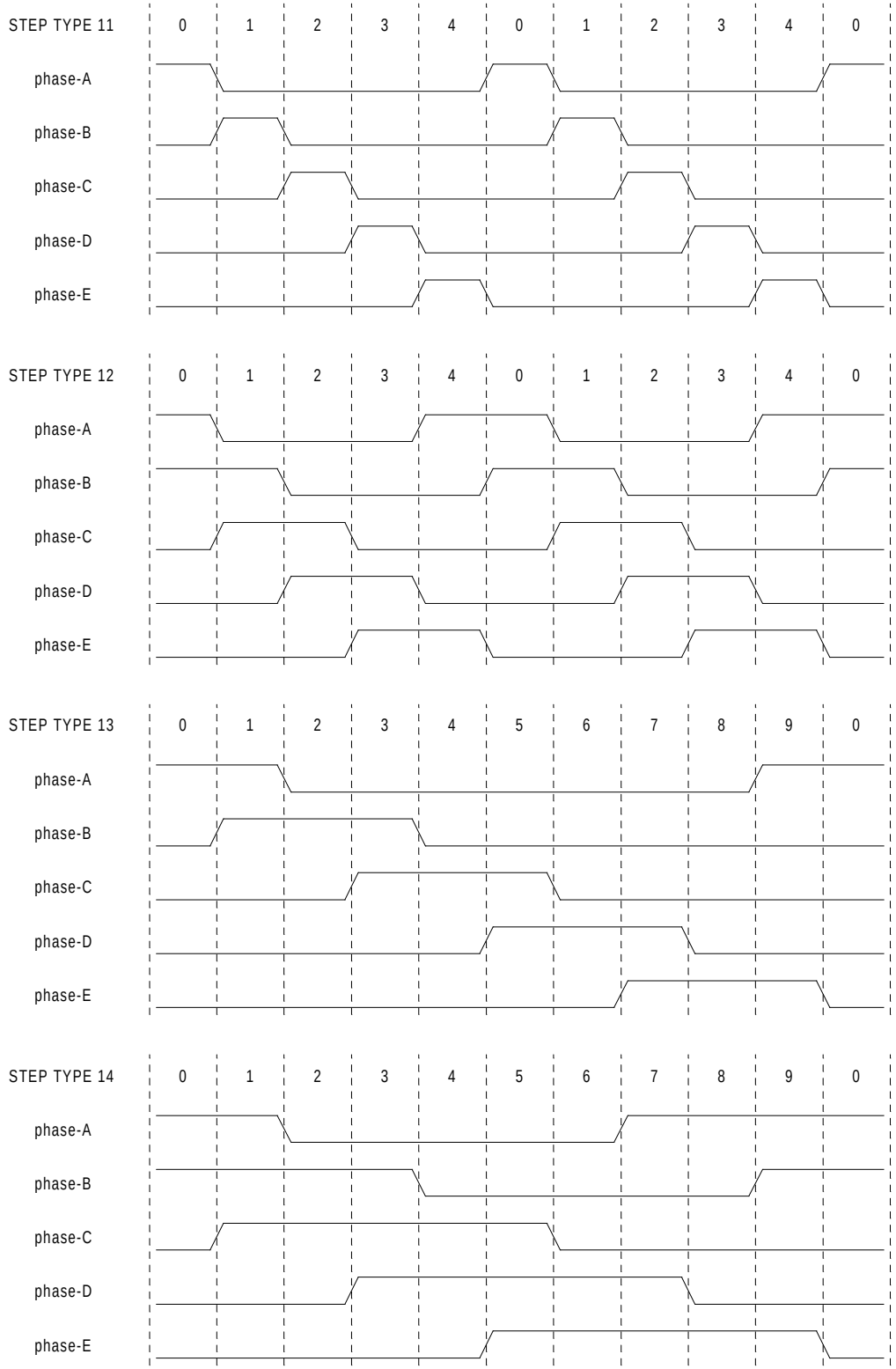


Figure 14.6: Séquence de pas à cinq phases

14.2 PWMgen

Ce composant fournit un générateur logiciel de PWM (modulation de largeur d'impulsions) et PDM (modulation de densité d'impulsions). C'est un composant temps réel uniquement, dépendant de plusieurs facteurs comme la vitesse du CPU, etc, Il est capable de générer des fréquences PWM de quelques centaines de Hertz en assez bonne résolution, à peut-être 10kHz avec une résolution limitée.

14.2.1 L'installer

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

<config-array> est une série d'entiers décimaux séparés par des virgules. Chaque chiffre provoquera le chargement d'un simple générateur de PWM, la valeur de ce chiffre determinera le type de sortie. Par exemple:

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

va installer trois générateurs de PWM. Le premier utilisera une sortie de type '0' (PWM seule), le suivant utilisera une sortie de type 1 (PWM et direction) et le troisième utilisera une sortie de type 2 (UP et DOWN). Il n'y a pas de valeur par défaut, si <config-array> n'est pas spécifié, aucun générateur de PWM ne sera installé. Le nombre maximum de générateurs de fréquences est de 8 (comme définit par MAX_CHAN dans pwmgen.c). Chaque générateur est indépendant, mais tous sont mis à jour par la même fonction(s), au même instant. Dans les descriptions qui suivent, <chan> est le nombre de générateurs spécifiques. La numérotation des générateurs de PWM commence à 0.

14.2.2 Le désinstaller

```
emc2$ halcmd unloadrt pwmgen
```

14.2.3 Pins

Chaque générateur de PWM aura les pins suivantes:

- (FLOAT) pwmgen.<chan>.value – Valeur commandée, en unités arbitraires. Sera mise à l'échelle par le paramètre d'échelle (voir ci-dessous).
- (BIT) pwmgen.<chan>.enable – Active ou désactive les sorties du générateur de PWM.

Chaque générateur de PWM aura également certaines de ces pins, selon le type de sortie choisi:

- (BIT) pwmgen.<chan>.pwm – Sortie PWM (ou PDM), (types de sortie 0 et 1 seulement).
- (BIT) pwmgen.<chan>.dir – Sortie direction (type de sortie 1 seulement).
- (BIT) pwmgen.<chan>.up – Sortie PWM/PDM pour une valeur positive en entrée (type de sortie 2 seulement).
- (BIT) pwmgen.<chan>.down – Sortie PWM/PDM pour une valeur négative en entrée (type de sortie 2 seulement).

14.2.4 Paramètres

- (FLOAT) pwmgen.<chan>.scale – Facteur d'échelle pour convertir les valeurs en unités arbitraires, en coefficients de facteur cyclique.
- (FLOAT) pwmgen.<chan>.pwm-freq – Fréquence de PWM désirée, en Hz. Si égale à 0.0, la modulation sera PDM au lieu de PWM. Si elle est réglée plus haute que les limites internes, au prochain appel de la fonction update_freq() elle sera ramenée aux limites internes. Si elle est différente de zéro et si le lissage est faux, au prochain appel de la fonction update_freq() elle sera réglée au plus proche entier multiple de la période de la fonction make_pulses().
- (BIT) pwmgen.<chan>.dither-pwm – Si vrai, active le lissage pour affiner la fréquence PWM ou le rapport cyclique qui ne pourraient pas être obtenus avec une pure PWM. Si faux, la fréquence PWM et le rapport cyclique seront tous les deux arrondis aux valeurs pouvant être atteintes exactement.
- (FLOAT) pwmgen.<chan>.min-dc – Rapport cyclique minimum compris entre 0.0 et 1.0 (Le rapport cyclique sera à zéro quand il est désactivé, indépendamment de ce paramètre).
- (FLOAT) pwmgen.<chan>.max-dc – Rapport cyclique maximum compris entre 0.0 et 1.0.
- (FLOAT) pwmgen.<chan>.curr-dc – Rapport cyclique courant, après toutes les limitations et les arrondis (lecture seule).

14.2.5 Types de sortie

Le générateur de PWM supporte trois "types de sortie". Le type 0 a une seule pin de sortie. Seules, les commandes positives sont acceptées, les valeurs négatives sont traitées comme zéro (elle seront affectées par min-dc si il est différent de zéro). Le type 1 a deux pins de sortie, une pour le signal PWM/PDM et une pour indiquer la direction. Le rapport cyclique d'une pin PWM est basé sur la valeur absolue de la commande, de sorte que les valeurs négatives sont acceptables. La pin de direction est fausse pour les commandes positives et vraie pour les commandes négatives. Finalement, le type 2 a également deux sorties, appelées "up" et "down". Pour les commandes positives, le signal PWM apparaît sur la sortie up et la sortie down reste fausse. Pour les commandes négatives, le signal PWM apparaît sur la sortie down et la sortie up reste fausse. Les sorties de type 2 sont appropriées pour piloter la plupart des ponts en H.

14.2.6 Fonctions

Le composant exporte deux fonctions. Chaque fonction agit sur tous les générateurs de PWM, lancer différents générateurs dans différents threads n'est pas supporté.

- (FUNCT) pwmgen.make-pulses – Fonction haute vitesse de génération de fréquences PWM (non flottant).
- (FUNCT) pwmgen.update – Fonction basse vitesse de mise à l'échelle, limitation des valeurs et traitement d'autres paramètres.

La fonction haute vitesse "pwmgen.make-pulses" devrait être lancée dans un thread très rapide, entre 10 et 50s selon les capacités de l'ordinateur. C'est la période de ce thread qui détermine la fréquence maximale de la porteuse PWM, ainsi que la résolution des signaux PWM ou PDM. L'autre fonction peut être appelée beaucoup plus lentement.

14.3 Codeur

Ce composant fournit, en logiciel, le comptage des signaux provenant d'encodeurs en quadrature. Il s'agit d'un composant temps réel uniquement, il est dépendant de divers facteurs comme la vitesse du CPU, etc, il est capable de compter des signaux de fréquences comprises entre 10kHz à peut être 50kHz. La figure 14.7 est le diagramme bloc d'un canal de comptage de codeur.

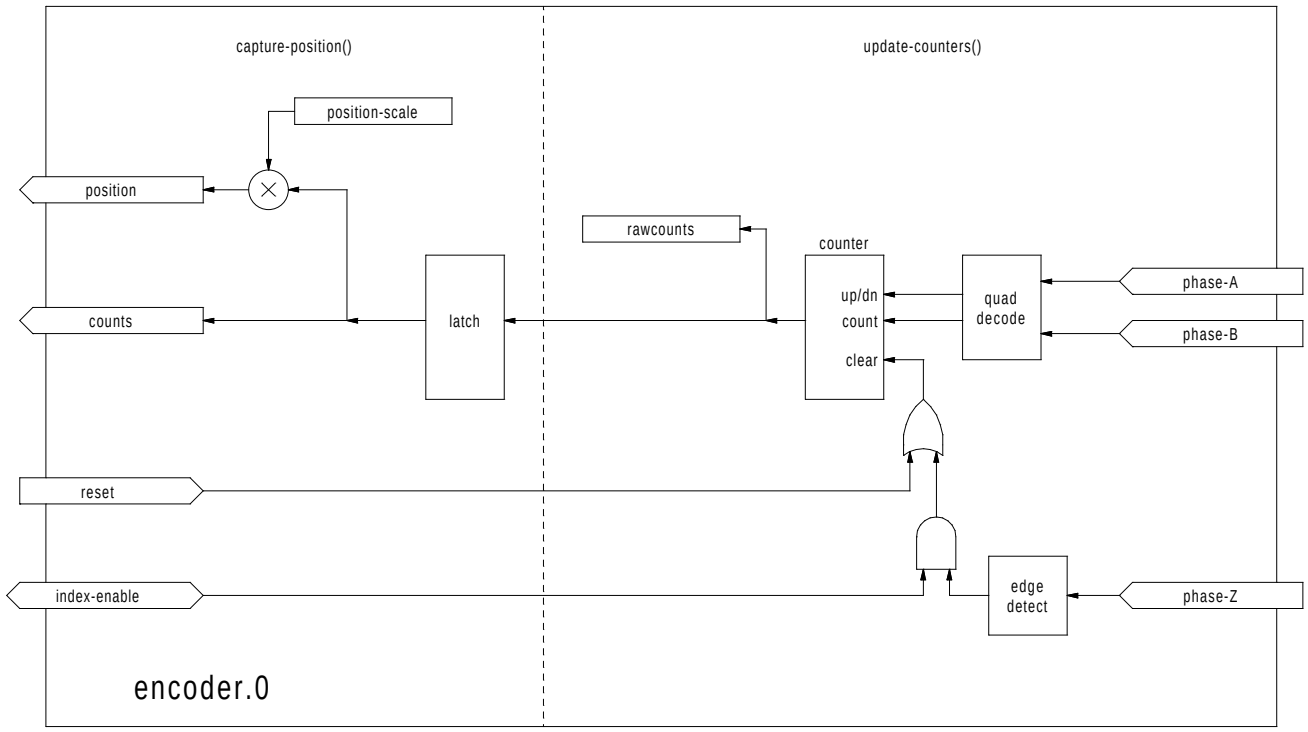


Figure 14.7: Diagramme bloc du compteur de codeur

14.3.1 L'installer

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

`<counters>` est le nombre de compteurs de codeur à installer. Si `numchan` n'est pas spécifié, trois compteurs seront installés. Le nombre maximum de compteurs est de 8 (comme définit par MAX_CHAN dans encoder.c). Chaque compteur est indépendant, mais tous sont mis à jour par la même fonction(s) au même instant. Dans les descriptions qui suivent, `<chan>` est le nombre de compteurs spécifiques. La numérotation des compteurs commence à 0.

14.3.2 Le désinstaller

```
emc2$ halcmd unloadrt encoder
```

14.3.3 Pins

- (BIT) encoder.<chan>.phase-A – Signal de la phase A du codeur en quadrature.
- (BIT) encoder.<chan>.phase-B – Signal de la phase B du codeur en quadrature.
- (BIT) encoder.<chan>.phase-Z – Signal de la phase Z (impulsion d'ndex) du codeur en quadrature.
- (BIT) encoder.<chan>.reset – Voir l'interface canonique des codeurs à la section 9.5.
- (BIT) encoder.<chan>.velocity – Vitesse estimée du signal de quadrature.
- (BIT) encoder.<chan>.index-enable – Voir l'interface canonique des codeurs.
- (S32) encoder.<chan>.count – Voir l'interface canonique des codeurs.
- (FLOAT) encoder.<chan>.position – Voir l'interface canonique des codeurs.

14.3.4 Paramètres

- (S32) encoder.<chan>.raw-count – Valeur de comptage brute, actualisée par la fonction update-counters().
- (BIT) encoder.<chan>.x4-mode – Ajuste le comptage en mode 4x ou 1x. Le mode 1x est intéressant pour les manivelles de jog.
- (FLOAT) encoder.<chan>.position-scale – Voir l'interface canonique des codeurs à la section 9.5.

14.3.5 Fonctions

Le composant exporte deux fonctions. Chaque fonction agit sur tous les compteurs de codeur, lancer différents compteurs de codeur dans différents threads n'est pas supporté.

- (FUNCT) encoder.update-counters – Fonction haute vitesse de comptage d'impulsions (non flottant).
- (FUNCT) encoder.capture-position – Fonction basse vitesse d'actualisation des latches et mise à l'échelle de la position.

14.4 PID

Ce composant fournit une boucle de contrôle Proportionnel/Intégrale/Dérivée. C'est un composant temps réel uniquement. Par souci de simplicité, cette discussion suppose que nous parlons de boucles de position, mais ce composant peut aussi être utilisé pour implémenter d'autres boucles de rétroaction telles que vitesse, hauteur de torche, température, etc. La figure 14.8 est le schéma fonctionnel d'une simple boucle PID.

14.4.1 L'installer

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

<loops> est le nombre de boucles PID à installer. Si numchan n'est pas spécifié, une seule boucle sera installée. Le nombre maximum de boucles est de 16 (comme définit par MAX_CHAN dans pid.c). Chaque boucle est complétement indépendante. Dans les descriptions qui suivent, <loopnum> est le nombre de boucles spécifiques. La numérotation des boucle PID commence à 0.

Si debug=1 est spécifié, le composant exporte quelques paramètres destinés au déboguage et aux réglages. Par défaut, ces paramètres ne sont pas exportés, pour économiser la mémoire partagée et éviter d'encombrer la liste des paramètres.

14.4.2 Le désinstaller

```
emc2$ halcmd unloadrt pid
```

14.4.3 Pins

Les trois principales pins sont:

- (FLOAT) pid.<loopnum>.command – La position désirée (consigne), telle que commandée par un autre composant système.
- (FLOAT) pid.<loopnum>.feedback – La position actuelle (mesure), telle que mesurée par un organe de rétroaction comme un codeur de position.
- (FLOAT) pid.<loopnum>.output – Une commande de vitesse qui tend à aller de la position actuelle à la position désirée.

Pour une boucle de position, 'command' et 'feedback' sont en unités de longueur. Pour un axe linéaire, cela pourrait être des pouces, mm, mètres, ou tout autre unité pertinente. De même pour un axe angulaire, ils pourraient être des degrés, radians, etc. Les unités sur la pin 'output' représentent l'écart nécessaire pour que la rétroaction coïncide avec la commande. Pour une boucle de position, 'output' est une vitesse exprimée en pouces/seconde, mm/seconde, degrés/seconde, etc. Les unités de temps sont toujours des secondes et les unités de vitesses restent cohérentes avec les unités de longueur. Si la commande et la rétroaction sont en mètres, la sortie sera en mètres par seconde.

Chaque boucle PID a deux autres pins qui sont utilisées pour surveiller ou contrôler le fonctionnement général du composant.

- (FLOAT) pid.<loopnum>.error – Egal à .command moins .feedback. (consigne - mesure)
- (BIT) pid.<loopnum>.enable – Un bit qui active la boucle. Si .enable est faux, tous les intégrateurs sont remis à zéro et les sorties sont forcées à zéro. Si .enable est vrai, la boucle opère normalement.

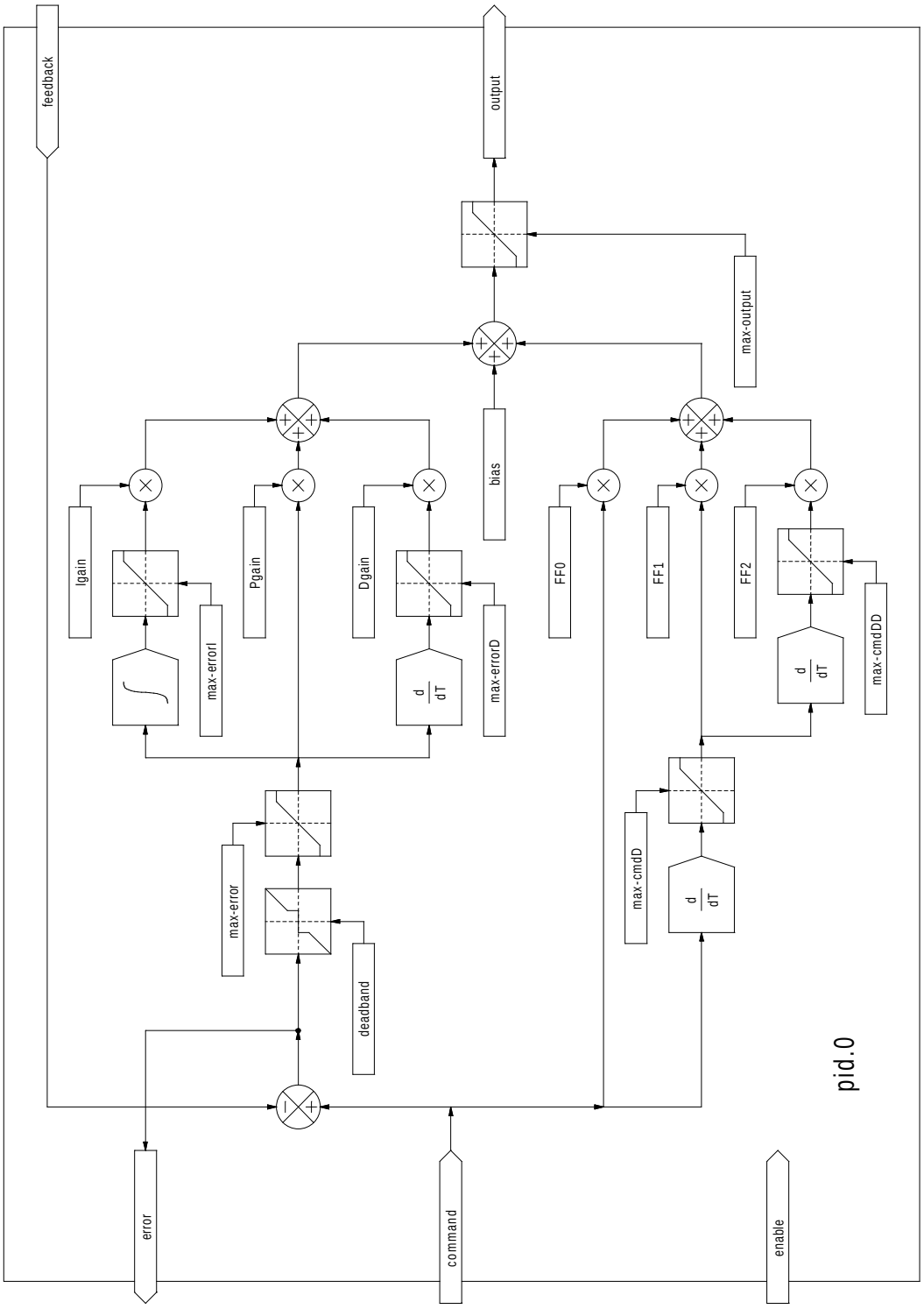


Figure 14.8: Diagramme bloc d'une boucle PID

14.4.4 Paramètres

Le gain PID, les limites et autres caractèristiques 'accordables' de la boucle sont implémentés comme des paramètres.

- (FLOAT) pid.<loopnum>.Pgain – Gain de la composante proportionnelle
- (FLOAT) pid.<loopnum>.Igain – Gain de la composante intégrale
- (FLOAT) pid.<loopnum>.Dgain – Gain de la composante dérivée
- (FLOAT) pid.<loopnum>.bias – Constante du décalage de sortie
- (FLOAT) pid.<loopnum>.FF0 – Correcteur prédictif d'ordre zéro (feedforward) - sortie proportionnelle à la commande (position).
- (FLOAT) pid.<loopnum>.FF1 – Correcteur prédictif de premier ordre (feedforward) - sortie proportionnelle à la dérivée de la commande (vitesse).
- (FLOAT) pid.<loopnum>.FF2 – Correcteur prédictif de second ordre (feedforward) - sortie proportionnelle à la dérivée seconde de la commande (accélération)[1].
- (FLOAT) pid.<loopnum>.deadband – Définit la bande morte (tolérance)
- (FLOAT) pid.<loopnum>.maxerror – Limite d'erreur
- (FLOAT) pid.<loopnum>.maxerrorI – Limite d'erreur intégrale
- (FLOAT) pid.<loopnum>.maxerrorD – Limite d'erreur dérivée
- (FLOAT) pid.<loopnum>.maxcmdD – Limite de la commande dérivée
- (FLOAT) pid.<loopnum>.maxcmdDD – Limite de la commande dérivée seconde
- (FLOAT) pid.<loopnum>.maxoutput – Limite de la valeur de sortie

Toutes les limites max???, sont implémentées de sorte que si la valeur de ce paramètre est à zéro, il n'y a pas de limite.

Si debug=1 est spécifié quand le composant est installé, quatre paramètres supplémentaires seront exportés:

- (FLOAT) pid.<loopnum>.errorI – Intégrale de l'erreur.
- (FLOAT) pid.<loopnum>.errorD – Dérivée de l'erreur.
- (FLOAT) pid.<loopnum>.commandD – Dérivée de la commande.
- (FLOAT) pid.<loopnum>.commandDD – Dérivée seconde de la commande.

14.4.5 Fonctions

Le composant exporte une fonction pour chaque boucle PID. Cette fonction exécute tous les calculs nécessaires à la boucle. Puisque chaque boucle a sa propre fonction, les différentes boucles peuvent être incluses dans les différents threads et exécutées à différents rythmes.

- (FUNCT) pid.<loopnum>.do_pid_calcs – Exécute tous les calculs d'une seule boucle PID.

Si vous voulez comprendre exactement l'algorithme utilisé pour calculer la sortie d'une boucle PID, référez vous à la figure 14.8, les commentaires au début du source emc2/src/hal/components/pid.c et bien sûr, au code lui même. Les calculs de boucle sont dans la fonction C calc_pid().

[1] FF2 n'est actuellement pas implémenté, mais il pourrait l'être. Considérez cette note "FIXME" dans le code.

14.5 Codeur simulé

Le codeur simulé est exactement la même chose qu'un codeur. Il produit des impulsions en quadrature avec une impulsion d'index, à une vitesse contrôlée par une pin de HAL. Surtout utile pour les essais.

14.5.1 L'installer

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

<number> est le nombre de codeurs à simuler. Si aucun n'est spécifié, un seul codeur sera installé. Le nombre maximum de codeurs est de 8 (comme définit par MAX_CHAN dans sim_encoder.c).

14.5.2 Le désinstaller

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

14.5.3 Pins

- (FLOAT) sim-encoder.<chan-num>.speed – La vitesse commandée pour l'arbre simulé.
- (BIT) sim-encoder.<chan-num>.phase-A – Sortie en quadrature.
- (BIT) sim-encoder.<chan-num>.phase-B – Sortie en quadrature.
- (BIT) sim-encoder.<chan-num>.phase-Z – Sortie de l'impulsion d'index.

Quand .speed est positive, .phase-A mène .phase-B.

14.5.4 Paramètres

- (U32) sim-encoder.<chan-num>.ppr – Impulsions par tour d'arbre.
- (FLOAT) sim-encoder.<chan-num>.scale – Facteur d'échelle pour **speed**. Par défaut est de 1.0, ce qui signifie que **speed** est en tours par seconde. Passer l'échelle à 60 pour des tours par minute, la passer à 360 pour des degrés par seconde, à 6.283185 pour des radians par seconde, etc.

Noter que les impulsions par tour ne sont pas identiques aux valeurs de comptage par tour (counts). Une impulsion est un cycle complet de quadrature. La plupart des codeurs comptent quatre fois pendant un cycle complet.

14.5.5 Fonctions

Le composant exporte deux fonctions. Chaque fonction affecte tous les codeurs simulés.

- (FUNCT) sim-encoder.make-pulses – Fonction haute vitesse de génération d'impulsions en quadrature (non flottant).
- (FUNCT) sim-encoder.update-speed – Fonction basse vitesse de lecture de **speed**, de mise à l'échelle et d'activation de **make-pulses**.

14.6 Anti-rebond

L'anti-rebond est un composant temps réel capable de filtrer les rebonds créés par les contacts mécaniques. Il est également très utile dans d'autres applications, où des impulsions très courtes doivent être supprimées.

14.6.1 L'installer

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

<config-string> est une série d'entiers décimaux séparés par des espaces. Chaque chiffre installe un groupe de filtres anti-rebond identiques, le chiffre détermine le nombre de filtres dans le groupe. Par exemple:

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

va installer trois groupes de filtres. Le groupe 0 contient un filtre, le groupe 1 en contient quatre et le groupe 2 en contient deux. La valeur par défaut de <config-string> est "1" qui installe un seul groupe contenant un seul filtre. Le nombre maximum de groupes est de 8 (comme définit par MAX_GROUPS dans debounce.c). Le nombre maximum de filtres dans un groupe est limité seulement par l'espace de la mémoire partagée. Chaque groupe est complétement indépendant. Tous les filtres dans un même groupe sont identiques et ils sont tous mis à jour par la même fonction, au même instant. Dans les descriptions qui suivent, <G> est le numéro du groupe et <F> est le numéro du filtre dans le groupe. Le premier filtre est le filtre 0 dans le groupe 0.

14.6.2 Le désinstaller

```
emc2$ halcmd unloadrt debounce
```

14.6.3 Pins

Chaque filtre individuel a deux pins.

- (BIT) debounce.<G>.<F>.in – Entrée du filtre <F> du groupe <G>.
- (BIT) debounce.<G>.<F>.out – Sortie du filtre <F> du groupe <G>.

14.6.4 Paramètres

Chaque groupe de filtre a un paramètre[2].

- (s32) debounce.<G>.delay – Délai de filtrage pour tous les filtres du groupe <G>.

Le délai du filtre est dans l'unité de la période du thread. Le délai minimum est de zéro. La sortie d'un filtre avec un délai de zéro, suit exactement son entrée, il ne filtre rien. Plus le délai augmente, plus larges seront les impulsions rejetées. Si le délai est de 4, toutes les impulsions égales ou inférieures à quatre périodes du thread, seront rejetées.

[2]Chaque filtre individuel a également une variable d'état interne. C'est un switch du compilateur qui peut exporter cette variable comme un paramètre. Ceci est prévu pour des essais et devrait juste être un gaspillage de mémoire partagée dans des circonstances normales.

14.6.5 Fonctions

Chaque groupe de filtres exporte une fonction qui met à jour tous les filtres de ce groupe “simultanément”. Différents groupes de filtres peuvent être mis à jour dans différents threads et à différentes périodes.

- (FUNCT) debounce.<G> – Met à jour tous les filtres du groupe <G>.

14.7 Siggen

Siggen est un composant temps réel qui génère des signaux carrés, triangulaires et sinusoïdaux. Il est principalement utilisé pour les essais.

14.7.1 L’installer

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

<chans> est le nombre de générateurs de signaux à installer. Si numchan n’est pas spécifié, un seul générateur de signaux sera installé. Le nombre maximum de générateurs est de 16 (comme définit par MAX_CHAN dans siggen.c). Chaque générateur est complétement indépendant. Dans les descriptions qui suivent, <chan> est le numéro d’un générateur spécifique. Les numéros de générateur commencent à 0.

14.7.2 Le désinstaller

```
emc2$ halcmd unloadrt siggen
```

14.7.3 Pins

Chaque générateur a cinq pins de sortie.

- (FLOAT) siggen.<chan>.sine – Sortie de l’onde sinusoïdale.
- (FLOAT) siggen.<chan>.cosine – Sortie de l’onde cosinusoïdale.
- (FLOAT) siggen.<chan>.sawtooth – Sortie de l’onde en dents de scie.
- (FLOAT) siggen.<chan>.triangle – Sortie de l’onde triangulaire.
- (FLOAT) siggen.<chan>.square – Sortie de l’onde carrée.

Les cinq sorties ont les mêmes fréquence, amplitude et offset.

Trois pins de contrôle s’ajoutent aux pins de sortie:

- (FLOAT) siggen.<chan>.frequency – Réglage de la fréquence en Hertz, par défaut la valeur est de 1 Hz.
- (FLOAT) siggen.<chan>.amplitude – Réglage de l’amplitude de pic des signaux de sortie, par défaut, est à 1.
- (FLOAT) siggen.<chan>.offset – Réglage de la composante continue des signaux de sortie, par défaut, est à 0.

Par exemple, si siggen.0.amplitude est à 1.0 et siggen.0.offset est à 0.0, les sorties oscilleront entre -1.0 et +1.0. Si siggen.0.amplitude est à 2.5 et siggen.0.offset est à 10.0, les sorties oscilleront entre 7.5 et 12.5.

14.7.4 Paramètres

Aucun. [3]

14.7.5 Fonctions

- (FUNCT) siggen.<chan>.update – Calcule les nouvelles valeurs pour les cinq sorties.

[3] Dans les versions antérieures à la 2.1, fréquence, amplitude et offset étaient des paramètres. Ils ont été modifiés en pins pour permettre le contrôle par d'autres composants.

Chapter 15

Pilotes matériels

15.1 Parport

Parport est un pilote pour le port parallèle traditionnel des PC. Le port dispose d'un total de 17 broches physiques. Le port parallèle originel a divisé ces broches en trois groupes: données, contrôle et état. Le groupe "données" consiste en 8 broches de sortie, le groupe "contrôle" consiste en 4 broches et le groupe "état" consiste en 5 broches d'entrée.

Au début des années 1990, le port parallèle bidirectionnel est arrivé, ce qui permet à l'utilisateur d'ajuster le groupe des données comme étant des sorties ou comme étant des entrées. Le pilote de HAL supporte le port bidirectionnel et permet à l'utilisateur de configurer le groupe des données en entrée ou en sortie. Si il est configuré en sortie, un port fournit un total de 12 sorties et 5 entrées. Si il est configuré en entrée, il fournit 4 sorties et 13 entrées.

Dans certains ports parallèles, les broches du groupe contrôle sont des collecteurs ouverts, ils peuvent aussi être mis à l'état bas par une porte extérieure. Sur une carte avec les broches de contrôle en collecteur ouvert, le mode "x" de HAL permet un usage plus flexible avec 8 sorties dédiées, 5 entrées dédiées et 4 broches en collecteur ouvert. Dans d'autres ports parallèles, les broches du groupe contrôle sont en push-pull et ne peuvent pas être utilisées comme des entrées.[1]

Aucune autre combinaison n'est supportée. Un port ne peut plus être modifié pour passer d'entrées en sorties dès que le pilote est installé. La figure 15.1 montre deux diagrammes, un montre le pilote quand le groupe de données est configuré en sortie et le second le montre configuré en entrée.

Le pilote parport peut contrôler au maximum 8 ports (défini par MAX_PORTS dans le fichier hal_parport.c). Les ports sont numérotés à partir de zéro.

15.1.1 Installation

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

[1] HAL ne peut pas déterminer automatiquement si les broches en mode bidirectionnel "x" sont effectivement en collecteur ouvert (OC). Si elles n'y sont pas, elles ne peuvent pas être utilisées comme des entrées. Essayer de les passer à l'état BAS par une source extérieure peut détériorer le matériel.

Pour déterminer si votre port a des broches de contrôle en "collecteur ouvert", charger hal_parport en mode "x", positionner les broches de contrôle à une valeur HAUTE. HAL doit lire des pins à l'état VRAI. Ensuite, insérer une résistance de 470Ω entre une des broches de contrôle et GND du port parallèle. Si la tension de cette broche de contrôle est maintenant proche de 0V et que HAL la lit comme une pin FAUSSE, alors vous avez un port OC. Si la tension résultante est loin de 0V ou que HAL ne la lit pas comme étant FAUSSE, votre port ne ne peut pas être utilisé en mode "x".

Le matériel extérieur qui pilote les broches de contrôle devrait également utiliser des portes en collecteur ouvert (ex: 74LS05...). Généralement, une pin de HAL -out devrait être VRAIE quand la pin physique est utilisée comme une entrée.

Sur certaines machines, les paramètres du BIOS peuvent affecter la possibilité d'utiliser le mode "x". Le mode "SPP" est le mode qui fonctionne le plus fréquemment.

La chaine config-string représente l'adresse hexadécimale du port, suivie, optionnellement, par une direction, le tout répété pour chaque port. Les directions sont "in", "out", ou "x", elles déterminent la direction des broches physiques 2 à 9 et s'il y a lieu de créer des pins d'entrée de HAL pour les broches de contrôle physiques. Si la direction n'est pas précisée, le groupe données sera par défault en sortie. Par exemple:

```
loadrt hal_parport cfg="0x278 0x378 in 0x20A0 out"
```

Cet exemple installe les pilotes pour un port 0x0278, avec les broches 2 à 9 en sorties (par défaut, puisque ni " in ", ni "out" n'est spécifié), un port 0x0378, avec les broches 2 à 9 en entrées et un port 0x20A0, avec les broches 2 à 9 explicitement spécifiées en sorties. Notez que vous devez connaître l'adresse de base des ports parallèles pour configurer correctement les pilotes. Pour les ports sur bus ISA, ce n'est généralement pas un problème, étant donné que les ports sont presque toujours à une adresse "bien connue", comme 0278 ou 0378 qui sont typiquement configurées dans le BIOS. Les adresses des cartes sur bus PCI sont habituellement trouvées avec "lspci -v" dans une ligne "I/O ports", ou dans un message du kernel après l'exécution de "`sudo modprobe -a parport_pc`". Il n'y a pas d'adresse par défaut, si `<config-string>` ne contient pas au moins une adresse, un message d'erreur s'affichera.

15.1.2 Pins

- `(BIT) parport.<portnum>.pin-<pinnum>-out` – Pilote une pin de sortie physique.
- `(BIT) parport.<portnum>.pin-<pinnum>-in` – Suit une pin d'entrée physique.
- `(BIT) parport.<portnum>.pin-<pinnum>-in-not` – Suit une pin d'entrée physique, mais inversée.

Pour chaque pin, `<portnum>` est le numéro du port et `<pinnum>` est le numéro de la broche physique du connecteur DB-25.

Pour chaque broche de sortie physique, le pilote crée une simple pin de HAL, par exemple `parport.0.pin-14-ou`
Les pins 2 jusqu'à 9 font partie du groupe donnée est sont des pins de sortie si le port est défini comme un port de sortie (par défaut en sortie). Les broches 1, 14, 16 et 17 sont des sorties dans tous les modes. Ces pin de HAL contrôlent l'état des pins physiques correspondantes.

Pour chaque pin d'entrée physique, le pilote crée deux pins de HAL, par exemple `parport.0.pin-12-in` et `parport.0.pin-12-in-not`. Les pins 10, 11, 12, 13 et 15 sont toujours des sorties. Les pins 2 jusqu'à 9 sont des pins d'entrée seulement si le port est défini comme un port d'entrée. Une pin de HAL `-in` est VRAIE si la pin physique est haute et FAUSSE si la pin physique est basse. Une pin de HAL `-in-not` est invertsée, elle est FAUSSE si la pin physique est haute. En connectant un signal à l'une ou l'autre, l'utilisateur peut décider de l'état de l'entrée. En mode "x", les pins 1, 14, 16 et 17 sont également des pins d'entrée.

15.1.3 Paramètres

- `(BIT) parport.<portnum>.pin-<pinnum>-out-invert` – Inverse une pin de sortie.
- `(BIT) parport.<portnum>.pin-<pinnum>-out-reset` (seulement les pins 2..9) – VRAIE si cette pin doit être réinitialisée quand la fonction de réinitialisation est exécutée.
- `(U32) parport.<portnum>.reset-time` – Le temps (en nanosecondes) entre le moment ou la broche est écrite et le moment ou elle est réititialisée par les fonctions de réinitialisation de HAL.

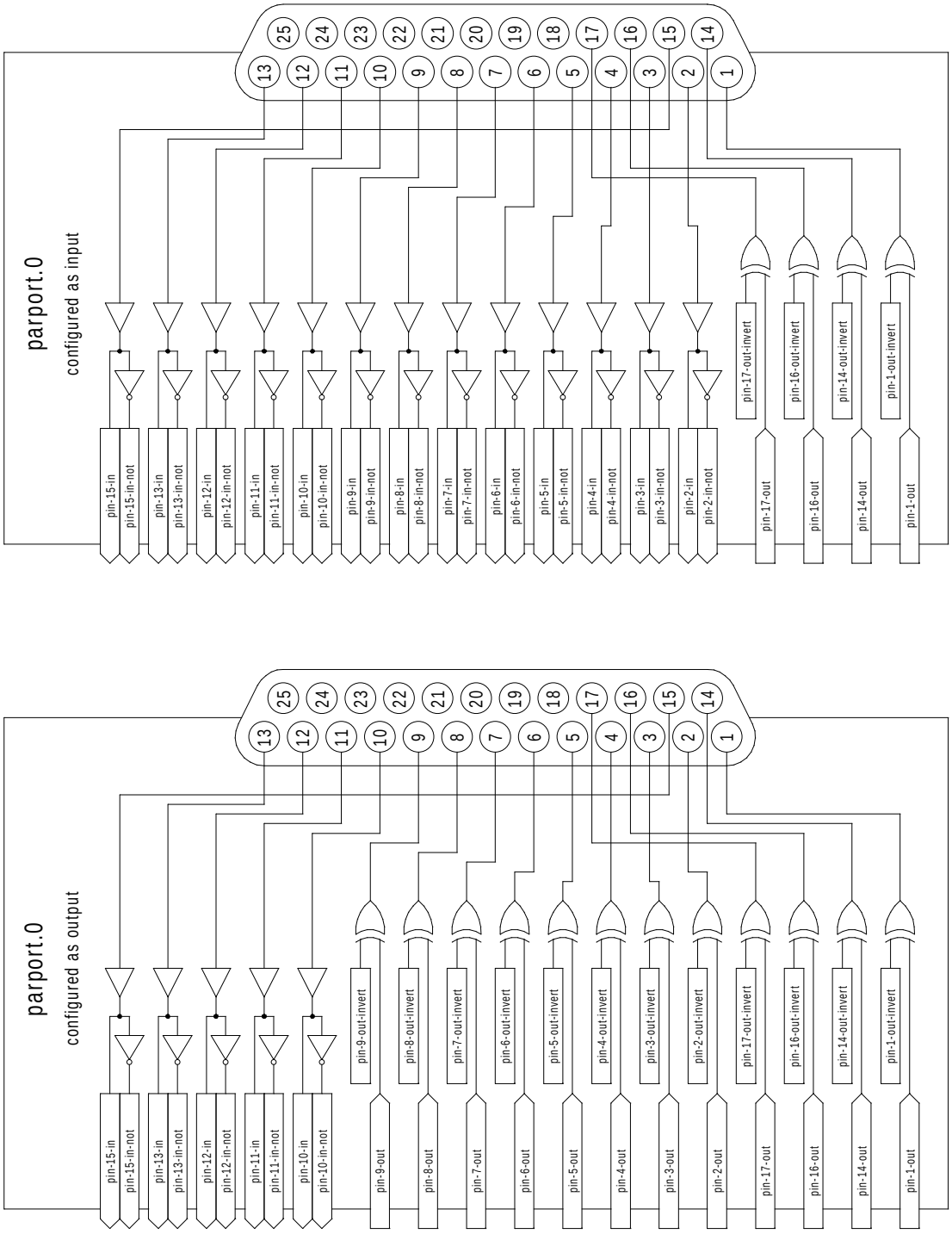


Figure 15.1: Diagrammes blocs de Parport

Le paramètre -invert détermine si une pin de sortie est active haute ou active basse. Si -invert est FAUX, mettre la pin HAL -out VRAIE placera la pin physique à l'état haut et mettre la pin HAL FAUSSE placera la pin physique à l'état bas. Si -invert est VRAI, mettre la pin HAL -out VRAIE va mettre la pin physique à l'état bas.

Si -reset est VRAI, la fonction de réinitialisation va passer la pin à la valeur de -out-invert. Ceci peut être utilisé en conjonction avec "stepgen doublefreq" pour produire un pas par période.

15.1.4 Fonctions

- (FUNCT) parport.<portnum>.read – Lit les pins physiques du port <portnum> et met à jour les pins de HAL -in et -in-not.
- (FUNCT) parport.read-all – Lit les pins physiques de tous les ports et met à jour les pins de HAL -in et -in-not.
- (FUNCT) parport.<portnum>.write – Lit les pins de HAL -out du port <portnum> et met à jour les pins de sortie physiques correspondantes.
- (FUNCT) parport.write-all – Lit les pins de HAL -out de tous les ports et met à jour toutes les pins de sortie physiques.
- (FUNCT) parport.<portnum>.reset – Attends que le délai de mise à jour soit écoulé depuis le dernière écriture, remet à jour les pins aux valeurs indiquées par -out-invert et les paramètres de -out-invert. La réinitialisation doit être plus tard dans le même thread que l'écriture.

Les différentes fonctions sont prévues pour les situations où un port doit être mis à jour dans un thread très rapide, mais d'autres ports peuvent être mis à jour dans un thread plus lent pour gagner du temps CPU. Ce n'est probablement pas une bonne idée d'utiliser en même temps, les fonctions -all et une fonction individuelle.

15.1.5 Problèmes courants

Si le chargement du module le message suivant:

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

s'assurer que le noyau du kernel standard, parport_pc, n'est pas chargé[2] et qu'aucun périphérique dans le système ne revendique les ports concernés.

SI le module est chargé mais ne semble pas fonctionner, l'adresse du port est incorrecte ou le module probe_parport est requis.

15.2 probe_parport

Dans les PC récents, le port parallèle peut exiger une configuration plug and play (PNP) avant d'être utilisable. Le module probe_parport effectue la configuration de tous les ports PNP présents et devrait être chargé avant hal_parport. Sur les machines sans ports PNP, il peut être chargé mais n'a aucun effet.

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

15.2.1 Installation

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

Si le kernel Linux affiche un message similaire à:

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

quand le module parport_pc est chargé, avec la commande: `sudo modprobe -a parport_pc; sudo rmmod parport_pc`, l'utilisation de ce module sera probablement nécessaire.

.

Les modules commerciaux ci-dessous pouront êtres traduits en français sur demande.

The commercial modules below could be translated into French on request.

.

15.3 AX5214H

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

15.3.1 Installing

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

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

15.3.2 Pins

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

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

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

[3] In fact it may be a pair of 8255 chips, but I'm not sure. If/when someone starts a driver for an 8255 they should look at the ax5214 code, much of the work is already done.

15.3.3 Parameters

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

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

15.3.4 Functions

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

15.4 Servo-To-Go

The Servo-To-Go is one of the first PC motion control cards[4] supported by EMC. It is an ISA card and it exists in different flavours (all supported by this driver). The board includes up to 8 channels of quadrature encoder input, 8 channels of analog input and output, 32 bits digital I/O, an interval timer with interrupt and a watchdog.

15.4.1 Installing:

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

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

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

This example installs the stg driver for a card found at the base address of 0x300, 4 channels of encoder feedback, DAC's and ADC's, along with 32 bits of I/O configured like this: the first 8 (Port A) configured as Input, the next 8 (Port B) configured as Output, the next 8 (Port C) configured as Input, and the last 8 (Port D) configured as Output

```
loadrt hal_stg
```

This example installs the driver and attempts to autodetect the board address and board model, it installs 8 axes by default along with a standard I/O setup: Port A & B configured as Input, Port C & D configured as Output.

[4] a motion control card usually is a board containing devices to control one or more axes (the control devices are usually DAC's to set an analog voltage, encoder counting chips for feedback, etc.)

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

15.4.2 Pins

- (s32) stg.<channel>.counts – Tracks the counted encoder ticks.
- (FLOAT) stg.<channel>.position – Outputs a converted position.
- (FLOAT) stg.<channel>.dac-value – Drives the voltage for the corresponding DAC.
- (FLOAT) stg.<channel>.adc-value – Tracks the measured voltage from the corresponding ADC.
- (BIT) stg.in-<pinnum> – Tracks a physical input pin.
- (BIT) stg.in-<pinnum>-not – Tracks a physical input pin, but inverted.
- (BIT) stg.out-<pinnum> – Drives a physical output pin

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

The in- HAL pin is TRUE if the physical pin is high, and FALSE if the physical pin is low. The in-<pinnum>-not HAL pin is inverted – it is FALSE if the physical pin is high. By connecting a signal to one or the other, the user can determine the state of the input.

15.4.3 Parameters

- (FLOAT) stg.<channel>.position-scale – The number of counts / user unit (to convert from counts to units).
- (FLOAT) stg.<channel>.dac-offset – Sets the offset for the corresponding DAC.
- (FLOAT) stg.<channel>.dac-gain – Sets the gain of the corresponding DAC.
- (FLOAT) stg.<channel>.adc-offset – Sets the offset of the corresponding ADC.
- (FLOAT) stg.<channel>.adc-gain – Sets the gain of the corresponding ADC.
- (BIT) stg.out-<pinnum>-invert – Inverts an output pin.

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

15.4.4 Functions

- (FUNCT) stg.capture-position – Reads the encoder counters from the axis <channel>.
- (FUNCT) stg.write-dacs – Writes the voltages to the DACs.
- (FUNCT) stg.read-adcs – Reads the voltages from the ADCs.
- (FUNCT) stg.di-read – Reads physical in- pins of all ports and updates all HAL in- and in-<pinnum>-not pins.
- (FUNCT) stg.do-write – Reads all HAL out- pins and updates all physical output pins.

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

15.5 Mesa Electronics m5i20 "Anything I/O Card"

The Mesa Electronics m5i20 card consists of an FPGA that can be loaded with a wide variety of configurations, and has 72 pins that leave the PC. The assignment of the pins depends on the FPGA configuration. Currently there is a HAL driver for the "4 axis host based motion control" configuration, and this FPGA configurations is also provided with EMC2. It provides 8 encoder counters, 4 PWM outputs (normally used as DACs) and up to 48 digital I/O channels, 32 inputs and 16 outputs.[7]

Installing:

```
loadrt hal_m5i20 [loadFpga=1|0] [dacRate=<rate>]
```

If **loadFpga** is 1 (the default) the driver will load the FPGA configuration on startup. If it is 0, the driver assumes the configuration is already loaded. **dacRate** sets the carrier frequency for the PWM outputs, in Hz. The default is 32000, for 32KHz PWM. Valid values are from 1 to 32226. The driver prints some useful debugging message to the kernel log, which can be viewed with **dmesg**.

Up to 4 boards may be used in one system.

15.5.1 Pins

In the following pins, parameters, and functions, <board> is the board ID. According to the naming conventions the first board should always have an ID of zero, however this driver uses the PCI board ID, so it may be non-zero even if there is only one board.

- (S32) m5i20.<board>.enc-<channel>-count – Encoder position, in counts.
- (FLOAT) m5i20.<board>.enc-<channel>-position – Encoder position, in user units.
- (BIT) m5i20.<board>.enc-<channel>-index – Current status of index pulse input?
- (BIT) m5i20.<board>.enc-<channel>-index-enable – when TRUE, and an index pulse appears on the encoder input, reset counter to zero and clear index-enable.
- (BIT) m5i20.<board>.enc-<channel>-reset – When true, counter is forced to zero.
- (BIT) m5i20.<board>.dac-<channel>-enable – Enables DAC if true. DAC outputs zero volts if false?
- (FLOAT) m5i20.<board>.dac-<channel>-value – Analog output value for PWM "DAC" (in user units, see -scale and -offset)
- (BIT) m5i20.<board>.in-<channel> – State of digital input pin, see canonical digital input.
- (BIT) m5i20.<board>.in-<channel>-not – Inverted state of digital input pin, see canonical digital input.
- (BIT) m5i20.<board>.out-<channel> – Value to be written to digital output, see canonical digital output.
- (BIT) m5i20.<board>.estop-in – Dedicated estop input, more details needed.
- (BIT) m5i20.<board>.estop-in-not – Inverted state of dedicated estop input.
- (BIT) m5i20.<board>.watchdog-reset – Bidirectional, - Set TRUE to reset watchdog once, is automatically cleared. If bit value 16 is set in watchdog-control then this value is not used, and the hardware watchdog is cleared every time the dac-write function is executed.

[7] Ideally the encoders, "DACs", and digital I/O would comply with the canonical interfaces defined earlier, but they don't. Fixing that is on the things-to-do list.

15.5.2 Parameters

- (FLOAT) m5i20.<board>.enc-<channel>-scale – The number of counts / user unit (to convert from counts to units).
- (FLOAT) m5i20.<board>.dac-<channel>-offset – Sets the DAC offset.
- (FLOAT) m5i20.<board>.dac-<channel>-gain – Sets the DAC gain (scaling).
- (BIT) m5i20.<board>.dac-<channel>-interlaced – Sets the DAC to interlaced mode. Use this mode if you are filtering the PWM to generate an anaolg voltage.[8]
- (BIT) m5i20.<board>.out-<channel>-invert – Inverts a digital output, see canonical digital output.
- (U32) m5i20.<board>.watchdog-control – Configures the watchdog. The value may be a bitwise OR of the following values:

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

  Typically, the useful values are 0 (watchdog disabled) or 3 (watchdog enabled, cleared by dac-write).
- (U32) m5i20.<board>.led-view – Maps some of the I/O to onboard LEDs. See table below.

15.5.3 Functions

- (FUNCT) m5i20.<board>.encoder-read – Reads all encoder counters.
- (FUNCT) m5i20.<board>.digital-in-read – Reads digital inputs.
- (FUNCT) m5i20.<board>.dac-write – Writes the voltages (PWM duty cycles) to the “DACs”.
- (FUNCT) m5i20.<board>.digital-out-write – Writes digital outputs.
- (FUNCT) m5i20.<board>.misc-update – Writes watchdog timer configuration to hardware. Resets watchdog timer. Updates E-stop pin (more info needed). Updates onboard LEDs.

15.5.4 Connector pinout

The Hostmot-4 FPGA configuration has the following pinout. There are three 50-pin ribbon cable connectors on the card: P2, P3, and P4. There are also 8 status LEDs.

[8] With normal 10 bit PWM, 50% duty cycle would be 512 cycles on and 512 cycles off = ca 30 kHz with 33 MHz reference counter. With fully interleaved PWM this would be 1 cycle on, 1 cycle off for 1024 cycles (16.66 MHz if the PWM reference counter runs at 33 MHz) = much easier to filter. The 5I20 configuration interlace is somewhat between non and fully interlaced (to make it easy to filter but not have as many transistions as fully interleaved).

15.5.4.1 Connector P2

m5i20 card connector P2	Function/HAL-pin
1	enc-01 A input
3	enc-01 B input
5	enc-00 A input
7	enc-00 B input
9	enc-01 index input
11	enc-00 index input
13	dac-01 output
15	dac-00 output
17	DIR output for dac-01
19	DIR output for dac-00
21	dac-01-enable output
23	dac-00-enable output
25	enc-03 B input
27	enc-03 A input
29	enc-02 B input
31	enc-02 A input
33	enc-03 index input
35	enc-02 index input
37	dac-03 output
39	dac-02 output
41	DIR output for dac-03
43	DIR output for dac-02
45	dac-03-enable output
47	dac-02-enable output
49	Power +5 V (or +3.3V ?)
all even pins	Ground

15.5.4.2 Connector P3

Encoder counters 4 - 7 work simultaneously with in-00 to in-11.

If you are using in-00 to in-11 as general purpose IO then reading enc-<4-7> will produce some random junk number.

m5i20 card connector P3	Function/HAL-pin	Secondary Function/HAL-pin
1	in-00	enc-04 A input
3	in-01	enc-04 B input
5	in-02	enc-04 index input
7	in-03	enc-05 A input
9	in-04	enc-05 B input
11	in-05	enc-05 index input
13	in-06	enc-06 A input
15	in-07	enc-06 B input
17	in-08	enc-06 index input
19	in-09	enc-07 A input
21	in-10	enc-07 B input
23	in-11	enc-07 index input
25	in-12	
27	in-13	
29	in-14	
31	in-15	
33	out-00	
35	out-01	
37	out-02	
39	out-03	
41	out-04	
43	out-05	
45	out-06	
47	out-07	
49	Power +5 V (or +3.3V ?)	
all even pins	Ground	

15.5.4.3 Connector P4

The index mask masks the index input of the encoder so that the encoder index can be combined with a mechanical switch or opto detector to clear or latch the encoder counter only when the mask input bit is in proper state (selected by mask polarity bit) and encoder index occurs. This is useful for homing. The behaviour of these pins is controlled by the Counter Control Register (CCR), however there is currently no function in the driver to change the CCR. See REGMAP4[9] for a description of the CCR.

[9] emc2/src/hal/drivers/m5i20/REGMAP4E

m5i20 card connector P4	Function/HAL-pin	Secondary Function/HAL-pin
1	in-16	enc-00 index mask
3	in-17	enc-01 index mask
5	in-18	enc-02 index mask
7	in-19	enc-03 index mask
9	in-20	
11	in-21	
13	in-22	
15	in-23	
17	in-24	enc-04 index mask
19	in-25	enc-05 index mask
21	in-26	enc-06 index mask
23	in-27	enc-07 index mask
25	in-28	
27	in-29	
29	in-30	
31	in-31	
33	out-08	
35	out-09	
37	out-10	
39	out-11	
41	out-12	
43	out-13	
45	out-14	
47	out-15	
49	Power +5 V (or +3.3V ?)	
all even pins	Ground	

15.5.4.4 LEDs

The status LEDs will monitor one motion channel set by the `m5i20.<board>.led-view` parameter. A call to `m5i20.<board>.misc-update` is required to update the viewed channel.

LED name	Output
LED0	IRQLatch ?
LED1	enc-<channel> A
LED2	enc-<channel> B
LED3	enc-<channel> index
LED4	dac-<channel> DIR
LED5	dac-<channel>
LED6	dac-<channel>-enable
LED7	watchdog timeout ?

15.6 Vital Systems Motenc-100 and Motenc-LITE

The Vital Systems Motenc-100 and Motenc-LITE are 8- and 4-channel servo control boards. The Motenc-100 provides 8 quadrature encoder counters, 8 analog inputs, 8 analog outputs, 64 (68?) digital inputs, and 32 digital outputs. The Motenc-LITE has only 4 encoder counters, 32 digital inputs and 16 digital outputs, but it still has 8 analog inputs and 8 analog outputs. The driver automatically identifies the installed board and exports the appropriate HAL objects.[10]

Installing:

[10] Ideally the encoders, DACs, ADCs, and digital I/O would comply with the canonical interfaces defined earlier, but they don't. Fixing that is on the things-to-do list.

```
loadrt hal_motenc
```

During loading (or attempted loading) the driver prints some usefull debugging message to the kernel log, which can be viewed with **dmesg**.

Up to 4 boards may be used in one system.

15.6.1 Pins

In the following pins, parameters, and functions, <board> is the board ID. According to the naming conventions the first board should always have an ID of zero. However this driver sets the ID based on a pair of jumpers on the baord, so it may be non-zero even if there is only one board.

- (S32) motenc.<board>.enc-<channel>-count – Encoder position, in counts.
- (FLOAT) motenc.<board>.enc-<channel>-position – Encoder position, in user units.
- (BIT) motenc.<board>.enc-<channel>-index – Current status of index pulse input.
- (BIT) motenc.<board>.enc-<channel>-idx-latch – Driver sets this pin true when it latches an index pulse (enabled by latch-index). Cleared by clearing latch-index.
- (BIT) motenc.<board>.enc-<channel>-latch-index – If this pin is true, the driver will reset the counter on the next index pulse.
- (BIT) motenc.<board>.enc-<channel>-reset-count – If this pin is true, the counter will immediately be reset to zero, and the pin will be cleared.
- (FLOAT) motenc.<board>.dac-<channel>-value – Analog output value for DAC (in user units, see -gain and -offset)
- (FLOAT) motenc.<board>.adc-<channel>-value – Analog input value read by ADC (in user units, see -gain and -offset)
- (BIT) motenc.<board>.in-<channel> – State of digital input pin, see canonical digital input.
- (BIT) motenc.<board>.in-<channel>-not – Inverted state of digital input pin, see canonical digital input.
- (BIT) motenc.<board>.out-<channel> – Value to be written to digital output, seen canonical digital output.
- (BIT) motenc.<board>.estop-in – Dedicated estop input, more details needed.
- (BIT) motenc.<board>.estop-in-not – Inverted state of dedicated estop input.
- (BIT) motenc.<board>.watchdog-reset – Bidirectional, - Set TRUE to reset watchdog once, is automatically cleared.

15.6.2 Parameters

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

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

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

  Typically, the useful values are 0 (watchdog disabled) or 20 (8ms watchdog enabled, cleared by dac-write).
- (U32) motenc.<board>.led-view – Maps some of the I/O to onboard LEDs?

15.6.3 Functions

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

15.7 Pico Systems PPMC (Parallel Port Motion Control)

Pico Systems has a family of boards for doing servo, stepper, and pwm control. The boards connect to the PC through a parallel port working in EPP mode. Although most users connect one board to a parallel port, in theory any mix of up to 8 or 16 boards can be used on a single parport. One driver serves all types of boards. The final mix of I/O depends on the connected board(s). The driver doesn't distinguish between boards, it simply numbers I/O channels (encoders, etc) starting from 0 on the first card.

Installing:

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

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

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

15.7.1 Pins

In the following pins, parameters, and functions, <board> is the board ID. According to the naming conventions the first board should always have an ID of zero. However this driver sets the ID based on a pair of jumpers on the baord, so it may be non-zero even if there is only one board.

- (s32) ppmc.<port>.encoder.<channel>.count – Encoder position, in counts.
- (s32) ppmc.<port>.encoder.<channel>.delta – Change in counts since last read.
- (FLOAT) ppmc.<port>.encoder.<channel>.position – Encoder position, in user units.
- (BIT) ppmc.<port>.encoder.<channel>.index – Something to do with index pulse.[11]
- (BIT) ppmc.<port>.pwm.<channel>.enable – Enables a PWM generator.
- (FLOAT) ppmc.<port>.pwm.<channel>.value – Value which determines the duty cycle of the PWM waveforms. The value is divided by pwm.<channel>.scale, and if the result is 0.6 the duty cycle will be 60%, and so on. Negative values result in the duty cycle being based on the absolute value, and the direction pin is set to indicate negative.
- (BIT) ppmc.<port>.stepgen.<channel>.enable – Enables a step pulse generator.
- (FLOAT) ppmc.<port>.stepgen.<channel>.velocity – Value which determines the step frequency. The value is multiplied by stepgen.<channel>.scale, and the result is the frequency in steps per second. Negative values result in the frequency being based on the absolute value, and the direction pin is set to indicate negative.
- (BIT) ppmc.<port>.in-<channel> – State of digital input pin, see canonical digital input.
- (BIT) ppmc.<port>.in.<channel>.not – Inverted state of digital input pin, see canonical digital input.
- (BIT) ppmc.<port>.out-<channel> – Value to be written to digital output, seen canonical digital output.

15.7.2 Parameters

- (FLOAT) ppmc.<port>.enc.<channel>.scale – The number of counts / user unit (to convert from counts to units).
- (FLOAT) ppmc.<port>.pwm.<channel-range>.freq – The PWM carrier frequency, in Hz. Applies to a group of four consecutive PWM generators, as indicated by <channel-range>. Minimum is 153Hz, maximum is 500KHz.
- (FLOAT) ppmc.<port>.pwm.<channel>.scale – Scaling for PWM generator. If scale is X, then the duty cycle will be 100% when the value pin is X (or -X).
- (FLOAT) ppmc.<port>.pwm.<channel>.max-dc – Maximum duty cycle, from 0.0 to 1.0.
- (FLOAT) ppmc.<port>.pwm.<channel>.min-dc – Minimum duty cycle, from 0.0 to 1.0.
- (FLOAT) ppmc.<port>.pwm.<channel>.duty-cycle – Actual duty cycle (used mostly for troubleshooting.)
- (BIT) ppmc.<port>.pwm.<channel>.bootstrap – If true, the PWM generator will generate a short sequence of pulses of both polarities when it is enabled, to charge the bootstrap capacators used on some MOSFET gate drivers.

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

- (U32) ppmc.<port>.stepgen.<channel-range>.setup-time – Sets minimum time between direction change and step pulse, in units of 100nS. Applies to a group fof four consecutive PWM generators, as indicated by <channel-range>.
- (U32) ppmc.<port>.stepgen.<channel-range>.pulse-width – Sets width of step pulses, in units of 100nS. Applies to a group fof four consecutive PWM generators, as indicated by <channel-range>.
- (U32) ppmc.<port>.stepgen.<channel-range>.pulse-space-min – Sets minimum time between pulses, in units of 100nS. The maximum step rate is 1/(100nS * (pulse-width + pulse-space-min)). Applies to a group fof four consecutive PWM generators, as indicated by <channel-range>.
- (FLOAT) ppmc.<port>.stepgen.<channel>.scale – Scaling for step pulse generator. The step frequency in Hz is the absolute value of velocity * scale.
- (FLOAT) ppmc.<port>.stepgen.<channel>.max-vel – The maximum value for velocity. Commands greater than max-vel will be clamped. Also applies to negative values. (The absolute value is clamped.)
- (FLOAT) ppmc.<port>.stepgen.<channel>.frequency – Actual step pulse frequency in Hz (used mostly for troubleshooting.)
- (BIT) ppmc.<port>.out.<channel>.invert – Inverts a digital output, see canonical digital output.

15.7.3 Functions

- (FUNCT) ppmc.<port>.read – Reads all inputs (digital inputs and encoder counters) on one port.
- (FUNCT) ppmc.<port>.write – Writes all outputs (digital outputs, stepgens, PWMs) on one port.

15.8 Pluto-P: generalities

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

15.8.1 Requirements

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

15.8.2 Connectors

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

15.8.3 Physical Pins

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

15.8.4 LED

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

15.8.5 Power

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

15.8.6 PC interface

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

15.8.7 Rebuilding the FPGA firmware

The `src/hal/drivers/pluto_servo_firmware/` and `src/hal/drivers/pluto_step_firmware/` subdirectories contain the Verilog source code plus additional files used by Quartus for the FPGA firmwares. Altera's Quartus II software is required to rebuild the FPGA firmware. To rebuild the

firmware from the .hdl and other source files, open the `.qpf` file and press CTRL-L. Then, recompile emc2.

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

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

15.8.8 For more information

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

15.9 pluto-servo: Hardware PWM and quadrature counting

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

This driver features:

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

15.9.1 Pinout

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

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

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

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

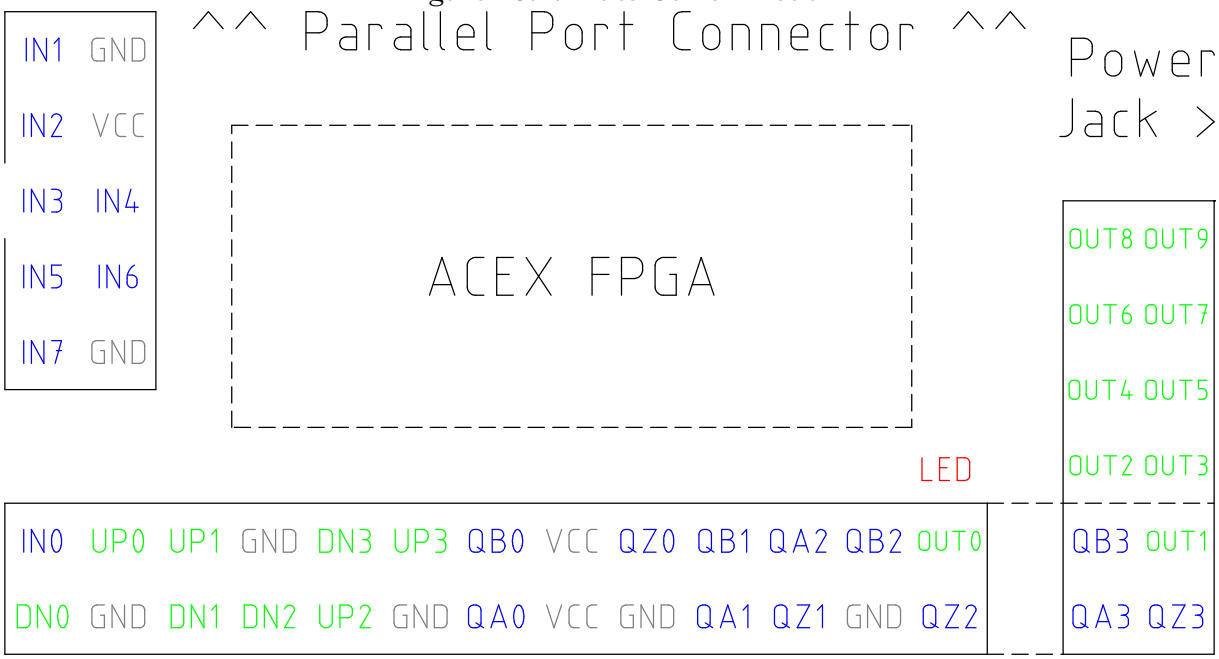
INx Dedicated digital input #x

OUTx Dedicated digital output #x

GND Ground

VCC +3.3V regulated DC

Figure 15.2: Pluto-Servo Pinout



15.9.2 Input latching and output updating

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

15.9.3 HAL Functions, Pins and Parameters

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

15.9.4 Compatible driver hardware

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

Table 15.1: Pluto-Servo Alternate Pin Functions

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

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

15.10 Pluto-step: 300kHz Hardware Step Generator

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

The board features:

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

15.10.1 Pinout

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

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

INx Dedicated digital input #x

OUTx Dedicated digital output #x

GND Ground

VCC +3.3V regulated DC

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

15.10.2 Input latching and output updating

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

15.10.3 Step Waveform Timings

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

Figure 15.3: Pluto-Step Pinout

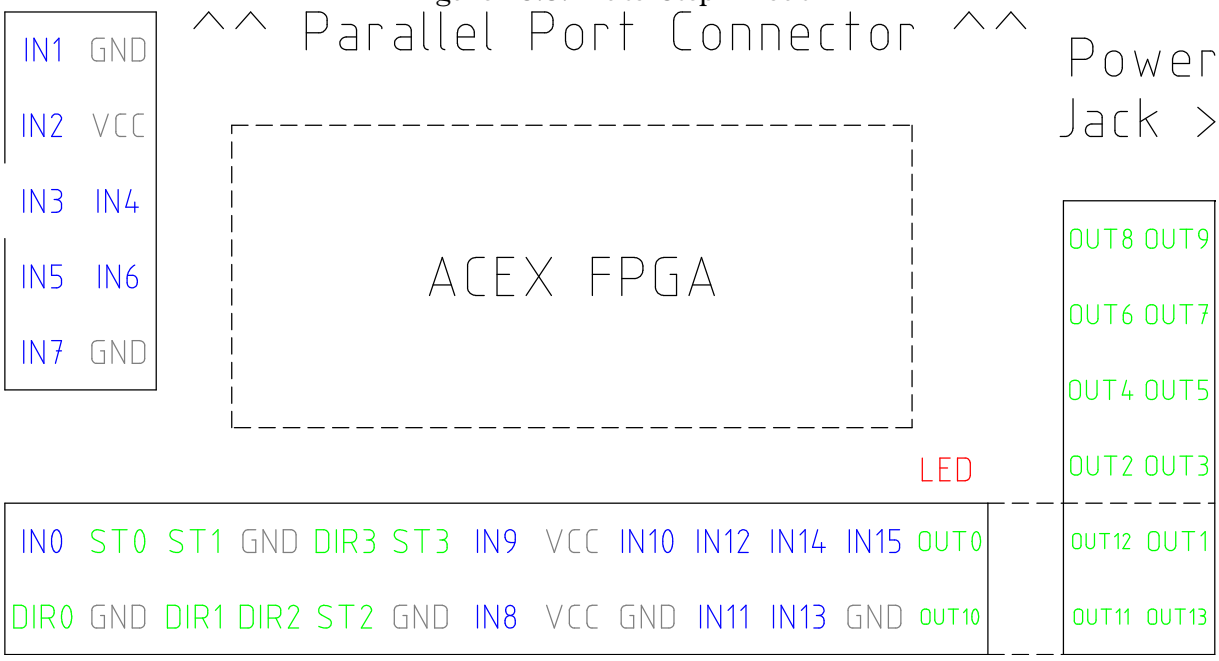
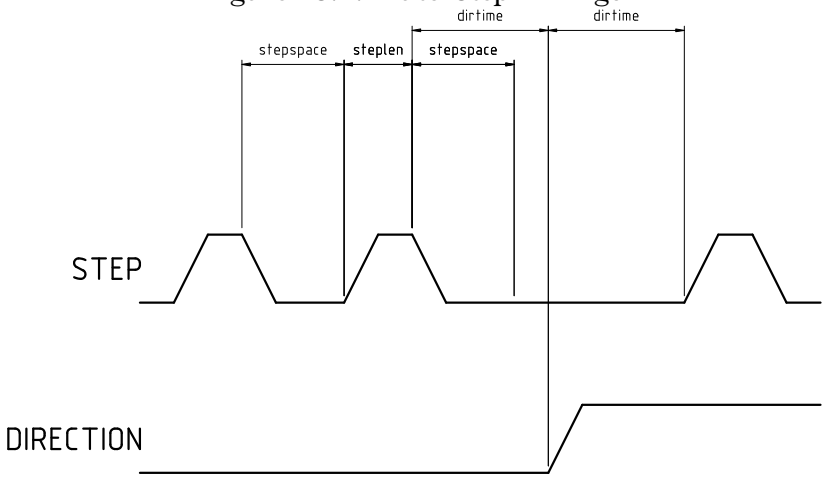


Figure 15.4: Pluto-Step Timings



15.10.4 HAL Functions, Pins and Parameters

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

Chapter 16

Halui

16.1 Introduction

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

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

HALUI = halui

in the [HAL] section.

16.2 Halui pin reference

16.2.1 Machine

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

16.2.2 E-Stop

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

16.2.3 Mode

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

16.2.4 Mist, Flood, Lube

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

16.2.5 Spindle

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

16.2.6 Joints

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

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

16.2.7 Jogging

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

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

16.2.8 Selecting a joint

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

16.2.9 Feed override

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

16.2.10 Spindle override

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

16.2.11 Tool

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

16.2.12 Program

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

16.2.13 General

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

16.2.14 MDI

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

- a MDI_COMMAND is added to the ini (in the section [HALUI]) (e.g. [HALUI] MDI_COMMAND = G0 X0
- when halui starts it will read/detect the MDI_COMMAND fields in the ini, and export pins of type (BIT) halui.mdi-command-<nr> (<nr> is a number from 00 to the number of MDI_COMMAND's found in the ini)
- when the pin halui.mdi-command-<nr> is activated halui will try to send the MDI command defined in the ini. This will not always succeed, depending on the operating mode emc2 is in (e.g. while in AUTO halui can't successfully send MDI commands).

16.3 Case - Studies

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

Chapter 17

Panneau de contrôle virtuel - Virtual Control Panels

17.1 Introduction

Les panneaux de contrôle des machines traditionnelles sont de grandes plaques d'acier avec des boutons poussoirs, des potentiomètres, des voyants et parfois quelques galvanomètres montés parmis tout cela. Ils présentent beaucoup d'avantages, les boutons sont beaucoup plus robustes qu'un clavier d'ordinateur, ils sont aussi suffisamment gros pour être manipulés tout en regardant autre chose, par exemple l'outil. Cependant, ils ont aussi des inconvénients. Ils occupent beaucoup de place sur le panneau, qui doit être de grande taille, ils sont chers et leur câblage vers le PC peut utiliser beaucoup de broches d'entrée/sortie. C'est là que le panneau virtuel entre en scène.

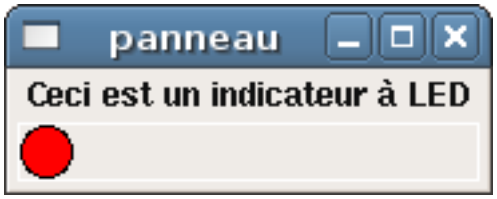
Un panneau virtuel de contrôle (VCP) est une fenêtre, sur l'écran de l'ordinateur, avec des boutons, des galvanomètres, des potentiomètres, des interrupteurs, etc. Quand vous cliquez sur un bouton du VCP, il change d'état une pin de HAL, exactement comme si vous aviez pressé sur un bouton physique raccordé à une broche d'entrée d'un périphérique d'entrée. De même, une led VCP s'allume lorsque la pin de HAL devient VRAIE, tout comme un voyant physique à lampe, raccordé à une broche de sortie d'un périphérique de sortie. Les panneaux virtuels de contrôle ne sont pas destinés à remplacer les panneaux physiques, parfois il n'y a pas de substitut à un bon gros bouton poussoir étanche aux huiles. Mais les panneaux virtuels peuvent être utilisés pour tester ou contrôler des fonctionnalités qui ne requiert ainsi, ni bouton ni voyant physique et qui remplacent temporairement des organes physiques réels d'entrée/sortie, par exemple, pendant la phase de déboguage du programme. Ou pour simuler un panneau de contrôle physique avant qu'il ne soit fabriqué, câblé et raccordé à une carte d'entrée/sortie.

Actuellement il y a deux implémentations de VCP dans EMC2: L'ancien, simplement nommé VCP, qui utilise les widgets de GTK et le nouveau appelé pyVCP, qui utilise les widgets Tkinter. VCP n'est plus recommandé, il ne doit plus être utilisé, il sera supprimé dans l'avenir.

17.2 pyVCP

La disposition d'un panneau pyVCP est spécifiée avec un fichier XML qui contient les balises des widgets entre <pyvcp> et </pyvcp>. Par exemple:

```
<pyvcp>
    <label text="Ceci est un indicateur à LED"/>
    <led/>
</pyvcp>
```



Si vous placez ce texte dans un fichier nommé tiny.xml et que vous le lancez avec:

```
pyvcp -c panneau tiny.xml
```

pyVCP va créer le panneau pour vous, il y inclut deux widgets, un Label avec le texte “Ceci est un indicateur à LED” et une LED rouge, utilisée pour afficher l’état d’un signal HAL de type BIT. Il va aussi créer un composant HAL nommé “panneau” (tous les widgets dans ce panneau sont connectés aux pins qui démarrent avec “panneau”). Comme aucune balise <halpin> n’était présente à l’intérieur de la balise <led>, pyVCP nomme automatiquement la pin HAL pour le widget LED panneau.led.0

Pour obtenir la liste des widgets, leurs balises et options, consultez la documentation des widgets:17.5

Un fois que vous avez créé votre panneau, connecter les signaux HAL de la forme à la pin pyVCP se fait avec la commande ’halcmd linksp’ habituelle. Si vous débutez avec HAL, le tutoriel de HAL 7 est vivement recommendé.

17.3 Sécurité avec pyVCP

Certaines parties de pyVCP sont évaluées comme du code Python, elles peuvent donc produirent n’importe quelle action disponible dans les programmes Python. N’utilisez que des fichiers pyVCP en .xml à partir d’une source de confiance.

17.4 Utiliser pyVCP avec AXIS

Puisque AXIS utilise le même environnement graphique et les même outils (Tkinter) que pyVCP, il est possible d’inclure un panneau pyVCP sur le côté droit de l’interface utilisateur normale d’AXIS. Un exemple typique est présenté ci-dessous.

Placer le fichier pyVCP XML décrivant le panneau dans le même répertoire que le fichier .ini. Nous voulons afficher la vitesse courante de la broche sur un widget barre de progression. Copier le code XML suivant dans un fichier appelé broche.xml:

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

Ici nous avons fait un panneau avec un label “Vitesse broche:” et un widget barre de progression. Nous avons spécifié que la pin HAL connectée à la barre de progression devait s’appeler “spindle-speed” et réglé la valeur maximum de la barre à 5000 (se reporter à la documentation des widgets, plus loin, pour toutes les options disponibles). Pour faire connaître ce fichier à AXIS et qu’il l’appelle au démarrage, nous devons préciser ce qui suit dans la section [DISPLAY] du fichier .ini:

```
PYVCP = broche.xml
```

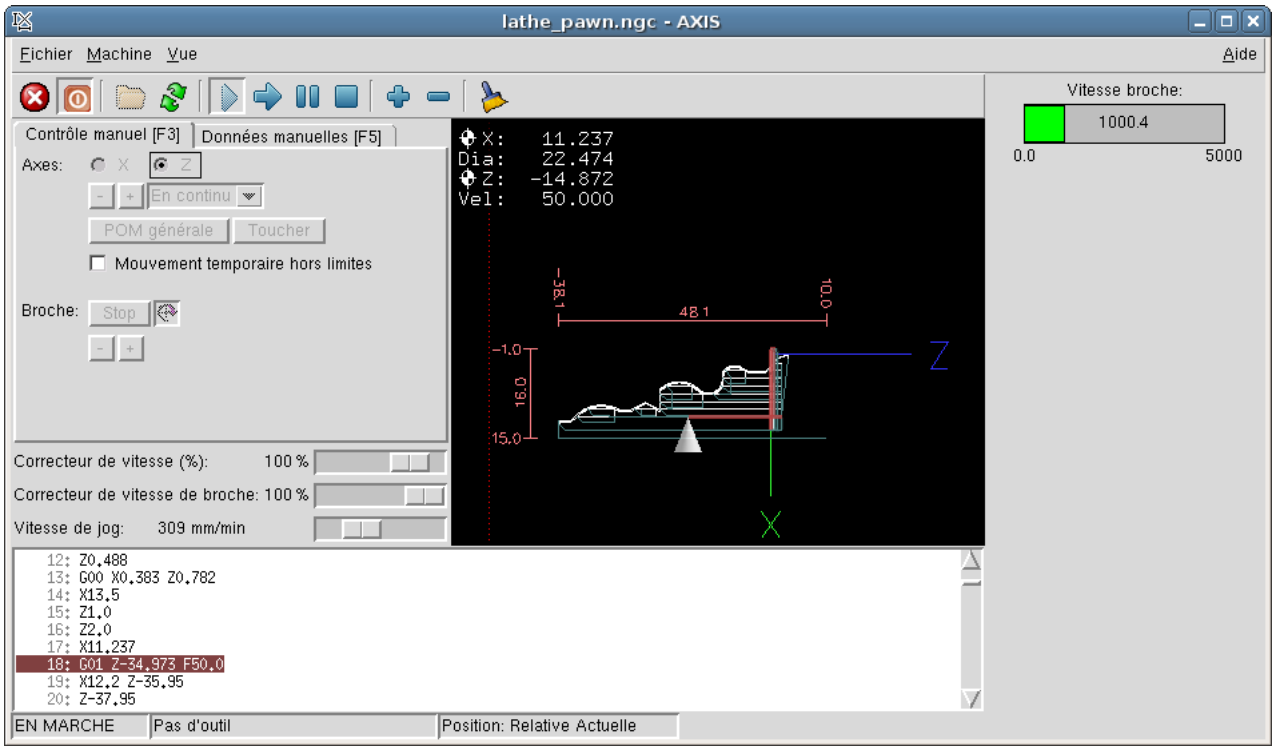
Pour que notre widget affiche réellement la vitesse de la broche “spindle-speed”, il doit être raccordé au signal approprié de HAL. Le fichier .hal qui sera exécuté quand AXIS et pyVCP démarreront doit être spécifié, de la manière suivante, dans la section [HAL] du fichier .ini:

```
POSTGUI_HALFILE = broche_vers_pyvcp.hal
```

Ce changement lancera la commande HAL spécifiée dans “broche_vers_pyvcp.hal”. Dans notre exemple, ce fichier contiendra juste la commande suivante:

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

ce qui suppose que le signal appelé “spindle-rpm-filtered” existe aussi. Noter que lors de l’exécution avec AXIS, toutes les pins des widgets de pyVCP ont des noms commençant par “pyvcp.”.



Voila à quoi ressemble le panneau pyVCP que nous venons de créer, incorporé à AXIS. La configuration `sim/lathe` fournie en exemple, est configurée de cette manière.

17.5 Documentation des widgets de pyVCP

Les signaux de HAL existent en deux variantes, BIT et FLOAT. pyVCP peut afficher la valeur d’un signal avec un widget indicateur, ou modifier la valeur d’un signal avec un widget de contrôle. Ainsi, il y a quatre classes de widgets pyVCP connectables aux signaux de HAL. Une cinquième classe de widgets d’aide permet d’organiser et d’appliquer des labels aux panneaux.

1. Widgets de signalisation, signaux BIT: LED
2. Widgets de contrôle, signaux BIT: Button, Checkbutton, Radiobutton

3. Widgets de signalisation, signaux FLOAT: Number, Bar, Meter
4. Widgets de contrôle, signaux FLOAT: Spinbox, Scale, Jogwheel
5. Widgets d'aide: Hbox, Vbox, Table, Label, Labelframe

17.5.0.1 Syntaxe

Chaque widget est décrit brièvement, suivi par la forme d'écriture utilisée et d'une capture d'écran. Toutes les balises contenues dans la balise du widget principal, sont optionnelles.

17.5.0.2 Notes générales

Ã l'heure actuelle, les deux syntaxes, basée sur les balises et basée sur les attributs, sont supportées. Par exemple, les deux fragments de code XML suivants sont traités de manière identique:

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

et

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

Quand la syntaxe basée sur les attributs est utilisée, les règles suivantes sont utilisées pour convertir les valeurs des attributs en valeurs Python:

1. Si le premier caractère de l'attribut est un des suivants: `{([" '` , Il est évalué comme une expression Python.
2. Si la chaine est acceptée par int(), la valeur est traitée comme un entier.
3. Si la chaine est acceptée par float(), la valeur est traitée comme un flottant.
4. Autrement, la chaine est acceptée comme une chaine.

Quand la syntaxe basée sur les balises est utilisée, le texte entre les balises est toujours évalué comme un expression Python.

Les exemples ci-dessous montrent un mélange des deux formats.

17.5.1 LED

Une LED est utilisée pour indiquer l'état d'un signal BIT. La couleur de la LED sera on_color quand le signal BIT est vrai et off_color autrement.

```
<led>
    <halpin>"ma-led"</halpin>
    <size>50</size>
    <on_color>"bleue"</on_color>
    <off_color>"noire"</off_color>
</led>
```

<halpin> définit le nom de la pin, par défaut: "led.n", où n est un entier
<size> définit la taille de la led, par défaut: 20
<on_color> définit la couleur de la led LED quand la pin est vraie, par défaut: "green"
<off_color> définit la couleur de la LED quand la pin est fausse, par défaut: "ref"

17.5.2 Bouton (button)

Un bouton permet de contrôler une pin BIT. La pin sera mise vraie quand le bouton sera pressé et maintenu enfoncé, elle sera mise fausse quand le bouton sera relâché.

```
<button>
    <halpin>"mon-button"</halpin>
    <text>"OK"</text>
</button>
```

17.5.3 Case à cocher (checkbutton)

Une case à cocher contrôle une pin BIT. La pin sera mise vraie quand la case sera cochée et fausse si la case est décochée.

```
<checkbutton>
    <halpin>"ma-case-à-cocher"</halpin>
</checkbutton>
```

Une case non cochée:  et une case cochée: 

17.5.4 Bouton radio (radiobutton)

Un bouton radio placera une seule des pins BIT vraie. Les autres seront mises fausses.

```
<radiobutton>
    <choices>["un","deux","trois"]</choices>
    <halpin>"mon-bouton-radio"</halpin>
</radiobutton>
```

Noter que dans cet exemple, les pins de HAL seront nommées mon-bouton-radio.un, mon-bouton-radio.deux et mon-bouton-radio.trois. Dans la capture d'écran, la valeur "trois" est sélectionnée.

17.5.5 Nombre (number)

Le widget nombre affiche la valeur d'un signal FLOAT.

```
<number>
    <halpin>"mon-nombre"</halpin>
    <font>('Helvetica',50)</font>
    <format>"+4.3f"</format>
</number>
```

<font> est une police de caractères de type Tkinter avec la spécification de sa taille. Noter que sous Ubuntu 6.06 'Helvetica' n'est pas disponible en taille supérieure à 40 ou 50. Une police qui peut être agrandie jusqu'à la taille 200 est la police 'courier 10 pitch', que vous pouvez spécifier de la manière suivante, pour afficher des chiffres réellement grands:

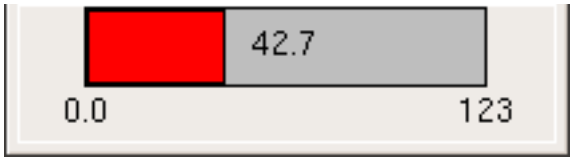
```
<font>('courier 10 pitch',100)</font>
```

<format> est un format 'style C', spécifié pour définir le format d'affichage du nombre.

17.5.6 Barre de progression (bar)

Le widget barre de progression affiche la valeur d'un signal FLOAT, graphiquement dans une barre de progression et simultanément, en numérique.

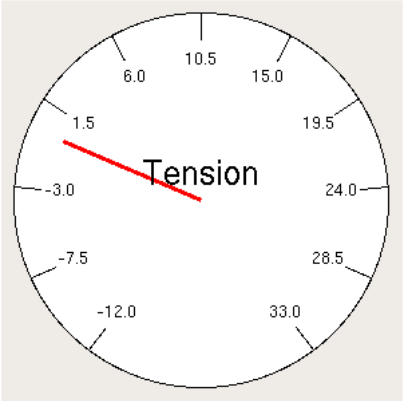
```
<bar>
    <halpin>"ma-bar"</halpin>
    <min_>0</min_>
    <max_>123</max_>
    <bgcolor>"grey"</bgcolor>
    <fillcolor>"red"</fillcolor>
</bar>
```



17.5.7 Galvanomètre (meter)

Le galvanomètre affiche la valeur d'un signal FLOAT dans un affichage à aiguille "à l'ancienne".

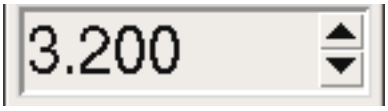
```
<meter>
    <halpin>"mon-galva"</halpin>
    <text>"Tension"</text>
    <size>300</size>
    <min_>-12</min_>
    <max_>33</max_>
</meter>
```



17.5.8 Roue codeuse (spinbox)

La roue codeuse contrôle une pin FLOAT. La valeur de la pin est augmentée ou diminuée de la valeur de 'resolution', à chaque pression sur une flèche, ou en positionnant la souris sur le nombre puis en tournant la molette de la souris.

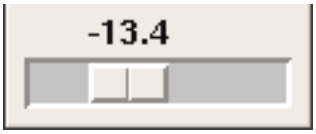
```
<spinbox>
    <halpin>"ma-roue-codeuse"</halpin>
    <min_>-12</min_>
    <max_>33</max_>
    <resolution>0.1</resolution>
    <format>"2.3f"</format>
    <font>('Arial',30)</font>
</spinbox>
```



17.5.9 Curseur (scale)

Le curseur contrôle une pin FLOAT. La valeur de la pin est augmentée ou diminuée en déplaçant le curseur, ou en positionnant la souris sur le curseur puis en tournant la molette de la souris.

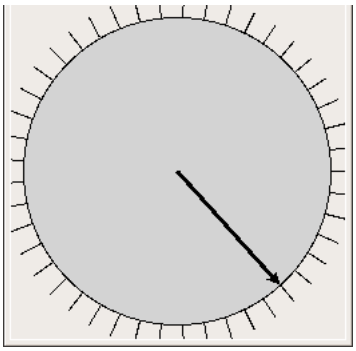
```
<scale>
    <halpin>"mon-curseur"</halpin>
    <resolution>0.1</resolution>
    <orient>HORIZONTAL</orient>
    <min_>-33</min_>
    <max_>26</max_>
</scale>
```



17.5.10 Bouton tournant (jogwheel)

Le bouton tournant imite le fonctionnement d'un vrai bouton tournant, en sortant sur une pin FLOAT la valeur sur laquelle est positionné le bouton, que ce soit en le faisant tourner avec un mouvement circulaire, ou en tournant la molette de la souris.

```
<jogwheel>
    <halpin>"mon-bouton-tournant"</halpin>
    <cpr>45</cpr>
    <size>250</size>
</jogwheel>
```



17.6 Documentation des containers de pyVCP

Les containers sont des widgets qui contiennent d'autres widgets.

17.6.1 Hbox

Utilisez une Hbox lorsque vous voulez aligner les widgets, horizontalement, les uns à côtés des autres.

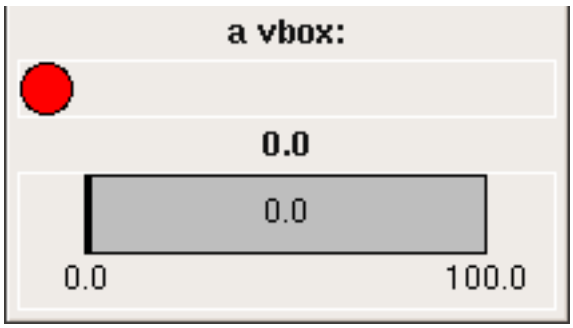
```
<hbox>
    <label><text>"une hbox:"</text></label>
    <led></led>
    <number></number>
    <bar></bar>
</hbox>
```

Ã l'intérieur d'une Hbox, vous pouvez utiliser les balises `<boxfill fill=""/>`, `<boxanchor anchor=""/>` et `<boxexpand expand=""/>` pour choisir le comportement des éléments contenus dans la boîte, lors d'un redimensionnement de la fenêtre. Pour des détails sur le comportement de fill, anchor, et expand, référez vous au manuel du pack Tk, `pack(3tk)`. Par défaut, `fill='y'`, `anchor='center'`, `expand='yes'`.

17.6.2 Vbox

Utilisez une Vbox lorsque vous voulez aligner les widgets verticalement, les uns au dessus des autres.

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



Ã l'intérieur d'une Hbox, vous pouvez utiliser les balises `<boxfill fill=""/>`, `<boxanchor anchor=""/>` et `<boxexpand expand=""/>` pour choisir le comportement des éléments contenus dans la boîte, lors d'un redimensionnement de la fenêtre. Pour des détails sur le comportement de fill, anchor, et expand, référez vous au manuel du pack Tk, `pack(3tk)`. Par défaut, `fill='y'`, `anchor='center'`, `expand='yes'`.

17.6.3 Label

Un label est un texte qui s'affiche sur le panneau.

```
<label>
    <text>"Ceci est un label:"</text>
    <font>('Helvetica',20)</font>
</label>
```

17.6.4 Labelframe

Un labelframe est un cadre entouré d'un sillon et un label en haut à gauche.

```
<labelframe text="Label: Leds groupées">
  <hbox>
    <led/> <led/></led>
  </hbox>
</labelframe>
```

17.6.5 Table

Une table est un container qui permet d'écrire dans une grille de lignes et de colonnes. Chaque ligne débute avec la balise `<tablerow/>`. Un widget container peut être en lignes ou en colonnes par l'utilisation de la balise `<tablespan rows= cols=/>`. Les bordures des cellules contenant les widgets "sticky" peuvent être réglées grâce à l'utilisation de la balise `<tablesticky sticky=/>`. Une table peut s'étirer sur ses lignes et colonnes flexibles (sticky).

Exemple:

```
<table flexible_rows="[2]" flexible_columns="[1,4]">
   <tablesticky sticky="new"/>
   <tablerow/>
    <label text="A (cell 1,1)"/>
    <label text="B (cell 1,2)"/>
    <tablespan columns="2"/><label text="C, D (cells 1,3 and 1,4)">
   <tablerow/>
    <label text="E (cell 2,1)"/>
    <tablesticky sticky="nsew"/><tablespan rows="2"/>
        <label text="'spans\n2 rows'"/>
    <tablesticky sticky="new"/><label text="G (cell 2,3)"/>
    <label text="H (cell 2,4)"/>
   <tablerow/>
    <label text="J (cell 3,1)"/>
    <label text="K (cell 3,2)"/>
    <label text="M (cell 3,4)"/>
</table>
```

17.7 VCP: Un petit exemple

NOTE: VCP n'est plus conseillé et ne devrait plus faire l'objet de nouveaux développements ou de widgets supplémentaires. Il est fortement recommandé d'utiliser pyVCP. Cependant, pyVCP ne sera mis à jour que pour la publication de la version 2.2 et VCP est en version 2.1. Ce qui signifie que tant que des utilisateurs utilisent encore VCP, nous ne pouvons pas simplement l'arrêter.[1]

Placer les lignes suivantes dans un fichier nommé `tiny.vcp`:

```
vcp {
 main-window {
   box {
     button {
       halpin = vcp.pushbutton
       label { text = "Pressez moi" }
     }
     LED {
       halpin = vcp.light
     }
   }
 }
}
```

Le fichier ci-dessus, décrit un minuscule panneau de contrôle virtuel, avec un bouton poussoir et une lampe. Pour voir à quoi il ressemble, il faut démarrer HAL:

[1] Un traducteur .vcp vers .xml qui prend un fichier VCP et le converti en pyVCP utilisable est sur ma liste "à faire". Il permettrait aux utilisateurs de VCP de migrer facilement vers pyVCP. Si un tel traducteur est écrit, VCP pourra être retiré à partir des versions 2.2.

```
$ halrun
```

Ensuite il faut charger halvcp et lui passer le nom de notre fichier .vcp:

```
halcmd: loadusr halvcp tiny.vcp
halcmd:
```

Il peut y avoir quelques messages affichés pendant que halvcp ouvre le fichier tiny.vcp, mais pour finir, il devrait y avoir une petite fenêtre sur votre écran, avec un bouton et une LED. Il devrait ressembler à la figure 17.1.

Figure 17.1: L'écran de tiny.vcp

Donc, nous avons un bouton et un voyant à LED, mais ils ne sont connectés à rien, de sorte que rien ne se passe quand vous appuyez sur le bouton. Cependant, la LED et le bouton ont tous les deux des pins HAL associées avec eux, pour les voir:

```
halcmd: show pin
Component Pins:
Owner  Type  Dir     Value      Name
 03    bit   IN      FALSE      vcp.light
 03    bit   OUT     FALSE      vcp.pushbutton
halcmd:
```

Pour que quelque chose se produise, il faut connecter un signal HAL entre le bouton et la lampe:

```
halcmd: newsig jumper bit
halcmd: linksp jumper vcp.pushbutton
halcmd: linksp jumper vcp.light
halcmd: show sig
Signals:
Type      Value      Name
bit       FALSE      jumper
                         ==> vcp.light
                         <== vcp.pushbutton
halcmd:
```

Maintenant pressez le bouton et la LED doit s'allumer!

17.8 VCP: Un autre petit exemple avec EMC

Placer les lignes suivantes dans un fichier nommé `estop.vcp`:

```
vcp {
   main-window {
      toggle { halpin = vcp.estop }
   }
}
```

Dans votre fichier .hal, enlevez tout ce qui est lié avec `iocontrol.0.emc-enable-in` et ajoutez-y les lignes suivantes:

```
loadusr -W halvcp estop.vcp
newsig estop bit
linkps vcp.estop => estop
linkps estop => iocontrol.0.emc-enable-in
```

Maintenant, quand vous démarrez votre machine, le bouton d'arrêt d'urgence de l'interface graphique est désactivé, il est remplacé par le bouton d'arrêt d'urgence du panneau VCP.

17.9 Syntaxe VCP

17.9.1 Block

Le format de block est le suivant:

```
balise { contenu }
```

Le contenu peut se composer d'attributs qui décrivent le block, ou d'autres blocks imbriqués dedans.

Le format des attributs est

```
nom = valeur
```

Les noms des attributs acceptables pour chaque block, dépendent dela balise block et seront listés ultérieurement.

Part VI

Advanced topics

Chapter 18

Kinematics in EMC2

18.1 Introduction

When we talk about CNC machines, we usually think about machines that are commanded to move to certain locations and perform various tasks. In order to have an unified view of the machine space, and to make it fit the human point of view over 3D space, most of the machines (if not all) use a common coordinate system called the Cartesian Coordinate System.

The Cartesian Coordinate system is composed of 3 axes (X, Y, Z) each perpendicular to the other 2. [1]

When we talk about a G-code program (RS274NGC) we talk about a number of commands (G0, G1, etc.) which have positions as parameters (X- Y- Z-). These positions refer exactly to Cartesian positions. Part of the EMC2 motion controller is responsible for translating those positions into positions which correspond to the machine kinematics[2].

18.1.1 Joints vs. Axes

A joint of a CNC machine is a one of the physical degrees of freedom of the machine. This might be linear (leadscrews) or rotary (rotary tables, robot arm joints). There can be any number of joints on a certain machine. For example a typical robot has 6 joints, and a typical simple milling machine has only 3.

There are certain machines where the joints are layed out to match kinematics axes (joint 0 along axis X, joint 1 along axis Y, joint 2 along axis Z), and these machines are called Cartesian machines (or machines with Trivial Kinematics). These are the most common machines used in milling, but are not very common in other domains of machine control (e.g. welding: puma-typed robots).

18.2 Trivial Kinematics

As we said there is a group of machines in which each joint is placed along one of the Cartesian axes. On these machines the mapping from Cartesian space (the G-code program) to the joint space (the actual actuators of the machine) is trivial. It is a simple 1:1 mapping:

```
pos->tran.x = joints[0];
pos->tran.y = joints[1];
```

[1] The word “axes” is also commonly (and wrongly) used when talking about CNC machines, and referring to the moving directions of the machine.

[2] Kinematics: a two way function to transform from Cartesian space to joint space

```
pos->tran.z = joints[2];
pos->a = joints[3];
pos->b = joints[4];
pos->c = joints[5];
```

In the above code snippet one can see how the mapping is done: the X position is identical with the joint 0, Y with joint 1 etc. The above refers to the direct kinematics (one way of the transformation) whereas the next code part refers to the inverse kinematics (or the inverse way of the transformation):

```
joints[0] = pos->tran.x;
joints[1] = pos->tran.y;
joints[2] = pos->tran.z;
joints[3] = pos->a;
joints[4] = pos->b;
joints[5] = pos->c;
```

As one can see, it's pretty straightforward to do the transformation for a trivial kins (or Cartesian) machine. It gets a bit more complicated if the machine is missing one of the axes.[34]

18.3 Non-trivial kinematics

There can be quite a few types of machine setups (robots: puma, scara; hexapods etc.). Each of them is set up using linear and rotary joints. These joints don't usually match with the Cartesian coordinates, therefor there needs to be a kinematics function which does the conversion (actually 2 functions: forward and inverse kinematics function).

To illustrate the above, we will analyze a simple kinematics called bipod (a simplified version of the tripod, which is a simplified version of the hexapod).

The Bipod we are talking about is a device that consists of 2 motors placed on a wall, from which a device is hanged using some wire. The joints in this case are the distances from the motors to the device (named AD and BD in figure 18.1).

The position of the motors is fixed by convention. Motor A is in (0,0), which means that its X coordinate is 0, and its Y coordinate is also 0. Motor B is placed in (Bx, 0), which means that its X coordinate is Bx.

Our tooltip will be in point D which gets defined by the distances AD and BD, and by the Cartesian coordinates Dx, Dy.

The job of the kinematics is to transform from joint lengths (AD, BD) to Cartesian coordinates (Dx, Dy) and vice-versa.

18.3.1 Forward transformation

To transform from joint space into Cartesian space we will use some trigonometry rules (the right triangles determined by the points (0,0), (Dx,0), (Dx,Dy) and the triangle (Dx,0), (Bx,0) and (Dx,Dy).

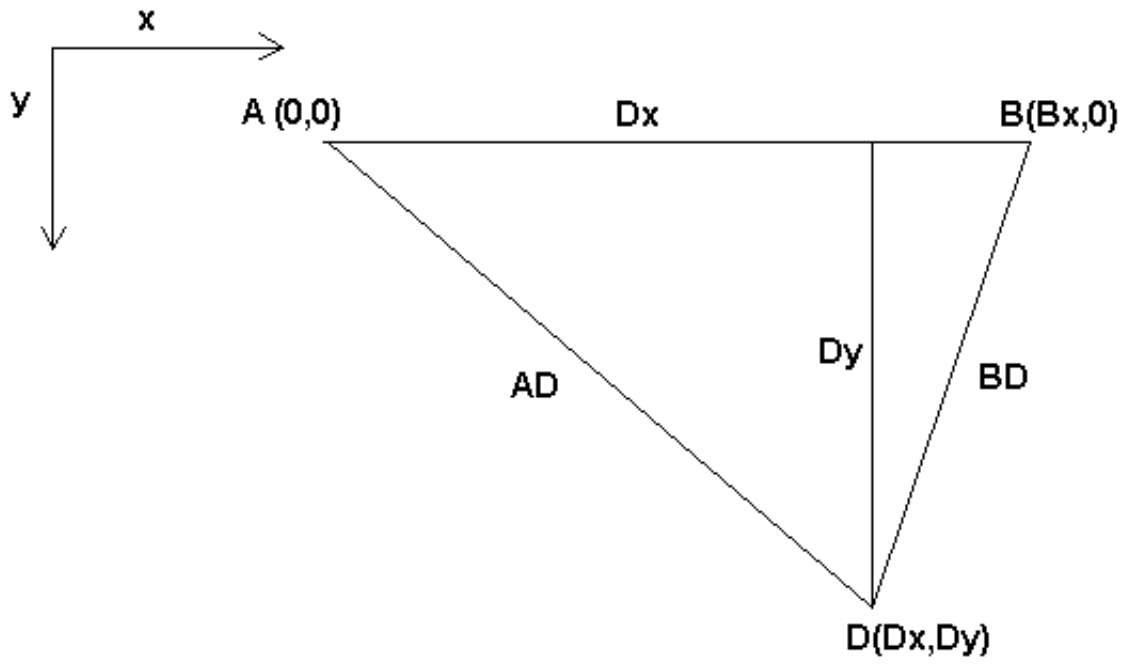
we can easily see that $AD^2 = x^2 + y^2$, likewise $BD^2 = (Bx - x)^2 + y^2$.

If we subtract one from the other we will get:

[3] If a machine (e.g. a lathe) is set up with only the axes X,Z & A, and the EMC2 inifile holds only these 3 joints defined, then the above matching will be faulty. That is because we actually have (joint0=x, joint1=Z, joint2=A) whereas the above assumes joint1=Y. To make it easily work in EMC2 one needs to define all axes (XYZA), then use a simple loopback in HAL for the unused Y axis.

[4] One other way of making it work, is by changing the matching code and recompiling the software.

Figure 18.1: Bipod setup



$$AD^2 - BD^2 = x^2 + y^2 - x^2 + 2 * x * Bx - Bx^2 - y^2$$

and therefore:

$$x = \frac{AD^2 - BD^2 + Bx^2}{2 * Bx}$$

From there we calculate:

$$y = \sqrt{AD^2 - x^2}$$

Note that the calculation for y involves the square root of a difference, which may not result in a real number. If there is no single Cartesian coordinate for this joint position, then the position is said to be a singularity. In this case, the forward kinematics return -1.

Translated to actual code:

```
double AD2 = joints[0] * joints[0];
double BD2 = joints[1] * joints[1];
double x = (AD2 - BD2 + Bx * Bx) / (2 * Bx);
double y2 = AD2 - x * x;
if(y2 < 0) return -1;
pos->tran.x = x;
pos->tran.y = sqrt(y2);
return 0;
```

18.3.2 Inverse transformation

The inverse kinematics is lots easier in our example, as we can write it directly:

$$AD = \sqrt{x^2 + y^2}$$

$$BD = \sqrt{(Bx - x)^2 + y^2}$$

or translated to actual code:

```
double x2 = pos->tran.x * pos->tran.x;
double y2 = pos->tran.y * pos->tran.y;
joints[0] = sqrt(x2 + y2);
joints[1] = sqrt((Bx - pos->tran.x)*(Bx - pos->tran.x) + y2);
return 0;
```

18.4 Implementation details

A kinematics module is implemented as a HAL component, and is permitted to export pins and parameters. It consists of several functions:

- `int kinematicsForward(const double *joint, EmcPose *world, const KINEMATICS_FORWARD_FL *fflags, KINEMATICS_INVERSE_FLAGS *iflags)`

  Implements the forward kinematics function as described in section 18.3.1.

- `extern int kinematicsInverse(const EmcPose * world, double *joints, const KINEMATICS_I *iflags, KINEMATICS_FORWARD_FLAGS *fflags)`

  Implements the inverse kinematics function as described in section 18.3.2.

- `extern KINEMATICS_TYPE kinematicsType(void)`

  Returns the kinematics type identifier.

- `int kinematicsHome(EmcPose *world, double *joint, KINEMATICS_FORWARD_FLAGS *fflags, KINEMATICS_INVERSE_FLAGS *iflags)`

  The home kinematics function sets all its arguments to their proper values at the known home position. When called, these should be set, when known, to initial values, e.g., from an INI file. If the home kinematics can accept arbitrary starting points, these initial values should be used.

- int rtapi_app_main(void)

- void rtapi_app_exit(void)

  These are the standard setup and tear-down functions of RTAPI modules.

Part VII

Tuning

18.5 Tuning servo systems

Chapter 19

PID Tuning

19.1 PID Controller

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

The Controller compares a measured value from a process (typically an industrial process) with a reference setpoint value. The difference (or "error" signal) is then used to calculate a new value for a manipulatable input to the process that brings the process' measured value back to its desired setpoint.

Unlike simpler control algorithms, the PID controller can adjust process outputs based on the history and rate of change of the error signal, which gives more accurate and stable control. (It can be shown mathematically that a PID loop will produce accurate, stable control in cases where a simple proportional control would either have a steady-state error or would cause the process to oscillate).

19.1.1 Control loop basics

Intuitively, the PID loop tries to automate what an intelligent operator with a gauge and a control knob would do. The operator would read a gauge showing the output measurement of a process, and use the knob to adjust the input of the process (the "action") until the process's output measurement stabilizes at the desired value on the gauge.

In older control literature this adjustment process is called a "reset" action. The position of the needle on the gauge is a "measurement", "process value" or "process variable". The desired value on the gauge is called a "setpoint" (also called "set value"). The difference between the gauge's needle and the setpoint is the "error".

A control loop consists of three parts:

1. Measurement by a sensor connected to the process (e.g. encoder),

2. Decision in a controller element,

3. Action through an output device such as an motor.

As the controller reads a sensor, it subtracts this measurement from the "setpoint" to determine the "error". It then uses the error to calculate a correction to the process's input variable (the "action") so that this correction will remove the error from the process's output measurement.

In a PID loop, correction is calculated from the error in three ways: cancel out the current error directly (Proportional), the amount of time the error has continued uncorrected (Integral), and anticipate the future error from the rate of change of the error over time (Derivative).

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

A PID controller can be used to control any measurable variable which can be affected by manipulating some other process variable. For example, it can be used to control temperature, pressure, flow rate, chemical composition, speed, or other variables. Automobile cruise control is an example of a process outside of industry which utilizes crude PID control.

Some control systems arrange PID controllers in cascades or networks. That is, a "master" control produces signals used by "slave" controllers. One common situation is motor controls: one often wants the motor to have a controlled speed, with the "slave" controller (often built into a variable frequency drive) directly managing the speed based on a proportional input. This "slave" input is fed by the "master" controllers' output, which is controlling based upon a related variable.

19.1.2 Theory

"PID" is named after its three correcting calculations, which all add to and adjust the controlled quantity. These additions are actually "subtractions" of error, because the proportions are usually negative:

19.1.2.0.0.1 Proportional To handle the present, the error is multiplied by a (negative) constant P (for "proportional"), and added to (subtracting error from) the controlled quantity. P is only valid in the band over which a controller's output is proportional to the error of the system. Note that when the error is zero, a proportional controller's output is zero.

19.1.2.0.0.2 Integral To learn from the past, the error is integrated (added up) over a period of time, and then multiplied by a (negative) constant I (making an average), and added to (subtracting error from) the controlled quantity. I averages the measured error to find the process output's average error from the setpoint. A simple proportional system either oscillates, moving back and forth around the setpoint because there's nothing to remove the error when it overshoots, or oscillates and/or stabilizes at a too low or too high value. By adding a negative proportion of (i.e. subtracting part of) the average error from the process input, the average difference between the process output and the setpoint is always being reduced. Therefore, eventually, a well-tuned PID loop's process output will settle down at the setpoint.

19.1.2.0.0.3 Derivative To handle the future, the first derivative (the slope of the error) over time is calculated, and multiplied by another (negative) constant D, and also added to (subtracting error from) the controlled quantity. The derivative term controls the response to a change in the system. The larger the derivative term, the more rapidly the controller responds to changes in the process's output.

More technically, a PID loop can be characterized as a filter applied to a complex frequency-domain system. This is useful in order to calculate whether it will actually reach a stable value. If the values are chosen incorrectly, the controlled process input can oscillate, and the process output may never stay at the setpoint.

19.1.3 Loop Tuning

"Tuning" a control loop is the adjustment of its control parameters (gain/proportional band, integral gain/reset, derivative gain/rate) to the optimum values for the desired control response. The optimum behavior on a process change or setpoint change varies depending on the application. Some processes must not allow an overshoot of the process variable from the setpoint. Other processes must minimize the energy expended in reaching a new setpoint. Generally stability of response is required and the process must not oscillate for any combination of process conditions and setpoints.

Tuning of loops is made more complicated by the response time of the process; it may take minutes or hours for a setpoint change to produce a stable effect. Some processes have a degree of non-linearity and so parameters that work well at full-load conditions don't work when the process is starting up from no-load. This section describes some traditional manual methods for loop tuning.

There are several methods for tuning a PID loop. The choice of method will depend largely on whether or not the loop can be taken "offline" for tuning, and the response speed of the system. If the system can be taken offline, the best tuning method often involves subjecting the system to a step change in input, measuring the output as a function of time, and using this response to determine the control parameters.

19.1.3.0.0.4 Simple method If the system must remain online, one tuning method is to first set the I and D values to zero. Increase the P until the output of the loop oscillates. Then increase I until oscillation stops. Finally, increase D until the loop is acceptably quick to reach its reference. A fast PID loop tuning usually overshoots slightly to reach the setpoint more quickly; however, some systems cannot accept overshoot.

Parameter	Rise Time	Overshoot	Settling Time	S.S. Error
P	Decrease	Increase	Small Change	Decrease
I	Decrease	Increase	Increase	Eliminate
D	Small Change	Decrease	Decrease	Small Change

Effects of increasing parameters

19.1.3.0.0.5 Ziegler-Nichols method Another tuning method is formally known as the "Ziegler-Nichols method", introduced by John G. Ziegler and Nathaniel B. Nichols. It starts in the same way as the method described before: first set the I and D gains to zero and then increase the P gain until the output of the loop starts to oscillate. Write down the critical gain (K_c) and the oscillation period of the output (P_c). Then adjust the P, I and D controls as the table shows:

Control type	P	I	D
P	$.5K_c$		
PI	$.45K_c$	$1.2/P_c$	
PID	$.6K_c$	$2/P_c$	$P \times P_c/8$

19.2 Tuning stepper systems

Chapter 20

Stepper Tuning

20.1 Getting the most out of Software Stepping

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

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

This chapter has some steps that can help you get the best results from software generated steps.

20.1.1 Run a Latency Test

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

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

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

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

> DO NOT TRY TO RUN EMC2 WHILE THE TEST IS RUNNING

On Ubuntu Dapper, you can run the test by opening a shell and doing:

```
sudo mkdir /dev/rtf;
sudo mknod /dev/rtf/3 c 150 3;
sudo mknod /dev/rtf3 c 150 3;
cd /usr/realtime*/testsuite/kern/latency; ./run
```

and then you should see something like this:

```
ubuntu:/usr/realtime-2.6.12-magma/testsuite/kern/latency$ ./run
*
*
* Type ^C to stop this application.
*
*
## RTAI latency calibration tool ##
# period = 100000 (ns)
# avrgtime = 1 (s)
# do not use the FPU
# start the timer
# timer_mode is oneshot
RTAI Testsuite - KERNEL latency (all data in nanoseconds)
RTH| lat min| ovl min| lat avg| lat max| ovl max| overruns
RTD|   -1571|   -1571|    1622|    8446|    8446|       0
RTD|   -1558|   -1571|    1607|    7704|    8446|       0
RTD|   -1568|   -1571|    1640|    7359|    8446|       0
RTD|   -1568|   -1571|    1653|    7594|    8446|       0
RTD|   -1568|   -1571|    1640|   10636|   10636|       0
RTD|   -1568|   -1571|    1640|   10636|   10636|       0
```

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

The last number in the column labeled "ovl max" is the most important. Write it down - you will need it later. It contains the worst latency measurement during the entire run of the test. In the example above, that is 10636 nano-seconds, or 10.6 micro-seconds, which is excellent. However the example only ran for a few seconds (it prints one line every second). You should run the test for at least several minutes; sometimes the worst case latency doesn't happen very often, or only happens when you do some particular action. I had one Intel motherboard that worked pretty well most of the time, but every 64 seconds it had a very bad 300uS latency. Fortunately that is fixable, see FixingDapperSMIIssues in the wiki found at wiki.linuxcnc.org.

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

Note that if you get high numbers, there may be ways to improve them. For example, one PC had very bad latency (several milliseconds) when using the onboard video. But a $5 used Matrox video card solved the problem - EMC does not require bleeding edge hardware.

20.1.2 Figure out what your drives expect

Different brands of stepper drives have different timing requirements on their step and direction inputs. So you need to dig out (or Google for) the data sheet that has your drive's specs.

For example, the Gecko G202 manual says this:
Step Frequency: 0 to 200 kHz

Step Pulse “0” Time: 0.5 uS min (Step on falling edge)
Step Pulse “1” Time: 4.5 uS min
Direction Setup: 1 uS min (20 uS min hold time after Step edge)

The Gecko G203V specifications are:
Step Frequency: 0 to 333 kHz
Step Pulse “0” Time: 2.0 uS min (Step on rising edge)
Step Pulse “1” Time: 1.0 uS min
Direction Setup:

200 nS (0.2uS) before step pulse rising edge
200 nS (0.2uS) hold after step pulse rising edge

A Xylotex drive datasheet has a nice drawing of the timing requirements, which are:

Minimum DIR setup time before rising edge of STEP Pulse 200nS Minimum
DIR hold time after rising edge of STEP pulse 200nS
Minimum STEP pulse high time 2.0uS
Minimum STEP pulse low time 1.0uS
Step happens on rising edge

Once you find the numbers, write them down too - you need them in the next step.

20.1.3 Choose your BASE_PERIOD

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

Let’s look at the Gecko example first. The G202 can handle step pulses that go low for 0.5uS and high for 4.5uS, it needs the direction pin to be stable 1uS before the falling edge, and remain stable for 20uS after the falling edge. The longest timing requirement is the 20uS hold time. A simple approach would be to set the period at 20uS. That means that all changes on the STEP and DIR lines are separated by 20uS. All is good, right?

Wrong! If there was ZERO latency, then all edges would be separated by 20uS, and everything would be fine. But all computers have some latency. Latency means lateness. If the computer has 11uS of latency, that means sometimes the software runs as much as 11uS later than it was supposed to. If one run of the software is 11uS late, and the next one is on time, the delay from the first to the second is only 9uS. If the first one generated a step pulse, and the second one changed the direction bit, you just violated the 20uS G202 hold time requirement. That means your drive might have taken a step in the wrong direction, and your part will be the wrong size.

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

The simplest way to avoid this problem is to choose a BASE_PERIOD that is the sum of the longest timing requirement of your drive, and the worst case latency of your computer. If you are running a Gecko with a 20uS hold time requirement, and your latency test said you have a maximum latency of 11uS, then if you set the BASE_PERIOD to 20+11 = 31uS (31000 nano-seconds in the ini file), you are guaranteed to meet the drive’s timing requirements.

But there is a tradeoff. Making a step pulse requires at least two periods. One to start the pulse, and one to end it. Since the period is 31uS, it takes 2x31 = 62uS to create a step pulse. That means

the maximum step rate is only 16,129 steps per second. Not so good. (But don't give up yet, we still have some tweaking to do in the next section.)

For the Xylotex, the setup and hold times are very short, 200nS each (0.2uS). The longest time is the 2uS high time. If you have 11uS latency, then you can set the BASE_PERIOD as low as 11+2=13uS. Getting rid of the long 20uS hold time really helps! With a period of 13uS, a complete step takes 2x13 = 26uS, and the maximum step rate is 38,461 steps per second!

But you can't start celebrating yet. Note that 13uS is a very short period. If you try to run the step generator every 13uS, there might not be enough time left to run anything else, and your computer will lock up. If you are aiming for periods of less than 25uS, you should start at 25uS or more, run EMC, and see how things respond. If all is well, you can gradually decrease the period. If the mouse pointer starts getting sluggish, and everything else on the PC slows down, your period is a little too short. Go back to the previous value that let the computer run smoothly.

In this case, sppose you started at 25uS, trying to get to 13uS, but you find that around 16uS is the limit - any less and the computer doesn't respond very well. So you use 16uS. With a 16uS period and 11uS latency, the shortest output time will be 16-11 = 5uS. The drive only needs 2uS, so you have some margin. Margin is good - you don't want to lose steps because you cut the timing too close.

What is the maximum step rate? Remember, two periods to make a step. You settled on 16uS for the period, so a step takes 32uS. That works out to a not bad 31,250 steps per second.

20.1.4 Use steplen, stepspace, dirsetup, and/or dirhold

In the last section, we got the Xylotex drive to a 16uS period and a 31,250 step per second maximum speed. But the Gecko was stuck at 31uS and a not-so-nice 16,129 steps per second. The Xylotex example is as good as we can make it. But the Gecko can be improved.

The problem with the G202 is the 20uS hold time requirement. That plus the 11uS latency is what forces us to use a slow 31uS period. But the EMC2 software step generator has some parameters that let you increase the various time from one period to several. For example, if steplen is changed from 1 to 2, then it there will be two periods between the beginning and end of the step pulse. Likewise, if dirhold is changed from 1 to 3, there will be at least three periods between the step pulse and a change of the direction pin.

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

If the computer has a latency of 11uS, then a combination of a 16uS base period, and a dirhold value of 2 ensures that we will always meet the timing requirements of the Gecko. For normal stepping (no direction change), the increased dirhold value has no effect. It takes two periods totalling 32uS to make each step, and we have the same 31,250 step per second rate that we got with the Xylotex.

The 11uS latency number used in this example is very good. If you work through these examples with larger latency, like 20 or 25uS, the top step rate for both the Xylotex and the Gecko will be lower. But the same formulas apply for calculating the optimum BASE_PERIOD, and for tweaking dirhold or other step generator parameters.

20.1.5 No Guessing!

For a fast AND reliable software based stepper system, you cannot just guess at periods and other configuration paremeters. You need to make measurements on your computer, and do the math to

ensure that your drives get the signals they need.

To make the math easier, I've created an Open Office spreadsheet (http://wiki.linuxcnc.org/uploads/StepTiming
You enter your latency test result and your stepper drive timing requirements and the spreadsheet calculates the optimum BASE_PERIOD. Next, you test the period to make sure it won't slow down or lock up your PC. Finally, you enter the actual period, and the spreadsheet will tell you the stepgen parameter settings that are needed to meet your drive's timing requirements. It also calculates the maximum step rate that you will be able to generate.

I've added a few things to the spreadsheet to calculate max speed and stepper electrical calculations.

Part VIII

Logique machine

Chapter 21

La programmation en Ladder

21.1 Introduction

La logique Ladder ou langage de programmation Ladder est une méthode pour tracer les schémas en logique électrique. Il s'agit maintenant d'un langage graphique vraiment populaire pour la programmation des automates programmables industriels (API). Il a été à l'origine inventé pour décrire la logique à relais. Son nom est fondé sur la constatation que les programmes dans cette langue ressemblent à une échelle (ladder), avec deux "rails" verticaux et, entre eux, une série "d'échelons". En Allemagne et ailleurs en Europe, le style consiste à placer les rails horizontaux, un en haut de la page et l'autre en bas avec les échelons verticaux dessinés séquentiellement de la gauche vers la droite.

Un programme en logique Ladder, également appelé schéma Ladder, est ressemblant au schéma d'un ensemble de circuits électriques à relais. C'est l'intérêt majeur du schéma Ladder de permettre à une large variété de personnels techniques, ingénieurs, techniciens électriciens, etc de le comprendre et de l'utiliser sans formation complémentaire grâce à cette ressemblance.

La logique Ladder est largement utilisée pour programmer les API, avec lesquels le contrôle séquentiel des processus de fabrication est requis. Le Ladder est utile pour les systèmes de contrôle simples mais critiques, ou pour reprendre d'anciens circuits à relais câblés. Comme les contrôleurs à logique programmable sont devenus plus sophistiqués, ils ont aussi été utilisés avec succès dans des systèmes d'automatisation très complexes.

Le langage Ladder peut être considéré comme un langage basé sur les règles, plutôt que comme un langage procédural. Un "échelon" en Ladder représente une règle. Quand elles sont mises en application avec des éléments électromécaniques, les diverses règles "s'exécutent" toutes simultanément et immédiatement. Quand elle sont mises en application dans la logique d'un automate programmable, les règles sont exécutées séquentiellement par le logiciel, dans une boucle. En exécutant la boucle assez rapidement, typiquement plusieurs fois par seconde, l'effet d'une exécution simultanée et immédiate est obtenu.

21.2 Exemple

Les composants les plus communs du Ladder sont les contacts (entrées), ceux-ci sont habituellement NC (normalement clos) ou NO (normalement ouvert) et les bobines (sorties).

- Le contact NO 

- Le contact NC 
- La bobine (sortie) 

Bien sûr, il y a beaucoup plus de composants dans le langage Ladder complet, mais la compréhension de ceux-ci aidera à appréhender le concept global du langage.

L'échelle se compose d'un ou plusieurs échelons. Ces échelons sont tracés horizontalement, avec les composants placés sur eux (entrées, sorties et autres), les composants sont évalués de la gauche vers la droite.

Cet exemple est un simple échelon: 

L'entrée B0 sur la gauche et un contact normalement ouvert, il est connecté sur la sortie Q0 sur la droite. Imaginez maintenant qu'une tension soit appliquée à l'extrême gauche, dès que B0 devient vraie (par exemple: l'entrée est activée, ou l'utilisateur a pressé le contact NO), la tension atteint l'extrême droite en traversant la bobine Q0. Avec comme conséquence que la sortie Q0 passe de 0 à 1.

Chapter 22

ClassicLadder

22.1 Introduction

ClassicLadder is a free implementation of a ladder interpreter, released under the LGPL. It has been written by Marc Le Douarain.

He describes the beginning of the project on his website:

> "I decided to program a ladder language only for test purposes at the start, in february 2001. It was planned, that I would have to participate to a new product after leaving the enterprise in which I was working at that time. And I was thinking that to have a ladder language in thoses products could be a nice option to considerate. And so I started to code the first lines for calculating a rung with minimal elements and displaying dynamically it under Gtk, to see if my first idea to realise all this works.
>
> And as quickly I've found that it advanced quite well, I've continued with more complex elements : timer, multiples rungs, etc...
>
> Voila, here is this work... and more : I've continued to add features since then."

ClassicLadder has been adapted to work with emc2's HAL, and is currently beeing distributed along with emc2. If there are issues/problems/bugs please report them to the Enhanced Machine Controller project.

22.2 Languages

The most common language used when working with ClassicLadder is 'ladder'. ClassicLadder allows one to use other variants (like sequential function chart - Grafcet) too, however those aren't covered by the current documentation.

In the next chapters the main components of ClassicLadder will be described.

22.3 Starting ClassicLadder

There are 2 components belonging to ClassicLadder. These must be placed in the main hal file or the Ladder Editor menu will be grayed out.

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

22.3.1 Realtime Module

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

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

22.3.2 Variables

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

Table 22.1: ClassicLadder realtime component options

Object name:	variable name:	Default value:
Number of rungs	(numRungs)	100
Number of bits	(numBits)	500
Number of word variables	(numWords)	100
Number of timers	(numTimers)	10
Number of monostables	(numMonostables)	10
Number of counters	(numCounters)	10
Number of hal inputs bit pins	(numPhysInputs)	15
Number of hal output bit pins	(numPhysOuputs)	15
Number of arithmetic expressions	(numArithmExpr)	50
Number of sections	(numSections)	10
Number of symbols	(numSymbols)	100
Number of S32 inputs	(numS32in)	0
Number of S32 outputs	(numS32out)	0

If you do not configure the number of ladder objects classicladder will use the default values. Objects of most interest are numPhysInputs and numPhysOutputs.

Changing these numbers will change the number of HAL bit pins available.

For example:

```
loadrt classicladder_rt numRungs=12 numBits=100 numWords=10 numTimers=10
numMonostables=10 numCounters=10 numPhysInputs=10 numPhysOutputs=10
numArithmExpr=100 numSections=4 numSymbols=200
```

22.3.3 Loading the ClassicLadder user module

Classicladder hal statements must be placed in the main .hal file for the menu to function.

To load the user module:

loadusr classicladder

To load a ladder file:

loadusr classicladder myladder.clp

To load the user module without the GUI:

loadusr classicladder –nogui

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

loadusr classicladder –modbus_port=port

To load the user module with a config file:

loadusr classicladder –config=file

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

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

Use the GUI when setting up your system then change it to –nogui when running. The only other thing you can do while loading the user module is specify a ladder program to load. ladder programs are specified by the .clp ending.

22.4 ClassicLadder GUI

If you load classicladder with the GUI it will display three windows: vars, section display, and section manager.

22.4.1 The Variables window

It displays some of the variable data and variable names. Notice all variable start with the % sign.

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

The check boxes allow you to set and un set variables but when classicladder is running hal will update the pins and change them.

Near the bottom are the %W variables. These are called word variable and represent positive and negative (signed) numbers and are used with compare and operate. By clicking on the variable, you can edit the number to display which ever you want.The edit box beside it is the number stored in the variable -you can change it- and the drop-down box beside that allow you to choose whether the number to be displayed is in hex, decimal or binary.

The %I variable represents HAL input bit pins. The %Q represents the relay coil and HAL output bit pins. The %B represents an internal relay coil or internal contact.

Figure 22.1: ClassicLadder Var window

22.4.2 The Section Display window

Most of the buttons are self explanitory:

The config button is not used in EMC.

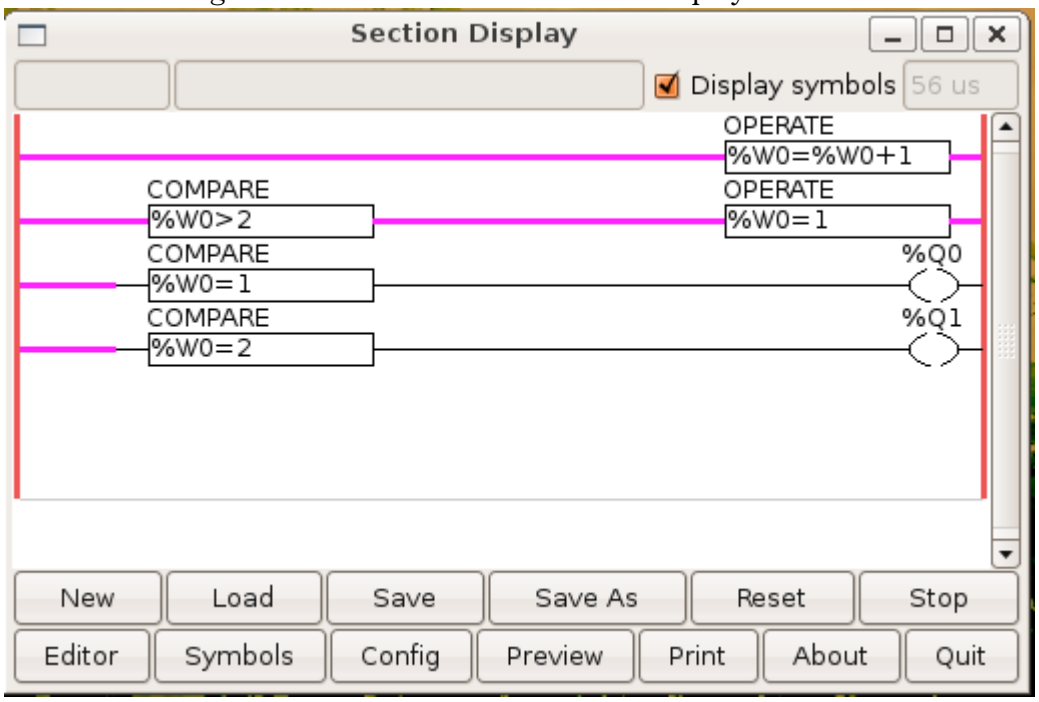
The symbols button will display an editable list of symbols for the variables (eg you can name the inputs, outputs, coils etc).

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

The quit button will only shut down the display-the ladder program will still run in the back ground.

The check box at the top right allows you to select whether variable names or symbol names are displayed

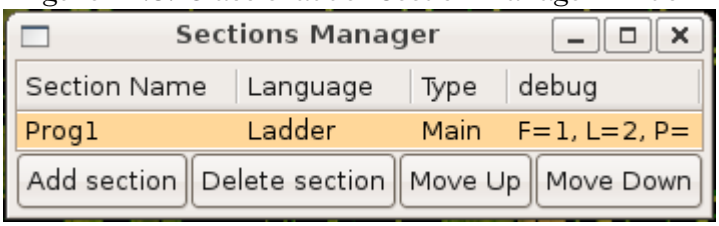
Figure 22.2: ClassicLadder Section Display window



22.4.3 The Section Manager window

This window allows you to name, create or delete sections. This is also how you name a subroutine for call coils.

Figure 22.3: ClassicLadder Section Manager window

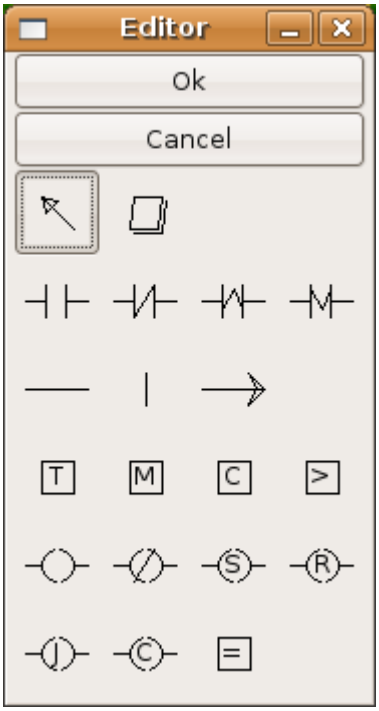


22.4.4 The Editor window

Starting from the top left image:

1. SELECTOR ARROW, ERASER
2. N.O., N.C. , RISING-EDGE ,FALLING-EDGE CONTACTS.
3. HORIZONTAL, VERTICAL , HORIZONTAL RUNNING-CONNECTIONS
4. TIMER, MONOSTABLE, COUNTER, COMPARE
5. N.O. COIL, N.C. COIL, SET COIL, RESET COIL
6. JUMP COIL, CALL COIL, OPERATE

Figure 22.4: ClassicLadder Editor window



A short description of each of the buttons:

- The SELECTOR ARROW button allows you to select existing objects and modify the information.
- The ERASER erases an object.
- The N.O. CONTACT is a normally open contact. It can be an extenal HAL-pin (%I) input contact, an internal-bit coil (%B) contact or a external coil (%Q) contact. The Hal-pin input contact is closed when the HAL-pin is true. The coil contacts are closed when the coresponding coil is active (%Q2 contact closes when %Q2 coil is active).
- The N.C. CONTACT is a normally closed contact. It is the same as the n.o. contact except that the contact is open when the hal-pin is true or the coil is active.
- The RISING-EDGE CONTACT is a contact that is closed when the HAL-pin goes from False to true, or the coil from not-active to active.
- The FALLING-EDGE CONTACT is a contact that is closed when the HAL-pin goes from true to false or the coil from active to not.
- The HORIZONTAL CONNECTION connects the 'signal' to objects horizontally.
- The VERTICAL CONNECTION connects the 'signal' to objects vertically.
- The HORIZONTAL-RUNNING CONNECTION is a quick way to connect a long run of 'signal wire' horizontally.
- The TIMER is a Timer Module.
- The MONOSTABLE is monostable module (one-shot)
- The COUNTER is a counter module.

- The COMPARE button allows you to compare variable to values or other variables. (eg %W1<=5 or %W1=%W2)
  The variable you can use are: W-words,T-timers,M-monostables,C-counters,X-sequential and their attributes D-done, E-empty, F-full, P-preset, R-running, and V-value (not all atributes are available to all variables) eg %T2.D.
  The math symbols are +,-,*,/,=,<,>,<=,>=,(,),^ (exponent),% (modulas),& (and),| (or),! (not).
  Math function are ABS (absolute), MOY (average). eg ABS(%W2)=1, MOY(%W1,%W2)<3 .
  Compare cannot be placed in the right most side of the section display.
- The OPERATE button allows you to assign values to variables. (eg %W2=7 or %W1=%W2) there are two math funtions MINI and MAXI that check a variable for maximum (0x80000000) and minimum values (0x05FFFFFF) (think signed values) and keeps them from going beyond.
  You may use all the math symbols and functions from above. OPERATE funtions can only be placed at the right most side of the section display.

22.5 ClassicLadder Variables

List of known variables :

Bxxx : Bit memory xxx (boolean)

Wxxx : Word memory xxx (32 bits integer)

Txx,R : Timer xx running (boolean, user read only)

Txx,D : Timer xx done (boolean, user read only)

Txx,V : Timer xx current value (integer, user read only)

Txx,P : Timer xx preset (integer)

Mxx,R : Monostable xx running (boolean)

Mxx,V : Monostable xx current value (integer, user read only)

Mxx,P : Monostable xx preset (integer)

Cxx,D : Counter xx done (boolean, user read only)

Cxx,E : Counter xx empty overflow (boolean, user read only)

Cxx,F : Counter xx full overflow (boolean, user read only)

Cxx,V : Counter xx current value (integer, user read only)

Cxx,P : Counter xx preset (integer)

Ixxx : Physical input xxx (boolean) - HAL input bit -

Qxxx : Physical output xxx (boolean) - HAL output bit -

Xxxx : Activity of step xxx (sequential language)

Xxxx,V : Time of activity in seconds of step xxx (sequential language)

22.6 Using JUMP COILs

JUMP COILs are used to 'JUMP' to another section-like a goto in BASIC programming language.

If you look at the top left of the sections display window you will see a small lable box and a longer comment box beside it. Now go to Editor->Modify then go back to the little box, type in a name.

Go ahead and add a comment in the comment section. This lable name is the name of this rung only and is used by the JUMP COIL to identify where to go.

When placing a JUMP COIL add it in the right most position and change the lable to the rung you want to JUMP to.

JUMP COILs should be placed as the last coil of a rung because of a bug. If there are coils after the JUMP COIL (in the same rung) they will be updated even if the JUMP COIL is true.[1]

22.7 Using CALL COILs

CALL COILs are used to go to a subroutine section then return-like a gosub in BASIC programming language.

If you go to the sections manager window hit the add section button. You can name this section, select what language it will use (ladder or sequential), and select what type (main or subroutine).

Select a subroutine number (SR0 for exampe). An empty section will be displayed and you can build your subroutine.

When your done that, go back to the section manager and click on the your 'main' section (default name prog1).

Now you can add a CALL COIL to your program. CALL COILs are to be placed at the right most position in the rung.

Remember to change the lable to the subroutine number you choose before.

There can only be one CALL COIL per rung-the rest wil not be called.

[1] If the JUMP COIL is true it should JUMP to the new rung right away and not update the rest of the coils of the current rung

Appendix A

Glossary of Common Terms Used in the EMC Documents

Appendix B

Glossary

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

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

Axis One of the computer control movable parts of the machine. For a typical vertical mill, the table is the X axis, the saddle is the Y axis, and the quill or knee is the Z axis. Additional linear axes parallel to X, Y, and Z are called U, V, and W respectively. Angular axes like rotary tables are referred to as A, B, and C.

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

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

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

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

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

Coordinate Measuring Machine A Coordinate Measuring Machine is used to make many accurate measurements on parts. These machines can be used to create CAD data for parts where no drawings can be found, when a hand-made prototype needs to be digitized for mold making, or to check the accuracy of machined or molded parts.

Display units The linear and angular units used for onscreen display.

DRO A Digital Read Out is a device attached to the slides of a machine tool or other device which has parts that move in a precise manner to indicate the current location of the tool with respect to some reference position. Nearly all DRO's use linear quadrature encoders to pick up position information from the machine.

EDM EDM is a method of removing metal in hard or difficult to machine or tough metals, or where rotating tools would not be able to produce the desired shape in a cost-effective manner. An excellent example is rectangular punch dies, where sharp internal corners are desired. Milling operations can not give sharp internal corners with finite diameter tools. A wire EDM machine can make internal corners with a radius only slightly larger than the wire's radius. A 'sinker' EDM cam make corners with a radius only slightly larger than the radius on the corner of the convex EDM electrode.

EMC The Enhanced Machine Controller. Initially a NIST project. EMC is able to run a wide range of motion devices.

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

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

Encoder A device to measure position. Usually a mechanical-optical device, which outputs a quadrature signal. The signal can be counted by special hardware, or directly by the parport with EMC2.

Feed Relatively slow, controlled motion of the tool used when making a cut.

Feed rate The speed at which a motion occurs. In manual mode, jog speed can be set from the graphical interface. In auto or mdi mode feed rate is commanded using a (f) word. F10 would mean ten units per minute.

Feedback A method (e.g., quadrature encoder signals) by which EMC receives information about the position of motors

Feed rate Override A manual, operator controlled change in the rate at which the tool moves while cutting. Often used to allow the operator to adjust for tools that are a little dull, or anything else that requires the feed rate to be "tweaked".

G-Code The generic term used to refer to the most common part programming language. There are several dialects of G-code, EMC uses RS274/NGC.

GUI Graphical User Interface.

- **General** A type of interface that allows communications between a computer and human (in most cases) via the manipulation of icons and other elements (widgets) on a computer screen.
- **EMC** An application that presents a graphical screen to the machine operator allowing manipulation of machine and the corresponding controlling program.

Home A specific location in the machine's work envelope that is used to make sure the computer and the actual machine both agree on the tool position.

ini file A text file that contains most of the information that configures EMC for a particular machine

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

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

kernel-space See real-time.

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

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

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

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

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

Offsets

Part Program A description of a part, in a language that the controller can understand. For EMC, that language is RS-274/NGC, commonly known as G-code.

Program Units The linear and angular units used for part programs.

Rapid Fast, possibly less precise motion of the tool, commonly used to move between cuts. If the tool meets the material during a rapid, it is probably a bad thing!

Real-time Software that is intended to meet very strict timing deadlines. Under Linux, in order to meet these requirements it is necessary to install RTAI or RTLINUX and build the software to run in those special environments. For this reason real-time software runs in kernel-space.

RTAI Real Time Application Interface, see https://www.rtai.org/, one of two real-time extensions for Linux that EMC can use to achieve real-time performance.

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

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

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

Servo Motor

Servo Loop

Spindle On a mill or drill, the spindle holds the cutting tool. On a lathe, the spindle holds the workpiece.

Stepper Motor A type of motor that turns in fixed steps. By counting steps, it is possible to determine how far the motor has turned. If the load exceeds the torque capability of the motor, it will skip one or more steps, causing position errors.

TASK The module within EMC that coordinates the overall execution and interprets the part program.

Tcl/Tk A scripting language and graphical widget toolkit with which EMC's most popular GUI's were written.

Units See "Machine Units", "Display Units", or "Program Units", above.

World Coordinates This is the absolute frame of reference. It gives coordinates in terms of a fixed reference frame that is attached to some point (generally the base) of the machine tool.

Appendix C

Legal Section

Appendix D

Legal Section

Copyright Terms

GNU Free Documentation License

GNU Free Documentation License Version 1.1, March 2000

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five). C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. H. Include an unaltered copy of this License. I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version. N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http:///www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index