

# Le manuel de HAL (Couche d'abstraction du matériel)

The EMC Team

16 mai 2009



# EMC<sup>2</sup>

## The Enhanced Machine Controller



[www.linuxcnc.org](http://www.linuxcnc.org)

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](mailto:emc-users@lists.sourceforge.net).

Copyright (c) 2000-9 LinuxCNC.org

---

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 no Invariant Sections, no Front-Cover Texts, and one Back-Cover Text : "This EMC Handbook is the product of several authors writing for linuxCNC.org. As you find it to be of value in your work, we invite you to contribute to its revision and growth." A copy of the license is included in the section entitled "GNU Free Documentation License". If you do not find the license you may order a copy from Free Software Foundation, Inc. 59 Temple Place, Suite 330 Boston, MA 02111-1307

# Table des matières

<b>Couverture</b>	<b>1</b>
<b>Table des matières</b>	<b>4</b>
<b>I Introduction et tutoriels</b>	<b>5</b>
<b>1 Introduction</b>	<b>6</b>
1.1 Qu'est-ce que HAL?	6
1.1.1 HAL est basé sur le système traditionnel d'étude des projets techniques	6
1.1.1.1 Choix des organes	6
1.1.1.2 Étude des interconnexions	7
1.1.1.3 Implémentation	7
1.1.1.4 Mise au point	7
1.1.2 En résumé	7
1.2 Concept de HAL	8
1.3 Composants HAL	9
1.3.1 Programmes externes attachés à HAL	9
1.3.2 Composants internes	9
1.3.3 Pilotes de matériels	10
1.3.4 Outils-Utilitaires	10
1.4 Tinkertoys, Erector Sets, Legos et le HAL	10
1.4.1 Une tour	10
1.4.2 Erector Sets (Meccano en France)	10
1.4.3 Tinkertoys	11
1.4.4 Un exemple en Lego	11
1.5 Problèmes de timing dans HAL	12
<b>2 Tutoriel de HAL</b>	<b>13</b>
2.1 Introduction	13
2.2 Un exemple simple	14
2.3 Visualiser HAL avec halmeter	18
2.4 Exemple plus complexe avec stepgen.	20
2.5 Voyons-y de plus près avec halscope.	24
2.5.1 Triggering	31

<b>II Les références de HAL</b>	<b>36</b>
<b>3 Informations générales</b>	<b>37</b>
3.1 Notation	37
3.1.1 Conventions typographiques	37
3.1.2 Noms	37
3.2 Conventions générales de nommage	37
3.3 Conventions de nommage des pilotes de matériels	38
3.3.1 Noms de pin/paramètre	38
3.3.1.1 Exemples	39
3.3.2 Noms des fonctions	39
3.3.2.1 Exemples	40
<b>4 Périphériques d'interfaces canoniques</b>	<b>41</b>
4.1 Entrée numérique (Digital Input)	41
4.1.1 Pins	41
4.1.2 Paramètres	41
4.1.3 Fonctions	41
4.2 Sortie numérique (Digital Output)	41
4.2.1 Pins	41
4.2.2 Paramètres	41
4.2.3 Fonctions	42
4.3 Entrée analogique (Analog Input)	42
4.3.1 Pins	42
4.3.2 Paramètres	42
4.3.3 Fonctions	42
4.4 Sortie analogique (Analog Output)	42
4.4.1 Paramètres	42
4.4.2 Fonctions	43
4.5 Codeur	43
4.5.1 Pins	43
4.5.2 Paramètres	43
4.5.3 Fonctions	44
<b>5 Outils et utilitaires pour HAL</b>	<b>45</b>
5.1 Halcmd	45
5.2 Halmeter	45
5.3 Halscope	45
<b>6 comp : un outil pour créer les modules HAL</b>	<b>47</b>
6.1 Introduction	47
6.2 Définitions	47
6.3 Création d'instance	48
6.4 Syntaxe	48
6.5 Stockage des données « par-instance »	50
6.6 Autres restrictions sur les fichiers comp	51
6.7 Conventions des macros	51

6.8 Composants avec une seule fonction . . . . .	52
6.9 « Personnalité » du composant . . . . .	52
6.10 Compiler un fichier <code>.comp</code> dans l'arborescence . . . . .	52
6.11 Compiler un composant temps réel hors de l'arborescence . . . . .	52
6.12 Compiler un composant de l'espace utilisateur hors de l'arborescence . . . . .	53
6.13 Exemples . . . . .	53
6.13.1 constant . . . . .	53
6.13.2 sincos . . . . .	53
6.13.3 out8 . . . . .	53
6.13.4 hal_loop . . . . .	54
6.13.5 arraydemo . . . . .	54
6.13.6 rand . . . . .	55
6.13.6.1 logic . . . . .	55
<b>7 Création de composants de l'espace utilisateur Python avec les modules de 'hal'</b>	<b>57</b>
7.1 Utilisation de base . . . . .	57
7.2 Composants de l'espace utilisateur et délais . . . . .	58
7.3 Créer les pins et les paramètres . . . . .	58
7.3.1 Changer le préfixe . . . . .	58
7.4 Lire et écrire les pins et les paramètres . . . . .	58
7.4.1 Pilotage des pins de sortie (HAL_OUT) . . . . .	59
7.4.2 Pilotage des pins bidirectionnelles (HAL_IO) . . . . .	59
7.5 Quitter . . . . .	59
7.6 Idées de projets . . . . .	59
 <b>III Annexes</b>	 <b>60</b>
<b>A Legal Section</b>	<b>61</b>
A.1 Copyright Terms . . . . .	61
A.2 GNU Free Documentation License . . . . .	61

**Première partie**

**Introduction et tutoriels**

# Chapitre 1

## Introduction

### 1.1 Qu'est-ce que HAL ?

HAL est le sigle de Hardware Abstraction Layer, le terme Anglais pour Couche d'Abstraction Matériel<sup>1</sup>. 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.

#### 1.1.1 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.

##### 1.1.1.1 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.

---

<sup>1</sup>Note du traducteur : nous garderons le sigle HAL dans toute la documentation.

### 1.1.1.2 Étude des interconnexions

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.

### 1.1.1.3 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.

### 1.1.1.4 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.

## 1.1.2 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 indispensable 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 boîtes noires logicielles.

## 1.2 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 mêmes restrictions, qui sont fondées sur leur type. Les "pins" et les "signals" ont tous un type, un "signals" ne peut 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 très 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 "thread" tournera.

Prenons un exemple, supposons que nous avons un composant de port parallèle nommé "hal\_parport". Ce composant définit 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 là 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". Ces deux fonctions font partie du pilote "hal\_parport".

## 1.3 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.

### 1.3.1 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.

### 1.3.2 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/cosinoïdale/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.)

### 1.3.3 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\_parpport** 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

### 1.3.4 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, complètement fonctionnel pour les signaux HAL.

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

## 1.4 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.

### 1.4.1 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.

### 1.4.2 Erector Sets<sup>2</sup> (Meccano en France)

C'était une grande série de boîtes de construction en métal, des tôles perforées, plates ou en cornières, toutes avaient des trous régulièrement 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 âgés 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

---

<sup>2</sup>Le jeu Erector Set est une invention de AC Gilbert

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.

### 1.4.3 Tinkertoys<sup>3</sup>

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 spatiale 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 ascenseurs 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 2) 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.

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.

### 1.4.4 Un exemple en Lego<sup>4</sup>

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

<sup>3</sup>Tinkertoy est maintenant registered trademark of the Hasbro company.

<sup>4</sup>The Lego name is a trademark of the Lego company.

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.

## 1.5 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.

Dans EMC on peut avoir un thread à 50 $\mu$ s 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 2

## Tutoriel de HAL

### 2.1 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.

#### 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 éléments 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.

#### 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 : loa<TAB>
halcmd : load
halcmd : loadrt
halcmd : loadrt deb<TAB>
halcmd : loadrt debounce
```

#### 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.

## 2.2 Un exemple simple

### Charger un composant temps réel

Pour ce tutoriel, nous allons supposer que vous avez installé avec succès le CD-Live ou que vous avez compilé correctement l'arborescence `emc2/src`. Si nécessaire, invoquez 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 :

```
~$ cd emc2
~/emc2$ halrun
halcmd :
```

Avec l'OS temps réel et RTAPI chargés, vous pouvez passer 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.

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 ?? de ce document. Il s'agit d'un composant temps réel, mis en oeuvre comme un module du noyau Linux. Pour charger `siggen` utiliser la commande de HAL, `loadrt` :

```
halcmd : loadrt siggen
```

### 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 5.1 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
3 RT siggen ready
2 User halcmd10190 10190 ready
```

Puisque `halcmd` lui même est un composant HAL, il sera toujours présent dans la liste. 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. La liste montre aussi le composant `siggen` que nous avons 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
3 float IN 1 siggen.0.amplitude
```

```

3 float OUT 0 siggen.0.cosine
3 float IN 1 siggen.0.frequency
3 float IN 0 siggen.0.offset
3 float OUT 0 siggen.0.sawtooth
3 float OUT 0 siggen.0.sine
3 float OUT 0 siggen.0.square
3 float OUT 0 siggen.0.triangle

```

Cette commande affiche toutes les pins présentes dans le HAL. Un système complexe peut avoir plusieurs dizaines ou centaines de pins. Mais pour le moment il y a seulement huit pins. Toutes ces huit 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, certaines pins ont une valeur de zéro.

L'étape suivante consiste à examiner les paramètres :

```

halcmd : show param
Parameters :
Owner Type Dir Value Name
3 s32 RO 0 siggen.0.update.time
3 s32 RW 0 siggen.0.update.tmax

```

La commande “show param” affiche tous les paramètres de HAL. Pour le moment chaque paramètre a la valeur par défaut attribuée quand le composant a été chargé. Notez dans la colonne `Dir`, les paramètres marqués `-W` sont en écriture possible, pour ceux qui ne sont jamais modifiés par le composant lui-même, mais qui sont modifiables 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 débogage, 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<sup>1</sup>.

## 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 )

```

<sup>1</sup>Les champs `codeaddr` et `arg` ont été utilisés pendant le développement et devraient probablement disparaître.

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 avons 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
3 float IN 1 siggen.0.amplitude
3 float OUT -0.9406941 siggen.0.cosine
3 float IN 1 siggen.0.frequency
3 float IN 0 siggen.0.offset
3 float OUT -0.1164055 siggen.0.sawtooth
3 float OUT 0.379820 siggen.0.sine
3 float OUT -1 siggen.0.square
3 float OUT -0.7728110 siggen.0.triangle
halcmd : show pin
Component Pins :
Owner Type Dir Value Name
3 float IN 1 siggen.0.amplitude
3 float OUT 0.9958036 siggen.0.cosine
3 float IN 1 siggen.0.frequency
3 float IN 0 siggen.0.offset
3 float OUT 0.9708287 siggen.0.sawtooth
3 float OUT -0.09151597 siggen.0.sine
3 float OUT 1 siggen.0.square
3 float OUT 0.9416574 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.

## 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 : show param
Parameters :
Owner Type Dir Value Name
3 s32 RO 397 siggen.0.update.time
3 s32 RW 109100 siggen.0.update.tmax
halcmd : show pin
Component Pins :
Owner Type Dir Value Name
3 float IN 5 siggen.0.amplitude
3 float OUT -4.179375 siggen.0.cosine
3 float IN 1 siggen.0.frequency
3 float IN 0 siggen.0.offset
3 float OUT 0.9248036 siggen.0.sawtooth
3 float OUT -2.744599 siggen.0.sine
3 float OUT 5 siggen.0.square
3 float OUT -3.150393 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.

## 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 réellement 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
# pin aliases
# signals
# nets
# parameter values
setp siggen.0.update.tmax 14687
# 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
halcmd : exit
~/emc2$
```

## 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  
halcmd : exit  
~/emc2$
```

## 2.3 Visualiser 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
```

### Lancement de halmeter

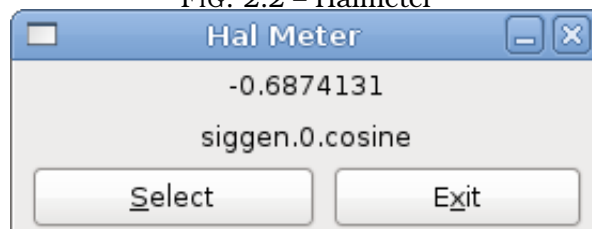
À ce stade, nous avons chargé le composant `siggen` qui est en cours d'exécution. Nous pouvons lancer `halmeter`. Puisque `halmeter` est une application graphique, X doit être actif.

```
halcmd : loadusr halmeter
```

Dans le même temps, une fenêtre s'ouvre sur votre écran, demandant de sélectionner le signal à observer [2.1](#).

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.cosine` en premier, il suffit de cliquer sur elle puis sur le bouton 'Close'. Le dialogue de sélection se ferme et la mesure s'affiche, quelque chose comme sur la figure [2.2](#).

FIG. 2.2 – Halmeter



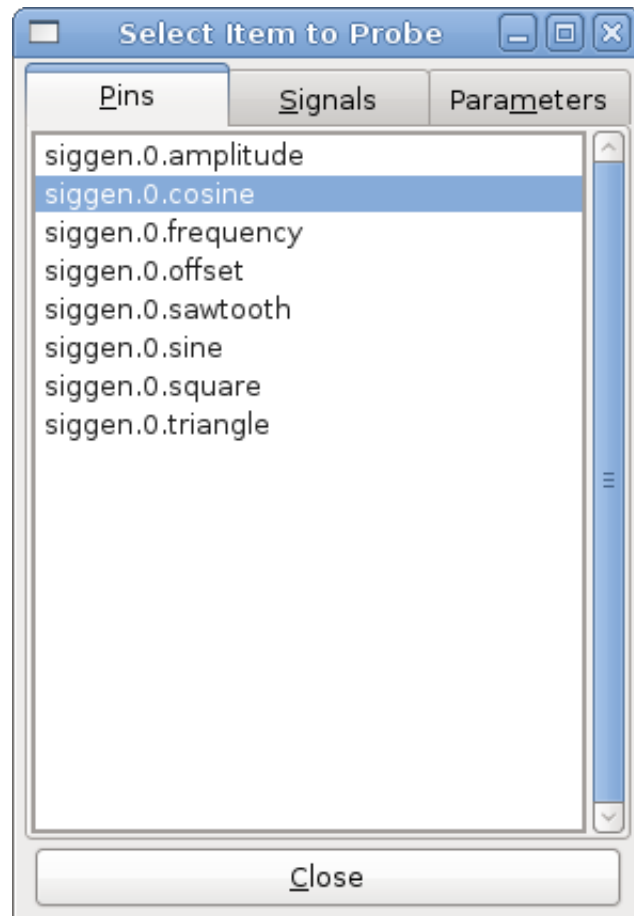


FIG. 2.1 – Fenêtre de sélection de HAL Meter

Pour modifier ce qui est affiché sur Hal meter pressez le bouton "Select" qui vous ramènera à la fenêtre de sélection précédente.

Vous devriez voir la valeur évoluer puisque siggen génère une onde cosinusoidale. Halmeter rafraîchi son affichage environ 5 fois par seconde.

Pour éteindre halmeter, cliquer sur le bouton exit.

Pour visualiser plusieurs pins, signaux ou paramètres en même temps, il est possible d'ouvrir plusieurs halmeters. La fenêtre de halmeter est intentionnellement petite justement pour permettre d'en ouvrir un grand nombre sur le même écran.

## 2.4 Exemple plus complexe avec stepgen.

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 bibliothèques de HAL en faisant :

```
halcmd : exit
~/emc2$ halrun
```

### 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 :<sup>2</sup>

Dans cet exemple nous utiliserons le type de contrôle "velocity" du composant stepgen.

```
halrun : loadrt stepgen step_type=0,0 ctrl_type=v,v
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 50 us 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 à HAL. Cette fois, nous aurons beaucoup plus de pins et de paramètres que précédemment :

```
halcmd : show pin
Component Pins :
Owner Type Dir Value Name
3 float IN 1 siggen.0.amplitude
3 float OUT 0 siggen.0.cosine
3 float IN 1 siggen.0.frequency
3 float IN 0 siggen.0.offset
3 float OUT 0 siggen.0.sawtooth
3 float OUT 0 siggen.0.sine
3 float OUT 0 siggen.0.square
3 float OUT 0 siggen.0.triangle
3 float OUT 0 stepgen.0.counts
2 bit OUT FALSE stepgen.0.dir
2 bit IN FALSE stepgen.0.enable
2 float IN 0 stepgen.0.position-fb
2 float OUT 0 stepgen.0.step
2 bit OUT FALSE stepgen.0.velocity-cmd
2 s32 OUT 0 stepgen.1.counts
2 bit OUT FALSE stepgen.1.dir
2 bit IN FALSE stepgen.1.enable
2 float IN 0 stepgen.1.position-fb
2 float OUT 0 stepgen.1.step
2 bit OUT FALSE stepgen.1.velocity-cmd
```

---

<sup>2</sup>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.

```

halcmd : show param
Parameters :
Owner Type Dir Value Name
3 s32 RO 0 siggen.0.update.time
3 s32 RW 0 siggen.0.update.tmax
2 u32 RW 00000001 stepgen.0.dirhold
2 u32 RW 00000001 stepgen.0.dirsetup
2 float RO 0 stepgen.0.frequency
2 float RW 0 stepgen.0.maxaccel
2 float RW 0 stepgen.0.maxvel
2 float RW 1 stepgen.0.position-scale
2 s32 RO 0 stepgen.0.rawcounts
2 u32 RW 00000001 stepgen.0.steplen
2 u32 RW 00000001 stepgen.0.stepspace
2 u32 RW 00000001 stepgen.1.dirhold
2 u32 RW 00000001 stepgen.1.dirsetup
2 float RO 0 stepgen.1.frequency
2 float RW 0 stepgen.1.maxaccel
2 float RW 0 stepgen.1.maxvel
2 float RW 1 stepgen.1.position-scale
2 s32 RO 0 stepgen.1.rawcounts
2 u32 RW 00000001 stepgen.1.steplen
2 u32 RW 00000001 stepgen.1.stepspace
2 s32 RO 0 stepgen.capture-position.time
2 s32 RW 0 stepgen.capture-position.tmax
2 s32 RO 0 stepgen.make-pulses.time
2 s32 RW 0 stepgen.make-pulses.tmax
2 s32 RO 0 stepgen.update-freq.time
2 s32 RW 0 stepgen.update-freq.tmax

```

## Connecter des pins avec des signaux

Nous avons donc deux générateurs d'impulsions de pas 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 de pas. Nous voulons déplacer la table en ronds. Pour ce faire, nous allons envoyer un signal cosinusoidal 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 de pas. 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
signals :
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 le nom du signal. Les flèches donnent la direction du flux de données, dans

ce cas, le flux va de la pin `siggen.0.cosine` vers le signal `X-vel`. Maintenant, connectons `X-vel` à l'entrée "velocity" du générateur d'impulsions de pas :

```
halcmd : net X-vel => stepgen.0.velocity-cmd
```

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 de pas. 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 => stepgen.1.velocity-cmd
```

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

```
halcmd : show sig
Signals :
Type Value Name (linked to)
float 0 X-vel
<== siggen.0.cosine
==> stepgen.0.velocity-cmd
float 0 Y-vel
<== siggen.0.sine
==> stepgen.1.velocity-cmd
```

La commande « `show sig` » montre clairement comment les flux de données circulent dans HAL. Par exemple, le signal `X-vel` provient de la pin `siggen.0.cosine` et va vers la pin `stepgen.0.velocity-cmd`.

## 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
00004 d8a3a120 d8bd322c YES 0 siggen.0.update
00003 d8bf45b0 d8bd30b4 YES 0 stepgen.capture-position
00003 d8bf42c0 d8bd30b4 NO 0 stepgen.make-pulses
00003 d8bf46b0 d8bd30b4 YES 0 stepgen.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 relatives au générateur d'impulsions de pas :

La première, `stepgen.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 `stepgen.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, `stepgen.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 `stepgen.0.update`. Nous voulons lancer `freqgen.update-freq` pour charger les nouvelles valeurs dans le générateur d'impulsions. Finalement nous lancerons `stepgen.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 `stepgen.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 `stepgen.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 `stepgen.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 stepgen.update-freq slow
halcmd : addf stepgen.make-pulses fast
```

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

```
halcmd : show thread
Realtime Threads :
Period FP Name (Time, Max-Time)
988960 YES slow ( 0, 0 )
1 siggen.0.update
2 stepgen.update-freq
49448 NO fast ( 0, 0 )
1 stepgen.make-pulses
```

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

## 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 10 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 `stepgen.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 stepgen.0.position-scale 10000  
halcmd : setp stepgen.1.position-scale 10000  
halcmd : setp stepgen.0.enable 1  
halcmd : setp stepgen.1.enable 1
```

Cela signifie que, avec la pin `stepgen.0.velocity-cmd` 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.

## 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.

## 2.5 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`.

### 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 2.3.

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" et laissez le multiplicateur à 1. Nous allons aussi passer la longueur d'enregistrement à 4000 samples (échantillons), de sorte que nous puissions utiliser jusqu'à 4 voies 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 2.4.

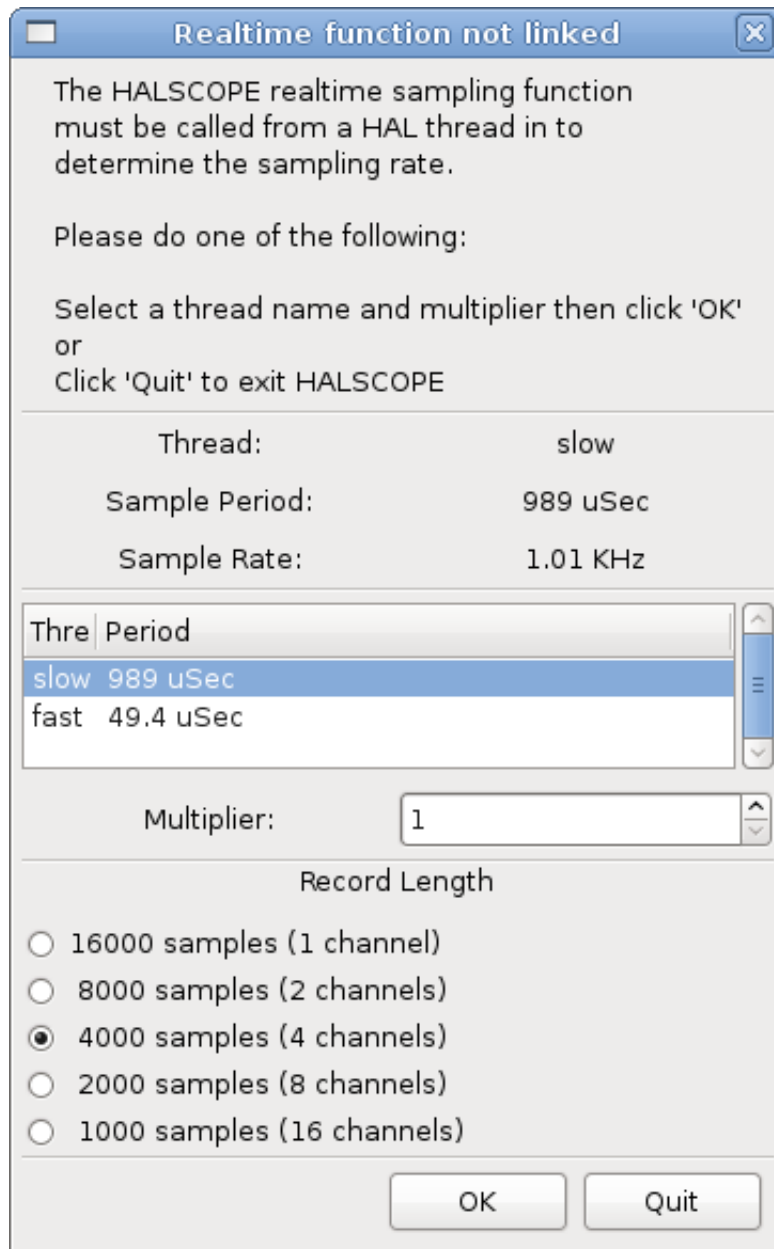


FIG. 2.3 – Dialogue "Realtime function not linked"

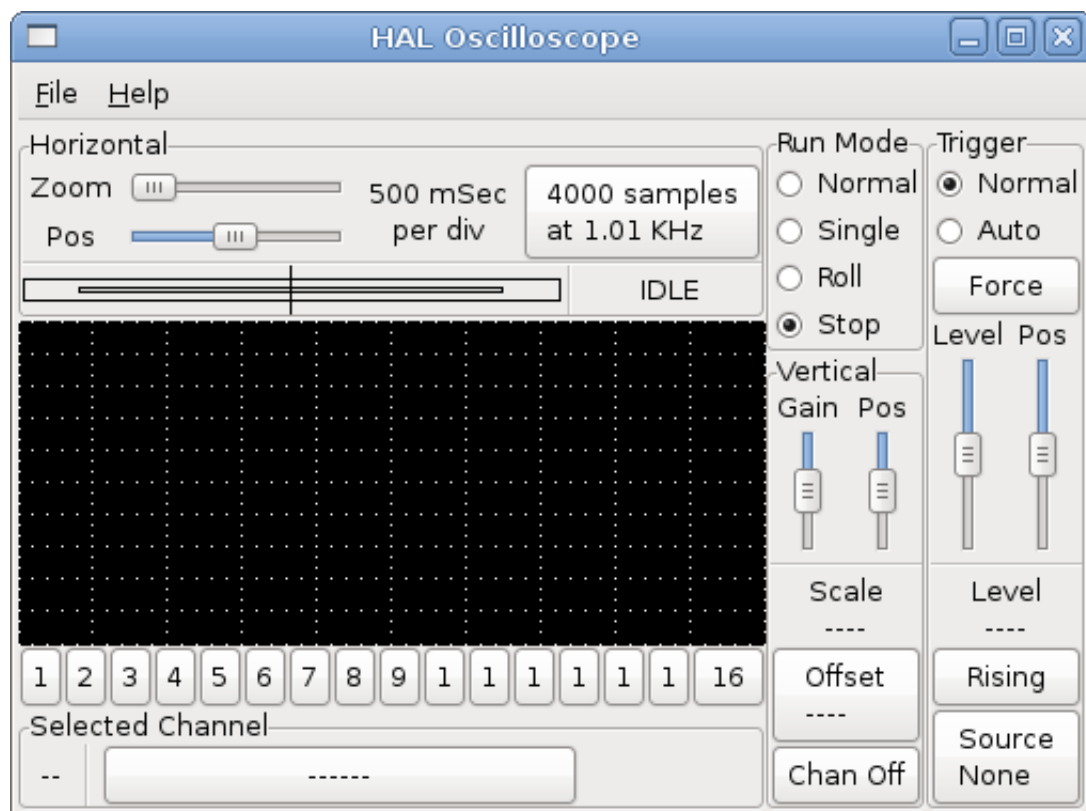


FIG. 2.4 – Fenêtre initiale du scope

## 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 voies, mais le nombre de voies utilisables à un moment donné dépend de la longueur d'enregistrement, plus il y a de voies, plus les enregistrements doivent être courts, car la mémoire disponible pour l'enregistrement est fixée à environ 16000 échantillons.

Les boutons des voies se situent en dessous de l'écran du scope. Cliquez le bouton “1” et vous verrez apparaître le dialogue de sélection “Select Channel Source”, pour sélectionner le signal à afficher sur la voie 1, comme sur la figure 2.5. Ce dialogue est très similaire à celui utilisé par Halmeter.

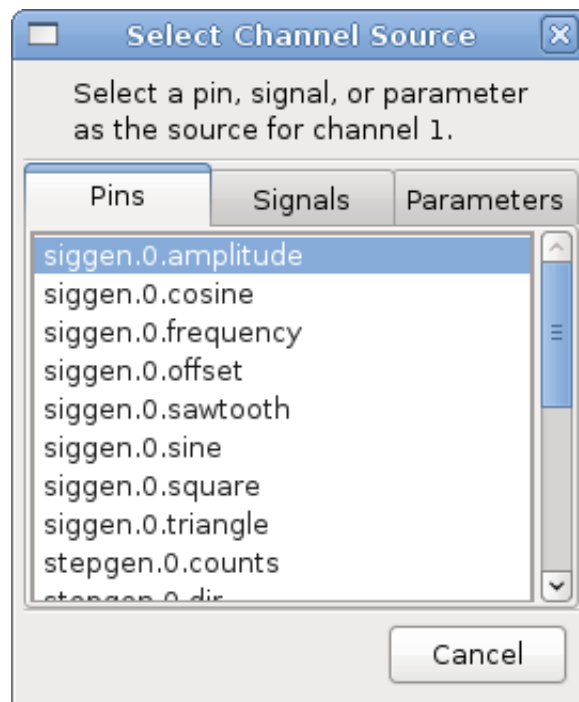
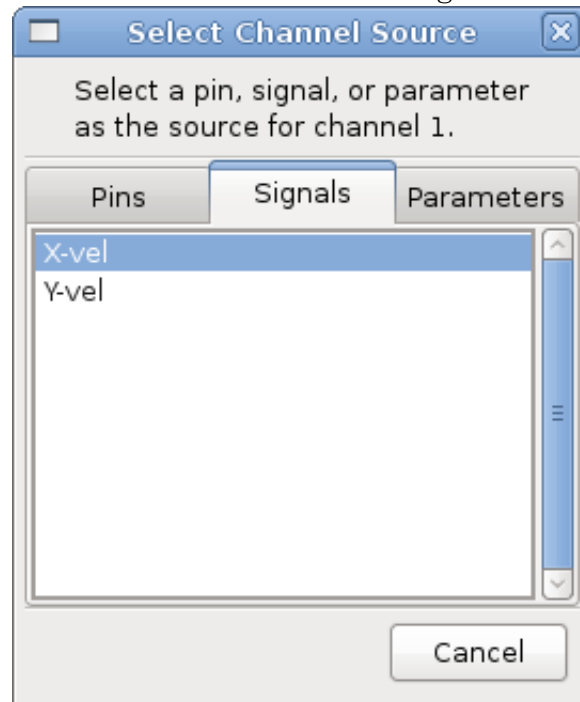


FIG. 2.5 – Dialogue de sélection des sources

Nous aimerions bien regarder les signaux que nous avons défini précédemment, pour cela, cliquons sur l'onglet “Signals” et le dialogue affichera tous les signaux existants dans le HAL (seulement deux dans notre exemple).2.6

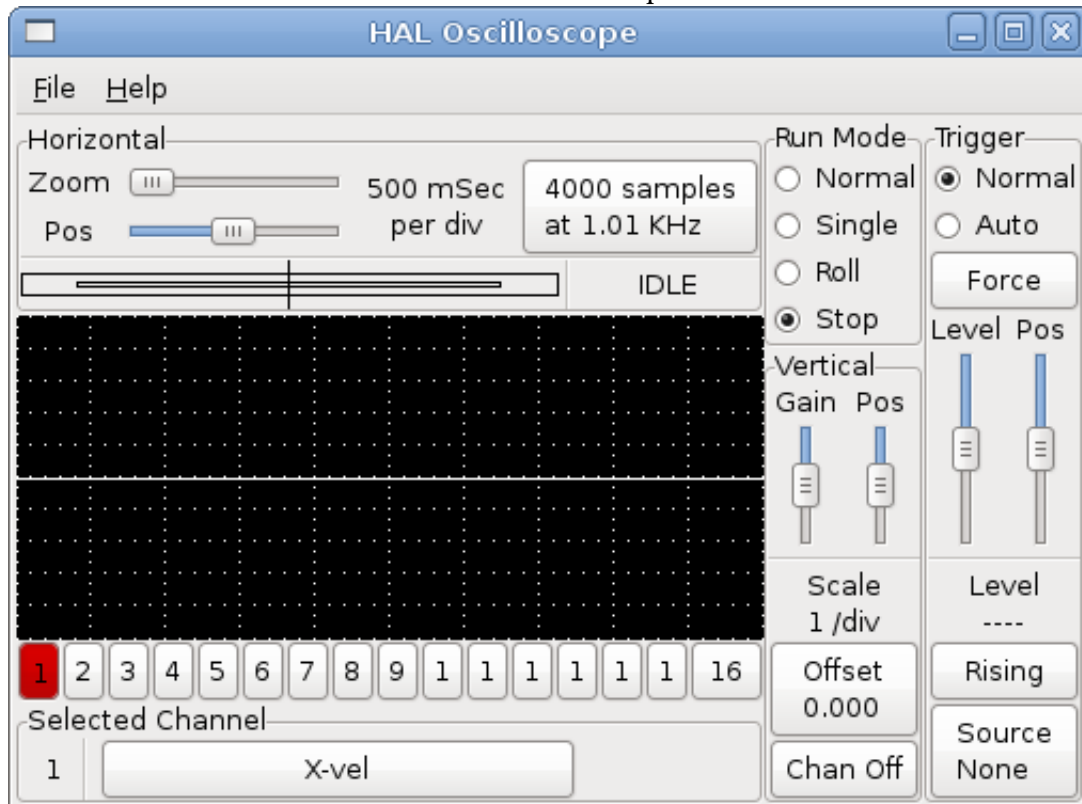
Pour choisir un signal, il suffit de cliquer dessus. Dans notre cas, nous voulons utiliser la voie 1 pour afficher le signal “X-vel”. Lorsque l'on clique sur “X-vel”, la fenêtre se ferme et le canal a été sélectionné.

FIG. 2.6 – Sélection du signal



Le bouton de la voie 1 est pressé, le numéro de la voie 1 et le nom “ X-vel” apparaissent sous la rangée de boutons. L’affichage indique toujours la voie sélectionnée, vous pouvez avoir beaucoup de voies sur l’écran, mais celle qui est active sera en surbrillance.

FIG. 2.7 – Halscope



Les différents contrôles comme la position verticale et l'amplitude sont toujours relatifs à la voie 1. Pour ajouter un signal sur la voie 2, cliquer sur le bouton "2". Dans la fenêtre de dialogue, cliquer sur l'onglet "Signals", puis cliquer 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".

## 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 2.8.

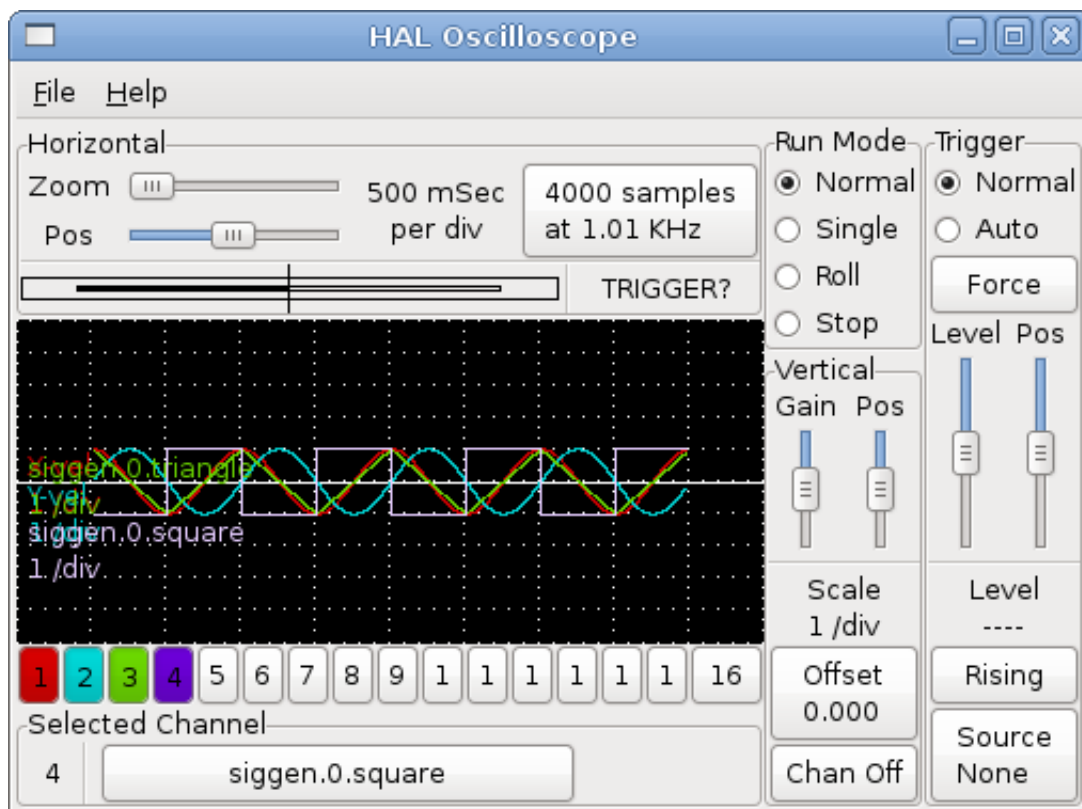


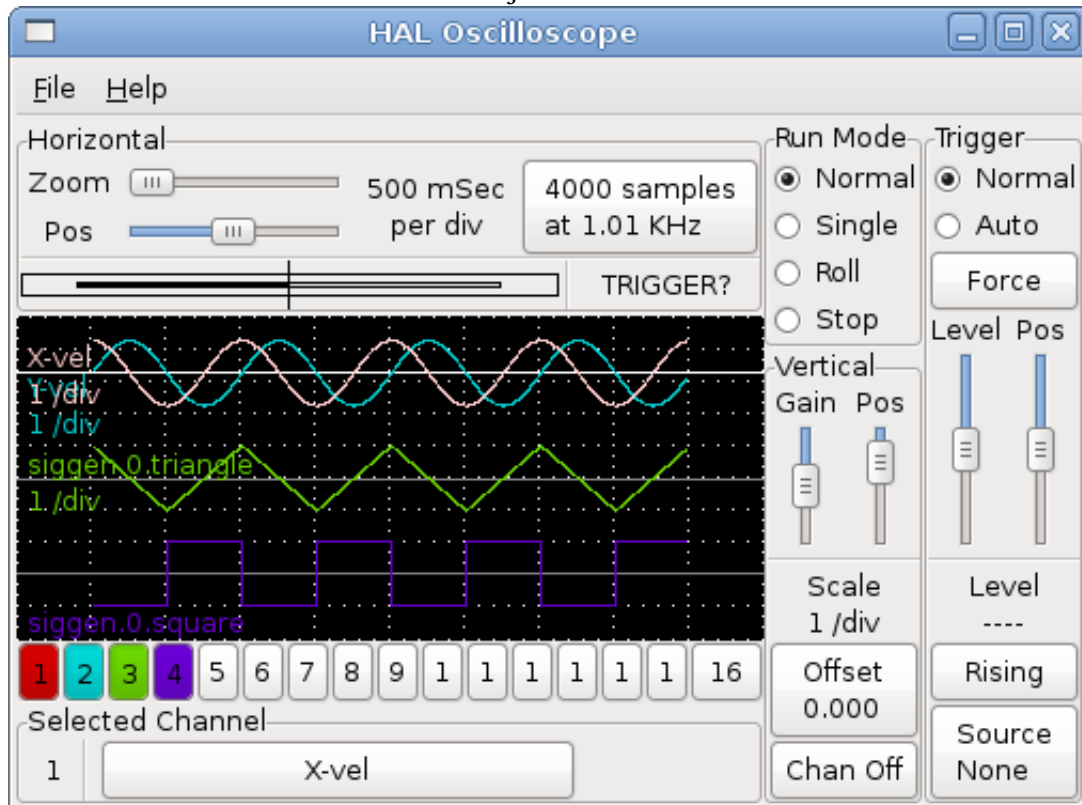
FIG. 2.8 – Capture d'ondes

Le grand bouton "Selected Channel" en bas, indique que la voie 4 est actuellement sélectionnée, donc, qu'elle correspond à la pin "siggen.0.square". Essayez de cliquer sur les autres voies pour mettre leurs traces en évidence.

## 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 la voie actuellement sélectionnée. 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 de halscope dans la section 5.3 pour plus de détails).

FIG. 2.9 – Ajustement vertical



### 2.5.1 Triggering

L'utilisation du bouton "Force" n'est parfois pas satisfaisante pour déclencher le scope. Pour régler un déclenchement (triggering) réel, cliquer 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 2.10). Sélectionner la sonde à utiliser pour déclencher en cliquant dessus. Pour notre exemple nous utilisons 3 canaux, essayons l'onde triangle.

Après avoir défini les sources de déclenchement, il est possible d'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

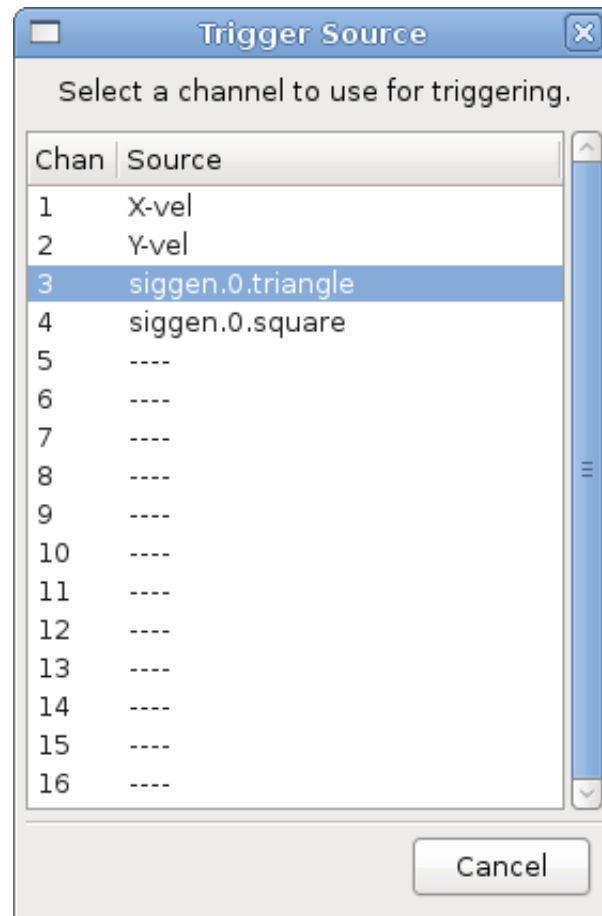


FIG. 2.10 – Dialogue des sources de déclenchement

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 2.11.

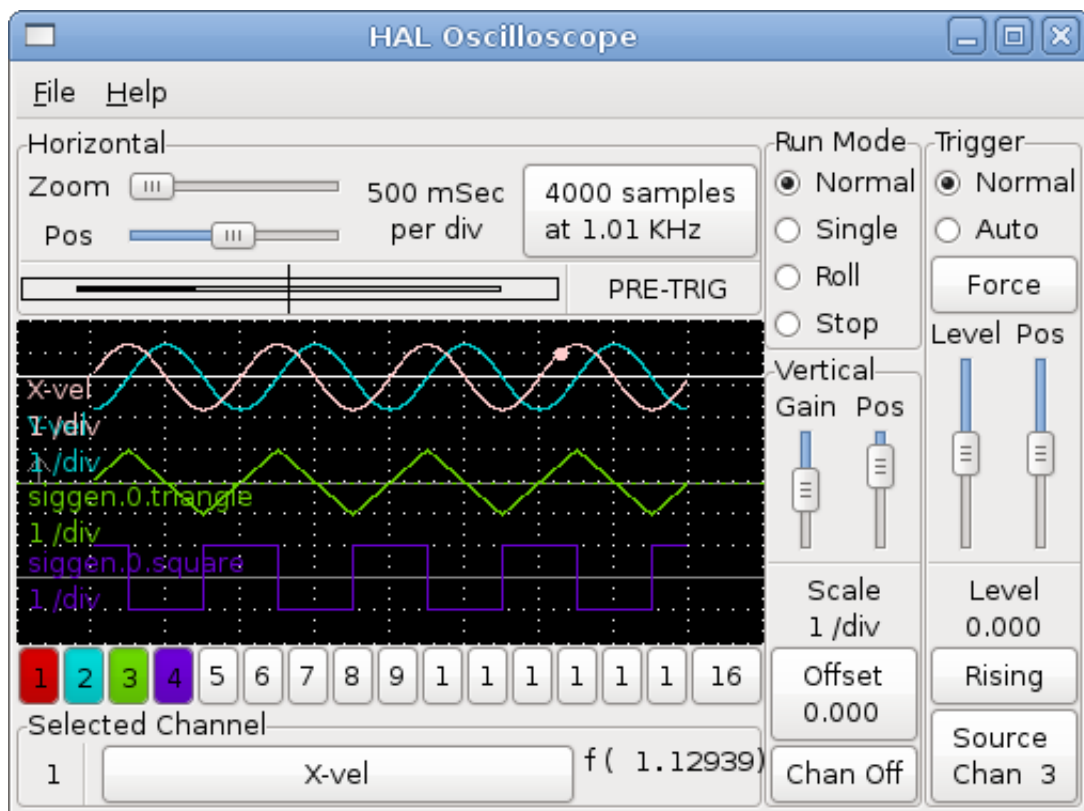


FIG. 2.11 – Formes d'ondes avec déclenchement

## 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 50 us de long, l'échantillonnage à 1kHz n'est pas assez rapide. Pour changer le taux d'échantillonnage, cliquer sur le bouton qui affiche la longueur de l'enregistrement et l'échantillonnage pour avoir le dialogue "Select Sample Rate", figure 2.12. Pour notre exemple, nous cliquerons sur le thread "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.

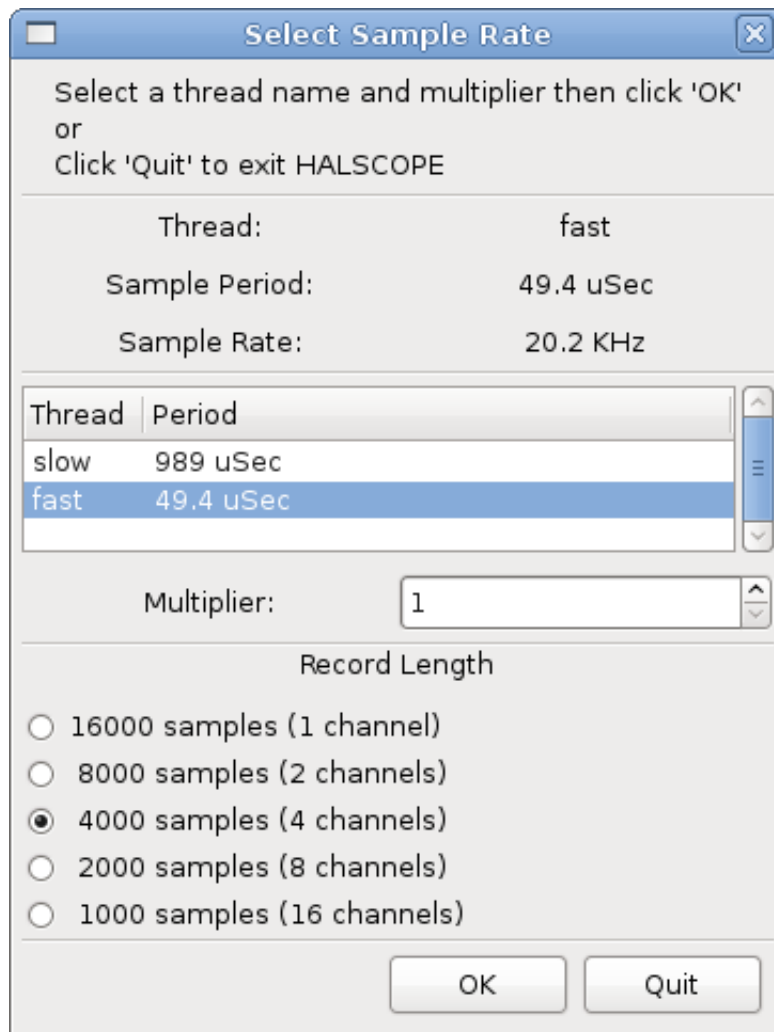


FIG. 2.12 – Dialogue de choix d'échantillonnage

## Plus de voies

Maintenant regardons les impulsions de pas. Halscope dispose de 16 voies, mais pour cet exemple, nous en utilisons seulement 4 à la fois. Avant de sélectionner toute autre voie, nous avons besoin d'en éteindre certaines. Cliquer sur la voie 2, puis sur le bouton "Off" sous le groupe "vertical". Ensuite, cliquez sur la voie 3, la mettre off et faire de même pour la voie 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 la voie 3 comme source de déclenchement. Pour ajouter de nouvelles voies, sélectionner la voie 5, choisir la pin "stepgen.1.dir", puis la voie 6 et sélectionner "stepgen.1.step". Ensuite, cliquer 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 (voie 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 la voie 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 2.13.

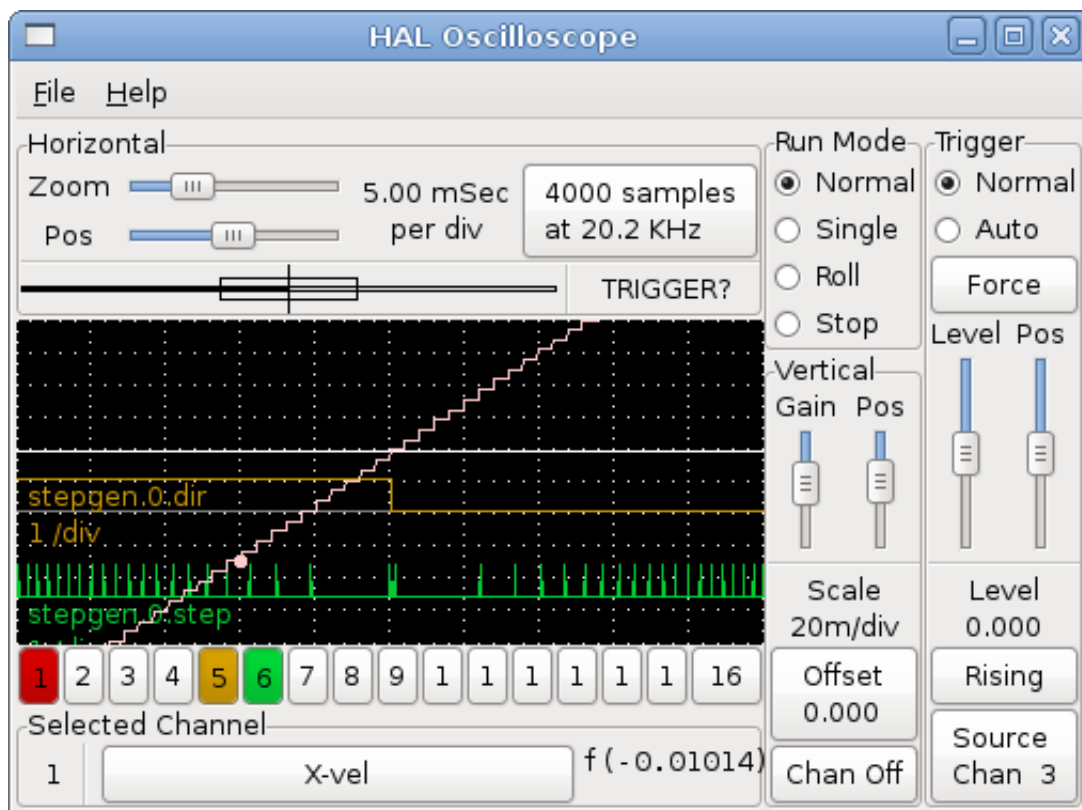


FIG. 2.13 – Observer les impulsions de pas

## 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 voies à la fois, il y aura 20000 échantillons par voie. (Si scope\_rt est déjà chargé, l'argument numérique passé à halscope sera sans effet)

**Deuxième partie**

**Les références de HAL**

# Chapitre 3

## Informations générales

### 3.1 Notation

#### 3.1.1 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.

#### 3.1.2 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.

### 3.2 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 reconnaît 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 reconnaître de telles structures si les noms fournissent les informations nécessaires. Pour cela, tous les modules de HAL devraient suivre 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.<sup>1</sup>
- Utiliser seulement des caractères minuscules, lettres et chiffres.

### 3.3 Conventions de nommage des pilotes de matériels<sup>2</sup>

#### 3.3.1 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.<sup>3</sup>

**<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 couleurs, 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.

**<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

<sup>1</sup> Les caractères soulignés ont été enlevés, mais il reste quelques cas de mélange de casses, par exemple "pid.0.Pgain" au lieu de "pid.0.p-gain".

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

<sup>3</sup> 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 porteront 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.

numéros de canaux, “chan-num”, commencent à zéro et s’incrémentent.<sup>4</sup> Si plusieurs périphériques sont installés, les numéros 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 inférieurs 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 quatre 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 4.

### 3.3.1.1 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.

### 3.3.2 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 auxquelles elles auront accès. Si une simple fonction lit toutes les entrées/sorties fournies par la carte, <io-type> n’est pas utilisé.<sup>5</sup>

**<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.

<sup>4</sup>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.

<sup>5</sup>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.

**3.3.2.1 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 4

# Périphériques d'interfaces canoniques

Les sections qui suivent expliquent les pins, paramètres et fonctions 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.

### 4.1 Entrée numérique (Digital Input)

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

#### 4.1.1 Pins

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

#### 4.1.2 Paramètres

- Aucun

#### 4.1.3 Fonctions

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

### 4.2 Sortie numérique (Digital Output)

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

#### 4.2.1 Pins

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

#### 4.2.2 Paramètres

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

### 4.2.3 Fonctions

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

## 4.3 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.

### 4.3.1 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**

### 4.3.2 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.

### 4.3.3 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.

## 4.4 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 dépend 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**.

### 4.4.1 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.

### 4.4.2 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  $(\text{scale} * \text{value}) + \text{offset}$  sont envoyés en sortie de l'adc. Si enable est faux, la sortie passe à 0.

## 4.5 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).

### 4.5.1 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 a été satisfaite.

### 4.5.2 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 a 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.

### 4.5.3 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 5

# Outils et utilitaires pour HAL

### 5.1 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 2 donne de nombreux exemples d'utilisation de halcmd.

### 5.2 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 apparaître, 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 plusieurs 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.

Se référer à la section 2.3 Halmeter dans le tutoriel.

### 5.3 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.

Se référer à la section Halscope [2.5](#) dans le tutoriel, qui explique les bases de son utilisation.

## Chapitre 6

# *comp* : un outil pour créer les modules HAL

### 6.1 Introduction

Ecrire un composant de HAL peut se révéler être une tâche ennuyeuse, la plupart de cette tâche consiste à appeler des fonctions `rtapi_` et `hal_` et à contrôler les erreurs associées à ces fonctions. *comp* va écrire tout ce code pour vous, automatiquement.

Compiler un composant de HAL est également beaucoup plus simple en utilisant *comp*, que le composant fasse partie de l'arborescence d'emc2, ou qu'il en soit extérieur.

Par exemple, cette portion des blocks « ddt » fait environ 80 lignes de code. Le composant équivalent est vraiment très court quand il est écrit en utilisant le préprocesseur *comp* :

```
component ddt "Calcule la dérivée de la fonction d'entrée";
pin in float in;
pin out float out;
variable float old;
function _;
license "GPL";
;;
float tmp = in;
out = (tmp - old) / fperiod;
old = tmp;
```

et il peut être compilé et installé très facilement en plaçant simplement `ddt.comp` dans « `src/hal/components` » puis en lançant « `make` », ou en le plaçant quelque part sur le système et en lançant « `comp --install ddt.comp` »

### 6.2 Définitions

**composant** Un composant est un simple module temps réel, qui se charge avec « `halcmd loadrt` ». Un fichier `.comp` spécifie un seul composant.

**instance** Un composant peut avoir zéro ou plusieurs instances. Chaque instance d'un composant est créée égale (elles ont toutes les mêmes pins, les mêmes paramètres, les mêmes fonctions et les mêmes données) mais elle se comporte de manière différente quand leurs pins, leurs paramètres et leur données ont des valeurs différentes.

**singleton** Il est possible pour un composant d'être un 'singleton' (composant dont il n'existe qu'une seule instance), dans ce cas, exactement une seule instance est créée. Il est rarement logique d'écrire un composant `singleton`, à moins qu'il n'y ait qu'un seul objet de ce type dans le

système (par exemple, un composant ayant pour but de fournir une pin avec le temps Unix courant, ou un pilote matériel pour le haut parleur interne du PC)

## 6.3 Création d'instance

Pour un singleton, une seule instance est créée quand le composant est chargé.

Pour un non-singleton, le paramètre 'count' du module détermine combien d'instances seront créées.

## 6.4 Syntaxe

Un fichier `.comp` commence par un certain nombre de déclarations, puis par un « ; » seul sur sa propre ligne. Il se termine par le code C implémentant les fonctions du module.

Déclarations d'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 ;
- author AUTHOR ;
```

Les parenthèses indiquent un item optionnel. Une barre verticale indique une alternative. Les mots en *CAPITALES* indiquent une variable texte, comme ci-dessous :

**HALNAME** Un identifiant.

Lorsqu'ils sont utilisés pour créer un identifiant de HAL, tous les caractères soulignés sont remplacés par des tirets, tous les points et les virgules de fin, sont supprimés, ainsi « ce\_nom\_ » est remplacé par « ce-nom » et si le nom « \_ », alors le point final est enlevé aussi, ainsi « function \_ » donne un nom de fonction HAL tel que `component.<num>` au lieu de `component.<num>.`

S'il est présent, le préfixe `hal_` est enlevé du début d'un nom de composant pour la création des pins des paramètres et des fonctions.

Dans l'identifiant de HAL pour une pin ou un paramètre, # indique un membre de tableau, il doit être utilisé conjointement avec une déclaration `[SIZE]`. Les « hash marks » sont remplacées par des nombres de 0-barrés équivalents aux nombres de caractères #.

Quand ils sont utilisés pour créer des identifiants C, les changements de caractères suivants sont appliqués au HALNAME :

1. Tous les caractères # sont enlevés ainsi que tous les caractères « . », « \_ » ou « - » immédiatement devant eux.
2. Dans un nom, tous les caractères « . » et « - » sont remplacés par « \_ »
3. Les caractères « \_ » répétitifs sont remplacés par un seul caractère « \_ ».

Un « \_ » final est maintenu, de sorte que les identifiants de HAL, qui autrement seraient en conflit avec les noms ou mots clé réservés (par exemple : « min »), puissent être utilisés.

HALNAME	Identifiant C	Identifiant HAL
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** Une expression impliquant la *personnalité* d'une variable non nulle quand la variable ou le paramètre doit être créé.

**SIZE** Un nombre donnant la taille d'un tableau. Les items des tableaux sont numérotés de 0 à *SIZE*-1.

**MAXSIZE : CONDSIZE** Un nombre donnant la taille maximum d'un tableau, suivi d'une expression impliquant la *personnalité* d'une variable et qui aura toujours une valeur inférieure à *MAXSIZE*. Quand le tableau est créé sa taille est égale à *CONDSIZE*.

**DOC** Une chaîne qui documente l'item. La chaîne doit être au format C, « entre guillemets », comme "Sélectionnez le front désiré : TRUE pour descendant, FALSE pour montant" ou au format Python « triples guillemets », pouvant inclure des caractères newlines et des guillemets, comme :

```
param rw bit zot=TRUE
"""L'effet de ce paramètre, également connu comme "the orb of zot",
requiert au moins deux paragraphes d'explications.
```

```
J'espère que ces paragraphes vous ont permis de comprendre "zot"
plus profondément.""";
```

La chaîne documentation est en format « groff -man ». Pour plus d'informations sur ce format de markup, voyez `groff_man(7)`. Souvenez vous que `comp` interprète backslash comme Echap dans les chaînes, ainsi par exemple pour passer le mot *example* en font italique, écrivez « `\\fiexample\\fB` ».

**TYPE** Un des types de HAL : `bit`, `signed` (signé), `unsigned` (non signé) ou `float` (flottant). Les anciens noms `s32` et `u32` peuvent encore être utilisés, mais `signed` et `unsigned` sont préférables.

**PINDIRECTION** Une des ces directions : `in`, `out`, ou `io`. Le composant pourra positionner la valeur d'une pin de sortie, il pourra lire la valeur sur une pin d'entrée et il pourra lire ou positionner la valeur d'une pin `io`.

**PARAMDIRECTION** Une des valeurs suivantes : `r` ou `rw`. Le composant pourra positionner la valeur d'un paramètre `r` et il pourra positionner ou lire la valeur d'un paramètre `rw`.

**STARTVALUE** Spécifie la valeur initiale d'une pin ou d'un paramètre. Si il n'est pas spécifié, alors la valeur par défaut est 0 ou `FALSE`, selon le type de l'item.

**fp** Indique que la fonction effectuera ses calculs en virgule flottante.

**nofp** Indique que la fonction effectuera ses calculs sur des entiers. Si il n'est pas spécifié, `fp` est utilisé. Ni `comp` ni `gcc` ne peuvent détecter l'utilisation de calculs en virgule flottante dans les fonctions marquées `nofp`.

**OPT, VALUE** Selon le nom de l'option `OPT`, les valeurs `VALUE` varient. Les options actuellement définies sont les suivantes :

**option singleton yes** (défaut : no)

Ne crée pas le paramètre numéro de module et crée toujours une seule instance. Avec `singleton`, les items sont nommés `composant-name.item-name` et sans `singleton`, les items des différentes instances sont nommés `composant-name.<num>.item-name`.

**option default\_count number** (défaut : 1)

Normalement, le paramètre `count` par défaut est 0. Si spécifié, `count` remplace la valeur par défaut.

**option count\_function yes** (défaut : no)

Normalement, le numéro des instances à créer est spécifié dans le paramètre `count` du module, si `count_function` est spécifié, la valeur retournée par la fonction `int get_count(void)` est utilisée à la place de la valeur par défaut et le paramètre `count` du module n'est pas défini.

**option rtapi\_app no** (défaut : yes)

Normalement, les fonctions `rtapi_app_main` et `rtapi_app_exit` sont définies automatiquement. Avec `option rtapi_app no`, elles ne le seront pas et doivent être fournies dans le code C.

Quand vous implémentez votre propre `rtapi_app_main`, appelez la fonction `int export(char *prefix, long extra_arg)` pour enregistrer les pins, paramètres et fonctions pour préfixer.

**option data type** (défaut : 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** (défaut : no)

Si spécifié, appelle la fonction définie par `EXTRA_SETUP` pour chaque instance. Si la définition automatique `rtapi_app_main` est utilisée, `extra_arg` est le numéro de cette instance.

**option extra\_cleanup yes** (défaut : no)

Si spécifié, appelle la fonction définie par `EXTRA_CLEANUP` depuis la fonction définie automatiquement `rtapi_app_exit`, ou une erreur est détectée dans la fonction automatiquement définie `rtapi_app_main`.

**option userspace yes** (défaut : no)

Si spécifié, ce fichier décrit un composant d'espace utilisateur, plutôt que le réel. Un composant d'espace utilisateur peut ne pas avoir de fonction définie par la directive de fonction. Au lieu de cela, après que toutes les instances soient construites, la fonction C `user_mainloop()` est appelée. Dès la fin de cette fonction, le composant se termine. En règle générale, `user_mainloop()` va utiliser `FOR_ALL_INSTS()` pour effectuer la mise à jour pour chaque action, puis attendre un court instant. Une autre action commune dans `user_mainloop()` peut être d'appeler le gestionnaire de boucles d'événements d'une interface graphique.

**option userinit yes** (défaut : no)

Si spécifiée, la fonction `userinit(argc, argv)` est appelée avant `rtapi_app_main()` (et cela avant l'appel de `hal_init()`). Cette fonction peut traiter les arguments de la ligne de commande ou exécuter d'autres actions. Son type de retour est `void`; elle peut appeler `exit()` et si elle le veut, se terminer sans créer de composant HAL (par exemple, parce que les arguments de la ligne de commande sont invalides).

Si aucune option `VALUE` n'est spécifiée, alors c'est équivalent à spécifier la valeur... `yes`. Le résultat consécutif à l'assignation d'une valeur inappropriée à une option est indéterminé. Le résultat consécutif à n'utiliser aucune autre option est indéfini.

**LICENSE** Spécifie la license du module, pour la documentation et pour le module déclaré dans `MODULE_LICENSE()`.

**AUTHOR** Spécifie l'auteur du module, pour la documentation

## 6.5 Stockage des données « par-instance »

**variable CTYPE NAME;**

**variable CTYPE NAME[SIZE];**

**variable CTYPE NAME = default;**

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

Déclare la variable « par-instance » *NAME* de type *CTYPE*, optionnellement comme un tableau de *SIZE* items et optionnellement avec une valeur *default*. Les items sans *default* sont initialisés « all-bits-zero ». *CTYPE* est un simple mot de type C, comme *float*, *u32*, *s32*, etc.

Les variables d'un tableau sont mises entre crochets.

Les commentaires de style C++ une ligne (*// ...*) et de style C multi-lignes (*/\* ... \*/*) sont supportés tous les deux dans la section déclaration.

## 6.6 Autres restrictions sur les fichiers comp

Bien que HAL permette à une pin, un paramètre et une fonction d'avoir le même nom, comp ne le permet pas.

## 6.7 Conventions des macros

En se basant sur les déclarations des items de section, *comp* crée une structure C appelée *structure d'état*. Cependant, au lieu de faire référence aux membres de cette structure (par exemple : *\*(inst->name)*), il leur sera généralement fait référence en utilisant les macros ci-dessous. Certains détails de la structure d'état et de ces macros peuvent différer d'une version de *comp* à la suivante.

**FUNCTION(name)** Cette macro s'utilise au début de la définition d'une fonction temps réel qui aura été précédemment déclarée avec *'function NAME'*. *function* inclus un paramètre *'period'* qui est le nombre entier de nanosecondes entre les appels à la fonction.

**EXTRA\_SETUP()** Cette macro s'utilise au début de la définition de la fonction appelée pour exécuter les réglages complémentaires à cette instance. Une valeur de retour négative (Unix *errno*) indique un défaut (par exemple : elle retourne *-EBUSY* comme défaut à la réservation d'un port d'entrées/sorties), une valeur égale à 0 indique le succès.

**EXTRA\_CLEANUP()** Cette macro s'utilise au début de la définition de la fonction appelée pour exécuter un nettoyage (*cleanup*) du composant. Noter que cette fonction doit nettoyer toutes les instances du composant, pas juste un. Les macros *'pin\_name'*, *'parameter\_name'* et *'data'* ne doivent pas être utilisées ici.

### *pin\_name*

**parameter\_name** Pour chaque pin, *pin\_name* ou pour chaque paramètre, *parameter\_name* il y a une macro qui permet d'utiliser le nom seul pour faire référence à la pin ou au paramètre.

Quand *pin\_name* ou *parameter\_name* sont des tableaux, la macro est de la forme *pin\_name(idx)* ou *param\_name(idx)* dans laquelle *idx* est l'index dans le tableau de pins. Quand le tableau est de taille variable, il est seulement légal de faire référence aux items par leurs *condsize*.

Quand un item est conditionnel, il est seulement légal de faire référence à cet item quand ses *conditions* sont évaluées à des valeurs différentes de zéro.

**variable\_name** Pour chaque variable, il y a une macro *variable\_name* qui permet au nom seul d'être utilisé pour faire référence à la variable. Quand *variable\_name* est un tableau, le style normal de C est utilisé : *variable\_name[idx]*

**data** Si l'« option data » est spécifiée, cette macro permet l'accès à l'instance de la donnée.

**fperiod** Le nombre de secondes en virgule flottante entre les appels à cette fonction temps réel.

**FOR\_ALL\_INSTS0 { . . . }** Pour les composants de l'espace utilisateur. Cette macro utilise la variable *struct state \*inst* pour itérer au dessus de toutes les instances définies. Dans le corps de la boucle, les macros ***pin\_name***, ***parameter\_name*** et ***data*** travaillent comme elles le font dans les fonctions temps réel.

## 6.8 Composants avec une seule fonction

Si un composant a seulement une fonction et que la chaîne « FUNCTION » n'apparaît nulle part après ; ; , alors la portion après ; ; est considérée comme étant le corps d'un composant simple fonction.

## 6.9 « Personnalité » du composant

Si un composant a n'importe combien de pins ou de paramètres avec un « if *condition* » ou « [*max-size : condsz*] », il est appelé un composant avec « *personnalité* ». La « personnalité » de chaque instance est spécifiée quand le module est chargé. La « Personnalité » peut être utilisée pour créer les pins seulement quand c'est nécessaire. Par exemple, la personnalité peut être utilisée dans un composant logique, pour donner un nombre variable de broches d'entrée à chaque porte logique et permettre la sélection de n'importe quelle fonction de logique booléenne de base **and**, **or** et **xor**.

## 6.10 Compiler un fichier .comp dans l'arborescence

Placer le fichier .comp dans le répertoire `emc2/src/hal/components` et lancer/relancer « make ». Les fichiers Comp sont automatiquement détectés par le système de compilation.

Si un fichier .comp est un pilote de périphérique, il peut être placé dans `emc2/src/hal/components` et il y sera construit excepté si `emc2` est configuré en mode simulation.

## 6.11 Compiler un composant temps réel hors de l'arborescence

`comp` peut traiter, compiler et installer un composant temps réel en une seule étape, en plaçant `rtexample.ko` dans le répertoire du module temps réel d'`emc2` :

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

Ou il peut aussi être traité et compilé en une seule étape en laissant `example.ko` (ou `example.so` pour la simulation) dans le répertoire courant :

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

Ou il peut simplement être traité en laissant `example.c` dans le répertoire courant :

```
comp rtexample.comp
```

`comp` peut aussi compiler et installer un composant écrit en C, en utilisant les options `--install` et `--compile` comme ci-dessous :

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

La documentation au format `man` peut être créée à partir des informations de la section « *declaration* » :

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

La manpage résultante, `exemple.9` peut être lue avec :

```
man ./exemple.9
```

ou copiée à un emplacement standard pour une page de manuel.

## 6.12 Compiler un composant de l'espace utilisateur hors de l'arborescence

comp peut traiter, compiler et installer un document de l'espace utilisateur :

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

Cela fonctionne seulement pour les fichiers .comp mais pas pour les fichiers .c.

## 6.13 Exemples

### 6.13.1 constant

Ce composant fonctionne comme dans 'blocks', y compris la valeur par défaut à 1.0. La déclaration « function \_ » crée les fonctions nommées 'constant.0', etc.

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

### 6.13.2 sincos

Ce composant calcule le sinus et le cosinus d'un angle entré en radians. Il a différentes possibilités comme les sorties 'sinus' et 'cosinus' de siggen, parceque l'entrée est un angle au lieu d'être librement basé sur un paramètre 'frequency'.

Les pins sont déclarées avec les noms sin\_ et cos\_ dans le code source pour que ça n'interfère pas avec les fonctions sin() et cos(). Les pins de HAL sont toujours appelées 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); }
```

### 6.13.3 out8

Ce composant est un pilote pour une carte imaginaire appelée « out8 », qui a 8 pins de sortie digitales qui sont traitées comme une simple valeur sur 8 bits. Il peut y avoir un nombre quelconque de ces cartes dans le système et elles peuvent avoir des adresses variées. La pin est appelée out\_ parceque out est un identifiant utilisé dans <asm/io.h>. Il illustre l'utilisation de EXTRA\_SETUP et de EXTRA\_CLEANUP pour sa requête de région d'entrées/sorties et libère cette région en cas d'erreur ou quand le module est déchargé.

```
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); }

```

### 6.13.4 hal\_loop

```

component hal_loop;
pin out float example;

```

Ce fragment de composant illustre l'utilisation du préfixe `hal_` dans un nom de composant. `loop` est le nom d'un module standard du kernel Linux, donc un composant `loop` ne pourrait pas être chargé si le module `loop` de Linux est également présent.

Quand il le charge, `halcmd` montre un composant appelé `hal_loop`. Cependant, les pins affichées par `halcmd` sont `loop.0.example` et non `hal-loop.0.example`.

### 6.13.5 arraydemo

Ce composant temps réel illustre l'utilisation d'un tableau de taille fixe :

```

component arraydemo "Registre à décalage 4-bits";
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;

```

### 6.13.6 rand

Ce composant de l'espace utilisateur modifie la valeur de ses pins de sortie vers une nouvelle valeur aléatoire dans l'étendue  $[0, 1)$  à chaque ms.

```

component rand;
option userspace;

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

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

```

#### 6.13.6.1 logic

Ce composant temps réel montre l'utilisation de la « personnalité » pour créer un tableau de taille variable et des pins optionnelles.

```

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;
}

```

Une ligne de chargement typique pourrait être :

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

qui créerait les pins suivantes :

- Une porte AND à 2 entrées : logic.0.and, logic.0.in-00, logic.0.in-01
- des portes AND et OR à 5 entrées : 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,
- des portes AND et XOR à 3 entrées : logic.2.and, logic.2.xor, logic.2.in-00, logic.2.in-01, logic.2.in-02

## Chapitre 7

# Création de composants de l'espace utilisateur Python avec les modules de 'hal'

### 7.1 Utilisation de base

Un composant de l'espace utilisateur commence par créer ses pins et ses paramètres, puis il entre dans une boucle de laquelle il va positionner périodiquement toutes ses sorties en fonction de ses entrées. Le composant suivant, un passe-tout, copie la valeur vue sur ses pins d'entrée (`passthrough.in`) vers ses pins de sortie (`passthrough.out`) approximativement une fois par seconde.

```
#!/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
```

Copier le listing précédent dans un fichier nommé « `passthrough` », le rendre exécutable par un « `chmod +x` » et le placer dans son `$PATH`. On peut alors l'essayer en faisant :

```
$ 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
```

## 7.2 Composants de l'espace utilisateur et délais

Si vous tapez rapidement « show pin », vous pourrez voir que `passthrough.out` conserve un moment son ancienne valeur de 0. Ceci est dû à l'appel de la fonction `'time.sleep(1)'`, qui fait que les pins de sortie changent d'état, au plus, une fois par seconde. Parceque ce composant appartient à l'espace utilisateur, ce délai peut apparaître plus long, par exemple si la mémoire utilisée par le composant `passthrough` est échangée avec le disque dur, le délai peut être allongé jusqu'au rafraîchissement de la mémoire.

Ces composants de l'espace utilisateur conviennent parfaitement pour des éléments tels que des panneaux de contrôle pour lesquels des délais de l'ordre de quelques millisecondes sont imperceptibles. Ils ne conviennent pas, en revanche, pour envoyer des impulsions de pas vers une carte de pilotage de périphériques pour lesquelles les délais doivent rester de l'ordre de quelques microsecondes, dans tous les cas).

## 7.3 Créer les pins et les paramètres

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

Le composant lui-même est créé par l'appel du constructeur `'hal.component'`. Les arguments sont le nom du composant HAL et optionnellement, le préfixe utilisé pour les noms de pin et de paramètre. Si le préfixe n'est pas spécifié, le nom du composant est utilisé.

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

Puis les pins sont créées par appels des méthodes sur l'objet composant. Les arguments sont : pin nom suffixe, type de pin et direction de la pin. Pour les paramètres, les arguments sont : paramètre nom suffixe, type de paramètre et direction du paramètre.

TAB. 7.1 – HAL Option Names

Types de Pin et Paramètre :	HAL_BIT	HAL_FLOAT	HAL_S32	HAL_U32
Directions des pins :	HAL_IN	HAL_OUT	HAL_IO	
Directions des paramètres :	HAL_RO	HAL_RW		

Le nom complet d'une pin ou d'un paramètre est formé en joignant le préfixe avec le suffixe par un « . », comme dans l'exemple où la pin créée est appelée `passthrough.in`.

```
h.ready()
```

Une fois toutes les pins et les paramètres créés, la méthode `.ready()` est appelée.

### 7.3.1 Changer le préfixe

Le préfixe peut être changé en appelant la méthode `.setprefix()`. Le préfixe courant peut être retrouvé en appelant la méthode `.getprefix()`.

## 7.4 Lire et écrire les pins et les paramètres

Pour les pins et les paramètres qui sont aussi des identifiants Python, la valeur est accessible ou ajustable en utilisant la syntaxe des attributs suivante :

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

Pour les pins et les paramètres qui sont aussi des identifiants Python, la valeur est accessible ou ajustable en utilisant la syntaxe de sous-script suivante :

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

### 7.4.1 Pilotage des pins de sortie (HAL\_OUT)

Périodiquement, habituellement dans le temps de réponse de l'horloge, toutes les pins HAL\_OUT doivent être « pilotées » en leur assignant une nouvelle valeur. Ceci doit être fait que la valeur soit différente ou non de la valeur précédemment assignée. Quand la pin est connectée au signal, l'ancienne valeur de sortie n'est pas copiée vers le signal, la valeur correcte n'apparaîtra donc sur le signal qu'une fois que le composant lui aura assigné une nouvelle valeur.

### 7.4.2 Pilotage des pins bidirectionnelles (HAL\_IO)

La règle mentionnée ci-dessus ne s'applique pas aux pins bidirectionnelles. Au lieu de cela, une pin bidirectionnelle doit seulement être pilotée par le composant et quand le composant souhaite changer sa valeur. Par exemple, dans l'interface codeur, le composant codeur positionne seulement la pin **index-enable** à **FALSE** quand une impulsion d'index est vue et que l'ancienne valeur est **TRUE**, mais ne la positionne jamais à **TRUE**. Piloter répétitivement la pin à **FALSE** pourrait faire qu'un autre composant connecté agisse comme si une nouvelle impulsion d'index avait été vue.

## 7.5 Quitter

Une requête « `halcmd unload` » pour le composant est délivrée comme une exception `KeyboardInterrupt`. Quand une requête de déchargement arrive, le processus doit quitter dans un court laps de temps ou appeler la méthode `.exit()` sur le composant si un travail substantiel, comme la lecture ou l'écriture de fichiers, doit être fourni pour terminer le processus d'arrêt.

## 7.6 Idées de projets

- Créer un panneau de contrôle extérieur avec boutons poussoirs, interrupteurs et voyants. Connecter le tout à un microcontrôleur et raccorder le microcontrôleur à un PC en utilisant une liaison série. Python est vraiment capable d'interfacer une liaison série grâce à son module `pyserial` <http://pyserial.sourceforge.net/> (Paquet « `python-serial` », dans les dépôts universels d'Ubuntu)
- Relier un module d'affichage à LCD LCDProc <http://lcdproc.omnipotent.net/> et l'utiliser pour afficher les informations de votre choix (Paquet « `lcdproc` », dans les dépôts universels d'Ubuntu)
- Créer un panneau de contrôle virtuel utilisant n'importe quelle librairie d'interface graphique supportée par Python (`gtk`, `qt`, `wxwindows`, etc)

## **Troisième partie**

### **Annexes**

# Appendix A

## Legal Section

### A.1 Copyright Terms

Copyright (c) 2000 LinuxCNC.org

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 no Invariant Sections, no Front-Cover Texts, and one Back-Cover Text: "This EMC Handbook is the product of several authors writing for linuxCNC.org. As you find it to be of value in your work, we invite you to contribute to its revision and growth." A copy of the license is included in the section entitled "GNU Free Documentation License". If you do not find the license you may order a copy from Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307

### A.2 GNU Free Documentation License

GNU Free Documentation License Version 1.1, March 2000

Copyright (C) 2000 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA  
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

#### 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, 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

blocks, [9](#)

ClassicLadder, [9](#)  
CNC, [6](#)

encoder, [9](#)

HAL, [6](#)  
HAL Broche, [8](#)  
HAL Composant, [8](#)  
HAL Fil, [9](#)  
HAL Fonction, [8](#)  
HAL Paramètre, [8](#)  
HAL Signal, [8](#)  
HAL Type, [8](#)  
hal-ax5214h, [10](#)  
hal-m5i20, [10](#)  
hal-motenc, [10](#)  
hal-parport, [10](#)  
hal-ppmc, [10](#)  
hal-stg, [10](#)  
hal-vti, [10](#)  
HAL: Broche physique, [8](#)  
halcmd, [10](#)  
halmeter, [10](#)  
halscope, [10](#)  
halui, [9](#)

iocontrol, [9](#)

motion, [9](#)

pid, [9](#)

siggen, [9](#)  
stepgen, [9](#)  
supply, [9](#)