

## **Manuel de HAL V2.7.4; 2016-02-08**

# Table des matières

<b>I</b>	<b>HAL</b>	<b>1</b>
<b>1</b>	<b>Introduction à HAL</b>	<b>2</b>
1.1	HAL est basé sur le système d'étude des projets techniques . . . . .	2
1.1.1	Choix des organes . . . . .	2
1.1.2	Étude des interconnexions . . . . .	2
1.1.3	Implémentation . . . . .	3
1.1.4	Mise au point . . . . .	3
1.1.5	En résumé . . . . .	3
1.2	Concept de HAL . . . . .	4
1.3	Composants HAL . . . . .	5
1.4	Programmes externes attachés à HAL . . . . .	5
1.5	Composants internes . . . . .	5
1.6	Pilotes de matériels . . . . .	5
1.7	Outils-Utilitaires . . . . .	6
1.8	Les réflexions qui ont abouti à la création de HAL . . . . .	6
1.8.1	Une tour . . . . .	6
1.8.2	Erector Sets . . . . .	6
1.8.3	Tinkertoys . . . . .	7
1.9	Un exemple en Lego . . . . .	7
1.10	Problèmes de timing dans HAL . . . . .	8
<b>2</b>	<b>Conventions générales</b>	<b>9</b>
2.1	Les noms . . . . .	9
2.2	Conventions générales de nommage . . . . .	9
2.3	Conventions de nommage des pilotes de matériels . . . . .	10
2.3.1	Noms de pin/paramètre . . . . .	10
2.3.1.1	Exemples . . . . .	10
2.3.2	Noms des fonctions . . . . .	11
2.3.2.1	Exemples . . . . .	11

2.4	Périphériques d'interfaces canoniques	11
2.5	Entrée numérique (Digital Input)	11
2.5.1	Pins	12
2.5.2	Paramètres	12
2.5.3	Fonctions	12
2.6	Sortie numérique (Digital Output)	12
2.6.1	Pins	12
2.6.2	Paramètres	12
2.6.3	Fonctions	12
2.7	Entrée analogique (Analog Input)	12
2.7.1	Pins	12
2.7.2	Paramètres	13
2.7.3	Fonctions	13
2.8	Sortie analogique (Analog Output)	13
2.8.1	Pins	13
2.8.2	Paramètres	13
2.8.3	Fonctions	14
<b>3</b>	<b>Commandes et composants de base</b>	<b>15</b>
3.1	Commandes de Hal	15
3.1.1	loadrt	16
3.1.2	addf	16
3.1.3	loadusr	17
3.1.4	net	17
3.1.5	setp	18
3.1.6	sets	19
3.1.7	unlinkp	19
3.1.8	Commandes obsolètes	19
3.1.9	linksp	19
3.1.10	linkps	20
3.1.11	newsig	20
3.2	HAL Data	20
3.2.1	Bit	20
3.2.2	Float	20
3.2.3	s32	20
3.2.4	u32	20
3.3	Fichiers Hal	21
3.4	Composants de HAL	21
3.5	Composants de logiques combinatoire	21

3.5.1	and2	21
3.5.2	not	22
3.5.3	or2	22
3.5.4	xor2	22
3.5.5	Exemples en logique combinatoire	23
3.6	Composants de conversion	23
3.6.1	Somme pondérée (weighted_sum)	23
<b>4</b>	<b>Le tutoriel de HAL</b>	<b>25</b>
4.1	Introduction	25
4.2	Halcmd	25
4.2.1	Tab-complétion	25
4.2.2	L'environnement RTAPI	25
4.3	Tutoriel simple	26
4.3.1	Charger un composant temps réel	26
4.3.2	Examiner HAL	26
4.3.3	Exécuter le code temps réel	27
4.3.4	Modifier des paramètres	29
4.3.5	Enregistrer la configuration de HAL	29
4.3.6	Quitter halrun	30
4.3.7	Restaurer la configuration de HAL	30
4.3.8	Suppression de la mémoire de HAL	30
4.4	Visualiser HAL avec halmeter	30
4.4.1	Lancement de halmeter	31
4.5	Tutoriel plus complexe avec stepgen	33
4.5.1	Installation des composants	33
4.5.2	Connexion des pins avec les signaux	34
4.5.3	Exécuter les réglages du temps réel - threads et fonctions	35
4.5.4	Réglage des paramètres	36
4.5.5	Lançons le!	37
4.6	Voyons-y de plus près avec halscope	37
4.6.1	Démarrer halscope	37
4.6.2	Branchement des sondes du scope	39
4.6.3	Capturer notre première forme d'onde	42
4.6.4	Ajustement vertical	43
4.6.5	Déclenchement (Triggering)	44
4.6.6	Ajustement horizontal	46
4.6.7	Plus de canaux	47
4.6.8	Plus d'échantillons	48

<b>5</b>	<b>Les fonctionnalités de Halshow</b>	<b>49</b>
5.1	Le script Halshow	49
5.1.1	Zone de l'arborescence de Hal	49
5.1.2	Zone de l'onglet MONTRER	51
5.1.3	Zone de l'onglet WATCH	54
<b>6</b>	<b>Les composants de HAL</b>	<b>56</b>
6.1	Composants de commandes et composants de l'espace utilisateur	56
6.2	Composants temps réel et modules du noyau	56
6.2.1	Composants du coeur de LinuxCNC	57
6.2.2	Composants binaires et logiques	57
6.2.3	Composants arithmétiques et flottants	57
6.2.4	Conversions de type	58
6.2.5	Pilotes de matériel	58
6.2.6	Composants cinématiques	58
6.2.7	Composants de contrôle moteur	59
6.2.8	BLDC and 3-phase motor control	59
6.2.9	Autres composants	59
6.3	Appels à l'API de HAL (liste de la section 3 des man pages)	60
6.4	Appels à RTAPI	61
<b>7</b>	<b>Les composants temps réel</b>	<b>63</b>
7.1	Stepgen	63
7.1.1	L'installer	64
7.1.2	Le désinstaller	65
7.1.3	Pins	65
7.1.4	Paramètres	65
7.1.5	Séquences de pas	66
7.1.6	Fonctions	70
7.2	PWMgen	70
7.2.1	L'installer	70
7.2.2	Le désinstaller	70
7.2.3	Pins	70
7.2.4	Paramètres	71
7.2.5	Types de sortie	71
7.2.6	Fonctions	71
7.3	Codeur	71
7.3.1	L'installer	72
7.3.2	Le désinstaller	72

7.3.3	Pins	72
7.3.4	Paramètres	73
7.3.5	Fonctions	73
7.4	PID	74
7.4.1	L'installer	74
7.4.2	Le désinstaller	75
7.4.3	Pins	75
7.4.4	Paramètres	75
7.4.5	Fonctions	76
7.5	Codeur simulé	76
7.5.1	L'installer	76
7.5.2	Le désinstaller	76
7.5.3	Pins	76
7.5.4	Paramètres	77
7.5.5	Fonctions	77
7.6	Anti-rebond	77
7.6.1	L'installer	77
7.6.2	Le désinstaller	77
7.6.3	Pins	77
7.6.4	Paramètres	78
7.6.5	Fonctions	78
7.7	Siggen	78
7.7.1	L'installer	78
7.7.2	Le désinstaller	78
7.7.3	Pins	78
7.7.4	Paramètres	79
7.7.5	Fonctions	79
7.8	lut5	79
<b>8</b>	<b>Exemples pour HAL</b>	<b>81</b>
8.1	Changement d'outil manuel	81
8.2	Calcul de vitesse	81
8.3	Amortissement d'un signal	83
8.4	HAL en autonome	85
<b>9</b>	<b>L'interface Halui</b>	<b>87</b>
9.1	Introduction	87
9.2	Nomenclature des pins d'Halui	87
9.3	Exemples de programme avec Halui	91
9.3.1	Démarrage à distance	91
9.3.2	Pause et Reprise	92

---

<b>10 comp: outil pour créer les modules HAL</b>	<b>94</b>
10.1 Introduction	94
10.2 Installation	94
10.3 Définitions	94
10.4 Création d'instance	95
10.5 Paramètres implicites	95
10.6 Syntaxe	95
10.6.1 Fonctions HAL	96
10.6.2 Options	97
10.6.3 Licence et auteur	97
10.6.4 Stockage des données <b>par instance</b>	98
10.6.5 Commentaires	98
10.7 Restrictions sur les fichiers comp	98
10.8 Conventions des macros	98
10.9 Composants avec une seule fonction	99
10.10 Personnalité du composant	99
10.11 Compiler un fichier <i>.comp</i> dans l'arborescence	99
10.12 Compiler un composant temps réel hors de l'arborescence	99
10.13 Compiler un composant de l'espace utilisateur hors de l'arborescence	100
10.14 Exemples	100
10.14.1 constant	100
10.14.2 sincos	100
10.14.3 out8	101
10.14.4 hal_loop	102
10.14.5 arraydemo	102
10.14.6 rand	102
10.14.7 logic	102
<b>11 Créer des composants de l'espace utilisateur</b>	<b>104</b>
11.1 Utilisation de base en Python	104
11.2 Composants de l'espace utilisateur et délais	105
11.3 Créer les pins et les paramètres	105
11.3.1 Changer le préfixe	105
11.4 Lire et écrire les pins et les paramètres	105
11.4.1 Pilotage des pins de sortie (HAL_OUT)	106
11.4.2 Pilotage des pins bidirectionnelles (HAL_IO)	106
11.5 Quitter	106
11.6 Idées de projets	106
<b>12 Index</b>	<b>107</b>



### The LinuxCNC Team

Ce manuel est en évolution permanente. Si vous voulez nous aider à son écriture, sa rédaction, sa traduction ou la préparation des graphiques, merci de contactez n'importe quel membre de l'équipe de traduction ou envoyez un courrier électronique à [emc-users@lists.sourceforge.net](mailto:emc-users@lists.sourceforge.net).

Copyright © 2000–2012 LinuxCNC.org

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. 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

LINUX® is the registered trademark of Linus Torvalds in the U.S. and other countries. The registered trademark Linux® is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

Permission est donnée de copier, distribuer et/ou modifier ce document selon les termes de la « GNU Free Documentation License », Version 1.3 ou toute version ultérieure publiée par la « Free Software Foundation »; sans sections inaltérables, sans texte de couverture ni quatrième de couverture. Une copie de la licence est incluse dans la section intitulée « GNU Free Documentation License ». Si vous ne trouvez pas la licence vous pouvez en commander un exemplaire chez Free Software Foundation, Inc. 59 Temple Place, Suite 330 Boston, MA 02111-1307

(La version de langue anglaise fait foi)

---

### AVIS

La version Française de la documentation de LinuxCNC est toujours en retard sur l'originale faute de disponibilité des traducteurs.

Il est recommandé d'utiliser la documentation en Anglais chaque fois que possible.

Si vous souhaitez être un traducteur bénévole pour la documentation française de LinuxCNC, merci de nous contactez.

---

### NOTICE

The French version of the LinuxCNC documentation is always behind the original fault availability of translators.

It's recommended to use the English documentation whenever possible.

If you would like to be a volunteer editor for the French translation of LinuxCNC, please contact us.

---



## **Première partie**

# **HAL**

# Chapitre 1

## Introduction à 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 de LinuxCNC 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 HAL est basé sur le système 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 contacts de fin de course, les panneaux de boutons de commande, les manivelles électroniques, 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 Choix des organes

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

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

#### 1.1.2 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

---

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

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.

### 1.1.3 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 interconnexion 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.

### 1.1.4 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 re-câblées 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 mêmes commandes utilisées pour construire le système, seront utilisées pour faire les changements indispensables.

### 1.1.5 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 LinuxCNC, par exemple pour une [commande distante](#) telle que décrite dans la documentation de Halui.

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 connexion 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 connexion 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. Vous vous demandez peut être quelle relation il y a entre les HAL\_pins, les Physical\_pins et les éléments extérieurs comme les codeurs ou une carte stg. Nous avons ici, affaire à des interfaces de type translation/conversion de données.

### 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 +4,294,967,295
- s32 - un entier signé de 32 bits, les valeurs légales vont de -2,147,483,648 à +2,147,483,647

### 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 *tread* 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 œuvre. 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.

## 1.4 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.

## 1.5 Composants internes

### **stepgen**

Générateur d'impulsions de pas avec boucle de position. Plus de détails [sur stepgen](#).

### **encoder**

Codeur/compteur logiciel. Plus de détails [sur le codeur](#).

### **pid**

Boucle de contrôle Proportionnelle/Intégrale/Dérivée. Plus de détails [sur le PID](#).

### **siggen**

Générateur d'ondes: sinusoïdale/cosinoïdale/triangle/carrée, pour la mise au point. Plus de détails [sur siggen](#).

### **supply**

Une simple alimentation, pour la mise au point

### **blocks**

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

## 1.6 Pilotes de matériels

---

**hal\_ax5214h**

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

**hal\_m5i20**

Un pilote pour la carte Mesa Electronics 5i20

**hal\_motenc**

Un pilote pour la carte Vital Systems MOTENC-100

**hal\_parport**

Pilote pour le(ou les) port(s) parallèle(s). Plus de détails sur les [ports parallèles](#).

**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

## 1.7 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. Plus de détails pour utiliser [halmeter](#).

**halscope**

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

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

## 1.8 Les réflexions qui ont abouti à la création de 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.

### 1.8.1 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.

### 1.8.2 Erector Sets<sup>2</sup>

---

2. Le jeu Erector Set est une invention de AC Gilbert (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 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.

### 1.8.3 Tinkertoys<sup>3</sup>

Le jouet en bois Tinkertoys est plus humain que l'acier froid de l'Erector. Le cœur 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 ascenseur 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 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.

Tous les détails dans le [tutoriel de HAL](#).

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.

## 1.9 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 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.

3. Tinkertoy est maintenant registered trademark of the Hasbro company.

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

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.

## 1.10 Problèmes de timing 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 Classic Ladder 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 LinuxCNC on peut avoir un thread à 50μs et un thread à 1ms. En se basant sur les valeurs de BASE\_PERIOD et de SERVO\_PERIOD. Valeurs fixées dans le fichier ini.

La prochaine étape consiste à décider de ce que chaque thread doit faire. Certaines de ces décisions sont les mêmes dans (presque) tous les systèmes LinuxCNC. 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. :lang: fr :toc:



## Chapitre 2

# Conventions générales

### 2.1 Les 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 mis en forme *<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 entre parenthèses (*float*) et suivi d'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.

### 2.2 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.

---

1. 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*.

## 2.3 Conventions de nommage des pilotes de matériels

2

### 2.3.1 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 codeurs, des générateurs d'impulsions de pas ou de PWM, des convertisseurs numérique/analogique, analogique/numérique et d'autres possibilités plus spécifiques. Le *I/O type* est utilisé pour identifier le type d'I/O avec lequel la pin ou le paramètre est associé. Idéalement, les pilotes qui implémentent les mêmes type d'I/O, même sur des dispositifs très différents, devraient fournir un jeu de pins et de paramètres cohérents et de comportements identiques. Par exemple, toutes les entrées numériques doivent se comporter de la même manière quand elles sont vues de l'intérieur de HAL, indépendamment du périphérique.

#### <chan-num>

Quasiment tous les périphériques d'I/O ont plusieurs canaux, le numéro de canal *chan-num* identifie un de ceux ci. Comme les numéros de périphériques *device-num*, les numéros de canaux, *chan-num*, commencent à zéro et s'incrémentent.<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 inférieur à 10 pour préserver l'ordre des tris. Certains modules ont des pins et/ou des paramètres qui affectent plusieurs canaux. Par exemple un générateur de PWM peut avoir quatre canaux avec quatre entrées *duty-cycle* indépendantes, mais un seul paramètre *frequency* qui contrôle les 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 [dans ce chapitre qui leur est dédié](#).

#### 2.3.1.1 Exemples

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

3. Certains matériels utilisent des cavaliers ou d'autres dispositifs pour définir une identification spécifique à chacun. Idéalement, le pilote fournit une manière à l'utilisateur de dire, le *device-num 0 est spécifique au périphérique qui a l'ID XXX*, ses sous-ensembles 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.

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

**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 sur la première carte Pico Systems ppmc.

**2.3.2 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.

**2.3.2.1 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 Pico Systems ppmc.

**2.4 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.

**2.5 Entrée numérique (Digital Input)**

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

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

### 2.5.1 Pins

**(bit) *in***

État de l'entrée matérielle.

**(bit) *in-not***

État inversé de l'entrée matérielle.

### 2.5.2 Paramètres

Aucun

### 2.5.3 Fonctions

**(funct) *read***

Lire le matériel et ajuster les HAL pins *in* et *in-not*.

## 2.6 Sortie numérique (Digital Output)

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

### 2.6.1 Pins

**(bit) *out***

Valeur à écrire (éventuellement inversée) sur une sortie matérielle.

### 2.6.2 Paramètres

**(bit) *invert***

Si TRUE, *out* est inversée avant écriture sur la sortie matérielle.

### 2.6.3 Fonctions

**(funct) *write***

Lit *out* et *invert* et ajuste la sortie en conséquence.

## 2.7 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.

### 2.7.1 Pins

**(float) *value***

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

---

## 2.7.2 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 aucune tension n'est appliquée sur la pin.

## 2.7.3 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.

## 2.8 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.

### 2.8.1 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*.

### 2.8.2 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.

---

### 2.8.3 Fonctions

**(funct) *write***

Ecrit la valeur calculée sur la sortie matérielle. Si *enable* est FALSE, 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 TRUE, l'échelle, l'offset et la valeur sont traités et  $(scale \_ value) + offset$  sont envoyés à la sortie du DAC. Si *enable* est FALSE, la sortie passe à 0.

## Chapitre 3

# Commandes et composants de base

### 3.1 Commandes de Hal

Des informations plus détaillées peuvent être trouvées dans la man page en tapant *man halcmd* dans une console. Pour voir la configuration de HAL ainsi que le statut de ses pins et paramètres utiliser la fenêtre HAL Configuration dans le menu *Machine* d'AXIS. Pour visualiser le statut des pins, ouvrir l'onglet *Watch* puis cliquer dans l'arborescence sur les pins qui doivent être visualisées dans la fenêtre watch.



FIGURE 3.1 – Fenêtre de configuration de HAL

### 3.1.1 loadrt

La commande *loadrt* charge un composant temps réel de HAL. Les composants temps réel doivent être ajoutés au thread temps réel pour être fonctionnels. Il n'est pas possible de charger un composant de l'espace utilisateur dans l'espace temps réel.

Syntaxe et exemple:

```
loadrt <component> <options>
```

```
loadrt mux4 count=1
```

### 3.1.2 addf

La commande *addf* ajoute une fonction à un thread temps réel. Si l'assistant StepConf a été utilisé pour créer la configuration, deux threads ont été créés.

- base-thread (le thread haute vitesse) ce thread prends en main les items nécessitant une réponse très rapide comme la génération d'impulsions, la lecture et l'écriture sur le port parallèle.
- servo-thread (le thread basse vitesse) ce thread prends en main les items n'étant pas influencés par la vitesse comme le contrôleur de mouvement, l'API Classic Ladder et les commandes manuelles.



**Syntaxe et exemple:**

```
addf <component> <thread>

addf mux4 servo-thread
```

**3.1.3 loadusr**

La commande *loadusr* charge un composant de HAL de l'espace utilisateur. Les programmes de l'espace utilisateur ont leur propre processus séparé qui optionnellement communique avec les autres composants de HAL via leurs pins et paramètres. Il n'est pas possible de charger un composant temps réel dans l'espace utilisateur.

Les drapeaux peuvent être un ou plusieurs parmi les suivants:

**-W**

pour attendre que le composant soit prêt. Le composant est supposé avoir le même nom que le premier argument de la commande.

**-Wn <nom>**

pour attendre un composant, qui porte le nom donné sous la forme <nom>.

**-w**

pour attendre la fin du programme

**-i**

pour ignorer la valeur retournée par le programme (avec -w)

**Syntaxe et exemple:**

```
loadusr <component> <options>

loadusr halui
loadusr -Wn spindle gs2_vfd -n spindle
```

En anglais ça donne *loadusr wait for name spindle component gs2\_vfd name spindle*. Le -n spindle est une partie du composant gs2\_vfd et non de la commande loadusr.

**3.1.4 net**

La commande *net* crée une *connexion* entre un signal et une ou plusieurs pins. Si le signal n'existe pas, net le crée. Les flèches de direction <=, => et <=> sont seulement là pour aider à la lecture de la logique, ils ne sont pas utilisés par la commande net. Un espace doit séparer les flèches de direction des noms de pin.

**Syntaxe et exemple:**

```
net signal-name pin-name <direction optionnelle> (<second pin-name optionnel>)

net home-x axis.0.home-sw-in <= parport.0.pin-11-in
```

Dans l'exemple ci-dessus, *home-x* est le nom du signal, *axis.0.home-sw-in* est une pin de direction IN, <= est une flèche de direction optionnelle et *parport.0.pin-11-in* est une pin de direction OUT. Cela peut paraître déroutant mais les labels in et out, pour une broche de port parallèle, indiquent la direction physique dans laquelle travaille la broche et non comment elle est traitée dans HAL.

Une pin peut être connectée à un signal si elle obéit aux règles suivantes:

- Une pin IN peut toujours être connectée à un signal.
- Une pin IO peut être connectée à moins qu'une pin OUT soit présente sur le signal.
- Une pin OUT peut être connectée seulement si il n'y a pas d'autre pin OUT ou IO sur le signal.

Le même *signal-name* peut être utilisé dans de multiples commandes net pour connecter des pins additionnelles, tant que les règles précédentes sont observées.



FIGURE 3.2 – Direction du signal

Voici un exemple qui montre le signal xStep avec la source qui est stepgen.0.out et avec deux lecteurs, parport.0.pin-02-out et parport.0.pin-08-out. Simplement la valeur de stepgen.0.out est envoyée au signal xStep et cette valeur est alors envoyée sur parport.0.pin-02-out.

```
# signal      source      destination
net xStep stepgen.0.out => parport.0.pin-02-out
```

Puisque le signal xStep contient la valeur de stepgen.0.out (la source) il est possible de ré-utiliser le même signal pour envoyer la valeur à d'autres lecteurs, utiliser simplement le signal avec les autres lecteurs sur de nouvelles lignes:

```
# signal      destination2
net xStep => parport.0.pin-08-out
```

Ce qui peut également s'écrire en une seule ligne:

```
# signal      source      destination1      destination2
net xStep stepgen.0.out => parport.0.pin-02-out parport.0.pin-08-out
```

**Pins I/O** Les pins appelées I/O pins comme *index-enable*, ne suivent pas cette règle.

### 3.1.5 setp

La commande *setp* ajuste la valeur d'une pin ou d'un paramètre. Les valeurs valides dépendront du type de la pin ou du paramètre.

C'est une erreur si les types de donnée ne correspondent pas.

Certains composants ont des paramètres qui doivent être positionnés avant utilisation. Il n'est pas possible d'utiliser *setp* sur une pin connectée à un signal.

Syntaxe et exemple:

```
setp <pin/parameter-name> <value>

setp parport.0.pin-08-out TRUE
```

### 3.1.6 sets

La commande *sets* positionne la valeur d'un signal.

Syntaxe et exemple:

```
sets <signal-name> <value>

net mysignal and2.0.in0 pyvcp.my-led
sets mysignal 1
```

C'est une erreur si:

- Le nom de signal n'existe pas
- Le signal a déjà été écrit
- La valeur n'est pas du type correct pour le signal

### 3.1.7 unlinkp

La commande *unlinkp* déconnecte la pin du signal auquel elle est connectée. Si aucun signal n'a été connecté à la pin avant de lancer cette commande, rien ne se passe.

Syntaxe et exemple:

```
unlinkp <pin-name>

unlinkp parport.0.pin-02-out
```

### 3.1.8 Commandes obsolètes

Les commandes suivantes sont dépréciées et seront retirées dans les futures versions. Toute nouvelle configuration doit utiliser la commande [net](#).

### 3.1.9 linksp

La commande *linksp* a été remplacée par la commande *net*.

La commande *linksp* créait une *connexion* entre un signal et une pin.

Syntaxe et exemple:

```
linksp <signal-name> <pin-name>

linksp X-step parport.0.pin-02-out
```

### 3.1.10 linkps

La commande *linkps* a été remplacée par la commande *net*.

La commande *linksp* créait une *connexion* entre une pin et un signal. C'est la même chose que *linksp* mais les arguments sont inversés.

Syntaxe et exemple:

```
linkps <pin-name> <signal-name>

linkps parport.0.pin-02-out X-Step
```

### 3.1.11 newsig

the command *newsig* creates a new HAL signal by the name <signame> and the data type of <type>. Type must be *bit*, *s32*, *u32* or *float*. Error if <signame> already exists.

Syntaxe et exemple:

```
newsig <signame> <type>

newsig Xstep bit
```

D'autres informations peuvent être trouvées dans le manuel de HAL ou la man page de *halrun*.

## 3.2 HAL Data

1

### 3.2.1 Bit

A bit value is an on or off.

- bit values = true or 1 and false or 0 (True, TRUE, true are all valid)

### 3.2.2 Float

A *float* is a floating point number. In other words the decimal point can move as needed.

- float values = a 64 bit floating point value, with approximately 53 bits of resolution and over 1000 bits of dynamic range.

For more information on floating point numbers see:

[http://fr.wikipedia.org/wiki/Nombre\\_flottant](http://fr.wikipedia.org/wiki/Nombre_flottant)

### 3.2.3 s32

An *s32* number is a whole number that can have a negative or positive value.

- s32 values = integer numbers -2147483648 to 2147483647

### 3.2.4 u32

A *u32* number is a whole number that is positive only.

- u32 values = integer numbers 0 to 4294967295

---

1. NDT la description des données de HAL reste en Anglais, elle sont suffisamment simples pour être comprises.

### 3.3 Fichiers Hal

Si l'assistant StepConf a été utilisé pour générer la configuration trois fichiers HAL ont dû être créés dans le répertoire de la configuration.

- `ma-fraiseuse.hal` (si le nom de la config est "ma-fraiseuse") Ce fichier est chargé en premier, il ne doit pas être modifié sous peine de ne plus pouvoir l'utiliser avec l'assistant StepConf.
- `custom.hal` Ce fichier est le deuxième à être chargé et il l'est avant l'interface utilisateur graphique (GUI). C'est dans ce fichier que se trouvent les commandes personnalisées de l'utilisateur devant être chargées avant la GUI.
- `custom_postgui.hal` Ce fichier est chargé après la GUI. C'est dans ce fichier que se trouvent les commandes personnalisées de l'utilisateur devant être chargées après la GUI. Toutes les commandes relatives aux widgets de pyVCP doivent être placées ici.

### 3.4 Composants de HAL

Deux paramètres sont automatiquement ajoutés à chaque composant HAL quand il est créé. Ces paramètres permettent d'encadrer le temps d'exécution d'un composant.

`.time`

`.tmax`

*time* est le nombre de cycles du CPU qu'il a fallu pour exécuter la fonction.

*tmax* est le nombre maximum de cycles du CPU qu'il a fallu pour exécuter la fonction. *tmax* est un paramètre en lecture/écriture, de sorte que l'utilisateur peut le mettre à 0 pour se débarrasser du premier temps d'initialisation de la fonction.

### 3.5 Composants de logiques combinatoire

Hal contient plusieurs composants logiques temps réel. Les composants logiques suivent une table de vérité montrant les états logiques des sorties en fonction de l'état des entrées. Typiquement, la manipulation des bits d'entrée détermine l'état électrique des sorties selon la table de vérité des portes.

#### 3.5.1 and2

Le composant *and2* est une porte *and* à deux entrées. Sa table de vérité montre la sortie pour chaque combinaison des entrées.

Syntaxe

```
and2 [count=N] or [names=name1[,name2...]]
```

Fonctions

`and2.n`

Pins

```
and2.N.in0 (bit, in)
and2.N.in1 (bit, in)
and2.N.out (bit, out)
```

Table de vérité

in0	in1	out
False	False	False
True	False	False
False	True	False
True	True	True

### 3.5.2 not

Le composant *not* est un simple inverseur d'état.

Syntaxe

```
not [count=n] or [names=name1[,name2...]]
```

Fonctions

```
not.all  
not.n
```

Pins

```
not.n.in (bit, in)  
not.n.out (bit, out)
```

Table de vérité

in	out
True	False
False	True

### 3.5.3 or2

Le composant *or2* est une porte OR à deux entrées.

Syntaxe

```
or2[count=n] or [names=name1[,name2...]]
```

Functions

```
or2.n
```

Pins

```
or2.n.in0 (bit, in)  
or2.n.in1 (bit, in)  
or2.n.out (bit, out)
```

Table de vérité

in0	in1	out
True	False	True
True	True	True
False	True	True
False	False	False

### 3.5.4 xor2

Le composant *xor2* est une porte XOR à deux entrées (OU exclusif).

Syntaxe

```
xor2[count=n] or [names=name1[,name2...]]
```

Fonctions

```
xor2.n
```

Pins

```
xor2.n.in0 (bit, in)
xor2.n.in1 (bit, in)
xor2.n.out (bit, out)
```

Table de vérité

in0	in1	out
True	False	True
True	True	False
False	True	True
False	False	False

### 3.5.5 Exemples en logique combinatoire

Un exemple de connexion avec un "and2", deux entrées vers une sortie.

```
loadrt and2 count=1
addf and2.0 servo-thread
net my-sigin1 and2.0.in0 <= parport.0.pin-11-in
net my-sigin2 and2.0.in1 <= parport.0.pin-12-in
net both-on parport.0.pin-14-out <= and2.0.out
```

Dans cet exemple un and2 est chargé dans l'espace temps réel, puis ajouté à servo thread. Ensuite la broche d'entrée 11 du port parallèle est connectée à l'entrée in0 de la porte. Puis la broche d'entrée 12 du port est connectée à l'entrée in1 de la porte. Enfin la sortie and2.0.out de la porte est connectée à la broche de sortie 14 du port parallèle. Ainsi en suivant la table de vérité du and2, si les broches 11 et 12 du port sont à 1, alors sa sortie 14 est à 1 aussi.

## 3.6 Composants de conversion

### 3.6.1 Somme pondérée (weighted\_sum)

La somme pondérée converti un groupe de bits en un entier. La conversion est la somme des *poids* des bits présents plus n'importe quel offset. C'est similaire au *binaire codé décimal* mais avec plus d'options. Le bit *hold* interrompt le traitement des entrées, de sorte que la valeur *sum* ne change plus.

La syntaxe suivante est utilisée pour charger le composant weighted\_sum.

```
loadrt weighted_sum wsum_sizes=size[,size,...]
```

Crée des groupes de weighted\_sum, chacun avec le nombre donné de bits d'entrée (size).

Pour mettre à jour la weighted\_sum, le process\_wsums doit être attaché à un thread.

```
addf process_wsums servo-thread
```

Ce qui met à jour le composant weighted\_sum.

Dans l'exemple suivant, une copie de la fenêtre de configuration de HAL d'Axis, les bits 0 et 2 sont TRUE, ils n'ont pas d'offset. Le poids (*weight*) du bit 0 est 1, celui du bit 2 est 4, la somme est donc 5.

**weighted\_sum (somme pondérée)**

## Component Pins:

Owner	Type	Dir	Value	Name
10	bit	In	TRUE	wsum.0.bit.0.in
10	s32	I/O	1	wsum.0.bit.0.weight
10	bit	In	FALSE	wsum.0.bit.1.in
10	s32	I/O	2	wsum.0.bit.1.weight
10	bit	In	TRUE	wsum.0.bit.2.in
10	s32	I/O	4	wsum.0.bit.2.weight
10	bit	In	FALSE	wsum.0.bit.3.in
10	s32	I/O	8	wsum.0.bit.3.weight
10	bit	In	FALSE	wsum.0.hold
10	s32	I/O	0	wsum.0.offset
10	s32	Out	5	wsum.0.sum



## Chapitre 4

# Le tutoriel de HAL

### 4.1 Introduction

Halrun peut être utilisé pour créer un système complet et fonctionnel. Il s'agit d'un outil de configuration et de mise au point très puissant, en ligne de commande ou en fichier texte. Les exemples suivants illustrent son installation et son fonctionnement.

### 4.2 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' LinuxCNC depuis ses sources ou depuis un paquet. Si LinuxCNC a été compilé en *run-in-place*, la man page n'est pas installée, mais elle est accessible, dans le répertoire principal de LinuxCNC, taper:

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

#### 4.2.1 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
```

#### 4.2.2 L'environnement RTAPI

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

Pour remédier à ces différences, l'équipe de LinuxCNC 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 de LinuxCNC, vous pouvez étudier *linuxcnc/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.

## 4.3 Tutoriel simple

### 4.3.1 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 linuxcnc/src. Si nécessaire, invoquez le script *rip-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 linuxcnc
$linuxcnc 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 signaux. Une description complète de ce composant est disponible à la [section siggen](#) de ce document. Il s'agit d'un composant temps réel, mis en œuvre comme un module du noyau Linux. Pour charger *siggen* utiliser la commande de HAL, *loadrt*:

```
halcmd: loadrt siggen
```

### 4.3.2 Examiner HAL

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

```
halcmd: show comp

Loaded HAL Components:
ID      Type  Name           PID    State
3       RT    siggen         2177   ready
2       User  halcmd2177     2177   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 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. Ils 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. Les paramètres *thread.time* et *thread.tmax* sont associés avec le thread créé quand le composant a été chargé. Quand la réécriture de HAL sera terminée, le thread ne sera plus créé à ce stade, de sorte que ces paramètres ne seront plus visibles.

Il n'y a pas de thread créé ici, mais il y a quand même les paramètres *siggen.0.update.time* et *siggen.0.update.tmax*.

Les paramètres de thread sont ceux du composant 02, le module siggen. C'est incorrect, ils devraient être ceux du module *hal\_lib*, parce que le thread lui-même n'est plus la propriété de siggen, et si siggen est retiré, les paramètres devraient rester.

Et bien finalement, *fixer* les paramètres de thread aura pris plus de temps que je ne pensais. Donc, je les ai éliminés pour l'instant. Quand la réécriture de HAL sera terminée, je les remettrai.

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
00003  f801b000  fae820b8  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 zéro.<sup>1</sup>

### 4.3.3 Exécuter le code temps réel

Pour faire tourner le code actuellement contenu dans la fonction *siggen.0.update*, nous avons besoin d'un thread temps réel. C'est le composant appelé *threads* qui est utilisé pour créer le nouveau thread. Créons un thread appelé *test-thread* avec une période de 1 ms (1000  $\mu$ s ou 1000000 ns):

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

1. Les champs *CodeAddr* et *Arg* ont été utilisés pendant le développement et devraient probablement disparaître.

Voyons si il fonctionne:

```
halcmd: show thread
```

```

Realttime Threads:
  Period  FP      Name      (      Time, Max-Time )
  999855  YES      test-thread (      0,      0 )

```

Il fonctionne. La période n'est pas exactement de 1000000 ns 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 HAL. Mais cette fois-ci, nous avons utilisé la commande *addf* (add function) pour changer quelque chose dans 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
```

```

Realttime Threads:
  Period  FP      Name      (      Time, Max-Time )
  999855  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 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  -0.1640929  siggen.0.cosine
  3  float  IN           1  siggen.0.frequency
  3  float  IN           0  siggen.0.offset
  3  float  OUT  -0.4475303  siggen.0.sawtooth
  3  float  OUT   0.9864449  siggen.0.sine
  3  float  OUT          -1  siggen.0.square
  3  float  OUT  -0.1049393  siggen.0.triangle

```

Regardons encore une fois:

```
halcmd: show pin
```

```

Component Pins:
Owner  Type  Dir      Value  Name
  3  float  IN           1  siggen.0.amplitude
  3  float  OUT   0.0507619  siggen.0.cosine
  3  float  IN           1  siggen.0.frequency
  3  float  IN           0  siggen.0.offset
  3  float  OUT  -0.516165  siggen.0.sawtooth
  3  float  OUT   0.9987108  siggen.0.sine
  3  float  OUT          -1  siggen.0.square
  3  float  OUT  0.03232994  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.

#### 4.3.4 Modifier des paramètres

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

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

Voyons encore une fois les paramètres et les pins:

```
halcmd: show param
```

Parameters:

Owner	Type	Dir	Value	Name
3	s32	RO	1754	siggen.0.update.time
3	s32	RW	16997	siggen.0.update.tmax

```
halcmd: show pin
```

Component Pins:

Owner	Type	Dir	Value	Name
3	float	IN	5	siggen.0.amplitude
3	float	OUT	0.8515425	siggen.0.cosine
3	float	IN	1	siggen.0.frequency
3	float	IN	0	siggen.0.offset
3	float	OUT	2.772382	siggen.0.sawtooth
3	float	OUT	-4.926954	siggen.0.sine
3	float	OUT	5	siggen.0.square
3	float	OUT	0.544764	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.

#### 4.3.5 Enregistrer la configuration de HAL

La plupart de ce que nous avons fait jusqu'ici avec *halcmd* a été de simplement regarder les choses avec la commande *show*. Toutefois, deux commandes ont 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
```

### 4.3.6 Quitter halrun

Pour quitter halrun, ne pas fermez simplement la fenêtre de terminal sans avoir arrêté la session de HAL, pour l'arrêter correctement tapez:

```
halcmd: exit  
~/linuxcnc$
```

### 4.3.7 Restaurer la configuration de HAL

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

```
~/linuxcnc$ halrun -I -f saved.hal
```

Noter 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  
~/linuxcnc$
```

### 4.3.8 Suppression de la mémoire de HAL

Si un arrêt inattendu d'une session de HAL survient, il sera peut être nécessaire de décharger HAL de la mémoire avant de pouvoir lancer une autre session. Pour cela, taper la commande suivante dans une fenêtre de terminal:

```
~/linuxcnc$ halrun -U
```

## 4.4 Visualiser HAL avec halmeter

Il est possible de 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 HAL, est *halmeter*. C'est un programme très simple qui s'utilise comme un multimètre. Il permet d'observer 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 items 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 *Fermer*. Sur la petite fenêtre, cachée sous la grande à l'ouverture, le bouton *Sélectionner*, ré-ouvre la fenêtre de sélection et le bouton *Quitter* arrête le programme et ferme les fenêtres.

Il est possible d'ouvrir et de faire fonctionner simultanément plusieurs *halmeter*, 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, la pin, ou le 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.

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:

```
~/linuxcnc$ 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
```

#### 4.4.1 Lancement de halmeter

À ce stade, nous avons chargé le composant siggen, il 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 l'item à observer.

##### Fenêtre de sélection de halmeter



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 *Fermer*. Le dialogue de sélection se ferme et la mesure s'affiche dans une fenêtre semblable à la figure ci-dessous.

#### Affichage de la valeur



Pour modifier ce qui est affiché sur halmeter pressez le bouton *Sélectionner* 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 *Quitter*.

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

## 4.5 Tutoriel 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. L'exemple suivant 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

~/linuxcnc$ halrun
```

### 4.5.1 Installation des composants

Maintenant, nous allons charger le composant générateur d'impulsions. Pour l'instant, nous pouvons nous passer des détails et exécuter les commandes suivantes:<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  $\mu$ s 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
4  float IN      1  siggen.0.amplitude
4  float OUT     0  siggen.0.cosine
4  float IN      1  siggen.0.frequency
4  float IN      0  siggen.0.offset
4  float OUT     0  siggen.0.sawtooth
4  float OUT     0  siggen.0.sine
4  float OUT     0  siggen.0.square
4  float OUT     0  siggen.0.triangle
3  s32  OUT      0  stepgen.0.counts
3  bit  OUT    FALSE  stepgen.0.dir
3  bit  IN     FALSE  stepgen.0.enable
3  float OUT     0  stepgen.0.position-fb
3  bit  OUT    FALSE  stepgen.0.step
3  float IN      0  stepgen.0.velocity-cmd
3  s32  OUT      0  stepgen.1.counts
3  bit  OUT    FALSE  stepgen.1.dir
```

2. 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.

```

3 bit IN FALSE stepgen.1.enable
3 float OUT 0 stepgen.1.position-fb
3 bit OUT FALSE stepgen.1.step
3 float IN 0 stepgen.1.velocity-cmd

```

```
halcmd: show param
```

```
Parameters:
```

Owner	Type	Dir	Value	Name
4	s32	RO	0	siggen.0.update.time
4	s32	RW	0	siggen.0.update.tmax
3	u32	RW	0x00000001	stepgen.0.dirhold
3	u32	RW	0x00000001	stepgen.0.dirsetup
3	float	RO	0	stepgen.0.frequency
3	float	RW	0	stepgen.0.maxaccel
3	float	RW	0	stepgen.0.maxvel
3	float	RW	1	stepgen.0.position-scale
3	s32	RO	0	stepgen.0.rawcounts
3	u32	RW	0x00000001	stepgen.0.steplen
3	u32	RW	0x00000001	stepgen.0.stepspace
3	u32	RW	0x00000001	stepgen.1.dirhold
3	u32	RW	0x00000001	stepgen.1.dirsetup
3	float	RO	0	stepgen.1.frequency
3	float	RW	0	stepgen.1.maxaccel
3	float	RW	0	stepgen.1.maxvel
3	float	RW	1	stepgen.1.position-scale
3	s32	RO	0	stepgen.1.rawcounts
3	u32	RW	0x00000001	stepgen.1.steplen
3	u32	RW	0x00000001	stepgen.1.stepspace
3	s32	RO	0	stepgen.capture-position.time
3	s32	RW	0	stepgen.capture-position.tmax
3	s32	RO	0	stepgen.make-pulses.time
3	s32	RW	0	stepgen.make-pulses.tmax
3	s32	RO	0	stepgen.update-freq.time
3	s32	RW	0	stepgen.update-freq.tmax

## 4.5.2 Connexion des pins avec les 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 *files* pour connecter les modules ensemble. Dans HAL, les *files* sont appelés signaux. Nous devons en créer deux. Nous pouvons les appeler comme on veut, dans cet exemple il y aura *X-vel* et *Y-vel*. Le signal *X-vel* partira de la sortie cosinus du générateur de signaux 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	(linked to)
float	0	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
```

Nous pouvons aussi connecter l'axe Y au signal *Y-vel*. Il doit partir de la sortie sinus du générateur de signaux 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*.

### 4.5.3 Exécuter les réglages du temps réel - threads et fonctions

Penser à ce qui circule dans les *files* rend les pins et les signaux assez faciles à comprendre. Les threads et les fonctions sont un peu plus délicates à appréhender. 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  f9992000  fc731278 YES   0      siggen.0.update
00003  f998b20f  fc7310b8 YES   0      stepgen.capture-position
00003  f998b000  fc7310b8 NO    0      stepgen.make-pulses
00003  f998b307  fc7310b8 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, dent de scie, triangle, carrée. Pour générer un signal régulier, il doit fonctionner à des intervalles très précis.

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 comptant 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 elle fixe les sorties en conséquence. Pour des pas plus doux, elle doit fonctionner le plus souvent possible. Parce qu'elle a besoin de fonctionner de manière rapide, *make-pulses* est hautement optimisée et n'effectue que quelques calculs. Contrairement aux autres, elle 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 allons faire tourner *siggen.0.update* à une vitesse modérée pour le calcul des valeurs sinus et cosinus. Immédiatement après avoir lancé *siggen.0.update*, nous lançons *stepgen.0.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 précise. Regardons de quels threads nous disposons:

```
halcmd: show thread

Realttime Threads:
  Period  FP      Name              (      Time, Max-Time )
  996980  YES      slow              (      0,      0 )
  49849   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 exécuter la commande *show thread* une nouvelle fois pour voir ce qui se passe:

```
halcmd: show thread

Realttime Threads:
  Period  FP      Name              (      Time, Max-Time )
  996980  YES      slow              (      0,      0 )
    1 siggen.0.update
    2 stepgen.update-freq
  49849   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.

#### 4.5.4 Réglage des paramètres

Nous sommes presque prêts à démarrer notre système HAL. Mais il faut auparavant régler quelques paramètres. Par défaut le composant *siggen* génère des signaux qui varient entre +1 et -1. Pour notre exemple, c'est très bien, nous voulons que la vitesse de la table varie de +1 à -1 pouce par seconde. Toutefois, l'échelle du générateur d'impulsions de pas n'est pas bonne. Par défaut, il génère une fréquence de sortie de 1 pas par seconde avec une capacité de 1000. Il est fort improbable qu'un pas par seconde nous donne une vitesse de déplacement de la table d'un pouce par seconde. Supposons que notre vis fasse 5 tours par pouce, couplée à un moteur pas à pas de 200 pas par tour et une interface qui fournit 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.

#### 4.5.5 Lançons le!

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

```
halcmd: start
```

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

### 4.6 Voyons-y de plus près avec halscope

L'exemple précédent génère certains signaux très intéressants. Mais beaucoup de ce qui se passe est beaucoup trop rapide pour être vu avec halmeter. Pour examiner de plus près ce qui se passe à l'intérieur de HAL, il faudrait un oscilloscope. Heureusement HAL en offre un, appelé *halscope*. Il permet de capturer la valeur des pins, des signaux et des paramètres en fonction du temps.

#### 4.6.1 Démarrer halscope

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

```
halcmd: loadusr halscope
```

La fenêtre graphique du scope s'ouvre, immédiatement suivie par un dialogue *Fonction temps réel non liée* visible sur la figure ci-dessous:

**Dialogue Fonction temps réel non liée**



C'est dans ce dialogue que 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 échantillons, de sorte que nous puissions utiliser jusqu'à 4 canaux simultanément. Quand vous sélectionnez un thread puis que vous cliquez sur le bouton *OK*, le dialogue disparaît et la fenêtre initiale du scope s'ouvre, comme ci-dessous.

#### Fenêtre initiale du scope



#### 4.6.2 Branchement des sondes du scope

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

Les boutons des canaux se situent en dessous de l'écran du scope. Cliquez le bouton 1 et vous verrez apparaître le dialogue de sélection des sources dans lequel vous devrez choisir la source qui devra s'afficher sur le canal 1, comme sur la figure ci-dessous. Ce dialogue est très similaire à celui utilisé par halmeter.

##### Dialogue de sélection des sources



Nous aimerions bien regarder les signaux que nous avons défini précédemment, pour cela, cliquons sur l'onglet *Signaux* et le dialogue affichera tous les signaux existants dans HAL, dans notre exemple nous avons seulement les deux signaux *X-vel* et *Y-vel*, comme ci-dessous.

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

#### **Sélection du signal**





Le bouton du canal 1 est pressé, le numéro du canal 1 et le nom *X-vel* apparaissent sous la rangée de boutons. L'affichage indique toujours le canal sélectionné, vous pouvez avoir beaucoup de canaux sur l'écran, mais celui qui est actif sera en surbrillance.

**halscope**



Les différents contrôles comme la position verticale et l'amplitude sont toujours relatifs au canal 1. Pour ajouter un signal sur le canal 2, cliquer sur le bouton 2. Dans la fenêtre de dialogue, cliquer sur l'onglet *Signaux*, 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*.

### 4.6.3 Capturer notre première forme d'onde

Maintenant que nous avons plusieurs sondes branchées sur HAL, nous pouvons capturer quelques formes d'ondes. Pour démarrer le scope, cochez la case *Normal* du groupe *Mode "Run"* (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 (Trigger). Puisque nous n'en avons pas encore configuré, il attendra toujours. Pour déclencher manuellement, cliquez sur le bouton *Forcer* 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 ci-dessous.

#### Capture d'ondes



#### 4.6.4 Ajustement vertical

Les traces sont assez difficiles à distinguer car toutes les quatre sont les unes sur les autres. Pour résoudre ce problème, nous utilisons les curseurs du groupe *Vertical* situé à droite de l'écran. Ces deux curseurs agissent sur le canal actuellement sélectionné. En ajustant le *Gain*, notez qu'il couvre une large échelle (contrairement aux oscilloscopes réels), celle-ci permet d'afficher des signaux très petits (pico unités) à très grands (Tera - unités). Le curseur *Pos* déplace la trace affichée de haut en bas sur toute la hauteur de l'écran. Pour de plus grands ajustements le bouton *Offset* peut être utilisé.

##### Ajustement vertical



Le grand bouton *Canal sélectionné* en bas, indique que le canal 1 est actuellement le canal sélectionné et qu'il correspond au signal *X-vel*. Essayez de cliquer sur les autres canaux pour mettre leurs traces en évidence et pouvoir les déplacer avec le curseur *Pos*.

#### 4.6.5 Déclenchement (Triggering)

L'utilisation du bouton *Forcer* n'est parfois pas satisfaisante pour déclencher le scope. Pour régler un déclenchement 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, voir la figure ci-dessous. Sélectionner la sonde à utiliser pour déclencher en cliquant dessus. Pour notre exemple nous utilisons 3 canaux, essayons l'onde triangle. Quand le dialogue se referme, après le choix, le bouton affiche *Source Canal n* où *n* est le numéro du canal venant d'être choisi comme déclencheur.

##### Dialogue des sources de déclenchement



Après avoir défini la source de déclenchement, il est possible d'ajuster le niveau de déclenchement avec les curseurs du groupe *Trigger* le long du bord droit. Le niveau peut être modifié à partir du haut vers le bas de l'écran, il 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 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 déclenchement. Le point de déclenchement est visible comme une petite 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 *Montant* situé juste sous l'affichage du niveau de déclenchement, il deviendra alors *descendant*. Notez que la modification de la position de déclenchement arrête le scope une fois la position ajustée, vous relancez le scope en cliquant sur le bouton *Normal* du groupe *Mode "Run"*.

Maintenant que nous avons réglé la position verticale et le déclenchement, l'écran doit ressembler à la figure ci-dessous.

### Formes d'ondes avec déclenchement



#### 4.6.6 Ajustement horizontal

Pour examiner de près une partie d'une forme d'onde, vous pouvez utiliser le *zoom* au dessus de l'écran pour étendre la trace horizontalement et le curseur de position horizontale, *Pos* du groupe *Horizontal*, 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 le nombre d'échantillons pour avoir le dialogue *Sélectionner un taux d'échantillonnage*, figure ci-dessous. 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 sera de 4000 échantillons à 20kHz, soit environ 0.20 seconde.

#### Dialogue de choix d'échantillonnage



#### 4.6.7 Plus de canaux

Maintenant regardons les impulsions de pas. halscope dispose de 16 canaux, mais pour cet exemple, nous en utilisons seulement 4 à la fois. Avant de sélectionner tout autre canal, nous avons besoin d'en éteindre certains. Cliquer sur le canal 2, puis sur le bouton *Canal Off* sous le groupe *vertical*. Ensuite, cliquez sur le canal 3, le mettre Off et faire de même pour le canal 4. Même si les circuits sont éteints, ils sont encore en mémoire et restent connectés, en fait, nous continuerons à utiliser le canal 3 comme source de déclenchement. Pour ajouter de nouveaux canaux, sélectionner le canal 5, choisir la pin *stepgen.0.dir*, puis le canal 6 et sélectionner *stepgen.0.step*. Ensuite, cliquer sur *mode Normal* pour lancer le scope, ajustez le zoom horizontal à 10 ms par division. Vous devriez voir les impulsions de pas ralentir à la vitesse commandée approcher de zéro, puis la pin de direction changer d'état et les impulsions de pas se resserrer de nouveau en même temps que la vitesse augmente. Vous aurez peut être besoin d'ajuster le gain sur le canal 1 afin de mieux voir l'action de la vitesse sur l'évolution des impulsions de pas. Le résultat devrait être proche de celui de la figure ci-dessous. Ce type de mesure est délicate car il y a un énorme écart d'échelle entre la fréquence des pas et l'action sur la vitesse, d'où la courbe X-vel assez plate et les impulsions de pas très resserrées.

#### Observer les impulsions de pas



#### 4.6.8 Plus d'échantillons

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

```
halscmd: loadusr halscope 80000
```

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



## Chapitre 5

# Les fonctionnalités de Halshow

### 5.1 Le script Halshow

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

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

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

#### 5.1.1 Zone de l'arborescence de Hal

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



FIGURE 5.1 – La fenêtre de Halshow

L'arborescence montre toutes les parties principales de HAL. En face de chacune d'entre elles, se trouve un petit signe + ou - dans une case. Cliquer sur le signe plus pour déployer cette partie de l'arborescence et affichera son contenu. Si cette case affiche un signe moins, cliquer dessus repliera cette section de l'arborescence.

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



FIGURE 5.2 – L'onglet Montrer

### 5.1.2 Zone de l'onglet MONTRER

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

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

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

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

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

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

- Loadrt, unloadrt (chargement / déchargement en temps réel du module)
- Loadusr, unloadusr (chargement / déchargement de l'espace utilisateur des composants)
- addf, delf (ajout / suppression d'une fonction de / vers un fil en temps réel)
- net (créer une connexion entre deux ou plusieurs articles)
- setp (jeu de paramètres (ou broches) à une valeur)

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

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

Ndt: le message qui s'affiche au chargement de blocks ne l'empêche pas de fonctionner.

```
loadrt blocks ddt=1
```

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

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

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

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

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

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

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

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

Comme c'est juste pour visualiser, nous laissons la position en blanc pour obtenir la dernière position dans le thread. La figure [sur la commande addf](#) affiche l'état de halshow après que cette commande a été exécutée.



FIGURE 5.3 – Commande Addf

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

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

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

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

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

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

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

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

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

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

### 5.1.3 Zone de l'onglet WATCH

En cliquant sur l'onglet Watch, une zone vide s'affichera.<sup>1</sup> Vous pouvez ajouter des pins ou des signaux quand l'onglet Watch est ouvert, en cliquant sur leurs noms. La figure 4 montre cette zone avec plusieurs signaux de type *bit*. Parmi ces signaux, les enable-out pour les trois premiers axes et deux de la branche iocontrol, les signaux *estop*. Notez que les axes ne sont pas activés même si les signaux estop disent que LinuxCNC n'est pas en estop. Un bref regard sur TkLinuxCNC en arrière plan, montre que l'état de LinuxCNC est ESTOP RESET. L'activation des amplis ne deviendra pas vraie tant que la machine ne sera pas mise en marche.

#### L'onglet WATCH

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



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

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

## Chapitre 6

# Les composants de HAL

### 6.1 Composants de commandes et composants de l'espace utilisateur

Certaines de ces descriptions sont plus approfondies dans leurs pages man. Certaines y auront une description exhaustive, d'autres, juste une description limitée. Chaque composant a sa man page. La liste ci-dessous, montre les composants existants, avec le nom et le N° de section de leur page man. Par exemple dans une console, tapez *man axis* pour accéder aux informations de la man page d'Axis. Ou peut être *man 1 axis*, si le système exige le N° de section des man pages.

- *axis* - L'interface graphique AXIS pour LinuxCNC (The Enhanced Machine Controller).
- *axis-remote* - Interface de télécommande d'AXIS.
- *comp* - Crée, compile et installe des composants de HAL.
- *linuxcnc* - LINUXCNC (The Enhanced Machine Controller).
- *gladevcp* - Panneau de contrôle virtuel pour LinuxCNC, repose sur Glade, Gtk et les widgets HAL.
- *gs2* - composant de l'espace utilisateur de HAL, pour le variateur de fréquence GS2 de la société *Automation Direct*.
- *halcmd* - Manipulation de HAL, depuis la ligne de commandes.
- *hal\_input* - Contrôler des pins d'entrée de HAL avec n'importe quel matériel supporté par Linux, y compris les matériels USB HID.
- *halmeter* - Observer les pins de HAL, ses signaux et ses paramètres.
- *halrun* - Manipulation de HAL, depuis la ligne de commandes.
- *halsampler* - Échantillonner des données temps réel depuis HAL.
- *halstreamer* - Créer un flux de données temps réel dans HAL depuis un fichier.
- *halui* - Observer des pins de HAL et commander LinuxCNC au travers d'NML.
- *io* - Accepte les commandes NML I/O, interagi avec HAL dans l'espace utilisateur.
- *iocontrol* - Accepte les commandes NML I/O, interagi avec HAL dans l'espace utilisateur.
- *pyvcp* - Panneau de Contrôle Virtuel pour LinuxCNC (Python Virtual Control Panel).
- *shuttlepress* - Contrôle des pins de HAL avec la manette ShuttleXpress, de la société *Contour Design*.

### 6.2 Composants temps réel et modules du noyau

Certaines de ces descriptions sont plus approfondies dans leur man page. Certaines auront juste une description limitée. Chaque composant a sa man page. A partir de cette liste vous connaîtrez quels composants existent avec le nom et le N° de leur man page permettant d'avoir plus de détails.

---

**Note**

Si le composant requière un thread avec flottant, c'est normalement le plus lent, soit *servo-thread*.

---



### 6.2.1 Composants du coeur de LinuxCNC

- *motion* - Accepte les commandes de mouvement NML, interagi en temps réel avec HAL.
- *axis* - Commandes de mouvement NML acceptées, interagi en temps réel avec HAL
- *classicladder* - Automate temps réel programmable en logique Ladder.
- *gladevcp* - Affiche un panneau de contrôle virtuel construit avec GladeVCP.
- *threads* - Crée des threads de HAL temps réel.

### 6.2.2 Composants binaires et logiques

- *and2* - Porte AND (ET) à deux entrées.
- *not* - Inverseur.
- *or2* - Porte OR (OU) à deux entrées.
- *xor2* - Porte XOR (OU exclusif) à deux entrées.
- *debounce* - Filtre une entrée digitale bruitée (typiquement antirebond).
- *edge* - Détecteur de front.
- *flipflop* - Bascule D.
- *oneshot* - Générateur d'impulsion monostable. Crée sur sa sortie une impulsion de longueur variable quand son entrée change d'état.
- *logic* - Composant expérimental de logique générale.
- *lut5* - Fonction logique arbitraire à cinq entrées, basée sur une table de correspondance.
- *match8* - Détecteur de coïncidence binaire sur 8 bits.
- *select8* - Détecteur de coïncidence binaire sur 8 bits.

### 6.2.3 Composants arithmétiques et flottants

- *abs* - Calcule la valeur absolue et le signe d'un signal d'entrée.
- *blend* - Provoque une interpolation linéaire entre deux valeurs
- *comp* - Comparateur à deux entrées avec hystérésis.
- *constant* - Utilise un paramètre pour positionner une pin.
- *sum2* - Somme de deux entrées (chacune avec son gain) et d'un offset.
- *counter* - Comptage d'impulsions d'entrée (obsolète).

Utiliser le composant *encoder* avec ... *counter-mode* = *TRUE*. Voir la section [codeur](#).

- *updown* - Compteur/décompteur avec limites optionnelles et bouclage en cas de dépassement.
- *ddt* - Calcule la dérivée de la fonction d'entrée.
- *deadzone* - Retourne le centre si il est dans le seuil.
- *hypot* - Calculateur d'hypoténuse à trois entrées (distance Euclidienne).
- *mult2* - Le produit de deux entrées.
- *mux16* - Sélection d'une valeur d'entrée sur seize.
- *mux2* - Sélection d'une valeur d'entrée sur deux.
- *mux4* - Sélection d'une valeur d'entrée sur quatre.
- *mux8* - Sélection d'une valeur d'entrée sur huit.
- *near* - Détermine si deux valeurs sont à peu près égales.
- *offset* - Ajoute un décalage à une entrée et la soustrait à la valeur de retour.
- *integ* - Intégrateur.

- *invert* - Calcule l'inverse du signal d'entrée.
- *wcomp* - Comparateur à fenêtre.
- *weighted\_sum* - Converti un groupe de bits en un entier.
- *biquad* - Filtre biquad IIR
- *lowpass* - Filtre passe-bas.
- *limit1* - Limite le signal de sortie pour qu'il soit entre min et max.<sup>1</sup>
- *limit2* - Limite le signal de sortie pour qu'il soit entre min et max. Limite sa vitesse de montée à moins de MaxV par seconde.<sup>2</sup>
- *limit3* - Limite le signal de sortie pour qu'il soit entre min et max. Limite sa vitesse de montée à moins de MaxV par seconde. Limite sa dérivée seconde à moins de MaxA par seconde carré.<sup>3</sup>
- *maj3* - Calcule l'entrée majoritaire parmi 3.
- *scale* - Applique une échelle et un décalage à son entrée.

## 6.2.4 Conversions de type

- *conv\_bit\_s32* - Converti une valeur de bit vers s32 (entier 32 bits signé).
- *conv\_bit\_u32* - Converti une valeur de bit vers u32 (entier 32 bit non signé).
- *conv\_float\_s32* - Converti la valeur d'un flottant vers s32.
- *conv\_float\_u32* - Converti la valeur d'un flottant vers u32.
- *conv\_s32\_bit* - Converti une valeur de s32 en bit.
- *conv\_s32\_float* - Converti une valeur de s32 en flottant.
- *conv\_s32\_u32* - Converti une valeur de s32 en u32.
- *conv\_u32\_bit* - Converti une valeur de u32 en bit.
- *conv\_u32\_float* - Converti une valeur de u32 en flottant.
- *conv\_u32\_s32* - Converti une valeur de u32 en s32.

## 6.2.5 Pilotes de matériel

- *hm2\_7i43* - Pilote HAL pour les cartes *Mesa Electronics 7i43 EPP*, toutes les cartes avec HostMot2.
- *hm2\_pci* - Pilote HAL pour les cartes *Mesa Electronics 5i20, 5i22, 5i23, 4i65 et 4i68*, toutes les cartes avec micro logiciel HostMot2.
- *hostmot2* - Pilote HAL pour micro logiciel *Mesa Electronics HostMot2*.
- *mesa\_7i65* - Support pour la carte huit axes Mesa 7i65 pour servomoteurs.
- *pluto\_servo* - Pilote matériel et micro programme pour la carte *Pluto-P parallel-port FPGA*, utilisation avec servomoteurs.
- *pluto\_step* - Pilote matériel et micro programme pour la carte *Pluto-P parallel-port FPGA*, utilisation avec moteurs pas à pas.
- *thc* - Contrôle de la hauteur de torche, en utilisant une carte Mesa THC.
- *serport* - Pilote matériel pour les entrées/sorties numériques de port série avec circuits 8250 et 16550.

## 6.2.6 Composants cinématiques

- *kins* - Définition des cinématiques pour linuxcnc.
- *gantrykins* - Module de cinématique pour un seul axe à articulations multiples.
- *genhexkins* - Donne six degrés de liberté en position et en orientation (XYZABC). L'emplacement des moteurs est défini au moment de la compilation.

---

1. Lorsque l'entrée est une position, cela signifie que la *position* est limitée.

2. Lorsque l'entrée est une position, cela signifie que la *position* et la *vitesse* sont limitées.

3. Lorsque l'entrée est une position, cela signifie que la *position*, la *vitesse* et l'*accélération* sont limitées.

- *genserkins* - Cinématique capable de modéliser une bras manipulateur avec un maximum de 6 articulations angulaires.
- *maxkins* - Cinématique d'une fraiseuse 5 axes nommée *max*, avec tête inclinable (axe B) et un axe rotatif horizontal monté sur la table (axe C). Fourni les mouvements UVW dans le système de coordonnées système basculé. Le fichier source, *maxkins.c*, peut être un point de départ utile pour d'autres systèmes 5 axes.
- *tripodkins* - Les articulations représentent la distance du point contrôlé à partir de trois emplacements prédéfinis (les moteurs), ce qui donne trois degrés de liberté en position (XYZ).
- *trivkins* - Il y a une correspondance 1:1 entre les articulations et les axes. La plupart des fraiseuses standard et des tours utilisent ce module de cinématique triviale.
- *pumakins* - Cinématique pour robot style PUMA.
- *rotatekins* - Les axes X et Y sont pivotés de 45 degrés par rapport aux articulations 0 et 1.
- *scarakins* - Cinématique des robots de type SCARA.

### 6.2.7 Composants de contrôle moteur

- *at\_pid* - Contrôleur Proportionnelle/Intégrale/dérivée avec réglage automatique.
- *pid* - Contrôleur Proportionnelle/Intégrale/dérivée.
- *pwmgen* - Générateur logiciel de PWM/PDM, voir la section [PWMgen](#)
- *encoder* - Comptage logiciel de signaux de codeur en quadrature, voir la section [codeur](#)
- *stepgen* - Générateur d'impulsions de pas logiciel, voir la section [stepgen](#)

### 6.2.8 BLDC and 3-phase motor control

- *bldc\_hall3* - Commutateur bipolaire trapézoïdal à 3 directions pour moteur sans balais (BLDC) avec capteurs de Hall.
- *clarke2* - Transformation de Clarke, version à deux entrées.
- *clarke3* - Transformation de Clarke, à 3 entrées vers cartésien.
- *clarkeinv* - Transformation de Clarke inverse.

### 6.2.9 Autres composants

- *charge\_pump* - Crée un signal carré destiné à l'entrée *pompe de charge* de certaines cartes de contrôle. Le composant *charg\_pump* doit être ajouté à *base thread*. Quand il est activé, sa sortie est haute pour une période puis basse pour une autre période. Pour calculer la fréquence de sortie faire  $1/(\text{durée de la période en secondes} * 2) = \text{fréquence en Hz}$ . Par exemple, si vous avez une période de base de 100000ns soit 0.0001 seconde, la formule devient:  $1/(0.0001 * 2) = 5000 \text{ Hz}$  ou 5kHz.
- *encoder\_ratio* - Un engrenage électronique pour synchroniser deux axes.
- *estop\_latch* - Verrou d'Arrêt d'Urgence.
- *feedcomp* - Multiplie l'entrée par le ratio vitesse courante / vitesse d'avance travail.
- *gearchange* - Sélectionne une grandeur de vitesse parmi deux.
- *ilowpass* - Filtre passe-bas avec entrées et sorties au format entier.  
Sur une machine ayant une grande accélération, un petit jog peut s'apparenter à une avance par pas. En intercalant un filtre *ilowpass* entre la sortie de comptage du codeur de la manivelle et l'entrée *jog-counts* de l'axe, le mouvement se trouve lissé.  
Choisir prudemment l'échelle, de sorte que durant une simple session, elle ne dépasse pas environ 2e9/scale impulsions visibles sur le MPG. Choisir le gain selon le niveau de douceur désiré. Diviser les valeurs de axis.N.jog-scale par l'échelle.
- *joyhandle* - Définit les mouvements d'un joystick non linéaire, zones mortes et échelles.
- *knob2float* - Convertisseur de comptage (probablement d'un codeur) vers une valeur en virgule flottante.
- *minmax* - Suiveur de valeurs minimum et maximum de l'entrée vers les sorties.
- *sample\_hold* - Échantillonneur bloqueur.
- *sampler* - Échantillonneur de données de HAL en temps réel.

- *siggen* - Générateur de signal, voir la section [siggen](#)
- *sim\_encoder* - Codeur en quadrature simulé, voir la section [codeur simulé](#)
- *sphereprobe* - Sonde hémisphérique.
- *steptest* - Utilisé par Stepconf pour permettre de tester les valeurs d'accélération et de vitesse d'un axe.
- *streamer* - Flux temps réel depuis un fichier vers HAL.
- *supply* - Set output pins with values from parameters (obsolète).
- *threadtest* - Composant de HAL pour tester le comportement des threads.
- *time* - Compteur de temps écoulé HH:MM:SS avec entrée *actif*.
- *timedelay* - L'équivalent d'un relais temporisé.
- *timedelta* - Composant pour mesurer le comportement temporel des threads.
- *toggle2nist* - Bouton à bascule pour logique NIST.
- *toggle* - Bouton à bascule NO/NF à partir d'un bouton poussoir momentané.
- *tristate\_bit* - Place un signal sur une pin d'I/O seulement quand elle est validée, similaire à un tampon trois états en électronique.
- *tristate\_float* - Place un signal sur une pin d'I/O seulement quand elle est validée, similaire à un tampon trois états en électronique.
- *watchdog* - Moniteur de fréquence (chien de garde) sur 1 à 32 entrées.

### 6.3 Appels à l'API de HAL (liste de la section 3 des man pages)

```
hal_add_funcnt_to_thread.3hal
hal_bit_t.3hal
hal_create_thread.3hal
hal_del_funcnt_from_thread.3hal
hal_exit.3hal
hal_export_funcnt.3hal
hal_float_t.3hal
hal_get_lock.3hal
hal_init.3hal
hal_link.3hal
hal_malloc.3hal
hal_param_bit_new.3hal
hal_param_bit_newf.3hal
hal_param_float_new.3hal
hal_param_float_newf.3hal
hal_param_new.3hal
hal_param_s32_new.3hal
hal_param_s32_newf.3hal
hal_param_u32_new.3hal
hal_param_u32_newf.3hal
hal_parport.3hal
hal_pin_bit_new.3hal
hal_pin_bit_newf.3hal
hal_pin_float_new.3hal
hal_pin_float_newf.3hal
hal_pin_new.3hal
hal_pin_s32_new.3hal
hal_pin_s32_newf.3hal
hal_pin_u32_new.3hal
hal_pin_u32_newf.3hal
hal_ready.3hal
```

hal\_s32\_t.3hal  
hal\_set\_constructor.3hal  
hal\_set\_lock.3hal  
hal\_signal\_delete.3hal  
hal\_signal\_new.3hal  
hal\_start\_threads.3hal  
hal\_type\_t.3hal  
hal\_u32\_t.3hal  
hal\_unlink.3hal  
intro.3hal  
undocumented.3hal

## 6.4 Appels à RTAPI

EXPORT\_FUNCTION.3rtapi  
MODULE\_AUTHOR.3rtapi  
MODULE\_DESCRIPTION.3rtapi  
MODULE\_LICENSE.3rtapi  
RTAPI\_MP\_ARRAY\_INT.3rtapi  
RTAPI\_MP\_ARRAY\_LONG.3rtapi  
RTAPI\_MP\_ARRAY\_STRING.3rtapi  
RTAPI\_MP\_INT.3rtapi  
RTAPI\_MP\_LONG.3rtapi  
RTAPI\_MP\_STRING.3rtapi  
intro.3rtapi  
rtapi\_app\_exit.3rtapi  
rtapi\_app\_main.3rtapi  
rtapi\_clock\_set\_period.3rtapi  
rtapi\_delay.3rtapi  
rtapi\_delay\_max.3rtapi  
rtapi\_exit.3rtapi  
rtapi\_get\_clocks.3rtapi  
rtapi\_get\_msg\_level.3rtapi  
rtapi\_get\_time.3rtapi  
rtapi\_inb.3rtapi  
rtapi\_init.3rtapi  
rtapi\_module\_param.3rtapi  
RTAPI\_MP\_ARRAY\_INT.3rtapi  
RTAPI\_MP\_ARRAY\_LONG.3rtapi  
RTAPI\_MP\_ARRAY\_STRING.3rtapi  
RTAPI\_MP\_INT.3rtapi  
RTAPI\_MP\_LONG.3rtapi  
RTAPI\_MP\_STRING.3rtapi  
rtapi\_mutex.3rtapi  
rtapi\_outb.3rtapi  
rtapi\_print.3rtapi  
rtapi\_prio.3rtapi  
rtapi\_prio\_highest.3rtapi  
rtapi\_prio\_lowest.3rtapi  
rtapi\_prio\_next\_higher.3rtapi  
rtapi\_prio\_next\_lower.3rtapi  
rtapi\_region.3rtapi  
rtapi\_release\_region.3rtapi  
rtapi\_request\_region.3rtapi  
rtapi\_set\_msg\_level.3rtapi

---

```
rtapi_shmem.3rtapi  
rtapi_shmem_delete.3rtapi  
rtapi_shmem_getptr.3rtapi  
rtapi_shmem_new.3rtapi  
rtapi_snprintf.3rtapi  
rtapi_task_delete.3rtapi  
rtapi_task_new.3rtapi  
rtapi_task_pause.3rtapi  
rtapi_task_resume.3rtapi  
rtapi_task_start.3rtapi  
rtapi_task_wait.3rtapi
```

## Chapitre 7

# Les composants temps réel

### 7.1 Stepgen

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

**Diagramme bloc du générateur de pas stepgen**



### 7.1.1 L'installier

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



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

```
halcmd: loadrt stepgen step_type=0,0,2 ctrl_type=p,p,v
```

va installer trois générateurs de pas. Les deux premiers utilisent le type de pas 0 (pas et direction) et fonctionnent en mode position. Le dernier utilise le type de pas 2 (quadrature) et fonctionne en mode vitesse. La valeur par défaut de `<config-array>` est 0,0,0 qui va installer trois générateurs de type 0 (step/dir). Le nombre maximum de générateurs de pas est de 8 (comme définit par MAX\_CHAN dans stepgen.c). Chaque générateur est indépendant, mais tous sont actualisés par la même fonction(s), au même instant. Dans les descriptions qui suivent, `<chan>` est le nombre de générateurs spécifiques. La numérotation des générateurs commence à 0.

### 7.1.2 Le désinstaller

```
halcmd: unloadrt stepgen
```

### 7.1.3 Pins

Chaque générateur d'impulsions de pas n'aura que certaines de ces pins, selon le type de pas et le mode de contrôle sélectionné.

- (float) `stepgen.<chan>.position-cmd` — Position désirée du moteur, en unités de longueur (mode position seulement).
- (float) `stepgen.<chan>.velocity-cmd` — Vitesse désirée du moteur, en unités de longueur par seconde (mode vitesse seulement).
- (s32) `stepgen.<chan>.counts` — Rétroaction de la position en unités de comptage, actualisée par la fonction `capture_position()`.
- (float) `stepgen.<chan>.position-fb` — Rétroaction de la position en unités de longueur, actualisée par la fonction `capture_position()`.
- (bit) `stepgen.<chan>.step` — Sortie des impulsions de pas (type de pas 0 seulement).
- (bit) `stepgen.<chan>.dir` — Sortie direction (type de pas 0 seulement).
- (bit) `stepgen.<chan>.up` — Sortie UP en pseudo-PWM (type de pas 1 seulement).
- (bit) `stepgen.<chan>.down` — Sortie DOWN en pseudo-PWM (type de pas 1 seulement).
- (bit) `stepgen.<chan>.phase-A` — Sortie phase A (séquences de pas 2 à 14 seulement).
- (bit) `stepgen.<chan>.phase-B` — Sortie phase B (séquences de pas 2 à 14 seulement).
- (bit) `stepgen.<chan>.phase-C` — Sortie phase C (séquences de pas 3 à 14 seulement).
- (bit) `stepgen.<chan>.phase-D` — Sortie phase D (séquences de pas 5 à 14 seulement).
- (bit) `stepgen.<chan>.phase-E` — Sortie phase E (séquences de pas 11 à 14 seulement).

### 7.1.4 Paramètres

- (float) `stepgen.<chan>.position-scale` — Pas par unité de longueur. Ce paramètre est utilisé pour les sorties et les rétroactions.
- (float) `stepgen.<chan>.maxvel` — Vitesse maximale, en unités de longueur par seconde. Si égal à 0.0, n'a aucun effet.
- (float) `stepgen.<chan>.maxaccel` — Valeur maximale d'accélération, en unités de longueur par seconde au carré. Si égal à 0.0, n'a aucun effet.
- (float) `stepgen.<chan>.frequency` — Fréquence des pas, en pas par seconde.
- (float) `stepgen.<chan>.steplen` — Durée de l'impulsion de pas (types de pas 0 et 1) ou durée minimum dans un état donné (séquences de pas 2 à 14), en nanosecondes.
- (float) `stepgen.<chan>.stepspace` — Espace minimum entre deux impulsions de pas (types de pas 0 et 1 seulement), en nanosecondes.
- (float) `stepgen.<chan>.dirsetup` — Durée minimale entre un changement de direction et le début de la prochaine impulsion de pas (type de pas 0 seulement), en nanosecondes.

- (float) *stepgen.<chan>.dirhold* — Durée minimale entre la fin d’une impulsion de pas et un changement de direction (type de pas 0 seulement), en nanosecondes.
- (float) *stepgen.<chan>.dirdelay* — Durée minimale entre un pas dans une direction et un pas dans la direction opposée (séquences de pas 1 à 14 seulement), en nanosecondes.
- (s32) *stepgen.<chan>.rawcounts* — Valeur de comptage brute (count) de la rétroaction, réactualisée par la fonction *make\_pulses()*.

En mode position, les valeurs de *maxvel* et de *maxaccel* sont utilisées par la boucle de position interne pour éviter de générer des trains d’impulsions de pas que le moteur ne peut pas suivre. Lorsqu’elles sont réglées sur des valeurs appropriées pour le moteur, même un grand changement instantané dans la position commandée produira un mouvement trapézoïdal en douceur vers la nouvelle position. L’algorithme fonctionne en mesurant à la fois, l’erreur de position et l’erreur de vitesse, puis en calculant une accélération qui tend à réduire vers zéro, les deux en même temps. Pour plus de détails, y compris les contenus de la boîte *d’équation de contrôle*, consulter le code source.

En mode vitesse, *maxvel* est une simple limite qui est appliquée à la vitesse commandée, *maxaccel* est utilisé pour créer une rampe avec la fréquence actuelle, si la vitesse commandée change brutalement. Comme dans le mode position, des valeurs appropriées de ces paramètres assurent que le moteur pourra suivre le train d’impulsions généré.

### 7.1.5 Séquences de pas

Le générateur de pas supporte 15 différentes *séquences de pas*. Le type de pas 0 est le plus familier, c’est le standard pas et direction (step/dir). Quand *stepgen* est configuré pour le type 0, il y a quatre paramètres supplémentaires qui déterminent le timing exact des signaux de pas et de direction. Voir la figure ci-dessous pour la signification de ces paramètres. Les paramètres sont en nanosecondes, mais ils doivent être arrondis à un entier, multiple de la période du thread qui appelle *make\_pulses()*. Par exemple, si *make\_pulses()* est appelée toutes les 16µs et que *steplen* est à 20000, alors l’impulsion de pas aura une durée de  $2 \times 16 = 32\mu s$ . La valeur par défaut de ces quatre paramètres est de 1ns, mais l’arrondi automatique prendra effet au premier lancement du code. Puisqu’un pas exige d’être haut pendant *steplen* ns et bas pendant *stepspace* ns, la fréquence maximale est  $1.000.000.000$  divisé par (*steplen*+*stepspace*). Si *maxfreq* est réglé plus haut que cette limite, il sera abaissé automatiquement. Si *maxfreq* est à zéro, il restera à zéro, mais la fréquence de sortie sera toujours limitée.

#### Timing pas et direction



Le type de pas 1 a deux sorties, up et down. Les impulsions apparaissent sur l’une ou l’autre, selon la direction du déplacement. Chaque impulsion a une durée de *steplen* ns et les impulsions sont séparées de *stepspace* ns. La fréquence maximale est la même que pour le type 0. Si *maxfreq* est réglé plus haut que cette limite il sera abaissé automatiquement. Si *maxfreq* est à zéro, il restera à zéro, mais la fréquence de sortie sera toujours limitée.

Les séquences 2 jusqu’à 14 sont basées sur les états et ont entre deux et cinq sorties. Pour chaque pas, un compteur d’état est incrémenté ou décrémenté. Les figures suivantes :

- [Trois phases en quadrature](#),
- [Quatre phases](#),
- [Cinq phases](#)

montrent les différentes séquences des sorties en fonction de l'état du compteur. La fréquence maximale est 1.000.000.000 ( $1 \times 10^9$ ) divisé par *steplen* et comme dans les autres séquences, *maxfreq* sera abaissé si il est au dessus de cette limite.

**Séquences de pas à trois phases**

**STEP TYPE 2**



**STEP TYPE 3**



**STEP TYPE 4**



**Séquences de pas à quatre phases**



Séquence de pas à cinq phases



## 7.1.6 Fonctions

Le composant exporte trois fonctions. Chaque fonction agit sur tous les générateurs d'impulsions de pas. Lancer différents générateurs dans différents threads n'est pas supporté.

- (funct) *stepgen.make-pulses* — Fonction haute vitesse de génération et de comptage des impulsions (non flottant).
  - (funct) *stepgen.update-freq* — Fonction basse vitesse de conversion de position en vitesse, mise à l'échelle et traitement des limitations.
  - (funct) *stepgen.capture-position* — Fonction basse vitesse pour la rétroaction, met à jour les latches et les mesures de position.
- La fonction à grande vitesse *stepgen.make-pulses* devrait être lancée dans un thread très rapide, entre 10 et 50µs selon les capacités de l'ordinateur. C'est la période de ce thread qui détermine la fréquence maximale des pas, de *steplen*, *stepspace*, *dirsetup*, *dirhold* et *dirdelay*, tous sont arrondis au multiple entier de la période du thread en nanosecondes. Les deux autres fonctions peuvent être appelées beaucoup plus lentement.

## 7.2 PWMgen

Ce composant fournit un générateur logiciel de PWM (modulation de largeur d'impulsions) et PDM (modulation de densité d'impulsions). C'est un composant temps réel uniquement, dépendant de plusieurs facteurs comme la vitesse du CPU, etc. Il est capable de générer des fréquences PWM de quelques centaines de Hertz en assez bonne résolution, à peut-être 10kHz avec une résolution limitée.

### 7.2.1 L'installer

```
halcmd: loadrt pwmgen output_type=<config-array>
```

<config-array> est une série d'entiers décimaux séparés par des virgules. Chaque chiffre provoquera le chargement d'un simple générateur de PWM, la valeur de ce chiffre déterminera le type de sortie.

#### Exemple avec pwmgen

```
halcmd: loadrt pwmgen output_type=0,1,2
```

va installer trois générateurs de PWM. Le premier utilisera une sortie de type 0 (PWM seule), le suivant utilisera une sortie de type 1 (PWM et direction) et le troisième utilisera une sortie de type 2 (UP et DOWN). Il n'y a pas de valeur par défaut, si <config-array> n'est pas spécifié, aucun générateur de PWM ne sera installé. Le nombre maximum de générateurs de fréquences est de 8 (comme définit par MAX\_CHAN dans pwmgen.c). Chaque générateur est indépendant, mais tous sont mis à jour par la même fonction(s), au même instant. Dans les descriptions qui suivent, <chan> est le nombre de générateurs spécifiques. La numérotation des générateurs de PWM commence à 0.

### 7.2.2 Le désinstaller

```
halcmd: unloadrt pwmgen
```

### 7.2.3 Pins

Chaque générateur de PWM aura les pins suivantes:

- (float) *pwmgen.<chan>.value* — Valeur commandée, en unités arbitraires. Sera mise à l'échelle par le paramètre d'échelle (voir ci-dessous).
  - (bit) *pwmgen.<chan>.enable* — Active ou désactive les sorties du générateur de PWM.
- Chaque générateur de PWM aura également certaines de ces pins, selon le type de sortie choisi:
- (bit) *pwmgen.<chan>.pwm* — Sortie PWM (ou PDM), (types de sortie 0 et 1 seulement).
  - (bit) *pwmgen.<chan>.dir* — Sortie direction (type de sortie 1 seulement).
  - (bit) *pwmgen.<chan>.up* — Sortie PWM/PDM pour une valeur positive en entrée (type de sortie 2 seulement).
  - (bit) *pwmgen.<chan>.down* — Sortie PWM/PDM pour une valeur négative en entrée (type de sortie 2 seulement).

### 7.2.4 Paramètres

- (float) *pwmgen.<chan>.scale* — Facteur d'échelle pour convertir les valeurs en unités arbitraires, en coefficients de facteur cyclique.
- (float) *pwmgen.<chan>.pwm-freq* — Fréquence de PWM désirée, en Hz. Si égale à 0.0, la modulation sera PDM au lieu de PWM. Si elle est réglée plus haute que les limites internes, au prochain appel de la fonction *update\_freq()* elle sera ramenée aux limites internes. Si elle est différente de zéro et si *le lissage* est faux, au prochain appel de la fonction *update\_freq()* elle sera réglée au plus proche entier multiple de la période de la fonction *make\_pulses()*.
- (bit) *pwmgen.<chan>.dither-pwm* — Si vrai, active le lissage pour affiner la fréquence PWM ou le rapport cyclique qui ne pourraient pas être obtenus avec une pure PWM. Si faux, la fréquence PWM et le rapport cyclique seront tous les deux arrondis aux valeurs pouvant être atteintes exactement.
- (float) *pwmgen.<chan>.min-dc* — Rapport cyclique minimum compris entre 0.0 et 1.0 (Le rapport cyclique sera à zéro quand il est désactivé, indépendamment de ce paramètre).
- (float) *pwmgen.<chan>.max-dc* — Rapport cyclique maximum compris entre 0.0 et 1.0.
- (float) *pwmgen.<chan>.curr-dc* — Rapport cyclique courant, après toutes les limitations et les arrondis (lecture seule).

### 7.2.5 Types de sortie

Le générateur de PWM supporte trois *types de sortie*.

- Le *type 0* - A une seule pin de sortie. Seules, les commandes positives sont acceptées, les valeurs négatives sont traitées comme zéro (elle seront affectées par le paramètre *min-dc* si il est différent de zéro).
- Le *type 1* - A deux pins de sortie, une pour le signal PWM/PDM et une pour la direction. Le rapport cyclique d'une pin PWM est basé sur la valeur absolue de la commande, de sorte que les valeurs négatives sont acceptables. La pin de direction est fausse pour les commandes positives et vraie pour les commandes négatives.
- Le *type 2* - A également deux sorties, appelées *up* et *down*. Pour les commandes positives, le signal PWM apparaît sur la sortie *up* et la sortie *down* reste fausse. Pour les commandes négatives, le signal PWM apparaît sur la sortie *down* et la sortie *up* reste fausse. Les sorties de type 2 sont appropriées pour piloter la plupart des ponts en H.

### 7.2.6 Fonctions

Le composant exporte deux fonctions. Chaque fonction agit sur tous les générateurs de PWM, lancer différents générateurs dans différents threads n'est pas supporté.

- (funct) *pwmgen.make-pulses* — Fonction haute vitesse de génération de fréquences PWM (non flottant).
- (funct) *pwmgen.update* — Fonction basse vitesse de mise à l'échelle, limitation des valeurs et traitement d'autres paramètres.

La fonction haute vitesse *pwmgen.make-pulses* devrait être lancée dans un thread très rapide, entre 10 et 50 us selon les capacités de l'ordinateur. C'est la période de ce thread qui détermine la fréquence maximale de la porteuse PWM, ainsi que la résolution des signaux PWM ou PDM. L'autre fonction peut être appelée beaucoup plus lentement.

## 7.3 Codeur

Ce composant fournit, en logiciel, le comptage des signaux provenant d'encodeurs en quadrature. Il s'agit d'un composant temps réel uniquement, il est dépendant de divers facteurs comme la vitesse du CPU, etc, il est capable de compter des signaux de fréquences comprises entre 10kHz à peut être 50kHz. La figure ci-dessous représente le diagramme bloc d'une voie de comptage de codeur.

### Diagramme bloc du codeur



### 7.3.1 L'installer

```
halcmd: loadrt encoder [num_chan=<counters>]
```

`<counters>` est le nombre de compteurs de codeur à installer. Si `numchan` n'est pas spécifié, trois compteurs seront installés. Le nombre maximum de compteurs est de 8 (comme défini par `MAX_CHAN` dans `encoder.c`). Chaque compteur est indépendant, mais tous sont mis à jour par la même fonction(s) au même instant. Dans les descriptions qui suivent, `<chan>` est le nombre de compteurs spécifiques. La numérotation des compteurs commence à 0.

### 7.3.2 Le désinstaller

```
halcmd: unloadrt encoder
```

### 7.3.3 Pins

- *Encodeur <chan> counter-mode* (bit, I/O) (par défaut: FALSE) — Permet le mode compteur. Lorsque TRUE, le compteur compte chaque front montant de l'entrée phase-A, ignorant la valeur de la phase-B. Ceci est utile pour compter la sortie d'un capteur simple canal (pas de quadrature). Si FALSE, il compte en mode quadrature.
- *encoder.<chan>.counts* (s32, Out) — Position en comptage du codeur.



- *encoder.<chan>.counts-latched* (s32, Out) — Non utilisé à ce moment.
- *encoder.<chan>.index-enable* (bit, I/O) — Si TRUE, *counts* et *position* sont remis à zéro au prochain front montant de la phase Z. En même temps, *index-enable* est remis à zéro pour indiquer que le front montant est survenu. La broche *index-enable* est bi-directionnelle. Si *index-enable* est FALSE, la phase Z du codeur sera ignorée et le compteur comptera normalement. Le pilote du codeur ne doit jamais mettre *index-enable* TRUE. Cependant, d'autres composants peuvent le faire.
- *encoder.<chan>.latch-falling* (bit, In) (par défaut: TRUE) — Non utilisé à ce moment.
- *encoder.<chan>.latch-input* (bit, In) (par défaut: TRUE) — Non utilisé à ce moment.
- *encoder.<chan>.latch-rising* (bit, In) — Non utilisé à ce moment.
- *encoder.<chan>.min-speed-estimate* (Float, In) — Effectue une estimation de la vitesse minimale réelle, à partir de laquelle, la vitesse sera estimée comme non nulle et la position interpolées, comme étant interpolée. Les unités de vitesse *min-speed-estimate* sont les mêmes que les unités de *velocity*. Le facteur d'échelle, en compte par unité de longueur. Régler ce paramètre trop bas, fera prendre beaucoup de temps pour que la vitesse arrive à 0 après que les impulsions du codeur aient cessé d'arriver.
- *encoder.<chan>.phase-A* (bit, In) — Signal de la phase A du codeur en quadrature.
- *encoder.<chan>.phase-B* (bit, In) — Signal de la phase B du codeur en quadrature.
- *encoder.<chan>.phase-Z* (bit, In) — Signal de la phase Z (impulsion d'index) du codeur en quadrature.
- *encoder.<chan>.position* (float, Out) - Position en unités mises à l'échelle (voir *position* échelle).
- *encoder.<chan>.position-interpolated* (float, Out) - Position en unités mises à l'échelle, interpolées entre les comptes du codeur. *position-interpolated* tente d'interpoler entre les comptes du codeur, basée sur la mesure de vitesse la plus récente. Valable uniquement lorsque la vitesse est approximativement constante et supérieure à *min-speed-estimate*. Ne pas utiliser pour le contrôle de position, puisque sa valeur est incorrecte en basse vitesse, lors des inversions de direction et pendant les changements de vitesse. Toutefois, il permet à un codeur à PPR faible (y compris les codeur à une impulsion par tour) d'être utilisé pour du filetage sur tour et peut aussi avoir d'autres usages.
- *encoder.<chan>.position-latched* (float, Out) — Non utilisé à ce moment.
- *encoder.<chan>.position-scale* (float, I/O) — Le facteur d'échelle, en comptes par unité de longueur. Par exemple, si *position-scale* est à 500, alors à 1000 comptes codeur, la position sera donnée à 2,0 unités.
- *encoder.<chan>.rawcounts* (s32, In) — Le compte brut, tel que déterminé par *\_update-counters*. Cette valeur est mise à jour plus fréquemment que compte et position. Il n'est également pas affecté par le reset ou l'impulsion d'index.
- *encoder.<chan>.reset* (bit, In) — Si TRUE, force *counts* et *position* immédiatement à zéro.
- *encoder.<chan>.velocity* (float, Out) — Vitesse en unités mises à l'échelle par secondes. *encoder* utilise un algorithme qui réduit considérablement la quantification du bruit comparé à simplement différencier la sortie *position*. Lorsque la magnitude de la vitesse réelle est inférieure à *min-speed-estimate*, la sortie *velocity* est à 0.
- *encoder.<chan>.x4-mode* (bit, I/O) (par défaut: TRUE) — Permet le mode x4. Lorsqu'il est TRUE, le compteur compte chaque front de l'onde en quadrature (quatre compte par cycle complet). Si FALSE, il ne compte qu'une seule fois par cycle complet. En mode compteur, ce paramètre est ignoré. Le mode 1x est utile pour certaines manivelles électroniques.

### 7.3.4 Paramètres

- *encoder.<chan>.capture-position.time* (s32, RO)
- *encoder.<chan>.capture-position.tmax* (s32, RW)
- *encoder.<chan>.update-counters.time* (s32, RO)
- *encoder.<chan>.update-counter.tmax* (s32, RW)

### 7.3.5 Fonctions

Le composant exporte deux fonctions. Chaque fonction agit sur tous les compteurs de codeur, lancer différents compteurs de codeur dans différents threads n'est pas supporté.

- (funct) *encoder.update-counters* — Fonction haute vitesse de comptage d'impulsions (non flottant).
- (funct) *encoder.capture-position* — Fonction basse vitesse d'actualisation des latches et mise à l'échelle de la position.

## 7.4 PID

Ce composant fournit une boucle de contrôle Proportionnelle/Intégrale/Dérivée. C'est un composant temps réel uniquement. Par souci de simplicité, cette discussion suppose que nous parlons de boucles de position, mais ce composant peut aussi être utilisé pour implémenter d'autres boucles de rétroaction telles que vitesse, hauteur de torche, température, etc. La figure ci-dessous est le schéma fonctionnel d'une simple boucle PID.

**Diagramme bloc d'une boucle PID**



### 7.4.1 L'installer

```
halcmd: loadrt pid [num_chan=<loops>] [debug=1]
```

*<loops>* est le nombre de boucles PID à installer. Si *numchan* n'est pas spécifié, une seule boucle sera installée. Le nombre maximum de boucles est de 16 (comme définit par *MAX\_CHAN* dans *pid.c*). Chaque boucle est complètement indépendante. Dans les descriptions qui suivent, *<loopnum>* est le nombre de boucles spécifiques. La numérotation des boucle PID commence à 0.

Si *debug=1* est spécifié, le composant exporte quelques paramètres destinés au débogage et aux réglages. Par défaut, ces paramètres ne sont pas exportés, pour économiser la mémoire partagée et éviter d'encombrer la liste des paramètres.

## 7.4.2 Le désinstaller

```
halcmd: unloadrt pid
```

## 7.4.3 Pins

Les trois principales pins sont:

- (float) *pid.<loopnum>.command* — La position désirée (consigne), telle que commandée par un autre composant système.
- (float) *pid.<loopnum>.feedback* — La position actuelle (mesure), telle que mesurée par un organe de rétroaction comme un codeur de position.
- (float) *pid.<loopnum>.output* — Une commande de vitesse qui tend à aller de la position actuelle à la position désirée.

Pour une boucle de position, *command* et *feedback* sont en unités de longueur. Pour un axe linéaire, cela pourrait être des pouces, mm, mètres, ou tout autre unité pertinente. De même pour un axe angulaire, ils pourraient être des degrés, radians, etc. Les unités sur la pin *output* représentent l'écart nécessaire pour que la rétroaction coïncide avec la commande. Pour une boucle de position, *output* est une vitesse exprimée en pouces/seconde, mm/seconde, degrés/seconde, etc. Les unités de temps sont toujours des secondes et les unités de vitesses restent cohérentes avec les unités de longueur. Si la commande et la rétroaction sont en mètres, la sortie sera en mètres par seconde.

Chaque boucle PID a deux autres pins qui sont utilisées pour surveiller ou contrôler le fonctionnement général du composant.

- (float) *pid.<loopnum>.error* — Egal à *.command* moins *.feedback*. (consigne - mesure)
- (bit) *pid.<loopnum>.enable* — Un bit qui active la boucle. Si *.enable* est faux, tous les intégrateurs sont remis à zéro et les sorties sont forcées à zéro. Si *.enable* est vrai, la boucle opère normalement.

Pins utilisé pour signaler la saturation. La saturation se produit lorsque la sortie de le bloc PID est à son maximum ou limiter au minimum.

- (bit) *pid.<loopnum>.saturated* — True lorsque la sortie est saturée.
- (float) *pid.<loopnum>.saturated\_s* — Le temps de la sortie a été saturé.
- (s32) *pid.<loopnum>.saturated\_count* — Le temps de la sortie a été saturé.

## 7.4.4 Paramètres

Le gain PID, les limites et autres caractéristiques *accordables* de la boucle sont implémentés comme des paramètres.

- (float) *pid.<loopnum>.Pgain* — Gain de la composante proportionnelle.
- (float) *pid.<loopnum>.Igain* — Gain de la composante intégrale.
- (float) *pid.<loopnum>.Dgain* — Gain de la composante dérivée.
- (float) *pid.<loopnum>.bias* — Constante du décalage de sortie.
- (float) *pid.<loopnum>.FF0* — Correcteur prédictif d'ordre zéro (retour vitesse) sortie proportionnelle à la commande (position).
- (float) *pid.<loopnum>.FF1* — Correcteur prédictif de premier ordre (retour vitesse) sortie proportionnelle à la dérivée de la commande (vitesse).
- (float) *pid.<loopnum>.FF2* — Correcteur prédictif de second ordre (retour vitesse) sortie proportionnelle à la dérivée seconde de la commande (accélération).<sup>1</sup>
- (float) *pid.<loopnum>.deadband* — Définit la bande morte tolérable.
- (float) *pid.<loopnum>.maxerror* — Limite d'erreur.
- (float) *pid.<loopnum>.maxerrorI* — Limite d'erreur intégrale.
- (float) *pid.<loopnum>.maxerrorD* — Limite d'erreur dérivée.
- (float) *pid.<loopnum>.maxcmdD* — Limite de la commande dérivée.
- (float) *pid.<loopnum>.maxcmdDD* — Limite de la commande dérivée seconde.

1. FF2 n'est actuellement pas implémenté, mais il pourrait l'être. Considérez cette note comme un "FIXME" dans le code.

– (float) *pid.<loopnum>.maxoutput* — Limite de la valeur de sortie.

Toutes les limites *max???*, sont implémentées de sorte que si la valeur de ce paramètre est à zéro, il n’y a pas de limite.

Si *debug=1* est spécifié quand le composant est installé, quatre paramètres supplémentaires seront exportés:

– (float) *pid.<loopnum>.errorI* — Intégrale de l’erreur.

– (float) *pid.<loopnum>.errorD* — Dérivée de l’erreur.

– (float) *pid.<loopnum>.commandD* — Dérivée de la commande.

– (float) *pid.<loopnum>.commandDD* — Dérivée seconde de la commande.

### 7.4.5 Fonctions

Le composant exporte une fonction pour chaque boucle PID. Cette fonction exécute tous les calculs nécessaires à la boucle. Puisque chaque boucle a sa propre fonction, les différentes boucles peuvent être incluses dans les différents threads et exécutées à différents rythmes.

– (func) *pid.<loopnum>.do\_pid\_calcs* — Exécute tous les calculs d’une seule boucle PID.

Si vous voulez comprendre exactement l’algorithme utilisé pour calculer la sortie d’une boucle PID, référez vous à la figure [PID](#), les commentaires au début du source *linuxcnc/src/hal/components/pid.c* et bien sûr, au code lui même. Les calculs de boucle sont dans la fonction C *calc\_pid()*.

## 7.5 Codeur simulé

Le codeur simulé est exactement la même chose qu’un codeur. Il produit des impulsions en quadrature avec une impulsion d’index, à une vitesse contrôlée par une pin de HAL. Surtout utile pour les essais.

### 7.5.1 L’installer

```
halcmd: loadrt sim-encoder num_chan=<number>
```

*<number>* est le nombre de canaux à simuler. Si rien n’est spécifié, un seul canal sera installé. Le nombre maximum de canaux est de 8 (comme défini par *MAX\_CHAN* dans *sim\_encoder.c*).

### 7.5.2 Le désinstaller

```
halcmd: unloadrt sim-encoder
```

### 7.5.3 Pins

– (float) *sim-encoder.<chan-num>.speed* — La vitesse commandée pour l’arbre simulé.

– (bit) *sim-encoder.<chan-num>.phase-A* — Sortie en quadrature.

– (bit) *sim-encoder.<chan-num>.phase-B* — Sortie en quadrature.

– (bit) *sim-encoder.<chan-num>.phase-Z* — Sortie de l’impulsion d’index.

Quand *.speed* est positive, *.phase-A* mène *.phase-B*.

### 7.5.4 Paramètres

- (*u32*) *sim-encoder.<chan-num>.ppr* — Impulsions par tour d'arbre.
- (*float*) *sim-encoder.<chan-num>.scale* — Facteur d'échelle pour *speed*. Par défaut est de 1.0, ce qui signifie que *speed* est en tours par seconde. Passer l'échelle à 60 pour des tours par minute, la passer à 360 pour des degrés par seconde, à 6.283185 pour des radians par seconde, etc.

Noter que les impulsions par tour ne sont pas identiques aux valeurs de comptage par tour (counts). Une impulsion est un cycle complet de quadrature. La plupart des codeurs comptent quatre fois pendant un cycle complet.

### 7.5.5 Fonctions

Le composant exporte deux fonctions. Chaque fonction affecte tous les codeurs simulés.

- (*funct*) *sim-encoder.make-pulses* — Fonction haute vitesse de génération d'impulsions en quadrature (non flottant).
- (*funct*) *sim-encoder.update-speed* — Fonction basse vitesse de lecture de *speed*, de mise à l'échelle et d'activation de *make-pulses*.

## 7.6 Anti-rebond

L'anti-rebond est un composant temps réel capable de filtrer les rebonds créés par les contacts mécaniques. Il est également très utile dans d'autres applications, où des impulsions très courtes doivent être supprimées.

### 7.6.1 L'installer

```
halcmd: loadrt debounce cfg=<config-string>
```

*<config-string>* est une série d'entiers décimaux séparés par des espaces. Chaque chiffre installe un groupe de filtres anti-rebond identiques, le chiffre détermine le nombre de filtres dans le groupe. Par exemple:

```
halcmd: loadrt debounce cfg=1,4,2
```

va installer trois groupes de filtres. Le groupe 0 contient un filtre, le groupe 1 en contient quatre et le groupe 2 en contient deux. La valeur par défaut de *<config-string>* est 1 qui installe un seul groupe contenant un seul filtre. Le nombre maximum de groupes est de 8 (comme définit par MAX\_GROUPS dans *debounce.c*). Le nombre maximum de filtres dans un groupe est limité seulement par l'espace de la mémoire partagée. Chaque groupe est complètement indépendant. Tous les filtres dans un même groupe sont identiques et ils sont tous mis à jour par la même fonction, au même instant. Dans les descriptions qui suivent, *<G>* est le numéro du groupe et *<F>* est le numéro du filtre dans le groupe. Le premier filtre est le filtre 0 dans le groupe 0.

### 7.6.2 Le désinstaller

```
halcmd: unloadrt debounce
```

### 7.6.3 Pins

Chaque filtre individuel a deux pins.

- (*bit*) *debounce.<G>.<F>.in* — Entrée du filtre *<F>* du groupe *<G>*.
- (*bit*) *debounce.<G>.<F>.out* — Sortie du filtre *<F>* du groupe *<G>*.

## 7.6.4 Paramètres

Chaque groupe de filtre a un paramètre.<sup>2</sup>

– (*s32*) *debounce.<G>.delay* — Délai de filtrage pour tous les filtres du groupe *<G>*.

Le délai du filtre est dans l'unité de la période du thread. Le délai minimum est de zéro. La sortie d'un filtre avec un délai de zéro, suit exactement son entrée, il ne filtre rien. Plus le délai augmente, plus larges seront les impulsions rejetées. Si le délai est de 4, toutes les impulsions égales ou inférieures à quatre périodes du thread, seront rejetées.

## 7.6.5 Fonctions

Chaque groupe de filtres exporte une fonction qui met à jour tous les filtres de ce groupe *simultanément*. Différents groupes de filtres peuvent être mis à jour dans différents threads et à différentes périodes.

– (*funct*) *debounce.<G>* — Met à jour tous les filtres du groupe *<G>*.

## 7.7 Siggen

Siggen est un composant temps réel qui génère des signaux carrés, triangulaires et sinusoïdaux. Il est principalement utilisé pour les essais.

### 7.7.1 L'installer

```
halcmd: loadrt siggen [num_chan=<chans>]
```

*<chans>* est le nombre de générateurs de signaux à installer. Si *numchan* n'est pas spécifié, un seul générateur de signaux sera installé. Le nombre maximum de générateurs est de 16 (comme définit par *MAX\_CHAN* dans *siggen.c*). Chaque générateur est complètement indépendant. Dans les descriptions qui suivent, *<chan>* est le numéro d'un générateur spécifique. Les numéros de générateur commencent à 0.

### 7.7.2 Le désinstaller

```
halcmd: unloadrt siggen
```

### 7.7.3 Pins

Chaque générateur a cinq pins de sortie.

- (*float*) *siggen.<chan>.sine* — Sortie de l'onde sinusoïdale.
- (*float*) *siggen.<chan>.cosine* — Sortie de l'onde cosinusoidale.
- (*float*) *siggen.<chan>.sawtooth* — Sortie de l'onde en dents de scie.
- (*float*) *siggen.<chan>.triangle* — Sortie de l'onde triangulaire.
- (*float*) *siggen.<chan>.square* — Sortie de l'onde carrée.

Les cinq sorties ont les mêmes fréquence, amplitude et offset.

Trois pins de contrôle s'ajoutent aux pins de sortie:

- (*float*) *siggen.<chan>.frequency* — Réglage de la fréquence en Hertz, par défaut la valeur est de 1 Hz.
- (*float*) *siggen.<chan>.amplitude* — Réglage de l'amplitude de pic des signaux de sortie, par défaut, est à 1.
- (*float*) *siggen.<chan>.offset* — Réglage de la composante continue des signaux de sortie, par défaut, est à 0.

Par exemple, si *siggen.0.amplitude* est à 1.0 et *siggen.0.offset* est à 0.0, les sorties oscilleront entre -1.0 et +1.0. Si *siggen.0.amplitude* est à 2.5 et *siggen.0.offset* est à 10.0, les sorties oscilleront entre 7.5 et 12.5.

2. Chaque filtre individuel a également une variable d'état interne. C'est un switch du compilateur qui peut exporter cette variable comme un paramètre. Ceci est prévu pour des essais et devrait juste être un gaspillage de mémoire partagée dans des circonstances normales.

### 7.7.4 Paramètres

Aucun.<sup>3</sup>

### 7.7.5 Fonctions

– *(funct) siggen.<chan>.update* — Calcule les nouvelles valeurs pour les cinq sorties.

## 7.8 lut5

Le composant lut5 est un composant de logique à 5 entrées basé sur une table de vérité.

– *lut5* ne requiert pas un thread à virgule flottante.

#### Installation

```
loadrt lut5 [count=N|names=name1[,name2...]]
addf lut5.N servo-thread | base-thread
setp lut5.N.function 0xN
```

**Calcul de la valeur de la fonction** Pour calculer la valeur hexadécimale de la fonction, démarrer par le haut et entrer un 1 où un 0 pour indiquer si cette colonne devra être vraie où fausse. Ensuite écrire les valeurs en dessous, d’abord dans la colonne de sortie en commençant par le haut puis en écrivant les valeurs correspondantes de la droite vers la gauche. Le nombre binaire sera celui contenu dans la colonne de sortie. Utiliser une calculette comme celle fournie sous Ubuntu, entrer ce nombre binaire et le convertir en hexadécimal pour obtenir la valeur pour la fonction.

TABLE 7.1: Table de vérité

Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Output
0	0	0	0	0	
0	0	0	0	1	
0	0	0	1	0	
0	0	0	1	1	
0	0	1	0	0	
0	0	1	0	1	
0	0	1	1	0	
0	0	1	1	1	
0	1	0	0	0	
0	1	0	0	1	
0	1	0	1	0	
0	1	0	1	1	
0	1	1	0	0	
0	1	1	0	1	
0	1	1	1	0	
0	1	1	1	1	
1	0	0	0	0	
1	0	0	0	1	
1	0	0	1	0	
1	0	0	1	1	
1	0	1	0	0	
1	0	1	0	1	
1	0	1	1	0	
1	0	1	1	1	

3. Dans les versions antérieures à la 2.1, fréquence, amplitude et offset étaient des paramètres. Ils ont été modifiés en pins pour permettre le contrôle par d’autres composants.

TABLE 7.1: (continued)

Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Output
1	1	0	0	0	
1	1	0	0	1	
1	1	0	1	0	
1	1	0	1	1	
1	1	1	0	0	
1	1	1	0	1	
1	1	1	1	0	
1	1	1	1	1	

**Un exemple de fonction.** Dans la table suivante nous avons sélectionné l'état de sortie pour chaque ligne que nous souhaitons vraie.

TABLE 7.2: Table de vérité

Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Output
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	0
0	0	0	1	1	1

En regardant la colonne de sortie de notre exemple, nous voulons que la sortie soit active quand le bit 0 OU le bit 0 ET le bit 1 soient actifs et rien d'autre. Le nombre binaire est *b1010* (rotation de la sortie de 90° en sens horaire). Entrer ce nombre dans une calculatrice, le convertir en hexadécimal et le nombre demandé pour cette fonction est *0xa*. Le préfixe *0x* étant celui des nombres hexadécimaux. :lang: fr :toc:



## Chapitre 8

# Exemples pour HAL

Tous ces exemples s'appuient sur une configuration créée par Stepconf, elle a deux threads, *base-thread* et *servo-thread*.

L'assistant de configuration Stepconf aura créé le fichier vide *custom.hal* et le fichier *custom\_postgui.hal*.

Le fichier *custom.hal* sera chargé après le fichier de configuration de HAL, le fichier *custom\_postgui.hal* sera chargé après que l'interface graphique ne le soit.

### 8.1 Changement d'outil manuel

Dans cet exemple, il est supposé que la configuration a été réalisée et qu'il faut lui ajouter la fenêtre de changement d'outil de HAL. Le composant de changement d'outil manuel de HAL est surtout intéressant si les outils sont mesurables avec précision en longueur et que les offsets sont stockés dans la table d'outils. Si il est nécessaire de faire un *Toucher* pour chaque outil, il sera préférable de scinder le programme G-code en plusieurs tronçons. Pour utiliser la fenêtre de changement manuel d'outil de HAL, il faut d'abord charger le composant *hal\_manualtoolchange* puis envoyer l'ordre *iocontrol tool change* vers le *change* de *hal\_manualtoolchange* ainsi qu'envoyer le *changed* de *hal\_manualtoolchange* en retour sur le *iocontrol tool changed*.

Voici un exemple *avec* utilisation du composant de HAL, pour clarifier tout cela:

```
loadusr -W hal_manualtoolchange
net tool-change iocontrol.0.tool-change => hal_manualtoolchange.change
net tool-changed iocontrol.0.tool-changed <= hal_manualtoolchange.changed
net tool-number iocontrol.0.tool-prep-number => hal_manualtoolchange.number
net tool-prepare-loopback iocontrol.0.tool-prepare => iocontrol.0.tool-prepared
```

Et voici un exemple *sans* le composant de changement manuel:

```
net tool-number <= iocontrol.0.tool-prep-number
net tool-change-loopback iocontrol.0.tool.-change => iocontrol.0.tool-changed
net tool-prepare-loopback iocontrol.0.tool-prepare => iocontrol.0.tool-prepared
```

### 8.2 Calcul de vitesse

Cet exemple utilise *ddt*, *mult2* et *abs* pour calculer la vitesse de déplacement sur un axe. Pour plus de détails, voir la section [sur les composants de HAL](#).

En premier il convient de vérifier si la configuration contient déjà des composants temps réel. Il est possible de le vérifier en ouvrant la fenêtre de Halshow et en cherchant les composants dans la section des pins. Si il y en a déjà en activité, il faudra augmenter leur nombre et ajuster l'instance de ces composants à la valeur correcte. Ajouter le code suivant dans le fichier *custom.hal*.

Charger les composants temps réel.

```
loadrt ddt count=1
loadrt mult2 count=1
loadrt abs count=1
```

Ajouter les fonctions au thread pour qu'elles soient rafraîchies.

```
addf ddt.0 servo-thread
addf mult2.0 servo-thread
addf abs.0 servo-thread
```

Faire les connections.

```
setp mult2.in1 60
net xpos-cmd ddt.0.in
net X-IPS mult2.0.in0 <= ddt.0.out
net X-ABS abs.0.in <= mult2.0.out
net X-IPM abs.0.out
```

Dans la dernière section, nous avons fixé le *mult2.0.in1* à 60 pour convertir les unités par seconde en unités par minute (dans cet exemple, des pouces/mn), nous l'obtenons sur la sortie *ddt.0.out*.

La commande *xpos-cmd* envoie la position commandée à l'entrée *ddt.0.in*. Le *ddt* calcule la dérivée de la variation du signal sur son entrée.

La sortie *ddt2.0.out* est multipliée par 60 pour obtenir des unités par minute.

La sortie *mult2.0.out* est envoyée au composant *abs* pour obtenir la valeur absolue.

La figure suivante montre le résultat quand l'axe X se déplace à 15 unités/mn dans la direction négative. Noter que la valeur absolue peut être prise sur la pin *abs.0.out* ou le signal X-IPM.

**Exemple avec la vitesse**



### 8.3 Amortissement d'un signal

Cette exemple montre comment les composants de HAL *lowpass*, *limit2* ou *limit3* peuvent être utilisés pour amortir de brusques changements d'un signal.

Nous sommes sur un tour dont la broche est pilotée par un servomoteur. Si nous envoyions directement la consigne de vitesse de broche sur le servo, celui-ci chercherait, à partir de la vitesse courante, à atteindre la vitesse commandée le plus vite possible. Cette situation est problématique et peut détériorer le matériel. Pour amortir ce changement de vitesse, nous pouvons faire passer la sortie *motion.spindle-speed-out* à travers un limiteur avant d'aller au PID, de sorte que la valeur de commande du PID soit amortie.

Les trois composants intégrés pour amortir le signal seront:

#### **limit2**

- Limite le signal de sortie pour qu'il soit entre min et max.
- Limite sa vitesse de montée à moins de MaxV par seconde. (dérivée première)

#### **limit3**

- Limite le signal de sortie pour qu'il soit entre min et max.
- Limite sa vitesse de montée à moins de MaxV par seconde. (dérivée première)
- Limite sa vitesse de montée à moins de MaxV par seconde<sup>2</sup>. (dérivée seconde)

#### **lowpass**

- Filtre passe-bas.

Pour plus de détails voir [les composants de HAL](#) ou les man pages des composants concernés.

Placer le code suivant dans un fichier appelé *softstart.hal*.

```
loadrt threads period1=1000000 name1=thread
loadrt siggen
loadrt lowpass
loadrt limit2
loadrt limit3
net square siggen.0.square => lowpass.0.in limit2.0.in limit3.0.in
net lowpass <= lowpass.0.out
net limit2 <= limit2.0.out
net limit3 <= limit3.0.out
setp siggen.0.frequency .1
setp lowpass.0.gain .01
setp limit2.0.maxv 2
setp limit3.0.maxv 2
setp limit3.0.maxa 10
addf siggen.0.update thread
addf lowpass.0 thread
addf limit2.0 thread
addf limit3.0 thread
start
loadusr halscope
```

Ouvrir un terminal et lancer le fichier avec la commande suivante:

```
halrun -I softstart.hal
```

Pour démarrer l'oscilloscope de HAL pour la première fois, cliquer *OK* pour accepter le thread par défaut.

Ensuite, il faut ajouter les signaux à suivre aux canaux du scope. Cliquer sur le canal *1* puis sélectionner *square* depuis l'onglet *Signaux*. Répéter pour les canaux suivants en ajoutant *lowpass*, *limit2* et *limit3*.

Ensuite, pour régler le signal du déclencheur cliquer sur le bouton *Source* et sélectionner *square*. Le bouton devrait changer pour *Source Canal 1*.

Puis, cliquer sur *Simple* dans le groupe *Mode Run*. L'oscillo devrait faire un balayage puis, afficher les traces.

Pour séparer les signaux et mieux les visualiser, cliquer sur un canal et utiliser le curseur de position verticale pour positionner les traces.



FIGURE 8.1 – Amortissement d'un signal carré

Pour voir l'effet d'un changement du point de réglage des valeurs des composants, il est possible de passer des commandes depuis le terminal. Par exemple, pour voir différentes valeurs de gain pour le passe-bas, taper la commande suivante, puis essayer différentes valeurs:

```
setp lowpass.0.gain .01
```

Après un changement de réglage, relancer Halscope pour visualiser l'effet.

Pour terminer, taper *exit* dans le terminal pour fermer halrun et halscope. Ne pas refermer le terminal avec halrun en marche, la mémoire ne serait pas vidée proprement, ce qui pourrait empêcher LinuxCNC de se charger.

Pour tout savoir sur Halscope et Halrun [voir le tutoriel de HAL](#).

## 8.4 HAL en autonome

Dans certains cas il peut être utile de lancer un écran GladeVCP avec juste HAL. Par exemple, nous avons un moteur pas à pas à piloter et tout ce qu'il nous faut pour notre application est une simple interface avec *Marche/Arrêt* plutôt que charger une application de CNC complète.

Dans l'exemple qui suit, nous allons créer ce simple panneau GladeVCP.

### Syntaxe de base

```
# charge l'interface graphique winder.glade et la nome winder
loadusr -Wn winder gladevcp -c winder -u handler.py winder.glade

# charge les composants temps réel
loadrt threads name1=fast period1=50000 fp1=0 name2=slow period2=1000000
loadrt stepgen step_type=0 ctrl_type=v
loadrt hal_parport cfg="0x378 out"

# ajoute les fonctions aux threads
addf stepgen.make-pulses fast
addf stepgen.update-freq slow
addf stepgen.capture-position slow
addf parport.0.read fast
addf parport.0.write fast

# effectue les connections de hal
net winder-step parport.0.pin-02-out <= stepgen.0.step
net winder-dir parport.0.pin-03-out <= stepgen.0.dir
net run-stepgen stepgen.0.enable <= winder.start_button

# démarre les threads
start

# commenter la ligne suivante pendant les essais et utiliser le mode interactif
# pour voir les pins etc.
# option halrun -I -f start.hal

# attends que la GUI gladevcp nommée winder soit terminée
waitusr winder

# arrête tous les threads
stop

# décharge tous les composants de HAL avant de quitter
unloadrt all
```

## Chapitre 9

# L'interface Halui

### 9.1 Introduction

Halui est une interface utilisateur pour LinuxCNC s'appuyant sur HAL, elle connecte les pins de HAL à des commandes NML. La plupart des fonctionnalités (boutons, indicateurs etc.) utilisées par les interfaces graphiques traditionnelles (mini, Axis, etc.), sont fournies par des pins de HAL dans Halui.

La façon la plus facile pour utiliser halui est de modifier votre dossier d'ini pour inclure

```
HALUI = halui
```

dans la section [HAL].

Une solution alternative pour l'invoquer (surtout si vous générez la config avec stepconf) est d'inclure

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

dans votre fichier custom.hal.

### 9.2 Nomenclature des pins d'Halui

#### Abandon

(abort)

- halui.abort (bit, in) - pin de requête d'abandon (efface les erreurs)

#### Axes

(axis)

- halui.axis.n.pos-commanded (float, out) - Position de l'axe commandée, en coordonnées machine
- halui.axis.n.pos-feedback (float, out) - Position de l'axe lue, en coordonnées machine
- halui.axis.n.pos-relative (float, out) - Position de l'axe, en coordonnées relatives

#### Arrêt d'urgence

(E-Stop)

- halui.estop.activate (bit, in) - pin de requête d'arrêt d'urgence (E-Stop)
- halui.estop.is-activated (bit, out) - indique si l'arrêt d'urgence est actif
- halui.estop.reset (bit, in) - pin de requête de relâchement de l'arrêt d'urgence (E-Stop reset)

#### Correcteur de vitesse d'avance

(Feed override)

---

- `halui.feed-override.count-enable` (bit, in) - doit être vraie pour que *counts* ou *direct-value* soient opérationnels.
- `halui.feed-override.counts` (s32, in) -  $counts * scale = FO$  pourcent. Peut être utilisé avec un codeur ou *direct-value*.
- `halui.feed-override.decrease` (bit, in) - pin pour diminuer la correction ( $=scale$ )
- `halui.feed-override.increase` (bit, in) - pin pour augmenter la correction ( $+scale$ )
- `halui.feed-override.direct-value` (bit, in) - fausse lors de l'utilisation un codeur pour changer counts, vraie pour ajuster counts directement. La pin *count-enable* doit être vraie.
- `halui.feed-override.scale` (float, in) - pin pour positionner l'échelle pour accroître ou décroître la correction de vitesse d'avance.
- `halui.feed-override.value` (float, out) - Valeur de la correction courante de vitesse d'avance

### Arrosage par gouttelettes

(Mist)

- `halui.mist.is-on` (bit, out) - indique si l'arrosage par gouttelettes est actif
- `halui.mist.off` (bit, in) - pin de requête d'arrêt de l'arrosage par gouttelettes
- `halui.mist.on` (bit, in) - pin de requête de l'arrosage par gouttelettes

### Arrosage fluide

(Flood)

- `halui.flood.is-on` (bit, out) - indique si l'arrosage fluide est actif
- `halui.flood.off` (bit, in) - pin de requête d'arrêt d'arrosage fluide
- `halui.flood.on` (bit, in) - pin de requête d'arrosage fluide

### Prise d'origine machine de tous les axes

(Homing)

- `halui.home-all` (bit, in) - pin de requête de prise d'origine machine de tous les axes. Cette pin sera présente seulement si `HOME_SEQUENCE` est fixée dans le fichier ini.

### Jog

(Manivelle) `<n>` est un nombre compris entre 0 et 8, ou `<selected>`.

- `halui.jog-deadband` (float, in) - bande morte pour le jogging analogique (les petites vitesses de jog sont sans effet)
- `halui.jog.speed` (float, in) - positionne la vitesse de jog
- `halui.jog.<n>.analog` (float, in) - entrée analogique de vitesse de jog (utilisé avec les joysticks ou autres matériels analogiques)
- `halui.jog.<n>.minus` (bit, in) - jog en direction négative
- `halui.jog.<n>.plus` (bit, in) - jog en direction positive
- `halui.jog.<selected>.minus` (bit, in) - jog l'axe `<selected>` en direction négative et à la vitesse de `halui.jog.speed velocity`
- `halui.jog.<selected>.plus` (bit, in) - jog l'axe `<selected>` en direction positive et à la vitesse de `halui.jog.speed velocity`

### Articulations

(Joints) `<n>` est un nombre compris entre 0 et 8, ou `<selected>`.

- `halui.joint.<n>.has-fault` (bit, out) - pin de status indiquant que l'articulation est en défaut
- `halui.joint.<n>.home` (bit, in) - pin pour la prise d'origine d'une articulation spécifique
- `halui.joint.<n>.is-homed` (bit, out) - pin de status indiquant que l'articulation est référencée
- `halui.joint.<n>.is-selected` (bit, out) - pin indiquant que l'articulation est `<selected>` - interne à halui
- `halui.joint.<n>.on-hard-max-limit` (bit, out) - pin de status indiquant que le joint est sur son fin de course de limite positive
- `halui.joint.<n>.on-hard-min-limit` (bit, out) - pin de status indiquant que le joint est sur son fin de course de limite négative
- `halui.joint.<n>.on-soft-max-limit` (bit, out) - pin de status indiquant que le joint est sur sa limite logicielle positive
- `halui.joint.<n>.on-soft-min-limit` (bit, out) - pin de status indiquant que le joint est sur sa limite logicielle négative



- `halui.joint.<n>.select (bit, in)` - select joint (0..8) - interne à halui
- `halui.joint.<n>.unhome (bit, in)` - unhomes this joint
- `halui.joint.selected (u32, out)` - selected joint (0..8) - interne à halui
- `halui.joint.selected.has-fault (bit, out)` - pin de status indiquant que le joint `<n>` est en défaut
- `halui.joint.selected.home (bit, in)` - pin pour la prise d'origine de l'articulation `<selected>`
- `halui.joint.selected.is-homed (bit, out)` - pin de status indiquant que le joint `<selected>` est référencé
- `halui.joint.selected.on-hard-max-limit (bit, out)` - pin de status indiquant que le joint `<selected>` est sur son fin de course de limite positive
- `halui.joint.selected.on-hard-min-limit (bit, out)` - pin de status indiquant que le joint `<selected>` est sur son fin de course de limite négative
- `halui.joint.selected.on-soft-max-limit (bit, out)` - pin de status indiquant que le joint `<selected>` est sur sa limite logique positive
- `halui.joint.selected.on-soft-min-limit (bit, out)` - pin de status indiquant que le joint `<selected>` est sur sa limite logique négative
- `halui.joint.selected.unhome (bit, in)` - pin for unhoming l'articulation `selected`.

### Graissage centralisé

(Lube)

- `halui.lube.is-on (bit, out)` - indique si le graissage est actif
- `halui.lube.off (bit, in)` - pin de requête d'arrêt du graissage
- `halui.lube.on (bit, in)` - pin de requête de graissage

### Machine

(Marche / Arrêt)

- `halui.machine.is-on (bit, out)` - indique que la machine est en marche
- `halui.machine.off (bit, in)` - pin de requête d'arrêt machine
- `halui.machine.on (bit, in)` - pin de requête de marche machine

### Vitesse maximum

La vitesse linéaire maximum peut être ajustée entre 0 et la valeur de la variable `MAX_VELOCITY` dans la section `[TRAJ]` du fichier ini.

- `halui.max-velocity.count-enable (bit, in)` - Doit être vraie pour que les *counts* ou *direct-value* soit opérationnels.
- `halui.max-velocity.counts (s32, in)` - `counts * scale = MV` pourcent. Utilisable avec un codeur ou *direct-value*.
- `halui.max-velocity.direct-value (bit, in)` - faux quand un codeur est utilisé pour modifier *counts*, vraie pour ajuster *counts* directement. La pin *count-enable* doit être vraie.
- `halui.max-velocity.decrease (bit, in)` - pin pour diminuer la vitesse max
- `halui.max-velocity.increase (bit, in)` - pin pour augmenter la vitesse max
- `halui.max-velocity.scale (float, in)` - Valeur appliquée sur le nombre de fronts montants des pins *increase* ou *decrease* en unités machine par seconde.
- `halui.max-velocity.value (float, out)` - Valeur de la vitesse linéaire maximum en unités machine par seconde.

### Données manuelles

Il arrive que l'utilisateur veuille ajouter des tâches plus complexes devant être effectuées par l'activation d'une pin de HAL. C'est possible en utilisant le schéma de commande en données manuelles (MDI) suivant:

- Une `MDI_COMMAND` est ajoutée dans la section `[HALUI]` du fichier ini, par exemple:

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

- Quand halui démarre il va lire/détecter le champ `MDI_COMMAND` dans le fichier ini et exporter les pins de type (bit) `halui.mdi-command-<nr>`, `<nr>` est un nombre compris entre 00 et le nombre de `MDI_COMMAND` trouvées dans le fichier ini, avec un maximum de 64 commandes.

- Quand la pin `halui.mdi-command-<nr>` est activée, `halui` va essayer d'envoyer au MDI la commande définie dans le fichier ini. Ca ne fonctionnera pas dans tous les modes de fonctionnement où se trouve LinuxCNC, par exemple, tant qu'il est en AUTO `halui` ne peut pas envoyer de commande MDI.

### Sélection d'une articulation

(Joint Selection)

- `halui.joint.select (u32, in)` - sélectionne l'articulation (0..7) - internal `halui`
- `halui.joint.selected (u32, out)` - articulation (0..7) sélectionnée - internal `halui`
- `halui.joint.x.select bit (bit, in)` - pins pour sélectionner une articulation - internal `halui`
- `halui.joint.x.is-selected bit (bit, out)` - pin de status indiquant une articulation sélectionnée - internal `halui`

### Mode de fonctionnement

(Mode)

- `halui.mode.auto (bit, in)` - pin de requête du mode auto
- `halui.mode.is_auto (bit, out)` - indique si le mode auto est actif
- `halui.mode.is-joint (bit, out)` - indique si le mode articulation par articulation est actif
- `halui.mode.is_manual (bit, out)` - indique si le mode manuel est actif
- `halui.mode.is_mdi (bit, out)` - indique si le mode données manuelles est actif
- `halui.mode.is-teleop (bit, out)` - indique que le mode jog coordonné est actif
- `halui.mode.joint (bit, in)` - pin de requête du mode jog articulation par articulation
- `halui.mode.manual (bit, in)` - pin de requête du mode manuel
- `halui.mode.mdi (bit, in)` - pin de requête du mode données manuelles
- `halui.mode.teleop (bit, in)` - pin de requête du mode jog coordonné

### Programme

(Program)

- `halui.program.block-delete.is-on (bit, out)` - status pin telling that block delete is on
- `halui.program.block-delete.off (bit, in)` - pin for requesting that block delete is off
- `halui.program.block-delete.on (bit, in)` - pin for requesting that block delete is on
- `halui.program.is-idle (bit, out)` - pin de status indiquant qu'aucun programme n'est lancé
- `halui.program.is-paused (bit, out)` - pin de status indiquant qu'un programme est en pause
- `halui.program.is-running (bit, out)` - pin de status indiquant qu'un programme est lancé
- `halui.program.optional-stop.is-on (bit, out)` - status pin telling that the optional stop is on
- `halui.program.optional-stop.off (bit, in)` - pin requesting that the optional stop is off
- `halui.program.optional-stop.on (bit, in)` - pin requesting that the optional stop is on
- `halui.program.pause (bit, in)` - pin pour passer un programme en pause
- `halui.program.resume (bit, in)` - pin pour lancer la reprise d'un programme
- `halui.program.run (bit, in)` - pin de lancement d'un programme
- `halui.program.step (bit, in)` - pin pour avancer d'une ligne de programme
- `halui.program.stop (bit, in)` - pin pour stopper un programme

### Correcteur de vitesse de broche

(Spindle Override)

- `halui.spindle-override.count-enable (bit, in)` - doit être vraie pour que *counts* ou *direct-value* soient opérationnels.
- `halui.spindle-override.counts (s32, in)` - comptage depuis un codeur, par exemple pour modifier la correction de vitesse de broche
- `halui.spindle-override.decrease (bit, in)` - pin pour diminuer la correction de vitesse de broche (*-=scale*)
- `halui.spindle-override.direct-value (bit, in)` - fausse en utilisant un codeur pour modifier *counts* directement. La pin *count-enable* doit être vraie.
- `halui.spindle-override.increase (bit, in)` - pin pour augmenter la correction de vitesse de broche (*+=scale*)

- `halui.spindle-override.scale` (float, in) - pin pour positionner l'échelle des corrections de vitesse de broche possibles
- `halui.spindle-override.value` (float, out) - Valeur courante de la correction de vitesse de broche

#### **Broche**

(Spindle)

- `halui.spindle.brake-is-on` (bit, out) - indique si le frein est actif
- `halui.spindle.brake-off` (bit, in) - pin de désactivation du frein de broche
- `halui.spindle.brake-on` (bit, in) - pin d'activation du frein de broche
- `halui.spindle.decrease` (bit, in) - Diminue la vitesse de broche
- `halui.spindle.forward` (bit, in) - Marche broche en sens horaire
- `halui.spindle.increase` (bit, in) - Augmente la vitesse de broche
- `halui.spindle.is-on` (bit, out) - indique la broche est en marche (les deux sens)
- `halui.spindle.reverse` (bit, in) - Marche broche en sens anti-horaire
- `halui.spindle.runs-backward` (bit, out) - indique la broche est en marche et en sens inverse
- `halui.spindle.runs-forward` (bit, out) - indique la broche est en marche et en marche avant
- `halui.spindle.start` (bit, in) - Marche de la broche
- `halui.spindle.stop` (bit, in) - Arrêt de la broche

#### **Outil**

(Tool)

- `halui.tool.length-offset` (float, out) - indique la correction de longueur d'outil appliquée
- `halui.tool.number` (u32, out) - indique l'outil courant sélectionné

## **9.3 Exemples de programme avec Halui**

Pour que ces exemples fonctionnent, il faut ajouter la ligne suivante dans la section [HAL] du fichier ini.

```
HALUI = halui
```

### **9.3.1 Démarrage à distance**

Pour connecter un bouton de démarrage à distance à LinuxCNC il faut utiliser la pin `halui.program.run` et la pin `halui.mode.auto`.

Il faut s'assurer qu'il est possible de démarrer en utilisant la pin `halui.mode.is-auto`. On peut faire cela avec un composant de HAL `and2`. La figure suivante montre comment faire.

Quand le bouton de commande à distance est pressé, il est connecté à `halui.mode.auto` et à l'entrée `and2.0.in0`. Si le mode auto est activé, la pin `halui.mode.is-auto` sera TRUE.

Si les deux entrées du composant `and2.0` sont TRUE, la sortie `and2.0.out` sera TRUE également et le programme sera démarré.



FIGURE 9.1 – Exemple de commande distante

Les commandes de Hal pour accomplir ces actions sont les suivantes:

```

net program-start-btn halui.mode.auto and2.0.in0 <= <la pin d'entrée>
net program-run-ok and2.0.in1 <= halui.mode.is-auto
net remote-program-run halui.program.run <= and2.0.out
  
```

Noter que sur la première ligne il y a deux pins en lecture, ce qui pourrait aussi se séparer en deux lignes comme ceci:

```

net program-start-btn halui.mode.auto <= <la pin d'entrée>
net program-start-btn and2.0.in0
  
```

### 9.3.2 Pause et Reprise

Cet exemple a été développé pour permettre à LinuxCNC de déplacer un axe rotatif selon un signal provenant d'une machine extérieure. La coordination entre les deux systèmes est assurée par deux composants de Halui:

- halui.program.is-paused
- halui.program.resume

Dans le fichier *custom.hal*, ajoutez les deux lignes suivantes qui seront connectées à vos entrées/sorties pour mettre le programme en pause ou pour le reprendre quand l'autre système veut qu'LinuxCNC soit relancé.

```

net ispaused halui.program.is paused => "la pin de sortie"
net resume halui.program.resume <= "la pin d'entrée"
  
```

Les pins d'entrée et de sortie, correspondent à celles qui sont câblées vers l'autre contrôleur. Elles peuvent être des broches du port parallèle ou toutes autres broches auxquelles nous avons accès.

Le fonctionnement est le suivant, quand un M0 est rencontré dans le programme G-code, *halui.program.is-paused* devient TRUE. Ce qui rend la broche de sortie également TRUE de sorte que l'autre contrôleur sait que LinuxCNC est arrêté.

Pour reprendre l'exécution du G-code, l'autre contrôleur devra rendre l'entrée TRUE. Ce qui relancera LinuxCNC jusqu'au prochain M0.

Difficultés de timing

- Le signal de reprise ne doit pas être plus long que le temps nécessaire pour exécuter le G-code.
- Le signal *Is Paused* ne doit plus être actif quand le signal suivant de reprise arrive.

Ces problèmes de timing pourraient être évités, en utilisant ClassicLadder pour activer le signal *is paused* avec une tempo et le désactiver en fin de tempo. La reprise pourrait également être fournie par un signal monostable très court.

## Chapitre 10

# comp: outil pour créer les modules HAL

### 10.1 Introduction

Écrire 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 de LinuxCNC, ou qu'il en soit extérieur.

Par exemple, cette portion des blocs *ddt*, codée en C, fait environ 80 lignes de code, alors que le composant équivalent est vraiment très court quand il est créé en utilisant le préprocesseur *comp*.

#### Exemple pour comp

```
component ddt "Calcule la dérivée de la fonction d'entrée";
pin in float in;
pin out float out;
variable float old;
function _;
license "GPLv2 or later";
;;
float tmp = in;
out = (tmp - old) / fperiod;
old = tmp;
```

### 10.2 Installation

Si une version pré-installée de LinuxCNC est utilisée, il sera nécessaire d'installer les paquets de développement en passant par Synaptic depuis le menu *Système → Administration → Gestionnaire de paquets Synaptic* ou en utilisant la commande suivante dans un terminal:

#### Installation des paquets de développement

```
sudo apt-get install linuxcnc-dev
```

### 10.3 Définitions

- *component* - 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 plusieurs instances ou aucune. 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)

## 10.4 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. Si *count* n'est pas spécifié, le paramètre *names* du module détermine combien d'instances nommées seront créées. Si ni *count*, ni *names* n'est spécifié, une simple instance numérotée sera créée.

## 10.5 Paramètres implicites

Le paramètre *period* est implicitement passé aux fonctions, c'est la durée, en nanosecondes, de la dernière période d'exécution du comp. Les fonctions qui utilisent des flottants peuvent aussi se référer à *fperiod*, qui est la durée en secondes, soit (*period*\*1e-9). Cela peut être utile pour les comps ayant besoin de l'information de timing.

## 10.6 Syntaxe

Un fichier *.comp* commence par un certain nombre de déclarations, puis par un délimiteur constitué de deux points virgule;; seuls sur leur propre ligne et enfin du 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 *MAJUSCULES* indiquent une variable texte, comme ci-dessous:

- *NAME* - Un identifiant C standard.
- *STARREDNAME* - Un identifiant C, précédé ou non d'une \*. Cette syntaxe est utilisée pour déclarer les variables qui sont des pointeurs. Noter qu'à cause de la grammaire, il ne doit pas y avoir d'espace entre \* et le nom de la variable.
- *HALNAME* - Un identifiant étendu. 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**, si le nom est "", 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 lors de 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:

+ . Tous les caractères "#" sont enlevés ainsi que tous les caractères ".", "" ou "-" *immédiatement devant eux*. . Dans un nom, tous les caractères "." et "-" sont remplacés par "". 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:

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

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

La chaîne de 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.

### 10.6.1 Fonctions HAL

- *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*.



## 10.6.2 Options

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 *count* 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) **obsolète** 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*). Dans les nouveaux *components*, *variable* doit être utilisé en remplacement.
- *option extra\_setup yes* - (défaut: no) Si spécifié, appelle la fonction définie par *EXTRA\_SETUP* pour chaque instance. Dans le cas de la *rtapi\_app\_main* automatiquement définie, *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.

## 10.6.3 Licence et auteur

- *LICENSE* - Spécifie la licence du module, pour la documentation et pour le module déclaré dans *MODULE\_LICENSE()*. Par exemple, pour spécifier que la licence des modules est la GPL v2 ou suivantes, `license "GPL";` // indique GPL v2 ou suivantes

Pour d'autres informations sur la signification du *MODULE\_LICENSE()* et les identificateurs de licence additionnels, voir `'<linux/module.h>'` ou la page `'rtapi_module_param(3)'` du manuel.

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

### 10.6.4 Stockage des données par instance

- *variable* `CTYPE STARREDNAME`;
- *variable* `CTYPE STARREDNAME[SIZE]`;
- *variable* `CTYPE STARREDNAME = DEFAULT`;
- *variable* `CTYPE STARREDNAME[SIZE] = DEFAULT`; Déclare la variable *par-instance* `STARREDNAME` 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.

Si une variable doit être de type pointeur, il ne doit y avoir aucun espace entre l'étoile "\*" et le nom de la variable. Néanmoins, la forme suivante est acceptable:

```
variable int *bonexemple;
```

Mais les formes suivantes ne sont pas acceptables:

```
variable int* mauvaexemple;
variable int * mauvaexemple;
```

### 10.6.5 Commentaires

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

## 10.7 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.

Les noms de variable et de fonction qui ne doivent pas être utilisés ou qui posent problème sont les suivants:

- Tous noms commençant par `_comp`.
- `comp_id`
- `fperiod`
- `rtapi_app_main`
- `rtapi_app_exit`
- `extra_setup`
- `extra_cleanup`

## 10.8 Conventions des macros

En se basant sur les déclarations des items de section, `comp` crée une structure C appelée `struct comp_state`. 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 `struct comp_state` et ces macros peuvent différer d'une version de `comp` à une autre.

- `FUNCTION(name)` - Cette macro s'utilise au début de la définition d'une fonction temps réel qui aura été précédemment déclarée avec `function NAME`. fonction 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* ou *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\_INSTS() {...}* - 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.

## 10.9 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.

### 10.10 Personnalité du composant

Si un composant n'importe combien de pins ou de paramètres avec un if condition ou [*maxsize* : *condsize*], 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*.

### 10.11 Compiler un fichier *.comp* dans l'arborescence

Placer le fichier *.comp* dans le répertoire *linuxcnc/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 *linuxcnc/src/hal/components* et il y sera construit excepté si LinuxCNC est configuré en mode simulation.

### 10.12 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 de LinuxCNC:

```
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.

## 10.13 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*.

## 10.14 Exemples

### 10.14.1 constant

Noter que la déclaration "function \_" crée les fonctions nommées "constant.0", etc. Le nom du fichier doit correspondre au nom du composant.

```
component constant;
pin out float out;
param r float value = 1.0;
function _;
license "GPL"; // indique la GPL v2 ou suivantes
;;
FUNCTION(_) { out = value; }
```

### 10.14.2 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, parce que 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 _;
```

```

license "GPL"; // indique la GPL v2 ou suivantes
;;
#include <rtapi_math.h>
FUNCTION(_) { sin_ = sin(theta); cos_ = cos(theta); }

```

### 10.14.3 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* parce que *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;

license "GPL";
;;
#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); }

```

### 10.14.4 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*.

### 10.14.5 arraydemo

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

```
component arraydemo "4-bit Shift register";
pin in bit in;
pin out bit out-# [4];
function _ nofp;
license "GPL"; // indique la GPL v2 ou ultérieures
;;
int i;
for(i=3; i>0; i--) out(i) = out(i-1);
out(0) = in;
```

### 10.14.6 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 1ms.

```
component rand;
option userspace;

pin out float out;
license "GPL";
;;
#include <unistd.h>

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

### 10.14.7 logic

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

```
component logic "LinuxCNC HAL component providing experimental logic functions";
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 11

# Créer des composants de l'espace utilisateur

### 11.1 Utilisation de base en Python

Un composant de l'espace utilisateur commence par créer ses pins et ses paramètres, puis il entre dans une boucle dans 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 (*passe\_tout.in*) vers ses pins de sortie (*passe\_tout.out*) approximativement une fois par seconde.

```
#!/usr/bin/python
import hal, time
h = hal.component("passe_tout")
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 *passe\_tout*, le rendre exécutable par un *chmod +x* et le placer dans son *\$PATH*. On peut alors l'essayer en faisant:

**halrun**

```
halcmd:loadusr passe_tout
```

```
halcmd:show pin
```

Component Pins:

Owner	Type	Dir	Value	Name
03	float	IN	0	passe_tout.in
03	float	OUT	0	passe_tout.out

```
halcmd:setp passe_tout.in 3.14
```

```
halcmd:show pin
```

Component Pins:

Owner	Type	Dir	Value	Name
03	float	IN	3.14	passe_tout.in
03	float	OUT	3.14	passe_tout.out



## 11.2 Composants de l'espace utilisateur et délais

Si vous tapez rapidement *show pin*, vous pourrez voir que *passe\_tout.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. Parce que ce composant appartient à l'espace utilisateur, ce délai peut apparaître plus long, par exemple si la mémoire utilisée par le composant *pass\_tout* 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.

## 11.3 Créer les pins et les paramètres

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

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.

TABLE 11.1: Noms des options de HAL

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 *passe\_tout.in*.

```
h.ready()
```

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

### 11.3.1 Changer le préfixe

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

## 11.4 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]
```

#### 11.4.1 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.

#### 11.4.2 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*. Positionner de manière répétitive la pin à *FALSE* pourrait faire qu'un autre composant connecté agisse comme si une nouvelle impulsion d'index avait été vue.

### 11.5 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.

### 11.6 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](#) (Paquet *python-serial*, dans les dépôts universe d'Ubuntu)
- Relier un module d'affichage à LCD [LCDProc](#) et l'utiliser pour afficher les informations de votre choix (Paquet *lcdproc*, dans les dépôts universe d'Ubuntu)
- Créer un panneau de contrôle virtuel utilisant n'importe quelle librairie d'interface graphique supportée par Python (gtk, qt, wxwindows, etc)

# Chapitre 12

## Index

Étude des interconnexions, [2](#)  
7i65, [58](#)

### A

abs, [57](#)  
Affichage de la valeur, [32](#)  
Ajustement vertical, [43](#)  
and2, [57](#)  
Anti-rebond, [77](#)  
at\_pid, [59](#)  
axis, [57](#)

### B

biquad, [58](#)  
Bit, [20](#)  
bldc\_hall3, [59](#)  
blend, [57](#)  
blocks, [5](#)

### C

Capture d'onde, [42](#)  
charge\_pump, [59](#)  
Choix d'échantillonnage, [46](#)  
Choix des organes, [2](#)  
Cinq phases, [68](#)  
clarke2, [59](#)  
clarke3, [59](#)  
clarkeinv, [59](#)  
ClassicLadder, [5](#)  
classicladder, [57](#)  
CNC, [2](#)  
Codeur, [5](#), [71](#)  
Commandes de HAL, [15](#)  
comp, [57](#)  
Comp HAL Component Generator, [94](#)  
Compiling realtime components outside the source tree, [99](#)  
Composants externes, [5](#)  
Composants HAL, [5](#)  
composants-temps-reel, [63](#)  
Concept de HAL, [4](#)  
constant, [57](#)  
conv\_bit\_s32, [58](#)

conv\_bit\_u32, [58](#)  
conv\_float\_s32, [58](#)  
conv\_float\_u32, [58](#)  
conv\_s32\_bit, [58](#)  
conv\_s32\_float, [58](#)  
conv\_s32\_u32, [58](#)  
conv\_u32\_bit, [58](#)  
conv\_u32\_float, [58](#)  
conv\_u32\_s32, [58](#)  
counter, [57](#)

### D

ddt, [57](#)  
deadzone, [57](#)  
debounce, [57](#)  
Diagramme bloc du codeur, [71](#)  
Diagramme bloc PID, [74](#)  
Diagramme bloc stepgen, [63](#)  
Dialogue des sources de déclenchement, [44](#)

### E

edge, [57](#)  
En résumé, [3](#)  
encoder, [59](#)  
encoder\_ratio, [59](#)  
estop\_latch, [59](#)  
Exemples pour HAL, [81](#)

### F

feedcomp, [59](#)  
Fenêtre de sélection, [31](#)  
Fenêtre initiale, [38](#)  
flipflop, [57](#)  
Float, [20](#)  
Fonction non liée, [37](#)  
Formes d'onde, [45](#)

### G

gantrykins, [58](#)  
gearchange, [59](#)  
genhexkins, [58](#)  
genserkins, [59](#)  
gladevcp, [57](#)

**H**

HAL, [2](#)  
HAL Composant, [4](#)  
HAL Fil, [5](#)  
HAL Fonction, [4](#)  
HAL Paramètre, [4](#)  
HAL pin, [4](#)  
HAL Signal, [4](#)  
HAL Type, [4](#)  
HAL User Interface, [87](#)  
hal-ax5214h, [6](#)  
hal-m5i20, [6](#)  
hal-motenc, [6](#)  
hal-parport, [6](#)  
hal-ppmc, [6](#)  
hal-stg, [6](#)  
hal-vti, [6](#)  
HAL: Broche physique, [4](#)  
halcmd, [6](#)  
halgui, [6](#)  
halmeter, [6](#)  
halscope, [6](#), [41](#)  
Halshow, [49](#)  
halui, [5](#)  
hm2\_7i43, [58](#)  
hm2\_pci, [58](#)  
hostmot2, [58](#)  
hypot, [57](#)

**I**

ilowpass, [59](#)  
Implémentation, [3](#)  
integ, [57](#)  
Introduction, [25](#)  
Introduction à HAL, [2](#)  
invert, [58](#)  
iocontrol, [5](#)

**J**

joyhandle, [59](#)

**K**

kins, [58](#)  
knob2float, [59](#)

**L**

L'onglet watch, [54](#)  
Les bases de HAL, [2](#)  
Les composants de HAL, [56](#)  
Les origines de HAL, [6](#)  
limit1, [58](#)  
limit2, [58](#)  
limit3, [58](#)  
logic, [57](#)  
lowpass, [58](#)  
lut5, [57](#), [79](#)

**M**

maj3, [58](#)  
match8, [57](#)  
maxkins, [59](#)  
MDI, [89](#)  
minmax, [59](#)  
Mise au point, [3](#)  
motion, [5](#), [57](#)  
mult2, [57](#)  
mux16, [57](#)  
mux2, [57](#)  
mux4, [57](#)  
mux8, [57](#)

**N**

near, [57](#)  
not, [57](#)

**O**

Observer les impulsions, [47](#)  
offset, [57](#)  
oneshot, [57](#)  
or2, [57](#)  
Outils et utilitaires, [6](#)

**P**

Périphériques canoniques, [11](#)  
pid, [5](#), [59](#), [74](#)  
Pilotes de matériel, [5](#)  
pluto\_servo, [58](#)  
pluto\_step, [58](#)  
pumakins, [59](#)  
pwmgen, [59](#), [70](#)

**Q**

Quatre phases, [67](#)

**R**

Références générales, [9](#)  
rotatekins, [59](#)

**S**

s32, [20](#)  
Sélection de la source, [39](#)  
Sélection du signal, [40](#)  
sample\_hold, [59](#)  
sampler, [59](#)  
scale, [58](#)  
scarakins, [59](#)  
select8, [57](#)  
serport, [58](#)  
setp, [18](#)  
sets, [19](#)  
siggen, [5](#), [60](#), [78](#)  
sim-encoder, [76](#)  
sim\_encoder, [60](#)  
sphereprobe, [60](#)  
stepgen, [5](#), [59](#), [63](#)  
steptest, [60](#)

streamer, [60](#)  
sum2, [57](#)  
supply, [5](#), [60](#)

## T

thc, [58](#)  
threads, [57](#)  
threadtest, [60](#)  
time, [21](#), [60](#)  
timedelay, [60](#)  
timedelta, [60](#)  
Timing pas et direction, [66](#)  
tmax, [21](#)  
toggle, [60](#)  
toggle2nist, [60](#)  
tripodkins, [59](#)  
tristate\_bit, [60](#)  
tristate\_float, [60](#)  
trivkins, [59](#)  
Trois phases, [67](#)  
Tutoriel de HAL, [25](#)  
Tutoriel Halcmd, [25](#)  
Tutoriel halmeter, [30](#)  
Tutoriel halscope, [37](#)  
Tutoriel simple, [26](#)

## U

u32, [20](#)  
updown, [57](#)

## V

Velocity exemple, [82](#)

## W

watchdog, [60](#)  
wcomp, [58](#)  
weighted\_sum, [58](#)

## X

xor2, [57](#)

---