



Université de Franche-Comté  
Master 2 Informatique



# Xenomai : Intégration et qualification d'un système temps réel sur plateforme ARMadeus.

Gwenhaël Goavec-Merou

<>



Besançon, du 9 février au 9 août 2009

**Tuteur de Stage** : Monsieur Fabien Peureux (UFR ST Franche-Comté)

**Maître de Stage** : Monsieur Julien Boibessot (Société *ARMadeus Systems*)

# Remerciements

Je tiens à remercier l'ensemble des personnes d'*ARMadeus Systems* pour leur accueil, leur aide et leurs conseils.

M. Julien Boibessot pour son aide et ses conseils sur les aspects liés à *Linux*.

M. Fabien Marteau en ce qui concerne toute la partie VHDL du stage. Je tiens également à remercier Mr Jean-Michel Friedt et toute l'équipe de *Temps Fréquence* pour leur accueil et pour le matériel mis à ma disposition.

# Table des matières

Remerciements	1
Introduction	5
<b>I Présentation</b>	<b>7</b>
<b>1 L'entreprise ARMadeus Systems</b>	<b>8</b>
1.1 Activités . . . . .	8
1.2 Organisation . . . . .	8
1.3 Carte de développement . . . . .	10
1.4 Environnement logiciel . . . . .	11
<b>2 Le stage</b>	<b>13</b>
2.1 Objectifs . . . . .	13
2.1.1 Intégration de <i>Xenomai</i> dans l'outil <i>Buildroot</i> . . . . .	13
2.1.2 Découverte et qualification de <i>Xenomai</i> . . . . .	13
2.1.3 Outils de Benchmark . . . . .	14
2.2 Conditions de stage . . . . .	15
2.2.1 Moyens matériels . . . . .	15
2.2.2 Moyens Logiciels . . . . .	15
2.2.3 Suivi de stage . . . . .	15
<b>3 Systèmes d'exploitations temps réel</b>	<b>16</b>
3.1 Généralités . . . . .	16
3.2 Xenomai . . . . .	17
3.2.1 Adeos . . . . .	18
3.2.2 Xenomai . . . . .	18
<b>II Technique</b>	<b>20</b>
<b>4 Intégration de <i>Xenomai</i></b>	<b>21</b>
4.1 Installation manuelle . . . . .	21
4.2 Intégration . . . . .	22

<i>TABLE DES MATIÈRES</i>	3
<b>5 Découverte et qualification de <i>Xenomai</i></b>	<b>25</b>
5.1 Mise en œuvre . . . . .	26
5.1.1 Attente sur un compteur . . . . .	27
5.1.2 Gestionnaire d'événements . . . . .	31
5.2 Fractale de Newton . . . . .	33
5.3 Résultats . . . . .	33
<b>6 Évaluation grâce au FPGA</b>	<b>35</b>
6.1 Présentation . . . . .	35
6.2 Mise en œuvre . . . . .	36
6.2.1 Pilote . . . . .	36
6.2.2 MACROS . . . . .	37
6.2.3 Composants <i>VHDL</i> . . . . .	38
6.3 Avancement . . . . .	41
<b>III Bilan</b>	<b>42</b>
<b>7 Conclusion</b>	<b>43</b>
7.1 <i>Xenomai</i> . . . . .	43
7.2 Apports personnel . . . . .	43
7.3 <i>Objectifs</i> . . . . .	44
<b>IV Annexes</b>	<b>46</b>
<b>A Code</b>	<b>47</b>
A.1 Script d'intégration de <i>Xenomai</i> dans le Buildroot d'Armadeus. . . . .	47
<b>B Proposition de planning</b>	<b>50</b>
B.1 Phase 1 (1 mois) . . . . .	50
B.2 Phase 2 (2 mois) . . . . .	50
B.3 Phase 3 (3 mois) . . . . .	50

# Table des figures

1.1	Carte processeur <i>APF9328</i> . . . . .	10
1.2	Carte de développement DevFull . . . . .	11
1.3	menu des packages . . . . .	12
2.1	Connexion entre le FPGA et le processeur . . . . .	14
3.1	Schéma général d'un système <i>Linux</i> et <i>Xenomai</i> . . . . .	18
4.1	Diagramme des dépendances entre les divers actions lors de l'installation de <i>Xenomai</i> . . . . .	22
4.2	Menu permettant la sélection de <i>Xenomai</i> . . . . .	24
5.1	Schéma de principe pour les exemples utilisant un oscilloscope . . . . .	27
5.2	Courbe théorique de génération de signal pour un compteur . . . . .	27
5.3	Code de gestion de compteur, en espace utilisateur. . . . .	28
5.4	Génération d'un signal en espace utilisateur. . . . .	29
5.5	Gestion de compteur en espace noyau . . . . .	30
5.6	Génération d'un signal en espace noyau. . . . .	30
5.7	Courbe théorique de gestion d'une interruption . . . . .	31
5.8	Gestion d'interruption en espace utilisateur, courbe du bas : signal de référence. . . . .	32
5.9	Gestion d'interruption en espace noyau, courbe du bas : signal de référence. . . . .	32
5.10	Rendu du calcul de la fractale de Newton sur l'écran LCD connecté à la carte <i>ARMadeus</i> . . . . .	33
6.1	Schéma de principe pour les tests utilisant le <i>FPGA</i> . . . . .	35
6.2	Structure de l'arborescence pour l'accès depuis le système de fichier . . . . .	37
6.3	Schéma global de l'outil de Benchmarking <i>VHDL</i> . . . . .	38
6.4	Machine d'états de l'IP Logger . . . . .	40

# Introduction

Un système informatique, du point de vue du processeur ne peut traiter qu'une instruction à la fois et par extension un seul processus<sup>1</sup> ne peut être actif à un instant donné.

Compte tenu de cette situation et afin que l'utilisateur soit en mesure de lancer plusieurs applications sur son ordinateur, les systèmes d'exploitations (*OS*) offrent un comportement multitâche. Ce type de fonctionnement repose sur un algorithme complexe, nommé *ordonnanceur*, permettant le basculement de processus selon une certaine politique. Dans le cas de *Linux*, *OS* multitâche préemptif à temps partagé, chaque processus se voit attribuer une tranche de temps, nommée *quantum*, au bout duquel il est suspendu pour qu'un autre processus puisse s'exécuter, en opposition à d'autres systèmes, non préemptif, où c'est le processus, une fois sa tâche finie, qui rend les ressources permettant à un autre de s'exécuter. Ce mécanisme permet un basculement rapide entre l'ensemble des processus. D'autre part le caractère préemptif du système permet, lors du déclenchement d'une interruption<sup>2</sup> de suspendre l'exécution du processus courant afin de donner la main au gestionnaire de cette interruption.

Afin de donner à l'utilisateur l'impression que l'éditeur de texte, le lecteur de musique et le logiciel de mail s'exécutent en parallèle, chaque processus se voit attribuer, en plus d'un *quantum*, une priorité. Celle-ci est dynamique, évoluant en fonction d'un ensemble de paramètres dont le plus important est la quantité totale de temps pendant laquelle il s'est exécuté. Plus celle-ci est importante plus sa priorité va diminuer, permettant de ce fait de favoriser un processus ayant été moins longtemps actif.

La réelle distinction entre système d'exploitation à temps partagé et système d'exploitation temps réel se fait au niveau des priorités. En effet, dans le premier cas, l'ordonnanceur réajuste régulièrement la valeur de la priorité pour chaque processus permettant, comme dit précédemment, que tous les processus puissent avoir accès aux ressources. Dans le second cas, les priorités sont fixes. Ainsi une tâche plus importante qu'une autre, même si celle-ci est souvent active, aura toujours la main, alors qu'une autre dont la priorité sera plus faible ne pourra s'exécuter que lorsque la précédente ne demandera pas les ressources.

Pour un ordinateur utilisé pour des tâches courantes, telles que celles présentées précédemment, le temps partagé est idéal car offrant une meilleure impression de réactivité du point de vue de l'utilisateur.

Toutefois, ce mécanisme entraîne un problème pour une utilisation plus critique : il n'est pas possible de prédire le temps que mettra une tâche pour être active et donc le temps avant qu'un traitement donné soit fait. D'autre part un processus pouvant à

---

1. instance d'un programme en cours d'exécution  
2. évènement matériel

n'importe quel moment être préempté<sup>3</sup>, suite à une interruption, il est impossible d'assurer que le traitement lié sera réalisé dans une durée acceptable.

Dans ce document, nous nous intéresserons à une solution particulière de temps réel qui allie à la fois un noyau à temps partagé et un noyau temps réel. Cette solution permettant de faire cohabiter des applications ayant besoin d'être gérées d'une manière prioritaire et d'autres n'ayant pas de contraintes de temps d'exécution ou d'accès aux ressources.

Ce travail se fera en trois temps :

- Dans un premier temps cette solution sera mise en place d'une manière manuelle puis un script sera réalisé afin d'automatiser l'installation.
  - Ensuite une découverte de son fonctionnement et une première évaluation de son comportement sera faite.
  - Et pour finir une évaluation plus pertinente à l'aide d'un outil qui devra être créé.
- Pour chaque étape, une documentation doit être réalisée.

---

3. suspendu

# Première partie

## Présentation

# Chapitre 1

## L'entreprise ARMadeus Systems

*ARMadeus Systems* est un bureau d'étude de quatre personnes, situé à Mulhouse, spécialisé dans les systèmes embarqués.

Cette entreprise développe et produit des cartes à base de processeur ARM<sup>Tm</sup><sup>1</sup> couplé à un FPGA, ainsi que le BSB<sup>2</sup> correspondant.

### 1.1 Activités

L'entreprise *ARMadeus Systems* a trois pôles d'activité :

- la base de l'activité de l'entreprise est la conception ainsi que la fabrication de cartes processeur. Celles-ci sont accompagnées de plusieurs modèles de cartes d'accueil permettant de tester les différentes fonctionnalités de la plateforme. Leur vente au public se fait via le site internet <http://www.armadeus.com>.
- sur la base de cette carte déjà conçue, *ARMadeus Systems* propose de créer des produits finis pour des clients industriels.
- en plus de la production et du développement, *ARMadeus Systems* est une société de service dans le domaine de l'électronique embarquée. L'entreprise propose des prestations de service au forfait ou en régie dans son domaine d'activité.

### 1.2 Organisation

ARMadeus Systems est composée de quatre personnes :

- Frédéric Burkhart : ingénieur systèmes embarqués.
- Nicolas Colombain : ingénieur en électronique.
- Julien Boibessot : ingénieur spécialiste en systèmes *Linux* embarqués.
- Fabien Marteau : ingénieur matériel.

F.Burkhart et N.Colombain sont les deux co-dirigeants. N.Colombain est le principal développeur hardware et F.Burkhart est chargé de la communication et de la promotion d'*ARMadeus Systems*. Tous deux développent en parallèle la partie logiciel des cartes. J.Boibessot est spécialisé dans les systèmes embarqués sous *Linux*. C'est le principal res-

---

1. i.MXL de chez Freescale

2. Base System Board ou carte d'accueil

responsable du portage de *Linux* pour les différentes plateformes que développe *ARMadeus Systems*. F.Marteau est le spécialiste *FPGA* et *VHDL*.

### 1.3 Carte de développement

*ARMadeus Systems* propose deux modèles de cartes processeur ainsi que les cartes d'accueil correspondantes, mais il ne sera fait mention ici que de celle qui a servit pour mon développement, l'*APF9328* et la carte *devFull*.



FIGURE 1.1 – Carte processeur *APF9328*

La carte processeur *APF9328*<sup>3</sup> (figure 1.1), est constituée de :

- un microprocesseur ARM9 Freescale MC9328L (i.MXL) ARM920T cadencé à 200 MHz.
- une RAM de 16 Mo.
- un FPGA<sup>4</sup> *Xilinx Spartan3*<sup>5</sup> XC3S200 de 200 K portes.

Le processeur et le FPGA dialoguent à travers une zone mémoire commune.

C'est une carte générique qui est utilisée dans l'ensemble des projets de la société.

Cette première carte est insérée sur un *BSB* pouvant être spécifique au besoin d'un client, ou faite en série pour les personnes intéressées par le monde de l'embarqué sans attentes particulières.

Dans le cadre du stage, la version générique, la *devFull*(Fig. 1.2), sera utilisée.

Celle-ci permet de pouvoir accéder à des connecteurs tels que SPI, I2C, RJ45, DB9, LCD, à un ensemble de broches et à l'audio, l'USB, le SVIDEO, etc...

---

3. [http://armadeus.com/english/products-processor\\_boards-apf9328.html](http://armadeus.com/english/products-processor_boards-apf9328.html)

4. Field-programmable gate array

5. [http://www.xilinx.com/support/documentation/data\\_sheets/ds099.pdf](http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf)

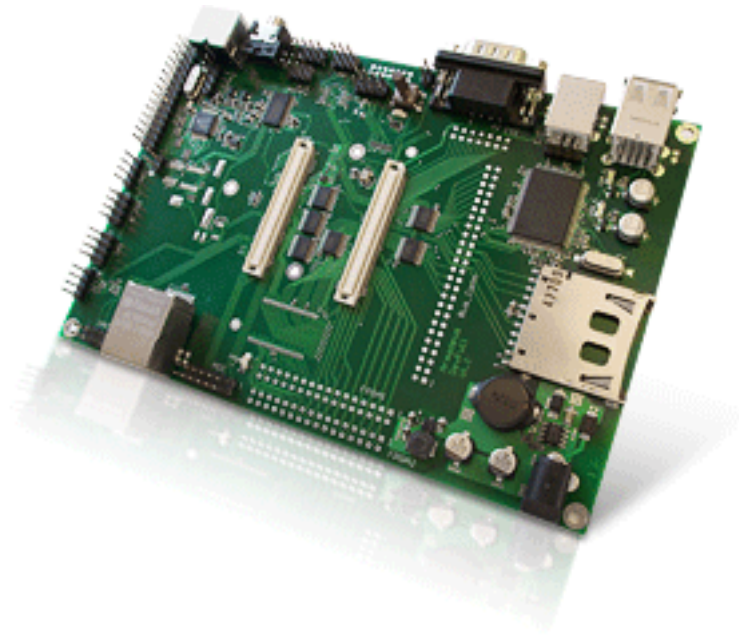


FIGURE 1.2 – Carte de développement DevFull

Cette version permet la réalisation d'applications en tirant profit de l'ensemble des fonctionnalités offertes par l'*APF9328* ainsi que d'autres plus spécifiques à la carte d'accueil.

## 1.4 Environnement logiciel

D'un point de vue logiciel, la carte *ARMadeus* est fournie avec *Linux*. Le système d'exploitation tournant sur la carte est généré grâce à un outil nommé *Buildroot*<sup>6</sup>.

Cet outil permet de construire une image qui sera ensuite installée dans la mémoire flash de la carte. La génération se faisant dans le cas présent par *cross-compilation*, à l'aide d'une *toolchain*<sup>7</sup> spécifique au microprocesseur de destination.

L'ensemble des options concernant la cross-compilation et les éléments installés se fait au travers de menus dans un terminal.

La Fig. 1.3 présente le menu principal permettant la sélection des outils et applications ensuite disponibles après compilation et *flashage* de la carte *APF9328*.

---

6. <http://buildroot.uclibc.org>

7. compilateur, bibliothèques, etc...

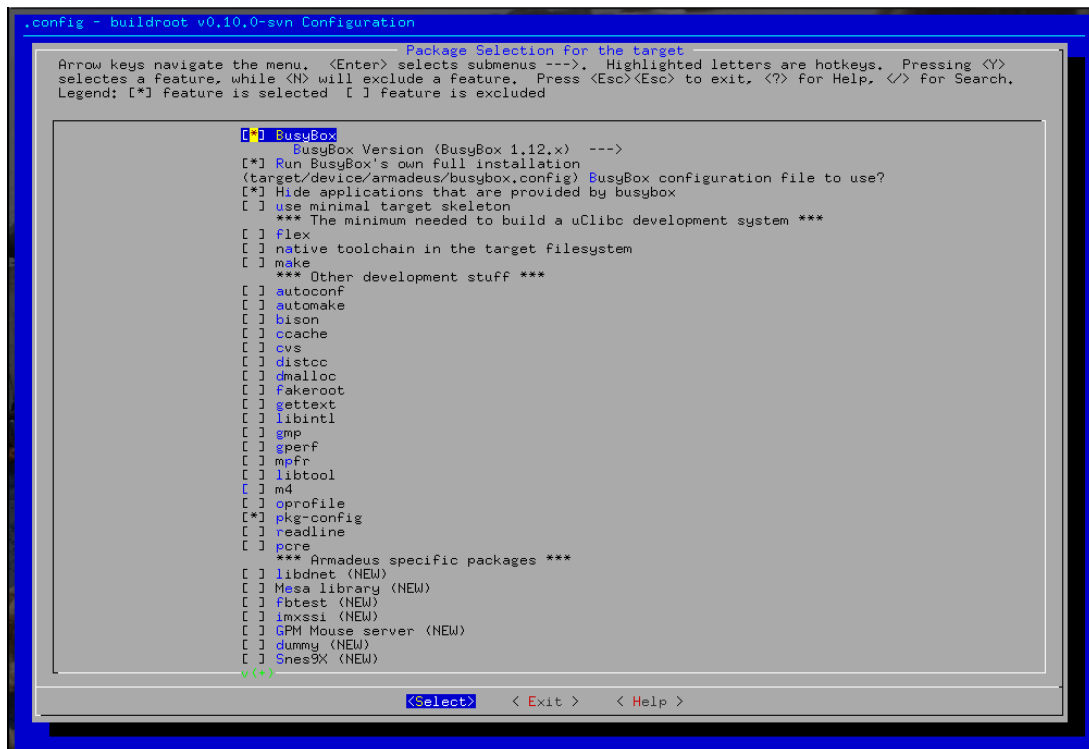


FIGURE 1.3 – Menu permettant la sélection des applications installées sur la carte

L'activation ou la désactivation d'un *package*<sup>8</sup> se fait à l'aide de la barre d'espace.

La compilation et la génération de l'image qui sera envoyée sur la carte, sont effectuées d'une manière totalement automatique, grâce à la commande `make`, l'ensemble des étapes et dépendances étant complètement géré avec des scripts fournis par *Buildroot*.

8. packages : nom donné aux applications dans la terminologie de Buildroot

# Chapitre 2

## Le stage

### 2.1 Objectifs

Le stage avait pour but la mise à disposition de *Xenomai* dans les outils *ARMadeus*, ainsi que le développement, en interne, d'une application servant à valider des applications temps réel pour des clients, dans un futur proche.

Le planning prévisionnel du stage est présenté en annexe B. Les objectifs sont les suivants.

#### 2.1.1 Intégration de *Xenomai* dans l'outil *Buildroot*

Les cartes réalisées et vendues par *ARMadeus Systems*, sont fournies avec un outil permettant la génération d'un système d'exploitation *Linux* complet qui sera flashé sur la carte.

Comme présenté dans la section 1.4, il est possible grâce à *Buildroot* d'installer des applications (*packages*) en plus de celles strictement nécessaires au fonctionnement minimal du système d'exploitation.

*Xenomai* n'étant pas déjà intégré dans *Buildroot*, la première tâche a donc été, à l'instar des autres applications, la création du script nécessaire à son installation, permettant ainsi à n'importe qui souhaitant utiliser ou essayer *Xenomai*, sur plateforme *ARMadeus*, de pouvoir le faire simplement, sans pour cela avoir besoin de compétences avancées.

Par ailleurs la documentation disponible sur le wiki[xenomai\_manuel] ne faisant référence qu'à l'installation manuelle, il a été nécessaire de la remettre à jour pour prendre en compte la disponibilité d'un *package* pour l'installation de *Xenomai*.

#### 2.1.2 Découverte et qualification de *Xenomai*

Une fois l'extension temps réel mise en place et fonctionnelle, l'étape suivante, à travers la création d'un ensemble d'applications ou de modules avait deux objectifs :

- d'une part l'étude du fonctionnement de *Xenomai* afin de pouvoir fournir un condensé des fonctions de base et de pouvoir en comprendre la logique de fonctionnement, permettant ainsi à *ARMadeus* de répondre à des demandes de clients.
- d'autre part d'évaluer les temps d'exécution ou de latence d'applications *Xenomai* vis-à-vis de *Linux*.

Les applications créées sont mises à disposition ensuite dans les outils d'*ARMadeus*, fournissant ainsi aux personnes intéressées par cette extension un ensemble d'exemples de démonstration.

### 2.1.3 Outils de Benchmark

La troisième et dernière étape a consisté à réaliser une application de Benchmarking. Celle-ci permet d'évaluer les temps de réponses d'applications ou de fonctions sans modifications du noyau.

Comme le *FPGA* et le microprocesseur sont reliés (Fig. 2.1) et pour que l'application ne soit pas soumise aux contraintes du système d'exploitation, cet outil sera implémenté en VHDL sur le FPGA intégré à la carte.

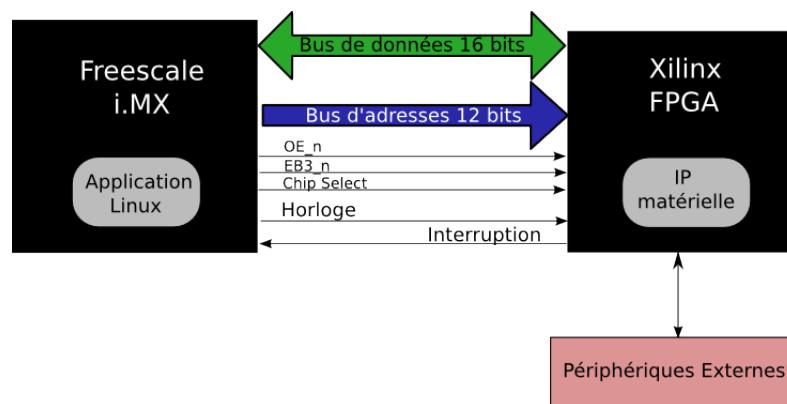


FIGURE 2.1 – Connexion entre le FPGA et le processeur

Cet outil peut être décomposé en trois parties :

- Le compteur, stockant les valeurs obtenues. Ce premier élément doit être réalisé en VHDL afin d'être utilisé dans le FPGA.
- Un pilote *Linux*, permettant la configuration des paramètres du module *VHDL*, ainsi que la récupération des valeurs obtenues.
- Un ensemble de *MACROS* prévues pour être insérées dans le code de l'application à tester. Elles ont pour but d'accéder à une zone mémoire particulière, partagée par le microprocesseur et le FPGA afin de pouvoir signaler le début et la fin du comptage.

Les contraintes au niveau des modifications nécessaires pour l'application à tester étaient les suivantes :

- l'ajout en terme de code doit être le plus simple possible.
- l'ensemble des *MACROS* doit pouvoir être utilisé pour une application mais également pour un module noyau.
- le fonctionnement des *MACROS* ne doit pas impacter sur le comportement normal de l'application et doit être atomique afin de ne pas avoir d'impact sur les valeurs obtenues.

## 2.2 Conditions de stage

Le stage s'est déroulé à l'ENSMM à Besançon car la société *ARMadeus*, bien que possédant une antenne dans cette ville, n'a pour le moment pas de locaux.

### 2.2.1 Moyens matériels

J'ai eu à ma disposition un ordinateur portable fonctionnant sous *Linux*, une carte *DevFull* complète ainsi que tout le matériel nécessaire pour son raccordement à l'ordinateur. Des documents papiers ont également été mis à ma disposition. A été mis aussi à ma disposition un oscilloscope numérique quatre voies et un synthétiseur basse fréquence afin de pouvoir réaliser les divers tests.

### 2.2.2 Moyens Logiciels

Afin de pouvoir exploiter la carte, un client pour le protocole *RS232*<sup>1</sup> a été installé sur l'ordinateur, ainsi que tous les outils me permettant la compilation du système nécessaire au fonctionnement de la carte.

La réalisation de modules VHDL a nécessité l'installation de :

- *POD*<sup>2</sup>[*PODWiki*] pour la création de la structure globale de connexion des composants et leur intégration avec le bus *Wishbone*<sup>3</sup>. Ce logiciel peut également servir à la synthèse<sup>4</sup>.
- *ISE* de la société Xilinx. Ce logiciel permet la création du binaire permettant la programmation du FPGA. Il a été également utilisé pour simplifier la phase de test car il propose un éditeur de texte pour la programmation. Il est par ailleur nécessaire à *POD* pour la phase de synthèse de code *VHDL*

Comme les outils d'*ARMadeus* sont mis à disposition avec *SVN*<sup>5</sup>, l'outil nécessaire pour ce protocole a également été installé.

### 2.2.3 Suivi de stage

Toutes les semaines, une réunion avec M Boibessot était organisée, me permettant ainsi de pouvoir présenter les travaux déjà réalisés et permettant de pouvoir fixer le planning du travail jusqu'à la prochaine réunion.

Afin de pouvoir présenter l'avancée du stage et les détails des travaux réalisés mais aussi pour fixer les modalités pour la suite, des déplacements sur Mulhouse ont été effectués.

Par ailleurs, pour pouvoir être en relation quasi permanente avec les personnes de la société *ARMadeus*, l'*IRC*<sup>6</sup> a été utilisé, ceci afin de pouvoir les tenir au courant tous les jours des avancées, fixer des points qui pouvaient ne pas être clairs ou pour évaluer une situation apparaissant comme bloquante évitant ainsi le risque de perdre du temps et de dépasser les délais prévus initialement.

---

1. communication série
2. Peripheral On Demand
3. bus de communication entre FPGA et microprocesseur
4. génération du binaire pour le FPGA à partir du code VHDL
5. outils de gestion de versions pour du travail collaboratif
6. *Internet Relay Chat*. protocole pour la discussion instantanée par groupe

# Chapitre 3

## Systemes d'exploitations temps réel

### 3.1 Généralités

La notion de temps réel caractérise un système ou un logiciel dont le temps de réponse est compatible avec la dynamique du système matériel. En d'autre terme, ce type de système permet d'assurer qu'une information après acquisition et traitement n'est pas obsolète.

Il y a deux catégories de systèmes temps réel :

- Le temps réel mou (ou souple) qui tolère une dérive dans le temps de traitement de l'ordre de 500ms à la seconde.
- Le temps réel dur qui a des contraintes temporelles strictes afin d'offrir une véritable garantie de pertinence de l'information traitée.

Les solutions temps réel peuvent être regroupées en trois ensembles :

Le premier consiste en l'utilisation d'un système complètement dédié à cette tâche. L'avantage de cette solution réside dans la possibilité de pouvoir avoir un environnement homogène car complètement temps réel.

Toutefois, celle-ci présente également des inconvénients :

- Certaines solutions telles que QNX<sup>1</sup> sont commerciales, entraînant de ce fait une augmentation du coût liée à l'aspect logiciel.
- Le support matériel est souvent réduit. Cela s'explique principalement par le fait que le matériel est prévu pour durer et que les systèmes temps réel se trouvent dans une sorte de niche qui ne profite pas du même engouement que les systèmes d'exploitations classiques ou bien, qui de par sa nature commerciale, empêche les personnes intéressées d'apporter le support. Il est possible de constater, par exemple, que la plupart des solutions existantes ne sont pas compatibles avec les processeurs équipant les cartes produites par la société *ARMadeus*.
- Ces systèmes impliquent un changement total du point de vue système. Le portage d'applications existantes n'est pas forcément triviale et entraîne également un surcoût en terme de développement.
- Pour finir, comme dans le cas de *uC/OS* ou de *RTEMS*<sup>2</sup>, tel que présenté dans [LmNDS] ce ne sont pas des systèmes d'exploitations mais des exécutifs. En d'autres

---

1. <http://fr.wikipedia.org/wiki/QNX>

2. <http://www.rtems.org>

termes, contrairement à *Linux* qui peut lancer dynamiquement de nombreuses applications, ces environnements sont des sortes de boîtes à outils de composants. Une fois compilées, elles se présentent sous la forme d'un binaire monolithique massivement multithreadé. Chaque tâche est contenue dans un Thread concurrent vis-à-vis des autres et avec une priorité définie. Il n'est pas possible de charger des applications. Ce sont donc des solutions très spécifiques réservées à un domaine spécialisé.

La seconde solution est l'application d'un patch<sup>3</sup> sur le noyau *Linux*, afin de le rendre temps réel. Les avantages vis-à-vis de la précédente solution sont :

- le système d'exploitation est totalement gratuit.
- il n'y a pas de contraintes de modifications des applications.
- cette solution profite de la très grande communauté des développeurs *Linux*.

Toutefois, cette solution n'apporte qu'un mécanisme temps réel mou et ne peut donc offrir les mêmes garanties que dans le cas précédent. Elle ne peut donc pas rivaliser avec une vraie solution temps réel.

La dernière solution peut être vue comme un compromis entre les deux précédentes. *Linux*, de par sa conception originelle, a pour vocation d'offrir un système d'exploitation multitâches, sans notion de temps réel. Il a dans le cas présent l'avantage d'être soutenu par une communauté importante rendant ce système pérenne dans le temps et disponible sur une très grande variété de plateformes et de processeurs.

Le principe, pour pouvoir profiter à la fois d'un OS généraliste disponible sur un nombre important de matériel, est d'ajouter un second noyau, celui-ci temps réel, gérant les applications critiques nécessitant des temps bornés et laissant à *Linux* la charge de celles de moindre importance.

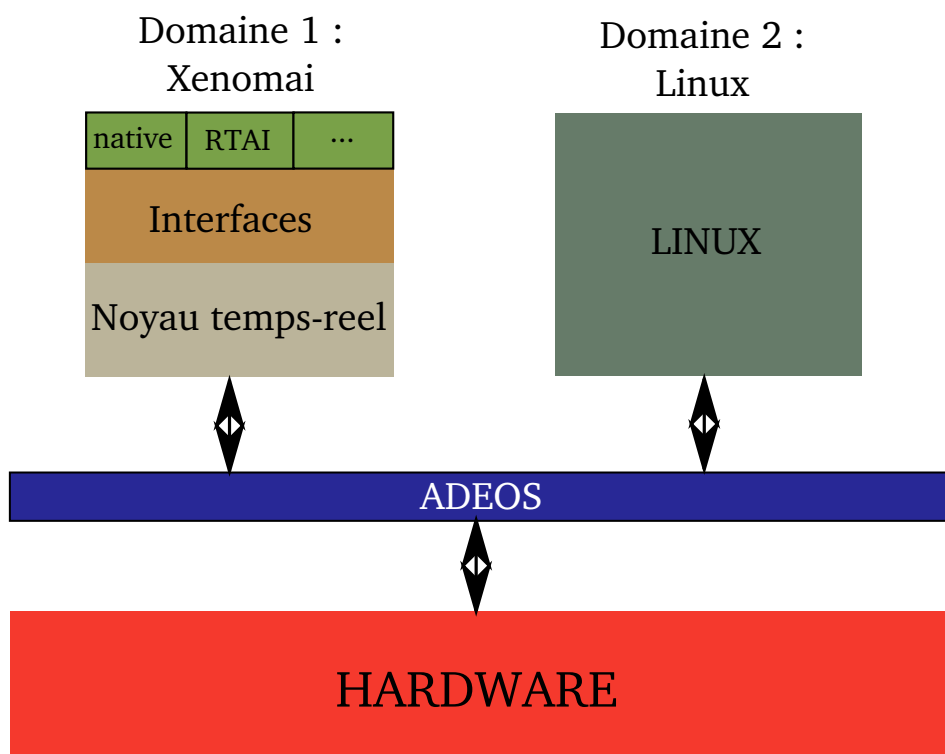
C'est sur ce principe que se base l'extension *Xenomai*. C'est cette solution qui sera exploitée.

## 3.2 Xenomai

Le principe sous-jacent est de fournir à la fois un environnement temps réel dur et un autre environnement plus classique, permettant ainsi d'avoir des processus dont les temps d'exécution sont garantis, mais également de profiter des avantages de *Linux* pour les autres processus.

---

3. mécanisme permettant l'application automatique de modifications de code

FIGURE 3.1 – Schéma général d'un système *Linux* et *Xenomai*

Comme le présente la figure 3.1, chaque système se trouve dans un domaine particulier. Un domaine est une sorte de boîte possédant une priorité et contenant un système d'exploitation. Ces domaines pourraient être comparés globalement avec les concepts généraux de la virtualisation. La concurrence au niveau des ressources matérielles se fait à travers la couche basse *Adeos*.

### 3.2.1 Adeos

La couche d'abstraction matérielle *Adeos*, s'intègre entre le matériel et le(s) système(s) d'exploitations. Son rôle est de distribuer aux *OS* les événements matériels. Chaque OS est contenus dans un domaine possédant une priorité fixe dans la chaîne de délivrance des signaux.

*Adeos* implémente une queue de signaux nommée *Ipipe* : lorsqu'un événement matériel se produit, *Adeos* en avertit à tour de rôle l'ensemble des domaines, en commençant par celui dont la priorité est la plus forte pour finir par celui dont la priorité est la plus faible. Le système d'exploitation peut gérer ou non le signal. Les signaux non gérés sont proposés au système suivant dans la chaîne.

### 3.2.2 Xenomai

*Xenomai* est un noyau temps réel dur, collaborant avec le noyau *Linux*, dont il peut utiliser les pilotes.

Il est contenu dans le domaine à plus forte priorité, le noyau *Linux* résidant dans un domaine de plus faible priorité. Il prend en charge toutes les tâches temps réels laissant

les autres au noyau *Linux*.

L'intérêt de *Xenomai* vis-à-vis des autres solutions, a déjà été présenté mais il est important de revenir plus précisément sur certains aspects.

Le portage d'une application temps réel depuis un autre système d'exploitation vers *Linux* pose des problèmes à plusieurs niveaux :

- Les APIs des divers *RTOS*<sup>4</sup>, bien qu'ayant les mêmes concepts, n'ont pas forcément le même « vocabulaire », entraînant la réécriture d'une grande partie du code.
- Certains comportements sous-jacent sont différents vis-à-vis des politiques implémentées dans l'API POSIX de *Linux*. Ceci impose de devoir repenser entièrement l'application en tenant compte des contraintes de *Linux*.

Pour éviter ces problèmes, *Xenomai*, tel que présenté Fig.3.1, propose une interface générique qui peut être considérée comme une couche d'abstraction.

*Xenomai* implémente une version générique de l'ensemble des besoins en terme de temps réel.

Les *skins*, se trouvant au-dessus, fournissent une implémentation des spécifications de l'interface de programmation du système d'exploitation concerné et en émulent le comportement.

Grâce à ce mécanisme, le portage d'une application vers *Xenomai* est relativement transparent, le *skin* correspondant au système d'exploitation d'origine offrant exactement le même comportement que si l'application était lancée sur celui-ci.

*Xenomai* ne met pas en valeur une API vis-à-vis d'une autre, permettant au développeur habitué au temps réel de réaliser une application avec l'API de son choix et à un développeur découvrant le temps réel d'utiliser l'API POSIX telle qu'il la connaît avec *Linux*.

Le dernier point important est que *Xenomai*, en plus d'offrir des APIs pour le développement d'applications en espace utilisateur offre également la possibilité de développer en espace noyau, grâce au *skin RTDM*.

---

4. Real Time Operating System

# Deuxième partie

## Technique

# Chapitre 4

## Intégration de *Xenomai*

*Buildroot* ne proposant pas nativement l'installation de *Xenomai*, la toute première étape a donc consisté à l'intégrer en son sein.

L'installation d'un logiciel à travers *Buildroot* repose sur un script, celui-ci ayant pour but de gérer l'ensemble des étapes nécessaires à la mise en place de l'application. Chaque action peut être ou non dépendante d'une autre.

Toutefois avant de pouvoir en automatiser l'installation, il a fallu comprendre la suite des étapes nécessaires à sa mise en œuvre. Ce n'est qu'une fois l'installation manuelle comprise que la réalisation d'un script pouvait être possible. De la sorte, n'importe qui voulant utiliser ou du moins tester *Xenomai* peut le faire en l'activant dans la liste des applications à installer.

### 4.1 Installation manuelle

Cette première installation est basée sur le document [xenomai\_manuel].

L'ajout de *Xenomai* à *Linux* nécessite l'application de plusieurs patches, afin de rajouter au noyau d'une part le support de l'abstraction logicielle *Adeos* et d'autre part le support de *Xenomai* à proprement parler.

La mise en place de *Xenomai* se déroule en quatre étapes :

- l'application du patch *Adeos* 1.12.0 afin d'ajouter la couche d'abstraction *Ipipe*.
- l'application des patches *Xenomai* pour l'ajout de ce dernier au noyau, à l'aide du script *prepare\_kernel* fourni dans l'archive de *Xenomai*.
- l'activation de *Ipipe* (issue de *Adeos*) et de *Xenomai*, ainsi que des options et modules pour ces deux au niveau des options du noyau.
- la compilation de la partie logicielle (bibliothèques) de *Xenomai*

Au terme de cette suite d'action, le système installé sur la carte est en mesure d'exécuter des applications ou modules temps réel.

Une fois l'installation manuelle comprise, il devient possible de synthétiser le principe de base permettant la création du *package*, se chargeant de l'ensemble des étapes automatiquement.

**Remarque :**

La première tentative de lancement du noyau modifié a présenté un problème avec *Adeos*. En effet, suite à son activation dans le noyau, le système semblait se bloquer lors du démarrage. Après une phase de debuggage de la séquence d'initialisation du système, il a été possible de déterminer que ce blocage venait d'une instruction manquante. Celle-ci empêchait l'incrémentation d'un compteur de temps global utilisé par le noyau. Une fois le problème corrigé, un patch a pu être réalisé et a été proposé à la communauté *Adeos*<sup>a</sup>. La correction nécessaire a été ajoutée dans la version *1.12-01* du patch *Adeos*.

a. <https://mail.gna.org/public/adeos-main/2009-02/msg00037.html>

## 4.2 Intégration

L'intégration de *Xenomai* dans *Buildroot* n'est pas une tâche triviale.

*Buildroot* utilise un mécanisme de script de type Makefile [buildroot\_mk]. Ce fichier prend en charge l'ensemble des étapes nécessaires depuis le téléchargement des sources jusqu'à l'installation des binaires.

Il a donc été nécessaire non seulement de suivre les étapes présentées précédemment, mais en plus, de prendre en compte plusieurs cas d'utilisation pouvant se produire et dévier du mécanisme de base.

Un autre point délicat concerne les dépendances entre actions : certaines étant indépendantes comme le téléchargement du patch *Adeos* et de l'archive de *Xenomai*. Le schéma 4.1 présente la suite de dépendances pour le script d'installation de *Xenomai*, donné en annexe A.1.

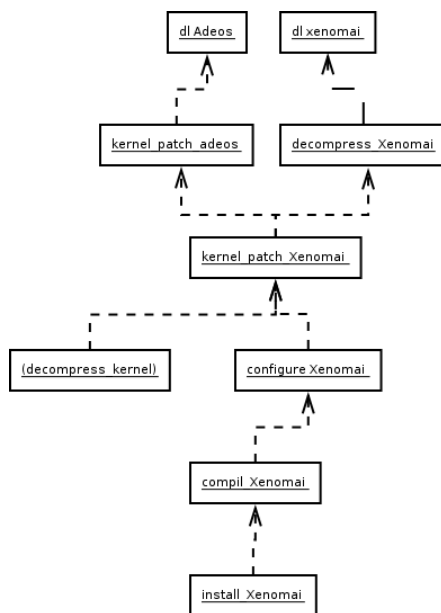


FIGURE 4.1 – Diagramme des dépendances entre les divers actions lors de l'installation de *Xenomai*.

Il est à noter que dans le schéma, certaines actions ne sont pas appelées en interne, telle que *decompress\_kernel*, celle-ci, réalisée par *Buildroot* avant toute autre tâche, est

utilisée lors de la phase de décompression du noyau.

Contrairement à la plupart des applications disponibles en tant que *packages* dans *Buildroot*, la sélection de *Xenomai* nécessite la modification du noyau, rajoutant une dépendance en plus.

Comme l'installation de *Xenomai* modifie le noyau *Linux*, elle se fait en deux temps :

1. d'une part, suite à la décompression du noyau, *Buildroot* applique une série de patches :
  - les premiers permettent de faire évoluer le noyau de sa version de base vers une révision (patch officiel du noyau)
  - d'autres, issus d'*ARMadeus*, corrigent des bugs relatifs aux composants de la carte ou ajoutent des modules non présents dans le noyau.

Il faut donc, dans le cas où *Xenomai* est activé, appliquer les patches suivants :

- le patch *Adeos* pour l'ajout de *Ipipe*, nécessitant le téléchargement de celui-ci.
- le patch pour le support de *Xenomai*, fait à l'aide du script `prepare.kernel`, nécessitant le téléchargement de l'archive de *Xenomai* et sa décompression.

L'utilitaire *make* utilisé par *Buildroot* se base sur la date de modification (ou l'existence) de fichier(ou de répertoires) afin de réaliser ou pas une tâche. Il a fallu, une fois les patches appliqués, créer un fichier caché<sup>1</sup> dans le répertoire des sources du noyau, sans quoi les patches sont appliqués à chaque nouvelle compilation du noyau.

2. la seconde partie de l'installation de *Xenomai*, consiste en la compilation et l'installation des bibliothèques nécessaires aux applications en espace utilisateur.

Cette étape est-elle même divisée en plusieurs sous-étapes dépendantes les unes des autres :

- Le lancement de la commande `./configure` pour configurer les options de bases (type de processeur, répertoire de destination, ...). Elle est dépendante de l'action d'application des patches.
- la compilation des sources dépendante de la précédente,
- l'installation des binaires et bibliothèques.

La difficulté, lors de la réalisation du script, a été de trouver la solution pour éviter une dépendance trop importante entre le noyau et la compilation de *Xenomai*. Il fallait, en effet, tenir compte de plusieurs cas de figures :

- ni le noyau, ni *Xenomai* n'ont été précédemment installés. Dans ce cas, la situation est simple.
- Le noyau a été supprimé et il doit être à nouveau patché. Il faut éviter dans ce cas que *Xenomai* soit également patché et recompilé.
- le répertoire contenant les sources de *Xenomai* a été supprimé. Il ne faut pas toucher dans ce cas au noyau.

C'est pourquoi l'ensemble des étapes a du être découpé de la manière la plus fine possible afin d'éviter les situations dans laquelle une étape est réalisée alors qu'elle ne devrait pas.

---

1. commençant par un point

```
[ ] sudo
[*] Xenomai
[ ] Database --->
[*] Networking --->
```

FIGURE 4.2 – Menu permettant la sélection de *Xenomai*

Une fois le script réalisé et intégré dans *Buildroot* il est possible, tel que présenté sur la figure 4.2, d'activer ou de désactiver le support pour *Xenomai*.

Par ailleurs, une fois le *package* réalisé le *wiki*<sup>2</sup> a été modifié [xenomai\_auto] afin de présenter la méthode d'installation *Xenomai* à travers l'outil *Buildroot*.

---

2. <http://www.armadeus.org>

# Chapitre 5

## Découverte et qualification de *Xenomai*

Une fois en mesure de pouvoir exploiter l'extension *Xenomai*, il est nécessaire d'une part, d'apprendre comment mettre en œuvre des applications l'utilisant et d'autre part, d'en évaluer les performances.

Les deux problèmes les plus courants, avec un système à temps partagé tel que *Linux*, sont :

- la durée que va mettre un processus à être activé suite à son réveil,
- le temps qu'il va mettre à traiter un événement extérieur.

Comme présenté plus tôt, le système attribue à chaque processus un volume de temps équivalent ainsi qu'une priorité dynamique. Lors de l'expiration de la durée d'un compteur ou bien lors de l'arrivée d'une interruption matériel, le processus en cours d'exécution est préempté<sup>1</sup> et l'ordonnanceur du système fait une nouvelle élection. Le processus gérant l'interruption, ou étant réveillé, peut obtenir rapidement la main si les autres processus ont tous consommé leur *quantum* ou s'il a la priorité la plus élevée.

Mais si d'autres processus ont des priorités plus élevées ou s'il a consommé tout son temps d'exécution, le processus sera juste mis dans l'état ACTIF mais ne pourra pas s'exécuter, attendant une future élection, plus favorable. Il est donc tout à fait possible que le processus devant être réveillé après 10 ms ne le soit qu'au bout de 30 ms ou dans le cas d'une gestion, suite à une interruption, que le temps de latence soit supérieur au temps entre deux interruptions. Dans ces deux cas, il n'est plus possible d'assurer la pertinence du traitement.

Ce sont ces deux types de traitements qui vont être analysés, permettant d'offrir ainsi une base de comparaison entre les applications en temps partagé et en temps réel. Les traitements seront réalisés dans l'ensemble des cas de figures possibles, à savoir en espace utilisateur et espace noyau *Linux* et *Xenomai*

Pour que la comparaison puisse être vraiment pertinente, il est primordial de pouvoir faire une évaluation non seulement dans le cas où la machine est faiblement chargée mais également quand celle-ci est stressée. Dans ce dernier cas de figure, non seulement le processeur est soumis à une forte charge de travail mais l'ordonnanceur est également sollicité par la forte augmentation du nombre de processus demandant l'accès au processeur. C'est le pire cas en terme de temps de réponse.

---

1. suspendu

La solution choisie, devant pouvoir fournir une charge de travail prédictive, donc qui soit la même à chaque exécution, a consisté à réaliser une application calculant une fractale de Newton. Le résultat étant ensuite affiché sur un écran LCD raccordé à la carte. Ce calcul, réalisé en parallèle par plusieurs processus aura pour but de déstabiliser le phénomène périodique généré par les exemples.

Les applications réalisées pour comparer *Linux* et *Xenomai* dans les cas de figures présentés, permettront par ailleurs de découvrir les bases des APIs<sup>2</sup> offertes par *Xenomai*.

## 5.1 Mise en œuvre

Afin de pouvoir réaliser un ensemble de tests exhaustifs, les concepts présentés sont évalués en espace utilisateur (application) et en espace noyau (module).

Deux ensembles de tests ont été réalisés :

1. Des applications et modules se réveillant de manière cyclique, au bout d'une durée définie globalement.
2. Des applications et modules en attente d'un événement extérieur (interruption). Celui-ci généré grâce à un synthétiseur de signaux basse fréquence, permettant une bonne précision.

Pour obtenir un résultat visuel grâce à un oscilloscope numérique en mode persistance, l'ensemble des applications et modules réalisés changent l'état d'une broche du microprocesseur raccordée à l'oscilloscope. Le résultat obtenu étant donc un signal carré qui :

- dans le cas de la mise en sommeil, la demi-période correspond au temps entre deux réveils de l'exemple. La forme et la coloration du signal à l'oscilloscope permet d'obtenir une valeur moyenne ainsi que les extrêmes (minimum et maximum) par rapport à la période attendue.
- Dans le cas de la gestion des interruptions, le front montant<sup>3</sup> pourra être comparé au front montant déclencheur du signal de référence, l'écart donnant la latence.

La Fig. 5.1 présente le schéma utilisé dans l'ensemble des tests. Ce montage impose du le raccordement du synthétiseur base fréquence à l'oscilloscope pour le signal de référence et à la carte pour la réception des interruptions par celle ci.

---

2. interface de programmation d'applications.

3. passage de la broche de la valeur 0V à +3.3V

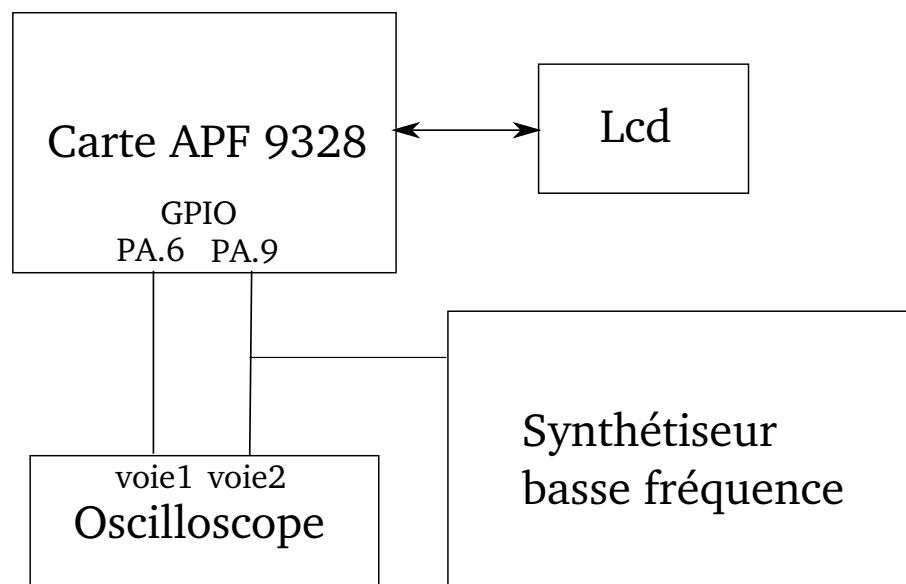


FIGURE 5.1 – Schéma de principe pour les exemples utilisant un oscilloscope

### 5.1.1 Attente sur un compteur

*Linux* offre deux solutions pour endormir un processus pendant une durée donnée :

- la fonction *sleep* qui bloque et endort le processus pendant le temps fourni,
- l'utilisation d'un compteur, qui génère un signal quand la temporisation configurée expire.

Bien qu'ayant réalisé des tests avec les deux solutions, seuls les exemples basés sur le compteur seront présentés. Étant plus précis, les compteurs permettent d'obtenir des valeurs les plus proches entre temps partagé et temps réel.

Le fonctionnement de ces applications est le suivant :

- l'application ou le pilote se met en attente de l'expiration d'un compteur,
- lors de la réception du signal, le programme change l'état d'une broche raccordée à une voie de l'oscilloscope,
- une fois le changement d'état réalisé, le processus se remet en attente.

La courbe idéale est présentée Fig. 5.2

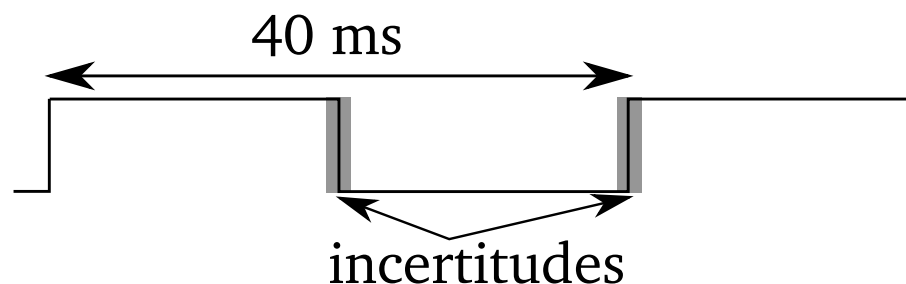


FIGURE 5.2 – Courbe théorique de génération de signal pour un compteur

Les zones en gris sur la Fig. 5.2 présentent les incertitudes dans le changement d'état de la broche.

Les premiers codes présentent l'implémentation en espace utilisateur du fonctionnement décrit précédemment, la version en espace noyau sera vue ensuite.

```
void isr(int signum) {
    [...]
}

int main(int argc, char **argv) {
    struct sigaction sa;
    struct itimerval timer;

    [...]

    memset(&sa, 0, sizeof(sa));
    sa.sa_handler = &isr;
    sigaction(SIGVTALRM, &sa, NULL);
    timer.it_value.tv_sec = 0;
    timer.it_value.tv_usec = SLEEP;
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = SLEEP;
    setitimer(ITIMER_VIRTUAL, &timer, →
        ↪NULL);
    while(1);
    return 0;
}
```

(a) *Linux*

```
RTTASK blink_task;

void blink(void *arg){
    [...]
    rt_task_set_periodic(NULL, →
        ↪TMNOW, TIMESLEEP*1000);

    while(1){
        rt_task_wait_period(NULL);
        [...]
    }
}

int main(int argc, char **argv) {
    [...]

    rt_task_create(&blink_task, "→
        ↪blinkLed", 0, 99, 0);
    rt_task_start(&blink_task, &blink, →
        ↪NULL);
    pause();
    rt_task_delete(&blink_task);
    return 0;
}
```

(b) *Xenomai*

FIGURE 5.3 – Code de gestion de compteur, en espace utilisateur.

Les codes donnés Fig. 5.3 présentent la gestion d'un compteur selon que l'application soit en espace utilisateur *Linux* ou *Xenomai*.

Il est toutefois à noter que dans le cas de *Xenomai*, le même résultat peut être obtenu de plusieurs manières selon le *skin* employé, il est par exemple possible d'avoir un code presque identique à celui pour *Linux* en utilisant le *skin POSIX*.

La *RT\_TASK* créée permet de transférer l'exécution du code dans l'espace de *Xenomai*.

Les résultats présentés Fig. 5.6 entre une application *Linux* et *Xenomai* permettent de constater une différence évidente.

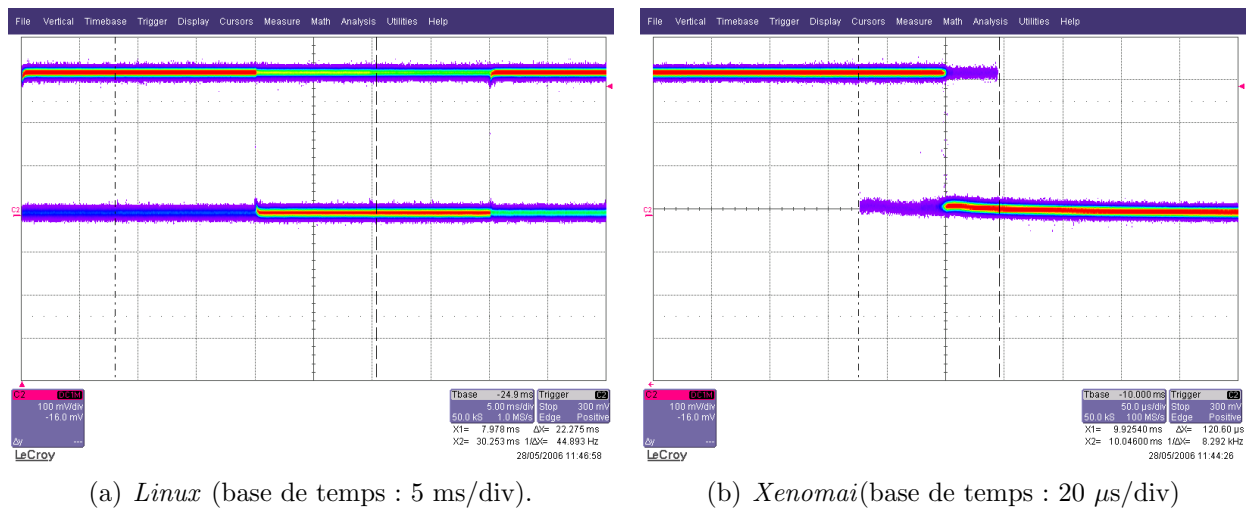


FIGURE 5.4 – Génération d'un signal en espace utilisateur.

En effet, dans le cas de *Linux*, déterminer les bornes minimum et maximum est difficile voire impossible. Il est tout au plus possible de déterminer que globalement le changement d'état se fait le plus souvent toutes les 20 ms mais la couleur vert-bleu tend à prouver qu'il est également fréquent que le changement soit fait avant ou après les 20 ms.

Dans le cas de *Xenomai*, en revanche, il est possible de dire que l'imprécision pour le changement d'état est de l'ordre de  $-30 \mu$ s à  $+20 \mu$ s autour du point idéal.

Le second jeu d'exemples (Fig. 5.5) concerne le même comportement que précédemment mais en espace noyau.

```

/* timer callback*/
void timer_isr(unsigned long arg) {
    [...]
    add_timer(&mt);
}

/* loading (insmod) */
static int __init blink_init(void) {
    current->state = →
        ↪TASK_INTERRUPTIBLE;
    init_timer(&mt);
    mt.expires = jiffies + TS;
    mt.data = (unsigned long) current;
    mt.function = fonctionTimer;
    add_timer(&mt);
    [...]
}
[...]
```

(a) *Linux*

```

[...]
```

```

static rtdm_task_t blink_task;
void timer_isr(void *arg){
    [...]
    while(!end){
        rtdm_task_wait_period();
        [...]
    }
    [...]
}

static int __init p_init(void) {
    [...]
    return rtdm_task_init(&blink_task,
        "blink", blink, NULL,
        99, TIMESLEEP*1000);
}

static void __exit p_exit(void) {
    [...]
    rtdm_task_join_nrt(&blink_task →
        ↪,1000);
    [...]
}

```

(b) *Xenomai (skin RTDM)*

FIGURE 5.5 – Gestion de compteur en espace noyau

L'implémentation sous *Linux* est différente. Par contre elle est assez semblable sous *Xenomai*, la différence se faisant principalement au niveau du nom des méthodes employées.

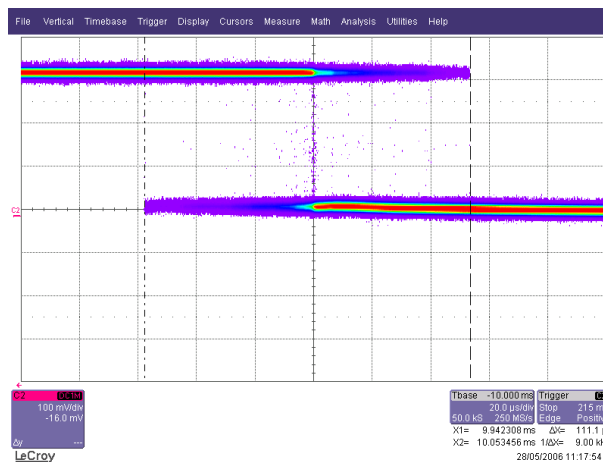
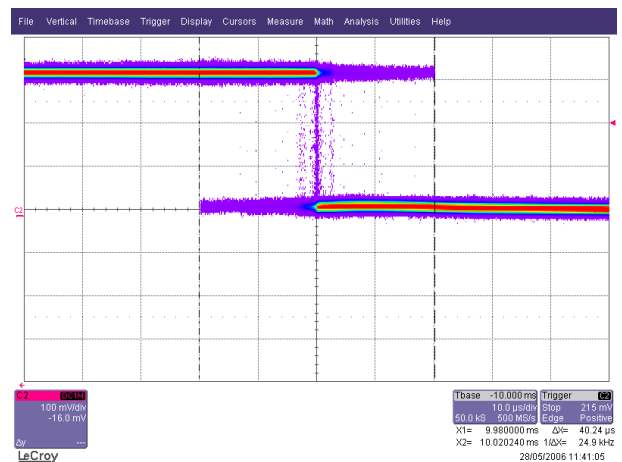
(a) *Linux* : Base de temps 20  $\mu\text{s}/\text{div}$ (b) *Xenomai* : Base de temps 10  $\mu\text{s}/\text{div}$ 

FIGURE 5.6 – Génération d'un signal en espace noyau.

Comme le présente la Fig. 5.6, il est beaucoup plus facile d'évaluer les résultats. Le driver *Linux* présente une imprécision de l'ordre de  $\pm 60 \mu\text{s}$ , toutefois bien supérieure

au +/-  $20\mu\text{s}$  du même driver sous *Xenomai*. Ces valeurs sont relativement imprécises car les couleurs tendent à montrer que dans le cas de *Linux*, les imprécisions sont bien plus fréquentes que pour *Xenomai*.

### 5.1.2 Gestionnaire d'événements

La gestion d'une interruption matérielle est globalement équivalente au cas de l'attente sur un compteur. Comme dans le cas précédent, selon le cas, un gestionnaire est mis en place ou le programme se met en attente sur une fonction bloquante.

Le processus sera averti du changement d'état (front montant) du signal de référence, à ce moment il change l'état d'une broche du processeur. La Fig. 5.7 présente le cas théorique. La zone grise correspond à la latence de gestion.

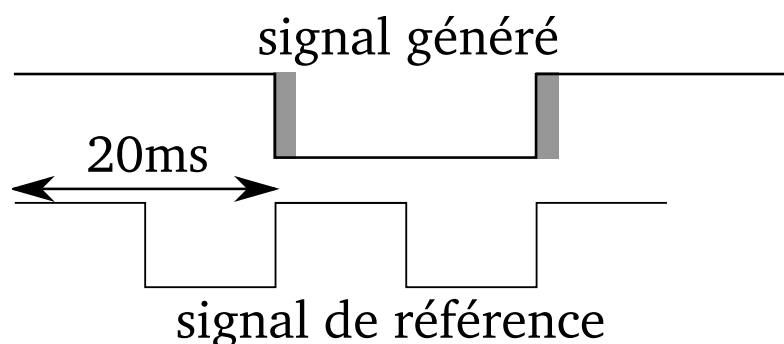


FIGURE 5.7 – Courbe théorique de gestion d'une interruption

Dans les figures présentées ci-après, le fait que le signal présentant des instabilités soit celui de référence et que le signal semble avoir de l'avance s'explique par le fait que l'oscilloscope se déclenche sur le front montant du signal généré par les applications.

Le code réalisé pour la mise en œuvre de cette série d'exemples est globalement relativement proche de ceux présentés précédemment. Seuls les points importants seront fournis.

Dans le cas d'une application en espace utilisateur *Linux*, la gestion d'une interruption se fait en ouvrant un descripteur de fichiers (méthode `open()`) sur un *device*, l'attente se faisant sur `read()`. Pour *Xenomai*, l'accès se fait également sur un descripteur de fichier ouvert avec `open()` mais l'attente se faisant avec `rt_intr_wait()`. Toutefois le principe est le même dans les deux cas.

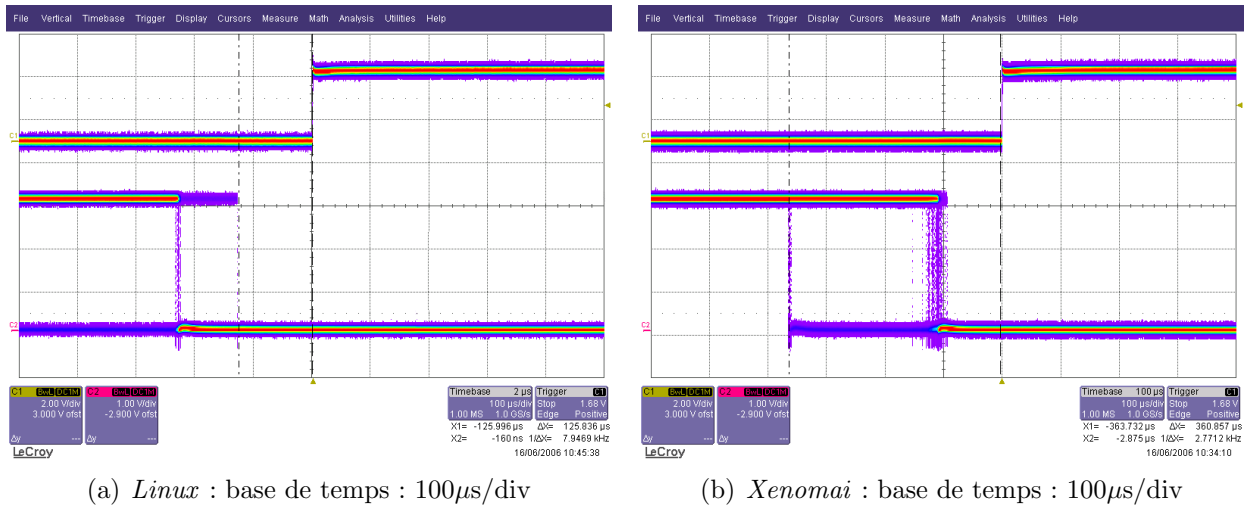


FIGURE 5.8 – Gestion d’interruption en espace utilisateur, courbe du bas : signal de référence.

En regardant les courbes de la Fig. 5.8, il est possible de constater que les valeurs minimums sont globalement proches entre *Linux* et *Xenomai* (125 µs contre 98 µs). Par contre dans le cas où la charge augmente, *Xenomai* présente une plus grande tolérance (de l’ordre de 360 µs ; pour *Linux* la valeur ne pouvant être visible).

Dans le cas de la gestion d’interruption en espace noyau, *Linux* nécessite `request_irq()` avec le numéro de l’interruption et un pointeur sur une fonction appelée lors de l’arrivée de l’interruption.

Pour *Xenomai* le principe est globalement le même. Il nécessite la déclaration d’une variable de type `RT_INTR`. La réservation de l’interruption utilise `rt_intr_create()` qui prend la variable précédemment déclarée, un pointeur de fonction et le numéro de l’interruption. Ensuite il est nécessaire d’activer l’interruption à l’aide de `rt_intr_enable()`.

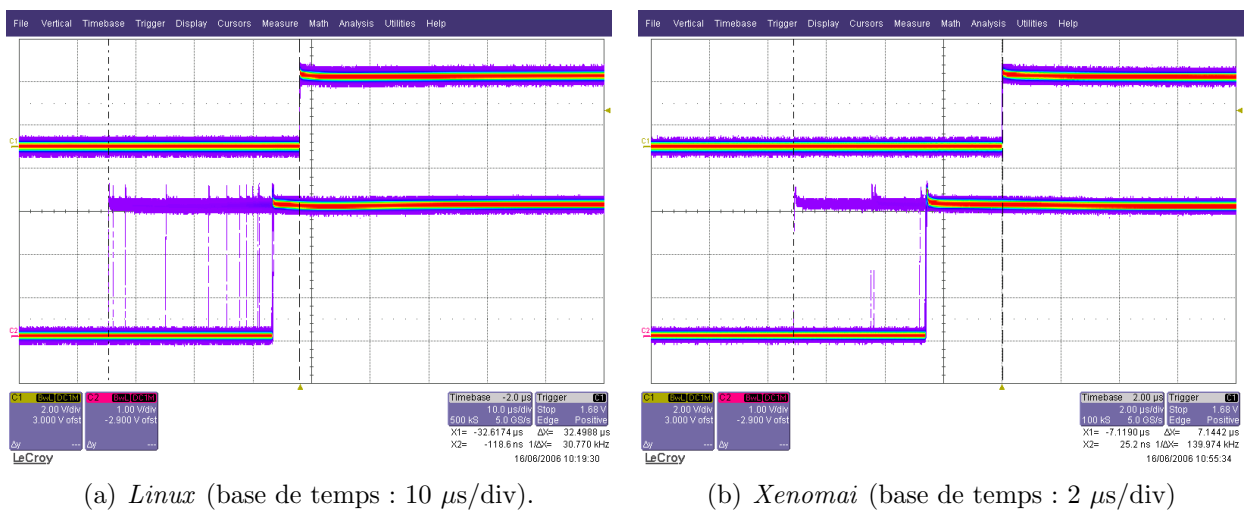


FIGURE 5.9 – Gestion d’interruption en espace noyau, courbe du bas : signal de référence.

Dans le cas des pilotes (Fig. 5.9), *Xenomai* offre des latences minimales 2 fois plus

faibles que *Linux* ( $2.5 \mu\text{s}$  contre  $5 \mu\text{s}$ ). Le cas des valeurs maximales est toutefois bien plus intéressant, en effet *Xenomai* présente une tolérance bien plus forte avec des latences 4,5 fois plus faibles que *Linux* ( $7 \mu\text{s}$  contre  $32 \mu\text{s}$ ).

## 5.2 Fractale de Newton

Afin d'être en mesure de pouvoir avoir une application entraînant une forte charge de l'OS, avec des caractéristiques prédictives et constantes, en terme de consommation de ressource au niveau du processeur et de l'ordonnanceur, il fallait une application mettant en œuvre un calcul complexe et ayant plusieurs instances. Pour cela, une application calculant le polygone  $z^3 - 1$ ,  $z \in \mathbb{C}$  a été réalisée. Ce type de calcul se prête idéalement à une implémentation en parallèle. D'autre part, le résultat obtenu étant affiché sur un écran, le passage des données entre espace utilisateur et espace noyau entraîne des interruptions logicielles, mettant en œuvre l'ordonnanceur.

Du point de vue de l'implémentation, le processus lancé par l'utilisateur en ligne de commande, crée, grâce à la commande `fork()` 10 fils, chacun d'eux ayant un volume de calcul équivalent. Une fois la portion de calcul réalisée, le résultat est renvoyé, à l'aide d'un *tube*, au processus père qui l'affiche sur l'écran LCD à travers un *framebuffer*. Le résultat obtenu est présenté Fig. 5.10.



FIGURE 5.10 – Rendu du calcul de la fractale de Newton sur l'écran LCD connecté à la carte *ARMadeus*

## 5.3 Résultats

Du point de vue des réactions d'une application ou d'un module à l'arrivée d'un événement ou sur le temps que met une application à réaliser son traitement lorsqu'elle

est réactivée, *Xenomai* semble, selon les valeurs obtenues par l'utilisation de l'oscilloscope, donner des résultats bien meilleurs que *Linux* dans le cas d'une forte montée en charge du système. D'une part les valeurs ont des bornes plus faibles mais d'autre part le pourcentage de dépassement est plus faible.

Pour rappel, les valeurs sont :

1. pour les compteurs :
  - en espace utilisateur, non défini pour *Linux* et de  $-20 \mu\text{s}$  à  $+30 \mu\text{s}$  pour *Xenomai*.
  - en espace noyau,  $\pm 60 \mu\text{s}$  pour *Linux* contre  $\pm 20 \mu\text{s}$  pour *Xenomai* (3 fois moins).
2. pour les interruptions :
  - en espace utilisateur, au mieux  $125 \mu\text{s}$  pour *Linux* contre  $98 \mu\text{s}$  pour *Xenomai* qui peut monter jusqu'à  $360 \mu\text{s}$ .
  - en espace noyau, *Linux* offre des latences de  $5 \mu\text{s}$  à  $32$ , alors que *Xenomai* offre des latences de  $2,5 \mu\text{s}$  à  $7 \mu\text{s}$ .

Du point de vue de la programmation, ces exemples ont montré que cet environnement ne présente pas une plus grande difficulté en terme de programmation que *Linux*, permettant de surcroît d'utiliser les pilotes de celui-ci. Les applications réalisées sont disponibles dans l'outil d'*ARMadeus*, la façon de les compiler et les utiliser a fait l'objet d'une documentation dans le wiki [xenomai\_examples]. La programmation sous *Xenomai* (skin native) est présentée à travers un tutoriel [xenomai\_led].

Toutefois ces résultats n'étant que qualitatifs et ne pouvant être déterminés que par la différence de teintes des courbes affichées sur l'oscilloscope, ils doivent être affinés par la mise en œuvre d'une solution apte à fournir des valeurs numériques plus précises. Cet outil, reposant sur l'utilisation du *FPGA*, est détaillé dans le chapitre suivant.

# Chapitre 6

## Évaluation grâce au FPGA

### 6.1 Présentation

L'oscilloscope a permis de mettre en évidence les différences quant à la stabilité des temps d'exécution dans le cas d'un système stressé sous *Linux* et avec *Xenomai*. Mais s'il est possible de constater qualitativement cette situation, il n'est en revanche pas possible de quantifier avec précision le minimum et le maximum de ses temps.

Afin de pouvoir qualifier et exploiter les durées, ainsi que de pouvoir, pour une application donnée, déterminer ses temps d'accès, l'utilisation d'un programme qui compterait le temps au niveau du système n'est pas viable.

Les cartes processeur *ARMadeus* contiennent, comme présenté en section 1.3, un *FPGA* connecté au processeur (Fig. 2.1). Ce composant, contrairement aux composants classiques ayant un fonctionnement défini d'origine, est reprogrammable. Ainsi, il est possible avec un langage de haut niveau (*VHDL*) et après synthèse de ce code, d'avoir un composant dont le fonctionnement est spécifique à un besoin précis. Il est donc possible d'implémenter un compteur tel que celui nécessaire sans pour autant être gêné par les contraintes liées au système d'exploitation.

C'est donc à l'aide du *FPGA*, utilisant une horloge à 96 MHz, que le compteur sera réalisé.

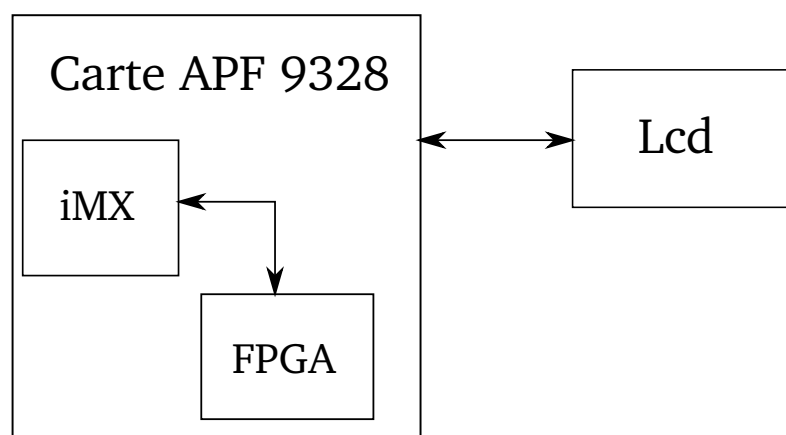


FIGURE 6.1 – Schéma de principe pour les tests utilisant le *FPGA*

Comparé au câblage (Fig. 5.1) nécessaire pour les exemples présentés au chapitre 5, le schéma (Fig. 6.1) relatif aux exemples suivants est bien plus simple car ne nécessite plus le moindre composant extérieur.

Globalement, le fonctionnement est le suivant :

- pour chaque application ou fonction devant être testée, un identifiant unique est attribué. Celui-ci peut être défini d’avance à partir d’une application dédiée à ce travail ou bien directement par l’application à tester.
- lors du lancement de l’application, celle-ci interroge un pilote afin d’obtenir l’identifiant, ainsi que l’ensemble des informations nécessaires pour dialoguer avec le module sur le *FPGA*,
- à travers l’application permettant de configurer le *FPGA* ou bien directement par l’espace utilisateur, le *FPGA* est mis dans un état qui le rend prêt à compter les durées.
- l’application à tester se sert de deux commandes : *START* et *STOP* afin de lancer et d’arrêter le comptage.

Ce mécanisme, du point de vue développement, comporte plusieurs éléments :

- Deux Composants *VHDL*, l’un servant à compter et stocker la durée écoulée entre le *START* et le *STOP*. Et le second qui aura pour but de générer un événement matériel pour l’application à tester et le *START* pour le composant *VHDL* précédemment cité.
- Un pilote *Linux*, permettant de configurer les composants (limite maximale, type de traitement, déclaration de l’application ou de la fonction à tester), de remettre à zéro les compteurs et d’obtenir les informations stockées (durée totale de test, nombre de valeurs obtenues, valeur minimum et maximum, nombre de dépassement).
- Un ensemble de MACROS insérables dans les applications, devant être les plus atomiques possibles. Celles-ci servent à obtenir les informations nécessaires, pour se connecter au *FPGA* et pour l’envoi des *START* et *STOP*. Elles doivent, en plus, être totalement transparentes, les plus simples possibles à mettre en place, désactivables lors de la compilation de l’application et se comporter de la même manière pour un test en espace utilisateur ou espace noyau.

## 6.2 Mise en œuvre

Nous commencerons la présentation par la partie module noyau pour continuer sur les MACROS et nous finirons cette présentation par les IPs VHDL.

### 6.2.1 Pilote

L’accès se fait selon deux techniques :

La première permet à l’utilisateur :

- de fixer les paramètres de configurations globalement tels que le facteur de division de l’horloge du FPGA (*prescaler*) ou la limite de durée maximale ne devant pas être dépassée par un code (*limit*).
- de déclarer un code à tester afin d’attribuer un identifiant.

- d’obtenir les informations acquises après un comptage à travers la sous arborescence relative à une *ID*, ainsi que les informations relatives au contexte de test.

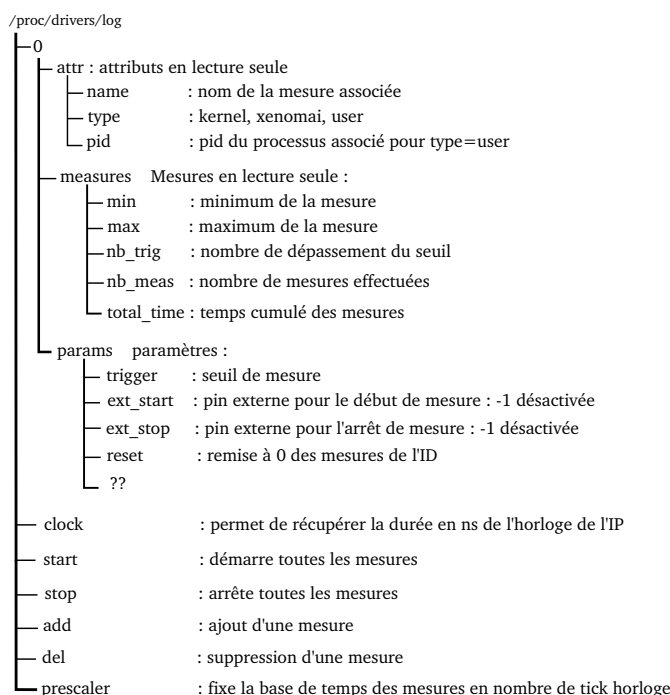


FIGURE 6.2 – Structure de l’arborescence pour l’accès depuis le système de fichier

Cet accès se fait à travers une sous arborescence dans le répertoire */proc/drivers*. La Fig.6.2 présente la structure de cette arborescence. Le noeud **0** correspondant à l’arborescence spécifique à l’identifiant d’une mesure.

La seconde technique concerne l’accès pour l’application à tester. Le pilote, lors de son chargement, crée un noeud *logger* dans le répertoire *logger*.

L’application testée ouvre ce fichier et emploie des constantes avec `ioctl()` sur le descripteur de fichier ouvert.

Les constantes sont les suivantes :

- **LOGGER\_GET\_ID** pour obtenir l’identifiant relatif au nom passé en paramètre ou le cas échéant s’en faire attribuer un.
- **LOGGER\_MEMBASE** pour obtenir l’adresse de base qui sera nécessaire pour l’accès direct au FPGA.

Ce mode d’accès est détaillé dans la section 6.2.2.

## 6.2.2 MACROS

Les MACROS ont pour but de pouvoir complètement cacher le code nécessaire à l’exploitation des tests. Elles doivent être activables ou non grâce à une option à la compilation et pouvoir être adaptées à un emploi en mode utilisateur et en mode noyau.

Le fonctionnement de celles-ci est le suivant. Pour chaque portion de code à tester, hors de toutes fonctions une première MACRO sert à définir une variable sur une structure qui va contenir l’ensemble des informations nécessaires.

Dans une fonction du programme (idéalement *main()*) sont appelées ensuite :

- OPEN\_LOGGER servant à ouvrir un descripteur de fichier sur /dev/logger pour l'obtention ensuite des informations liées au test et sur /dev/mem pour l'accès à la zone de mémoire partagée entre le microprocesseur et le FPGA.
- CREATE\_ALL(nom) appelée pour chaque cas de test (nom étant lié à un identifiant unique). Cette fonction à travers ioctl avec les constantes présentées précédemment remplit une structure avec toutes les informations nécessaires.

Ensuite les fonctions à tester font appels à :

- FPGA\_START pour démarrer le comptage.

```
#define FPGA_START(nom)    ({*(unsigned short*)((nom).ptr+(0x22))=(→
    ↪unsigned short)1;})
```

- FPGA\_STOP pour arrêter le comptage.

```
#define FPGA_STOP(nom)    ({*(unsigned short*)((nom).ptr+(0x24))=(→
    ↪unsigned short)1;})
```

### 6.2.3 Composants VHDL

Le VHDL est un langage permettant la description de systèmes matériels avec un haut niveau d'abstraction. Une fois le code réalisé, il est possible de le synthétiser afin de générer un code binaire qui, une fois mis en place dans le FPGA, lui donne le fonctionnement spécifié.

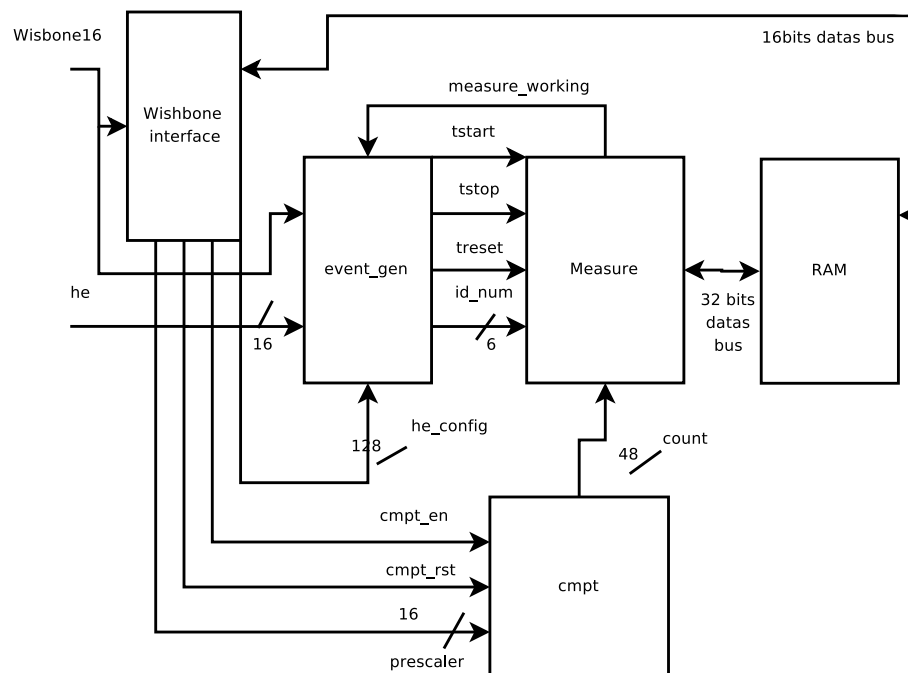


FIGURE 6.3 – Schéma global de l'outil de Benchmarking VHDL

Le schéma Fig.6.3 présente l'ensemble des composants VHDL devant être mis en œuvre dans le cadre de la réalisation de l'outil au niveau FPGA. Nous allons présenter plus en détail les parties les plus importantes, à savoir *Measure* et *event\_gen*, dans les sections suivantes. Les autres composants ne seront que mentionnés.

Pour pouvoir compter le temps pour plusieurs tests d'une manière simple, un compteur général (*cmpt* Fig. 6.3) sert à comptabiliser le temps depuis que la commande *START* a été envoyée. Ce compteur est incrémenté à chaque front d'horloge, modulo un *prescaler* servant à spécifier une valeur de base de temps.

*wishbone* permet le dialogue entre les divers *Composants* et le microprocesseur sur un principe d'aiguillage de données.

### Composant Measure

Le compteur devant être capable de pouvoir mémoriser les résultats obtenus pour plusieurs codes en même temps, le stockage s'effectue dans un bloc de RAM (composant *RAM*).

La Fig. 6.4 présente la machine d'états de ce composant.

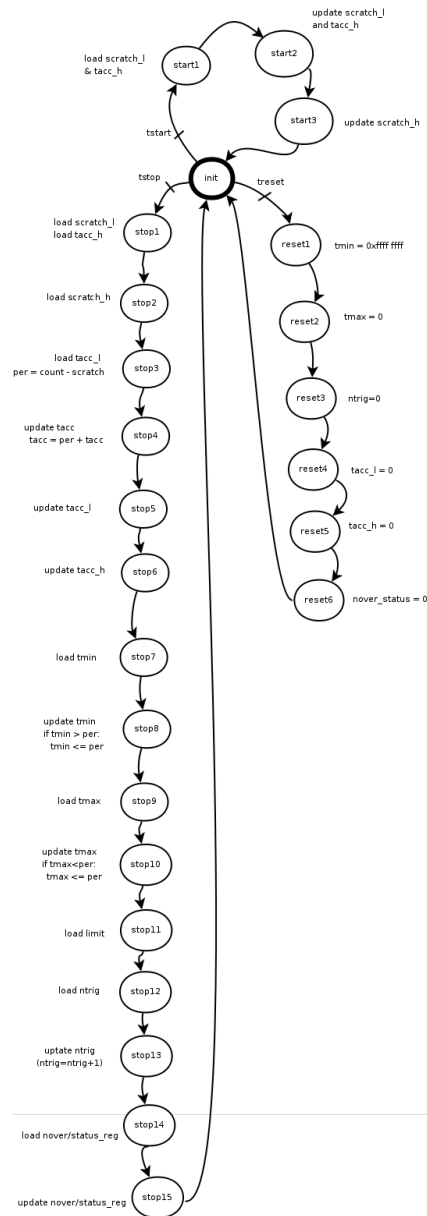


FIGURE 6.4 – Machine d'états de l'IP Logger

Son fonctionnement pour une application est le suivant :

- Lorsqu'une application émet l'événement *START*, le compteur stocke dans la RAM la valeur courante du compteur général (état *start\**).
- Lorsque l'application émet l'événement *STOP*, le compteur fait la différence entre la valeur courante du compteur général et celle stockée. Cette différence est ensuite comparée à la valeur minimale, maximale et le cas échéant à la valeur maximale de dépassement (*limit*) (état *stop\**).
- Lors du *STOP*, le compteur incrémente également le nombre de cycle et ajoute la durée calculée pour obtenir la valeur globale de la durée de test.
- Lors de la réception de l'évènement *RESET*, il remet l'ensemble des « variables » à leurs valeurs initiales.

Les valeurs peuvent être obtenues directement dans la RAM, en accédant à des registres

donnés.

### IP `event_gen`

C'est un générateur de signaux arbitraires. Celui-ci permet théoriquement de créer n'importe quel type de signaux ou de séquences. La fréquence est réglable et dans le cas des séquences il est possible de fournir le *pattern*<sup>1</sup> du signal.

Le signal est stocké en *RAM*, lu en bouclant à la fin de la forme et incrémenté à chaque signal d'horloge avec comme dans le cas du *Logger* un diviseur.

Ce composant réalise la commande *START*, en lieu et place de l'application à tester. Il génère également le signal sur une broche du *FPGA*, connectée à une broche du microprocesseur.

## 6.3 Avancement

Une première implémentation des spécifications du compteur VHDL a été réalisée. Celle-ci ne permet d'exploiter qu'une seule ID bien que son mécanisme global correspond à celui stipulé dans le cahier des charges. Par ailleurs, le cas de la génération de signaux n'est pas encore disponible.

Toutefois, les aspects liés au système (driver, MACROS) sont disponibles avec la gestion de plusieurs codes à tester et de l'ensemble des configurations.

Il a d'ailleurs été possible de réaliser plusieurs tests basés sur les applications réalisées dans la précédente étape, simplement en ajoutant le code nécessaire.

Bien que les tests aient permis d'obtenir des valeurs plus fines que celles acquises à l'oscilloscope et bien que globalement elles semblent bonnes et concorder avec les observations qualitatives, mais elles n'ont pu être suffisamment vérifiées pour être présentées.

A l'heure actuelle, cette étape n'est pas encore achevée. Il reste à remplacer la première implémentation *VHDL* par celle définitive. Les simulations pour celle-ci doivent être encore complétées du point de vue comportemental et temporel. Le pilote et les MACROS nécessitent encore une phase de validation et de corrections de *bugs* connus.

---

1. forme

# Troisième partie

## Bilan

# Chapitre 7

## Conclusion

Plusieurs bilans peuvent être présentés : d'une part concernant *Xenomai*, en tant que solution temps réel dur et d'autre part sur l'apport personnel et les objectifs du stage.

### 7.1 *Xenomai*

Ce système offre une véritable solution de comportement temps réel sans pour autant limiter l'utilisation d'applications classiques dans un environnement en temps partagé. Ceci permettant d'éviter le surcoût lié intrinsèquement au changement de système d'exploitation qui serait entraîné par la mise en œuvre d'un système d'exploitation temps réel complet.

Dans le cadre du portage d'une application, originellement conçue pour un système propriétaire, le coût est également réduit par la mise à disposition d'une interface émulant le fonctionnement de l'*OS* d'origine.

Pour le développeur confronté à la nécessité de créer une application ou un module temps réel, l'apprentissage de l'*API* de *Xenomai* est relativement aisé et ne le dépayse que peu, permettant de réduire le temps nécessaire au développement.

### 7.2 Apports personnel

Bien qu'ayant déjà eu par le passé l'occasion d'être confronté à du développement système, d'avoir eut la possibilité de réaliser des modules noyaux et d'avoir travaillé sur des plateformes embarquées, ce stage m'a permis de pouvoir renforcer, encore, mes connaissances dans ce domaine. Le *bug* lié au patch *ADEOS* qui empêchait le démarrage du système d'exploitation au tout début du stage, m'a forcé à me plonger dans l'initialisation de l'*OS*, m'apportant une bonne compréhension de cette étape importante.

Le travail réalisé m'a également permis de pouvoir acquérir une meilleure connaissance et compréhension du langage *VHDL* et de l'utilisation d'un *FPGA*. Apport très important en terme de fonctionnalités car permettant d'élargir grandement les capacités offertes par le système classique.

J'ai par ailleurs put découvrir d'une manière bien plus précise, que par le passé, les contraintes en terme de temps réel et de mieux comprendre les détails des mécanismes permettant aux systèmes d'exploitations d'offrir à l'utilisateur une impression de parallélisme

dans l'exécution des applications sur son ordinateur. Ce sujet est complexe car basé sur de nombreuses contraintes, algorithmes et détails d'implémentation.

### 7.3 Objectifs

Le stage comportait trois objectifs principaux :

L'intégration de *Xenomai* dans *Buildroot* rend sa mise en œuvre presque triviale pour n'importe qui souhaitant exploiter cette solution. Un développeur possédant une connaissance en temps réel n'a pas forcément les connaissances ou le temps pour mettre en place cette solution manuellement. C'est pourquoi ce *package*, en cachant l'ensemble des détails complexes de son installation, présente un réel intérêt. Il faut, également, rappeler que lors de la mise en place manuelle une impossibilité de faire fonctionner *Xenomai* est apparue. Suite à cela un patch a été réalisé et proposé à la communauté *ADEOS* et *Xenomai*, permettant de rendre fonctionnel ces extensions sur l'ensemble des plateformes exploitant ce type de processeur. Le *package* sera, dans le futur, proposé à la communauté *Buildroot* afin de simplifier l'installation sur l'ensemble des plateformes, utilisant cet outil et supportées par *Xenomai* et *ADEOS*.

Les premiers tests réalisés, à travers des applications permettant d'obtenir des résultats visuels à l'oscilloscope, ont permis de démontrer les réels avantages d'une telle solution. En effet, comparé aux capacités de *Linux*, cette solution offre de meilleures performances et surtout une meilleure tolérance dans le cas d'un système stressé.

La dernière phase, concernant la mise en place de vrais outils de tests et d'évaluation, bien qu'encore en cours de développement ont permis de renforcer les premières constatations à travers l'obtention de valeurs quantitatives et non plus seulement qualitatives. Cet outil permet, par ailleurs, d'offrir une solution d'évaluation bien plus simple et efficace, par rapport à celle nécessitant l'utilisation d'un oscilloscope et d'un synthétiseur de signaux basse fréquence. Cette solution, permettra dans le futur d'évaluer, par un jeu de test la non régression de l'extension temps réel et l'évaluation des performances d'une application temps réel afin de déterminer les goulots d'étranglement contenus dans le code, afin de garantir un code optimal et des temps de latences les plus faibles possibles. Ceci sans avoir à modifier le noyau ni mettre en œuvre une solution trop complexe, hors d'atteinte de beaucoup de développeurs.

Le travail réalisé pendant le stage donnera lieu, en juillet, à une conférence lors des 10<sup>èmes</sup> **Rencontres Mondiales du Logiciel Libre**<sup>1</sup> à *Nantes*

---

1. <http://2009.rml.info/Xenomai-sur-cible-ARM9-Freescale-i.html?lang=fr>

# Bibliographie

- [linuxKernel] D. P. Bovet & M. Cesati, *Understanding the Linux Kernel.*, O'Reilly (2001)
- [linuxDriver] A. Rubini & J. Corbet, *Linux Device Drivers, 2nd Ed.*, O'Reilly (2001)
- [Ficheux] P. Ficheux & P. Kadionik, *Temps réel sous Linux (reloaded)*, GNU/Linux Magazine France, Hors Série 24 (Fév/Mars 2006)
- [LmNDS] J.-M. Friedt & G. Goavec-Merou, *Interfaces matérielles et OS libres pour Nintendo DS : DSLinux et RTEMS*, GNU/Linux Magazine France, Hors Série 43 (juillet-août 2009)
- [xenomai\_manuel] Méthode manuelle pour l'installation de Xenomai sur APF9328.  
[http://www.armadeus.com/wiki/index.php?title=Xenomai\\_manual\\_installation](http://www.armadeus.com/wiki/index.php?title=Xenomai_manual_installation)
- [xenomai\_auto] Méthode pour l'installation de Xenomai à l'aide de *Buildroot*.  
<http://www.armadeus.com/wiki/index.php?title=Xenomai>
- [xenomai\_examples] Utilisation des exemples de démonstration temps réels .  
[http://www.armadeus.com/wiki/index.php?title=Xenomai:examples\\_usage](http://www.armadeus.com/wiki/index.php?title=Xenomai:examples_usage)
- [xenomai\_led] Création d'une application faisant clignoter un led sous Xenomai.  
[http://www.armadeus.com/wiki/index.php?title=Xenomai:Blinking\\_LEDs](http://www.armadeus.com/wiki/index.php?title=Xenomai:Blinking_LEDs)
- [buildroot\_mk] Intégration d'une application dans Buildroot.  
[http://www.armadeus.com/wiki/index.php?title=Buildroot\\_Packages](http://www.armadeus.com/wiki/index.php?title=Buildroot_Packages)
- [XenoApi] API de Xenomai.  
<http://www.xenomai.org/documentation/branches/v2.4.x/html/api/>
- [PODWiki] Peripherals On Demand.  
Wiki sur POD.  
[http://www.armadeus.com/wiki/index.php?title=Peripherals\\_On\\_Demand](http://www.armadeus.com/wiki/index.php?title=Peripherals_On_Demand)
- [bookVHDL] R. Airiau & J. M. Bergé & V. Olive & J. Rouillard, *VHDL, langage, modélisation, synthèse. 2nd Ed.*, Presses Polytechniques et Universitaires Romandes (1998)
- [bookFPGA] P. P. Chu, *FPGA Prototyping by VHDL Examples : Xilinx Spartan-3 Version*, Wiley (2008)

# Quatrième partie

## Annexes

# Annexe A

## Code

### A.1 Script d'intégration de Xenomai dans le Buildroot d'Armadeus.

```
#####  
#  
# xenomai :  
# URL   : http://xenomai.org  
# NOTE  : Real-Time Framework for Linux  
#  
# adeos :  
# URL   : http://home.gna.org/adeos/  
# NOTE  : The purpose of Adeos is to provide a flexible  
#         environment for sharing hardware resources among  
#         multiple operating systems, or among multiple  
#         instances of a single OS.  
#  
#####  
  
KERN_DIR:=$(PROJECT_BUILD_DIR)/linux-$(shell echo $(BR2_LINUX26-VERSION))  
XENOMALVERSION:=2.4.7  
XENOMALSOURCE:=xenomai-$(XENOMALVERSION).tar.bz2  
XENOMALSITE:=http://download.gna.org/xenomai/stable  
XENOMALDIR:=$(BUILD_DIR)/xenomai-$(XENOMALVERSION)  
XENOMALCAT:=bzcat  
XENOMALBINARY:=xeno-load  
XENOMALTARGET.BINARY:=usr/xenomai/bin/xeno-load  
  
ADEOS.VERSION:=1.12-00  
ADEOS.SOURCE:=adeos-ipipe-$(shell echo $(BR2_KERNEL_THIS_VERSION))-arm-$(→  
↪ADEOS.VERSION).patch  
ADEOS_SITE:=http://download.gna.org/adeos/patches/v2.6/arm/older/  
  
$(DL_DIR)/$(XENOMALSOURCE) :  
$(WGET) -P $(DL_DIR) $(XENOMALSITE)/$(XENOMALSOURCE)  
  
$(DL_DIR)/$(ADEOS.SOURCE) :  
$(WGET) -P $(DL_DIR) $(ADEOS_SITE)$(ADEOS.SOURCE)
```

```

xenomai-source: $(DL_DIR)/$(ADEOS.SOURCE)

$(KERN_DIR)/.patched.adeos: $(DL_DIR)/$(ADEOS.SOURCE)
    toolchain/patch-kernel.sh $(LINUX26_DIR) package/xenomai \
        00-adeos-compatibility_with_armadeus.patch
    toolchain/patch-kernel.sh $(LINUX26_DIR) $(DL_DIR) \
        $(ADEOS.SOURCE)
    toolchain/patch-kernel.sh $(LINUX26_DIR) package/xenomai \
        01-adeos-$(ADEOS.VERSION)-imx-compatibility.patch
    toolchain/patch-kernel.sh $(LINUX26_DIR) package/xenomai \
        02-xenomai-imx-gpio-set-value-inline.patch
    touch $@

$(XENOMALDIR)/.unpacked: $(DL_DIR)/$(XENOMALSOURCE)
    $(XENOMALCAT) $(DL_DIR)/$(XENOMALSOURCE) | tar -C $(BUILD_DIR) $(→
        ↪TAR_OPTIONS) -
    touch $@

$(KERN_DIR)/.patched.xenomai: $(KERN_DIR)/.patched.adeos $(XENOMALDIR)/.→
    ↪unpacked
    $(XENOMALDIR)/scripts/prepare-kernel.sh \
        --linux=$(LINUX26_DIR) \
        --arch=$(BR2_ARCH)
    cat package/xenomai/xeno-kernel.config >> $(LINUX26_DIR)/.config
    touch $@

xenomai-patch-kernel: $(KERN_DIR)/.patched.xenomai

$(XENOMALDIR)/.configured: $(KERN_DIR)/.patched.xenomai
    (cd $(XENOMALDIR); rm -rf config.cache; \
        $(TARGET_CONFIGURE_OPTS) \
            $(TARGET_CONFIGURE_ARGS) \
            CFLAGS_FOR_BUILD="$(HOST_CFLAGS)" \
            ./configure \
            --enable-arm-mach=imx \
            --host=arm-linux \
            --datarootdir=/xenodoc \
        )
    touch $@

$(XENOMALDIR)/scripts/$(XENOMALBINARY): $(XENOMALDIR)/.configured
    $(MAKE) -C $(XENOMALDIR) \
        CC="$(TARGET_CC)" LD="$(TARGET_LD)"

$(TARGET_DIR)/$(XENOMALTARGET_BINARY): $(XENOMALDIR)/scripts/$(→
    ↪XENOMALBINARY)
    echo $@
    $(STAGING_DIR)/usr/bin/fakeroot $(MAKE) -C $(XENOMALDIR) \
        CC=$(TARGET_CC) LD=$(TARGET_LD) \
        DESTDIR=$(TARGET_DIR) install
    rm -rf $(TARGET_DIR)/xenodoc
    echo "/usr/xenomai/lib" > $(TARGET_DIR)/etc/ld.so.conf
    echo -e 'export PATH=/usr/xenomai/bin:${${PATH}}' >> $(TARGET_DIR)/etc/→
    ↪profile

```

```

xenomai: uclibc host-fakeroot kernel-headers $(TARGET_DIR)/$(→
↳XENOMALTARGET.BINARY)

xenomai-clean:
$(MAKE) DESTDIR=$(TARGET_DIR) -C $(XENOMALDIR) uninstall
-$(MAKE) DESTDIR=$(TARGET_DIR) -C $(XENOMALDIR) clean

xenomai-dirclean: xenomai-clean
rm -rf $(XENOMALDIR)

#####
#
# Toplevel Makefile options
#
#####
ifeq ($(strip $(BR2_PACKAGE_XENOMAI)),y)
TARGETS+=xenomai
endif

```

Listing A.1 – fichier d'intégration dans BuildRoot

# Annexe B

## Proposition de planning

### B.1 Phase 1 (1 mois)

- installation de Xenomai / ADEOS sur la carte APF9328 (15j)
- intégration au système de build (Buildroot) Armadeus (5j)
- documentation

### B.2 Phase 2 (2 mois)

- création d'exemples de test pour Xenomai (20j) :
  - génération d'un signal carré par GPIO avec timer soft (= GPIO\_CLOCK)
  - compteur soft de transitions d'un signal carré (par interruption) (= GPIO\_COUNTER)
  - essais avec n x GPIO\_CLOCK + n x GPIO\_COUNTER
- création d'exemples de tests Xenomai + Linux (20j) :
  - test des exemples précédents avec Linux en charge (ex : transfert Ethernet)
  - passage d'infos entre Xenomai et Linux et vice versa :
    - réception RC5 soft dans Xenomai et remontée des infos à Linux
    - envoi RC5 de codes venant d'un prog Linux
- documentation

### B.3 Phase 3 (3 mois)

- création d'une macro, pour les tâches Xenomai, permettant de stocker des infos de mesure/debug (ID unique en fonction du nom de la fonction où se trouve la macro, écriture dans le FPGA) (10j)
- mise en place de la structure FPGA pour stocker ces infos ainsi que les temps d'exécution (collaboration avec Fabien) (35j)
- écriture du logiciel permettant de récupérer les données ainsi stockées (15j)
- documentation

## Résumé

Ce stage s'est déroulé au sein de l'entreprise *ARMadeus Systems*, entreprise basée à Mulhouse, spécialisée dans l'électronique et l'informatique embarqué.

Le but de stage a été dans un premier temps l'intégration d'une solution temps réel dans l'outil de génération du système d'exploitation *Linux*, utilisé sur les plateformes développées par la société.

Une fois cette première étape réalisée la seconde tâche a consisté en l'évaluation des performances de *Xenomai* sur des tests correspondants aux cas critiques d'un système d'exploitation.

Au delà de l'observation qualitative des latences grâce à un oscilloscope, un outil dédié a été développé sur FPGA pour obtenir des données quantitatives.

## Mots clefs

Xenomai, GNU/Linux, ARMadeus Systems, temps réel, ordonnanceur, préemption, temps partagé, priorité, processeur, ARM9, Freescale, i.MXL, FPGA, Xilinx, VHDL, espace utilisateur, espace noyau, interruption, latences, oscilloscope, intégration, qualification, MACRO, tâche, processus, pilote, noyau, matériel, compteur, système d'exploitation

## Abstract

This traineeship was performed with *ARMadeus Systems*, a company based in Mulhouse dedicated to the development of embedded electronics and software.

The purpose of this traineeship was the port of a real time extension – *Xenomai* – of the *Linux* operating system to the platforms developed by the company.

Once this step was functional, the second objective was the evaluation of the performances of *Xenomai* on test cases representative of critical conditions met by the operating system. Beyond the use of an oscilloscope to qualitatively monitor the latencies, a FPGA-based dedicated tool was developed to acquire quantitative data.

## Key words

Xenomai, GNU/Linux, ARMadeus Systems, real time, scheduler, preemption, time sharing, priority, processor, ARM9, Freescale, i.MXL, FPGA, Xilinx, VHDL, user space, kernel space, interrupt, latencies, oscilloscope, integration, qualification, MACRO, task, process, driver, kernel, hardware, timer, operating system