République Tunisienne
Ministère de l'Enseignement Supérieur, de la Recherche Scientifique

Université de Sfax
École Nationale d'Ingénieurs de Sfax

Ecole Doctorale
Sciences et Technologies

Thèse de *DOCTORAT*

ISI

N° d'ordre: 297/2013

THESE

Présentée à

L'École Nationale d'Ingénieurs de Sfax

En vue de l'obtention du

DOCTORAT

Dans la discipline *Génie Electrique*
Ingénierie des Systèmes Informatiques

Par

Mossaad BEN AYED

Environnement de Co-Simulation / Emulation des systèmes Continus/Discrets

Soutenu le 30 Novembre 2013, devant le jury composé de :

M.	**Mongi LAHIANI** (Professeur)	Président
M.	**Mohamed MASMOUDI** (Professeur)	Rapporteur
M.	**Carlos VALDERRAMA** (Professeur)	Rapporteur
M.	**Ashraf SALEM** (Professeur)	Examinateur
M.	**Mohamed ABID** (Professeur)	Directeur de Thèse

Dédicaces

A mon cher père et ma chère mère

pour toute la peine qu'ils se sont donné pour moi,

pour leur amour et leurs encouragements.

Le plaisir que j'ai de leur dédier ce travail n'arrivera nullement à compenser leurs sacrifices qu'ils ont consentis pour m'aider à réussir.

A ma femme HibatAllah qui m'a toujours encouragé,

acceptant tout ce temps soustrait à ma présence auprès d'elle,

Tu es une épouse exemplaire, ton affectation, ton aide et ta sympathie

sont l'essence de ma vie et le garent de ma réussite.

A mes belles filles Houda & Mariem

pour le peu de temps que je leur ai consacré pendant mon travail.

Que Dieu vous garde et vous bénisse.

A toutes la famille …

A mes amis…

A tous ceux que j'aime et qui me sont chers, je dédie ce travail

en témoignage de ma profonde gratitude et inestimable respect.

Remerciements

Je voudrais exprimer ma gratitude et mes remerciements les plus sincères à l'égard de toutes les personnes qui m'ont aidé aussi bien par leur soutient moral que par leur savoir et savoir-faire pour mener à bien ce travail :

Je remercie infiniment mon directeurs de thèse Monsieur Mohamed Abid, professeur à l'École National d'Ingénieur de Sfax et responsable du laboratoire CES. Il m'a toujours traité comme un collègue, un ami, un des leurs enfants. Il a été un vrai plaisir de travailler et vivre avec lui.

Un remerciement particulier à mon co-encadreur Monsieur Faouzi Bouchhima, maitre assistant à l'Institut Supérieur d'Electronique et de Communication de Sfax pour son soutient moral et technique. Merci infiniment pour sa disponibilité et pour le temps de discussion et de réflexion qu'on a passé ensemble.

Mes remerciements s'adressent pareillement à Monsieur Mongi Lahiani, Professeur à l'École National d'Ingénieur de Sfax, pour l'intérêt qu'il a porté à ce travail en acceptant de me faire l'honneur de présider le jury de ma soutenance.

J'adresse également mes sincères remerciements à Monsieur Mohamed Masmoudi, professeur á l'École National d'Ingénieur de Sfax, à Monsieur Carlos Valderrama professeur á l'Université de Mons Belgique, à Monsieur Ashraf Salem professeur à l'Université de Ain Shams et directeur d'ingénierie à Mentor Graphics Egypt pour m'avoir fait l'honneur d'accepter de faire partie de mon jury de soutenance.

Je tiens à remercier très chaleureusement Monsieur Lazhar Ben Hmida et Monsieur Hatem Ben Taher pour leurs rôles de « consultant linguistique » et Monsieur Ismail Ketata pour toutes les compétences techniques partagées.

﴿قُلْ إِنَّ صَلَاتِي وَنُسُكِي وَمَحْيَايَ وَمَمَاتِي

لِلَّهِ رَبِّ الْعَالَمِينَ﴾

الأنعام 162

Table des matières

Liste des figures

Liste des tableaux

Glossaire

API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
CAO	Conception Assistée par Ordinateur
CAN	Convertisseur Analogique Nnumérique
CNA	Convertisseur Numérique Analogique
CSP	Communicating Sequential Processes
CODIS	Continuous DIscrete Simulation
CPLD	Complex Programmable Logic Device
DC	Device Controller
DECOC	Data-driven Error Correcting Output Codes
DSP	Digital Signal Processor
ECOC	Error Correcting Output Codes
EDOs	Ordinary Differential Equations
FAR	False Acceptance Rate
FIFO	First In First Out
FSM	Finite State Machine
FPGA	Field Programmable Gate Array
FRR	False Rejection Rate
GUI	Graphical User Interface
HC	Host Controller

HDL	Hardware Description Languages
HIL	Hardware In the Loop
HSIL	Hardware Software In the Loop
IEEE	Institute of Electrical and Electronics Engineers
ISS	Instruction Set Simulator
IP	Intellectual Property
LSB	Least Significant Bit
OS	Operating System
OTG	On-The-Go
SceMi	Standard CoEmulation Modeling Interface
SDF	Synchronous Dataflow
TLM	Transaction Layer Modeling
PID	Proportional Integral Derivative
RISC	Reduced Instruction Set Computer
RTL	Regiter Transfer Level
VHDL	Very Hardware Description Language
VLSI	Very Large Scale Integration
UML	Unified Modeling Language
USB	Universel Serial Bus
WDK	Windows Driver Kit

INTRODUCTION GENERALE

I. Problématique et Motivation

Dans ces dernières années, la conception des systèmes automatiques-embarqués est devenue de plus en plus complexe. Cette complexité qui est due à l'intégration des composants hétérogènes à un niveau élevé d'abstraction nécessite un nouveau cadre conceptuel pour l'adaptation entre les composants hétérogènes ainsi que des nouvelles méthodologies pour la vérification et la validation. L'hétérogénéité des composants est devenue une nécessité à cause de l'utilisation des modèles en temps continu ainsi que des modèles à événements discrets dans un modèle global, donnant une vue d'ensemble du système. Dans la littérature, plusieurs systèmes hétérogènes (ITRS, 2003), (Senturia S., 1998), (Jie L., 2004) ont été développés. Ce travail s'inscrit dans le domaine de la conception multi-langages des systèmes hétérogènes.

Étant donné l'hétérogénéité des concepts manipulés par ces deux types de modèles, la validation globale demande un environnement spécialisé capable de vérifier le système en cours de développement. En respectant la facilité de la modélisation et les sémantiques nécessaires de chaque modèle (continu et discret), un environnement de co-vérification hétérogène s'impose. Cet environnement de co-vérification met en place des interfaces de simulation / émulation et des modèles de synchronisation entre simulateur-émulateur capable de simuler le modèle continu et le modèle discret.

Ces systèmes hétérogènes ont créé un besoin pour les outils de CAO (Conception Assistée par Ordinateur) capables de vérifier et de valider le comportement du système ainsi conçu. Les environnements de co-vérification éliminent la détection tardive des erreurs et réduisent le temps de conception. Il est donc nécessaire de définir un modèle d'exécution globale dont les éléments de base sont (Bouchhima F., 2005), (Nicolescu G., 2002):

- Les modèles des composants du système hétérogène qui sont décrits en temps continu ou bien dans le domaine à événement discret.
- Les interfaces de co-vérification qui réalisent l'adaptation de chaque modèle au bus de co-vérification, l'adaptation des différents protocoles de communication et la synchronisation entre les deux modèles.

- Le bus de co-vérification qui est responsable de l'interprétation des interconnections entre les deux modèles compose le modèle global.

Les aspects qui rendent difficile la modélisation et la simulation des systèmes continus et discrets sont (Bouchhima F., 2007):

- Pour le modèle discret, le temps est une notion globale pour tous les modules du système, il avance discrètement en passant par les instants discrets définis par les temps de notification des événements discrets. Pour le modèle continu le temps est une variable globale qui avance par le pas d'intégration (fixe ou variable) et qui intervient dans le calcul des signaux.
- Pour le modèle discret, les processus sont sensibles aux événements alors que, pour le modèle continu, les processus sont exécutés à chaque pas d'intégration.
- Pour le modèle discret, la communication est réalisée par des ensembles d'événements alors que pour le modèle continu, la communication est réalisée par des signaux continus (un signal continu possède une valeur à tout instant).
- Chaque modèle doit être capable de détecter, de localiser en temps et de réagir aux événements envoyés par l'autre modèle.

Les techniques de vérification pour le cas des systèmes matériels/logiciels sont déjà matures grâce au nombre de travaux qui sont impliqués (Ismail T.B., 1994), (Valderrama C.A, 1995), (Abid M., 1998).

Cependant, les techniques de vérification sont faiblement exploitées pour le cas des systèmes continus/discrets à cause des difficultés de mise en place des modèles de co-simulation. L'environnement CODIS (Continuous DIscrete Simulation), est le fruit de plusieurs travaux de recherches dans cet axe (Bouchhima F., 2005). CODIS se base en fait sur la synchronisation entre un simulateur continu et un simulateur discret. Cet environnement supporte deux modes de synchronisations : (1) synchronisation complète et (2) synchronisation d'évènements prédictibles. Dans le premier mode, l'environnement CODIS supporte la modélisation conjointe matérielle/logicielle en se basant sur un simulateur de jeux d'instructions (ISS) pour la simulation des applications logicielles. Dans ce cas, la simulation de la partie discrète est lente vue l'utilisation de l'ISS. Dans le deuxième mode, la modélisation de la partie discrète est purement matérielle. En fait, ce mode diminue le temps de simulation mais ne supporte pas la modélisation matérielle/logicielle.

II. Objectifs

Cette thèse présente une extension de l'environnement CODIS (Bouchhima F., 2007) afin de supporter la modélisation matériel/logiciel pour le modèle discret et d'accélérer la vitesse de la simulation d'autre part.

L'environnement cible doit être capable, par co-simulation/émulation, de surmonter le problème de la modélisation hétérogène et multi-niveau des contrôleurs numériques d'une part et de diminuer le temps de simulation en utilisant une carte FPGA à base d'architecture cible d'autre part. L'accélération de la simulation présente un point clé qui impose la création d'interfaces de synchronisation et de communication entre l'environnement de co-simulation et la carte de prototypage FPGA.

Les objectifs de la thèse sont organisés comme suit :

- Proposer un modèle et un environnement de co-simulation multi niveau basé sur les techniques de simulation et d'émulation pour le cas des systèmes matériels/logiciels.
- Proposer un modèle et un environnement de simulation Matériel/Logiciel en boucle (″Hardware Software In the Loop″) pour les systèmes de contrôles.
- Proposer une extension du modèle et de l'environnement CODIS+ assurant la synchronisation entre le simulateur continu d'une part et le simulateur SystemC et une carte FPGA pour le modèle discret d'autre part.
- Valider l'environnement de vérification à travers des exemples d'applications : un système de reconnaissance par empreinte digitale, un régulateur de la vitesse d'un moteur à courant continu, un système de contrôle en boucle fermée de la vitesse d'un moteur et un système limiteur de vitesse pour voiture.

III. Contributions

Ce travail présente cinq contributions :

- ✓ Une étude des environnements de vérifications continus et discrets pour les différents niveaux d'abstraction.

- ✓ Modélisation d'un environnement de co-simulation matériels/logiciels multi niveau tout en respectant à la fois l'accélération du temps de simulation et la description dans le haut niveau. Ainsi, un modèle de communication et de synchronisation entre le simulateur SystemC et une carte FPGA à base d'architecture cible est implémenté.
- ✓ Modélisation d'une interface générique entre le simulateur Simulink et la carte FPGA à base d'architecture cible respectant la simulation Matériel/Logiciel en boucle.
- ✓ Modélisation d'un moteur de synchronisation qui interface et adapte le simulateur du modèle continu avec le simulateur/émulateur du modèle discret. L'environnement ainsi conçu est nommé CODIS+. Le but du moteur de synchronisation est de gérer le simulateur Simulink du domaine continu, le simulateur SystemC et l'architecture cible du domaine discret.
- ✓ Développer un système de reconnaissance par empreinte digitale pour l'utiliser lors de la validation des environnements.

IV. Plan de la thèse

Ce rapport est composé de 4 chapitres. Le premier est consacré à une étude bibliographique sur les systèmes continus, discrets et hétérogènes. Le deuxième présente les différents environnements de vérifications ainsi la méthodologie de co-simulation matériel/logiciel multi niveau. Dans le troisième chapitre, nous proposons la méthodologie de vérification des systèmes continus/discrets. Dans ce dernier, les interfaces de communication et les schémas de synchronisation sont décrits. Enfin nous présentons dans le chapitre 4, à travers plusieurs applications, la validation de l'environnement de co-simulation matériel/logiciel multi niveau, la simulation matériel/logiciel en boucle et l'environnement de co-simulation/émulation continu/discret (CODIS+).

Chapitre 1 : ETAT DE L'ART

I. Introduction

Compte tenu de la diversité et de la complexité des systèmes, plusieurs méthodes de descriptions sont étudiées dans la littérature. Chaque méthode dépend de la nature des systèmes à concevoir. En fait, plusieurs outils et environnements de modélisation et de vérification existent pour les systèmes continus et pour les systèmes discrets. En contre partie, les outils destinés aux systèmes continus/discrets souffrent encore de plusieurs lacunes. Le temps de simulation, la modélisation dans différents niveaux d'abstractions et le temps de mise en marché sont principalement les insuffisances présentes dans les environnements supportant les systèmes continus/discrets.

Dans ce chapitre, nous présentons tout d'abord une description des méthodes de modélisations et de simulations du modèle continu. Un exemple illustre la modélisation au niveau comportementale et fonctionnelle. Ensuite une description du principe de modélisation des systèmes discrets. Finalement une étude sur les méthodes de descriptions des systèmes continus/discrets basées sur l'approche homogène et hétérogène est détaillée. Cette dernière section cite les différentes caractéristiques des outils présents en soulignant les avantages et les inconvénients de chaque méthode.

II. Principe de modélisation et vérification du modèle continu

Tout modèle continu se base sur la résolution des équations différentielles ordinaires (EDOs) (Ordinary Differential Equations). Ainsi, les diagrammes de blocs utilisent ses EDOs pour la modélisation des systèmes. Une étude détaillée est présentée dans cette section.

II.1. Modélisation du modèle continu

Par définition, les systèmes continus couvrent tous les systèmes dynamiques à variables continus dans le temps. Leur modélisation se fait au niveau comportemental ou fonctionnel.

II.1.1. Au niveau comportemental

Le système continu est modélisé dans son ensemble de fonctionnement. Le modèle est décrit, dans ce cas, par des EDOs. Les équations utilisées sont des équations différentielles d'ordre 1 données par l'équation (1). Pour les EDOs d'ordre supérieur peuvent être réduites à un système d'équations différentielles d'ordre 1. Bien que celles supérieur à 1 puissent être

parfois résolues directement, très peu d'algorithmes sont disponibles pour le faire (Gupta G.K., 1985).

$$\dot{y} = \frac{dy}{dx} = f(x, y), \qquad y(x_0) = y_0 \qquad \text{où } y \text{ est un vecteur} \qquad (1)$$

L'équation (1) est appelée EDO explicite. Il existe une autre forme appelée EDO complètement implicite donnée par la forme suivante

$$f(x, y, \dot{y}) = 0 \qquad (2)$$

La plupart des EDOs complètement implicites peuvent être écrites sous la forme suivante (Gupta G.K., 1985)

$$\dot{y} = M(x, y) \qquad \text{où } M \text{ est une matrice} \qquad (3)$$

La forme (3) est appelée EDO linéairement implicite.

Dans le cas des systèmes continus l'équation (1) devient:

$$\dot{x} = \frac{dx}{dy} = f(x, u, t), \qquad x(t_0) = x_0 \qquad (4.1)$$

$$y = g(x, u, t) \qquad (4.2)$$

Où, *t* représente le temps, *u* représente le vecteur d'entrée, *x* représente le vecteur des variables d'états et *y* représente le vecteur de sortie. Ainsi, un espace d'états complètement spécifié par les équations (4.1) et (4.2) est obtenu.

L'équation (4.1) représente l'ensemble des équations d'états avec une condition initiale, et l'équation (4.2) donne l'ensemble des équations de sortie. Assumons que nous avons *n* variables d'états, *m* variables d'entrées et *r* variables de sorties, ces équations peuvent être écrites sous la forme scalaire suivante :

Il y aura *n* équations d'états

$$\begin{cases} \dot{x}_1 = f_1(x_1(t), \ldots\ldots, x_n(t), u_1(t), \ldots\ldots\ldots, u_m(t), t), & x_1(t_0) = x_{10} \\ . \\ . \\ \dot{x}_n = f_n(x_1(t), \ldots\ldots, x_n(t), u_1(t), \ldots\ldots\ldots, u_m(t), t), & x_n(t_0) = x_{n0} \end{cases} \qquad (4.3)$$

et *r* équations de sortie

$$\begin{cases} y_1 = g_1(x_1(t),.....,x_n(t),u_1(t),.....,u_m(t),t) \\ . \\ . \\ y_r = g_n(x_1(t),.....,x_n(t),u_1(t),.....,u_m(t),t) \end{cases} \quad (4.4)$$

La linéarité :

La notion de linéarité est fondamentale dans les domaines scientifiques et les domaines d'ingénieur. La nature des fonctions *f* et *g*, donnée par les équations (4.1) et (4.2), sert à fixer la nature du système. Ce dernier est appelé linéaire si ces deux fonctions sont toutes les deux linéaires. Dans ce cas, les équations (4.1) et (4.2) se réduisent à :

$$\dot{x}(t) = A(t)\,x(t) + B(t)\,u(t) \quad (5.1)$$

$$\dot{y}(t) = C(t)\,x(t) + D(t)\,u(t) \quad (5.2)$$

A(t) (n,n), B(t) (n,m), C(t) (r,n) et *D(t)(r,m)* sont des matrices où *n*, *m* et *r* sont les mêmes variables données ci-dessus. La classe des systèmes linéaires est en effet restreinte. Par exemple, une simple fonction comme $f(x) = x^n$ ($n > 1$ est un entier) est non linéaire.

L'invariance par rapport au temps :

Une autre propriété est l'invariance du modèle du système par rapport au temps. Dans le cas où les fonctions *f* et *g* ne dépendent pas explicitement du temps, le système est dit invariant par rapport au temps, dans ce cas les équations (4.1) et (4.2) se transforment en :

$$\dot{x}(t) = f(x(t),u(t)) \quad (6.1)$$

$$\dot{y}(t) = g(x(t),u(t)) \quad (6.2)$$

En assumant la propriété de l'invariance par rapport au temps, nous pouvons restreindre la classe des systèmes linéaires en une autre classe où les matrices *A(t), B(t), C(t)* et *D(t)* sont constantes, et on obtient :

$$\dot{x} = Ax + Bu \quad (7.1)$$

$$y = Cx + Du \quad (7.2)$$

Les équations différentielles-algébriques :

Si l'ensemble des équations qui décrivent le système continu sont composées d'équations algébriques et différentielles, elles sont nommées donc équations différentielles-algébriques et données par (Gupta G.K., 1985)

$$\begin{aligned} &F_1(x, y, z, \dot{y}) = 0, \quad y(x_0) = y_0 \\ &F_2(x, y, z) = 0 \end{aligned} \tag{8}$$

Où F_1 est un ensemble de N équations et F_2 de M équations.

Il est indéniable que la modélisation au niveau comportemental s'avère une tâche pénible pour les concepteurs. C'est pourquoi la modélisation au niveau fonctionnel évolue rapidement.

II.1.2. Au niveau structurel

Le système continu est modélisé par un ensemble de fonctions, prenons par exemple : un régulateur PID. Dans ce cas, le modèle est décrit par un diagramme de blocs prédéfinis où chaque bloc est caractérisé par un ensemble de relations, linéaires ou non linéaires, entre les variables d'entrées et les variables de sorties, citons par exemple: sommation, fonction de transfert, intégration, etc. Les blocs sont interconnectés par des chemins orientés représentant des signaux. A ce niveau d'abstraction et à partir des blocs prédéfinis, il est possible de construire des modèles et des sous modules pour des systèmes dynamiques complexes.

Actuellement, grâce à des blocs spéciaux, les EDOs modélisant un système continu au niveau comportemental peuvent être programmées et connectées aux autres blocs du diagramme. Ainsi, le formalisme du diagramme de blocs est le plus adapté puisqu'il supporte aussi le formalisme des EDOs. En fait, c'est plus facile de modéliser un système continu en utilisant le diagramme de blocs que par la résolution des EDOs.

II.1.3. Modélisation : exemple continu

A travers cet exemple nous présentons le formalisme de diagramme de blocs. Pour une meilleure explication, nous utilisons aussi le formalisme des EDOs où nous expliquons la technique de réduction d'ordre. Prenons l'exemple du circuit RLC représenté par la figure 1.

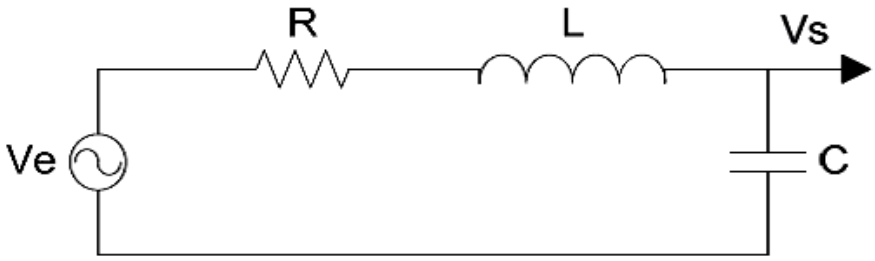


Figure 1. Circuit RLC série

Modélisation comportementale

Le comportement de ce circuit est décrit par l'équation différentielle (9) d'ordre deux issues de la loi des mailles :

$$V_e = LC\frac{d^2V_s}{dt} + RC\frac{dV_s}{dt} + V_s \qquad (9)$$

Pour résoudre cette équation numériquement, nous aurons besoin de la réécrire sous un système équivalant d'équations différentielles d'ordre 1. Nous supposons :

$$y_1 = V_s \qquad (10)$$
$$y_2 = \dot{V}_s$$

En combinant les deux équations (9) et (10), on obtient le système d'équations différentiel d'ordre 1 suivant :

$$\begin{cases} \dot{y}_1 = y_2 \\ \dot{y}_2 = 1/LC(V_e - RC\,y_2 - y_1) \\ V_s = y_1 \end{cases} \qquad (11)$$

Par la même démarche donnée par (10) et (11), toute équation différentielle d'ordre supérieur à 1, peut être réécrite sous un système équivalent d'équations différentielles d'ordre 1. Le même circuit peut être facilement décrit par l'espace d'états (forme (7.1) et (7.2)) donné par :

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1/C \\ -1/L & -R/L \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1/L \end{bmatrix} V_e \qquad (12)$$

$$V_s = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad , où\ x_1 = V_e\ et\ x_2 = I\ représentent\ les\ \text{var}iables\ d'états$$

Nous pouvons remarquer que les systèmes (11) et (12) sont équivalents, ainsi ils peuvent être facilement programmés en utilisant un éditeur de texte ou modélisés par un diagramme de blocs.

Modélisation fonctionnelle

Le diagramme de blocs : La figure 2 montre le même circuit modélisé par un seul bloc qui décrit sa fonction de transfert donnée par l'équation (13)

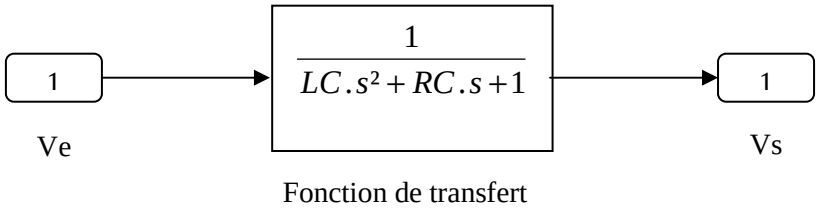


Figure 2. Modélisation par diagramme de blocs (fonction de transfert)

$$H(s) = \frac{V_s(s)}{V_e(s)} = \frac{1}{LCs^2 + RCs + 1} \qquad (13)$$

En utilisant (12), le circuit peut être décrit par un diagramme de blocs, en utilisant des blocs prédéfinis appelés primitifs, qui sont l'intégrateur, l'additionneur et le gain où l'intégrateur représente le bloc principal (Callier F.M., 1991), voir figure 3.

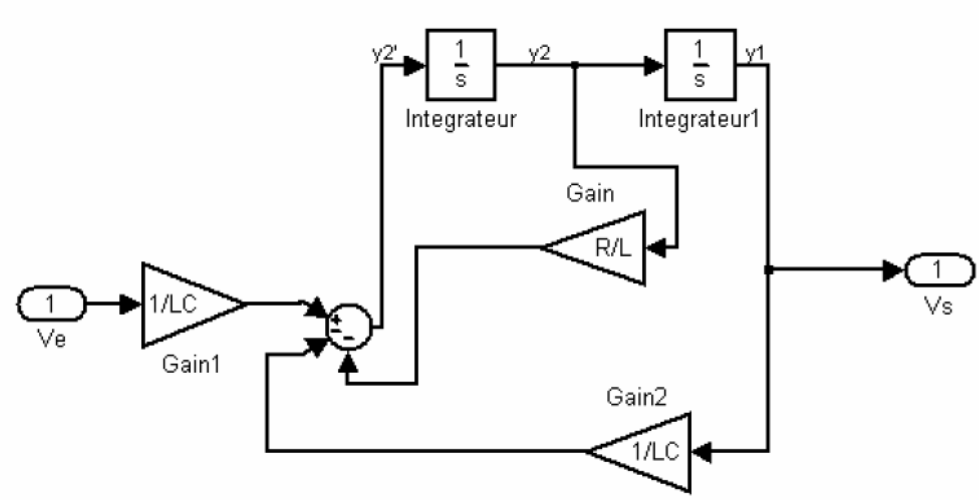


Figure 3. Modélisation par diagramme de blocs (bloc primitifs)

Il est clair, à travers cet exemple, que la modélisation des systèmes continus par le diagramme de blocs est plus simple. Dans la section suivante nous introduisons le modèle de simulation des systèmes continus.

II.2. Modèle de Simulation

La simulation des systèmes continus se base sur la résolution numérique du système d'équations différentielles et algébriques. Plusieurs algorithmes de résolutions essaient de résoudre les EDOs en un temps le plus court possible afin de pouvoir traiter des problèmes de grande taille. Une large classe d'algorithmes discrétisent le temps en un ensemble d'instants discrets croissants et calculent numériquement les variables du modèle à ces instants. Un pas d'intégration correspond à l'intervalle entre deux instants consécutifs. Ce pas peut être fixe ou variable.

Durant la simulation, le temps avance par le pas d'intégration. À chaque pas, les blocs qui modélisent le système continu sont exécutés (résolus) et l'ensemble des états continus sont mis à jour. L'ordre de résolution de ces blocs est donné par la règle de dépendance des données.

La précision, la stabilité et la continuité des signaux sont les trois critères responsables aux choix du pas d'intégration. Lorsque la précision est le seul critère à prendre en compte (c'est-à-dire le système est à la fois stable et continu), on peut utiliser un algorithme à pas fixe. En contrepartie, l'utilisation d'un algorithme à pas variable augmente la vitesse de la simulation, puisque l'algorithme réduit le pas quand le modèle évolue rapidement et vice versa. Ceci évite tous les calculs non nécessaires et réduit le nombre total des pas

d'intégration. Bien que le calcul de la largeur du pas d'intégration ajoute un temps de calcul additionnel à chaque pas, l'impact positif de la réduction du nombre total de ces pas d'intégration s'impose.

Lorsque le modèle continu présente des discontinuités et/ou des problèmes de stabilité alors il faut utiliser :

- Des algorithmes à pas variable (Gear C.W., 1984), pour surmonter les problèmes de discontinuités observés au niveau des solutions, surtout lorsqu'il interagit avec un environnement discret où les signaux changent leurs valeurs d'une manière discontinue. Dans le cas de discontinuité, l'algorithme recalcule les solutions en raffinant les pas d'intégration autour de ces points de discontinuité.

- Des algorithmes spécifiques à pas variable pour résoudre les problèmes dû aux méthodes numériques pour la résolution des équations différences qui sont numériquement instable. Ces problèmes sont appelés les problèmes *stiff* qui apparaissent surtout dans les modèles non linaires dans le cas des systèmes mécaniques, électriques, etc. Ces algorithmes sont conçus pour la résolution des problèmes *stiff*, car dans leur cas le pas d'intégration est contrôlé par précision plutôt que par stabilité (Gupta G.K., 1985).

Le problème *stiff* apparait lorsque les variables d'états évoluent d'une manière très rapide sur un intervalle de temps très court par rapport au pas d'intégration. Ceci peut être observé au niveau de la matrice Jacobine (Sameh A.H, 1971) qui peut avoir des valeurs propres qui sont négatives (ou complexes avec des parties réelles négatives) avec des modules largement supérieurs par comparaison aux autres valeurs propres. Cela implique que des composants de la solution vont dégrader très vite et deviennent non significatifs. Alors, l'algorithme doit changer le pas d'intégration sans tenir compte des valeurs propres liées à ces solutions.

III. Principe de modélisation et vérification du modèle discret

Nous présentons ici les concepts de base utilisés pour la modélisation et la vérification des systèmes discrets.

III.1. Modélisation du modèle discret

Par définition les systèmes discrets sont tous les systèmes numériques ou d'une manière plus générale tous les systèmes à événements discrets. Leurs comportements sont souvent décrits par des processus concurrents en utilisant des expressions booléennes, logiques et/ou arithmétiques selon le niveau d'abstraction. Dans le niveau RTL (Regiter Transfer Level), ces

processus sont connectés par des signaux à travers leurs ports d'entrée/sortie (figure 4). Ces signaux qui représentent un support physique, assurent la communication et l'échange des évènements entre les processus. Un événement représenté par le couple (valeur du signal, temps d'occurrence) est un événement dû à un changement de la valeur d'un signal à un instant précis. Un événement pur est un événement qui se représente par son temps d'occurrence seulement. L'exécution d'un processus est déclenchée si un évènement dans sa liste de sensibilité est aussi déclenché. Par définition, une liste de sensibilité contient une liste de signaux qui réveillent le processus lors d'un changement d'un des signaux. Si plusieurs processus sont sensibles à un ou à plusieurs événements qui ont le même temps d'occurrence alors, dans les deux cas, ces processus doivent être exécutés en parallèle. Le parallélisme est un aspect qui est assuré par le modèle de simulation mais qui doit être pris en compte par le modèle. Le problème est dû à l'exécution à partir d'une machine séquentielle, capable d'exécuter une instruction à la fois, toutefois cette machine ne peut pas paralléliser réellement les différents processus en même temps. La solution repose sur une idée très simple mais efficace : le processus exécuté ne doit pas changer les valeurs de sortie des processus jusqu'à la fin de l'exécution des autres processus qui lui sont en parallèle. Ainsi, l'ordre d'exécution de ces processus n'a plus d'importance et tout se passe comme s'ils s'exécutaient en parallèle. Pour parvenir à ce résultat, il faut que les signaux d'un processus conservent leurs valeurs jusqu'à ce que tous ces processus aient fini leur exécution (Valderrama C.A., 1995).

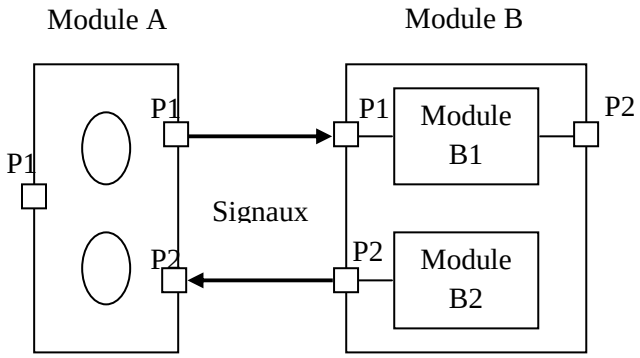


Figure 4. Un exemple du modèle discret

III.2. Modèle de simulation

La simulation dans le domaine discret désigne la vérification d'un modèle qui évolue dans le temps à travers des variables (grandeurs caractéristiques des systèmes) qui ne changent qu'en nombre fini dans le temps. Ces points représentent les instants où se déroulent les événements (changement des variables). La simulation prend en compte les tâches actives

à des instants précis. Toute une description des différents types de simulation est décrite dans le chapitre 2.

IV. Principe de modélisation et vérification du modèle continu/discret

A l'heure actuelle, les recherches concernant les méthodes, les outils formels relatifs à l'analyse du comportement des systèmes hétérogènes et à la synthèse de leurs lois de commande en sont encore à leur début. La simulation reste donc un passage nécessaire pour la validation du fonctionnement de ces systèmes hybrides. Vu l'hétérogénéité des modèles à valider, la simulation n'est pas une tâche facile. Pour surmonter cette difficulté, deux approches sont proposées : l'approche homogène et l'approche hétérogène.

IV.1. Approche homogène : conception et simulation

Cette approche consiste à utiliser un seul langage pour la spécification complète du fonctionnement du système. Cela suppose qu'il possède une sémantique consistante et assez riche pour qu'il puisse supporter l'hétérogénéité des modules continus/discrets.

Une première solution consiste à étendre les langages existants, en modifiant le noyau de simulation afin de supporter l'hétérogénéité. L'avantage de cette solution est que le style d'écriture est formalisé de façon à faciliter l'analyse formelle et la construction de nouveaux outils. L'inconvénient majeur réside dans la difficulté de la construction de nouvelles bibliothèques, mécanismes et formalismes qui demandent un temps d'apprentissage important pour les nouveaux langages.

En citant les différents travaux et outils, on a essayé de les classifier selon leurs points communs. Plusieurs travaux et outils portant sur l'extension des langages matériels étaient proposés pour enrichir ou améliorer leur capacité descriptive et simulatrice. Ces extensions étaient pour le domaine continu et pour d'autres domaines discrets plus spécifiques.

IV.1.1. Validation analogique / numérique

Ce type d'outils est le sujet de plusieurs travaux, citant Diana (De Man H.J., 1980), Splice (Newton A.R., 1978), Motis (Chen C.F, 1984) Samson (Sakallah K.A., 1985), Spice (Banzhaf W., 1989). Ces outils se basent sur la combinaison de deux algorithmes dans le noyau de simulation : algorithme de résolution des EDOs et algorithme de gestion des événements discrets. L'important dans ces outils est la simulation rapide de la partie numérique. Mais malheureusement, ces outils supportent seulement le niveau transistor pour

la modélisation, augmente le temps de mise en marché et rend pénible la phase de modélisation.

D'autre travaux comme Mode Circuit Simulator (Odryna P., 1986), MAST de SABER (Getreu I.E., 1989), ALFA (Kazmierski T.J., 1992), S++SDL (Brown A.D., 1992), Verilog-A (Fitzpatrick F., 1998) et HDL-A (Pabst D., 1995) modélisent les modèles analogiques à plus haut niveau (macro-modèles). Mais de même que précédemment, les modèles analogiques restent à un niveau beaucoup plus bas.

IV.1.2. Extension du langage VHDL et Verilog

Ce type d'outils utilisent deux noyaux différents : un pour l'analogique et l'autre pour les événements discrets, pour simuler le comportement global du système. Les outils comme VHDL-AMS (Std VHDL-AMS, 1999) développé par le standard IEEE 1076.1, (Drager S.L., 1998), (Pêcheux F., 2005) et Verilog-AMS développé par le co-standard IEEE 1364 (Frey P., 2000), (Pêcheux F., 2005) assurent la validation des systèmes hétérogènes à différents niveaux d'abstractions. Une autre approche (Pichon F., 1995), est fondée à la fois sur une description par fonctions de transfert et une approximation des signaux analogiques par PWL (Piece Wise Liner). La même technique est employée dans (Long D.I., 1997) pour étendre VHDL. Mais la conception des circuits analogiques et mixtes reste à un niveau beaucoup plus bas que celui des circuits numériques. Ces langages n'offrent pas un niveau d'abstraction suffisamment élevé (Vachoux A., 2003) et souhaitable pour la simulation des systèmes sur puce intégrant du matériel numérique, du logiciel et d'autres composants non électriques et ne supportent pas la simulation conjointe des systèmes matériels / logiciels. Ces outils présentent aussi une limitation au niveau interaction entre les modules continus et discrets, ce qui oblige l'utilisateur à créer d'une manière explicite les interfaces nécessaires. De plus, le temps de passage du sous-modèle continu au sous-modèle discret et vice versa est toujours difficile à repérer. Ces langages sont toujours considérés comme souhaitables pour la modélisation des systèmes mixtes « big-D-little-A » c'est-à-dire pour des systèmes numériques intégrant une faible composante analogique (Antao B.A., 1996). En contre partie, ces outils utilisent un seul solveur pour la résolution des systèmes d'équations algébriques et différentielles ce qui rend nécessaire d'indiquer dans le code l'emplacement où le modèle change de fonctionnement c'est-à-dire que la discontinuité du modèle n'est pas résolue automatiquement par le simulateur.

IV.1.3. Extension de SystemC

Cette extension se base sur la construction d'une nouvelle bibliothèque et des solveurs pour la résolution des équations différentielles et algébriques SystemC-AMS (Vachoux A., 2003), SystemC-A (Al-Junaid H., 2005), (Al-Junaid H., 2004). Dans (Vachoux A., 2003), les auteurs indiquent la possibilité d'utilisation d'un mécanisme de synchronisation pour l'intégration d'autres simulateurs et solveur pour des systèmes assez complexes et pour des niveaux d'abstraction qui ne sont pas couverts par SystemC-AMS, ce qui rend leur approche intéressante. Dans (Bonnerud T.E., 2001), d'autres travaux sont proposés pour étendre SystemC par classes, pour la simulation mixte d'un convertisseur pipeline analogique/numérique. Dans (Patel D.H., 2004), les auteurs proposent l'extension des capacités de description de SystemC en ajoutant un nombre de noyaux spécifiques à quelques domaines discrets qui sont : le domaine de Flux de données synchrones (SDF: Synchronous Dataflow), le domaine CSP (Communicating Sequential Processes) et le domaine de Machine d'états finis (FSM: Finite State Machine).

Cette extension permet d'étendre la modélisation et la simulation des systèmes continus. La simulation devient plus performante spécialement dans le domaine de communication et de traitement de signal (Vachoux A., 2003). Les travaux cités montrent par quelques exemples que la précision de SystemC a été améliorée et les performances de la simulation ont augmenté quand les noyaux spécifiques ont été utilisés. Les auteurs créent aussi un ensemble d'interfaces de programmation (API : Application Programming Interface) pour permettre aux développeurs d'ajouter d'autres noyaux spécifiques à d'autre domaines de modélisation. Mais la simulation du modèle continu reste moins puissante au niveau de la précision de la simulation et de la disponibilité des bibliothèques par rapport à l'environnement complet que Matlab / Simulink (Matlab/Simulink, 2012) l'offre d'une part, et un manque de solveurs adéquats pour les différents domaines continus (mécanique, hydraulique, robotique,…) d'autre part.

IV.1.4. Ptolemy II

Ptolemy (Eker J., 2003) est développé au sein de l'université de Berkeley. Il utilise un environnement et un langage qui sont unifiés. C'est une approche hétérogène de point de vue composition disjointe des modèles de calcul appelé actors. Il utilise des directors qui implémentent les modèles de calcul et qui permette d'établir le style de communication entre les actors et de fixer leur ordre d'exécution. La composition des modèles de calcul dans Ptolemy II assure la spécification des systèmes hétérogènes multi-disciplines et multi-

domaines. Cet outil est à source ouverte pour les développeurs et indépendant des plateformes grâce à la technologie Java. Les inconvénients de cet outil résident dans le temps d'apprentissage important et le non utilisation des bibliothèques (en particulier les composants matériels comme les processeurs et les accélérateurs) et IPs conçu pour chaque modèle.

IV.1.5. MLdesigner

MLdesigner (Mission Level designer) (Schorcht G., 2003) se base sur l'approche Ptolemy. C'est une plateforme unifiée, dédiée à la modélisation (fonctionnelle et architecturale) et à la simulation au niveau système. L'environnement établit une connexion par appel de service (callback) avec SatLab (Schorcht G., 2003) pour assurer des calculs de trajectoire ou des analyses pour les systèmes de navigation et de communication. L'utilisateur peut construire son modèle en utilisant une interface graphique proche de celle du Simulink (Matlab/Simulink, 2012). Les blocs fonctionnels fournis par la bibliothèque sont écrits par un langage proche de C++ et sont paramétrables. Malheureusement, les exemples fournis avec MLdesigner ciblent seulement les systèmes à architecture numérique mono et multi-processeurs. De plus, Mldesigner est complexe dans son environnement et le langage utilisé nécessite un temps d'apprentissage important.

IV.1.6. Modelica

Modelica (Modelica, 1997) est un langage et environnement unifié pour la spécification et la modélisation des systèmes physiques. Les composants du système sont mathématiquement décrits par des équations différentielles et algébriques. Cet outil montre une bonne capacité de modélisation et de réutilisation en se basant sur les concepts d'orienté objet et de non-causalité. Modelica fournit un ensemble de bibliothèques dans plusieurs disciplines et domaines : continu, électrique, mécanique, thermique, discret et logique booléen, réseau de Petri, logique floue, VehicleDynamics, etc. Le langage, les librairies et les outils de simulation de Modelica sont à usage libre, mais il existe des environnements de simulation commerciaux basés sur ce langage qui sont Dymola (Ferretti G., 2006) de Dynasim et MathModelica (Mathmodelica, 2006) de MathCore Engineering. Mais, il est incapable de supporter la notion d'événements discrets exploitée dans la simulation des systèmes numériques.

IV.1.7. Outils basés sur l'approche UML

Paragon (Pinki M., 2003), (Riihimaki J., 2005) et (Kajtazovic S., 2005) peuvent appartenir aussi à l'approche hétérogène car ils partent avec un seul langage de modélisation

(UML) mais, dans la majorité des cas, après ils font appel aux langages existants (VHDL, SystemC, etc.). Au niveau modélisation, les auteurs décrivent la structure du système en utilisant des IPs (Intellectual property) mais lors de la simulation, ils utilisent la technique de la co-simulation.

SysML (Azam F., 2005) (Systems Modeling language) est développé à partir de l'UML pour la spécification, l'analyse et la validation des systèmes matériels/logiciels et des systèmes d'information. Pour la vérification, SysML génère un code adéquat à un langage cible (comme VHDL et C) en respectant le simulateur utilisé.

Paragon (Pinki M., 2003) qui est un outil de modélisation indépendant des langages matériels utilisés pour la simulation ou pour la conception. Il fournit une sémantique qui est capable de décrire des systèmes continus/discrets. Le fonctionnement (les expressions et les calculs) est décrit par MathML qui représente une application de l'XML pour la description des notations mathématiques. L'important ici c'est que l'utilisateur peut créer son modèle à partir d'une interface graphique.

Tous ces outils sont plutôt utilisés pour décrire la structure et l'hiérarchie des systèmes. Le comportement est difficilement décrit par ces langages qui ne peuvent pas fournir la sémantique donnée par les langages matériels.

IV.1.8. Outil basé sur les métriques approximatives

(Antoine G., 2007) a développé un environnement qui supporte le modèle continu et le modèle discret en utilisant le langage d'inclusion approximative et la bisimulation approximative. L'utilisation de cette approche approximative qui est basée sur la machine d'état infinie pour le modèle continu et la machine d'état finie pour le modèle discret, permet de résoudre la complexité des EDOs et d'accélérer la simulation d'une part et souffre de manque d'algorithme pour le calcul des fonctions linéaires et non-linéaires d'autre part.

IV.1.9. Hybride Automata

(Vladimeros V., 2012) a développé un outil en se basant sur la solution « Hybride Automata » qui permet à la fois la modélisation continu/discret. Cet outil utilise la classe mathématique o-minimal qui se base sur une résolution géométrique des EDOs. Une amélioration de la classe o-minimal est présentée par Vladimeros pour considérer le modèle discret. Cet outil exploite mieux les propriétés des systèmes temps réels mais il est plus adapté pour le modèle continu que pour le modèle discret.

IV.1.10. Synthèse

L'approche homogène repose sur deux types d'environnements :

- ✓ Nouvel environnement comme par exemple Ptolemy, Mldesigner et Modelica.
- ✓ Extensions des langages comme VHDL-Verilog et SystemC-AMS.

Le tableau 1 résume la majorité des travaux en mettant l'accent sur les avantages et les inconvénients de chaque environnement.

Outils	**Avantages**	**Inconvénients**
Validation analogique / numérique	- Simulation de la partie numérique assez rapide. -Modélisation analogique au niveau macro-modèle	- Simulation au niveau transistor seulement. - Modélisation reste à un niveau beaucoup plus bas.
Extension du langage VHDL et Verilog	- Validation des systèmes hétérogènes à différents niveaux d'abstractions.	- Niveau d'abstraction limité. - Simulation conjointe des systèmes matériels / logiciels non supporté. - Interaction continu/discret non définie. - Un seul solveur est utilisé pour la résolution des EDOs.
Extension de SystemC	- Etendre la modélisation et la simulation des systèmes continus. - Simulation performante.	- Simulation analogique moins puissante au niveau de la précision. - Manque de solveurs adéquats pour différents domaines.
Ptolemy II	- Spécification des systèmes hétérogènes multi-disciplines et multi-domaines. - Une source ouverte pour les développeurs.	- Nécessite un temps d'apprentissage important.
MLdesigner	- Une interface graphique proche de celle du Simulink.	- Complexité de l'environnement. - Nécessite un temps d'apprentissage important.
Modelica	- Bonne capacité de modélisation et de réutilisation en se basant sur les concepts d'orienté objet et de non-	- Ne supporte pas la notion d'événements discrets.

	causalité. - Existence des bibliothèques dans plusieurs disciplines et domaines. - Une source ouverte pour les développeurs.	
Outils basés sur l'approche UML	- Création du modèle à partir d'une interface graphique.	- Le comportement est difficilement décrit par ces langages.
Métrique approximative	- Résoudre la complexité des EDOs. - Simulation rapide.	- Moins de précision.
Hybride Automata	- Exploitation des propriétés des systèmes temps réels.	- Plus adaptée pour le modèle continu que pour le modèle discret.

Tableau 1 : Avantages / inconvénients des outils basés sur l'approche homogène

IV.2. L'approche hétérogène

Cette approche permet de modéliser le système complet en utilisant des langages spécifiques. La technique de co-simulation permet l'utilisation de plusieurs simulateurs pour la validation globale de ce système. Pour cela, il faut disposer d'un modèle de communication qui décrit la synchronisation et les interconnexions entre les différents modules. La difficulté réside dans la construction de ce modèle. Une autre technique, la technique mono-simulateur, consiste à traduire les langages utilisés pour la description du système entier vers une sorte de langage unique ou un format accepté par le simulateur (Nicolescu G., 2002).

L'avantage de cette approche est que chaque module du système peut être modélisé avec un langage spécifique et approprié. Cela permet d'intégrer les IPs et d'exploiter au mieux les performances des langages existants.

Généralement les raisons et les avantages pour lesquels on fait recours à plus qu'un langage sont :

- ✓ L'hétérogénéité du système qui combine plusieurs domaines physiques, par exemples : mécanique, électronique numérique (matériel et logiciel), chimie, etc.
- ✓ Le système possède des modules qui appartiennent à plusieurs niveaux d'abstraction, et donc l'utilisation du langage le plus adéquat pour chaque niveau.

- ✓ Le besoin de créer des testbenchs complexes (d'habitude on utilise le langage C et ses dérivés)
- ✓ L'exploitation des bibliothèques déjà existantes pour certains langages.

Nous allons classifier ces outils en deux catégories :

- ✓ Outils qui utilisent plusieurs langages/plusieurs simulateurs : technique de co-simulation
- ✓ Outils qui utilisent plusieurs langages/un seul simulateur : technique mono-simulateur

IV.2.1. Technique mono-simulateur

Avec cette technique le système est toujours composé d'un ensemble de sous-systèmes spécifiés dans différents langages mais la simulation nécessite le passage par un langage unifié ou un format connu par le simulateur, voir figure 5.

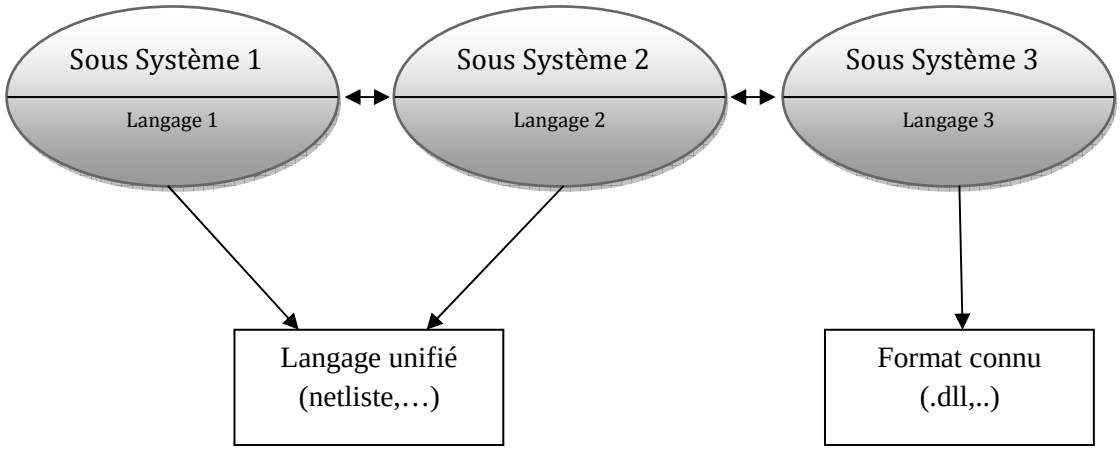


Figure 5. Le principe du technique mono-simulateur

La validation par simulation consiste à exécuter la spécification du système afin de reproduire le fonctionnement du système entier. Dans la figure 5, le sous-système 1 et le sous-système 3 sont des composants matériels modélisés par deux différents langages (comme par exemple, VHDL et Verilog) et le sous-système 2 représente une application logicielle ou un modèle continu (Zorzi M., 2003).

(Dubois M., 2011) propose un approche qui consiste au développement d'un simulateur compilé multi-langage où chaque modèle peut être décrit en employant différents langages de

modélisation tel que SystemC, ESyS.Net ou autres. Chaque modèle contient généralement des modules et des moyens de communications entre eux. Les modules décrivent des fonctionnalités propres à un système souhaité. Cette approche se base sur un seul noyau au lieu de plusieurs et d'enlever le bus de co-simulation pour accélérer le temps de simulation. Mais cet environnement ne supporte que le niveau RTL et TLM (Lukai, 2003).

IV.2.2. Technique de co-simulation

La co-simulation consiste à exécuter des simulateurs communicants, voir figure 6. Chacun des simulateurs exécute un sous-système décrit dans un langage approprié. Pour assurer l'échange correct des données et la communication entre ces sous-systèmes, le besoin d'un modèle de synchronisation s'impose. Ce modèle prend en compte les spécificités du modèle de simulation adopté par chaque simulateur.

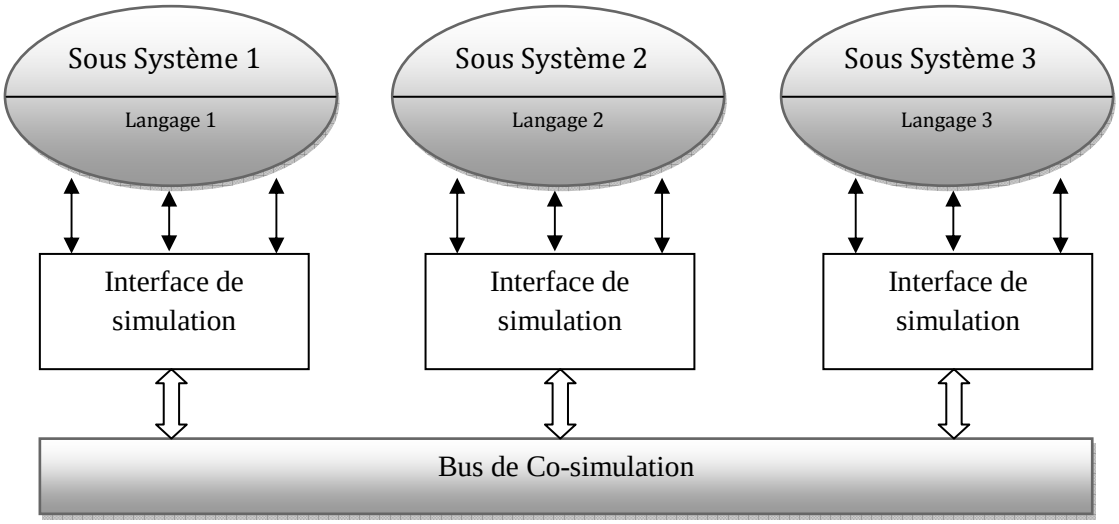


Figure 6. Le principe de la co-simulation

Des interfaces de simulation assurent l'interconnexion entre les différents sous systèmes. Ces interfaces communiquent à travers un bus de co-simulation qui peut être une mémoire partagée avec une structure bien définie permettant des interconnexions complexes ou autre technique de communication inter-processus. Les interfaces de simulation sont composées par des couches de communication et de synchronisation et selon les simulateurs utilisés ils peuvent implémenter des comportements assez complexes (Bouchhima F., 2005) (Nicolescu G., 2002). Les avantages de la technique de co-simulation sont :

- Bénéficier au mieux des performances des langages et des simulateurs existants (sémantique, précision de simulation, bibliothèques, etc.).
- Réutiliser des composants existants comme les IPs.

- Eliminer le temps d'apprentissage puisque les langages utilisés sont très connus par les concepteurs.

V. Discussion

Plusieurs outils de modélisation et de simulation ont été décrits tout le long de ce chapitre. Essentiellement deux approches sont utilisées : approche homogène et approche hétérogène. L'approche homogène consiste à développer des environnements supportant à la fois le modèle continu et le modèle discret. Ce type d'outil permet une simulation rapide mais présente plusieurs inconvénients citant :

- ✓ Non utilisation des bibliothèques et des IPs dédiées pour chaque modèle.
- ✓ Limitation au niveau d'abstraction : Il y a un manque d'environnement supportant la description dans tous les niveaux d'abstractions conjointement avec le modèle continu.
- ✓ Apprendre un nouveau langage.

En contre partie, la méthode de co-simulation de l'approche hétérogène permet la synchronisation entre le simulateur continu et le simulateur discret. Nous proposons d'utiliser l'environnement Matlab / Simulink pour la description et la simulation du modèle continu à cause de plusieurs avantages :

- ✓ Matlab / Simulink appartient aux langages métiers (Domain Specific Languages) (Consel C., 2004).
- ✓ Matlab / Simulink est spécialisés dans les domaines particuliers comme l'automatique et les systèmes de contrôle (Chapoutot A, 2008).
- ✓ Le temps de simulation pour le modèle continu est considéré plus rapide que l'environnement Ptolemy (SJÖSTEDT C., 2009).

L'avantage de cette approche lorsqu'il est liée avec le simulateur Matlab / Simulink, réside dans l'utilisation des simulateurs discrets existants, conçu pour tous les niveaux d'abstraction ainsi l'utilisation des bibliothèques et IPs standards.

Dans ce cadre notre environnement propose un modèle de co-simulation continu /discret matériel/logiciel multi niveau en ajoutant un émulateur à base d'une architecture cible

permettant à la fois d'accélérer la simulation et la modélisation conjointe matériel/logiciel de la partie numérique.

Les stratégies de co-simulation nous permettent donc de simuler et de vérifier des systèmes matériels/logiciels avant la mise en place d'une plateforme réelle. Dans ce domaine, il y a une grande variété d'approches qui utilisent des différents mécanismes de communication pour mettre en œuvre une interface efficace entre les applications logiciels et le simulateur matériels. Le besoin est important pour intégrer et synchroniser des simulateurs hétérogènes, comme, par exemple, le noyau de simulation du SystemC pour les composants matériels et le simulateur de jeu d'instruction (ISS) pour les applications logicielles.

L'objectif de cette thèse est de surmonter le problème de l'hétérogénéité des systèmes continus/discrets tout en fournissant des simulations précises et des temps de simulation assez satisfaisants. L'accélération de la simulation est un point clé qui impose la création d'interfaces de synchronisation et de communication entre l'environnement de simulation et la carte de prototypage sur FPGA.

VI. Conclusion

Nous avons présenté tout le long du premier chapitre les caractéristiques des modèles continus, discrets et hétérogène. La simulation de tels systèmes constitue un grand défi pour les concepteurs des environnements de CAO. En fait il existe principalement trois axes pour la modélisation et la vérification des systèmes continus/discrets. Le premier repose sur l'extension des langages afin d'étendre le noyau de simulation pour supporter à la fois le modèle continu et le modèle discret. Cet axe souffre des limitations au niveau d'abstraction. Le deuxième se base sur des nouveaux langages et environnements ce qui approuve l'accélération de la simulation. Malheureusement, ces langages ne supportent pas les différents niveaux d'abstractions et demande un temps important pour apprendre les nouveaux langages. Notre contribution s'intègre dans le troisième axe basé sur la co-simulation qui utilise les simulateurs existants pour chaque modèle et crée des interfaces de synchronisation entre-simulateurs. Malgré le temps de simulation plus au moins important, ces outils montrent une simulation performante à travers les différents niveaux d'abstractions. La réutilisation des IPs ainsi développés facilite la modélisation matériel/logiciel. Nous adaptons la troisième solution grâce aux nombreux avantages offertes par les outils basés sur la co-simulation. Le chapitre suivant présente une étude détaillée des techniques de vérifications matériel/logiciel ainsi l'approche de simulation/émulation utilisée.

Chapitre 2 : METHODOLOGIE DE MODELISATION ET DE VERIFICATION DES SYSTEMES MATERIELS/LOGICIELS

I. Introduction

Vue la complexité des systèmes et le taux d'intégration croissants, la modélisation traditionnelle des architectures matérielles/logicielles s'avère une tâche pénible, complexe, couteuse et limitée. Cette méthode repose sur une description bas niveau (composant, porte logique, transistor et dessin de masque) et séparer entre la partie matérielle et la partie logicielle. Cette méthode n'assure pas la vérification matériel/logiciel au cours de développement ce qui augmente le taux de rejet des circuits après fabrication.

La modélisation conjointe représente le fruit de plusieurs travaux de recherche afin de supporter les systèmes numériques complexes et les systèmes mono-puces. La conception basée sur la stratégie Co-design permet la vérification entre la partie matérielle et la partie logicielle conjointement avant la phase de fabrication. Cependant, plusieurs techniques et outils de modélisations et de vérifications sont décrits dans la littérature en respectant à la fois les langages adéquats (matériels et logiciels) et les niveaux d'abstractions utilisées. Toutes ces techniques assurent la communication et la synchronisation entre les applications logicielles et les composants matériels. Dans ce contexte, nous proposons quatre modèles de synchronisation basés sur un environnement de simulation/émulation afin de diminuer le temps de simulation. La synchronisation présente le point clé de la simulation/émulation. Elle doit prendre en compte les concepts de temps et d'activation des processus.

La première partie présente le principe de modélisation des systèmes mono-puces ainsi les différentes méthodes de vérification utilisées. Dans la deuxième partie, nous présentons l'approche de simulation/émulation en proposant un modèle de synchronisation entre simulateur/émulateur et les interfaces de communication matériel/logiciel.

II. Modélisation des systèmes mono-puces

La conception des systèmes en puce se basent sur un flot qui assure un développement parallèle des modules matériels et des modules logiciels. Ce flot se décompose en cinq étapes comme le montre la figure 7.

- Spécification système, on s'intéresse à la fonctionnalité au niveau système, indépendamment de l'implémentation finale (étape 1). Durant cette phase, on recherche les algorithmes et les représentations de données les mieux adaptés aux besoins et aux spécifications. La spécification fonctionnelle obtenue est généralement validée par une simulation.
- Spécification fonctionnelle : C'est l'étape qui suit l'étape précédente. Le but de la spécification fonctionnelle est la recherche d'une architecture pour implémenter les algorithmes déterminés par la spécification systèmes. Cette étape (étape 2) du flot de conception détermine les fonctionnalités qui seront implémentées en matériel et celles qui seront logicielles. En général, le principe du partitionnement se base sur la règle suivante : « les composants nécessitant des performances élevées sont réalisés par des modules matériels alors que les composants nécessitant essentiellement de la flexibilité sont implémentés en logiciel ». Finalement, cette étape permet l'obtention des spécifications de chacun des composants du système.
- La conception matérielle et logicielle (étape 3) correspond à la conception des composants matériels et au développement des logiciels embarqués. Pour cette étape, un gain de temps important est obtenu lorsqu'il y a utilisation des composants existants.
- Vérification et intégration : Lorsque tous les composants matériels et logiciels développés sont vérifiés chacun à part, la phase d'intégration et de vérification (étape 4) assure la communication et le bon déroulement entre les différents composants.
- Validation : Cette phase consiste à vérifier le système complet s'il répond bien au cahier de charge et aux fonctionnalités demandées (étape 5). Enfin, une fois cette dernière étape de conception est effectuée avec succès, on peut fabriquer le produit en grand nombre en passant par la fonderie.

A chaque étape de conception, les concepteurs doivent vérifier que les nouveaux composants ou les nouveaux détails de réalisation assurent une fonctionnalité correcte.

Cette vérification s'articule autour de cinq points illustrés par la figure 7 :

• Vérification de la spécification fonctionnelle.

• Vérification de l'architecture du système.

• Vérification de l'implémentation des composants du système.

• Vérification de l'intégration des composants.

• Vérification du système complet dans son environnement de fonctionnement avant la mise en fabrication et en production.

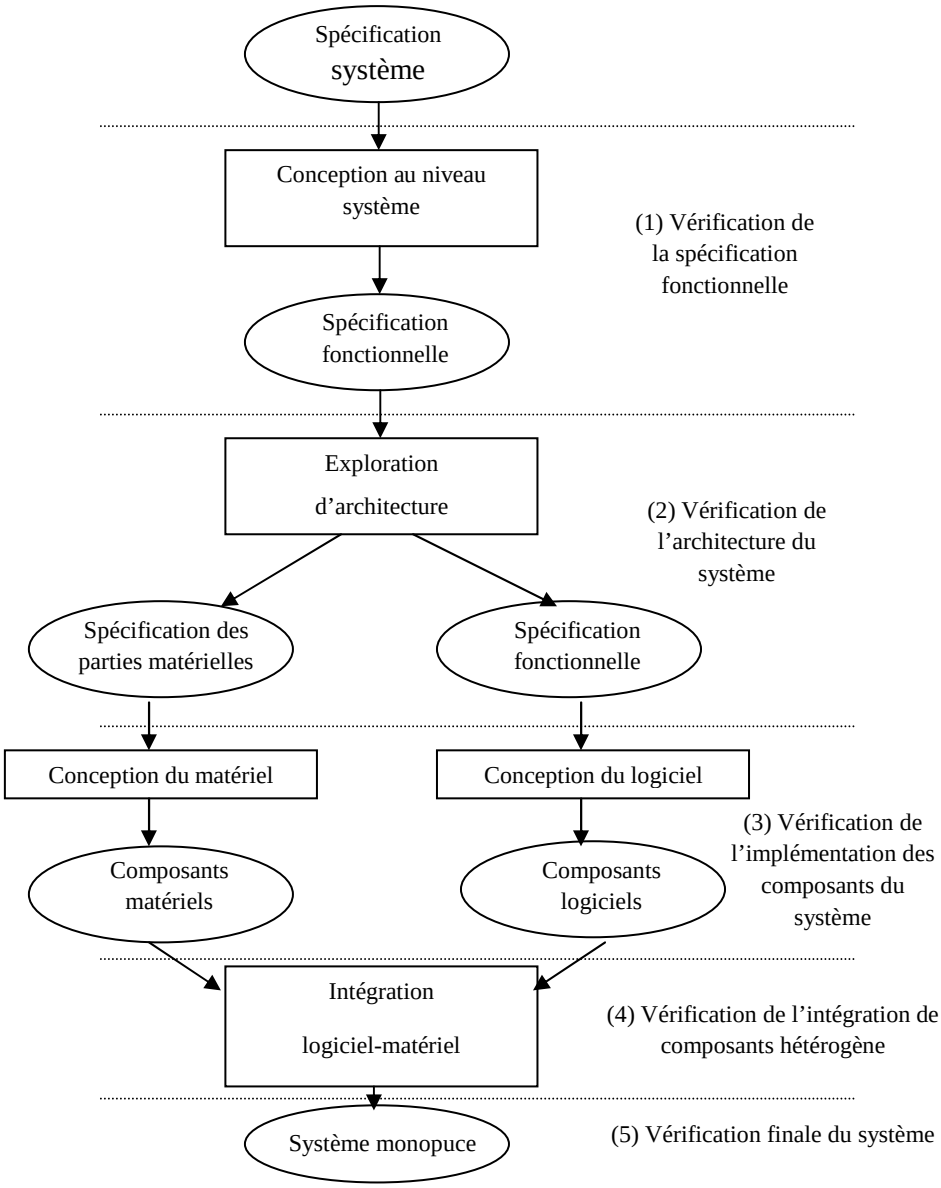


Figure 7. Flot de conception d'un système sur puce

La vérification peut occuper jusqu'à 70% du temps de conception, cette étape représente un élément important dans la durée de la conception d'un système. La vérification influence beaucoup en termes de temps ainsi qu'au niveau économique. Le coût de cette erreur est

estimé à quatre cent millions de dollars (Evans, 2003). En général, plus vite une erreur est détectée plus son coût de correction est faible.

Pour conclure, la vérification est une étape essentielle dans la conception des circuits mono-puces. Un des défis actuels consiste à améliorer les techniques de vérification, à augmenter la productivité des techniques et à réduire la durée et le coût de la vérification.

III. Les techniques de vérification

Lors d'une conception d'un système mono-puce, il existe plusieurs techniques de vérification: la vérification formelle, la simulation, la co-émulation, l'émulation à travers le prototypage. Chacune de ces techniques possède un modèle différent qui sera détaillé par la suite.

Le principal but de la vérification d'un système mono-puce tout au long du flot de conception est de prévoir les erreurs et les scénarios indésirables le plus tôt possible et par suite diminuer le temps de mise en marché du produit désiré. Afin de comparer et de mieux choisir la technique de vérification, des critères de comparaisons sont présentée.

Critères de comparaisons :

Chacune des techniques de vérification est caractérisée par un coût financier, une durée de mise en œuvre, une vitesse d'exécution, un niveau d'observabilité, un niveau de contrôlabilité et enfin un niveau de répétabilité.

Le **coût financier** de la vérification obéit à la règle suivante : « plus une erreur matérielle est tardivement détectée, plus son coût de correction est élevé ». L'importance de ce critère se présente dans la manière de maitriser le coût de vérification. L'investissement dans les outils de vérification s'avère utile lorsque leur prix est amorti pour une seule erreur matérielle détectée avant la fabrication du premier circuit. Cependant, les prix des outils de vérification varient dans une large gamme de prix allant de quelques milliers de dollars pour une licence perpétuelle (Rizatti L., 2003) au million de dollars pour une licence annuelle (Lardière C., 2004).

La durée de mise en place de la plateforme de vérification est un autre critère important. Selon la taille et la complexité du système, le choix de la méthode de la vérification est fait. Parfois l'utilisation d'une méthode plus lente est mieux adaptée parce que la mise en place de cette méthode est beaucoup plus courte par rapport au temps nécessaire

d'une deuxième méthode qui est plus compliquée et qui demande plus de temps malgré la rapidité de son temps de fonctionnement.

La **vitesse d'exécution** doit être considérée en relation avec la durée de mise en place de la plateforme de vérification et la longueur des séquences de test. Dans le cas de vérification temps réel avec des composants externes, la vitesse peut devenir une contrainte imposant une solution.

L'**observabilité**, la **contrôlabilité** et la **répétabilité** sont des critères liés à la puissance du débogage matériel, c'est à dire l'efficacité de la technique pour la détection des erreurs matérielles.

L'**observabilité** est la capacité d'observer les interactions des différents composants du système.

La **contrôlabilité** est la capacité de suspendre l'exécution du modèle, de modifier les valeurs de certains paramètres au cours de l'exécution.

La **répétabilité** est le fait de reproduire un scénario de test avec un niveau de précision donné.

Selon les paramètres déjà cités, il n'est pas facile de choisir la méthode la plus adaptée au problème considéré. Il convient donc de présenter les différentes techniques de vérification existantes en soulignant les avantages et les inconvénients de chaque technique.

III.1. Vérification formelle

La vérification formelle consiste à prouver mathématiquement qu'une description de circuit possède certaines propriétés. La vérification formelle se manifeste dans le débogage de la spécification qui vérifie si tous les besoins sont bien inclus et se manifeste aussi dans la vérification de l'implémentation qui vérifie si la spécification est bien implémentée. Cette technique est peu utilisée dans la conception des systèmes monopuces et possède plusieurs points faibles. Les principaux obstacles sont :

• La complexité du processus de vérification est très grande, ce qui limite l'utilisation aux simples applications.

• Cette technique demande une grande interaction entre concepteurs. Seuls des spécialistes peuvent utiliser cette technique.

• Cette technique n'est pas conçue pour les systèmes matériel/logiciel. En fait, la vérification formelle ne supporte pas le traitement d'un système synchrone parallèle (matériel) et le traitement d'un système asynchrone séquentiel (logiciel) à la fois. En fait, le système synchrone est basé sur le principe de simultanéité alors que le système asynchrone est basé sur le principe de l'entrelacement ce qui explique bien la différence de formalisme entre une architecture matérielle et les applications logicielles.

Cette technique peut se révéler très efficace pour le modèle formel qui incorpore de nombreux paramètres. De plus, elle est très souple, très flexible puisque les modèles sont construits à la main mais ceci est un inconvénient car la construction des modèles est un travail difficile, fastidieux et coûteux en temps.

Dans l'industrie, cette technique est couramment utilisée au niveau composant pour vérifier que la «netlist» obtenue après la synthèse assure bien la même fonctionnalité que celle décrite dans les fichiers VHDL/Verilog.

III.2. Simulation

La simulation se base sur l'utilisation d'un modèle comportemental du système en cours de développement. Un système possède plusieurs modèles selon les différents niveaux d'abstraction. Plus la description est à bas niveau d'abstraction plus le modèle est précis, plus les calculs pour la simulation sont nombreux et par conséquent, plus l'exécution est lente.

Cette technique de vérification est la technique la plus utilisée dans les conceptions des circuits numériques mono-puces grâce à sa flexibilité. Elle est utilisée à six différents niveaux d'abstraction.

Le **niveau spécification fonctionnelle** modélise le comportement global du système. Le but de la simulation dans ce niveau d'abstraction est la vérification fonctionnelle. Puisque le niveau d'abstraction utilisé est le plus haut alors la simulation est très rapide.

Le **niveau architectural** modélise le système comme étant un ensemble de modules qui se communiquent entre eux. A ce niveau, les différentes tâches du système sont définies à des sous-systèmes. Chaque sous-système est modélisé au niveau fonctionnel. Ce niveau est appelé aussi niveau transactionnelle qui s'intéresse aux interactions de type transaction entre les sous-systèmes. Ce type de simulation est utile pour l'exploration d'architecture et le développement des parties logicielles du système. L'outil Vista (Mentor, 2012) représente un

outil puissant de vérification matériel/logiciel au niveau architectural en adoptant le niveau TLM 2.0.

Le **niveau micro-architecture** diffère de ce qui précède au niveau type d'interaction entre les sous-systèmes. Les liaisons mis en discussion sont des signaux. La précision du modèle est donc au cycle d'horloge prêt au niveau de la communication entre les sous-systèmes. Ce niveau de modélisation permet la réalisation des premières mesures de performances et le développement des pilotes de bas niveau (des logiciels embarqués).

Le **niveau RTL** modélise un circuit comme un ensemble de registres et de relations logiques entre eux. Ce modèle est à bas niveau d'abstraction, le système entier est simulé au cycle d'horloge prêt. Ce niveau est particulièrement utilisé pour la mise au point des sous-ensembles matériels qui composent le système. Il existe plusieurs outils comme ModelSim et Questa (Mentor, 2012).

Le **niveau porte logique** décrit le système complet comme un assemblage de portes logiques. Dans la plupart des cas, ce niveau est obtenu via des outils permettant le passage du niveau RTL vers le niveau porte logique.

Le **niveau analogique** est le plus bas niveau d'abstraction utilisé en simulation. A ce niveau, existent des outils d'extraction de paramètres électriques à partir du plan de masse, citant l'outil SPICE (SPICE, 2012). On travaille ici avec des modèles précis de transistors, dépendants de la technologie utilisée (Rizatti L., 2003).

Il existe plusieurs méthodes dans la littérature pour simuler et vérifier les modules logiciels. La plupart des travaux se basent sur une simulation à base d'ISS (Instruction Set Simulator) ou bien sur une exécution natives sans ou encore avec prise en compte du système d'exploitation.

Un "Instruction Set Simulator" (ISS) est un simulateur de jeu d'instructions qui permet de simuler le logiciel à un bas niveau. C'est une simulation qui exécute le logiciel au niveau instruction assembleur. Ce type de simulation est le plus précis et le plus flexible, mais ces simulations logicielles sont lentes. De plus, pour effectuer ce type de simulation, il faut être déjà avancé dans le flot de conception car il fait usage de détails précis (comme le jeu d'instructions) sur le matériel simulé et d'un compilateur permettant de compiler le code pour le processeur cible.

Exécution native sans système d'exploitation : La simulation native permet de simuler le comportement d'un processeur. Elle utilise à la fois les codes sources des

programmes embarqués qu'elle doit exécuter, et certaines spécifications fonctionnelles du processeur, tels que les ports, les interruptions et leur gestion, contenues dans un composant spécifique. Elle permet une simulation très rapide. Les inconvénients sont l'absence de précision sur les mesures de performance, l'obligation d'avoir l'ensemble des codes sources en langage de haut niveau et la synchronisation de haut niveau. Afin de prendre en compte le système d'exploitation, une exécution native avec OS est présentée dans le paragraphe suivant.

Exécution native avec systèmes d'exploitation : La simulation native avec système d'exploitation permet une exécution multitâche. Afin que la simulation soit proche de la réalité, l'exécution native communique avec un Ordonnanceur assurant une gestion entre les tâches et leurs priorités.

Pour conclure, la simulation présente un outil très puissant vu son efficacité, sa souplesse, sa grande flexibilité, observabilité, contrôlabilité et son temps de mise en œuvre souvent court. La simulation trouve ses limites lorsqu'il faut simuler de longues séquences de tests à un bas niveau d'abstraction (Rizatti L., 2003).

La vitesse de simulation des systèmes compliqués ne dépassera pas quelques dizaines de cycles par secondes. Lorsque le simulateur doit simuler des centaines de millions de cycles, le temps de simulation devient un grand problème. Pour surmonter cette limitation, des techniques d'émulation et de prototypage matériel sont utilisées.

III.3. Emulation et prototypage matériel

Cette technique repose sur des plateformes spécifiques et reconfigurables, capables de reproduire le comportement physique d'un circuit avec une précision au niveau du cycle d'horloge. Ces plateformes sont basées sur l'utilisation d'une architecture permettant une reconfigurabilité, à savoir des FPGAs (Rizatti L., 2003), des réseaux de processeurs spécialisés (Lardière C., 2004), ou des FPGAs modifiés et adaptés aux besoins de l'émulation comme la famille Veloce de Mentor Graphics (Mentor, 2012). L'avantage des plateformes d'émulations et de prototypages est la grande vitesse d'exécution.

En fait, la différence entre l'émulation et le prototypage se manifeste au niveau de la capacité de débogages.

Les émulateurs sont conçus pour réaliser un débogage matériel rapide et efficace (figure 8). Elles ont des flots de mise en œuvre assez rapides, de l'ordre de quelques heures. D'autre part, les émulateurs offrent une très grande observabilité, proche de celle des simulateurs HDL (tous les signaux). L'émulateur a le même principe qu'un simulateur HDL sauf que sa vitesse d'exécution est bien plus rapide et peut atteindre quelques mégahertz. Ainsi, les émulateurs sont très coûteux, de l'ordre d'un million de dollars pour une licence annuelle (Rizatti L., 2003).

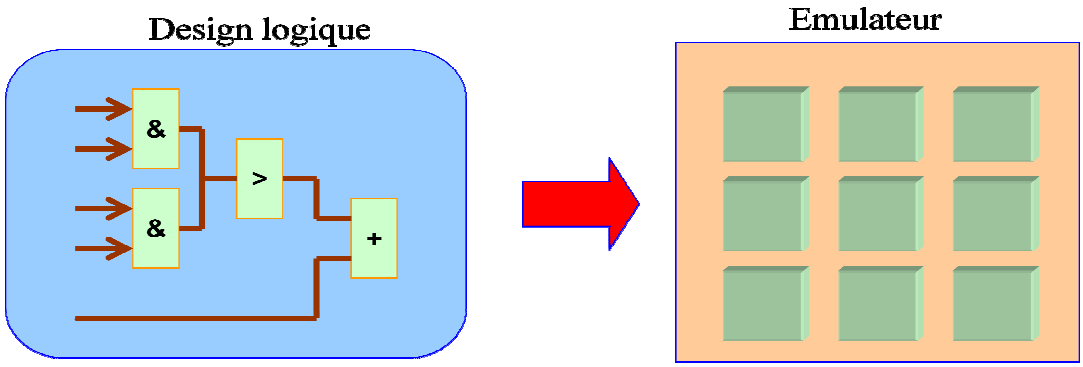


Figure 8. Principe d'émulation

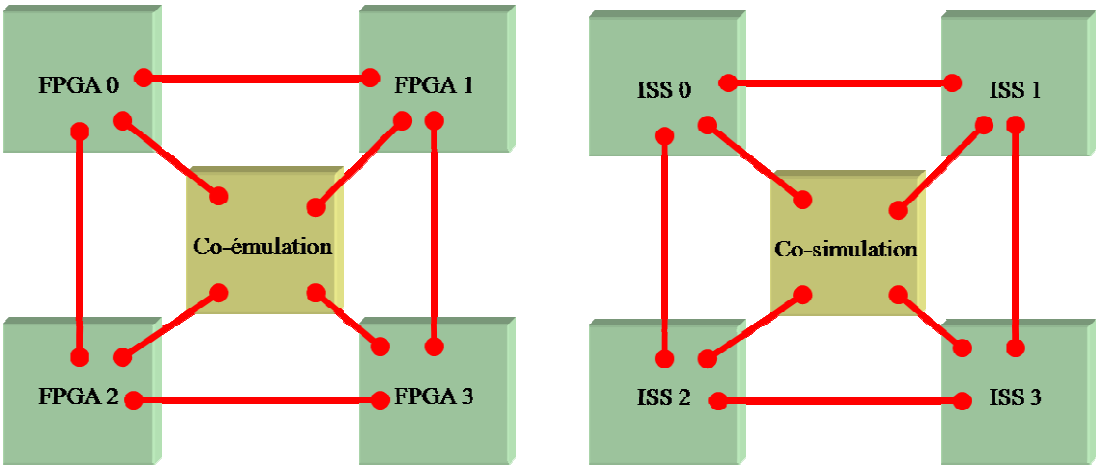
Les plateformes de prototypage sont des solutions basées sur des cartes FPGAs. Ces composants sont reconfigurables et simples à utiliser. De plus, ils sont très répandus et donc coûtent nettement moins cher que les composants spécifiques des émulateurs. Les plateformes de prototypage sont donc plus abordables que les émulateurs et offrent une excellente vitesse d'exécution, souvent supérieure à celle des émulateurs. Les meilleures solutions permettent, à mieux observer les registres du circuit sur une courte fenêtre temporelle. D'autre part, le partitionnement d'un circuit entre les différents FPGAs de la plateforme n'est pas aisé et engendre des temps de mise en œuvre assez longs, pouvant atteindre plusieurs mois (Abid M., 1998).

III.4. Co-émulation et co-simulation

La co-simulation est une technique qui se base sur plusieurs simulateurs. Dans la figure 9.b, une architecture Matériel/Logiciel est divisée en module dans chacun sera simulé par le simulateur adéquat. Prenons comme exemple une architecture qui est décrite en VHDL, SystemC et en C. Dans ce cas, les modules décrits en VHDL seront simulés par le simulateur HDL, les modules décrits en SystemC seront simulés par le simulateur SystemC et les modules logiciels seront simulés par C/C++ simulateur. En fait, cette méthode offre une

grande flexibilité au niveau modélisation et vérification mais elle consomme beaucoup plus de temps de simulation. La co-simulation est mieux exploitable lorsque la conception est dans un même niveau d'abstraction, alors que la conception multi-niveau présente un défi pour les outils de co-simulation. (Hassairi W., 2012) présente un environnement de co-simulation basé sur l'intégration du SystemC dans Matlab / Simulink. L'avantage de cette approche est la possibilité de la modélisation multi-niveau. En fait, la conception matériel et multi-niveau est assuré par SystemC et les parties logiciels sont décrit par Simulink.

La co-émulation combine l'émulation/prototypage et la simulation. Il s'agit d'une technique couramment utilisée et caractérisée par des performances en vitesse souvent faibles mais plus rapide que les environnements de co-simulation. Le concept de base est d'émuler/prototyper les parties du circuit dont la description est déjà simulée, à un plus ou moins haut niveau d'abstraction. La co-émulation (figure 9.a) se base sur plusieurs émulateurs (dans notre figure ce sont des cartes FPGAs) dont chacun un composant matériel est en interaction avec les autres composants matériels émulés dans d'autres FPGAs ou bien simulés par d'autres simulateurs. Pour cela, la plateforme d'émulation/prototypage retenue doit être capable de travailler conjointement avec un simulateur.



a) *Principe de co-émulation* b) *Principe de co-simulation*

Figure 9. Méthode de co-simulation et co-émulation

Cette technique permet de bénéficier des vitesses des émulateurs et des plateformes de prototypage. En outre, recourir à la co-émulation offre également plusieurs avantages concernant le banc de tests. Celui-ci peut en effet être décrit à un assez haut niveau d'abstraction, implémenté en C, C++ ou SystemC, ce qui est plus simple et plus rapide. Cependant, l'environnement logiciel ne peut pas atteindre la même vitesse d'exécution que l'environnement matériel donc, avec ce type d'émulation, les émulateurs/plateformes de

prototypage fonctionnent à vitesse réduite. La vitesse globale du test va alors dépendre de plusieurs paramètres :

• Vitesse d'exécution des environnements logiciel et matériel.

• Qualité de l'interface de communication.

• Nombre des points de synchronisation entre les deux environnements.

Le mécanisme de l'interface de communication et de la synchronisation présente le point clé pour chaque environnement de co-émulation. Certaines solutions fonctionnent en mode dit «ping pong» : c'est-à-dire lorsqu'un simulateur ou bien un émulateur est en exécution, l'autre est en repos. Ce mode permet une simple implémentation du modèle de synchronisation et offre une grande répétabilité des résultats mais malheureusement il fournit une accélération minime. D'autres solutions, au contraire, font fonctionner les deux environnements logiciel et matériel en parallèle, chaque environnement travaillant à son rythme, le plus rapide attendant parfois le plus lent. Cette solution est beaucoup plus rapide que la précédente. L'inconvénient majeur se présente non seulement au niveau de difficulté des règles définissant la synchronisation mais aussi pose des problèmes de répétabilité.

Enfin, le nombre des points de synchronisations entre les deux environnements impacte sur les performances, surtout lorsque la co-émulation fonctionne en «ping pong». Plus les points de synchronisation sont élevés, plus la plateforme est lente. Ce nombre des points de synchronisations varie en fonction des types de co-émulation qui vont être présentés ensuite.

III.4.1. Co-émulation en mode vecteurs de test

Le principe de ce mode se base sur la vérification du circuit entier avec émulation. Ce type de co-émulation est le plus simple de point de vue de l'environnement logiciel. A chaque cycle d'horloge, on applique un vecteur d'entrée (vecteur de test) sur les ports entrant du circuit et on compare les valeurs des ports de sorties avec le vecteur de sortie prédéterminé (figure 10).

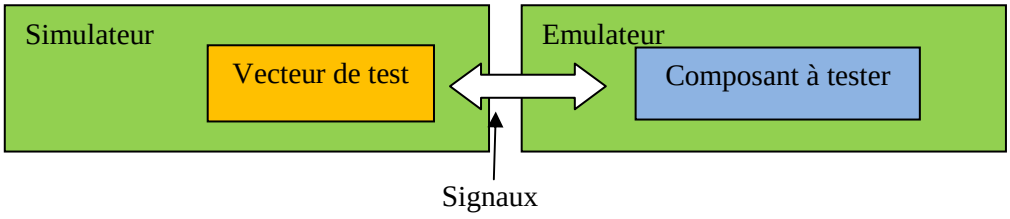


Figure 10. Principe de Co-émulation en mode vecteurs de test

III.4.2. Co-émulation avec synchronisation cycle à cycle

Dans une co-émulation avec synchronisation cycle à cycle, l'environnement logiciel représente le maître et génère les horloges du circuit émulé. L'implémentation de ce type de co-émulation est très simple. Le problème réside dans le nombre d'interaction entre l'environnement logiciel et matériel qui est important, ce qui provoque un ralentissement au niveau de la vitesse de simulation. Les facteurs limitant sont principalement, le nombre des points de synchronisation, le nombre des signaux d'entrée/sortie, la bande passante de l'infrastructure de communication et la charge de calcul de la partie simulée. Malgré le problème de synchronisation à chaque cycle d'horloge, cette méthode est utilisée surtout pour tester les composants matériels.

III.4.3. Co-émulation avec synchronisation clairsemée

Afin de surmonter le problème de synchronisation à chaque cycle d'horloge, une étude détaillée montre qu'il existe plusieurs points de synchronisation inutiles. Ces points sont marqués par l'invariance des signaux d'entrée/sortie. Une co-émulation clairsemée vise à réduire ces communications superflues. Pour cela, les horloges du circuit ne sont plus gérées par l'environnement logiciel mais par l'émulateur. Des signaux de contrôle servent alors à synchroniser l'émulateur et le simulateur. L'implémentation de cette méthode n'est pas très facile et demande certaines conditions pour être réalisable. L'étape importante et difficile consiste à définir les instants de synchronisation utile.

III.4.4. Accélération

On parle d'accélération lorsque le circuit et le banc de test sont décrits en langage matériel (VHDL et/ou Verilog) et qu'une partie du code est synthétisable alors que l'autre est non synthétisable et sera simulé à l'aide d'un simulateur HDL. Dans la figure 11, le banc de test, Module 0 et Module 1 seront simulés par un simulateur matériel alors que Module 2 sera prototypé sur FPGA afin d'accélérer la simulation. En pratique, l'accélération est une technique de vérification au niveau composant, comme la co-émulation à synchronisation cycle à cycle. Seuls les émulateurs supportent cette technique.

La principale limitation de la co-émulation à synchronisation cycle à cycle précédemment présentée réside dans le nombre élevé des points de synchronisations entre l'environnement logiciel et l'environnement matériel. La co-émulation clairsemée cherche à réduire cet impact mais, la gestion des signaux échangés est souvent très complexe. L'idée de base de la co-émulation transactionnelle est de réduire ce nombre de synchronisations au

minimum nécessaire, et ceci en faisant une abstraction de la communication. En fait, on passe d'une communication à base des signaux à une communication qui permet l'échange des données (lire/écrire) sous forme des valeurs algébriques. Ce type de communication est appelé communication à base des transactions.

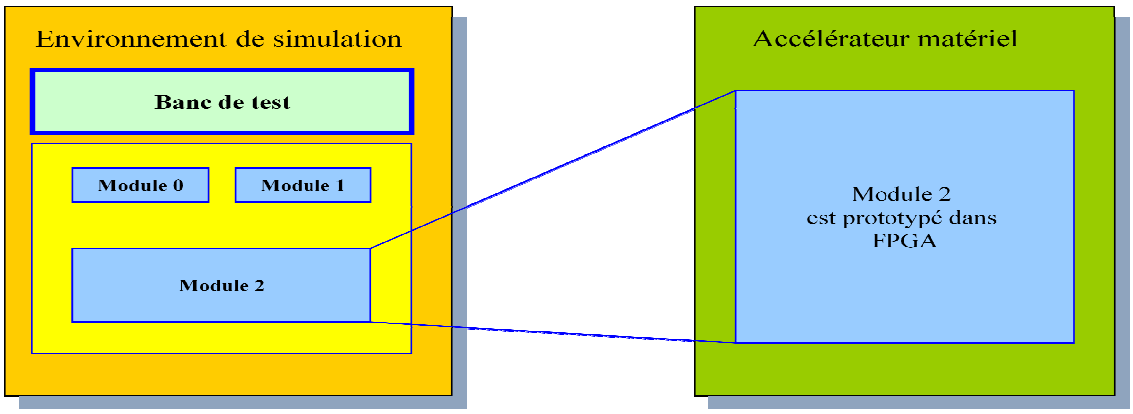


Figure 11. Simulation par accélération

III.4.5. Co-émulation transactionnelle

La transaction a donc complètement abstrait le protocole de communication de l'interface d'entrée/sortie du circuit à vérifier. Lors de la mise en application de ce concept, les utilisateurs doivent recourir à des modules matériels capables de comprendre les transactions et de les convertir en signaux et vice et versa. Ces convertisseurs matériels sont nommés «transacteurs» (Kudlugi M., 2001). Leur réalisation est complexe et représente la principale difficulté des équipes de vérification désireuses de recourir à cette performante technique de co-émulation. L'architecture d'une co-émulation transactionnelle comporte un environnement logiciel et un environnement matériel communiquant à l'aide des canaux de communication. Une norme nommée SceMi (Standard CoEmulation Modeling Interface) a standardisé ces canaux.

Les applications les mieux adaptées à ce type de co-émulation sont les circuits gérant des flux de données comme les circuits de télécommunication. De plus, il est à noter que la contrainte du développement des transacteurs peut être réduite par l'utilisation d'une bibliothèque de transacteurs.

III.4.6. Emulation avec banc de test intégré

Dans une émulation avec banc de test intégré, le circuit à vérifier et son banc de test sont tous deux des composants matériels émulés. Cette méthode représente une amélioration

de la méthode d'accélération. L'émulateur ou la plateforme de prototypage fonctionne d'une manière complètement autonome ce qui rend le fonctionnement en pleine vitesse. Cette technique est donc particulièrement bien indiquée pour les vérifications nécessitant de longues séquences de test à savoir la vérification au niveau système. Cette technique est également bien adaptée pour la mise au point des logiciels embarqués. La principale difficulté de cette technique d'émulation réside dans le développement du banc de test synthétisable. Cela représente une grosse charge de travail car il faut développer un composant matériel de test spécifique et le valider.

III.4.7. Emulation avec dépendances extérieures

Une émulation avec dépendances extérieures consiste en un émulateur connecté à un environnement physique extérieur et le circuit fonctionne alors en temps réel. Ce mode est utilisé avec des plateformes de prototypage pour développer essentiellement des logiciels embarqués ou pour développer des logiciels associés au circuit (drivers).

Dans cette section, de différentes méthodes de vérification ont été citées. La méthode la plus adéquate pour tel environnement et pour tel système continu/discret se base sur une co-simulation accélérée par un accélérateur matériel réel. La section suivante présentera notre approche de vérification adaptée.

IV. Approche de simulation/émulation matériel/logiciel

Nous avons montré dans le chapitre 1, que les modèles de co-simulation souffrent encore du temps de simulation important. En effet, l'environnement CODIS donne un temps de simulation assez important car il utilise un ISS pour simuler les parties logicielles. Notre approche propose le remplacement de l'ISS par une architecture à base d'un processeur cible implanté sur une carte FPGA. Il est indéniable qu'une simulation sur un processeur réal est énormément plus rapide qu'un ISS et de l'ordre de quelques nano secondes. Les travaux antérieurs utilisent le simulateur pour simuler les modules logiciels et l'émulateur pour simuler les composants matériels. Deux points faibles suivent ces travaux. La première se manifeste au niveau temps de modélisation important des composants matériels au niveau RTL. La deuxième consiste au temps de simulation important des modules logiciels.

Afin d'étendre l'environnement CODIS et d'accélérer la simulation, nous proposons tout d'abord une modélisation au niveau transactionnelle des composants matériels en

utilisant SystemC. Ensuite, une simulation des modules logiciels sur l'architecture cible est adoptée. La figure 12 résume l'approche de simulation/émulation choisie.

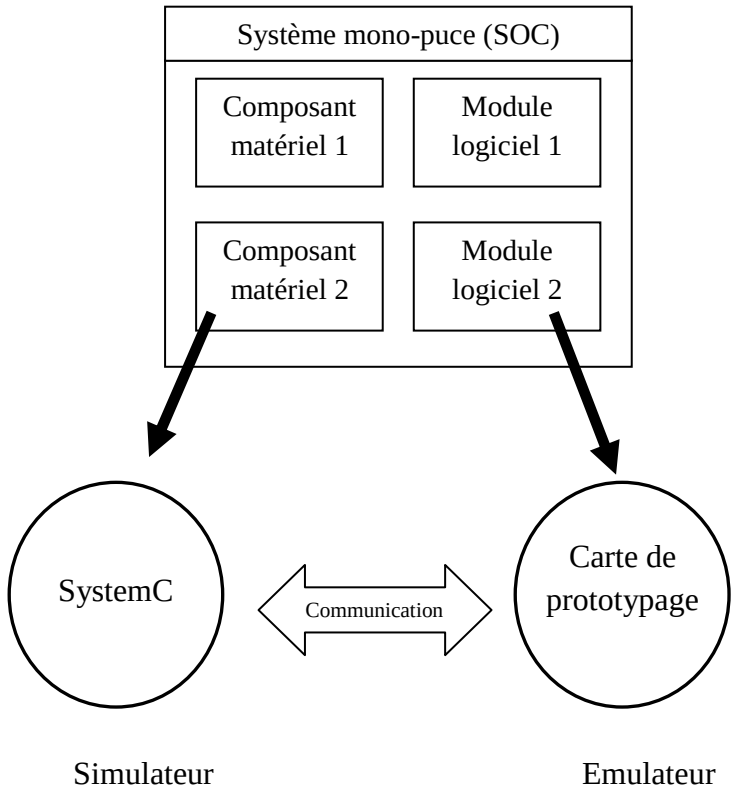


Figure 12. Approche de simulation/émulation

On peut résumer les avantages de notre environnement essentiellement en trois points :

- Le remplacement d'un ISS par le processeur cible accélère le temps d'exécution des applications logicielles. Dans notre cas la simulation est basée sur une architecture à base d'un processeur cible hard-processor implanté sur FPGA.
- Les composants matériels sont modélisés et simulés avec SystemC, ce qui diminue le temps de mise en marché et assure la modélisation suivant plusieurs niveaux d'abstractions.
- Les modèles de synchronisation entre le simulateur SystemC et l'émulateur sont adaptés au modèle de co-simulation de l'environnement CODIS.

La section suivante présente le moteur de la simulation/émulation.

V. Moteur de simulation / émulation

La conception d'un moteur de simulation / émulation constitue un point clé pour chaque environnement de co-vérification. Le principal rôle de ce moteur est d'assurer à la fois la communication et la synchronisation entre le simulateur SystemC et l'émulateur à base d'une architecture cible implantée sur une carte FPGA.

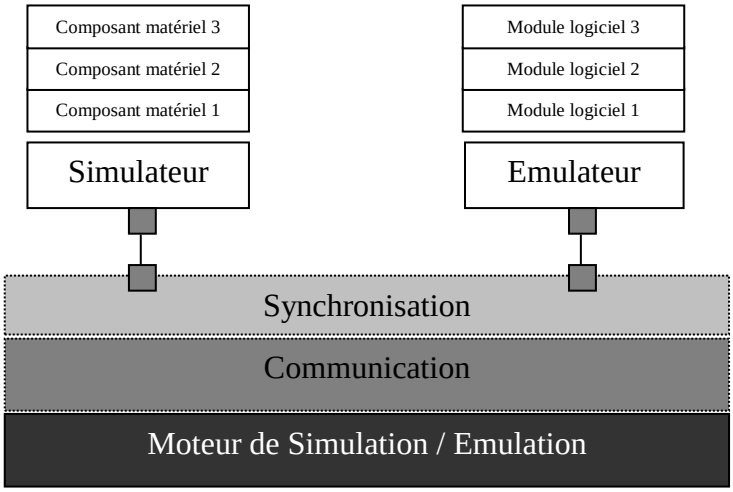


Figure 13. Architecture du moteur de simulation / émulation

L'architecture du modèle de co-simulation est illustrée par la figure 13. Le moteur de simulation/émulation supporte une couche de synchronisation et une couche de communication.

- La couche de communication est chargée de transférer les données entre les deux modèles, les conversions nécessaires des signaux et le changement de contexte.
- La couche de synchronisation assure à la fois le contrôle et l'exécution du simulateur/émulateur à des instants précis.

Des modèles de synchronisation sont utilisés pour exécuter le simulateur et l'émulateur matériel/logiciel avec respect du temps d'échange de donnée.

Principalement trois difficultés ont été étudiées :

1. La détection de la fin d'un événement par l'émulateur.

2. La détection des interruptions par l'émulateur.

3. Le principe de changement de contexte qui permet le passage du simulateur vers l'émulateur et vice versa.

Par la suite, nous présentons une description détaillée de la couche communication et de la couche synchronisation.

V.1. Communication

Le principal objectif de la communication est l'échange sans perte de données. Dans la littérature, deux modes de communication (série et PCI) entre simulateur/émulateur sont utilisés (Soha H., 2005). Afin d'accélérer le temps de simulation qui est notre premier objectif, une communication Universel Serial Bus (USB) est utilisée. Le tableau 2 montre les différences de vitesse de transfert entre les différentes modes de communication.

	Série	PCIe	USB 1.0	USB 2.0
Vitesse	0.11 Mo/s	250 Mo/s	12Mo/s	**480 Mo/s**

Tableau 2: Vitesse de transfert

Nous proposons une communication à base d'USB 2.0 entre un ordinateur et une carte FPGA grâce à deux avantages : (1) vitesse de transfert important (2) supporte le mode interruption qui engendre une interruption matériel sur la carte. Pour cela, un pilote composé d'une partie logicielle et une partie matérielle doit être développé. La figure 14 présente l'architecture matériel/logiciel utilisée pour la communication. Il faut noter que le contrôleur USB ne représente pas un composant de l'architecture cible.

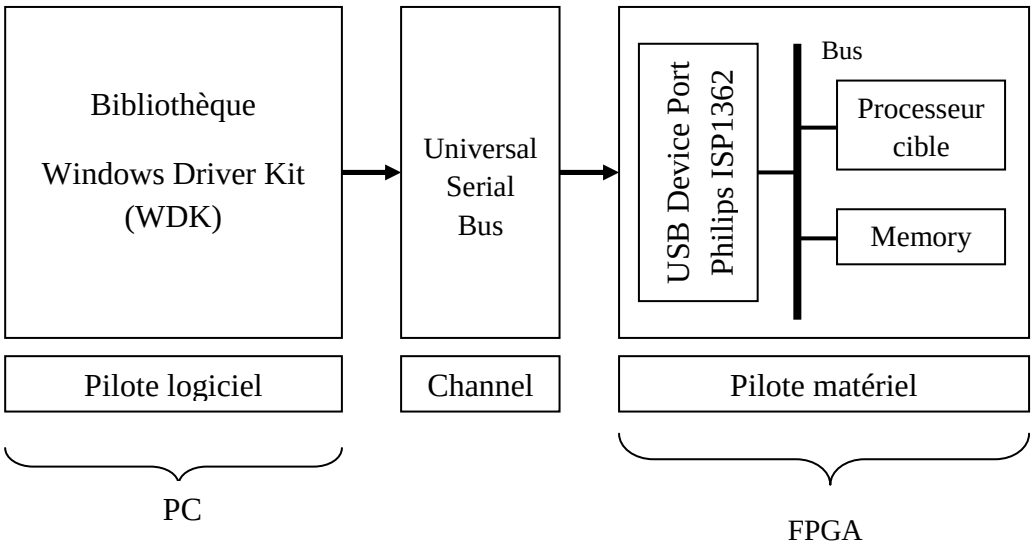


Figure 14. Modèle de communication

- **Coté logiciel :**

Le pilote du côté logiciel se base sur le Windows Driver Kit (WDK) Version 7.1. Ce Kit offre les éléments de base pour la création des pilotes sous le système d'exploitation Windows. Nous avons développé une bibliothèque qui contient essentiellement deux fonctions *Ecrire( )* et *Lire( )* pour l'échange de donnée. Ces deux fonctions utilisent quatre fonctions de la bibliothèque Win32 :

- *CreateFile*: permet la connexion avec le contrôleur USB ISP1362.
- *WriteFile*: permet le transfert de données vers le contrôleur USB ISP1362.
- *ReadFile*: permet la réception des données de la part du contrôleur USB ISP1362.
- *ControlIODevice*: permet la configuration du pilote.

  - **Coté matériel :**

Le côté matériel se base sur le contrôleur USB ISP1362 (ISP1362, 2002) en interaction avec le processeur cible. Le rôle du contrôleur USB est d'assurer les fonctionnalités suivantes :

- La fonction Host Controller (HC) est basée sur un transfert avancé et atteint une vitesse de transfert élevé avec une faible intervention du processeur.
- La fonction On-The-Go (OTG) est adoptée lorsque la liaison USB ne demande pas l'intervention du processeur.
- La fonction Device Controller (DC) assure principalement le transfert de donnée et laisse le rôle du contrôle au processeur.

Dans notre cas, nous avons utilisé la fonction DC pour que le processeur cible gère le modèle de communication au lieu du contrôleur USB.

Une interruption est signalée au processeur par un signal électrique sur la borne *INT0*. Lors de la réception de ce signal, le processeur traite l'interruption dès la fin de l'instruction qu'il était en train d'exécuter. Le traitement de l'interruption consiste soit à :

- Ignorer et passer normalement à l'instruction suivante : c'est possible uniquement pour certaines interruptions, nommées interruptions masquables. Il est en effet parfois nécessaire de pouvoir ignorer les interruptions pendant un certains temps, pour effectuer des traitements très urgents par exemple. Lorsque le traitement est terminé, le processeur démasque les interruptions et les prend alors en compte.

- Exécuter un traitant d'interruption (interrupt handler). Un traitant d'interruption est un programme qui est appelé automatiquement lorsqu'une interruption survient. L'adresse de début du traitant est donnée par la table des vecteurs d'interruptions. Lorsque le programme d'interruption traitant a effectué son travail, il exécute l'instruction spéciale *IRET* qui permet de reprendre l'exécution à l'endroit où elle avait été interrompue. Un ordonnanceur est mis en place pour gérer le traitant d'interruption.

La communication ainsi décrite représente un cadre solide pour le modèle de synchronisation présenté dans la section suivante.

V.2. Modèle de synchronisation

Vu l'importance du modèle de synchronisation et son interaction avec le noyau de SystemC, une description de l'environnement SystemC est présentée. Nous proposons par la suite dans une première partie les différents modèles de synchronisation possible entre le simulateur SystemC et l'émulateur. Dans la deuxième partie, une étude sur les interfaces matérielles / logicielles et les scénarios de synchronisation sont décrits.

SystemC

SystemC est un simulateur à noyau libre qui décrit toute une bibliothèque contenant des composants matériaux. Son langage est une extension par classes du langage orienté objet C++ pour la description des systèmes numériques. SystemC offre la possibilité de la description au niveau RTL comme il la permet au niveau système (SystemC 2.0 et les versions ultérieures) pour les systèmes implémentés en logiciel, matériel ou une combinaison des deux. Un modèle décrit en SystemC est compilé, exécuté et débogué en utilisant les outils standards de programmation C++. SystemC diffère aux autres langages de descriptions matérielles comme VHDL et Verilog par la possibilité de supporter plus qu'un niveau de description. SystemC permet de fournir encore une spécification du système à des niveaux d'abstraction élevés et avec une meilleure vitesse de simulation. Malheureusement, SystemC ne peut décrire que des systèmes discrets, cependant il ne supporte pas la description des systèmes continus.

SystemC 2.0 et les versions ultérieures combinent les caractéristiques des langages matériels existants, la technique d'orienté objet et de nouvelles méthodologies pour la conception et le raffinement des systèmes matériels/logiciels. Sa méthodologie est inspirée du modèle de communication introduit par Gajski (Gajski D.D., 2000). Dans cette méthodologie

le modèle est composé par des modules et la communication entre eux est assurée par des canaux. Un module est constitué par des méthodes et des interfaces. Les méthodes utilisées pour la communication sont définies dans les interfaces des modules. Leur implémentation est effectuée au niveau de ces canaux. Un module peut appeler une méthode fournie par un canal et des événements dans le canal peuvent activer les processus du module connecté à ce canal. Ce concept est assez générique pour décrire des systèmes en utilisant plusieurs domaines de description comme les réseaux de Khan, les processus séquentiels communicants, les flux de données multi-cadencés, les événements discrets, etc. Chaque module contient les processus décrivant le comportement du système. La connexion entre les différents modules est effectuée au niveau de la fonction *sc_main ()* qui représente l'entête du modèle. Aujourd'hui, plusieurs outils de conception, supportant SystemC aux différents niveaux d'abstraction, sont disponibles sur le marché. SystemC-RTL est synthétisable et un flot de conception partant du niveau spécification au niveau circuit est aujourd'hui disponible.

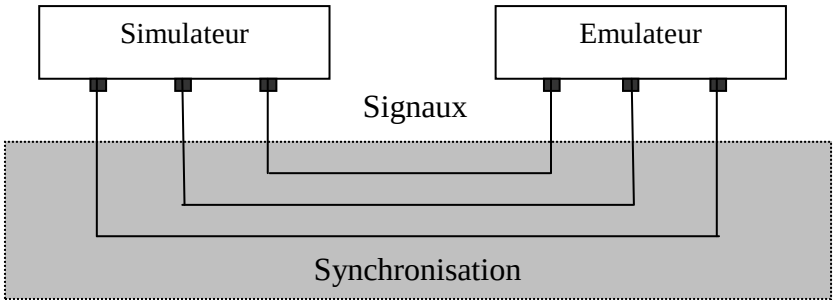
Le simulateur de SystemC est à base d'un ordonnanceur à événements discrets. Dans SystemC, un cycle delta comporte deux phases à savoir : phase d'évaluation pour l'exécution des processus et phase de mise à jour pour la mise à jour des signaux modifiés pendant l'évaluation des processus, ce qui garantit l'aspect parallèle des processus (Salem A, 2003). Le principal rôle de l'ordonnanceur est de déterminer l'ordre d'exécution des processus en considérant leurs listes de sensibilité et les événements dans sa file d'attente. Ainsi, le premier élément dans cette file représente le prochain événement à déclencher. Les événements sont classés en deux types : événements différés par une durée de temps et événements différés par un delta. Le temps d'occurrence du premier type d'événements représente le prochain temps réel alors que le temps d'occurrence du deuxième type d'événements est constitué de deux composants: le temps courant réel et le nombre de cycles delta; la file d'attente est ordonnée selon ces deux composants.

V.2.1. Les modèles de synchronisation simulateur/émulateur

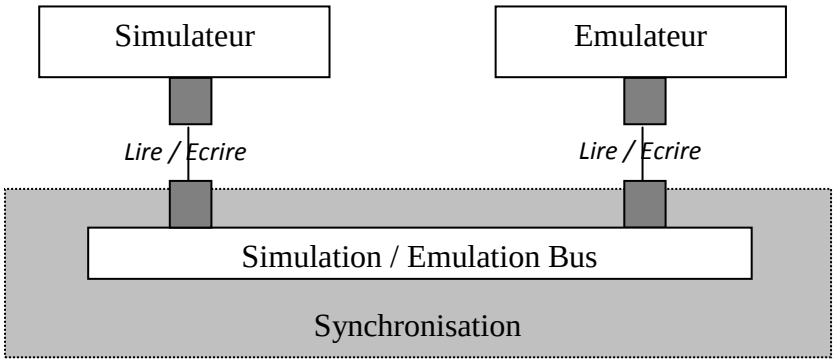
Nous précisons tout d'abord que le simulateur SystemC est le maître de l'environnement de la vérification. Le modèle de synchronisation peut être décrit dans différents niveaux d'abstraction. Dans le niveau RTL le simulateur et l'émulateur sont connectés via des signaux ce qui augmente le nombre des points de synchronisation, figure 15 a).

En contre partie, le niveau transactionnel (TLM) est adopté pour les premières phases de description des systèmes car il assure l'abstraction de la partie communication ce qui réduit le

nombre des points de synchronisation. Le niveau TLM repose sur la notion de bus de synchronisation comme l'indique la figure 15 b).



a) Bus de Simulation/Emulation (RTL)



b) Bus de Simulation/Emulation (TLM)

Figure 15. Bus de simulation/émulation

Le bus de la simulation/émulation implémente les caractéristiques du bus en relation avec le processeur cible et assure des transactions de deux types d'interruptions comme l'indique la figure 16.

Les fonctions *Lire()* et *Ecrire()* définissent les fonctions élémentaires des paquets utilisés dans le bus de simulation/émulation. Deux types de paquets sont construits afin d'assurer le bon fonctionnement des modèles de synchronisation : paquet d'interruption et paquet de donnée (figure 16). Le paquet d'interruption contient un en-tête qui définit le type du paquet et un corps composé du numéro de routine à exécuter et le temps éventuel de la fin de la tâche. Le paquet de donnée contient aussi un en-tête qui définit à la fois le type du paquet, la taille de donnée et un corps qui enveloppe les données à transférer.

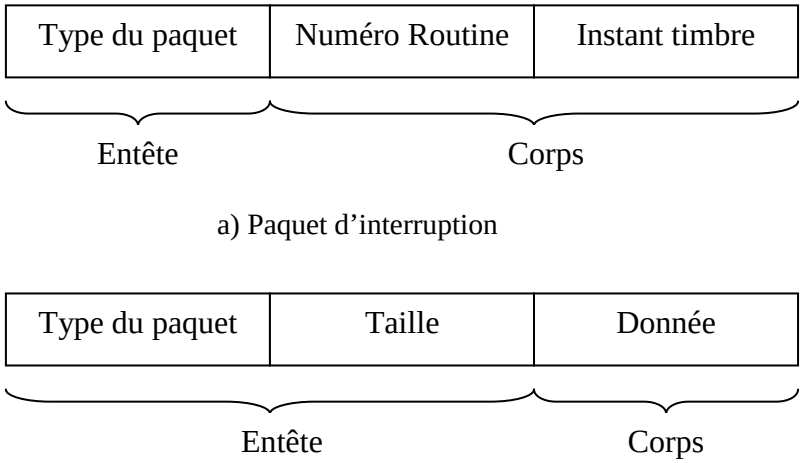


Figure 16. Forme de synchronisation

Quatre schémas de synchronisation possibles sont réalisés entre le simulateur et l'émulateur :

- **Schéma 1: L'application logicielle reçoit périodiquement les données de la tâche matérielle.**

Le modèle de synchronisation est basé sur les mémoires de type FIFO. L'idée de ce modèle consiste à fixer une période de synchronisation (T_{sync}) entre le simulateur (SystemC) et l'émulateur (architecture cible) (figure 17) imposé par le simulateur. Dans ce cas, le processeur cible suit le rythme du simulateur. Une période de synchronisation est le temps qui sépare deux points de synchronisations successifs. La période de synchronisation doit être fixée supérieure aux temps d'exécution du tâche logicielle la plus longue.

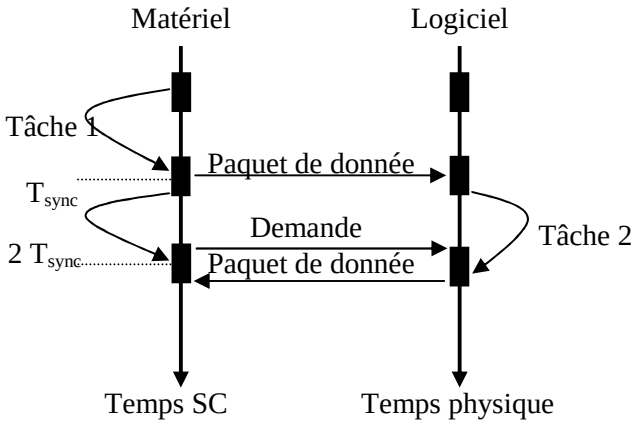


Figure 17. Modèle de synchronisation: schéma 1

- **Schéma 2: L'application logicielle est en attente de la fin de la tâche matérielle.**

Lorsque les composants matériels sont simulés en SystemC, les applications logicielles sont en attente. Une interruption est envoyée pour indiquer la fin de la tâche lorsque le simulateur termine les siennes. A cet instant l'émulateur commence à exécuter la routine correspondante (figure 18) et la tâche matérielle entre en repos pour le prochain point de synchronisation.

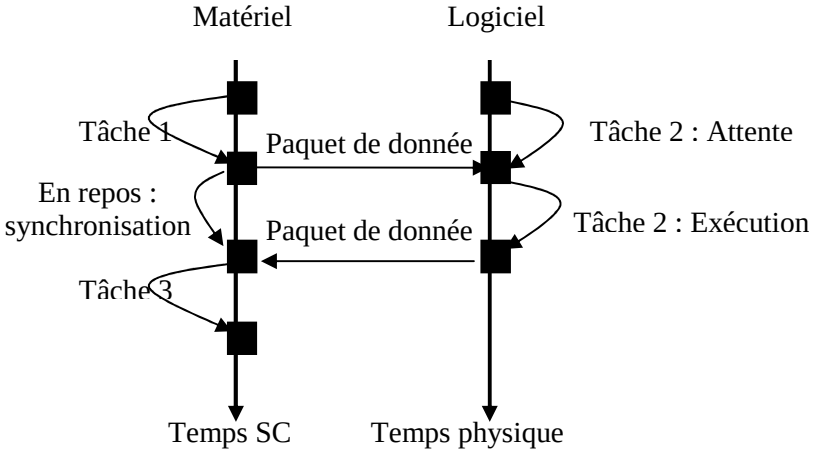


Figure 18. Modèle de synchronisation: schéma 2

- **Schéma 3: L'application logicielle reçoit une interruption avant la terminaison de la tâche matérielle.**

Dans ce modèle (figure 19) l'application logicielle peut s'exécuter plutôt qu'elle reste en attente lorsque la simulation des composants matériels est en cours. En effet, ce parallélisme est assuré par le mode d'interruption matérielle de la liaison USB. L'ordonnanceur de l'architecture cible envoie une interruption déclenchant (flèche 0) vers la tâche matérielle pour commencer la simulation. A la fin de la simulation de la tâche 1 (flèche 1), le simulateur SystemC envoi une interruption (flèche 2) afin d'informer l'application logicielle du prochain éventuel instant de synchronisation.

Eventuellement, cet instant qui correspond à la fin de la tâche hardware est envoyé en utilisant la fonction *wait_for_interrupt(sc_time t)* (figure 20). A cet instant l'ordonnanceur active un temporisateur et exécute une tâche intermédiaire (flèche 3). Lorsque le temps est atteint, la tâche intermédiaire s'arrête. L'ordonnanceur envoie une demande de donnée pour recevoir le paquet de donnée et activer la tâche 2 (mentionné par le paquet). La figure 21 montre un modèle de code de synchronisation.

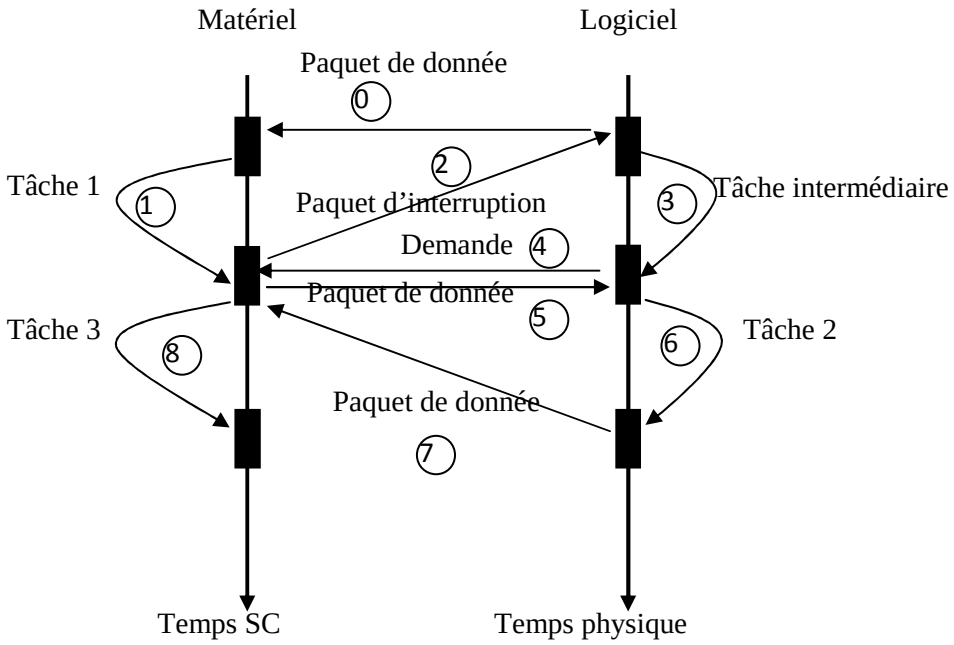


Figure 19. Modèle de synchronisation: schéma 3

```
void wait_for_interrupt(sc_time t)
{
wait(t);
send_interruption_packet(…..) ;
}
```

Figure 20. Code de la fonction attente d'une interruption

Avec *t* une estimation de la durée d'exécution de la tâche

```
            /* Task1 code */
Instructions
….
….
Wait_for_interrupt (t);
Switch_context();/* switch context to SC*/
```

Figure 21. Modèle du code de synchronisation

- **Schéma 4: L'application logicielle reçoit les données au hasard**

Dans ce cas (figure 22), le SystemC exécute la tâche 1 et lorsque ce dernier est fini, un paquet de donnée est envoyé vers l'application logicielle. Lorsque la tâche 2 est en exécution,

le SystemC exécute la fonction *Hardware_Input_Interface.* Cette fonction qui modélise les interfaces d'entrée du composant matériel, s'exécute sans avancer le temps local du SystemC. *Hardware_Input_Interface* peut générer une interruption au hasard pour informer l'application logicielle de l'instant d'arriver d'un paquet de donnée. L'interruption est bien assurée par l'interruption matérielle générée par l'USB.

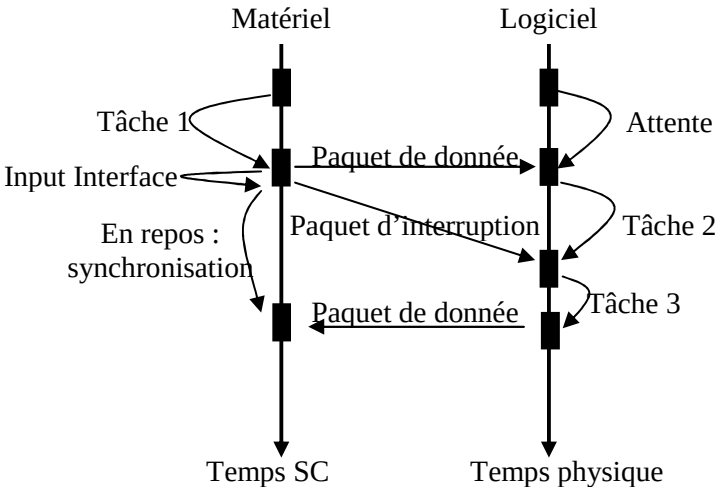


Figure 22. Modèle de synchronisation: schéma 4

Les modèles de synchronisation ainsi présentés peuvent être utilisés ensemble en se basant sur les fonctions dédiées de chaque modèle.

V.2.2. Les interfaces de synchronisation

Afin d'assurer la synchronisation, des interfaces logicielles et matérielles sont implémentées. Du côté de la carte FPGA un tableau de registres est utilisé pour sauvegarder le contexte de la synchronisation.

Les composants matériels sont décrits en SystemC, ce qui facilite le mouvement, l'addition et la soustraction des modules. Le niveau TLM est le niveau d'abstraction adopté pour les interfaces *Interface_In*, *Interface_Out* et pour la modélisation. Un module interface est implémenté pour assurer une bonne synchronisation. La figure 23 montre les interfaces utilisées.

Notre environnement de simulation / émulation possède essentiellement deux types d'interfaces dans SystemC :

1. L'interface *Interface_In* : c'est une interface qui permet de lire les données reçues de la part de l'émulateur. Cette interface implémente la couche de communication et permet non seulement l'accès au tableau de registre partagé

pour récupérer les données désirées mais aussi d'effectuer un changement de contexte.

2. L'interface *Interface_Out* : c'est une interface qui permet d'envoyer les données vers les applications logicielles et de lancer le changement du contexte. Cette interface implémente la même couche que l'interface *Interface_In*.

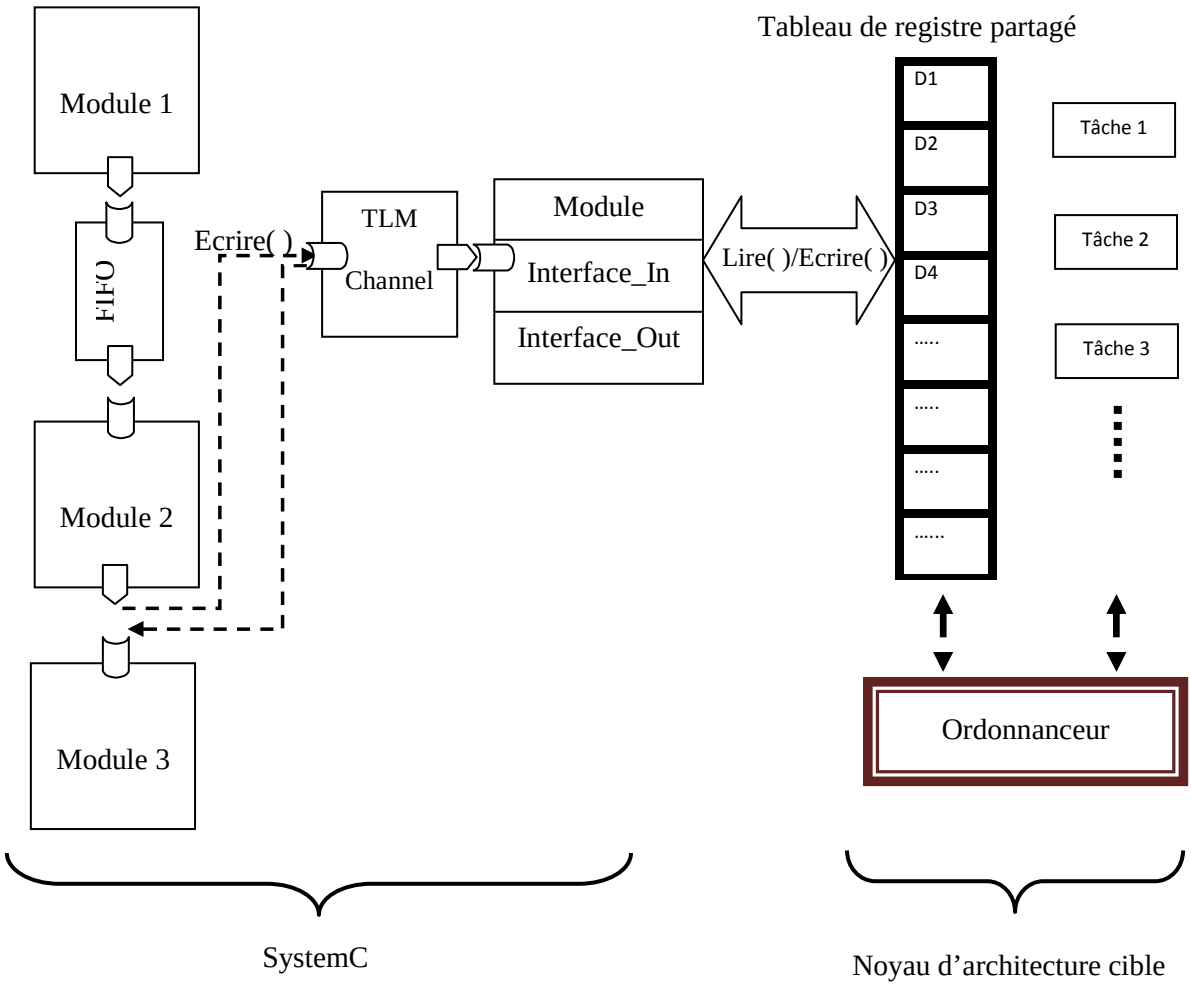


Figure 23. Les interfaces de synchronisation

VI. Conclusion

Une présentation des méthodes de vérification des systèmes mono-puces basés sur une architecture matériel / logiciel a été détaillée dans ce chapitre. En se basant sur les techniques décrites, nous avons présenté notre modèle et environnement de simulation/émulation qui permet d'accélérer le temps de simulation et la modélisation multi-niveau. Notre environnement est composé d'une couche de communication à base de la liaison USB et une couche de synchronisation. Cette dernière couche assure à la fois l'abstraction de l'échange de

donnée entre simulateur et émulateur sous forme de paquet et fournis les modèles de synchronisations possibles. En fait cette abstraction permet de conserver les modèles de synchronisations même si le type de liaison change. Seule la couche de communication sera modifiée.

Les méthodes de synchronisation ainsi décrites sont génériques et ne demandent pas de changement du noyau de la simulation du SystemC et ceci est grâce au mode d'interruption assuré par la liaison USB. Ces modèles assurent à la fois une communication rapide et une synchronisation supportant plusieurs scénarios possibles. Les résultats expérimentaux sont présentés dans le chapitre 4.

Dans le chapitre suivant, nous proposons une extension de l'environnement de simulation/émulation que nous avons ultérieurement présenté afin de supporter le modèle continu sur lequel CODIS est basé. Le principe consiste à interfacer notre environnement avec le simulateur Simulink.

Chapitre 3 : METHODOLOGIES DE MODELISATION ET DE VERIFICATION POUR LES SYSTEMES HETEROGENES

I. Introduction

La modélisation des systèmes hétérogènes présente un grand défi pour les concepteurs vu l'hétérogénéité entre simulateurs qui demandent une interaction adéquate entre le modèle continu et le modèle discret d'une part et entre la partie matérielle et la partie logicielle d'autre part. Les systèmes de contrôle-commande sont des systèmes industriels très utilisés et ils obéissent aux règles des systèmes hétérogènes. En fait, les systèmes de contrôle-commande sont composés principalement de deux parties : partie commande et partie contrôle. La partie commande est constituée par des modules continus comme les moteurs et les réacteurs. La partie contrôle est constituée d'une unité de traitement numérique adéquate au modèle discret. Le nombre de contrôleurs numériques ainsi leur complexité ne cesse d'augmenter et plus d'efforts sont consacrés à la conception et à la vérification. Grâce à la grande révolution des technologies numériques, plusieurs outils conçus pour les contrôleurs numériques sont créés. Les plateformes de type VLSI (Very Large Scale Integration), comme par exemples les cartes FPGAs et les ASICs (Application Specific Integrated Circuit), réalisent des contrôleurs entièrement numériques. Par conséquent, l'unité de contrôle des systèmes de commande implantée souvent sur une carte électronique, migre vers une implantation sur une puce unique, offrant l'avantage d'être compact et de supporter un très grand nombre de traitements arithmétiques. De plus, l'utilisation des cartes reconfigurables telles que les FPGAs permet le développement et le prototypage rapide du contrôleur numérique, (Rodriguez J.J., 2007).

La méthodologie de conception et de vérification s'avère une demande exigeante à cause de la complexité croissante des algorithmes à implanter dans ces contrôleurs numériques et les contraintes de la mise en marché.

Ce chapitre est constitué de deux parties essentielles : dans la première nous proposons une nouvelle technique de conception et de simulation des contrôleurs numériques nommée la technique de simulation matériel/logiciel en boucle ("Hardware Software In the Loop"). Nous décrivons dans la deuxième partie l'environnement CODIS+ en se basant sur les concepts de base de l'environnement CODIS.

II. Simulation matériel/logiciel en boucle des contrôleurs numériques

La conception des contrôleurs au sein des systèmes commandes présente un défi à cause de l'hétérogénéité du modèle. Une présentation des techniques de modélisation et de vérification est citée dans cette section.

Dans tout système de commande, les contrôleurs numériques interagissent avec les différents modules continus. Les différents signaux qui peuvent interagir avec ce contrôleur rendent la conception plus difficile. On peut y distinguer les signaux reçus par le système de commande et ceux qu'il émet. Afin d'interfacer le modèle continu et le modèle discret des convertisseurs sont utilisés.

Les signaux émis correspondent aux ordres de commande à l'ouverture et à la fermeture des interrupteurs des convertisseurs. De plus, le contrôleur numérique est, à part sa nature, un système discontinu qui ne réagit avec son environnement qu'à des instants discrets. Ces instants sont soumis à des contraintes temporelles dont l'ordre de grandeur peut varier de la seconde à la microseconde selon la dynamique des grandeurs à réguler.

II.1. Travaux antérieurs

Plusieurs travaux se trouvent dans la littérature pour la modélisation et la vérification des contrôleurs numériques. Cette section détaille les différentes méthodes en soulignant les points forts et les points faibles de chaque méthode.

II.1.1. Simulation utilisant une carte électronique

Les premiers travaux réalisent le contrôleur numérique sur une carte électronique à travers des différents composants discrets. Ces composants réalisent des fonctions particulières plus ou moins complexes : addition, mémorisation, interfaçage, gestion d'interruption, …etc. L'inconvénient majeur de cette technique se manifeste lors d'une erreur de conception. Afin de la corriger, on doit ajouter des liaisons entre les composants ou bien refaire totalement la carte électronique. Ce problème n'obéit pas à la contrainte de la mise en marché. De plus la conception de ces cartes devient de plus en plus difficile à cause de la complexité des contrôleurs électroniques et des composants qui deviennent nombreux. Cela engendre une élévation du prix des cartes conçus.

II.1.2. Simulation matériel en boucle

Afin de surmonter ces lacunes, les techniques basées sur la simulation matérielle en boucle ("Hardware In the Loop") (HIL) sont proposées pour pouvoir modifier le contrôleur numérique sans modifier la carte électronique et diminuer le nombre de composants numériques sur cette carte. L'évolution des technologies de fabrication de circuits numériques permettent l'intégration d'un contrôleur numérique sur une mono-puce. La conception de tels systèmes numériques intégrés se base sur un langage numérique de description de matériel.

Cette évolution a également ouvert la voie aux langages de haut niveau de description de matériel, encore appelés HDLs pour "Hardware Description Languages", il s'agit en particulier de VHDL et de Verilog. Tous deux sont supportés par un grand nombre de logiciels. Les avantages majeurs d'une description basée sur un HDL résident dans sa portabilité et dans son caractère exécutable. En effet, un modèle fonctionnel numérique décrit à haut niveau par un HDL peut être vérifié par simulation, avant que la conception finale ne soit réalisée. D'autre part, la révolution dans les outils CAO permet le passage directement d'une description HDL synthétisable à un schéma à base de portes logiques.

Une première technique utilise un simulateur mixte analogique/numérique supportant un HDL et intégrant un noyau de simulation unique, par exemples Advanced Design System (ADS) d'Agilent (Agilent, 2012), ADVanceMS (Mentor, 2012), Simplorer (Simplorer, 2012) et SMASH (Smash, 2012).

La co-simulation est la seconde technique possible. Elle est basée sur la communication entre deux simulateurs, l'un numérique et l'autre analogique. Les modèles sont conjointement exécutés par ces deux simulateurs, chaque simulateur modélisant une partie spécifique du circuit à concevoir ou de son environnement. La co-simulation est basée sur une interface qui permet non seulement l'échange de données entre les deux simulateurs tout en respectant les contraintes de types et de tailles mais aussi en respectant la synchronisation temporelle des deux simulateurs par exemples Modelsim/SpectreS (langages VHDL/SpectreHDL) (Aubepart F., 2003), Modelsim/Saber (Lienhardt A.M., 2006), Modelsim/Matlab (Katrib J., 2008).

Dans ces exemples, le simulateur Modelsim simule le contrôleur numérique décrit en langage HDL numérique. Le modèle du contrôleur s'intègre sous la forme d'un bloc numérique dans l'environnement global. Le modèle peut être décrit à différents niveaux d'abstraction, du plus haut niveau jusqu'au niveau synthétisable.

En résumé, les avantages de la technique basée sur un simulateur mixte supportant un HDL sont : (1) noyau de simulation unique (2) temps de simulation réduit.

L'inconvénient majeur réside à la limitation de bibliothèques des composants analogiques. La co-simulation surmonte l'inconvénient précédent, mais la synchronisation et le temps de simulation représentent le grand défi.

Comme une première solution pour réduire le temps de simulation, les sociétés Altera et Xilinx, fabricants de composants FPGAs, ont développé un code VHDL niveau synthétisable associé aux modèles hauts niveaux de la bibliothèque Matlab/Simulink. L'outil DSP Builder de la société Altera et l'outil System Generator de la société Xilinx permettent alors la génération "automatique" d'une description VHDL synthétisable à partir d'un modèle dans l'environnement de simulation Matlab/Simulink.

Cette nouvelle approche bénéficie de nombreux avantages : bibliothèques riches en composants numériques, analogiques et possibilité de réaliser la synthèse numérique.

La simulation HIL traditionnelle, basée sur un simulateur ou bien sur une plateforme matérielle spécifique à une application, permet entre autres aux concepteurs d'évaluer un algorithme de commande conjointement avec le contrôleur numérique (partie matérielle) par une simulation qui reproduit le comportement dynamique du système. Il est dès lors possible d'évaluer l'algorithme de commande dans un environnement virtuel, non destructif où les modifications de l'algorithme sont souvent réalisables sans itération matérielle coûteuse. Cette technique de simulation entraîne une réduction des temps de développement ainsi que la réduction du coût d'un projet. Ainsi, la simulation HIL permet d'évaluer la robustesse et les performances de l'algorithme de commande et les points faibles du système.

La simulation HIL peut être réalisée en temps réel ou hors ligne, selon le type de simulateur utilisé. Dans le cas de la simulation HIL hors ligne, à chaque pas de simulation, le système est simulé en utilisant un simulateur "off line". Les signaux de sortie sont envoyés au contrôleur numérique qui exécute l'algorithme implémenté. Le contrôleur retourne ensuite les signaux de commande. À cet instant, un cycle de simulation HIL hors ligne est effectué. De ce fait, la simulation ne peut pas être exécutée en temps réel et peut devenir très lente lorsque l'on diminue le pas de simulation ou lorsque le système est complexe avec une dynamique lente. En dépit de ce point faible, cette approche peut notamment être très efficace pour évaluer un algorithme de commande, en particulier lorsque le pas de simulation est très faible.

En effet, dans ce cas, certains simulateurs temps réel ne peuvent plus simuler correctement tels systèmes.

Dans l'autre cas, le simulateur HDL est en temps réel, le contrôleur électronique décrit en HDL est implanté sur une carte cible (des cartes FPGA dans la plupart des cas) ce qui réduit le temps de simulation. Un simulateur temps réel permet de modéliser et de reproduire la dynamique et le comportement du système de sorte qu'il peut dialoguer, en temps réel, avec le contrôleur. Ce dialogue est fait à l'aide d'interfaces entrées/sorties. Selon la complexité du système à simuler et sa dynamique, plusieurs processeurs peuvent être utilisés pour garantir une modélisation temps réel acceptable. Par exemple, (Harakawa, 2005) a mis en œuvre un simulateur temps réel ayant trois processeurs pour simuler un moteur synchrone à aimants permanents et son alimentation, avec un pas de simulation égal à $10\mu s$.

II.1.3. Simulation par carte de prototypage en boucle

Une nouvelle méthodologie de prototypage dite simulation par carte de prototypage en boucle ("FPGA in the loop") est utilisé récemment (Karimi S., 2009). Un composant FPGA est un circuit intégré numérique composé d'un grand nombre de blocs logiques programmables et reconfigurables sans modification matérielle significative. Les composants FPGAs sont devenus nécessaires dans les systèmes numériques et sont utilisés dans de multiples domaines d'applications en raison de nombreux avantages obtenus lors de leur utilisation (Detrey J., 2007). Parmi tous ces avantages, on peut citer :

1- L'augmentation croissante du niveau de performance temps réel tout en réduisant le coût et l'encombrement.

2- L'utilisation des FPGAs permet l'amélioration des performances en réduisant le temps d'exécution d'un algorithme afin de permettre au contrôleur d'atteindre le niveau de performance des contrôleurs analogiques, sans présenter les inconvénients de ces derniers.

3- Leur grande souplesse de programmation permet de les réutiliser.

4- La rapidité et la facilité de reconfigurer un FPGA autant de fois que nécessaire pour implanter les fonctionnalités désirées.

En raison de tous ces avantages, les FPGAs sont aujourd'hui utilisés dans plusieurs applications nécessitant des traitements numériques importants tels que le traitement du signal et de l'image, le contrôle/commande des machines électriques, la mesure de vitesse, le contrôle des convertisseurs statiques de puissance, l'aéronautique, la télécommunication, les équipements médicaux, les transports, la bio-informatique, l'automobile, la robotique.

II.1.4. Synthèse

La simulation HIL dans Matlab/Simulink peut être considérée comme la technique la plus adaptée pour la simulation des contrôleurs numériques. Ce type de simulation offre plusieurs avantages, citant :

- Le contrôleur ou le régulateur sera vérifié aux premiers stades de développement.
- La fiabilité du contrôleur numérique (bruit, température,…etc) sera testée rapidement.
- Les erreurs et les failles seront détectées très tôt.

Malgré ces différents avantages et la grande utilisation de la simulation HIL, plusieurs limitations et faiblesses entour cette méthode :

- Peu de bibliothèques qui modélisent les architectures cibles sont supportées. Le contrôleur numérique obéira à l'architecture cible développée par Matlab/Simulink. En effet, le DSP builder d'Altera, par exemple, ne supporte pas toutes les bibliothèques de MegaCore (DSP, 2013).
- Limitation des bibliothèques qui modélisent le contrôleur numérique. Chaque contrôleur numérique sera intégré dans l'architecture cible.
- L'architecture cible décrite en Matlab/Simulink est peu modifiable. En effet, les bibliothèques représentant les architectures cibles offre peu de flexibilité pour le changement d'architecture.
- Pour chaque modification du contrôleur numérique, toute l'architecture doit être re-implémentée sur la carte FPGA.

La section suivante présentera une nouvelle technique de vérification des contrôleurs numériques basée sur la notion de la simulation HIL et une carte FPGA Altera pour surmonter les problèmes et les limitations citées.

II.2. La Simulation matériel/logiciel en boucle

Afin de surmonter les limitations citées ci dessus, nous présentons une nouvelle méthodologie de la simulation appelée simulation matériel/logiciel en boucle ("Hardware Software In the Loop") (HSIL).

La méthodologie sera présentée en citant tout d'abord le principe de la simulation, puis les logiciels mis en œuvre, ensuite la couche de communication et finalement la couche de synchronisation.

II.2.1. Principe

Vu la complexité croissante des contrôleurs numériques, la conception matériel / logiciel devient de plus en plus exigée. Notre méthodologie propose une conception basée sur la stratégie de la Co-design pour la modélisation des contrôleurs numériques. La figure 24 décrit l'architecture de la simulation HSIL.

L'architecture proposée est composée essentiellement de deux couches indispensables : la couche communication et la couche synchronisation. Matlab/Simulink offre une grande flexibilité pour une conception hétérogène. Les blocs S-Fonction permettent l'utilisation de plusieurs langages (Matlab, C, C++, Ada) dans une même description. De l'autre coté une architecture basée sur une architecture cible est implanté sur une carte FPGA Altera.

L'idée consiste à considérer le contrôleur numérique comme étant une application logicielle qui s'exécute sur l'architecture cible.

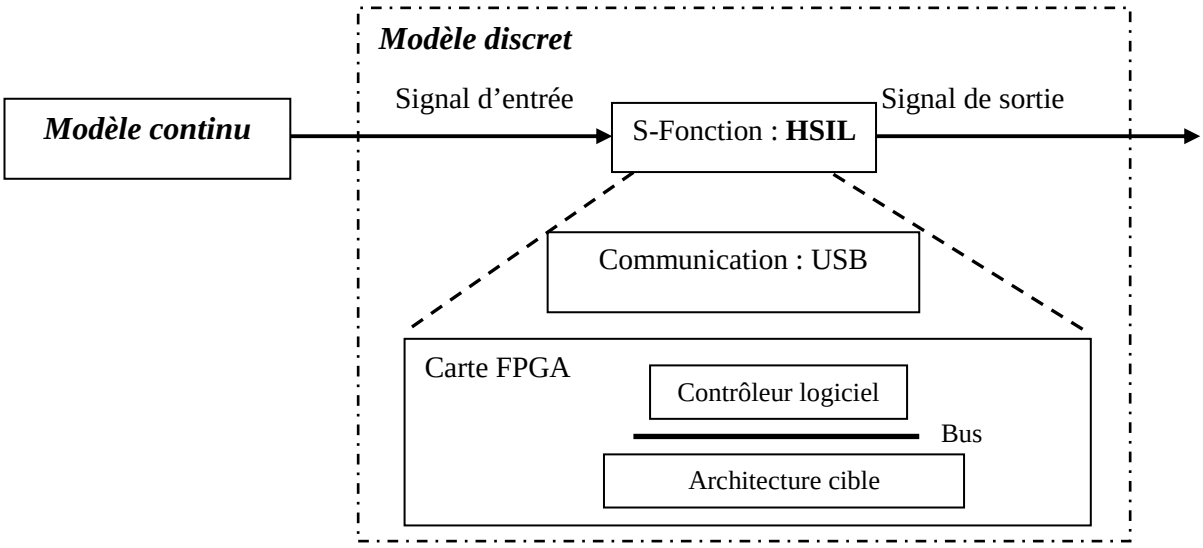


Figure 24. Architecture de la simulation matériel/logiciel en boucle

La simulation HSIL, une extension de la simulation HIL, présente plusieurs avantages citons :

- La conception des contrôleurs numériques complexes obéit à la stratégie de la Co-design. Par la suite, la modélisation devient de plus en plus facile, flexible et le temps de la mise en marché diminue.

- Un seul et même bloc S-Fonction permet le remplacement de tous les contrôleurs numériques. En fait, le contrôleur sera décrit en langage C indépendamment des blocs existants en Matlab/Simulink. Ce fait assure la portabilité et la réutilisation des contrôleurs.

- La vérification et la correction d'un ou de plusieurs contrôleurs numériques se fait sans modifier à chaque fois l'architecture cible.

Les sections suivantes décrivent l'architecture de la simulation HSIL et les logiciels mis en œuvre.

II.2.2. Logiciels mis en œuvre

Dans cette section, nous présentons les logiciels utilisés pour mettre en place la simulation HSIL.

Simulink

Simulink, très populaire pour la communauté de modélisation et de simulation, est un environnement qui s'intègre dans Matlab. Cet environnement possède une vaste gamme d'outils et de bibliothèques permettant de modéliser, simuler et analyser un grand nombre de systèmes dynamiques réels (linéaires ou non linéaires) citons comme exemples : les systèmes électriques, mécaniques, thermodynamiques, électronique de puissance etc. Simulink possède plusieurs bibliothèques dans les domaines de l'automobile, de l'électronique de puissance, du contrôle, etc. et des algorithmes de résolution d'équations différentielles, conçus pour les systèmes, fournissent un bon rapport vitesse/précision de simulation.

L'environnement Simulink possède une interface graphique interactive, particulièrement conviviale, permet à l'utilisateur de construire facilement et rapidement des modèles à travers des blocs fonctionnels existants dans sa bibliothèque, citons par exemples : sources, oscilloscope, intégrateur, additionneur, des composants plus complexes linéaires et non linéaires, etc. Simulink offre la possibilité de créer des blocs non standard grâce aux blocs personnalisables comme les S-fonctions (*S-function : system function)* qui consistent à programmer les équations du système à simuler en utilisant des langages étrangers (C, C++, Ada).

L'algorithme de résolution divise le temps de simulation en un ensemble de pas d'intégration mineurs et pas d'intégration majeurs où le pas mineur représente une subdivision du pas majeur. Le simulateur produit un résultat à chaque pas d'intégration majeur. Ce résultat

utilise ceux de résolution calculés à chaque pas d'intégration mineur afin d'améliorer la précision.

Simulink utilise la règle de dépendance de donnée pour fixer l'ordre d'exécution des blocs durant la phase d'initialisation. Un bloc appelé *direct-feedthrough* est celui dont ces sorties sont en fonction de ses entrées alors il doit être exécuté après ceux qui calculent ses entrées (exemple: additionneur, gain). Tous les autres blocs sont appelés *nondirect-feedthrough* (exemple: intégrateur). Pour assurer l'ordre d'exécution, Simulink commence à exécuter les blocs *nondirect-feedthrough*, en premier lieu, dans n'importe quel ordre, puis il exécute les blocs *direct-feedthrough* dans un ordre qui respecte la règle de dépendance déjà citée.

Quartus II

Le logiciel Quartus II est un outil de CAO dédié à la programmation des CPLDs et FPGAs du fabricant Altera. La figure 25 décrit le flot de conception sous Quartus II. Il permet la description d'un projet, sa compilation, sa simulation logique et temporelle, son analyse temporelle et la programmation d'un circuit cible. Quartus II permet la création des systèmes complexes comportant des processeurs, des périphériques, des mémoires, des bus, des arbitres, et des noyaux d'IPs. Il comprend une suite de fonctions de conception au niveau système, permettant d'accéder à la large bibliothèque d'IP d'Altera et un moteur de placement routage intégrant la technologie d'optimisation de la synthèse physique et des solutions de vérification.

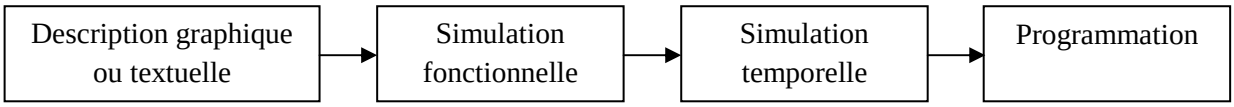


Figure 25. Les différentes parties du flot de conception de QUARTUS II

Quartus II est un logiciel qui travaille sous forme de projets c'est-à-dire il gère un design sous forme d'entités hiérarchiques. Un projet est l'ensemble des fichiers d'un design sous formes graphiques, VHDL ou de bonnes configurations des composants (affectation de pins par exemple).

SOPC Builder

Le SOPC Builder permet, entre autres, de concevoir des microcontrôleurs spécifiques à une application. Ces microcontrôleurs comportent donc une partie processeur à laquelle on associe des périphériques (PIO, Timers, UART, USB, composants propriétaires, …) et de la mémoire. Cette dernière peut-être embarquée dans le FPGA (on parle alors de RAM/ROM On

Chip) ou à l'extérieur du composant FPGA. La partie microprocesseur proprement dite est le NIOS II de ALTERA, processeur de 32 bits qui se décline en trois versions : économique, standard, rapide. La version économique, la moins puissante, utilise le moins de ressources du FPGA. Bien sûr il est possible d'intégrer d'autres types de processeurs pour peu qu'on dispose de leurs modèles (VHDL, Verilog, …). La création d'une application SOPC comprend les étapes suivantes :

- Création du composant matériel (processeur + périphériques) dans l'environnement Quartus.
- Téléchargement dans le composant FPGA (configuration).

SOPC builder peut être divisé en deux parties séparées : une interface utilisateur graphique (GUI) et le programme générateur. Dans l'interface graphique, le concepteur organise tout son système, ajoutant et configurant les composants. Pour le programme générateur, il génère tous les fichiers nécessaires pour la conception.

NIOS II

Le NIOS II est un processeur embarqué à jeu d'instruction réduit (RISC) 32 bits, développé par Altera et conçu pour la mise en œuvre des FPGAs. Cela signifie qu'il s'agit d'un processeur soft-core qui n'est pas produit comme un ASIC. Le NIOS II a des bus séparés pour les données et les instructions (architecture de Harvard), vaste ensemble de possibilités de construire en série des périphériques et des interfaces externes (hors puce) des périphériques.

II.2.3. Couche de communication

La communication représente la première couche de la simulation HSIL. La communication est basée sur la liaison USB qui est déjà décrite dans le chapitre précédent à la section V.1. La communication est assurée par les blocs S-Fonction du Simulink.

Les S-fonctions fournissent un mécanisme puissant pour étendre les capacités de Simulink. Une S-fonction permet de décrire les fonctionnalités du système à l'aide d'un langage de programmation autre que le langage Matlab comme les langages C/C++, Ada, ou Fortran. La commande *mex* permet la compilation de la S-Fonction écrite en langage étrangère pour générer une bibliothèque dynamique qui porte son nom. Une fois la S-Fonction est compilée, le bloc peut interagir avec les autres blocs du système. Les S-fonctions utilisent une syntaxe d'appel particulière qui permet d'interagir avec le moteur de résolution d'équations de Simulink. Cette interaction qui est très semblable à l'interaction entre le moteur

et les autres blocs de Simulink, utilise un cycle d'exécution spéciale. Une S-fonction est composée par un ensemble de fonctions prédéfinies, nous nous sommes intéressés en particulier aux fonctions citées par la figure 26.

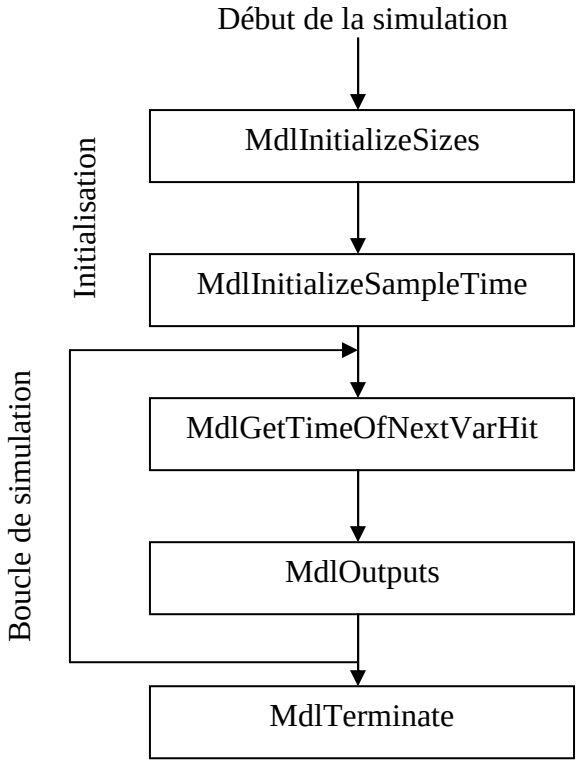


Figure 26. Cycle de simulation d'une S-fonction

Les fonctions MdlInitializeSizes et MdlInitializeSampleTimes sont exécutées durant la phase d'initialisation de Simulink.

La première fonction sert à :

1- Initialiser les largeurs et le nombre de ports d'entrée et de sortie.

2- Fixer le nombre de modes de temps utilisés.

3- Evaluer les paramètres de la S-fonction.

La deuxième fonction est en charge de fixer la nature des modes de temps utilisés par la S-fonction.

Les fonctions MdlGetTimeOfNextVarHit et MdlOutputs sont exécutées à chaque pas d'intégration durant la boucle de simulation (figure 26).

La première fonction sert à fixer le prochain temps d'exécution de la S-fonction. Elle est utilisée seulement si la S-fonction possède le mode de temps VARIABLE_SAMPLE_TIME. La deuxième fonction, MdlOutputs calcule les signaux de sortie de la S-fonction.

Finalement, la fonction MdlTerminate est appelée à libérer la mémoire, à détruire des objets, etc.

Une partie des fonctionnalités de ces fonctions peut être accomplie par l'appel des méthodes existantes dans la bibliothèque *SimStruct* de Simulink. Cette dernière fournit un ensemble assez vaste de méthodes très utiles lors de la programmation, exemple : *ssGetT ()* qui retourne le temps courant, *ssGetOutputPortRealSignal()* qui permet d'accéder aux ports de sortie de la S-fonction. L'utilisateur peut ajouter son code à l'intérieur des fonctions prédéfinies. Lors de la création d'une S-Fonction, un squelette de code utilisant les fonctions prédéfinies ainsi citées peut être modifié en appelant des méthodes par exemple.

II.2.4. Couche de synchronisation

Vue l'hétérogénéité du modèle continu et du modèle discret, des convertisseurs analogique-numérique et numérique-analogique doivent être insérés. La S-Fonction qui supporte la simulation HSIL est composée comme suit (figure 27) :

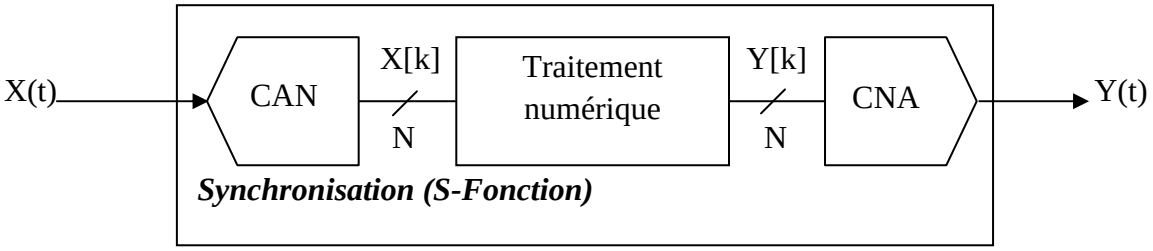


Figure 27. Structure de la S-Fonction synchronisation

Convertisseur Analogique-Numérique (CAN) :

Un convertisseur analogique-numérique permet de convertir un signal analogique vers un signal discret. Cette conversion est basée sur deux phases nécessaires : échantillonnage et quantification.

Un signal analogique, X(t) continu en temps et en amplitude est échantillonné à une période d'échantillonnage constante T_{ech} respectant le théorème de Shannon ($F_{ech}/2 > F_{max}$). On obtient alors un signal échantillonné $X_{ech}(k.T_{ech})$ discret en temps et continu en amplitude.

Ce dernier est ensuite quantifié, pour obtenir un signal numérique X[k] discret en temps et en amplitude. On définit le quantum q (figure 28), ou LSB (Least Significant Bit) le bit de poids faible comme étant la dimension de ces plages avec :

$q = LSB = X/2^N$ avec N présente le nombre de bits dont le convertisseur est codé. (14)

$X_k = k.q \quad k \in \{1,\ldots,7\}$

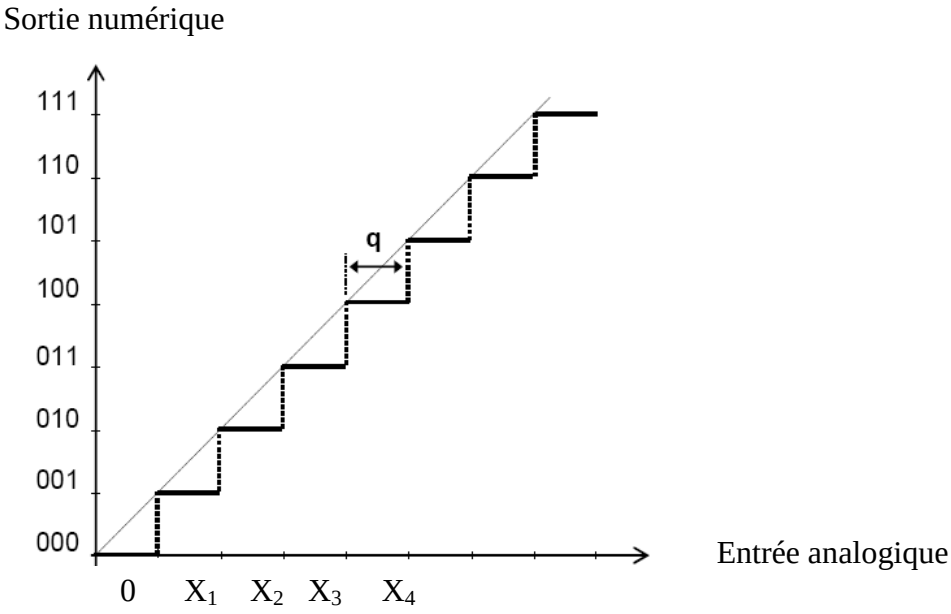


Figure 28. Caractéristique du Convertisseur Analogique-Numérique

Traitement numérique :

Le noyau du traitement numérique se base sur l'envoie de donnée sous forme de paquet respectant le modèle de synchronisation adapté. Le paquet est composé d'un en-tête et un corps comme l'indique la figure 29. L'en-tête informe l'ordonnanceur du contrôleur à exécuter ainsi de la taille des données envoyées. Ici notre module de synchronisation peut être utilisé pour différents contrôleurs numériques en changeant seulement le numéro de routine à exécuter qui correspond à l'algorithme du contrôleur désiré. Quand aux corps, ils contiennent les données envoyées et le temps de synchronisation.

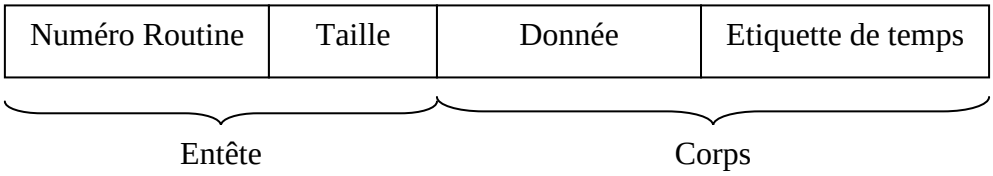


Figure 29. Forme du paquet

Les paquets présentent un point clé d'échange entre le simulateur Simulink et l'architecture cible implanté sur la carte FPGA.

La figure 30 décrit le modèle de synchronisation utilisé dans la simulation HSIL.

Le modèle de synchronisation qui se base sur l'interruption matérielle USB s'exécute en mode « Ping Pong ». La figure 30 montre que lorsque le simulateur continu est en exécution l'émulateur est en repos. Lorsque ce dernier reçoit un paquet de donnée du simulateur, il décode le paquet, change le contexte et exécute le contrôleur désiré.

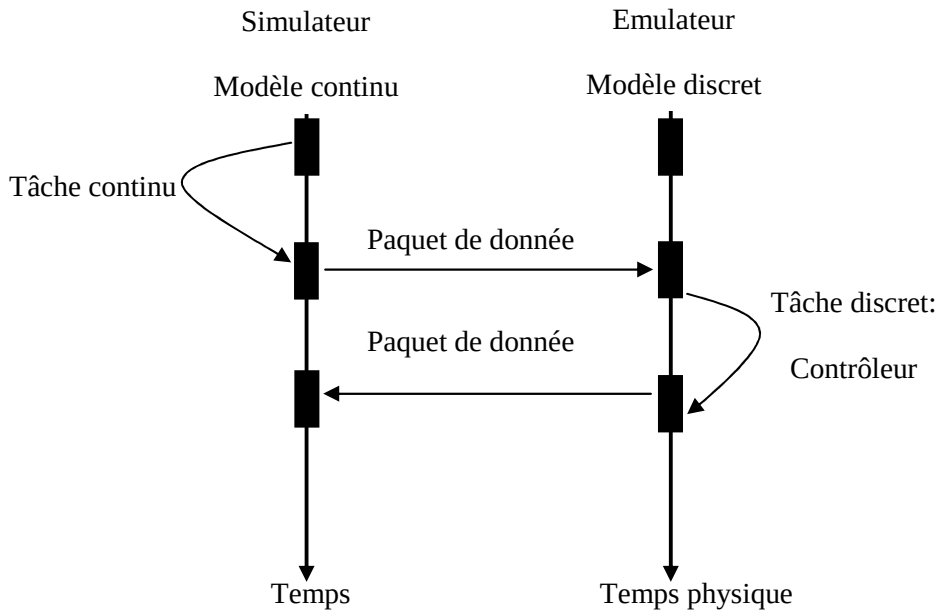


Figure 30. Schéma de synchronisation de la simulation matériel/logiciel en boucle

Convertisseur Numérique-Analogique (CNA) :

Un convertisseur numérique-analogique permet de convertir un signal discret vers un signal analogique comme l'indique la figure 31.

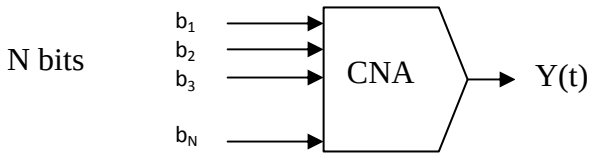


Figure 31. Principe du Convertisseur Numérique-Analogique

Chacun des 2^N mots binaires pouvant être appliqué en entrée est associé à un signal de sortie analogique, telle que:

$Y= (b_1.2^{N-1}+b_2.2^{N-2}+….+b_N.2^0). (X/(2^N-1))$ avec b_1 est le MSB et b_n le LSB. (15)

On définit le LSB, ou quantum, comme étant la plus petite variation possible du signal de sortie correspondant à un changement du bit de poids faible :

$$1\ LSB = X / (2^N - 1). \qquad (16)$$

III. Modèle et environnement de Co-simulation/Emulation des systèmes continu/discret (CODIS+)

Nous présentons dans cette section en premier lieu l'environnement CODIS. Ensuite, une étude détaillée de l'environnement global CODIS+ qui supporte d'une part le simulateur Simulink pour le modèle continu et d'autre part le simulateur SystemC et la carte FPGA pour le modèle discret décrite.

III.1. L'environnement CODIS

Une description selon le modèle de synchronisation utilisé est décrite dans cette partie.

III.1.1. Présentation

L'une des plus grandes difficultés lors de la simulation continue/discrète est la synchronisation du temps entre la simulation à événements discrets et l'intégration numérique du simulateur continu.

La synchronisation est un point clé qui influence la précision et la vitesse de simulation. Il existe deux approches fondamentales de synchronisation : l'approche optimiste et l'approche pessimiste (Langeanu D., 2001).

L'approche optimiste permet à chaque simulateur d'effectuer quelques pas optimistes. Si un simulateur génère un événement avant la fin de ces pas, l'autre simulateur doit être capable de reculer son temps.

Dans le cas de l'approche pessimiste, les simulateurs avancent avec le même pas de temps, ce qui évite tout besoin de recul. À partir du modèle de synchronisation basé sur l'approche pessimiste nous définissons un modèle de synchronisation supportant à la fois le simulateur continu et le simulateur/émulateur discret présenté dans le chapitre 2.

L'environnement COntinuous DIscrete Simulation (CODIS) (Bouchhima F., 2007) présente un environnement de modélisation et simulation des systèmes continus/discrets. Une présentation de cet environnement est décrite en se basant sur le principe de simulation et les modèles de synchronisation utilisés.

III.1.2. Principe de l'environnement CODIS :

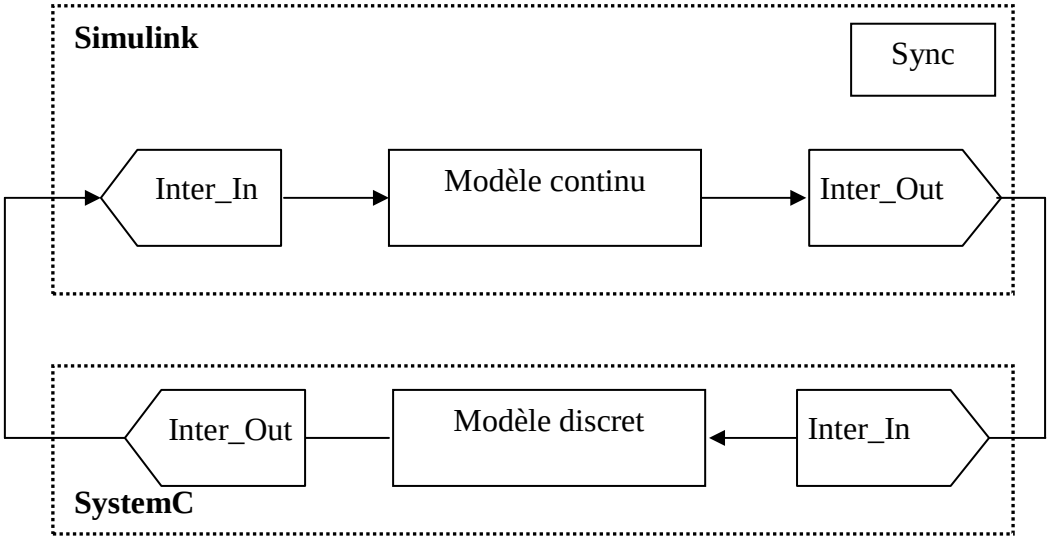


Figure 32. Schéma global de l'environnement CODIS

La figure 32 montre le schéma global qui relie le modèle continu et le modèle discret ainsi les interfaces utilisées dans l'environnement CODIS.

Pour **Simulink** les interfaces peuvent être paramétrées à partir de leurs boites de dialogue. CODIS possède principalement trois types d'interfaces dans Simulink :

- L'interface *Inter_In* : c'est une interface qui permet de lire les données reçues de la part du modèle discret. Cette interface implémente la couche de communication et permet de détecter les événements discrets pour effectuer le changement de contexte. Cette dernière étape est responsable de détecter le passage du temps de simulateur par les étiquettes de temps des événements d'échantillonnage. Cette interface a comme paramètre :
    1. Le nom, le nombre et le type des données des ports d'entrée du modèle discret.
    2. Les périodes d'échantillonnage.
    3. Le mode utilisé.
- L'interface *Inter_Out* : c'est une interface qui permet d'envoyer les données vers le modèle discret et de lancer le changement du contexte. Cette interface implémente la même couche que l'interface *Inter_In* et a comme paramètre :
    1. Le nom, le nombre et le type de données des ports de sorties du modèle discret.
    2. Le mode utilisé.
- L'interface *Sync* : Cette interface implémente la partie la plus importante de la phase de la détection des événements discrets. Elle crée les points d'arrêt que l'algorithme de résolution doit atteindre sans dépassement. Ces points sont les étiquettes de temps des

événements reçus. Quand un événement est reçu, l'interface fixe son prochain temps d'activation égal à l'étiquette de temps de cet événement, ceci grâce au mode de temps de la S-fonction de l'interface *Sync*. Une fois ce temps est atteint, l'interface *Inter_In* ou *Inter_Out* est exécutée pour se synchroniser avec l'événement et pour changer le contexte vers le simulateur discret. Une fois que Simulink reprend l'exécution, l'interface *Sync* est exécutée pour fixer son prochain temps d'exécution égal à l'étiquette de temps du nouvel événement reçu. L'interface est exécutée au temps égal zéro pour fixer son premier prochain temps.

Ces interfaces sont manipulées comme n'importe quel bloc de la bibliothèque de Simulink. Leurs ports d'entrée ou de sortie sont compatibles avec les ports du modèle continu et peuvent être connectés directement en utilisant les signaux de Simulink. L'utilisateur doit placer les interfaces à partir de la bibliothèque des interfaces dans la fenêtre du modèle continu, puis il fixe leurs paramètres et finalement il les connecte avec les ports d'entrée et de sortie du modèle continu. Durant la phase d'initialisation de la simulation, Simulink charge les fonctionnalités de ces interfaces à partir de la bibliothèque dynamique (.dll). Les interfaces sont générées automatiquement par un outil de génération de code qui a comme entrée les paramètres définis par l'utilisateur.

Pour **SystemC** les interfaces peuvent être appelées à partir de la bibliothèque de simulation. CODIS possède principalement deux types d'interfaces :

- L'interface *Inter_In* : c'est une interface qui permet de lire les données reçues de la part du modèle continu. Cette interface implémente la couche de communication et assure l'échange de données, la conversion des signaux et le changement de contexte en envoyant des étiquettes du temps des événements d'échantillonnage. Elle assure aussi la synchronisation avec les données échantillonnées à l'entrée du modèle discret grâce aux horloges d'échantillonnage.
- L'interface *Inter_Out* : c'est une interface qui permet d'envoyer les données vers le modèle continu et lancer le changement du contexte au niveau du noyau de SystemC.

Pour SystemC, l'outil génère aussi la fonction 'sc_main' (ou la modifie si elle existe déjà) qui connecte les interfaces avec le modèle discret. Le modèle est compilé et l'éditeur de lien appelle la bibliothèque de SystemC et la bibliothèque statique, appelé bibliothèque de simulation.

III.1.3. Modèle de synchronisation de l'environnement CODIS :

Une fois les interfaces générées et connectées aux modèles, le concepteur simule son système en exploitant les outils de débogage des deux simulateurs intégrés par l'environnement. Dans cette section, une description du modèle de synchronisation pessimiste est présentée.

La figure 33 présente le modèle de synchronisation. Ce modèle est utilisé lorsque le modèle continu est en avance par rapport au modèle discret et respecte l'algorithme canonique (Ghasemi H.R., 2005). Dans cet algorithme, le simulateur continu prend en considération l'occurrence d'un événement discret à son tour le simulateur discret tient en compte les événements d'état envoyés par le simulateur continu.

Dans la figure 33, le simulateur continu et le simulateur discret sont synchronisés à l'instant A. Ce dernier simulateur commence à exécuter tous les processus qui sont sensibles aux événements déclenchés à l'instant courant A et met à jour les signaux sans avancer le temps, ce qui constitue un cycle de simulation. Il faut souligner que le noyau de SystemC modifié permet l'exécution des processus sans avancer le temps sauf lorsqu'un changement de contexte du simulateur continu vers le simulateur discret est effectué. Ensuite, le simulateur discret envoie au simulateur continu le temps d'occurrence de son prochain événement de sortie (point B: prochain événement), il change le contexte de simulation vers le simulateur continu (flèche 1). Ce dernier calcule les signaux en résolvant les équations différentielles du modèle jusqu'à atteindre avec précision le temps envoyé par le simulateur discret (point C : temps d'événement discret atteint). Deux cas se présentent:

- Le temps du point C représente le temps d'occurrence d'un événement d'échantillonnage. Dans ce cas, le simulateur continu met à jour les signaux de sortie avec leurs valeurs calculées à cet instant et change le contexte vers le simulateur discret (flèche 3). Ce dernier avance pour le temps d'occurrence de l'événement d'échantillonnage (flèche 4) et commence un nouveau cycle de simulation.

- Le temps du point C est le temps d'occurrence d'un événement de mise à jour des signaux. Dans ce cas, le simulateur continu change le contexte vers le simulateur discret qui avancera pour le temps d'occurrence de l'événement indiqué, calcule les signaux et envoie leurs valeurs et le temps d'occurrence du prochain événement. Finalement, il change le contexte vers le simulateur continu qui va lire les nouvelles

valeurs des signaux et procède pour le prochain temps d'événement discret et le cycle recommence (flèche 5 et 6).

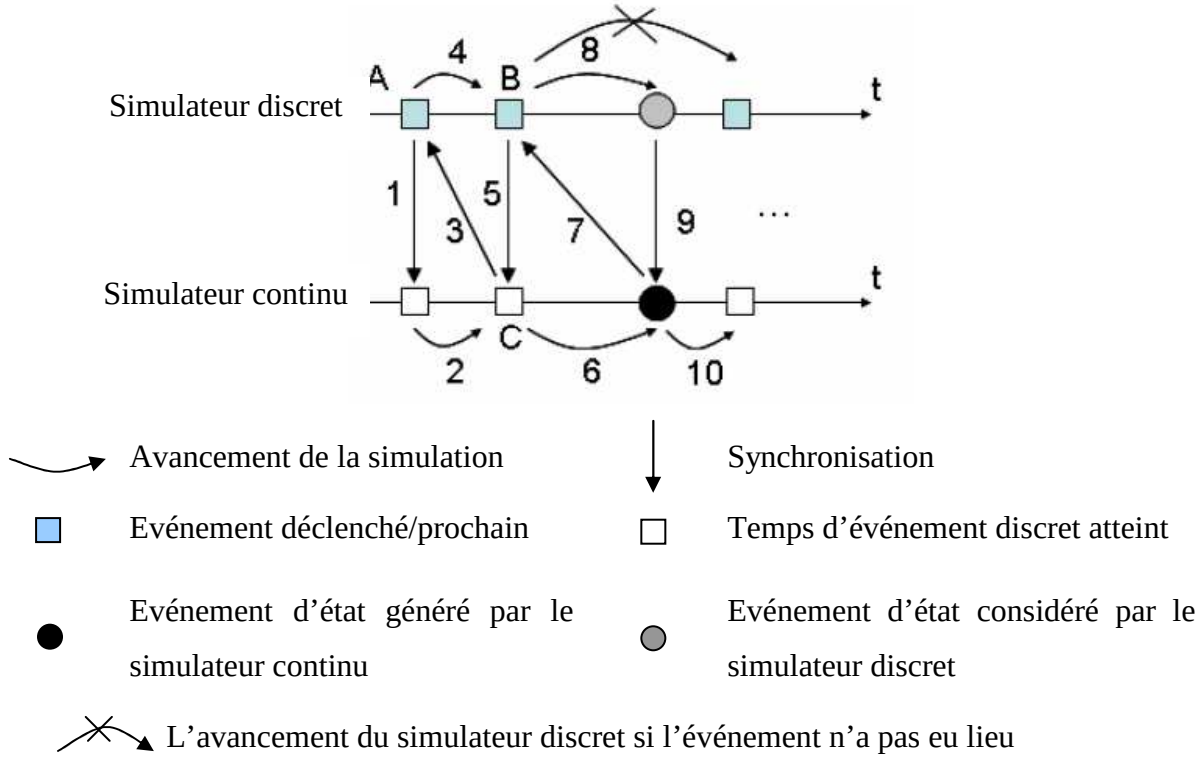


Figure 33. Le modèle de synchronisation pessimiste

Le modèle continu peut générer un événement d'état. Dans ce cas, le simulateur continu indique sa présence, envoie son temps d'occurrence au simulateur discret et change le contexte de simulation (flèche 7). Le simulateur discret doit considérer cet événement en avançant le temps vers son temps d'occurrence et d'exécuter les processus qui lui sont sensibles.

III.2. Discussion

Cet environnement a été validé par plusieurs applications hétérogènes. CODIS est un environnement de vérification des systèmes hétérogènes basé sur une co-simulation entre le simulateur Simulink et le simulateur SystemC. L'environnement CODIS présente plusieurs avantages citant :

- Utilisation des langages Matlab/Simulink pour le modèle continu et SystemC pour le modèle discret. Ces deux langages sont classés comme étant les premiers langages a utilisés dans les premiers phases de conception afin de vérifier le fonctionnement du système à réaliser. Ce point permet d'utiliser les bibliothèques existantes conçues pour les modèles continus et discrets.

- CODIS utilise le SystemC pour modéliser les parties matériels au lieu de VHDL ou Verilog. Ceci permet d'accélérer à la fois le temps de la mise en marché et le temps de simulation ainsi plusieurs niveaux d'abstraction peuvent être utilisés.

Comme les autres environnements CODIS possède quelques limitations, citons :

- Les temps de simulation et plus important par rapport aux nouveaux environnements qui supportent à la fois les modèles continu/discret.
- L'environnement CODIS ne supporte pas la stratégie du Co-design lors de la modélisation des contrôleurs numériques. Ce qui rend CODIS non efficace pour les contrôleurs complexes.

La méthodologie qui sera présentée par la suite à pour but d'adapter CODIS à la stratégie Co-design et d'accélérer le temps de simulation en utilisant une carte FPGA dont l'architecture cible est implantée.

III.3. Modèle de synchronisation de l'environnement CODIS+

Nous présentons dans cette section le modèle de synchronisation global proposé (figure 34). Ce modèle se base d'une part sur le modèle de synchronisation pessimiste entre le simulateur Simulink et le simulateur SystemC et d'autre part entre le simulateur SystemC et une architecture cible implantée sur une carte FPGA pour la modélisation conjointe faite dans le chapitre 2. Le temps de synchronisation représente le point clé à décrire et on va ignorer les types de paquets échangés entre les différents simulateurs et émulateurs.

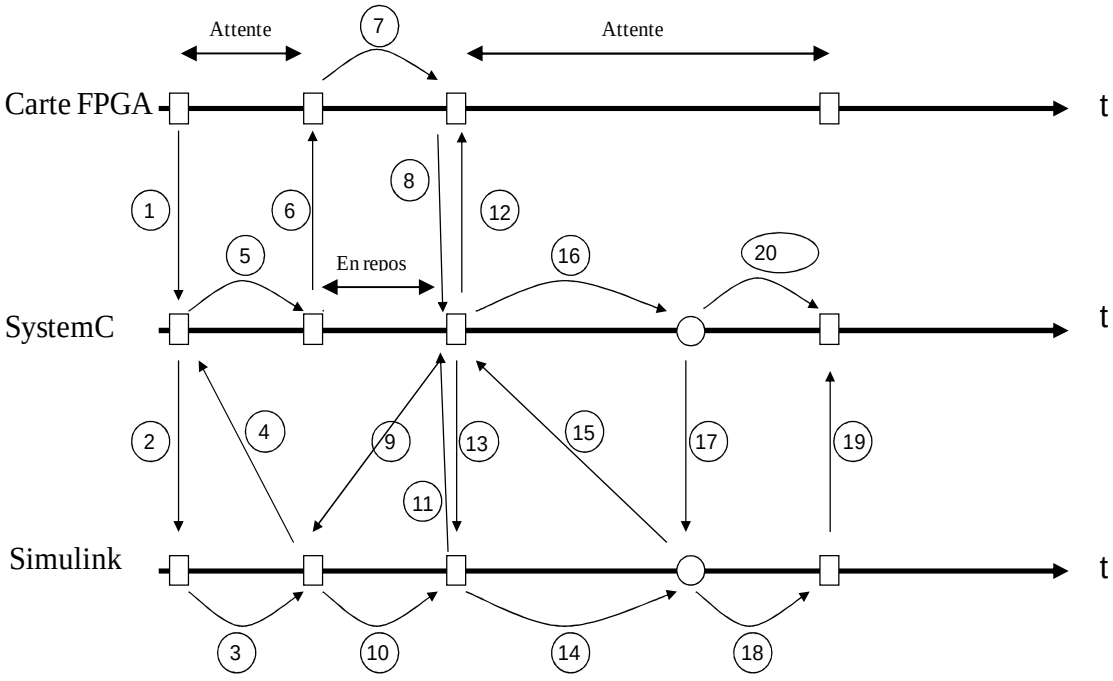


Figure 34. Modèle de synchronisation de l'environnement CODIS+

A l'instant t=0s, le processeur cible implanté sur la carte FPGA déclenche la co-simulation émulation ainsi un changement de contexte vers le simulateur SystemC est effectué (flèche 1). Ce dernier commence à exécuter les modules sensibles sans avancer le temps de simulation. Une fois l'exécution est finie, le simulateur discret envoie le prochain temps éventuel de synchronisation vers le simulateur continu. Un changement de contexte est accompli par la suite (flèche 2). Lorsque le Simulink reçoit le prochain temps de synchronisation, il commence à exécuter le modèle continu. Deux cas peuvent se présenter :
1- le temps étiquette est atteint avant la génération d'un pas d'intégration.

2- Un pas d'intégration est généré avant l'atteinte du temps étiquette.

On s'intéresse à ce stade là au premier cas. Lorsque le temps est atteint (flèche 3), le simulateur Simulink fait un changement de contexte vers le simulateur SystemC (flèche 4). A cet instant SystemC avance son temps (flèche 5) d'où un cycle de simulation est terminé. Puis, une tâche logicielle est simulée par la carte (flèche 7) et le simulateur SystemC entre dans une phase de repos. Lorsque l'émulateur termine sa tâche, un changement de contexte vers le simulateur Simulink est fait à travers le SystemC (flèche 8 et flèche 9) en envoyant le temps étiquette qui correspond au temps d'exécution de la tâche logicielle. Dans ce cas Simulink continue l'exécution du modèle continu jusqu'à atteindre le nouveau temps étiquette envoyé (flèche 10) et se bloque pour donner l'accès au SystemC. Ensuite, le SystemC exécute une tâche matérielle sans avancer son temps et envoie le prochain temps de synchronisation vers le simulateur continu. Lorsque le changement de contexte est accompli le modèle continu commence à s'exécuter jusqu'à l'apparition d'un événement d'intégration, et avant que le temps d'étiquette soit atteint, c'est le deuxième cas. Dans ce cas, Simulink envoie le temps d'apparition de l'événement d'intégration (temps étiquette) vers le SystemC pour que ce dernier avance son temps de simulation jusqu'au temps d'étiquette ainsi un changement de contexte est fait. Ce cas représente un cas critique car il perturbe le parallélisme de l'exécution des simulateurs. Le mécanisme décrit permet de conserver le bon fonctionnement des simulateurs ainsi les échanges des instants. Le Simulink poursuit l'exécution jusqu'au nouvel instant (flèche 18). Ce cas n'a pas d'influence sur l'émulateur car il est en phase d'attente.

Le simulateur SystemC représente le maître de l'environnement de co-simulation / émulation vue le mécanisme du noyau de la simulation (initialisation puis exécution) et vue la possibilité de la modification du noyau. L'émulation (déjà décrite dans le chapitre 2) est assurée à travers des fonctions qui seront intégrées dans le code de haut niveau et ne nécessite

pas la modification du noyau de la simulation de SystemC grâce à l'interruption matérielle faite par la liaison USB.

Les interfaces utilisées pour Simulink sont les mêmes interfaces faites par l'environnement CODIS. La seule modification est réalisée au niveau du type de paquet échangé.

Pour SystemC deux nouvelles interfaces sont ajoutées :

- *Interface_In* : Cette interface fait appelle aux fonctions de la bibliothèque de la simulation/émulation. Elle implémente la couche de communication et permet de lire les paquets envoyés de la part du processeur cible.
- *Interface_Out* : Cette interface fait appelle aux fonctions de la bibliothèque de la simulation/émulation. Elle implémente la couche de communication et permet d'envoyer les paquets vers le processeur cible.

IV. Conclusion

Dans ce chapitre, nous avons détaillé les différentes méthodologies adaptées. En premier lieu, une présentation de la simulation matériel/logiciel en boucle est annoncée pour le cas de la simulation des contrôleurs numériques. Cette technique est une extension de la fameuse technique HIL utilisée dans Matlab/Simulink. Notre technique respecte la conception Co-design non utilisée dans la conception des contrôleurs numériques. La simulation matériel/logiciel en boucle représente une technique efficace pour les contrôleurs complexes et diminue le temps de la mise en marché.

En deuxième lieu, une extension de l'environnement CODIS est proposée afin d'accélérer le temps de simulation et de supporter des systèmes plus complexes. En fait, l'environnement CODIS souffre essentiellement du temps de simulation important lors de l'utilisation de l'ISS et n'obéit pas à la stratégie de conception Co-design pour les systèmes numériques. Notre environnement résolu ces deux problèmes en utilisant une carte de prototypage.

Dans le chapitre suivant, une implémentation de l'environnement simulateur/émulateur, de la simulation HSIL et de l'environnement de co-simulation/ émulation est validé à travers plusieurs exemples d'application.

Chapitre 4 : EXPERIMENTATION : APPLICATIONS ET ENVIRONNEMENTS

I. Introduction

Les environnements de co-simulations présentent une nécessité croissante pour la modélisation des systèmes continus/discrets. En fait, ces environnements non seulement facilitent la tâche de la modélisation et accélère la phase de conception mais aussi diminue le coût de fabrication des systèmes. L'environnement de co-simulation/émulation présenté dans le chapitre précédent représente un outil puissant pour la simulation continu/discret vu le schéma de synchronisation qui assure un temps de simulation minime et offre la possibilité de modéliser les systèmes numériques conjointement et à différent niveau d'abstraction.

Dans ce chapitre, nous présentons plusieurs applications pour valider les environnements détaillés précédemment, citons : le système de reconnaissance par empreinte digitale, un système de régulateur de la vitesse d'un moteur à courant continu, un système de contrôle en boucle fermée de la vitesse d'un moteur et un système de contrôle de vitesse d'un véhicule en se basant sur une identification biométrique. Une étude algorithmique d'un système de reconnaissance par empreinte digitale est décrite au cours de la première partie de ce chapitre. L'objectif de ces applications est la validation de l'environnement de simulation/émulation matériels/logiciels, de la technique de simulation matériel/logiciel en boucle et l'environnement CODIS+ de co-simulation/émulation des systèmes continus/discrets.

II. Implémentation de l'architecture cible sur FPGA

La communication étudiée dans les chapitres précédents était basée sur une architecture cible. On fixera dans cette section que l'architecture cible est composé du processeur NIOS II, du bus Avalon et des mémoires.

On ajoute à cette architecture le contrôleur USB ISP1362 de la carte DE2-70 (famille Altera) qui représente l'arbitre de la communication et n'appartient pas à l'architecture cible. A l'aide de l'outil SOPC Builder l'architecture cible est implémenté (figure 35).

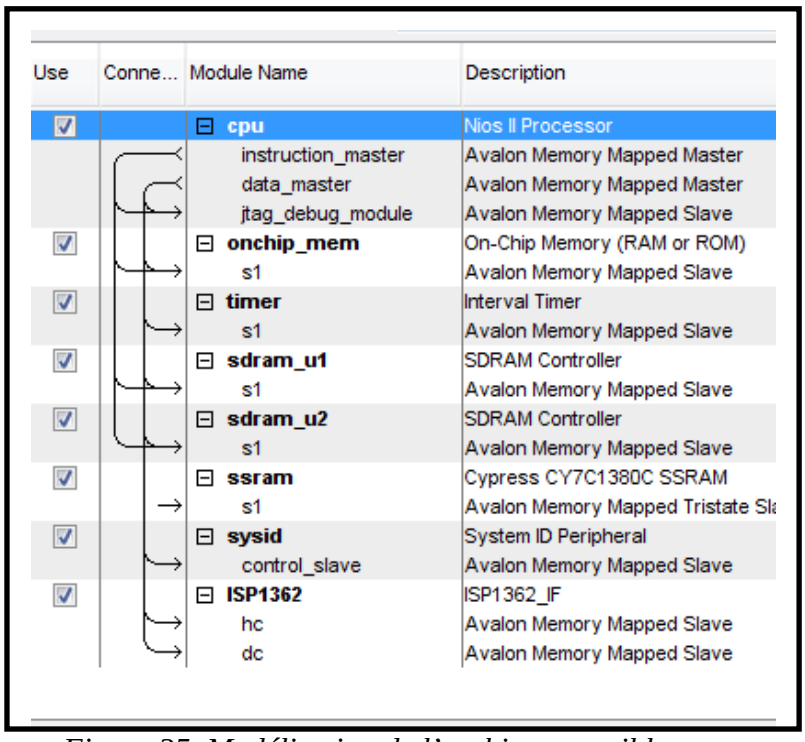


Figure 35. Modélisation de l'architecture cible

III. Expérimentation de l'environnement de Simulation/Emulation matériel/logiciel

Dans cette section nous présentons dans la première partie une étude algorithmique du système de reconnaissance par empreinte digitale. L'implémentation de l'application et les résultats de la simulation/émulation sont exploitées dans la deuxième partie.

III.1. Application : Système de reconnaissance par empreinte digitale

La reconnaissance par empreinte digitale est le système biométrique le plus répandu dans le monde sécuritaire. Il est indéniable qu'un tel système de reconnaissance soit le meilleur non seulement grâce à son faible coût par rapport à d'autre modalité mais aussi grâce à sa sureté. La figure 36 montre les principales phases de reconnaissance par empreinte digitale.

III.1.1. Phase de prétraitement

La phase de prétraitement présente une phase essentielle pour l'amélioration de l'image de l'empreinte. Cette phase est constituée de l'étape de filtrage, de la binarisation et de la squelettisation.

III.1.1.1. Filtrage

Toute image de basse qualité provoque de gros problèmes dans le domaine de traitement d'images. Dans ce cadre la plupart des images d'une empreinte digitales demandent un filtrage afin d'extraire ses informations utiles.

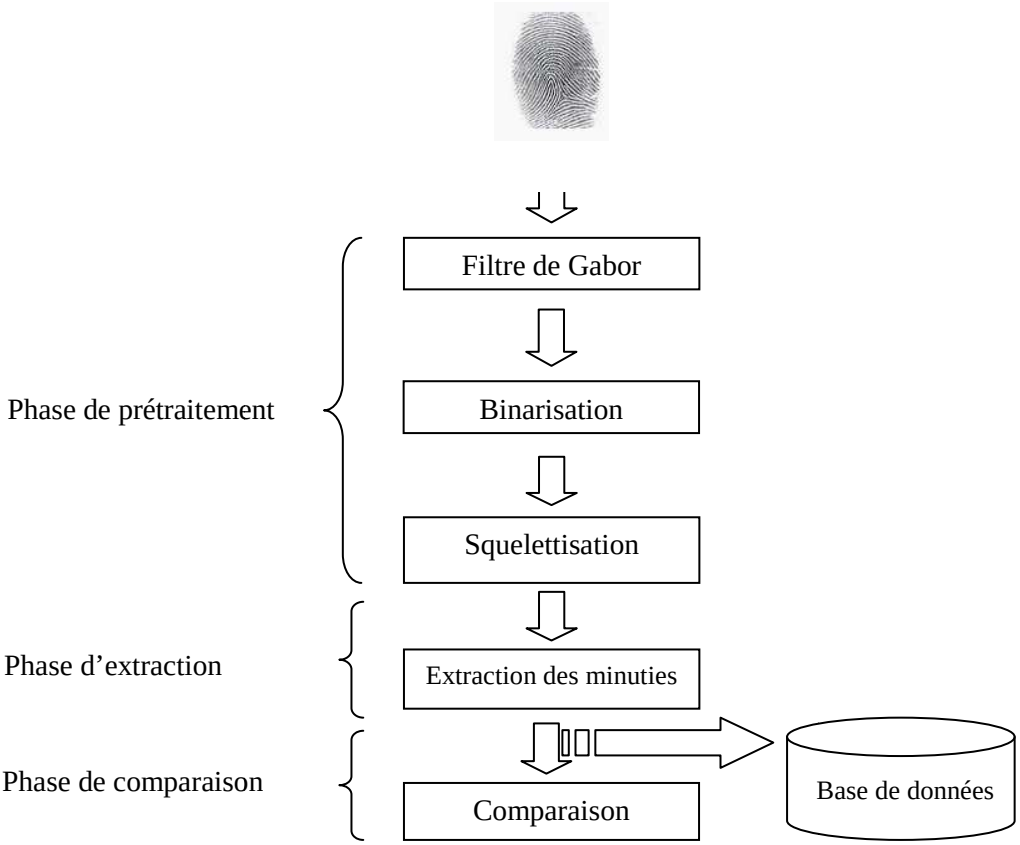


Figure 36. Chaine de reconnaissance

Le filtre de base que nous avons utilisé est un filtre de Gabor à symétrie paire et orienté à 0 degré.

$$h(x,y,\varphi,f) = e^{-\frac{1}{2}\left(\frac{x_\varphi^2}{\delta_x^2}+\frac{y_\varphi^2}{\delta_y^2}\right)}.\cos(2\pi f x_\varphi) \qquad (16)$$

Pour obtenir les autres orientations, il suffit d'effectuer une rotation des axes coordonnés:

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} \cos(\theta_i) & \sin(\theta_i) \\ -\sin(\theta_i & \cos(\theta_i) \end{bmatrix} . \begin{bmatrix} x \\ y \end{bmatrix} \qquad (17)$$

Selon les différents blocs de l'image, le filtre peut avoir plusieurs directions privilégiées. Dans ce cas-là, le filtre final est une somme de filtres de base placée à chaque direction.

$$h = \sum_i h_b(x_i, y_i) \qquad (18)$$

$$E(i,j) = \sum_{v=\frac{wx}{2}}^{u=\frac{wx}{2}} \sum_{v=\frac{wy}{2}}^{u=\frac{wy}{2}} h[u,v,o(i,j),F(i,j)].N(i-u,j-v) \qquad (19)$$

Où -E (i, j): nouvelle valeur du pixel (i, j).

-F (i, j): la fréquence du pixel (i, j).

-O (i, j): direction du pixel (i, j).

Sélection des paramètres de Gabor

Pour l'extraction de la réponse des crêtes et des vallées de diverses orientations du filtre de Gabor, les paramètres f et θ sont fixés aux valeurs suivantes:

- La fréquence f correspond à la distance inter-crêtes dans l'image de l'empreinte digitale. Après plusieurs essais, on l'a fixée à 0,3.
- Les orientations examinées correspondantes aux valeurs de θ sont: 30°, 45°, 60° et 90°. Pour les images présentées, on a fixé θ à 60°.
- Les paramètres écarts types σ_x et σ_y contrôlent la bande passante du filtre, ils doivent être convenablement choisis, vu leurs effets significatifs sur l'amélioration des résultats.
- La valeur de σ_x détermine le degré d'amélioration de contraste entre les rides et les vallées alors que σ_y détermine le degré de lissage appliqué aux rides tout au long d'une orientation locale.

III.1.2. Binarisation

La binarisation de l'image est le processus qui transforme une image en niveau de gris en une autre noir et blanc. Dans une image en niveau de gris, un pixel peut prendre 256 valeurs d'intensité différentes tandis qu'un pixel dans une image noire et blanche ne peut être aussi que noir ou blanc. Cette transformation est faite en appliquant un seuillage à l'image. La valeur 1 signifie que le pixel est blanc alors que la valeur zéro indique que le pixel est noir. L'échelle de gris est formée par des valeurs décimales entre 0 et 1. Lorsque le seuil est appliqué à l'image, tous les pixels sont comparés à la valeur du seuil qui est calculée à travers les seuils des couleurs RGB. N'importe quelle valeur de pixel inférieure au seuil prend zéro, et n'importe quelle valeur de pixel supérieure au seuil prend 1. À la fin de ce processus, toutes les valeurs des pixels sont soit zéro soit un. Ainsi, l'image sera transformée en format binaire avec la valeur 0 pour les crêtes et la valeur 1 pour les vallées. Après cette opération, les crêtes

dans l'empreinte digitale sont accentuées avec la couleur noire tandis que les vallées sont blanches.

Dans le processus de binarisation, le choix de la valeur seuil calculé par l'équation suivante est critique.

$$Seuil = \frac{\sum M(i, j)}{nl * nc} \qquad (20)$$

Avec M(i,j) la matrice de l'image, nl:nombre de ligne, nc: nombre de colonnes

Il existe deux approches pour le calcul de seuil :

- Seuil Global : Le principe est de calculer la valeur moyenne de toute l'image. L'avantage est que cette solution est très rapide ; alors qu'elle cause des problèmes si l'image présente une hétérogénéité au niveau luminance.

- Seuil Local : Le principe est calculer la valeur moyenne par masque. Le principal avantage de cette méthode est la bonne qualité issue de la binarisation mais le temps de calcul est très important.

La figure 37 montre le résultat de binarisation par seuil global et local et prouve que la binarisation avec un seuil local est le plus adéquat pour une image d'empreinte.

a) Image capturée b) Seuillage global c) Seuillage local

Figure 37. Les méthodes de Binarisation

III.1.3. Squelettisation

Après la binarisation, un autre processus important doit être appliqué à l'image: Il s'agit de la squelettisation. Ce processus réduit l'épaisseur de toutes les rides à un pixel.

Squelettisation à base de « Neighborhood » :

Cette méthode est basée sur le principe d'élimination qui représente le noyau d'un algorithme d'amincissement. Cet algorithme se base sur les valeurs de poids c'est-à-dire le nombre de pixel noir autour du pixel en question. Nous utilisons ainsi une fenêtre de taille 3 × 3. Tous les types de relation (256) formés par 8 pixels voisins de l'objet ont été examinés. À partir de ces cas, un groupe de règles d'élimination peut être obtenu.

Cet algorithme est un algorithme itératif et s'arrête lorsque le long d'un traitement aucune modification n'est faite.

Le majeur problème lors de la squelettisation se manifeste via l'occurrence d'effet de ZIGZAG sur la strie. Ce problème cause la détection de fausse minutie. Nous proposons comme solution d'amélioration, d'appliquer le filtre de lissage sur l'image squelette. En effet, l'expérience montre une image résultante améliorée. La figure 38 décrit la méthode de squelettisation sans/avec le filtre de lissage.

a) Squelettisation sans filtre de lissage b) Squelettisation avec filtre de lissage

Figure 38. Squelettisation sans/avec filtrage

III.1.2. Phase d'extraction

L'extraction des minuties à partir d'une empreinte squelettisée nécessite une méthode capable de distinguer et de classer les différentes formes et types de minuties. Donc il s'agit d'un problème de classification. Nous proposons d'appliquer une méthode de classification basée sur la distance de Hamming appelée DECOC. Cette méthode développée dans ce chapitre est basée sur le travail de **Jie Zhou** (Jie Z., 2007).

Motivé par les nouvelles solutions de la décomposition multi-classe qui sont des extensions de la méthode ECOC, on propose l'exploitation d'une nouvelle méthode appelée Data-driven ECOC (DECOC) pour résoudre le problème de classification dans le cas de l'empreinte digitale.

Principe de base de la méthode ECOC

Error correcting output codes (ECOC) est utilisé dans les domaines de la communication et de la théorie de l'information dans le but d'améliorer la fiabilité de la transmission de signaux binaires et de maintenir l'intégrité des informations. Son principe est d'ajouter des bits de parité pour chaque redondance de l'information. La distance entre deux mots est définie à l'aide de la distance Hamming, qui représente le nombre de différence de bits dans les deux mots. Enfin, un processus de décodage examine les distances de Hamming entre les binaires reçus et valide l'ensemble des mots pour détecter et récupérer les erreurs.

ECOC est basée sur la plus courte distance de Hamming formulé comme suit :

$$y = \text{argmin}_k H(w_k, w(x)),\ k = 1, \ldots, K \qquad (21)$$

Avec Wk est la k^{ème} ligne de la matrice. H (W_k, W(x)) est la fonction qui permet de calculer la distance de Hamming. Nous attribuons le label de classe codé de la plus proche, c'est-à-dire, avec la plus courte distance de Hamming, à l'échantillon de test.

Principe de la méthode DECOC

Nous proposons Data-driven ECOC (DECOC) pour concevoir le code de la matrice ECOC en utilisant les données représentées par les pixels de l'image. L'idée clé de DECOC est de sélectionner certaines bases binaires dans la matrice selon son score de confiance. Cette mesure nous aidera à déterminer comment nous allons probablement inclure la matrice sous test dans l'ensemble.

Avant de présenter le score de confiance, il faut tout d'abord définir le critère de séparabilité d'un groupe de plusieurs classes, avec G présente le groupe qui contient les classes de même famille.

$$S(G)=\begin{cases} \dfrac{2}{|G|^2-|G|} \displaystyle\sum_{0\, j\neq k, c_j c_k \in G} d(c_j, c_k) & |G|\neq 1 \ and\ |G|\neq K-1 \end{cases} \qquad (22)$$

Avec d(C_j, C_k) est la distance entre deux classes C_j et C_k, qui est la distance de Hamming entre les vecteurs de même classe; | G | est la taille du groupe, c'est-à-dire, le nombre de motifs de même classe ; 2 / (| G |^2 - | G |) est le facteur de normalisation.

La confiance d'une base binaire DECOC est alors définie comme:

$$C(f)=\begin{cases} \dfrac{S(G_{+/-}(f))}{S(G_+(f))+S(G_-(f))}; |G_+|\neq 1 \ and |G_+|\neq K-1 \end{cases} \qquad (23)$$

G+(f) est l'ensemble des classes qui ont une distance Hamming minimale par rapport aux pixels noirs de la classe f, G-(f) est l'ensemble des classes, qui ont une distance Hamming minimale par rapport aux pixels blancs. S (G + / - (f)) est égale à la distance entre la matrice sous test et tous les patterns d'une classe en calculant le nombre de pixels blancs non utilisés.

Le principe du flot d'apprentissage de la figure 39 se base sur le calcul de la distance de Hamming entre le bloc en question et chaque classe. Ensuite, on calcule le S(G+(f)) pour la classe dont la distance de Hamming est minimale et on calcule le S(G+/-(f)). Ainsi on calcule le score confidentiel pour le cas de la terminaison, de la bifurcation et du non minutie. Le score qui possède la valeur maximale présente le type du bloc en question. D'où la décision.

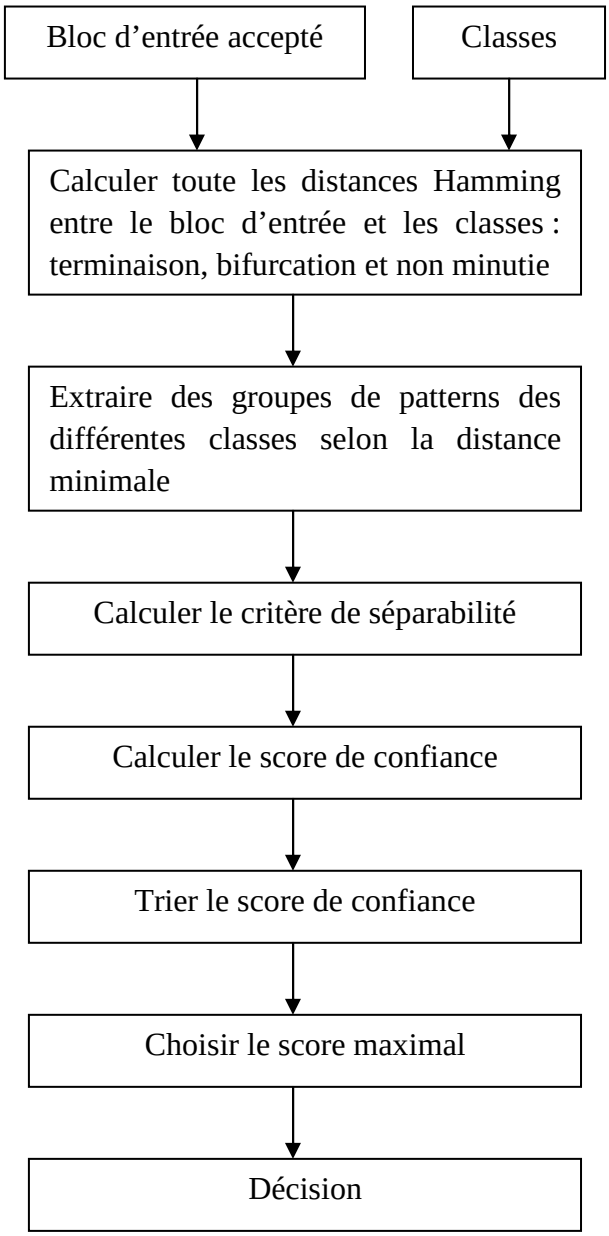
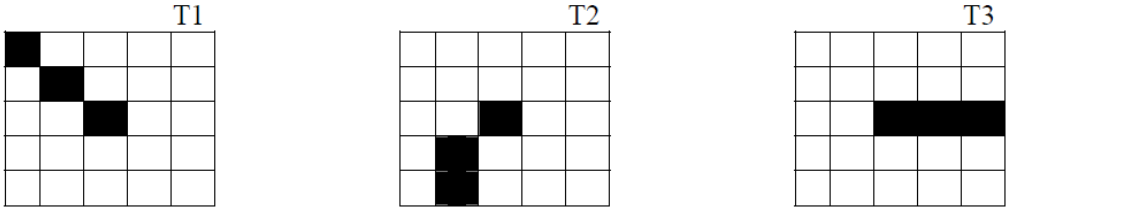


Figure 39. Flot d'apprentissage de l'algorithme DECOC

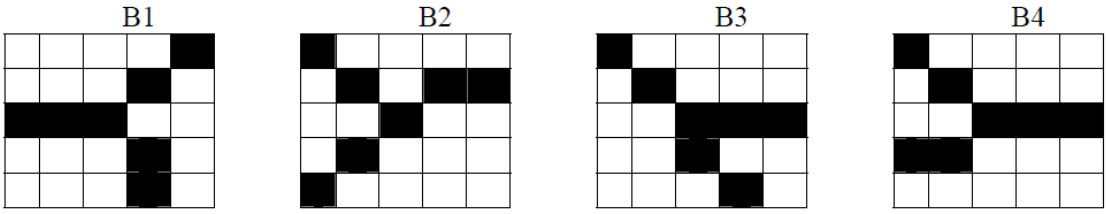
Afin de choisir les bons motifs pour chaque classe, on a utilisé plusieurs empreintes squelettisées différentes. On a trouvé le total de 357 motifs dont 32 pour la classe terminaison, 104 pour la classe bifurcation et 221 pour la classe non minutie. Le commun entre ces motifs est que le centre de la fenêtre (bloc) est un pixel noir et représente le point terminaison ou bifurcation.

La figure 40 illustre quelques exemples pour chaque classe.

- Classe des terminaisons :



- Classe des bifurcations :



- Classe des non minuties :

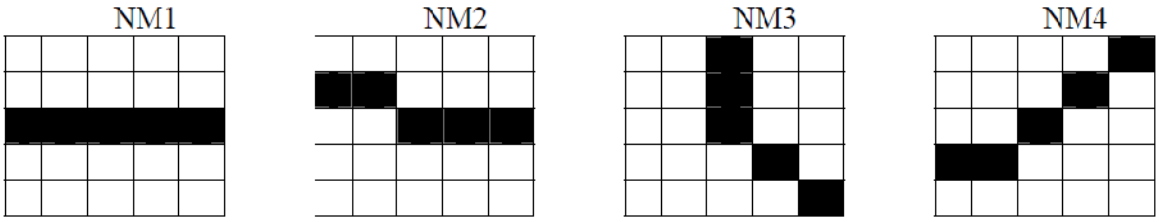


Figure 40. Exemples de chaque classe

III.1.3. Phase de comparaison

Plusieurs méthodes de comparaison sont traitées dans la littérature. La plus connue se base sur les cordonnées de chaque point minutie, le type et l'orientation. Les grands problèmes de cette méthode consistent au déplacement, à la rotation et à la pression de l'empreinte. Afin de résoudre ces problèmes, plusieurs travaux se basent sur la recherche du

centre de l'empreinte. Mais ces travaux manquent encore de précision lors de la détection du centre.

Nous proposons comme solution de ces problèmes une méthode qui se base sur la relation entre point minutie et indépendante des cordonnées.

La méthode de comparaison se base sur le calcul de la distance Euclidienne de l'équation suivante entre deux points minuties successives en se balayant verticalement (figure 41).

$$Dis\tan ce_{(M1,M2)} = \sqrt{(x_2 - x_1)^2 - (y_2 - y_1)^2} \qquad (24)$$

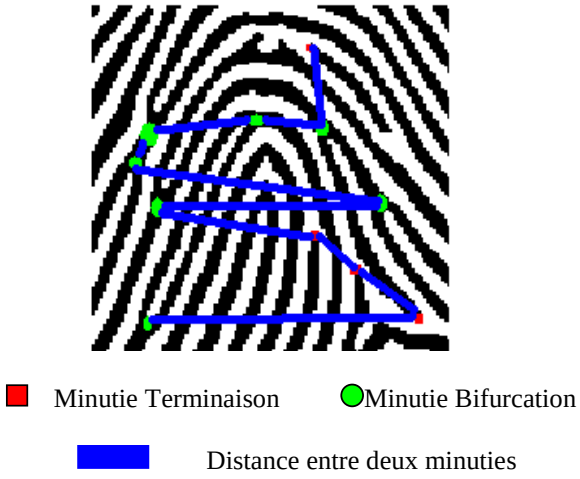


Figure 41. Méthode de comparaison

Notre méthode se base aussi sur le type de minutie et la direction entre deux points successifs. Les équations illustrent la méthode.

$$Direction_{(M1,M2)} = \frac{y_2 - y_1}{x_2 - x_1} \qquad (25)$$

$$Type_{(M1,M2)} = \begin{cases} 11 & \text{Bifurcation} - \text{Bifurcation} \\ 10 & \text{Bifurcation} - \text{Terminaison} \\ 01 & \text{Terminaison} - \text{Bifurcation} \\ 00 & \text{Terminaison} - \text{Terminaison} \end{cases} \qquad (26)$$

avec M1(x1,y1) et M2(x2,y2)

Afin d'optimiser l'algorithme de comparaison, nous avons proposé de remplacer la direction par un calcul d'orientation comme l'indique l'équation de la direction et la figure 42 pour gagner en terme de mémoire et en terme de calcul.

$$Alpha = |\arctan(direction1)| + |\arctan(direction2)| \quad (27)$$

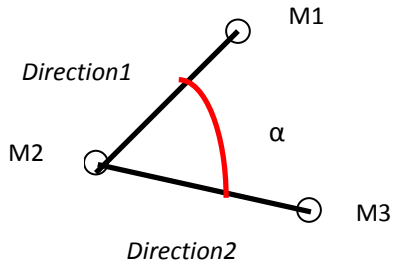


Figure 42. Angle entre trois minuties

III.1.4. Validation et performance:

Afin de valider la méthode de classification DECOC pour ce type de classification, on a utilisé la base de données universelle FVC2004 DB3_B.

La performance d'un système biométrique basé sur l'empreinte digitale n'est validée qu'à travers certaine taux, pour cela on définit les termes suivants :

- ✓ « False Acceptance Rate » (FAR) : Ce facteur présente le taux des fausses reconnaissances. Plus ce taux est faible plus la méthode est meilleure.
- ✓ « False Rejection Rate » (FRR) : Ce facteur présente le taux d'élimination de correcte empreinte. Plus ce taux est faible plus la méthode est meilleure.

Le tableau 3 compare tous le système de reconnaissance par rapport à quelques travaux antérieurs.

	FAR	**FRR**
Méthode de (HAO G., 2005)	4.18%	9.93%
Méthode de (Omer S., 2009)	1,12%	Not indicated
Méthode de (Ying HAO)	1%	2.5%
Méthode de (Jiong Z, 2008)	0.04%	1.31%

La nouvelle méthode	**0%**	**0.02%**

Tableau 3 : Comparaison entre différentes méthodes de reconnaissance par empreinte digitale

Notre système de reconnaissance d'une part montre des bons résultats au niveau du taux d'acceptation et d'autre part atteint un taux de reconnaissance de 88.88% avec un temps d'exécution globale de 7s sur un ordinateur core 2 duo 1.66 Ghz.

III.2. Validation de la Simulation/Emulation

La première étape de l'implémentation représente la phase du partitionnement. Cette étape a pour but de diviser le système en des modules logiciels et d'autres modules comme étant des composants matériels. Le principe de partitionnement est basé en grande partie sur le critère temps d'exécution : « Le module qui consomme beaucoup plus de temps sera sous forme matérielle afin d'atténuer le temps d'exécution».

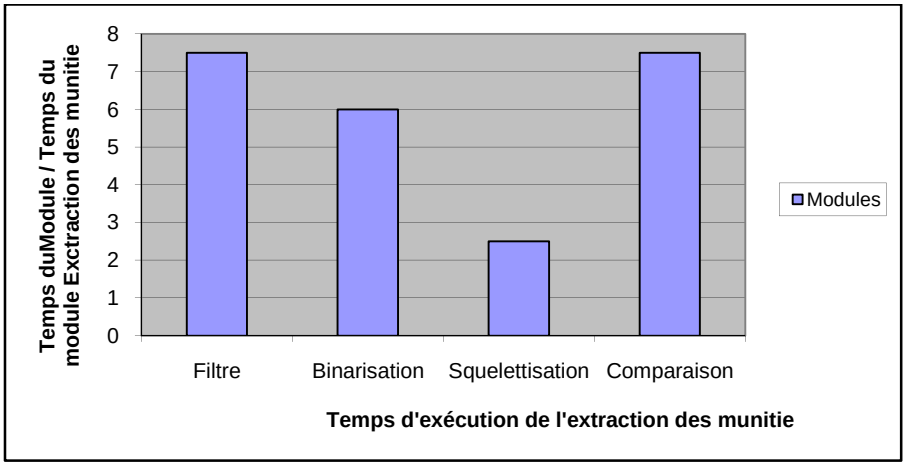


Figure 43. Rapport de temps d'exécution

Un calcul de rapport entre le temps d'exécution de chaque étape de la chaine de reconnaissance par empreinte digitale (sauf le module extraction) par rapport au temps d'exécution du module extraction des minuties (qui représente le temps d'exécution minimal) est effectué. La figure 43 représente l'histogramme correspondant du rapport.

En se basant sur ces résultats, on partitionne notre système statiquement comme suit :

Composants matériels

- Lecture de l'empreinte
- Filtrage
- Binarisation

- Comparaison

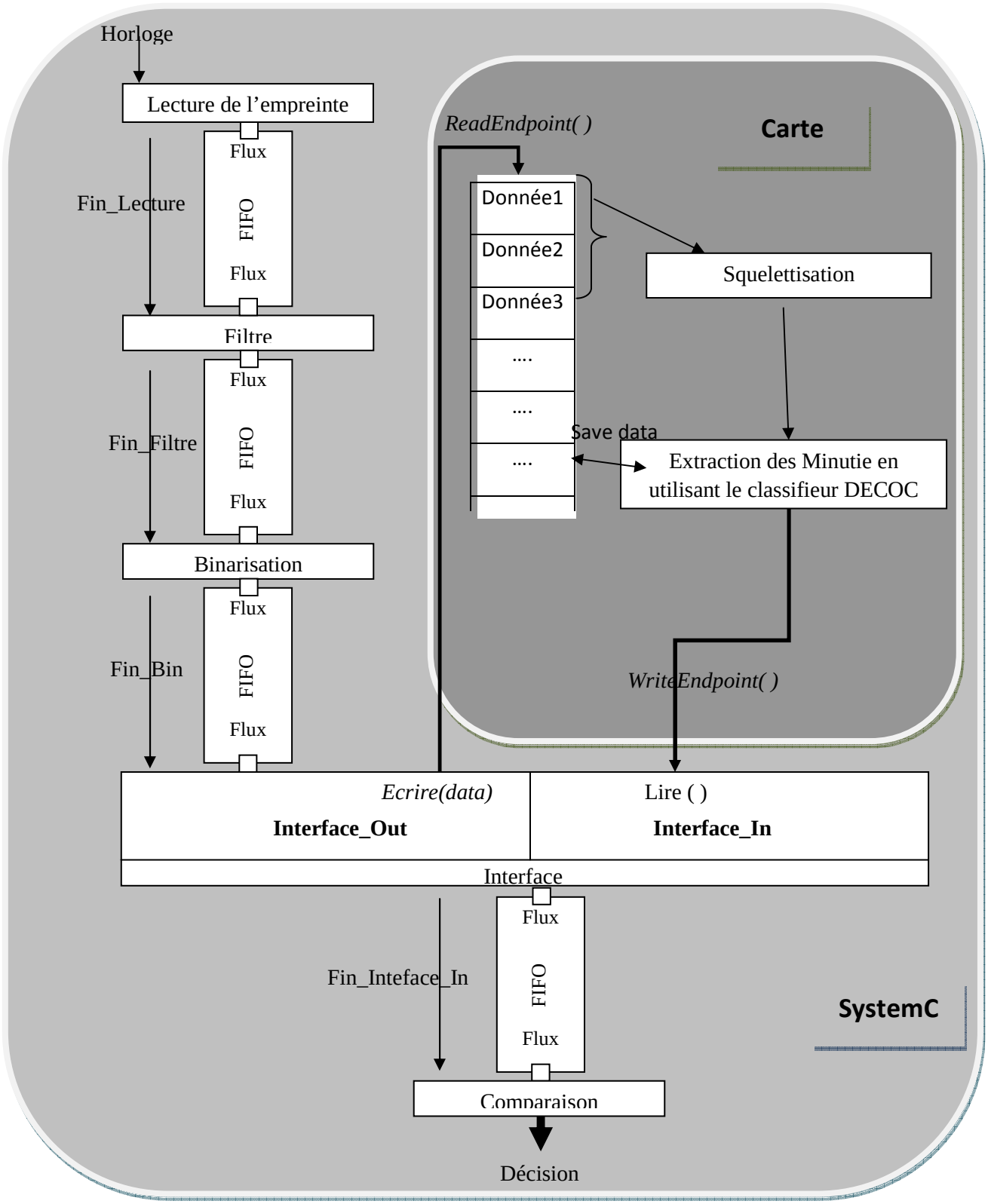


Figure 44. Implémentation de l'application

Applications logicielles

- Squelettisation
- Extraction des minuties

La figure 44, détaille l'implémentation de l'application selon les interfaces décrit dans le chapitre 2

III.3. Résultats de la Simulation/Emulation

La validation de l'environnement de simulation/émulation matériel/logiciel se base sur le temps de simulation global. Le tableau 4 décrit le temps de simulation de chaque module et montre une grande opportunité de tel environnement. En fait, les deux points clé qui soulignent le temps de simulation court de notre environnement sont la liaison USB et le modèle de synchronisation.

	Module	**Temps (s)**
Composant matériel	Lecture de l'empreinte	**0.03**
	Filtre	
	Binarisation	
	Comparaison	
Interface	Interface	**0.5**
Application logicielle	Squelettisation	**0.01**
	Extraction des Minutia	
Simulation		**0.54**

Tableau 4: Temps de simulation de l'application

IV. Expérimentation de la simulation matériel/logiciel en boucle

Deux applications sont utilisées pour valider la simulation HSIL décrite dans le chapitre précédent.

- La première application désigne la régulation et le contrôle de la vitesse d'un moteur à courant continu.
- La deuxième présente un système de contrôle en boucle fermée de la vitesse du moteur.

IV.1. Présentation des applications de test

Nous avons utilisé dans cette section deux applications pour valider la simulation matériel/logiciel en boucle : un régulateur de la vitesse d'un moteur à courant continu et un système de contrôle en boucle fermée de la vitesse du moteur.

IV.1.1. Régulateur de la vitesse d'un moteur à courant continu

Les systèmes de contrôle moteur à courant continu (figure 45) se basent sur un actionneur commun qui fournit un mouvement de rotation. Le circuit électrique équivalent de l'induit et du rotor sont présentés dans la figure suivante.

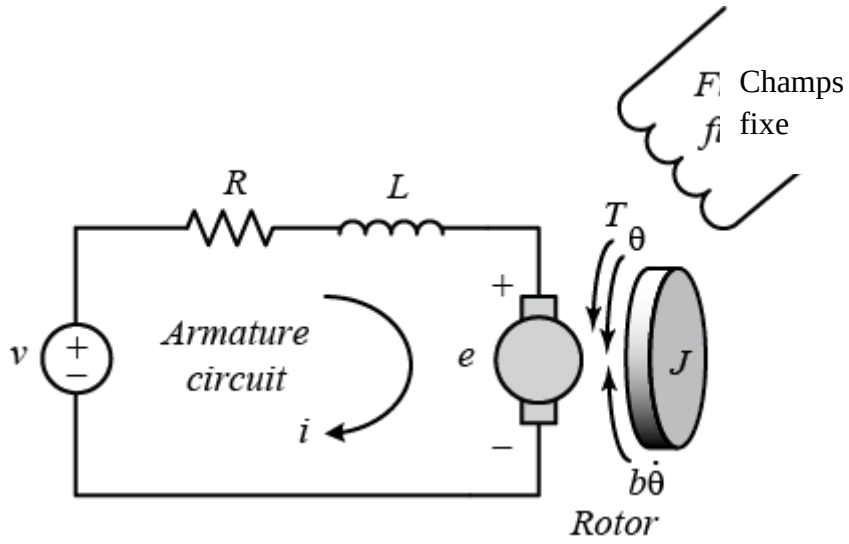


Figure 45. Schéma équivalent d'un moteur à courant continu

On considère que V est la tension d'entrée appliquée sur le moteur et la vitesse de rotation de l'arbre est la sortie du système. Le rotor et l'arbre sont supposés rigides.

Les EDOs

Les paramètres du système sont :

- θ : La vitesse du moteur exprimée en $tr.min^{-1}$
- Ke : Gain statique exprimé en $tr.min^{-1}.V^{-1}$
- Kt : Force électromotrice.
- J : Moment d'inertie du rotor exprimé en $Kg.m^2.s^{-2}$
- b : Rapport d'amortissement du système mécanique.

- R : Résistance électrique exprimée en Ohm.
- L : Inductance électrique exprimée en H.
- V : Voltage d'entrée du moteur.

Les relations issues du système décrit sont :

Le moment du torque prend l'équation suivante : $T = K_t i$ (28)

La force contre-électromotrice est proportionnelle à la vitesse angulaire de l'arbre : $e = K_e \dot{\theta}$ (29)

D'après les Lois de Newton et de Kirchhoff on obtient :

$$J\ddot{\theta} + b\dot{\theta} = Ki \qquad (30)$$

$$L\frac{di}{dt} + Ri = V - K\dot{\theta} \qquad (31)$$

On Applique par la suite le transformé de Laplace :

$$s(Js + b)\Theta(s) = KI(s) \qquad (32)$$

$$(Ls + R)I(s) = V(s) - Ks\Theta(s) \qquad (33)$$

La fonction de transfert en boucle ouverte du système est le suivant :

$$P(s) = \frac{\dot{\Theta}(s)}{V(s)} = \frac{K}{(Js + b)(Ls + R) + K^2} \qquad \left[\frac{rad/\sec}{V}\right] \qquad (34)$$

Les figures 46 et 47 montrent le schéma bloc du système Simulink et l'implémentation sous Matlab/Simulink avec :

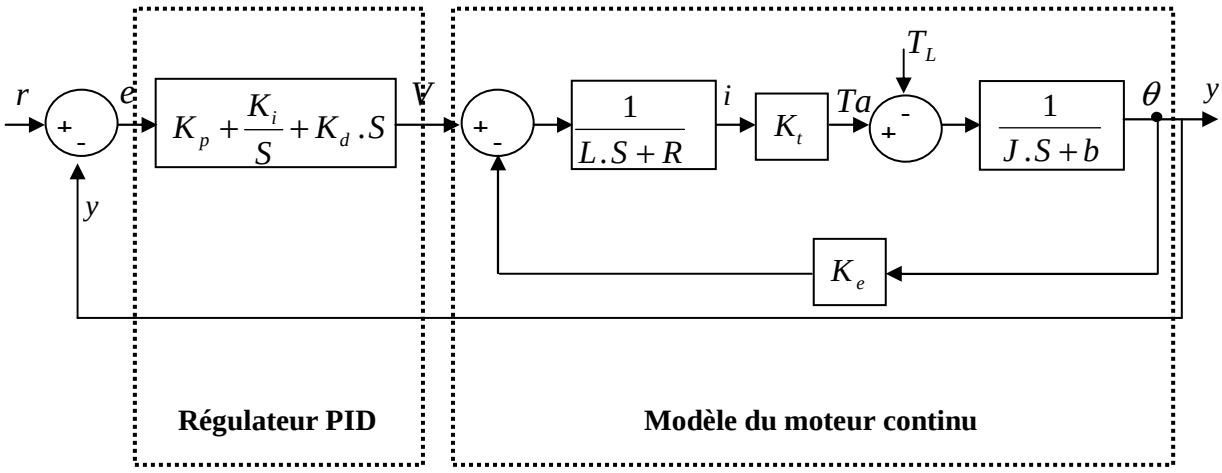


Figure 46. Diagramme de bloc d'un régulateur de vitesse d'un moteur continu

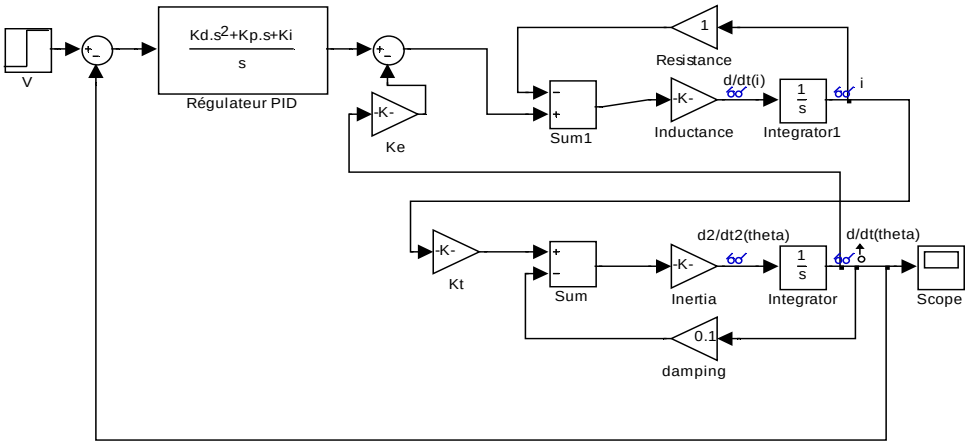


Figure 47. Implémentation du système en Simulink

IV.1.2. Système de contrôle en boucle fermée de la vitesse du moteur

Cet exemple qui est décrit dans la démonstration de Simulink présente le model d'un moteur. Le collecteur d'admission déclenche le transfert de l'air-carburant vers les cylindres par l'intermédiaire des soupapes à événements discrets. En même temps, les processus à temps continu flux d'admission, la génération de couple et l'accélération sont en exécution. L'actionneur de papillon de gaz assure la régulation de la vitesse.

Ce modèle est basé sur les résultats publiés par Crossley et Cook (Crossley P.R., 1991). Il décrit la simulation d'un moteur à quatre cylindres à allumage par étincelle interne. Le travail Crossley et Cook montre aussi comment une simulation basée sur ce modèle a été validée par des données d'essais dynamométriques. Le modèle est composé essentiellement de cinq modules :

1. Accélérateur ("Throttle")
2. Collecteur d'admission ("Intake manifold")
3. Débit massique d'admission ("Intake Mass Flow Rate")
4. Course de compression ("Compression Stroke")
5. Génération de couple et d'accélération ("Torque Generation and Acceleration")

Accélérateur ("Throttle")

Le premier élément du modèle est l'accélérateur. L'entrée de commande est l'angle de la plaque et à vitesse à laquelle le modèle introduit de l'air dans le collecteur d'admission. La vitesse peut être exprimée comme le produit de deux fonctions:

1. une fonction empirique de l'angle de papillon des gaz.

2. une fonction de la pression atmosphérique et de collecteur.

En cas de pression sur le collecteur, le débit qui traverse le module accélérateur est en fonction de l'angle d'accélération. Ce modèle tient compte de ce comportement à basse pression avec un état de commutation dans les équations de compressibilité indiquées dans l'équation 35.

$$\dot{m}_{ai} = f(\theta).g(P_m) = \text{débit massique dans le collecteur (g/s)} \qquad (35)$$

avec

$$f(\theta) = 2.821 - 0.05231.\theta + 0.10299.\theta^2 - 0.00063.\theta^3$$

$$g(P_m) = 1; \quad Si \ P_m \le P_{amb}/2$$

$$g(P_m) = \frac{2}{P_{amb}}\sqrt{P_m P_{amb} - P_m^2}; \quad Si \ P_{amb}/2 \le P_m \le P_{amb}$$

$$g(P_m) = -\frac{2}{P_{amb}}\sqrt{P_m P_{amb} - P_m^2}; \quad Si \ P_{amb} \le P_m \le 2P_{amb}$$

$$g(P_m) = -1; \quad Si \ 2P_{amb} \le P_m$$

θ : Angle d' accélérati on (deg)

P_m : Pression du collecteur (bar)

P_{amb} : Pression ambiante (atmosphérique) (bar)

Collecteur d'admission ("Intake manifold")

Le modèle du collecteur d'admission se base sur une équation différentielle de la pression d'admission. La différence dans les taux d'entrée et de sortie de débit massique représente le taux de changement net de la masse d'air par rapport au temps. Cette quantité, selon la loi des gaz parfaits, est proportionnelle à la dérivée temporelle de la pression du collecteur (voir équation 36). Notez que, contrairement au modèle de Crossley et Cook, ce

modèle ne tient pas compte de la recirculation des gaz d'échappement, mais cela peut facilement être ajouté.

$$\dot{P}_m = \frac{RT}{V_m}(\dot{m}_{ai} - \dot{m}_{ao}) \qquad (36)$$

Avec

R : Constant du gaz.

T : température (K).

V_m : Volume du collecteur (m^3).

$\dot{m}_{ao}$: Débit massique de l'air de sortie du collecteur.

$\dot{P}_m$: Taux de changement de pression dans le collecteur (bar / s).

Débit massique d'admission ("Intake Mass Flow Rate")

Le débit massique de l'air que les pompes des cylindres du collecteur est décrit par l'équation 37. Cette équation est dérivée d'une manière empirique. Ce taux est une fonction de la masse de la pression d'admission et de la vitesse du moteur.

$$m_{ao} = -0.366 + 0.08979.N.P_m - 0.0337.N.P_m^2 + 0.0001.N^2.P_m \qquad (37)$$

Avec

N : Vitesse angulaire du moteur (rad/s).

Pm : Pression du collecteur (bar).

Pour déterminer le volume d'air total dans les cylindres, la simulation intègre le débit massique à partir du collecteur d'admission. Ceci permet de déterminer la masse d'air totale qui est présente dans chaque cylindre après la course d'admission et avant la compression.

Course de compression ("Compression Stroke")

Le vilebrequin assure le déclenchement des cylindres du moteur un par un. Le passage d'un cylindre à un autre est indiqué par une tour de manivelle. Dans ce modèle, l'admission, la compression, la combustion, et l'échappement se produisent simultanément. Pour prendre en considération la compression, la combustion est retardée de 180 degrés lors de la rotation du vilebrequin.

Génération du couple et d'accélération ("Torque Generation and Acceleration")

Le dernier élément de la simulation décrit le couple développé par le moteur. Une relation empirique dépendante de la masse de l'air, du rapport air / carburant et de la vitesse du moteur est utilisée pour le calcul du couple (voir équation 38).

$$Torque_{eng} = -181.3 + 379.36.m_a + 21.91.\left(\frac{A}{F}\right) - 0.85.\left(\frac{A}{F}\right)^2 + 0.26.\sigma - 0.0028.\sigma^2 + 0.027.N - 0.000107.N^2 + 0.00048.N.\sigma + 2.55.\sigma.m_a - 0.05.\sigma^2.m_a \quad (38)$$

m_a : Masse de l'air dans le cylindre (g).

$\left(\frac{A}{F}\right)$: Rapport air-carburant.

σ : Avance de l'allumage.

$Torque_{eng}$: Couple développé par le moteur (Nm).

Puis l'accélération angulaire du moteur est calculée en utilisant l'équation 39.

$$J\dot{N} = Torque_{eng} - Torque_{load} \quad (39)$$

J : Moment d'inertie du moteur ($kg.m^2$).

$\dot{N}$: Accélération angulaire du moteur (rad/s^2).

Modèle en boucle fermé

La figure 48 décrit le modèle du système au niveau fonctionnel.

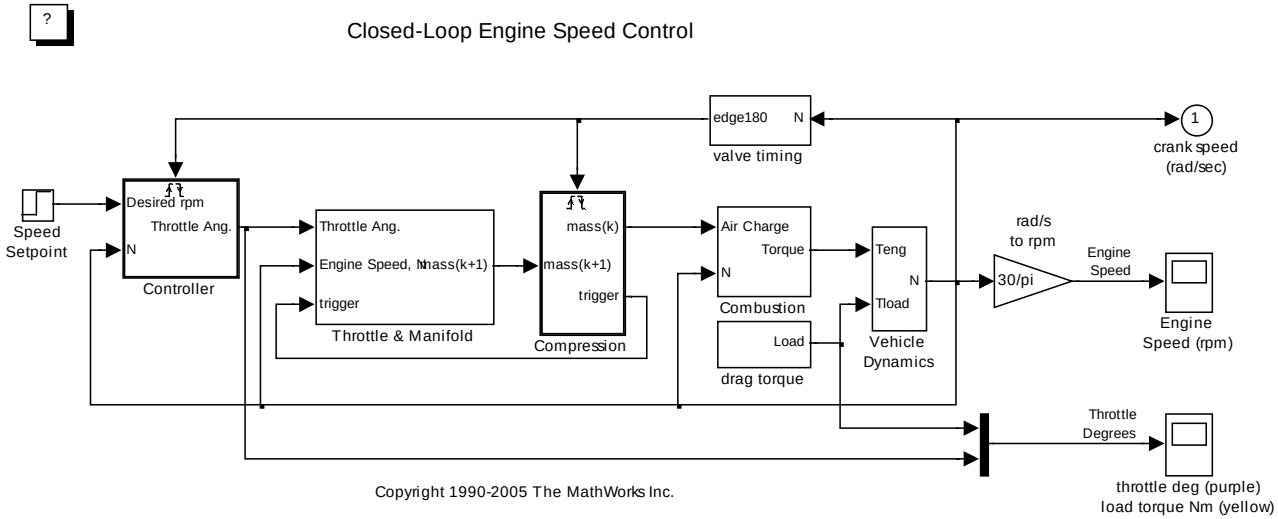


Figure 48. Modèle en boucle fermé d'un contrôleur de la vitesse d'un moteur

IV.2. Validation de la simulation matériel/logiciel en boucle

1ère Application : Régulateur de la vitesse d'un moteur à courant continu

Les intégrateurs de moteur et le régulateur PID sont les contrôleurs numériques à simuler en utilisant la simulation HSIL. Pour cela une intégration du bloc Synchronisation est faite dans la figure 49.

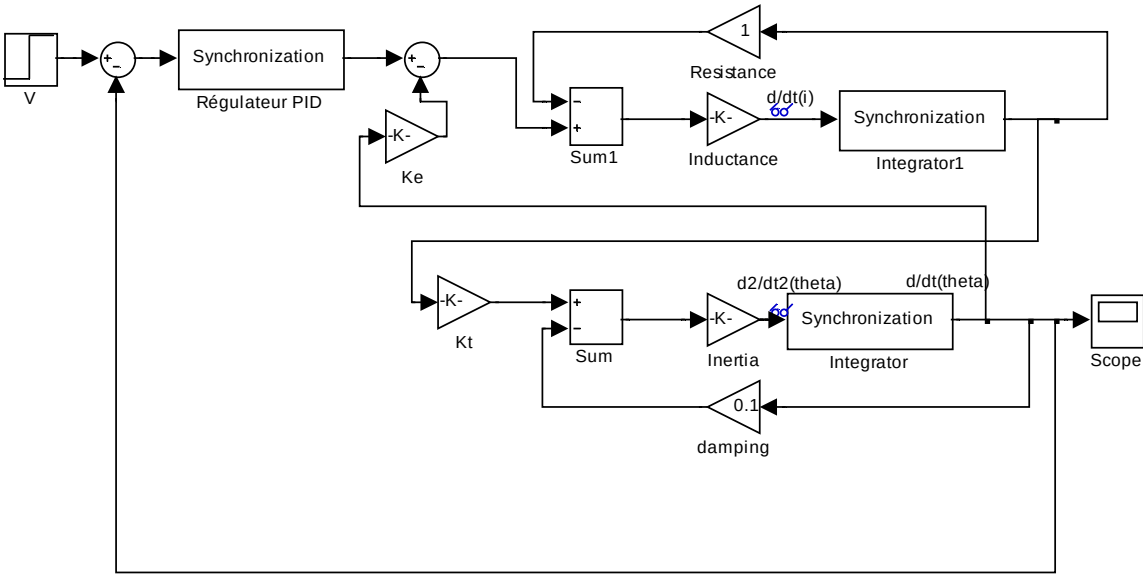


Figure 49. Modèle bloc de l'application 1 basé sur la simulation HSIL

2ème Application: Contrôle en boucle fermée de la vitesse du moteur

Deux blocs sont considérés comme des contrôleurs numériques : "controller" et "compression". Ces deux blocs sont implantés comme des applications logicielles. La figure 50 montre le modèle en utilisant la technique HSIL.

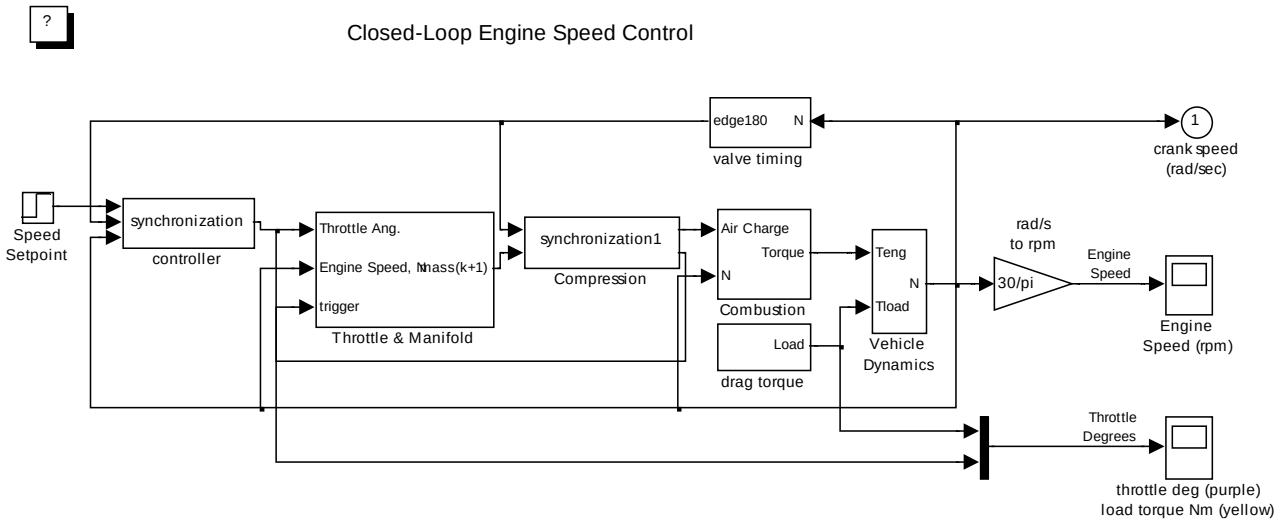


Figure 50. Modèle bloc de l'application 2 basé sur la simulation HSIL

IV.3. Résultats de la simulation matériel/logiciel en boucle

1ère Application : Régulateur de la vitesse d'un moteur à coudant continu

La simulation est basée sur l'environnement Matlab/Simulink et l'environnement NIOS II (figure 51).

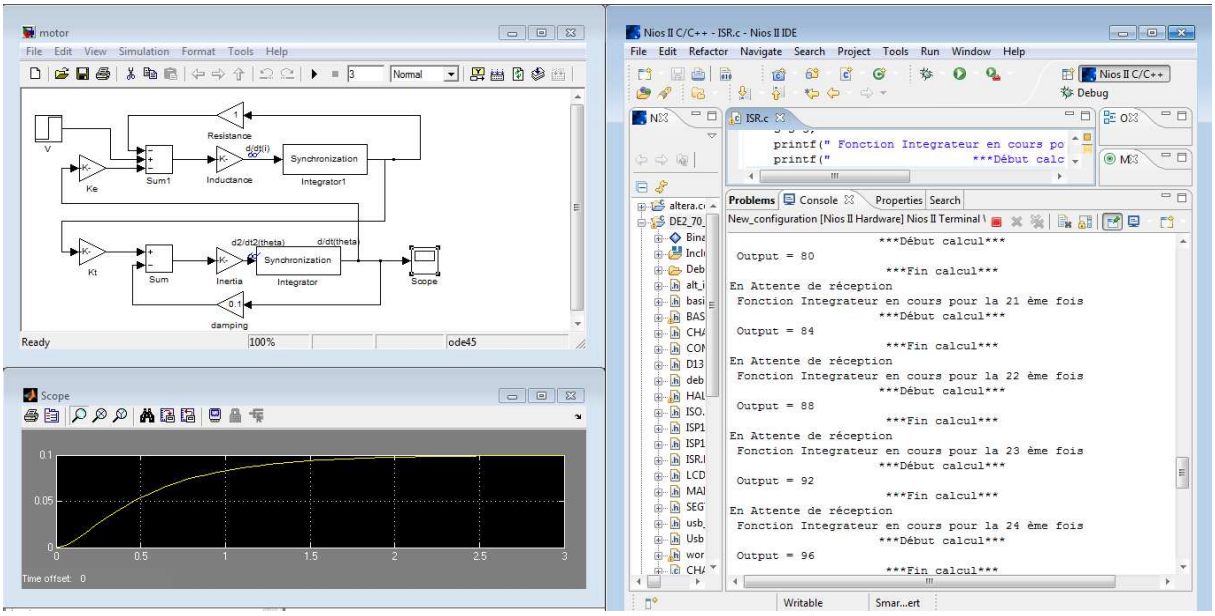
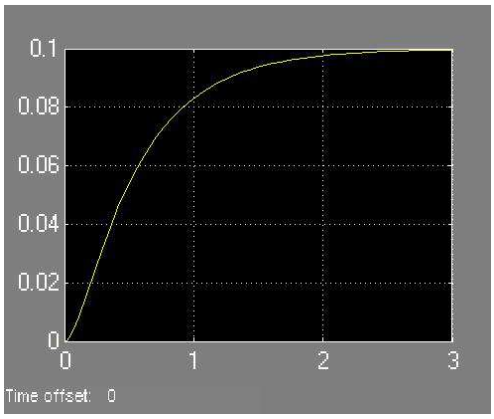
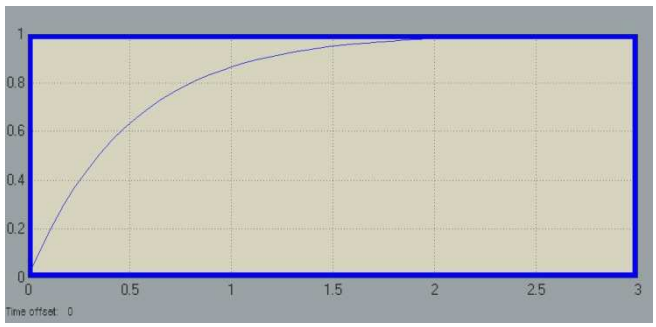


Figure 51. Environnement Simulink/NIOSII pour la simulation HSIL

Afin de valider la technique HSIL proposée deux points de vérification clés sont indispensables :

- Fiabilité du système : Ceci est vérifié en comparant les courbes d'entrées / sorties pour chaque bloc de synchronisation par rapport au modèle prédéfini. La figure 52 montre les différents signaux utilisés.





a) Signal de la vitesse du moteur b) Signal d'intensité *i*

c) Signal du sortie régulateur

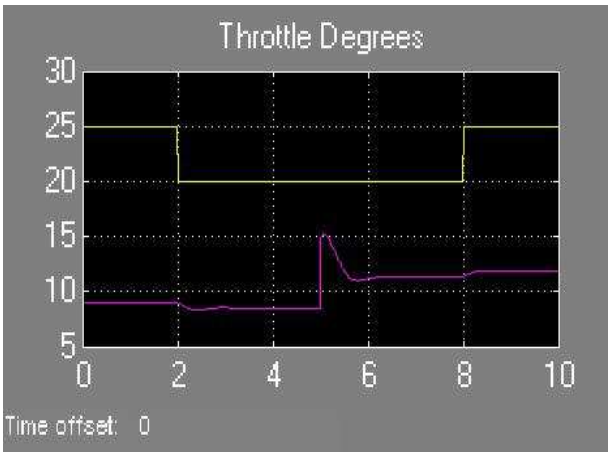
Figure 52. Les signaux critiques utilisés pour la vérification de l'application 1

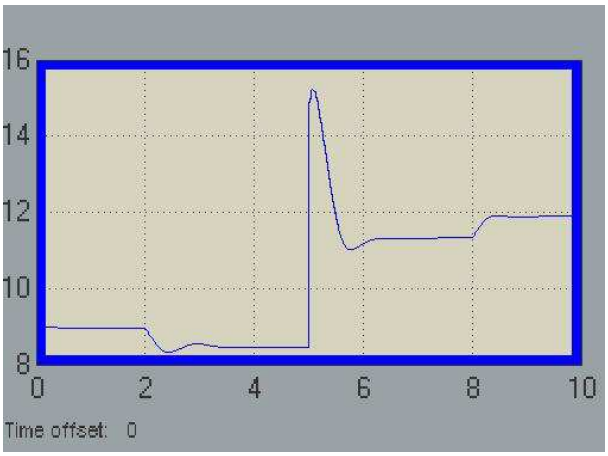
- Le temps de simulation est 1 seconde.

2ème Application: Contrôle en boucle fermée de la vitesse du moteur

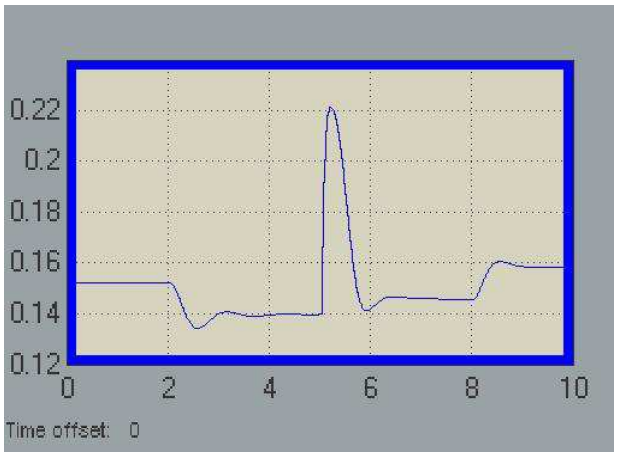
De même que l'application 1, la simulation de l'application 2 est vérifiée surtout par les signaux de sortie des blocs "controller" et "compression" (figure 53).

Le temps de simulation global du modèle en boucle fermée est 2 secondes.





a) Signal d'accélération b) Accélération angulaire



c) Débit massique d'admission

Figure 53. Les signaux critiques utilisés pour la vérification de l'application 2

V. Expérimentation de l'environnement CODIS+

Les systèmes de sécurité et de contrôle pour les automobiles deviennent de plus en plus complexes grâce à la grande révolution des technologies numériques. Afin de valider l'environnement CODIS+, nous proposons un système limiteur de vitesse d'un véhicule.

V.1. Application : système limiteur de vitesse

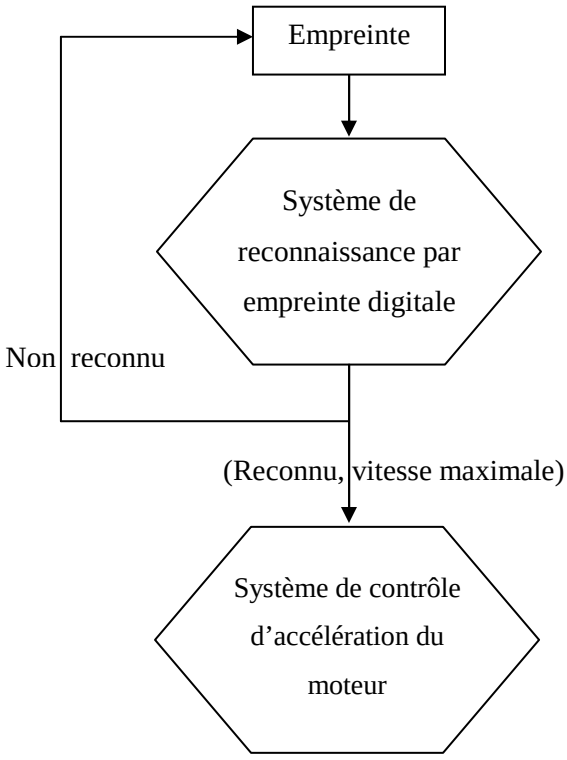


Figure 54. Graphe fonctionnel du système

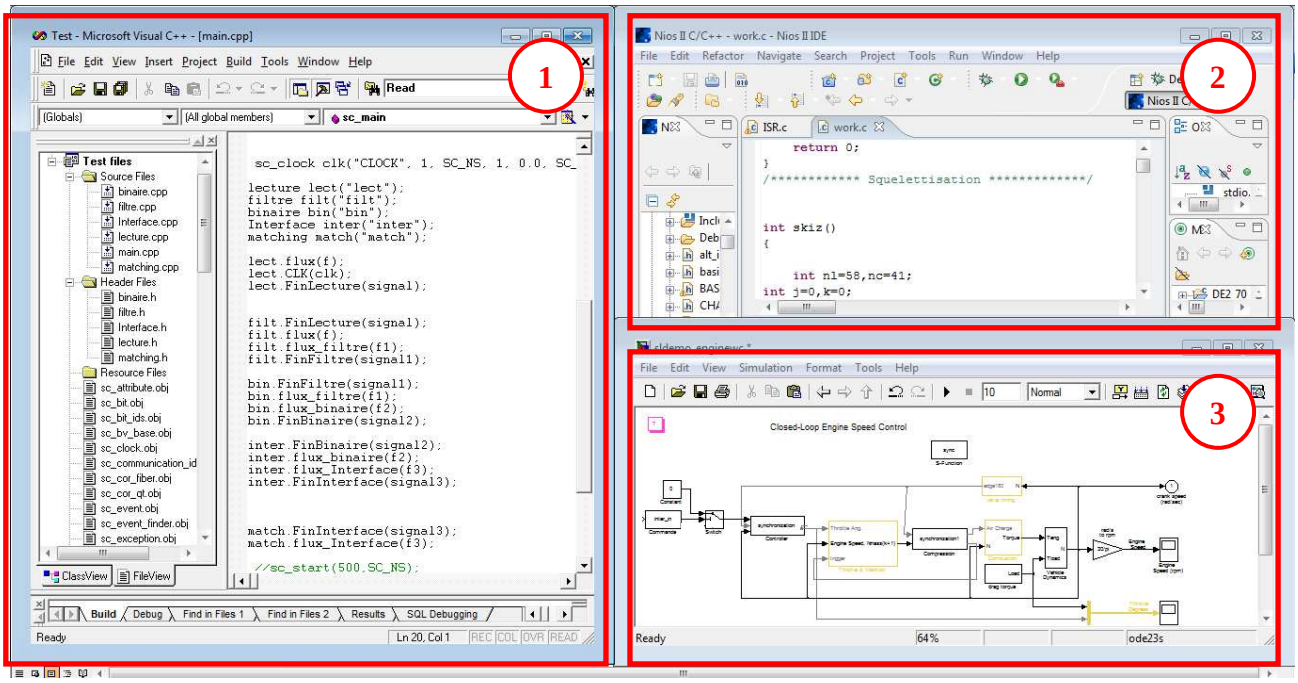
Il existe plusieurs systèmes mécatroniques qui traite le problème de contrôle de vitesse d'un véhicule. Il existe essentiellement deux types : contrôle obligatoire et contrôle facultatif. Nous proposons dans ce contexte un système qui limite la vitesse du moteur selon le conducteur. Pour cela une identification du conducteur par empreinte digitale permet de fixer la valeur maximale que le conducteur peut atteindre.

Le système limiteur de vitesse représente le couplage entre le système de contrôle du moteur ainsi décrit dans la section précédente et le système de reconnaissance par empreinte digitale. La figure 54 donne le graphe fonctionnel du modèle.

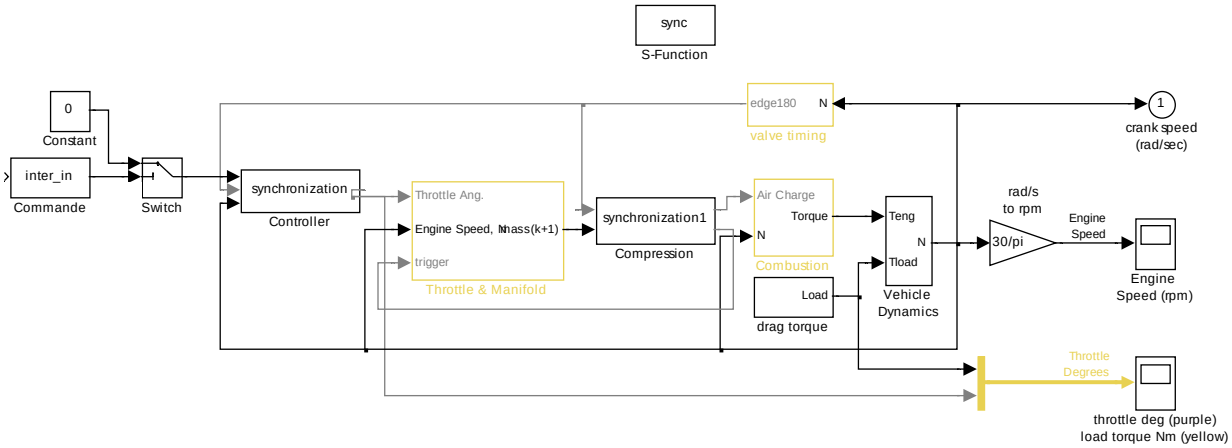
V.2. Implémentation et résultats

Le cycle de la simulation commence à partir du SystemC. Les modules matériels seront simulés par le noyau de SystemC et les applications logicielles seront simulées sur l'architecture cible implantée sur la carte FPGA. Lorsqu'une personne est identifiée par le système de reconnaissance par empreinte digitale l'interface *Interface_Out* envoie un signal déclencheur vers le bloc *Inter_In* et un changement de contexte vers l'environnement Simulink est procédé.

Le bloc *Inter_In* est lié directement à la position 2 d'un commutateur. Ce dernier joue le rôle d'un déclencheur pour le système de contrôle d'accélération du moteur. Une personne n'est pas reconnue, le système a pour vitesse d'entrée égale à 0 (position 1 du commutateur). Lorsque l'identification d'une personne est réussie l'entrée 2 du commutateur, qui correspond à la sortie du *Inter_In*, prend la valeur vrai ce qui permet le changement de position vers 3 et le système de contrôle d'accélération du moteur commence son cycle de simulation. La figure 55 a) montre l'implémentation de l'application dans notre environnement CODIS+ :co-simulation/émulation continu/discret. La zone 1 représente la description matérielle de l'étape de filtrage, binarisation et comparaison en SystemC. La zone 2 décrit les applications logicielles – squelettisation et extraction des minuties – en utilisant l'outil NIOSII IDE. La zone 3 et la figure 55 b) décrit le schéma de bloc du système de contrôle d'accélération du moteur dans Simulink.



a) L'environnement CODIS+ : Co-simulation/émulation des systèmes continus/discrets



b) schéma bloc du système de contrôle d'accélération du moteur dans Simulink

Figure 55. Implémentation de l'application

Notre système permet à la fois d'utiliser les modèles de synchronisation matériel/logiciel, la simulation HSIL et le modèle de synchronisation continu/discret présenté dans le chapitre précédent à la section III.2.

Le temps de simulation global est 2.55s.

VI. Conclusion

Dans la première partie de ce chapitre, nous avons validé l'environnement de Simulation/Emulation. Nous avons développé pour cela le système de reconnaissance par

empreinte digitale puisque tel système est considéré comme étant un système complexe qui se compose nécessairement des composants matériels et des applications logicielles.

Dans la deuxième partie, l'implémentation de la technique de la simulation matériel/logiciel en boucle est présentée. L'application de régulation de la vitesse d'un moteur à courant continu et l'application de contrôle en boucle fermée de la vitesse du moteur sont utilisées pour valider la simulation HSIL.

Dans la troisième partie de ce chapitre, nous avons implémenté l'environnement CODIS+ de co-simulation/émulation des systèmes continu/discret. Le système limiteur de vitesse a été utilisé pour exploiter CODIS+. Les résultats trouvés prouvent l'importance de notre environnement.

CONCLUSION GENERALE

I. Conclusion

L'hétérogénéité présente une caractéristique indispensable dans les systèmes actuels. Les systèmes continus/discrets représente l'intérêt de notre travail. La validation globale de ces systèmes demande des langages fournissant les formalismes nécessaires pour la modélisation et la demande des outils de simulation précis et performants. Actuellement, il existe plusieurs langages et outils pour chaque domaine. Le grand problème c'est que les concepteurs appartenant à chaque domaine utilisent ces outils pour simuler leurs modèles mais sans avoir une validation globale du système entier. Pour résoudre ce problème, ce travail a proposé une extension de l'environnement CODIS afin d'accélérer le temps de simulation d'une part et d'ajouter d'autre fonctionnalité pour supporter la complexité croissante du domaine discret d'autre part. Notre environnement utilise SystemC et une carte FPGA à base d'une architecture cible pour le domaine discret et le simulateur Simulink pour le modèle continu. Ceci permet de bénéficier de toute l'expertise de leurs langages et des outils de débogage, et permet également d'exploiter les modèles et les bibliothèques existants.

Le premier chapitre a présenté le principe de modélisation et de simulation de chaque domaine (discret, continu). Une étude approfondie sur les solutions et les travaux antérieurs pour la modélisation et la simulation des systèmes hétérogènes. En s'appuyant sur cet état de l'art, nous avons fixé le meilleur chemin à suivre pour la modélisation et la simulation.

Le chapitre 2 a abordé les différentes méthodes de simulation d'un modèle discret dans la première partie. Plusieurs modèles de synchronisation entre le simulateur SystemC et une architecture cible sont proposés. En fait, cette architecture est implantée sur une carte FPGA afin de remplacer l'ISS par le processeur cible et d'accélérer le temps de simulation. La communication qui se base sur la liaison USB assure le transfert des paquets en mode interruption (interruption matérielle).

Dans la première partie du chapitre 3 nous avons annoncé une nouvelle technique de simulation pour les contrôleurs numériques. La simulation matériel/logiciel en boucle permet de surmonter le problème de la complexité des contrôleurs en adaptant la stratégie de Co-design pour la modélisation. En fait, cette technique diminue le temps de mise en marché et

facilite la modélisation. Une interface est utilisée dans Simulink pour assurer la synchronisation avec l'architecture cible implantée dans la carte.

Dans sa deuxième partie, une présentation de l'environnement CODIS permet de proposer un modèle de synchronisation global qui englobe les deux simulateurs et l'émulateur. Une étude théorique d'un exemple est détaillée afin de démontrer tous les scénarios possibles lors d'une simulation.

Dans le chapitre 4, dans un premier lieu, nous avons présenté le système de reconnaissance par empreinte digitale comme étant une partie de l'application globale. Une étude détaillée sur l'apport ajouté dans la phase d'extraction (utilisation du classifieur DECOC) durant la présentation de l'application. Afin de valider les différents environnements, une architecture à base du processor NIOS II est implantée sur la carte. Dans un deuxième lieu, nous avons validé l'environnement de simulation/émulation matériel/logiciel, la simulation HSIL et l'environnement de co-simulation/émulation continu/discret. Les expérimentations ont montré une excellente précision et une vitesse de simulation acceptable.

En conclusion, le niveau de difficulté d'implémentation des modèles de vérification réside dans la nature des simulateurs utilisés. Dans le cas des simulateurs commercialisés (fermés), cette implémentation devient difficile surtout si le constructeur ne fournit pas d'outils supplémentaires à son simulateur. Elle est plus facile dans le cas des simulateurs à source ouverte.

II. Perspectives

Il existe plusieurs outils de CAO pour la modélisation et la vérification des systèmes. Chaque outil possède des avantages et des inconvénients. La meilleure solution consiste à utiliser les avantages de chaque outil et à éviter ces limites. En fait, cette solution se base sur des interfaces génériques et assurant la communication et la synchronisation entre différents simulateur.

Dans ce contexte nous proposons de

- Voir l'intégration des simulateurs continus à source libre (Modelica, Scilab, PtolemyII), ce qui va permettre de voir d'autres implémentations plus optimisées des interfaces dans le cas du simulateur continu.
- Proposer un modèle de synchronisation matériel/logiciel multi-niveau entre le simulateur SystemC et la carte dont des composants décrit en bas niveau. En effet, le

modèle doit supporter la bibliothèque SCE-MI (Standard Co-Emulation Modeling Interface) afin d'assurer la communication entre deux processus dans différent niveau d'abstraction.

BIBLIOGRAPHIES

(Abid M., 1998) Abid M., « Rapid prototyping environment for design of hardware/software electronic systems Electronics », IEEE International Conference on Circuits and Systems, 1998, vol.1, p. 531 – 535.

(Agilent, 2012) Simulateur d'Agilent, disponible à http://www.home.agilent.com

(Al-Junaid H., 2004) Al-Junaid H. and Kazmierski T. J., « SEAMS - a SystemC Environment with Analog and Mixed- Signal Extensions », IEEE International Symposium on Circuits and Systems, 2004.

(Al-Junaid H., 2005) Al-Junaid H., Kazmierski T., « Analogue and mixed-signal extension to SystemC », IEE Proceedings of Circuits, Devices and Systems, 9 Dec. 2005, p. 682 – 690.

(Antao B. A., 1996) Antao B. A. A., « AHD languages- A Must for Time-Critical Designs », IEEE Circuits and Devices Magazine, Volume 12, Issue 4, July 1996, p. 12 – 17.

(Antoine G., 2007) Antoine Girard and George J. Pappas, "Approximation Metrics for Discrete and Continuous Systems », IEEE Transactions On Automatic Control, Vol. 52, No. 5, May 2007, p.782-798

(Aubepart F., 2003) Aubepart F., P. Poure, Y.A. Chapuis, C. Girerd, F. Braun, "HDL-based
methodology for VLSI design of AC motor" *IEE Proceedings Circuits, Devices and Systems*, vol. 150, no. 1, pp. 38 – 44, Feb. 2003.

(Azam F., 2005) Azam F., Zhang Li, Ahmad R., « Using UML profile for connecting information architecture and detailed information design », Proceedings of the IEEE Symposium on Emerging Technologies, 17-18 Sept. 2005, p. 423 – 428.

(Banzhaf W., 1989) Banzhaf W., Computer-aided circuit analysis using SPICE, Prentice Hall, 1989.

(Bonnerud T.E., 2001) Bonnerud T.E., Hernes B., Ytterdal, T., « A mixed-signal,

functional level simulation framework based on SystemC for system-on-a-chip applications », IEEE Conference on CustomIntegrated Circuits, 2001, p. 541 – 544.

(Bouchhima F., 2005) Bouchhima F., Nicolescu, G., Aboulhamid M., Abid M., « Discrete-continuous simulation model for accurate validation in component-based heterogeneous SoC design », *The 16th IEEE International Workshop on Rapid System Prototyping*, 8-10 June 2005, p.181 – 187.

(Bouchhima F., 2007) Bouchhima F., Gabriela Nicolescu, El Mostapha Aboulhamid, Mohamed Abid. Generic discrete-continuous simulation model for accurate validation in heterogeneous systems design. Microelectronics Journal, Volume 38, Number 1, January 2007, pp. 805-815.

(Brown A. D., 1992) Brown A. D. et al., « The Design of a Language for Mixed-Mode Circuit Simulation », Research Journal: Department of Electronics and Computer Science, University of Southampton, 1992, p. 103-105.

(Callier F.M., 1991) Callier F. M., Desoer C. A., Linear System Theory, Germany, Springer-Verlag, 1991.

(Chang W.T., 1997) Chang W. T., Ha S., Lee E. A., « Heterogeneous simulation - mixing discrete-event models with dataflow », Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology, v 15, n 1-2, 1997, p 127-144.

(Chapoutot A, 2008) Chapoutot Alexandre, « Simulation abstraite : une analyse statique de modèles Simulink », rapport de thèse, 2008.

(Chen C.F., 1984) Chen C. F., Lo C. Y., Nham H. N., Subramaniam P., « The Seconde Generation MOTIS Mixed-Mode Simulation », The 21st IEEE/ACM Design Automation Conference, p. 10-16, 1984.

(Consel C., 2004) Consel Charles, “Generative Programming from a Domain-Specific Language Viewpoint”. In Unconventional Programming Paradigms

(UPP'04), 2004.

(Crossley P.R., 1991) Crossley P. R.and J. A. A. Cook. A nonlinear engine model for drive train system developement. InProc.IEE Int. Conf., Control'91, 2:921–925, Edinburgh,UK, (1991). Conference publication 332.

(De Man H.J., 1980) De Man H. J., Rabaey J., Arnout G., Vandewalle J. « Practical implementation of a general computer aided design technique for switched-capacitor circuits », IEEE Journal of Solide State Circuits SC-15, 1980, p. 190-200.

(Detrey J., 2007) Detrey J., "Arithmétiques réelles sur FPGA, virgule fixe, virgule flottante et système logarithmique", Thèse de doctorat de l'École Normale Supérieure de Lyon, France,2007.

(Drager S.L., 1998) Drager S.L., Carter H.W., Hirsch H.L, « A VHDL-AMS mixed-signal, mixed-technology design tool », IEEE National Aerospace and Electronics Conference, 1998.

(DSP, 2013) DSP Builder Handbook, Volume 2: DSP Builder Standard Blockset, Mai 2013.

(Dubois M., 2011) Dubois Mathieu, "Simulateur compilé d'une description multi-langage des systèmes hétérogènes", Université de Montréal Faculté des arts et des sciences, Juin 2011.

(Eker J., 2003) Eker J. et al. « Taming heterogeneity - the Ptolemy approach », Proceedings of the IEEE Volume 91, Issue 1, Jan. 2003, p.127 – 144.

(Evans, 2003) Evans Data Corporation - «Embedded Systems Development Survey» - Volume 1, 2003,

(Ferretti G., 2006) Ferretti G., Magnani, G., Rocco, P., Vigano, L., «Modelling and simulation of a gripper with Dymola » *Mathematical and Computer Modelling of Dynamical Systems, v 12, n 1*, Feb. 2006, p. 89-102.

(Fitzpatrick D., 1998) Fitzpatrick D., Miller I., Analog Behavioral Modeling with the

Verilog-A Language, Kluwer Academic Publishers, 1998

(Frey P., 2000) Frey P., O'Riordan D., « Verilog-AMS: Mixed-signal simulation and cross domain connect modules », IEEE/ACM International Workshop on Behavioral Modeling and Simulation, 2000, p. 103 – 108.

(Gajski D.D., 2000) Gajski D. D., Zhu J., Dömer R., Gerstlauer A., Zhao S., SpecC Specification Language and Methodology, Kluwer Academic Publisher, 2000.

(Gear C.W., 1985) Gear C. W., Osterby O., « Solving Ordinary Differential Equations with Discontinuities », ACM Transaction on Mathematical Software, vol. 10, 1984, pp. 23- 44.

(Getreu I.E, 1989) Getreu I. E., « Behavioral Modeling of Analog Blocks Using the SABER Simulator », 32nd Midwest Symposium on CAS, 1989, p. 977-980.

(Ghasemi H.R., 2005) Ghasemi H.R., Navabi, Z., « An effective VHDL-AMS simulation algorithm with event », International Conference on VLSI Design, 2005, p. 762-767.

(Gupta G.K.,1985) Gupta G. K., Sacks-Davis R., Tescher P. E., « A review of recent developments in solving ODEs », ACM Computing Surveys (CSUR), Volume 17 Issue 1, 1985.

(Hassairi W., 2012) Walid Hassairi, Moncef Bousselmi, Mohamed ABID, Carlos Valderrama ”MATLAB/SYSTEMC FOR THE NEW CO-SIMULATION ENVIRONMENT BY JPEG ALGORITHM ” INTECH volume 2 chapitre 6 page119,138 , 2012.

(Hao G., 2005) HAO GUO “A Hidden Markov Model Fingerprint Matching Approach”, Proceedings of the Fourth International Conference on Machine Learning and Cybernetics, Guangzhou, IEEE Print ISBN: 0-7803- 9091-1 , 5055 - 5059 Vol. 8 , August 2005.

(Harakawa M., 2005) Harakawa M., H. Yamasaki, T. Nagano, S. Abourida, C. Dufour, J. Bélanger, "Real-Time simulation of a complete PMSM drive at 10 ms time step", International Power Electronics Conference, IPEC'05, April 2005.

(Ismail T.B., 1994) Ismail T.B., Abid M., O'Brien K., Jerraya A. «An approach for hardware-software codesign», *Rapid System Prototyping*, 1994, p. 73-80.

(ITRS, 2003) International Technology Roadmap for Semiconductor Design, 2003 disponible en ligne à http://public.itrs.net/

(ISP1362, 2002) ISP1362 Embedded Programming Guide Version 9 June 2002.

(Jie L., 2004) Jie L., Eker J., Janneck J.W, Xiaojun L., Lee E.A., « Actor-oriented control system design: a responsible framework perspective » IEEE Transactions on Control Systems Technology, Volume 12, Issue 2, March 2004, p. 250 – 262.

(Jie Z., 2007) Jie Zhou, Hanchuan Peng, Ching Y Suen , "Data Driven Decomposition for Multi-class Classification", published on Pattern Recognition, 2007.

(Jiong Z., 2008) Jiong Zang, Jie Yuan, Fei Shi, Si-dan Du "A Fingerprint Matching Algorithm of Minutia Based on Local Characteristic", ISBN 978-0-7695-3304-9/08, IEEE 2008.

(Kajtazovic S., 2005) Kajtazovic S., Steger C., Pistauer M., « A HDL-independent modeling methodology for heterogeneous system designs », IEEE Behavioral Modeling and Simulation Workshop, 22-23 Sept. 2005, p. 88 – 93.

(Karimi S ., 2009) Karimi Shahrem, "Continuité de service des convertisseurs triphasés de puissance et prototypage "FPGA in the loop":application au filtre actif parallèle", Rapport de thèse, Janvier 2009.

(Katrib J., 2008) Katrib J., P. Poure, S. Karimi, S. Saadate, "Design methodology of VLSI power electronics digital controller based on Matlab-

Modelsim co-simulation", IEEE, International Symposium on Industrial Electronics, pp. 1751-1756, June-July 2008.

(Kazmierski T. J., 1992) Kazmierski T. J., Brown A. D., Nicolas K. G., Zwolinski M., « A General Purpose Network Solving System », IFIP Trans. AI VLASI-91, 1992, p. 147-156.

(Kudlugi M., 2001) Kudlugi M., S. Hassoun, C. Selvidge, D. Pryor - «A Transaction-Based Unified Simulation/Emulation Architecture for Functional Verification» - IEEE Transactions on Design Automation Conference (DAC), 2001, pages 623 – 628.

(Langeanu D., 2001) Langeanu D. et al. « Distributed event-driven simulation of VHDL-SPICE mixed-signal circuits » Int. Conference on Computer Design ICCD, 2001, p 302-307.

(Lardière C., 2004) Lardière Cédric - «Système d'émulation et d'accélération : l'habit ne fait plus le moine» - Electronique Internationnal, N576 – 28 Octobre 2004, pages 27.

(Lienhardt A.M., 2006) Lienhardt A.M, G. Gateau, T.A. Meynard, "Cosimulation tool for FPGAbased algorithm validation" IEEE International Conference on Industrial Technology, Dec. 2006.

(Long D.I., 1997) Long D.I., « Behavioural modelling of mixed-signal circuits using PWL waveforms », IEE Colloquium on Mixed-Signal AHDL/VHDL Modelling and Synthesis, 1997.

(Matlab/Simulink, 2012) Matlab/Simulink, disponible à www.mathworks.com

(Mathmodelica, 2006) Mathmodelica, disponible à http://www.mathcore.com/products/mathmo-delica/

(Mentor, 2012) Mentor http://www.mentor.com/products/fv/emulation/ , 2012.

(Modelica, 1997) Modelica - A unified object-oriented language for physical systems modeling, specifications report, September 1997, version 1.0, disponible à www.modelica.org.

(Newton A.R., 1978) Newton A. R., Pederson D. O., «A Simulation Program with Large-Scale Integrated Circuit Empahsis», IEEE International Symposium on Circuit and Systems, New York, 1978, p. 1-4.

(Nicolescu G., 2002) Nicolescu G. et al, « Desiderata pour la spécification et la conception des systèmes électroniques », Journal Technique et Science Informatiques, 2002, volume 21- n 3.

(Odryna P.,1986) Odryna P., Nazareth K., Christensen C., « A Workstation-Based Mixed Mode Circuit Simulator », 23 rd IEEE/ACM DAC, 1986, p. 186-192.

(Omer S., 2009) Omer Saeed, Atif Bin Mansoor, and M Asif Afzal Butt "A Novel Contourlet Based Online Fingerprint Identification",BioID MultiComm2009, Springer-Verlag, LNCS 5707, 2009, pp. 308–317.

(Pabst D., 1995) Pabst D., « HDL-A VHDL-Based Analog and Mixed signal Model Description Language », Tutorial TI of Simulation Congress EUROSIM'95, 1995.

(Patel D.H., 2004) Patel D.H, Shukla S. K., SystemC Kernel – Extensions for heterogeneous System Modeling, Kluwer Academic Publishers, Boston, 2004

(Pêcheux F., 2005) Pêcheux F., Lallement C., Vachoux A., « VHDL-AMS and Verilog-AMS as Alternative Hardware Description Languages for Efficient Modeling of Multidiscipline Systems », IEEE transactions on computer-aided design of integrated circuits and systems, vol. 24, no. 2, 2005.

(Pichon F.,1995) Pichon F. et al. « Mixed-Signal Modeling in VHDL for System-on-Chip Applications », European Design and Test Conference, 1995. ED&TC, 1995, p. 218 – 222

(Pinki M., 2003) Pinki M., Francis M., Chandrasekhar V., Austin A., Mantooth, H.A, « Achieving language independence with Paragon », International Workshop on Behavioral Modeling and Simulation, 2003, p. 149-153.

(Riihimaki J., 2005) Riihimaki J., Kukkala P., Kangas T., Hannikainen M., Hamalainen T.D., « Interfacing UML 2.0 for Multiprocessor System-on-Chip Design Flow », International Symposium on System-on- Chip, 2005, p. 108-111.

(Rizatti L., 2003) Rizzatti Lauro - «Choosing an emulation tool», 2003 http://www.eetimes.fr/bus/news/showArticle.jhtml?articleID=171202288

(Rodriguez J.J., 2007) Rodriguez-Andina J.J., M. J. Moure, M. D. Valdes, "Features, design tools, and application domains of FPGAs", IEEE Transactions on Industrial Electronics, vol. 54, no. 4, pp. 1810-1823, Aug. 2007.

(Sakallah K.A., 1985) Sakallah K. A., Director S. W., « SAMSON2: An event driven VLSI circuit simulator », IEEE Trans. CAD 4, 1985, p. 668-684.

(Salem A, 2003) Salem Ashraf, "Formal Semantics of Synchronous SystemC", Design, Automation and Test in Europe Conference and Exhibition, 2003, 376-381.

(Sameh A.H, 1971) A.H. Sameh. On Jacobi and Jacobi-Like Algorithms for a Parallel Computer. Mathematics of Computation, 25(579–590), 1971.

(Schorcht G., 2003) Schorcht G. et al., « System-level simulation modeling with MLDesigner, Modeling, Analysis and Simulation of Computer Telecommunications Systems », MASCOTS'03, 11th IEEE/ACM International Symposium, 2003, p 207 – 212.

(Senturia S., 1998) Senturia S.D., « CAD challenges for microsensors, microactuators, and Microsystems », Proceedings of the IEEE, Volume 86, Issue 8, Aug. 1998, p. 1611 – 1626.

(Simplorer, 2012) Simplorer, disponible à www.ansoft.com/products/em/simplorer/, 2012

(SJÖSTEDT C., 2009) SJÖSTEDT CARL-JOHAN, « Modeling and Simulation of Physical Systems in a Mechatronic Context », rapport de thèse, 2009.

(Smash, 2012) Smash, disponible à www.dolphin.fr/medal/smash/smash_overview.html, 2012.

(Soha H., 2005) Soha Hassoun, Murali Kudlugi, Duaine Pryor, and Charles Selvidge "A Transaction-Based Unified Architecture for Simulation and Emulation" IEEE transactions on very large scale integration (VLSI) systems, vol. 13, no 2, 2005.

(SPICE, 2012) http://bwrcs.eecs.berkeley.edu/Classes/IcBook/SPICE/

(Std VHDL-AMS, 1999) Std VHDL-AMS, IEEE Standard VHDL Analog and Mixed-Signal Extensions, IEEE Std 1076.1- 1999, 23 Dec. 1999

(Lukai, 2003) Lukai C. and Gajski D., "Transaction Level Modeling: An Overview", International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS'03, October 1–3, 2003.

(Vachoux A., 2003) Vachoux A. Grimm C. Einwich K., « SystemC-AMS requirements, design objectives and rationale », Design, Automation and Test in Europe Conference and Exhibition, 2003, p. 388 – 393.

(Valderrama C.A., 1995) Valderrama C.A., Changuel, A., Raghavan P.V., Abid, M., Ben Ismail T., Jerraya, A.A., « A unified model for co-simulation and co-synthesis of mixed hardware/software systems », *European Design and Test Conference*, 1995, p.180-184.

(Vladimeros V., 2012) Vladimeros Vladimerou, Pavithra Prabhakar, Mahesh Viswanathan, and Geir Dullerud, "Verification of Bounded Discrete Horizon Hybrid Automata », IEEE Transactions On Automatic Control, Vol. 57, No. 6, June 2012.

(Ying Hao) Ying HAO, Tieniu TAN, Yunhong WANG "AN EFFECITVE ALGORITHM FOR FINGERPRINT MATCHING" National Lab of Pattern Recognition, CAS, Institute of Automation, Beijing, P. R. China, 100080.

(Zorzi M., 2003) Zorzi M., Franze F., Speciale N., « Construction of VHDL-AMS simulator in Matlab » International Workshop on Behavioral Modeling and Simulation, 7-8 Oct. 2003, p. 113 – 117.

PUBLICATIONS

▪ Journaux internationaux

- **Mossaad Ben Ayed**, Faouzi Bouchhima, Mohamed Abid, "A Fast Simulation Emulation Engine", Informacije MIDEM - Journal of Microelectronics, Electronic Components and Materials. Vol. 43, No. 3(2013), 162 – 172. IF=0.27 indexed by ISI Thomson.
- **Mossaad Ben Ayed**, Faouzi Bouchhima, Mohamed Abid, "Automated Fingerprint Recognition Using the DECOC Classifier", International Journal of Computer Information Systems and Industrial Management Applications. ISSN 2150-7988 Volume 4 (2012) pp. 546-553. "Articles in IJCISIM are indexed or abstracted in: INSPEC Database".
- **Mossaad Ben Ayed**, Faouzi Bouchhima, Mohamed Abid, "A novel verification technique for control units", The International Journal of Engineering and Technology (IJET) Volume 5 N°2 (2013) pp. 1990-1999. Indexed by Scopus.

▪ Conférences internationales

- **Mossaad Ben Ayed**, Faouzi Bouchhima, Mohamed Abid, "A Novel Application of the Classifier DECOC Based on Fingerprint Identification", Interactive Multimodal Pattern Recognition in Embedded Systems IMPRESS 2010 Workshop on Database and Expert Systems Applications DEXA 2010. 1 September 2010 University of Deusto, Spain. "All accepted conference papers will be published in IEEE Xplore Digital library". (Classe B)

- **Mossaad Ben Ayed**, Faouzi Bouchhima, Mohamed Abid, "Classifier DECOC based Minutiae Extraction", 10th International conference on Sciences and Techniques of Automatic control & computer engineering STA 2009.
- **Mossaad Ben Ayed, Faouzi Bouchhima, Mohamed Abid,** "A Fast Hardware/Software Co-Verification Method using a real hardware acceleration", the

24th International Conference of Microelectronics ICM2012. IEEE conference and indexed by Scopus.

- **Mossaad Ben Ayed**, Faouzi Bouchhima, Mohamed Abid, "A Hardware Software In the Loop architecture for control units", the International Conference on Control, Engineering & Information Technology (CEIT'13).

République Tunisienne
Ministère de l'Enseignement Supérieur, de
la Recherche Scientifique
et de la Technologie

Université de Sfax
École Nationale d'Ingénieurs de Sfax

Ecole Doctorale
Sciences et Technologies

Thèse de *DOCTORAT*
ISI

N° d'ordre: 2013– 297

Environnement de Co-Simulation / Emulation des systèmes Continus/Discrets

Mossaad BEN AYED

الخلاصة: في السنوات الأخيرة، نمت وتطورت العديد من النظم و خاصة النظم ذات الخصائص الغير متجانسة. أدى هذا التطور إلى ظهور حاجة ماسة لبرامج قادرة على التصميم والتحقق من أداء هذه النظم بشكل عام.
موضوع هذه الأطروحة يرتكز على تعريف وتنفيذ نموذج co-simulation/émulation لمحاكاة دقيقة للأنظمة ذات الخاصية المستمرة / منفصلة على أساس برنامج الإختبار CODIS. في الجزء الأول، نقترح نموذج محاكاة للنظم ذات الخصائص المنفصلة في مجال الميكروإلكترونيك. في الجزء الثاني، نبرز تقنية محاكاة جديدة " Hardware Software In the Loop" لأنظمة التحكم. وأخيرا، يتم اقتراح و التحقق من صحة نموذج co-simulation/émulation للنظم المستمرة / المنفصلة استنادا إلى نموذج CODIS.

Résumé : Dans ces dernières années, plusieurs intérêts sont orientés vers les systèmes continus/discrets. Ces systèmes ont crée un besoin pour des outils de CAO capables de modéliser et de vérifier leur fonctionnement global.
Le sujet de cette thèse porte sur la définition et la mise en œuvre d'un modèle de co-simulation/émulation pour une simulation précise de systèmes continus/discrets en se basant sur l'environnement CODIS. En première partie, un modèle de simulation/émulation matériel/logiciel est proposé. En deuxième partie, une nouvelle technique de simulation"Hardware Software In the Loop" pour les systèmes de contrôle est présentée. Finalement, un modèle de co-simulation/émulation continu/discret basé sur le modèle de l'environnement CODIS est validé.

Abstract: In the last years, several interests have been oriented to the heterogeneous systems. Among these systems, the continuous/discrete systems received an attention in the Microsystems, the analog/digital systems and the control systems. These systems created a need for CAD tools, able to validate their global behavioural.
The main goal of thesis is to define and to implement a co-simulation / emulation model for an accurate simulation of continuous/discrete systems. The first step, presents a simulation/emulation environment for Hardawre/Software design. Then, a new simulation technique titled Hardware Software In the Loop for designing and verifying the controller unit is described. Finally, an environment based on CODIS tool and the two described environments is presented.

الكلمات المفاتيح: نموذج محاكاة, نموذج تزامن, CODIS, Simulink, SystemC

Mots clés: Modèles de simulation, modèles de synchronisation, SystemC, Simulink, CODIS
Key-words: Simulation models, synchronization models, SystemC, Simulink, CODIS