UNIVERSITE SAAD DAHLEB DE BLIDA

Faculté des sciences

Département d'informatique

MEMOIRE DE MAGISTER

Option: Ingénierie des systèmes et de la connaissance

PROPOSITION D'UNE APPROCHE D'EVALUATION DES PERFORMANCES TEMPORELLES EN CODESIGN À PARTIR DE COSPECIFICATIONS DE HAUT NIVEAU

Par

BOUGHERARA Maamar

Devant le jury composé de:

Mme H. ABED	Maître de Conférences Université de Blida	Présidente
M. M. KOUDIL	Maître de Conférences INI, Alger	Rapporteur
Melle K. BENATCHBA	Maître de Conférences INI, Alger	Examinatrice
Mme S. OUKID	Maître de Conférences Université de Blida	Examinatrice

Blida 2008

A la mémoire de la lumière des mes yeux à qui aucun mot ne suffit pour dire merci

A mon père

A tous ceux qui m'on dit: "quand termines-tu ton travail ?"

REMERCIEMENTS

Je tiens à remercier en premier lieu mon dieu « *ALLAH* » tout puissant qui m'a accordé la volonté et le courage pour la réalisation de ce projet.

Je tiens à remercier chaleureusement monsieur KOUDIL Mouloud, Maître de conférences à l'Institut National d'Informatique d'Alger, mon promoteur et le directeur de ce travail, pour son expérience dans le domaine, sa gentillesse et son amabilité, ses conseils et sa disponibilité, et pour avoir accepté de diriger ce travail et de corriger ce document.

A tous ceux qui m'ont aidé à finir ce travail de prés ou du loin

Merci a tous

الملـخـــــــــــــص

عملية إنتاج الأنظمة في أيامنا هذه تتطور يوم بعد يوم, الآن نحن بصـدد إنتاج الأنظمة المختـلطة التي تحتوي على جزء مخصص للنظام البرمجي و الأخر مخصص للنظام المحتوى في الشريحة الالكترونية, و تطوير هذا النوع من الأنظمة في تقـدم مستمر.

CODESIGN.يتخصص بحثـنا في هذه المجال, الطريقة المتبعة في إنتاج هذا النوع من الأنظمة هي

من أهم المراحل المتبعة في هذه الطريقة و هي مرحلة الفصل بين العناصر المبرمجة و العناصر التي تنتج في الشرائح الالكترونية.

performance, ينبغي لهذه المرحلة مراعاة بعض الشروط المهمة و الأساسية في عملية الفصل و التي تسمى ب قبل البدء بـمرحلة الفصل. performance يجب علينا بعملية قياس ل

وقت التنفيذ, المساحة, الاتصالات و القوة. performance:من أهم عناصر أل

واحدة من هذه العناصر المهمة هي مجال بحتنا في هذه الرسالة و هو عنصر الوقت, يجب علينا تقدير الوقت قبل بدء عملية الفصل.

نتاج بحثنا المتمثل في هذه المذكرة التي نعرض من حلالها طريقتين ل تقدير وقت التنفيذ قبل بداية عملية الفصل.

الطريقة الأولى المتمثلة في تقدير وقت التنفيذ المخصص للجزء المبرمج,و الثانية التي تشمل وقت التنفيذ للجزء المحتوى في الشريحة الالكترونية.

اقتراحات و حلول قدمناها في هذه المذكرة للحصول على أفضل النتائج.

performance.**الكلمات الرئيسية**: وقت التنفيذ, الأنظمة المختـلطة,الشريحة الالكترونية, النظام البرمجي,

RESUME

La conception des systèmes s'améliore de jour en jour. On en est maintenant à la conception de systèmes dits mixtes (qui englobent des parties matérielles et logicielles). Cette méthodologie de conception des systèmes mixtes s'appelle CODESIGN.

Parmi les étapes essentielles du codesign, la plus importante est l'étape du partitionnement qui choisit les éléments à implémenter en matériel et ceux qui doivent être exécutés par un processeur logiciel. Cette étape nécessite une prise en considération de certaines caractéristiques importantes. Il est, en particulier, nécessaire de recourir à une évaluation des performances avant de faire un partitionnement. En général, ces caractéristiques sont: le temps, l'espace, la communication et la puissance.

Ce travail aborde l'élément le plus important qui est le temps d'exécution. Ce temps doit être estimé avant de commencer le partitionnement prenant en considération les caractéristiques de l'architecture cible.

Dans ce mémoire, deux approches automatiques pour l'estimation du temps d'exécution à partir d'une spécification du système en haut langage de niveau sont présentées. La première approche se base sur le temps d'exécution pour le logiciel et la deuxième sur le temps d'exécution en matériel. Des propositions et solution sont présentées pour permettre un partitionnement de meilleure qualité.

Mots clés: Codesign, système mixte, hardware, software, performance, temps d'exécution.

ABSTRACT

System design is getting more efficient every day. Designer are today able to develop mixed systems (including hardware and software parts). The design methodology for such mixed systems is called “codesign”.

One of the most essential steps in the codesign process is partitioning whose objective is to select the element to be implemented in hardware and the ones to be mapped on software parts.

This step requires to take into account some important elements as performance and cost. It is, in particular, to perform an evaluation of system performance before making any partitioning. In general, the performance elements to be evaluated are: time, space, communication and power.

In this work, we focused on the most important performance characteristic which is the execution time. This latter must be estimated before starting any partitioning, considering the target architecture parameters.

In this document, two automatic approaches have been introduced. They aim at estimating the execution time of the system high level specification. The first one is based on software execution time, while the second one deals with hardware execution time.

Key words: Codesign, analog and digital system, hardware, software, performance, time execution.

TABLE DES MATIERES

LISTE DES ILLUSTRATIONS GRAPHIQUES

LISTE DES TABLEAUX

INTRODUCTION

Les techniques classiques de conception de systèmes imposent de séparer le hardware du software très tôt dans le cycle de conception. Les chemins suivis par les concepteurs des différentes parties restent indépendants, n'autorisant l'interaction que lors de la phase d'intégration. Ainsi, les concepteurs du hardware spécifient leurs produits sans totalement apprécier les besoins en ressources matérielles des logiciels qui tourneront sur leurs produits.

De leur côté, les concepteurs de logiciels ne tirent pas suffisamment profit de ressources matérielles de la plate-forme hardware sur laquelle tourneront leurs produits. Finalement, la phase d'intégration des deux parties nécessite souvent des modifications du logiciel et/ou du matériel.

Depuis quelques années, les ordinateurs à architectures reconfigurables ont fait leur apparition. Ces machines permettent une répartition de l'exécution du programme sur des plates-formes étroitement couplées. Ces systèmes offrent des possibilités substantielles d'accélération

Lors d'un partitionnement entre le hardware et le software, il faut une bonne analyse des différentes performances. L'analyse des performances est une activité complexe. En effet, elle permet de déterminer un certain nombre de paramètres qui caractérisent les différentes parties du système, dans le but de guider l'ordonnancement et le partitionnement. Ces paramètres renseignent le concepteur sur la qualité des solutions trouvées, afin de prédire les résultats de la conception sans aller jusqu'à la réalisation totale. Il est donc important de disposer d'outils d'analyse/estimation rapides et de choisir des paramètres suffisamment significatifs, afin de pouvoir comparer, avec suffisamment de précision, la qualité des solutions obtenues et donc de parcourir l'espace des solutions à la recherche de la meilleure.

Dans la littérature, de nombreux paramètres sont cités, parmi lesquels figurent: la flexibilité, le type d'application, la concurrence, la testabilité, la sécurité, la réutilisabilité, les contraintes de sûreté de fonctionnement... Cependant, ces contraintes sont difficiles à quantifier, et aucun des travaux rencontrés n'y fait référence dans une quelconque fonction d'évaluation.

Par contre, les paramètres les plus souvent utilisés peuvent être regroupés en quatre catégories qui sont: les paramètres temporels, les paramètres d'espace, les paramètres de communication et la dissipation en puissance. Les paramètres temporels englobent

essentiellement: le nombre de cycles d'horloge, le temps d'exécution du logiciel et du matériel, ou des contraintes de débit associées aux variables pour les systèmes temps réel, par exemple. Les paramètres spatiaux peuvent être divisés en espace matériel et espace logiciel. En ce qui concerne le premier, la capacité joue un rôle important pour les composants matériels (exemple: ASICs, FPGA...) où l'espace est une ressource plus rare et par conséquent plus coûteuse que sur les processeurs logiciels où le stockage se fait dans des mémoires. Elle peut être exprimée en termes d'espace (surface de silicium, nombre de portes ou de transistors...) et de coût associé au circuit concerné car ce dernier varie en fonction de la nature du circuit. Le nombre de broches d'entrées/sorties est également une ressource critique qui doit être prise en compte. En ce qui concerne le logiciel, il s'agit en général du nombre de mots occupés par les données et le code sur les différentes mémoires utilisées sur l'architecture cible (RAM, mémoire cache, registres...).

La communication peut être caractérisée par deux types de paramètres: les paramètres temporels et le volume des transferts. Les premiers décrivent essentiellement le temps de communication et peuvent permettre de détecter d'éventuels goulots d'étranglement. Le volume des transferts est étroitement lié aux données transférées à travers les interfaces matériel/logiciel et à la bande passante des bus de l'architecture cible.

La dissipation en puissance est un autre paramètre important permettant, par exemple, d'évaluer les temps d'autonomie de systèmes alimentés par batteries.

Dans ce mémoire, nous nous intéressons à l'un des paramètres les plus important qui est le temps. Il est indispensable d’avoir une estimation du temps d'exécution du logiciel et du matériel. Le temps d'exécution du pire cas d'un programme peut être estimé en mesurant son temps d'exécution pour un cas de test. Il peut l’être également en utilisant des méthodes d'analyse statique de son code source. Notre approche est basée sur l'utilisation d'une méthode d'analyse statique.

Cette approche a l'avantage d'être sûre, mais cette sûreté des estimations a une contrepartie : les estimations obtenues sont pessimistes. Ce pessimisme induit une surestimation des moyens matériels nécessaires au fonctionnement du système.

Les propositions de ce mémoire portent sur la réduction du pessimisme des méthodes d'analyse statique d'estimation du temps d'exécution du pire cas. Nous proposons d'une part une amélioration de la précision de l'identification du pire chemin d'exécution par l'utilisation d'une nouvelle technique d'annotations des boucles: les annotations symboliques. D'autre part,

nous proposons l'amélioration des techniques de prise en compte de l'architecture matérielle existantes, ainsi qu'une technique originale de prise en compte du mécanisme de prédiction des branchements.

Enfin, notre dernière proposition est l'implantation des propositions précédentes dans un prototype d'analyseur statique de WCET ayant une structure modulaire. Le but d'une telle structure est l'amélioration des possibilités d'adaptation de l'analyseur à de nouvelles architectures matérielles.

L'organisation du document est la suivante.

Le chapitre 1 présente le cadre et la méthode de conception des systèmes mixtes. Le deuxième chapitre est consacré aux différentes étapes du processus de codesign. Le 3ème chapitre présente l'analyse des performances. Le 4ème chapitre présente le temps d'exécution en particulier pour le logiciel. Le 5eme chapitre se base sur le hardware et les différent étapes pour concevoir les circuit à l'aide d'un langage hardware.Le 6ème chapitre présente l'approche proposée pour analyser le temps d'exécution du pire cas pour le logiciel. Dans le chapitre 7, une nouvelle approche est proposée pour estimer le temps d'exécution dans le hardware. Le dernier chapitre présente la mise en œuvre de l'outil que nous avons développé pour les deux approches introduites ci-dessus.

Enfin, nous concluons ce travail en mettant en avant les apports de notre travail et les perspectives ouvertes par ce dernier.

CHAPITRE 1

CODESIGN MATERIEL/LOGICIEL

1.1.Introduction

La conception des systèmes contenant des parties matérielles et logicielles n'est pas un problème nouveau. Le développement des systèmes informatiques a été caractérisé par le fait que les ingénieurs matériels fournissaient des systèmes informatiques à usage général, programmés ensuite par les ingénieurs logiciel.

Même si l'utilisation des microprocesseurs durant les années 80 a permis la mise en œuvre d'applications à dominante logicielle plutôt que matérielle [9], les idées n'ont pas beaucoup changé. Les mises en œuvre du logiciel et du matériel sont deux activités qui ont toujours été menées de manière relativement indépendante. Et donc, les spécialistes du logiciel peuvent penser qu'ils n'ont aucun besoin de se préoccuper des détails de bas niveau du matériel des calculateurs. Cette approche présente peu de flexibilité dans l'évaluation des différentes options de conception et de répartition du matériel et du logiciel.

Avec des méthodologies de conception différentes pour le matériel et le logiciel, il est également difficile d'optimiser le produit à concevoir en tant que système global. Une telle approche est inadéquate, spécialement lors du processus de conception de systèmes digitaux requérant des performances strictes et un temps de cycle de conception réduit. Les systèmes mixtes sont traités depuis longtemps et souvent d'une unique manière: dès les différents besoins identifiés, on sépare ce qui deviendra la partie logicielle et la partie matérielle du système. Cette séparation était nécessaire car les compétences à mettre en oeuvre étaient différentes (informatique, électronique numérique et analogique). Ainsi une équipe "logiciel" s'occupera de la première des parties tandis qu'une équipe "matériel" concevra l'équipement électronique sur lequel, en fin du processus de production, le logiciel sera intégré.

Cette méthode présente de nombreux problèmes dont voici les plus importants [11]:

– Les spécialistes du logiciel connaissent généralement mal les fonctionnalités proposées par le matériel, l'exploite donc mal et recrée même parfois ce qui existe déjà.

– Les spécialistes du matériel apprécient mal les besoins des gens du logiciel ce qui conduit souvent à un mauvais dimensionnement de cette partie du système.

– De la phase d'intégration des différentes parties du système peuvent émerger des problèmes qui nécessitent parfois d'importantes modifications d'une ou l'autre des parties.

– Un ajustement ou un changement des spécifications tardif peut remettre en question une des parties du système.

1.2.Le Codesign

1.2.1.Définition du terme « Codesign »

Le terme "Hardware/Software Concurrent Design" souvent abrégé par "Hw/Sw Codesign" et qui se traduit par conception conjointe matériel et logiciel représente un processus de conception complet basé sur la trilogie: modèles, méthodes et outils ESDA (Electronic System Design Automation). Ce processus doit permettre aux concepteurs de transformer correctement du premier coup les spécifications d'un système en un produit industriel comportant une partie logicielle et une partie matérielle et satisfaisant les contraintes fonctionnelles et non fonctionnelles de son cahier des charges. Il doit également permettre d'accroître la qualité de conception et de réduire le temps de développement.

Dans la trilogie, un modèle est une représentation formelle d'un système à un niveau d'abstraction donné. En plus une méthode efficace doit reposer sur un ensemble de concepts de modélisation restreint mais suffisant pour décrire n'importe quel système. Le terme méthode représente une procédure ou démarche bien définie et structurée permettant de résoudre un problème. Les outils ESDA se composent d'outils de capture de modèles, de simulation, d'analyse statique ou dynamique, de recherche de compromis, de synthèse et de co-vérification. Les spécifications non fonctionnelles représentent tout ce qui dans le cahier des charges n'est pas une exigence fonctionnelle. Il s'agit habituellement d'un ensemble de contraintes d'intégration avec l'environnement (taille, puissance consommée), de performances (temps de réponse, débit), d'ordre économique (coût, délai de fabrication), de qualité (durée de vie), de sûreté de fonctionnement, etc.

Gajski Donne la définition suivante [14]: "Codesign is defined as amethodology and technique for designing software and hardware concurrently, thus reducing the design time and time to market". Dans cette définition, le terme le plus important est "concurrently" qui

signifie que le développement simultané de la partie logicielle et de la partie matérielle du système s'effectue avec une interaction forte et permanente entre les deux parties.

La définition de [13]: Le terme de Codesign est apparu pour définir une méthodologie de conception des systèmes mixtes: matériels (ASIC'S incluant les composants logiques programmables) et logiciels (algorithmes programmes sur microprocesseurs).

Cette approche diffère fondamentalement du cycle de conception conventionnel des systèmes qui repose sur un développement séparé des deux parties.

1.3. Avantages du Codesign

Le Codesign hardware/software pose une question très importante: comment peut-on intégrer deux disciplines très différentes, non seulement dans leurs techniques, mais aussi dans leurs culture ? L'intégration du software et du hardware dans un seul système intégral constitue un problème non trivial. Les bénéfices immédiats sont très variés dans ce domaine de recherche.

Différents avantages des méthodes co-design par rapport aux méthodes traditionnelles ont été identifiés par [10] et [12]:

1.3.1.Flexibilité de la conception

Durant la conception d'un système, la spécification est souvent modifiée à cause des retours Arrières ou du changement des besoins de l'utilisateur. Il est donc important d'être capable de changer la conception en conséquence sans avoir à refaire tout le travail déjà accompli.

Le Codesign permet de spécifier plusieurs versions d'un même système en fonction des coût, performance, rapidité, technologie, etc. La différence entre ces versions est minime et peut être par exemple le déplacement d'une portion de code du hardware vers le software.

1.3.2. Rapidité d'accès au marché

Dans le domaine commercial, un produit se vend plus facilement s'il a peu de concurrents. Le Codesign permet de produire des systèmes mixtes plus rapidement; par conséquent, ils sont conçus dans des délais réduits permettant une mise sur le marché rapide.

1.3.3.Exploration globale efficace de l'espace de conception

Afin de bien concevoir un système complexe, il est nécessaire de le diviser en sous systèmes. L'une des caractéristiques du Codesign est qu'un sous système, malgré qu'il soit

considéré comme étant une entité individuelle, est ordonnancé d'une telle manière que les décisions d'un sous système influent sur le fonctionnement d'un autre.

Afin de permettre l'exploration de tout l'espace de la conception, il est préférable de retarder la division du système global le plus tard possible dans le cycle de conception; le Codesign permet de naviguer à travers les frontières des sous systèmes très aisément.

1.3.4.Phase de test et intégration optimisé

Afin de garder une vue cohérente du système durant les phases de conception et de mise en œuvre, le Codesign élimine toute mauvaise surprise lors de l'intégration des sous système et le test du système mixte global. Ceci réduit non seulement la durée de ces phases mais aussi le risque d'avoir des révisions majeures dans la conception. Les techniques de cosimulation et covérification permettent d'être utilisées pour réduire le besoin considérable d'un système en matières de test et d'intégration.

1.3.5.Performance de conception

En facilitant l'exploration de l'espace de la conception, le Codesign apporte un outil d'analyse systématique des produits en un temps raisonnable. Les informations de la conception d'un sous système peuvent être obtenues et utilisées pour éviter des optimisations locales non nécessaire pour un autre sous système. Le résultat est l'obtention d'un produit à grande performance et à coût faible.

1.3.6. Prototypage rapide

On utilise des prototypes afin de prédire les propriétés d'un système avant sa production. Un prototype est un système qui se rapproche le plus du système réel dans les aspects servant à l'évaluation sans recourir à une mise en œuvre globale. Un sous système de prototype fonctionne de la même manière que le sous système lui correspondant en réalité.

En utilisant le Codesign, le développement des prototypes est simplifié grâce à sa large vision des systèmes, et la possibilité qu'il offre de produire rapidement plusieurs versions d'une même conception.

1.4.Domaines d'application

Le Codesign ne s'applique pas aux applications à usage général (logiciels commerciaux), mais s'applique surtout aux applications complexes où le hardware et le software travaillent étroitement ensemble pour résoudre des problèmes ayant des contraintes temporelles très aiguës, là où la rapidité d'exécution est primordiale. Il s'agit en fait

d'applications où le Codesign s'avère la meilleure méthode de conception afin d'atteindre les meilleures performances avec des coûts moindres. Parmi ces domaines, citons [15], [16]:

1.4.1.Les télécommunications

C'est le marché le plus prospère. Elle constitue un domaine d'application très intéressant pour le Codesign car il apporte des produits performants et à moindre coût, et ceci doit être fait dans des délais pour être présentés dans le marché. Ces produits sont constamment mis à jour et présentés dans des versions différentes. Ce sont des caractéristiques très importantes pour la fabrication de produits telles que les modems/fax, les téléphones mobiles, etc.

1.4.2.Les systèmes embarqués

Un système embarqué est un sous système informatique d'un système global tel qu'un ordinateur de bord ou un système électroménager numérique. Ils sont produits intensément dans des espaces d'intégrations très réduits et opèrent, fréquemment, dans un environnement temps réel. Ils peuvent aller du contrôleur de robot au système de pilotage d'avion ou navette spatiale en passant par les applications médicales, telles que stimulateurs cardiaques et les prothèses intelligentes.

1.4.3.Les systèmes de contrôle

Les systèmes informatiques sont fréquemment utilisés pour contrôler d'autres systèmes, par exemple les processus industriels. Un système de contrôle consiste normalement en une partie hardware qui interagit avec l'environnement, et une partie software qui prend des décisions.

Un tel système exige une flexibilité de conception (afin d'adapter des changements dans l'environnement de contrôle fréquemment pendant l'exécution) et des propriétés temps réel (afin de garantir une rapidité de réponse aux événements).

1.4.4.Le traitement de signaux

Les systèmes dans ce domaine exécutent des opérations de calcul très complexes dans des conditions temporelles très strictes. Dans ce domaine, les circuits hardware sont utilisés pour les calculs afin d'accélérer la vitesse d'exécution, alors que les entités software (code exécutable) sont utilisées pour diriger ces circuits

1.5.Architectures pour le Codesign

Une architecture indique comment le système sera réellement mis en œuvre. Une architecture est constituée d'un ensemble de composants, d'une organisation et d'un modèle d'interaction entre les différents composants [17]. Le but de la conception d'architecture est de décrire pour un système donné le nombre de composants, le type de chaque composant, et le type de connexion entre ces divers composants. Les composants sont les éléments de base pour réaliser une application donnée. Dans le cas des architectures mixtes logicielles/matérielles on distinguera les composants logiciels, les composants matériels et les composants de communication.

Les architectures couvrent le large spectre des systèmes partant des simples contrôleurs jusqu'aux systèmes à processeurs parallèles hétérogènes. Généralement, certaines architectures sont plus efficaces pour l'implémentation de certains modèles ou pour supporter une gamme d'applications.

1.5.1.Organisation de l'architecture

L'organisation de l'architecture est définie par la composition ou l'assemblage d'éléments de base.

Dans le cas des architectures mixtes logiciel/matériel on s'intéressera aux rôles joués par le contrôle global du système et par les différents éléments de base. On distinguera deux types d'organisation:

1.5.1.1.L'architecture monoprocesseur et l'architecture multiprocesseur.

Une architecture monoprocesseur est composée d'un processeur maître et d'un ensemble de composants matériels spécifiques. Dans ce cas, un seul processeur est responsable de l'état global du système. Il réalise le contrôle au niveau global et les autres éléments agissent en tant qu'esclaves. Dans le second type d'organisation, l'architecture est composée d'un réseau de processeurs autonomes communicants. Les processeurs agissent de manière indépendante. L'interaction entre les différents composants constitue la communication. Cette communication permet l'échange des informations et du contrôle. Ces échanges peuvent être synchrones ou asynchrones. Ils peuvent être aussi distribués ou gérés par des contrôleurs spécifiques.

Une architecture peut être représentée par un diagramme de composants interconnectés (figure 1.1). Sa représentation contient un ensemble de nœuds et un ensemble d'arcs. Les nœuds représentent les composants. Ils sont des objets définis par leur type et par le nombre d'entrées/sorties (Unité Arithmétique et Logique, microprocesseur, mémoire, unité d'entrée/sortie, ASIC, FPGA ou même des sous systèmes complexes). Les arcs sont les différents types de connexions entre ces composants (bus, fils ou même modules de communication sophistiqués).

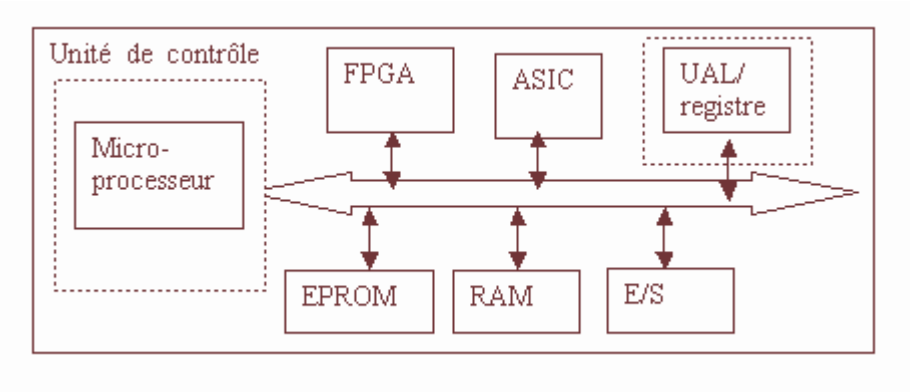

Figure.1.1: Exemple d'architecture

1.5.2.Composants de l'architecture

On appelle architecture mixte logicielle/matérielle une organisation qui combine à la fois des composants logiciels et des composants matériels. Les systèmes complexes existants comportent une multitude de composants. Les composants trouvés dans ce type de système sont les processeurs (composants logiciels), des ASIC et des FPGA (composants matériels).

1.5.2.1.Processeurs

Nous pouvons classer les processeurs en processeurs d'usage universel, processeurs parallèles et processeurs spécifiques [18].

Les processeurs d'usage universel, tels que le RISC (Reduced-Instruction-Set Computer), sont optimisés pour réaliser des cycles d'horloge courts, un nombre restreint de cycles par instruction, et une parallélisation efficace du flot d'instruction (pipelining). Ils offrent un degré maximum de flexibilité, ainsi qu'un temps de développement très réduit puisque aucune conception matérielle n'est nécessaire et que les outils de développement logiciel sont disponibles. Par contre le bas coût en surface, la basse consommation et surtout les performances ne sont pas assurés.

L'architecture à processeurs parallèles telles que les machines VLIW, SIMD ou MIMD exploite le parallélisme en utilisant de multiples unités fonctionnelles dans son chemin de données [18] [19]. Une instruction VLIW (Very-Long-Instruction-Word) contient une zone pour chaque unité fonctionnelle. Chaque zone d'une instruction VLIW indique l'adresse de la source et les opérandes destination, ainsi que l'opération à exécuter par l'unité fonctionnelle. Comme résultat, une instruction VLIW est généralement très large, puisqu'elle doit contenir approximativement une instruction standard pour chaque unité fonctionnelle.

Le processeur programmable spécifique, aussi appelé ASIP (Application-Specific Instruction-set Processor) est développé pour une application donnée tout en optimisant son architecture interne[18][20], son ensemble d'instructions et les programmes de son application. Les applications adaptées varient d'un simple algorithme à un vaste domaine comme celui du traitement du signal. L'approche consiste par la suite à réutiliser ce processeur avec son logiciel pour la conception de systèmes distribués complexes.

La réutilisation des composants diminue le temps de conception et le coût des systèmes.

1.5.2.2.Les ASICs

Les ASICs (Application-Specific Integrated Circuits) offrent les garanties de bas coût en surface, basse consommation et de performances. Par contre la flexibilité, le temps de développement et les possibilités de réutilisation sont ses points faibles. Les concepteurs utilisent plus les ASIC pour les parties d'une application bien maîtrisées, laissant le reste pour une implantation en logiciel, et ceci dans un but de flexibilité et de coût. Les circuits ASIC ont fait l'objet de beaucoup d'intérêt pendant la dernière décennie à cause du développement d'outils de synthèse, souvent appelés outils de synthèse de haut niveau. Ces outils ont été développés pour augmenter la productivité de conception des circuits en réduisant le temps de conception. Pourtant, cette approche de conception n'est pas toujours adoptée [21].

Certaines applications doivent être programmables. C'est pourquoi les constructeurs des microprocesseurs mettent à disposition des concepteurs le cœur de leurs processeurs pour être utilisés comme une fonction à incorporer dans le circuit. L'utilisation d'un cœur de processeur programmable avec un chemin de données sur commande offre plusieurs avantages. Il permet l'amélioration des performances, car les composants critiques sont réalisés par un chemin de données sur mesure. Par conséquent, la communication logiciel/matériel peut être plus rapide. On obtient ainsi, une réduction de la surface et de la consommation par l'intégration du logiciel et du matériel sur un même support. Ce type de

conception de circuit est attractif pour les applications embarquées, tels que les applications digitales pour le téléphone cellulaire. La conception de ce type de système nécessite le découpage de l'application en parties logicielles et matérielles, tout en explorant les différentes possibilités d'implantation.

1.5.2.3.Les FPGAs

Les FPGAs (Field Programmable Gate Arrays) est souvent utilisés à côté des composants cités plus haut [21]. Ils fournissent des circuits logiques et numériques avec une complexité moyenne de l'ordre de quelques milliers de portes environ et peuvent être utilisés pour des applications spécifiques dans un seul circuit intégré. La popularité des FPGAs peut s'expliquer par leur facilité d'utilisation.

Les FPGA peuvent servir pour le prototypage rapide, l'émulation matérielle et l'accélération de fonctions. Pour le prototypage rapide, un FPGA permet de transférer une nouvelle application sur silicium en quelques heures, alors que la réalisation des circuits intégrés (ASIC) nécessite des semaines voire des mois. Ceci est très utile pour l'évaluation de plusieurs options d'architecture pour une application donnée. Les FPGAs permettent l'émulation des applications pour lesquelles la simulation logique s'avère difficile à cause d'événements survenant en temps réel. Ainsi, un FPGA peut servir comme circuit de prototypage. Il peut aussi fournir une aide précieuse pour le développement d'algorithmes parallèles en supportant des configurations multiples.

1.5.3.La communication

Les architectures multiprocesseurs utilisent trois types de communication: point à point, par mémoire partagée (communication processeur/mémoire/processeur) et par passage de message dans un système à mémoire distribuée (communication processeur/processeur) [18] Les trois types de communication sont:

La communication point à point entre deux processeurs est le type de communication le plus simple. Le canal physique est consacré à l'interconnexion des deux processeurs. S'il n'y a aucune mémoire tampon (buffer) la communication nécessite un mécanisme de rendez-vous où les deux processeurs sont actifs et prêts à participer de la communication en même temps.

La communication par mémoire partagée utilise une mémoire globale comme moyen d'échange de donnés entre les processeurs. La figure montre un ensemble de processeurs connectés par des bus indépendants à une mémoire commune. L'accès à la mémoire doit être protégé par des arbitres.

Par exemple, on doit prévoir le cas où deux processeurs essayent d'écrire dans la même adresse de la mémoire en même temps. Pour éviter de telles situations de conflit on doit contrôler l'accès de la mémoire. Les systèmes à mémoire distribuée évitent les problèmes de chevauchement de données. Dans un système à mémoire distribuée, chaque processeur a accès seulement à sa propre mémoire D'autre part l'accès aux données dans une mémoire commune est possible par passage de messages entre les processeurs à travers le bus. Le bus relie plusieurs processeurs et il est employé pour partager le canal physique de communication.

1.5.4.Discussion des architectures

Il y a une grande variété d'architectures cible pour la conception conjointe logicielle/matérielle.

Cette variété est le résultat de l'optimisation manuelle d'architecture à une application simple ou à un domaine d'application sous de diverses contraintes et fonctions de coût. L'optimisation mène aux systèmes spécialisés. Plusieurs facteurs de coût et de performances dépendent directement du choix de l'architecture d'implémentation. L'utilisation d'une architecture spécifique, bien adaptée à une application déterminée, est plus efficace. Cependant, ce type d'architecture n'est pas assez flexible pour bien s'adapter dans un environnement de Codesign à d'autres types d'application.

1.6. Travaux utilisant le Codesign

Une approche typique de synthèse part d'un langage de spécification de niveau système pour réaliser une répartition des fonctions entre logiciels et matériels pour, ensuite, synthétiser la communication entre les blocs résultants (figure 2). L'étape suivante consiste en la génération de code qui est suivie de la compilation du logiciel et de la synthèse comportementale des parties matérielles [22] [23]. Le prototypage virtuel, qui est une étape intermédiaire entre la spécification au niveau système et la génération de l'architecture cible, établit un lien entre les outils du Codesign et les outils spécifiques pour la synthèse logicielle/matérielle.

La conception mixte fournit au concepteur des méthodes et outils pour explorer plusieurs possibilités de découpage du système et évaluer les performances des partitions [24]. L'analyse des compromis de mise en œuvre, tant au niveau matériel qu'au niveau logiciel, est très utile pour permettre la sélection de l'architecture du système qui s'accommode aux mieux

des contraintes diverses (coûts de développement, contraintes temps réel, fiabilité, etc.) [25]. Par ailleurs, la combinaison dans une même représentation des ensembles matériels et logiciels nécessite un modèle ou un langage permettant d'unifier les sémantiques associées à ces ensembles. Certaines méthodes de conception mixte utilisent un seul langage pour la description de tout le système. D'autres utilisent à la fois un langage pour le logiciel et un langage pour le matériel et essaient de les intégrer dans un même environnement [18] [26]. Ainsi, l'un des problèmes inhérents à la conception mixte matérielle/logicielle, et auquel il faut souvent faire face, est la modélisation de l'interaction entre plusieurs ensembles matériels et logiciels.

Le cycle de développement d'un système logiciel/matériel est présenté dans la figure 1.2. Le point de départ est toujours un cahier des charges qui correspond à une formulation des besoins. Les informations fournies concernent l'application dans son ensemble et expriment les objectifs souhaités.

Une fois le cahier des charges défini, le système à concevoir doit être décrit selon une vue purement externe. La spécification inclut toutes les contraintes que doit satisfaire ce système. La phase de conception générale exprime la structure du système sur le plan fonctionnel. L'organisation interne et le comportement de chacune des fonctions sont explicités. La phase de conception détaillée a pour objectif de trouver des schémas d'implantation pour le logiciel et les structures du matériel qui est nécessaires. Ensuite, la phase de réalisation décrit la solution finale en termes de logiciel et de matériel. Elle conduit à un système opérationnel.

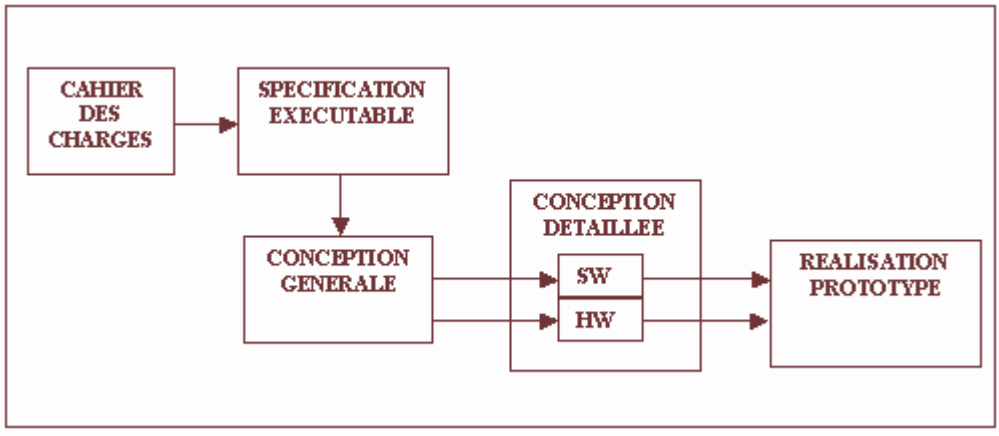

Figure 1.2 Etape du cycle de développement d'un système

Une approche de niveau système est souvent invoquée naturellement en tant que condition nécessaire à la maîtrise de la complexité. Dans le contexte des systèmes

électroniques complexes, ceci se traduit par une approche de développement simultané du matériel et du logiciel. Parmi les avantages d'une telle approche, on peut citer celui du compromis réaliste entre matériel et logiciel qui répond au mieux aux performances souhaitées, où celui d'une meilleure utilisation des possibilités de la technologie actuelle en ce qui concerne les répartitions matérielles/logicielles.

Plusieurs projets (Cosyma, SpecSyn, MCSE, Cosmos) sont en train d'être développés afin de créer des environnements de Codesign. Certaines approches proposent de partir d'une spécification système type (SDL, StateChart, Esterel). Ces langages offrent la possibilité de spécifier un système à un haut niveau d'abstraction où les parties logicielles et matérielles ne sont pas déterminées. A partir de ces spécifications, on procède à des étapes de découpage et d'allocations pour aboutir à un modèle abstrait composé de parties matérielles et logicielles communicant à travers des unités de communication. Les parties matérielles sont généralement spécifiées en VHDL, mais la description des parties logicielles varie d'un environnement à un autre. De plus, d'autres projets en cours se distinguent par des choix pragmatiques en limitant le flot de conception et/ou en limitant le domaine d'application. Parmi ces projets on peut citer RASSP [29] et CoWare [51]. La suite de cette section introduit quelques méthodologies de conception de systèmes mixtes logiciels/matériels.

1.6.1.RASSP

Le projet RASSP [25] propose une approche pragmatique qui consiste à spécifier les parties logicielles et matérielles à l'aide d'un langage de description matériel (VHDL). Cette méthodologie a pour cible la réalisation des applications spécifiques dédiées au traitement du signal. Le modèle d'architecture est composé d'un processeur numérique du signal (Digital Signal Processor) préalablement spécifié au niveau transfert de registre (RTL) et des modules matériels.

La communication entre les deux parties est pilotée par le processeur. Une bibliothèque fournit les modèles de description de l'architecture du processeur. Le modèle ISA (Instruction Set Architecture) décrit au niveau cycle d'horloge toutes les instructions que le processeur peut traiter. Le modèle FBM (Full Behavioral Model) décrit le fonctionnement du circuit et spécifie son interface externe.

La particularité de la méthodologie du projet RASSP consiste en l'utilisation de modèles très détaillés du processeur qui exécute le code logiciel. Cela permet d'avoir une simulation très précise au cycle d'horloge près. Néanmoins, cette approche présente quelques

limitations. En effet, RASSP n'offre pas d'outil de découpage logiciel/matériel efficace, et l'allocation logiciel/matériel est effectuée manuellement. Pour les parties logicielles, le langage VHDL n'est pas approprié pour la spécification du code des processeurs logiciels.

1.6.2.LIRMM

Un environnement d'aide à la conception conjointe de systèmes dédiés matériels/logiciels est en cours de développement au LIRMM (Montpellier) [19]. Il s'agit d'une méthodologie interactive qui vise essentiellement le prototypage des systèmes hétérogènes. Les applications traitées touchent principalement le traitement d'images en temps réel. Les descriptions d'entrée sont de haut niveau pour le matériel, et en code assembleur pour le processeur de traitement du signal. Le découpage logiciel/matériel est effectué manuellement suivant les performances requises pour chaque module.

L'architecture cible est construite autour d'un noyau de processeur de traitement du signal, une mémoire et des modules matériels programmables. Le résultat de la conception est validé sur une carte de prototypage à base des FPGA.

Cet environnement se distingue par la conception des interfaces de communication. Ces interfaces intègrent le protocole de contrôle des sous modules du système. La réutilisation des modules est la principale contrainte de conception dans cet environnement.

1.6.3.MCSE

L'approche MCSE (Méthode et modèle de Conception des systèmes Electroniques) définit une méthodologie complète qui couvre toutes les étapes de développement des systèmes [36]. La spécification d'entrée comprend des spécifications fonctionnelles, exprimées dans un langage graphique fondé sur le modèle de processus communicants, et des spécifications non fonctionnelles exprimées sous forme de contraintes (temps, performances, technologie, etc.). Le partitionnement repose sur les estimations de performances recueillies à partir du modèle de performances de MCSE.

Le modèle, dit non interprété, est formé de deux parties: la structure et le comportement. Le modèle structural résulte de la composition de une structure fonctionnelle et de l'architecture matérielle donnée par le processus d'allocation. Le modèle comportemental de chaque fonction est une abstraction du comportement algorithmique. Les attributs et les paramètres associés à ce modèle indiquent les propriétés de tous les composants. Ce modèle de performances est utilisé pour la vérification du système dans une étape de cosimulation matériel/logiciel au niveau modèle (simulation VHDL dite non interprétée) [37].

Des simulations en VHDL et en C++ sont également possibles. Elles permettent d'assister l'activité de partitionnement matériel/logiciel. Dans une deuxième phase, le modèle de performances et l'architecture matérielle sont employés pour obtenir par la synthèse les descriptions matérielles et logicielles. Les deux descriptions sont utilisées, pour une vérification finale, par une cosimulation interprétée détaillée.

1.6.4.Ptolemy

L'environnement de Codesign Ptolemy, de l'université de Berkeley, permet le développement d'applications de traitement du signal et de systèmes communicants [30] [31]. Le modèle initial est formé par un ou plusieurs programmes, s'exécutant sur des composants programmables. Au niveau système, cette description peut inclure des contraintes de conception telles que: les contraintes temps réel, la vitesse, la surface, la taille du code, la consommation, etc. L'architecture cible comprend une variété de processeurs qui peuvent avoir une configuration parmi plusieurs: système monoprocesseur, architectures parallèles à mémoire partagée, avec bus partagé ou avec passage de messages.

L'approche est formée de trois étapes: le découpage matériel/logiciel, la synthèse du logiciel et du matériel et la synthèse des interfaces matérielles/logicielles. Le découpage, guidé par les contraintes d'entrée, essaie d'optimiser des fonctions de coûts telles que le coût des communications, la surface ou la vitesse. La simulation du système a lieu une fois que la synthèse du logiciel et du matériel est faite.

1.6.5.Cosyma

L'environnement de Codesign Cosyma, de l'université de Braunschweig, est dédié aux systèmes complexes [32] [33]. La description en entrée est une spécification textuelle en langage Cx. Le langage Cx est une extension au langage C permettant d'inclure des contraintes de temps et le parallélisme entre les processus. L'architecture cible utilise un seul processeur avec un seul circuit intégré (ASIC), une mémoire et un bus.

La description d'entrée est traduite dans une représentation interne appelée graphe syntaxique étendu. Ensuite, une simulation préliminaire est réalisée pour extraire les informations nécessaires au découpage. Le découpage matériel/logiciel est automatique et basé sur un algorithme de recuit simulé (ang. "Simulated annealing"). Les parties à réaliser en logiciel sont traduites en langage C et les parties à réaliser en matériel sont traduites en langage HardwareC.

1.6.6.CoWare

L'environnement nommé CoWare permet la conception des systèmes distribués comportant plusieurs processeurs logiciels programmables du type DSP (Digital Signal Processor) ou microcontrôleurs, et des modules matériels dédiés [34] [35]. L'architecture utilisée permet d'avoir un modèle composé de processeurs interconnectés par des canaux de communications du type point à point. La communication entre les modules utilise un protocole rendez-vous, qui est synchronisé par des signaux d'attentes. Un module logiciel est constitué par un cœur de processeur programmable, une mémoire et une unité d'E/S qui assure la communication matérielle avec l'environnement. Les composants sont décrits en langage C (pour les modules logiciels) et en langage VHDL (pour les parties matérielles).

Les modules utilisés sont généralement des composants existants rassemblés pour former une architecture multiprocesseur. Cet environnement permet la génération automatique d'interfaces et la cosimulation à plusieurs niveaux d'abstraction.

1.6.7.Revue des environnements de codesign

Cette section classe les outils les plus connus pour la conception conjointe logiciel/matériel. Désigne [08] [18] [38] La liste des outils ne se veut pas exhaustive mais représentative du domaine. Ces outils sont développés soit dans des universités, soit dans des centres de recherche et développement appartenant à des industriels. Pour chaque outil, le tableau donne:

- le langage de spécification utilisé en entrée,
- le type d'application visée,
- l'architecture cible.

Outils	Spécification des systèmes	Type d'application	Architecture cible
Specsyn (ivrine) [47]	SpecChart	Système de contrôle et de communication	Multi-processeurs+ asics
Ptolemly (berkely) [48]	Blocs interconnecté Multi langages	Traitement de signales et système communicant	Mono processeur
Vulcan (stand ford) [46]	Hardware c	Système temps réel	CPU, ASIC+bus+mémoire
Cosyma (braunschweig) [41]	Cx	Système complexe	Multi-processeurs+ fpga+asic
Codes (siemens) [40]	Sdl, state chart	Système de communication	Multiprocesseur +ASIC+FPGA
Tosca (iteltel) [39]	Speedchart	Système de communication	Monoprocesseur+ coprocesseur
Syndex (inria) [43]	Signal	Traitement de signal	Multiprocesseur
SAW (Carnegie melion) [44]	CSP	Optimisation de fonction de logiciel	Cpu+FPGA+ASIC
Coware (imec) [51]	C+vhdl+coware	Traitement de signale et systèmes communicants	Multiprocesseur +asics
MSCE ireste [42]	Formalisme personal de MSCE	Système de communication et de contrôle	Monoprocesseur +asip+dsp
Rassp [49]	Vhdl	Traitement de signale	CPU+DSP
Rapid [50]	Vhdl	Système de communication et de contrôle	Au moins un processeur pour le logiciel
Lycos [45]	C	/	CPU+ASIC
Cosmos [08]	SDL	System de contrôle et de communication	Multiprocesseur +ASIC+FPGA

Tableau 1.1: Tableau récapitulatif des outils de conception logiciel /matériel

Les langages de spécifications peuvent être soit mono flot (par exemple le langage C), soit multi flot (les systèmes décrits en langage SDL). L'architecture cible peut être soit monoprocesseur, soit multiprocesseurs. Une architecture monoprocesseur se compose généralement d'une unité de contrôle centrale (CPU) avec un ou plusieurs ASIC (ou FPGA). Dans ce cas, les composants matériels jouent le rôle d'accélérateurs pour la partie logicielle. Pour les architectures multiprocesseur, le contrôle est distribué entre ces éléments. Pour ce qui concerne les étapes de synthèse, celles-ci dépendent du langage de spécification initial et de l'architecture cible. Généralement, les différentes approches réalisent un découpage logiciel/matériel basé sur des estimations, ensuite compilation des parties logicielles et synthèse des parties matérielles. La phase finale est l'intégration des différentes parties sur une architecture prototype. Le résultat peut servir à l'émulation ou bien au prototypage.

1.7.Conclusion

Les objectifs de ce chapitre étaient éclairés de Codesign, et de présenter la méthodologie de ce processus de conception, ses intérêts, ses domaines d’application et l’organisation des architectures cibles. Le chapitre suivant va se baser aux étapes de ce processus.

CHAPITRE 2

DIFFERENTES ETAPES DE PROCESSUS DU CODESIGN

2.1.Introduction

La stratégie de conception et de développement conjoint de systèmes composé du matériel et du logiciel nécessite plusieurs étapes. Il s'agit de permettre la spécification, la validation (exhaustive par exemple), la synthèse, la simulation (ou co-simulation) et la réalisation de systèmes sur des architectures cibles.

Avec une méthode de co-design, à partir du cahier des charges on va décrire le fonctionnement du système sans discriminer le logiciel et le matériel. Le partitionnement peut se faire de manière automatique pour optimiser certains critères: plusieurs configurations logicielles/matérielles peuvent être testées, ce qui ne serait pas si simple avec un cycle traditionnel (cela impliquerait le développement de plusieurs solutions en parallèle). De plus, un changement de spécification ne remettra pas autant de travail en cause que dans le cas d'un développement traditionnel pour lesquels, au fur et à mesure de la conception, des contraintes sont définitivement adoptées. Il en est de même pour l'ajout de fonctionnalités.

Le cheminement de ces étapes de Codesign nous a permet de construire ce schéma qui est fait par [11].

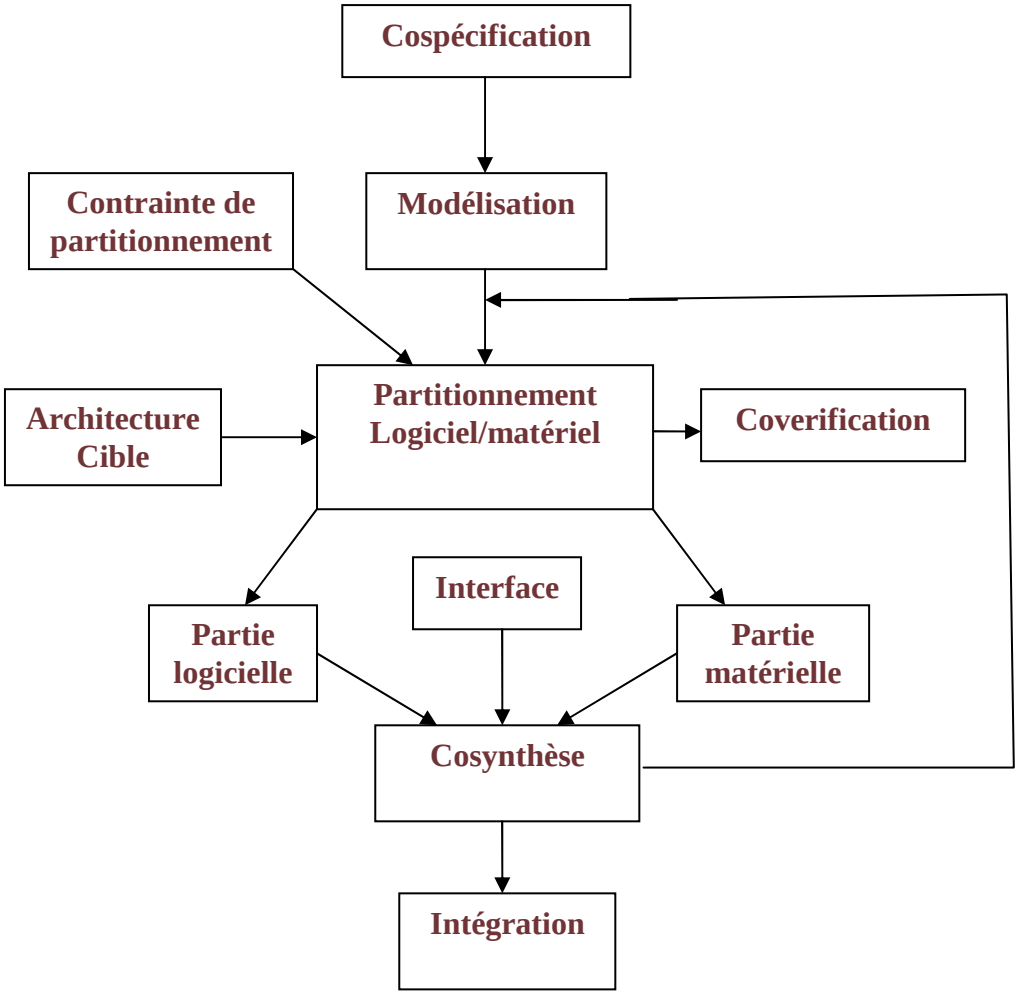

Figure 2.1:principe étape de processus de Codesign [11]

2.2.Les étapes du Codesign

Dans ce qui suit, nous revenons plus en détail sur ces différentes étapes.

2.2.1.La cospécification

Cette section discute sur les langages utilisés pour la conception des systèmes mixtes logiciel/matériel. Le choix du langage le plus approprié pour définir le cahier des charges est une tâche importante, car il y a une pléthore de langages de spécification [29]. Le choix d'un langage résulte généralement d'une pondération de plusieurs critères tels que la puissance expressive du langage, les capacités d'automatisation fournies par le modèle sous-tendant le langage et de la disponibilité des outils et méthodes supportant le langage. D'ailleurs, pour une

application donnée, plusieurs langages peuvent être utilisés pour la spécification des divers modules qui font partie du système.

La spécification d'un système mixte logiciel/matériel peut suivre une des deux approches:

Spécification homogène: un langage unique est utilisé pour la spécification du système global comprenant des parties matérielles et des parties logicielles.

Spécification hétérogène: des langages spécifiques sont utilisés pour les parties matérielles et logicielles.

Chacune des deux stratégies de conception implique une organisation différente du processus de Codesign, donc de son environnement.

2.2.1.1.Spécification homogène

La première étape du processus de conception consiste à spécifier le système global. La spécification est une représentation abstraite de la solution envisagée [125], L'étape de spécification consiste, en général, à décrire les fonctionnalités du système à concevoir ainsi que les contraintes qu'il doit respecter. Il est donc important que le formalisme utilisé soit assez puissant pour inclure le logiciel, le matériel, ainsi que toutes les contraintes associées au système.

Le problème essentiel dans l'étape de cospécification est de permettre de décrire les différentes opérations d'un système à concevoir, tout en capturant toutes les informations nécessaires à la bonne exécution du système (par exemple les contraintes architecturales et de fonctionnement), sans prendre de décision de mise en œuvre ni introduire de biais vers l'une ou l'autre des implémentations (logiciel ou le matériel), ce qui pourrait influencer négativement les étapes ultérieures [126] [11].

Le cahier des charges d'une spécification homogène implique l'utilisation d'un langage unique pour spécifier le système global. Un environnement générique de Codesign basé sur un modèle homogène est schématisé dans la figure 2.2. Le système est décrit comme étant un ensemble de fonctions et de contraintes, et cela indépendamment de toute considération matérielle ou logicielle. A ce niveau, un langage de spécification système peut convenir, ainsi que toute autre représentation fournissant un modèle exécutable du système. Le cahier des charges passe par une étape de découpage en parties logicielles et matérielles, visant à obtenir une première approche de l'architecture cible. Le résultat qui en découle, le prototype virtuel,

est une architecture faite de composants matériels et logiciels. La dernière étape, qui consiste à effectuer la synthèse du matériel et la génération de code pour les processeurs, produit un premier prototype du système.

Dans de tels environnements de Codesign, la question clef est la correspondance entre les concepts utilisés dans le modèle initial et les concepts fournis par l'architecture cible. Par exemple, établir la correspondance entre la spécification au niveau système, comprenant des concepts de très haut niveau tels que le contrôle distribué ou la communication, et la représentation faite par des langages de bas niveau tels que C et VHDL est une tâche non négligeable [8][30][31].

Erreur ! Des objets ne peuvent pas être créés à partir des codes de champs de mise en forme.

Figure 2.2: Flot de Codesign pour la spécification homogène

Plusieurs environnements de Codesign suivent cette option. Néanmoins, afin de réduire la distance entre le modèle de spécification et le prototype, ces outils démarre par un modèle de spécification de bas niveau par rapport au niveau système. Certaines de ces approches ont étendu des langages mono flot pour supporter des concepts matériels et de la communication. L'environnement de Codesign Cosyma accepte une spécification en langage Cx extension du langage C [24] [25]. La description en entrée de l'environnement Vulcan, de Standford, est une autre extension du langage C appelée HardwareC [32] [41] [57]. Lycos et Castle utilisent le langage C [33] [34] [104]. D'autres outils de Codesign utilisent le langage VHDL [35].

On peut noter que peu d'outils ont essayé de partir d'un modèle de niveau plus élevé. Dans ces derniers nous trouvons Polis [36] qui utilise Esterel [37], SpecSyn de Irvine [38] [58] qui utilise SpecCharts [39], et l'environnement décrit dans [31] qui utilise LOTOS. Cosmos, qui utilise le langage SDL, appartient aussi à ce groupe. Peu d'approches ont utilisé des langages de spécification de systèmes distribués tel que SDL, LOTOS ou ESTELLE [16] [31] [46]. Cela est dû principalement à la distance existant entre les concepts manipulés par ces langages et ceux des langages de description matérielle. En général, la traduction directe de tels langages est seulement possible en introduisant des restrictions sur le modèle d'exécution, les aspects dynamiques ou les modèles de communications disponibles.

Dans [110], le modèle de communication Estelle est traduit en VHDL. Dans [31], les processus LOTOS sont traduits en automates d'états finis VHDL et chaque schéma de communication à une seule implémentation basée sur un module de synchronisation prédéfini

extrait d'une bibliothèque VHDL. Dans [16], Glunz présente une approche de traduction SDL vers VHDL. Le modèle de communication SDL peut être changé à l'aide d'une bibliothèque de protocoles.

2.2.1.2.Spécification hétérogène

Le cahier des charges des systèmes à base de spécification hétérogène permet l'utilisation de langages spécifiques pour les parties matérielles et logicielles. Un environnement générique de Codesign basé sur un modèle hétérogène est donné par la figure 2.3.

Le Codesign commence par un prototype virtuel. Dans ce cas, le Codesign est une correspondance des parties logicielles et matérielles sur des processeurs dédiés.

La partie logicielle, traditionnellement écrite en assembleur, est souvent décrite en langage C. L'utilisation de ce langage donne la possibilité de réutilisation du code et sa validation indépendamment du processeur cible [4] [5]. Pour les parties matérielles, les deux langages de description les plus utilisés actuellement sont VHDL et Verilog [42].

Ces langages traitent les caractéristiques spécifiques de la conception de matériel et les niveaux d'abstraction, permettant l'ajout des détails résultant de la synthèse.

Les points clef avec un tel arrangement sont la validation de la spécification et la génération des interfaces. L'utilisation d'une description multi langages exige de nouvelles techniques de validation capables de manipuler de tels modèles. Au lieu de la simulation nous aurons besoin de cosimulation et au lieu de la vérification nous aurons besoin de coverification. Par ailleurs, le cahier des charges multi langages entraîne un problème d'interface entre sous-ensembles qui sont décrits par des langages différents. Ces interfaces doivent être raffinées par rapport à l'architecture cible à partir de la spécification initiale.

CoWare [18] [27] est un environnement typique qui supporte un tel flot de Codesign. L'environnement utilise une description mixte donnée en VHDL pour le matériel, et C pour le logiciel. Cet environnement tient compte de la cosimulation.

Erreur ! Des objets ne peuvent pas être créés à partir des codes de champs de mise en forme.

Figure 2.3:flot de conception pour la spécification hétérogène

2.2.1.3.Spécification multi langages

La spécification multi langages peut être vue comme une généralisation de la description hétérogène d’architectures mixtes logicielles/matérielles [34]. Il s’agit de traiter des systèmes

composés des modules hétérogènes. La caractéristique de la méthode de spécification multi langage est l'utilisation de plusieurs langages pour la spécification de la fonctionnalité du système. La figure 3.4 illustre un environnement générique de Codesign basé sur un modèle multi langage.

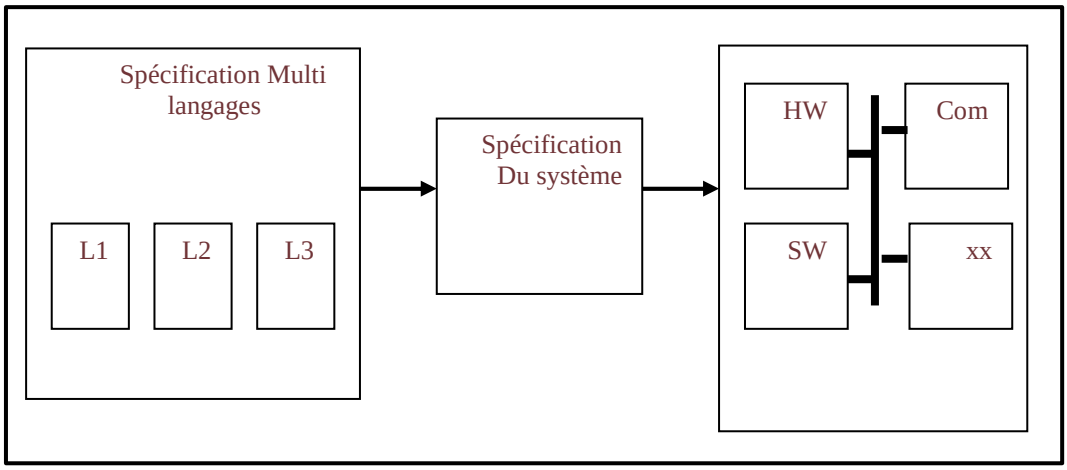

Figure 2.4:Flot de conception pour une spécification multi langage [34].

2.2.1.4.Les langages de description du matériel

Plusieurs approches utilisent un langage de description de matériel pour décrire les parties matérielles d'un système en conjonction avec un langage logiciel [20]. VHDL et VERILOG figurent parmi les logiciels de description de matériel les plus utilisés soit directement comme langages de spécification des applications mixtes (matériel/logiciel), soit à l'issue de la dernière étape du Codesign afin de permettre l'utilisation d'outils de synthèse commerciaux.

2.2.1.5.Les approches de description de logiciels classiques

Dans l'approche de description de logiciels classiques, le système est spécifié en logiciel, en utilisant un langage classique servant habituellement à décrire le logiciel. Les premiers langages utilisés pour la spécification des systèmes mixtes sont les langages d'assemblage. Etant donné le manque de souplesse et la relative pauvreté d'expression de ces langages [46], ils sont actuellement désertés au profit de langages de plus haut niveau (C, C++, Java…).

2.2.1.6.Les langages synchrones

Les langages synchrones ont été définis pour d'écrire le fonctionnement des systèmes temps réel. Parmi les langages synchrones, deux langages ont été utilises pour des projets de Codesign.

- Esterel:

Le langage Esterel est un langage synchrone oriente contrôle. Il est utilise dans le projet de co-design Polis [61] qui retranscrit les spécifications en un ensemble de machines d'états finis supportant la concurrence et la communication Il existe aussi des compilateurs du langage Esterel en un ensemble d'équations booléennes utilisées dans la conception de processeur matériel.

- Signal:

Le langage Signal [38] est un langage synchrone oriente activité. Il est utilise dans le projet Syndex pour construire un graphe de flots de données utilise ensuite pour repartir les taches sur une architecture multiprocesseurs.

2.2.2.La modélisation

Cette étape consiste à remplacer le système physique mixte qu'on veut concevoir par une représentation symbolique ou modèle [46]. Un modèle peut être défini comme étant une abstraction de ce qui existe dans le monde réel. Il peut être mathématique ou constructif.

En Codesign, le choix d'un modèle interne pour un outil de conception assistée par ordinateur dépend fortement du type de système à concevoir. Objectifs d'un modèle

Parmi les objectifs d'un modèle dans le Codesign, citons:

Permettre au concepteur de capturer les fonctionnalités et les contraintes du système à concevoir.

Il doit être synthétisable (c'est-à-dire permettre le passage automatique à la phase de synthèse).

Permettre la vérification et/ou simulation.

2.2.2.1. Quelques modèles utilisés en Codesign

Nous présentons ici quelques modèles utilisés pour le processus de Codesign. L'objectif n'est pas de détailler chaque modèle, mais juste de citer et de donner des exemples de modèles les plus utilisés dans le Codesign sont classés par

Model a base de graphe: Ils sont utilisés pour modéliser les applications de traitement de signaux digitaux. On situe les

DFD: Les graphes de flot de données représentent un calcul ou les arcs représentent le transfert d'une donnée et les noeuds représentent une activité comme une instruction, une opération arithmétique, une fonction, etc. On distingue aussi deux autres types de noeuds, les noeuds d'entrées et de sorties et les noeuds de stockage (mémoire). Ce modèle supporte la hiérarchie: l'activité d'un noeud peut être un autre graphe de flots de données.

CDFG: Les graphes de flot de données contrôlé: Dans ces diagrammes, les arcs, qui indiquent la mise en séquence des opérations, représentent un flot de contrôle. Ils sont adaptés pour représenter des algorithmes comportant des branchements et des itérations. Ce type de graphe permet de spécifier l'ordonnancement et les activités qui ne dépendent pas d'évènements externes.

Les model orienté contrôlée: Ils sont utilisés pour modéliser les systèmes mixtes et les flots de contrôle dans les systèmes synchrones. Cette modélisation est adaptée pour représenter les systèmes dits de contrôle ou réactifs. Ils permettent de représenter le comportement temporel d'un système par rapport aux évènements qui peuvent survenir. Ce sont des modèles qui permettent de recenser les états que peut prendre le système et qui expriment les conditions de passage d'un état à l'autre. On situe:

RDP: Les réseaux de Pétri: Les applications peuvent être représentées de manière algébrique en considérant le réseau de Pétri comme un graphe contenant deux types de noeuds: les places et les transitions. L'application représente alors les arcs qui relient une place à une transition et l'application Post représente les arcs qui relient les transitions aux places.

Un réseau de pétri permet de représenter les systèmes ayant des états concurrents. Il est possible de représenter un grand nombre de caractéristiques d'un système temps réel

FSM: Les automates à états finis: Le principe des automates est de représenter la solution d'un problème par une succession d'étapes définies, séparées les unes des autres.

Chaque étape est représentée par une valeur faisant partie d'un ensemble _ni que l'on appelle l'état (interne) du système. Cet état est physiquement matérialisé par un nombre contenu dans une mémoire.

Model hybride: Plusieurs modèles hybrides ont vu le jour, essayant de profiter des avantages de certains modèles de calcul pour masquer les inconvénients des autres. Parmi ces modèles ont peut citer:

State charts: Les Statecharts, définis par David Harel, correspondent à un langage graphique permettant de représenter des machines à états finis avec en plus une description de concurrence et de communication. Ce langage est utilisé pour la spécification des logiciels de systèmes temps réels embarqués Il existe un langage très proche des StateCharts défini dans le cadre du co-design comme une extension, le langage SpecChart de D. Gajski

SDL: Le langage sdl (Specification and Description Language) a été défini pour décrire les systèmes à temps réels distribués et dédiés à la télécommunication. Il est standardisé par l'itu3. Un système décrit en SDL est composé d'un ensemble de processus communiquant au travers de signaux [49]. Ces processus sont ensuite définis comme des automates à états finis. La communication (implicite au langage) est totalement asynchrone.

Model orienté objet: Le langage de modélisation orienté objet émergeant actuellement et utilisé principalement pour le développement de systèmes logiciels tel que UML.

Le choix du modèle dépend fortement du système à concevoir. Il est guidé par l’ensemble des contraintes et des caractéristiques de l’application. Un modèle choisi doit tenir compte de toutes les étapes du processus de Codesign: le partitionnement, la synthèse de la communication et des interfaces, la covérification/covalidation. Les travaux de [40, 20, 69, 34] modélisent une unité de communication comme étant un objet fournissant un ensemble de primitives de communication (méthodes) réalisant un protocole spécifique.

La tendance actuelle dans de nombreuses équipes de recherche est à l'unification, pour aboutir à un modèle de calcul concurrent unique [46]. Ceci pourrait être accompli en réalisant un mélange d’un certain nombre de modèles, mais un tel modèle serait extrêmement complexe et difficile à utiliser, et les outils de synthèse et de simulation seraient difficiles à concevoir [48].

2.2.3.Le partitionnement hardware/software

Le problème du partitionnement matériel/logiciel est au coeur de l’activité de co-design. Le choix de l’architecture matérielle est un élément de décision essentiel et la

démarche diffère selon que l'architecture se trouve imposée ou choisie d'emblée ou que l'architecture et les Méthodologie de co-design et estimation des performances composants de celle-ci sont à déterminer.

La première situation est la plus commune et la plus simple. L'architecture matérielle est généralement une architecture générique constituée d'un microprocesseur, d'un ensemble de circuits matériels programmables (FPGA) ou d'ASIC et d'une mémoire commune. Le problème du partitionnement se réduit alors à un problème de partitionnement binaire matériel/logiciel pour l'allocation des éléments fonctionnels sur les constituants de l'architecture et peut se résoudre de manière automatique.

Dans ce cas, face à la nature hétérogène de l'architecture cible et à la diversité des contraintes imposées, une démarche itérative et guidée par le concepteur s'impose. Il s'agit alors d'offrir au concepteur des moyens rapides d'estimation des propriétés de l'implantation résultant du choix de l'architecture, du partitionnement et de l'allocation pour vérifier si celles-ci répondent aux contraintes imposées.

Pour les parties du système relevant du co-design, les contraintes imposées sont surtout des contraintes de performances. En effet, les contraintes telles que la flexibilité, la testabilité, l'utilisation de composants du commerce ou de technologies maîtrisées par l'entreprise, la sûreté de fonctionnement et les coûts interviennent principalement au niveau du partitionnement système qui a pour but le découpage du système en un ensemble de partitions où chaque partition devra s'exécuter soit en logiciel soit en matériel. Les contraintes de performances sont de nature statique ou dynamique. L'estimation des performances statiques telles que la surface de silicium occupée, la puissance consommée repose sur des techniques de synthèse. La plupart des travaux de la communauté du co-design sur l'estimation des performances dynamiques d'un partitionnement, sont basés sur une analyse des contraintes temporelles et un calcul de la charge du processeur par des techniques proches de celles utilisées en ordonnancement de tâches pour des systèmes temps réels.

Le partitionnement matériel/logiciel assure la transformation des spécifications de la partie du système relevant de l'activité co-design en une architecture composée d'une partie matérielle et d'une partie logicielle. Les spécifications considérées sont en réalité une description fonctionnelle détaillée résultant d'une approche système. Cette transformation s'effectue habituellement en deux phases: la sélection d'une architecture matérielle et l'allocation des éléments (fonctions et éléments de relations) du modèle fonctionnel sur les

éléments de cette architecture. Plusieurs critères peuvent intervenir sur le double choix (architecture et allocation) d'un partitionnement tels que par exemple:

Les performances statiques (consommation, surface de silicium, coûts, taille du code, taille de la mémoire, etc.) et dynamiques (contraintes de temps, débit, temps de latence, taux d'occupation, etc.). Elles influent surtout sur l'allocation des fonctions.

La sécurité: La prise en compte de la sûreté de fonctionnement peut induire des contraintes au partitionnement matériel/logiciel (redondance de composants matériels et de tâches logicielles par exemple),

La flexibilité: l'implantation logicielle d'une fonction offre des possibilités d'évolution plus importantes qu'une implantation matérielle,

La réutilisation: La réutilisation de composants est un facteur important de productivité, mais introduit des contraintes au niveau du partitionnement, la testabilité: l'extraction d'informations en temps réel nécessite l'ajout de composants matériels supplémentaires (Bist, Boundary Scan) ou l'ajout d'instructions de capture.

2.2.3.1.La recherche interactive**:**

Cette recherche se base sur l'expérience et l'esprit d'analyse du concepteur qui décide des partitionnements à estimer afin de trouver l'architecture qui permet de respecter des contraintes. Dans ce cas, l'environnement de co-design doit fournir a l'utilisateur une boite a outils lui permettant de sélectionner un ensemble d'atomes et de les affecter a un composant particulier. Ce type d'outil a été implante dans le projet de co-design Cosmos [49].

Le projet Ptolemy utilise des mécanismes identiques permettant de découper les spécifications en plusieurs taches et d'associer un langage particulier pour la synthèse de chaque tache. Le découpage doit en plus tenir compte des interfaces de communications entre les taches qui ne sont pas sur le même composant.

2.2.3.2.La recherche automatique

La recherche automatique est basée sur un algorithme dit \heuristique" qui ne parcourt qu'une partie de l'arbre des solutions. Cette approche a été utilisée dans plusieurs projets. Cependant la complexité du problème étant importante, les projets fixent le choix des composants. Ce choix correspond à l'association d'un processeur logiciel et d'un processeur matériel. L'algorithme est alors un algorithme dit de partitionnement.

Le problème du partitionnement se ramène alors à un problème de répartition des taches entre le processeur logiciel et matériel. Pour la résolution automatique du partitionnement, un graphe de type flots de données et de contrôle est généralement utilise. Ce type de graphe permet en effet de résoudre une partie du problème de l'ordonnancement des atomes. En effet, l'algorithme de partitionnement doit, pour repartir les différents atomes sur le processeur logiciel et le processeur matériel, déterminer l'ordre d'exécution de ceux-ci. La répartition se fait généralement en considérant trois critères:

1. le temps de calcul,

2. la taille du système: elle correspond, dans le cas du processeur matériel, au nombre de portes logiques et de bascules utilisées et, dans le cas du processeur logiciel, au nombre d'octets nécessaires a la sauvegarde du programme et des données associées,

3. la communication entre le processeur matériel et le processeur logiciel (quantité de données transférées).

Ces trois critères dans certains algorithmes sont normalises et composes pour construire une fonction coût:

Fonction coût = k1 * temps calcul + k2 * taille + k3 * communication

Les méthodes de partitionnement sont alors construites de la manière suivante. Une première partition est élaborée manuellement ou automatiquement. Certains projets partent d'une partition qui est soit entièrement logicielle (le projet Cosyma [57]) et dans ce cas elle ne répond pas, en général, au critère de temps de calcul, soit entièrement matérielle (projet Vulcan [29]) et elle a un coût important en terme de taille.

A partir de la partition initiale, l'algorithme de partitionnement va déplacer des atomes entre le processeur logiciel et le processeur matériel afin de diminuer la fonction coût et de respecter les contraintes. Le déplacement des atomes n'est cependant pas aléatoire mais se base sur des critères de proximité afin de limiter le nombre de communications entre le logiciel et le matériel.

Apres chaque déplacement la fonction coût est revalue en estimant les différents critères et un partitionnement est de nouveau effectue jusqu'a obtenir un minimum de la fonction coût. Ce minimum est cependant très souvent un minimum local étant donne les algorithmes utilises. En effet, on utilise des heuristiques comme les algorithmes gloutons [29], le recuit simule [57] ou bien la programmation linéaire entière [55].

Les deux approches se basent sur une estimation des performances du système. Ces performances se composent essentiellement du temps de calcul et de la taille de chaque répartition. Notons cependant que d'autres critères peuvent aussi s'ajouter comme la consommation et le coût du système. L'estimation des performances est donc une étape très importante dans l'exploration des architectures qui ne peut se dissocier, comme nous allons le voir, de la synthèse des parties matérielles et logicielles.

2.2.4.La synthèse de l'interface hardware/software

Les deux premières parties de la synthèse ne constituent pas un réel problème, car elles sont envoyées vers des compilateurs et outils de synthèse hardware. Par contre, la synthèse de l'interface constitue une tâche plus ardue. L'interface est composée de protocoles de communication hardware/software en utilisant par exemple des canaux de communication ou des ports d'entrée sortie (par exemple entre le microprocesseur vers les FPGA via un bus). La synthèse de cette partie implique la réalisation physique de ces protocoles en garantissant la synchronisation dans la communication hardware/software. Des circuits propres à la communication sont générés à la fin de cette étape.

Les autres étapes du Codesign peuvent aider l'étape de la synthèse de deux manières:

En lui fournissant des informations sur le système: la simulation et l'évaluation extraient des informations temporelles, le volume moyen du flux de transfert d'informations, etc.

En exécutant une partie de la synthèse: par exemple, le partitionneur fait la synthèse des parties software en les envoyant vers des compilateurs.

Après l'étape de la synthèse, des retours arrière peuvent être effectués à cause des contraintes software ou hardware. Ce qui suit détaille ces propos.

2.2.4.1. Les contraintes hardware

Des contraintes technologiques émanant des circuits hardware peuvent provoquer des retours arrière pour revoir le partitionnement du système.

Exemple:

Lors de la phase de cosimulation hw/sw (intégrée dans la phase de partitionnement), il se peut qu'on suppose que la fréquence des circuits est de l'ordre de 100 MHZ. Lors de la synthèse hardware de ce circuit, il s'avère qu'il n'existe pas sur le marché de circuits fonctionnant à cette cadence. La fréquence étant diminuée, par exemple jusqu'à 60 MHZ, la synchronisation et la cohérence du système peuvent être affectées (provoquées par le

ralentissement du temps de fonctionnement des circuits hardware). Le processus de partitionnement doit donc être ré exécuté pour réajuster les partitions.

2.2.4.2.Les contraintes software

Supposons par exemple que lors du partitionnement, nous nous ne sommes pas préoccupés de l'espace mémoire réservé au code software exécutable sur le microprocesseur. Après la phase de synthèse, il s'avère que le code software dépasse largement la capacité de la mémoire présente dans l'architecture cible. Le processus de partitionnement doit être ré exécuté pour déplacer des portions de code software vers le hardware.

2.2.5. La covérification

Une attention particulière est donnée à la vérification de la concordance entre le système mixte produit à la fin du processus du Codesign, et sa spécification initiale. La méthode commune est la simulation: c'est la plus simple et la plus intuitive, mais elle demeure une solution non fiable devant la complexité des systèmes modernes: elle peut prouver que le système ne fonctionne pas correctement (apparition d'un bogue en cours de simulation) mais elle n'affirme en aucun cas que le système mixte est correct. Il existe des environnements de simulation hétérogènes qui permettent d'exécuter les différentes parties du système [10]:

Exécuter différentes parties software avec différents types de microprocesseurs

Simuler l'exécution des circuits de la partie hardware

Par contre, la solution la plus fiable est celle qui essaye d'utiliser les preuves formelles de vérification (automates de vérification à états finis [145], modèle équationnel [144] [143]). Les preuves formelles demeurent jusqu'à présent une tâche assez compliquée et suscite l'objet de plusieurs recherches, surtout dans le traitement du non déterminisme que génère quelques types de systèmes (ce qu'on ne peut pas prévoir leur dynamique). Il existe trois manières pour faire cela [10]:

Prouver que la mise en œuvre implique la spécification: nous démarrons la covérification à partir du système mixte pour arriver à la spécification; ceci implique la construction d'un système de preuves qui atteint, à partir d'une construction de programme de bas niveau, une seconde structure plus abstraite.

Prouver que la spécification implique l'implémentation: prouver la validité de chaque étape à partir de la spécification jusqu'à l'implémentation. Dans ce cas, nous démarrons la covérification à partir de la spécification pour arriver au système mixte.

Prouver la correction du processus du Codesign; par conséquent, il produit des systèmes valides. Ceci est effectué en prouvant la correction dans toutes les transformations employées pour la mise en œuvre, et qu'elle préserve toutes les propriétés du système.

Remarque:

Malgré l'automatisation du processus de partitionnement, le choix final de la combinaison hw/sw peut revenir en dernier lieu au concepteur. Il se peut alors qu'il exister un partitionneur mixte (manuel et automatique) afin de permettre au concepteur d'intervenir sur les cas triviaux que l'outil automatique n'a pu analyser. Cependant, le but de l'élaboration d'un environnement pour le Codesign demeure toujours l'automatisation complète des étapes du Codesign, de telle manière que le concepteur intervient très rarement dans les différents cycles.

2.2.5.1. La cosimulation

La simulation est aujourd'hui la méthode la plus couramment utilisée [11]. Vu que la conception des systèmes mixtes devient de plus en plus complexe, la cosimulation est le moyen le plus utilisé. Elle consiste à faire fonctionner le système conçu et de comparer ses réponses avec celles du système souhaiter.

La cosimulation permet la simulation concurrente de l'ensemble du système en coordonnant et en échangeant les données entre les modules [34]. C'est le fait d'élaborer, à partir de la spécification d'un système intégré, un code compilé pour sa partie logicielle et une description HDL pour sa partie matérielle [4]. Elle est un moyen d'obtenir des informations sur différentes caractéristiques du système à concevoir à partir de la spécification, telles que le temps d'exécution, la surface, le volume et le débit de communication, etc. L'approche proposée dans [51] permet la co-simulation multi niveaux des IPs (Intellectual Property), à partir des descriptions comportementales ou d'un graphe de communication « GC » fournis par le concepteur, et de modèles « fonctionnels ».

2.2.5.2.La covalidation/covérification formelle

Les approches de vérification classiques par simulation et test ne sont pas très efficaces pour des systèmes complexes (tailles importantes). C'est très coûteux, voire impossible, de tester ou de simuler toutes les entrées possibles d'un système dont le nombre d'entrées est excessivement grand [4].

C'est pourquoi une autre approche plus fiable, consiste à étendre les méthodes formelles au domaine de Codesign matériel/logiciel et utiliser ces méthodes pour la validation. La vérification formelle permet de révéler un certain nombre de propriétés d'un système, difficile à mettre en évidence par des techniques usuelles [76]. La validation par des techniques formelles consiste à utiliser de méthodes formelles afin de vérifier le comportement du système pour tous les cas possibles.

L'introduction des méthodes formelles dans le flot de conception est motivée par un besoin de correction et d'assurance plus élevée pour les concepteurs. Jusqu'ici, la simulation était le seul moyen de vérification, et elle intervenait tard dans le processus de conception [11]. Il est donc d'un grand intérêt de concevoir et de développer des méthodes formelles dédiées à la conception des systèmes.

Les techniques de la vérification formelle cherchent à démontrer si un design est correct ou non sans avoir besoin d'une simulation exhaustive. La vérification formelle d'un système se fait à la manière des preuves mathématiques des théorèmes, sans se soucier des valeurs des entrées [4]. Les travaux de Chiodo et al [16] utilisent la validation formelle sur le modèle des CFSM, pour prouver mathématiquement qu'une certaine propriété spécifiée formellement est vérifiée.

Les travaux de [67] utilisent les réseaux de Pétri comme formalisme de description en Codesign, afin de modéliser les différentes parties des systèmes hétérogène, pour permettre d'effectuer des preuves formelles. Les travaux de [11] intègrent les méthodes formelles dans le flot de synthèse asynchrone TAST. Dans ce flot, les spécifications initiales sont écrites dans le langage CHP, elles sont interprétées en termes de réseaux de Pétri.

La validation formelle donne des résultats plus précis par rapport à simulation. Par contre, elle ne permet pas de déduire les caractéristiques temporelles ou les goulots d'étranglement par exemple comme le permet la cosimulation.

2.3.Conclusion

pour conclure cette section, ce que nous avons fait jusqu a maintenant une présentation dans étapes sont présenté par [11].

on a dit que l'étape de partitionnement est très intéressante et se base sur l'estimation des performance afin d'applique un algorithme de partitionnement

le chapitre suivant va nous présentez les différent performance et comment estimé ces performance.

CHAPITRE 3

LES PERFORMANCES DANS LA METHODOLOGIE DE CODESIGN

3.1.Introduction

Le découpage logiciel/matériel génère une réalisation au niveau comportemental qui doit satisfaire aux contraintes de conception telles que la performance et la surface.

L'utilisateur conçoit des architectures qui doivent satisfaire aux besoins fonctionnels, mais aussi aux restrictions de coût et aux objectifs de performance [47]. Une fois les besoins fonctionnels établis, lors de l'étape de spécification, l'utilisateur doit réaliser son système en optimisant l'architecture. Le choix de la solution optimale dépend de plusieurs critères. Les critères les plus courants concernent les performances et le coût, mais d'autres critères mesurables peuvent être aussi importants pour certains domaines d'application.

Ce chapitre va présenter une définition générale sur les performances ainsi des méthodes de calculer les performances dans un système hw/sw.

3.2.Les performances

Le terme performance est très souvent utilisé sans que la signification soit claire. Il est par exemple usuel de l'employer pour décrire la qualité d'un système à satisfaire un objectif demandé.

Nous définissons le terme performance comme une quantification d'un système vis à vis des critères d'observation externes ou internes. Ainsi par performance d'un système, nous entendons ici des performances globales telles que la capacité de traitement, le taux d'utilisation de ressources, le rendement, etc., mais aussi des performances locales ainsi que des contraintes temporelles impératives en particulier pour les systèmes temps réel: temps maximum de réaction à des événements, fréquence d'activation d'une tâche par exemple.

Les contraintes de performances sont obligatoirement des grandeurs quantitatives et mesurables déterminées par des valeurs numériques. Ces valeurs quantifient la qualité d'un aspect particulier du système placé dans un contexte donné d'exploitation.

3.2.1.Classification des performances

Une première approche consiste à classer les performances en deux catégories:

• les *performances statiques* qui sont des exigences indépendantes du temps. Citons comme exemples: La capacité mémoire nécessaire, la consommation maximale, le poids, l'encombrement, le coût, etc.

• les *performances dynamiques* qui spécifient les caractéristiques d'évolution temporelle du système dans son environnement (contraintes de temps, de débit, etc.), ainsi que les caractéristiques internes (taux d'utilisation de ressources internes, disponibilité d'un bus, etc.).

Les performances statiques peuvent se déterminer à l'aide d'estimateurs (calcul analytique et/ou heuristique). Par contre l'estimation des performances dynamiques nécessite l'utilisation de modèles analytiques, d'un prototype ou d'un modèle de simulation.

La complexité actuelle des systèmes sort souvent du domaine d'application stricte des modèles analytiques.

Le prototypage est aussi trop cher et/ou trop long à mettre en oeuvre. De plus, il ne peut intervenir que tard dans le cycle de développement. Pour estimer les performances dynamiques d'un système, le concepteur a donc recours de plus en plus à la simulation.

Par la suite, nous nous intéressons principalement aux performances dynamiques d'un système qui peuvent aussi se structurer selon deux catégories:

- les *performances externes* au système telles que:
    - o débit en entrée du système,
    - o débit en sortie du système,
    - o temps de réponse ou de réaction sortie(s) par rapport à l'entrée(s), temps de latence,
    - o contraintes d'interactivité,
    - o précisions et erreurs tolérées,
    - o capacité globale du système.
- les *performances internes* au système telles que:
    - o taux d'utilisation des ressources internes: processeur, ligne de communication, bus, etc.
    - o Précisions et erreurs tolérées.

Les spécifications externes correspondent à une appréciation externe du comportement du système considéré comme une "boite noire", basée sur l'évolution temporelle ou fréquentielle des entrées et des sorties. Ces performances sont de type contraintes de temps ou contraintes de capacité ou de débit.

3.3.Les métriques de performance

Dans la littérature, de nombreux paramètres sont cités, parmi lesquels figurent: la flexibilité, le type d'application, la concurrence, la testabilité, la sécurité, la réutilisabilité, les contraintes de sûreté de fonctionnement. Cependant, ces contraintes sont difficiles à quantifier, et aucun des travaux rencontrés n'y fait référence dans une quelconque fonction d'évaluation.

Par contre, les paramètres les plus souvent utilisés peuvent être regroupés en quatre catégories qui sont:

- les paramètres temporels,
- les paramètres d'espace,
- les paramètres de communication
- la dissipation en puissance.

Les paramètres temporels englobent essentiellement: le nombre de cycles d'horloge, le temps d'exécution du logiciel et du matériel, ou des contraintes de débit associées aux variables pour les systèmes temps réel, par exemple.

Les paramètres spatiaux peuvent être divisés en espace matériel et espace logiciel. En ce qui concerne le premier, la capacité est importante pour les composants matériels (exemple: ASIC, FPGA...) où l'espace est une ressource plus rare et par conséquent plus coûteuse que sur les processeurs logiciels où le stockage se fait dans des mémoires. Elle peut être exprimée en termes d'espace (surface de silicium, nombre de portes ou de transistors...) et de coût associé au circuit concerné car ce dernier varie en fonction de la nature du circuit. Le nombre de broches d'entrées/sorties est également une ressource critique qui doit être prise en compte. En ce qui concerne le logiciel, il s'agit en général du nombre de mots occupés par les données et le code sur les différentes mémoires utilisées sur l'architecture cible (RAM, mémoire cache, registres...).

La communication peut être caractérisée par deux types de paramètres: les paramètres temporels et le volume des transferts. Les paramètres temporels décrivent essentiellement le

temps de communication et peuvent permettre de détecter d'éventuels goulots d'étranglement. Le volume des transferts est étroitement lié aux données transférées à travers les interfaces matériel/logiciel et à la bande passante des bus de l'architecture cible.

3.4.Les méthodes d’estimation de performances:

Les méthodes d’estimations que l’on trouve dans la littérature peuvent être classées dans trois catégories: statique, dynamique et mixte:

- Dynamique: les mesures de performance d’une architecture sont les résultats de l’exécution d’un modèle (exemple: simulation).
- Statique: l’estimation de performance d’une architecture est le résultat d’une analyse statique d’une spécification (exemple: analyse du chemins dans une spécification de flot de contrôle).
- Mixte dynamique/statique: c’est l’utilisation de quelques éléments des deux approches précédentes pour l’analyse de performance d’une architecture.

Les approches dynamiques sont en général très précises. Leur inconvénient majeur est le temps nécessaire pour l’obtention du modèle à simuler (synthèse, génération, compilation), ainsi que le temps de la simulation. Ce qui les rend en pratique inutilisable dans le contexte particulier de l’exploration où le nombre de modèles à analyser est énorme. D’un autre part, les approches statiques sont certes très rapides (pas de génération des modèles à simuler, ni de simulation), mais les tâches de modélisation et d’estimation sont complexes à cause de la distance qui sépare les concepts de spécification de l’implémentation.

Une approche d'estimation qui donne des résultats proches des valeurs réelles permet une meilleure sélection de la réalisation. Par contre, cette précision dégrade le temps de réponse, ce qui restreint beaucoup l'exploration de l'espace des solutions. Ce conflit oblige les méthodologies à fixer une stratégie ou une heuristique afin de réduire l'espace des solutions explorée.

L'objectif de l'opération d'estimation est de permettre à l'utilisateur de prendre la plus grande partie des décisions de conception au niveau système, pendant les premières étapes de conception. Ces décisions, basées sur des valeurs fidèles, réduisent le temps de conception. Elles réduisent aussi les retours à des étapes antérieures de la conception, nécessaires pour la correction des erreurs dues à l'adoption des solutions architecturales inadéquates. Le temps de conception croit en fonction du nombre de fois où l'utilisateur doit revenir à des étapes

antérieures de la conception jusqu'à l'obtention de l'architecture désirée (boucle de la conception). L'estimation permet aussi la réduction du coût de réalisation, puisqu'il n'est plus nécessaire de surdimensionner l'architecture afin d'être sûr de satisfaire les contraintes.

Principalement, il y a deux défis imposés aux algorithmes d'estimation de performance:

- Côté logiciel, la performance dépend du processeur utilisé mais aussi des outils et méthodes de compilation du logiciel. En fait, il est difficile de modéliser les techniques utilisées pour augmenter la vitesse d'exécution du code, comme l'hiérarchie de mémoire, le pipeline d'instruction et les optimisations du compilateur;
- Côté matériel, la difficulté est de modéliser les différentes possibilités de réalisation et les différentes décisions prises par la synthèse d'architecture.

3.4.1.L'estimation du logiciel

L'estimation du temps d'exécution des modules logiciels sur des processeurs programmables a pour objectif de calculer des informations sur le temps d'exécution de chaque module, la taille du code et la taille des données.

Pour l'estimation du nombre de cycles d'horloge nécessaires à l'exécution d'une instruction, sur un processeur cible, nous avons choisi de distinguer le calcul entre le temps pris par les processeurs et le temps pris par les systèmes de mémoire. Cette séparation est utile car les méthodes d'évaluation de ces composants sont différentes.

Le temps d'exécution lié au processeur dépend de ses caractéristiques et des caractéristiques de la réalisation. Cette dernière peut être calculée à partir des annotations résulter de l'étape de calcul de la fréquence d'exécution. La figure suivante illustre le flot d'estimation des modules logiciels. La première étape est la conversion des modules destinés au logiciel dans un code générique, plus proche de l'architecture des processeurs.

Cette conversion est considérée comme une compilation très simplifiée. La seconde étape est le choix du processeur. Lors de cette étape, la bibliothèque des processeurs est utilisée. La dernière étape est l'application du modèle de calcul d'estimation et la mise en forme des résultats pour l'utilisateur. Ces étapes sont détaillées dans les sections qui suivent.

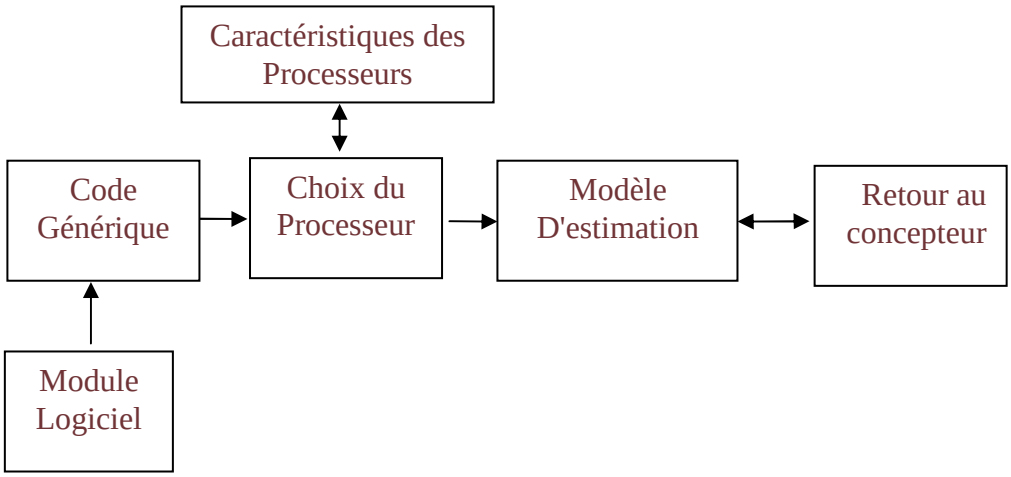

Figure 3.1:Flots d'estimation des modules logiciels

3.4.2.l'estimation du matériel

L'estimation du matériel est basée sur la synthèse comportementale, suivie de l'estimation au niveau transfert des registres (RTL).

La synthèse comportementale est nécessaire pour l'estimation car nous ne disposons pas d'informations suffisantes sur la réalisation, au niveau système, pour permettre une estimation représentative [57]. Au niveau système il manque des informations primordiales sur l'ordonnancement des opérations et sur l'allocation des ressources matérielles. Le niveau transfert de registres permet une estimation assez significative, capable de valider la réalisation ou de guider l'utilisateur vers de nouvelles alternatives.

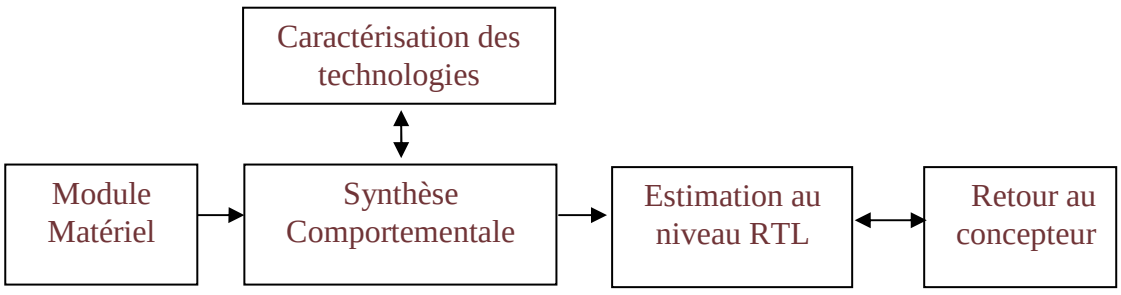

Figure 3.2: Flots d'estimation des modules matériels

L'estimation, basée sur la synthèse comportementale de la partie matérielle ne provoque pas une augmentation exagérée du temps de conception. La première raison est que, dans la méthodologie Cosmos, l'estimation est utilisée pour valider une réalisation, contrairement aux approches automatiques où l'estimation est utilisée par l'algorithme de découpage. La deuxième raison est que, actuellement, l'utilisateur dispose d'un système intégré contenant les outils Cosmos et les outils Amical au sein d'un seul environnement de synthèse. Cet environnement est appelé Music (*MUltilaguage Codesign for system on Chip*). Dans cet environnement, les processus matériels peuvent être synthétisés automatiquement après le découpage et la vérification des caractéristiques de la réalisation en tenant compte que la synthèse peut être faite dans le même environnement du travail.

L'estimation du matériel avec Amical donne à l'utilisateur une information permanente sur l'évolution du processus de synthèse. Les résultats calculés par l'estimateur comprennent les mesures de surface, de vitesse et de consommation du circuit [57].

Ce modèle d'estimation utilise en entrée trois types de spécifications:

- Une spécification comportementale, générée automatiquement et décrite en Solar ou VHDL. Une flexibilité dans la spécification permet la description des circuits complexes, avec des instructions du type boucles, branchements, appels de fonctions et appels de procédures;

- Une bibliothèque de fonctions externes (*Functional units*), qui contient des unités fonctionnelles standard et des unités fonctionnelles privées définies par l'utilisateur. La bibliothèque décrit les caractéristiques de chaque unité fonctionnelle. Une unité fonctionnelle peut réaliser plusieurs opérations, comme des opérations standard (addition, multiplication, logiques) mais aussi des opérations complexes programmées par l'utilisateur (unité d’entrée/sortie, mémoire cache). Les unités fonctionnelles sont utilisées dans la spécification comportementale et offrent la possibilité d’utiliser des circuits réalisés préalablement [57].

- Un fichier de technologie, qui spécifie des contraintes, les tailles des composants, les paramètres de consommation, et les paramètres de délai maximal des composants.

3.4.3.L'estimation de la communication

Cette section présente l'estimation du temps lié à la communication dans un système. Le temps de communication, ne prend pas en considération le temps d'attente lié à la synchronisation des processeurs. Nous supposerons que les processeurs et les données sont toujours disponibles pour la communication. C'est une grande simplification du modèle de communication [57].

La mesure du temps de communication permet à l'utilisateur de vérifier l'impact des décisions de raffinement sur la communication. Par exemple, l'utilisateur peut vérifier la relation entre le temps d'exécution d'un bloc du comportement et le temps pris par la communication, où il peut identifier des accès aux canaux de communication occasionnés par un découpage non optimisé du comportement d'un bloc.

Le taux des données échangées entre les processeurs peut être facilement calculée, à partir de la taille des données et la fréquence d'accès au canal. L'algorithme de calcul du temps de communication utilise le résultat du calcul dynamique de la fréquence d'exécution et les informations disponibles dans la description de chaque protocole.

Les protocoles de communication de la bibliothèque contiennent les paramètres suivants:

- le **coût de réalisation** du protocole est lié au coût intrinsèque de réalisation (complexité, surface, capacité).
- le **temps de transmission** des données correspond au temps nécessaire à une transmission simple sur le canal.
- le **débit moyen** du canal correspond au taux de transmission des données sur le canal.

Le calcul du temps de communication prend en considération plusieurs aspects:

- le **temps d'émission et de réception** (*ProcEmiss* et *ProcRecep*), qui fait référence au temps pris par la procédure d'émission pour injecter les données sur le réseau et le temps pris par la procédure de réception pour retirer les données du réseau;
- le **temps de transmission** (*prot.TempsStab*);

- le **temps du transfert**, qui est le temps nécessaire au message pour traverser le réseau.

Cette valeur est égale à la division de la taille du message (*donnée.Taille*) par le débit du réseau d'interconnexion (*prot.Débit*). Le débit de transmission de chaque protocole est spécifié dans la bibliothèque de communication [57].

3.5.Conclusion

Ce chapitre illustre la performance dans un système contenant des parties logicielles et des parties matérielles.

L'analyse des performances est une activité complexe. En effet, elle permet de déterminer un certain nombre de paramètres qui caractérisent les différentes parties du système, dans le but de guider l'ordonnancement et le partitionnement. Ces paramètres renseignent le concepteur sur la qualité des solutions trouvées, afin de prédire les résultats de la conception sans aller jusqu'à la réalisation totale. Il est donc important de disposer des outils d'analyse/estimation rapides et de choisir des paramètres suffisamment significatifs, afin de pouvoir comparer, avec suffisamment de précision, la qualité des solutions obtenues et donc de parcourir l'espace des solutions à la recherche de la meilleure.

On s’intéresse dans notre thèse sur le terme le plus important dans la performance c’est le temps d’exécution. Dans ce qui suit on donne une définition sur le temps d’exécution et aussi notre domaine de recherche dans le temps d’exécution.

CHAPITRE 4

ANALYSE DU TEMPS POUR LE SOFTWARE

4.1.Introduction

Dans le cadre de cette thèse, seul le critère de temps d'exécution est considéré. Ainsi le terme estimation de performance sera utilisé dans le reste de ce document pour designer l'estimation du temps d'exécution. D'autre part peu de travaux visant l'analyse de performance en vue de l'exploration d'architectures pour la conception conjointe matérielle/logicielle existent dans la littérature.

Nous ne nous intéressons ici qu'a la partie de l'analyse temporelle qui concerne l'estimation du temps d'exécution au pire cas (WCET), et non à son utilisation par les tests d'ordonnançabilité.

4.2.Estimation par test et mesure - analyse dynamique

Les techniques d'analyse dite dynamique sont fondées sur l'observation du comportement dynamique du programme analyse (exécution ou simulation) pour en estimer le WCET.

4.2.1.L'étape de spécification

Ces techniques se basent sur les données d'entrée du programme (et non sur sa structure) pour prédire son temps d'exécution au pire cas. A partir d'un test, c'est-à-dire d'un jeu de données d'entrée provoquant un comportement particulier du programme, on mesure son temps d'exécution. Cette mesure peut être effectuée de différentes manières.

(i) L'utilisation de matériel spécialisé tel qu'un analyseur logique permet de compter le nombre de cycles nécessaires à l'exécution d'un code en environnement réel. Cette technique a l'avantage de ne pas être intrusive.

(ii) Certaines architectures (l'Intel Pentium par exemple,) comportent des registres de débogage et sont programmables pour pouvoir compter différents types d'évènements, et en particulier les cycles processeur.

(iii) Si on ne dispose pas d'un moyen de mesurer précisément le temps, on peut effectuer des mesures moins précises sur un grand nombre de répétitions du même test, puis en établit

une moyenne. La technique dite de "dual loop benchmark"[74] consiste à mesurer le temps d'exécution d'une boucle contenant le code sous test ainsi que le temps d'exécution de la même boucle à vide pour corriger la première mesure du coût de la gestion de boucle et du code d'instrumentation.

(iv) Enfin, il est possible d'exécuter le code dans un environnement simulé. Les simulateurs d'architectures matérielles permettent d'obtenir de nombreuses informations pendant l'exécution du code, et, entre autres, le nombre de cycles. Cette technique nécessite un simulateur de processeur très précis non seulement au niveau fonctionnel, mais aussi au niveau temporel.

En théorie, le test dynamique est approprie pour l'analyse de WCET de code. Le temps d'exécution du programme est mesuré pour une exécution avec des données d'entrée particulières. Un outil de test idéal exécutera le programme pour chaque ensemble possible de valeurs de ces données d'entrée et mesurera le temps d'exécution. Mais ceci n'est pas possible en pratique car la complexité des problèmes à traiter conduit à un nombre de tests possibles extrêmement grand (problème d'explosion combinatoire). Il faut donc réduire l'espace de recherche en ne considérant qu'un sous-ensemble des tests possibles.

Peu de méthodes d'analyse dynamique pour calculé le temps d'exécution WCET sont présentée dans la littérature.

4.3.Estimation analytique: l'analyse statique

Cette deuxième catégorie regroupe les techniques basées sur l'analyse statique de programmes c'est notre axe de recherche.

Dans le contexte de l'estimation de WCET, on parle de techniques d'analyse statique de WCET. Le but de ces techniques d'estimation du WCET est de s'affranchir du test pour l'estimation du WCET. La connaissance du code du programme est nécessaire pour pouvoir l'analyser statiquement, c'est-à-dire sans l'exécuter ni simuler une exécution particulière avec un jeu de données.

Contrairement aux méthodes par tests et mesures pour lesquelles le résultat de l'estimation est au plus égal au WCET réel mais peut être inférieur, le résultat d'une analyse statique de WCET est au moins égal au WCET réel et peut être supérieur (en pratique, il l'est toujours). En effet, l'analyse statique de WCET cherche à calculer un majorant du WCET réel, elle est donc sure, mais pessimiste. La qualité des techniques d'analyse statique de WCET est

d'autant plus grande que le facteur de surestimation (ou pessimisme) qui affecte les résultats est faible.

Un autre atout des techniques d'analyse statique de WCET est la possibilité de les inclure plus tôt dans le processus de développement du processus du Codesign.

En effet, ces techniques permettent d'analyser du code pendant sa phase de développement et d'utiliser les résultats d'analyse pour l'améliorer défis pour l'obtention des WCET.

❑Méthodes dynamiques (tests explicites)

- Temps mesuré □ WCET
- WCET sous-estimé: le test de faisabilité peut accepter des ensembles de tâches non faisables

➨Non sûr

❑Méthodes statiques

- Temps estimé □ WCET
- WCET surestimé: le test de faisabilité acceptera seulement des jeux de tâches faisables

➨Sûr

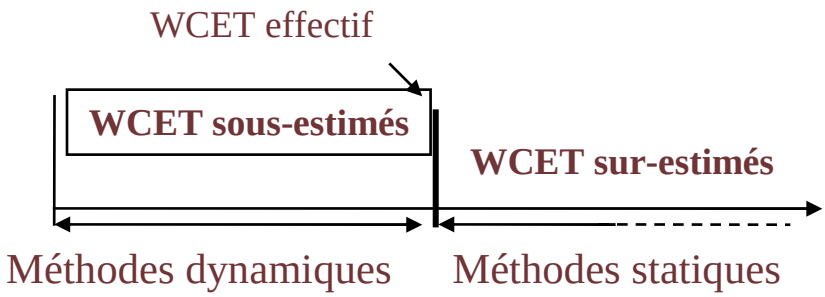

Figure 4.1: Comparaison entre les méthodes dynamiques et statiques

Les estimations de WCET doivent être sûres (WCET > tout temps d'exécution possible),dans la suite de ce document, nous ne nous intéresserons qu'à cette classe de technique d'estimation du WCET.

4.3.1. Les données de l'analyse statique de WCET

Comme on l'a vu précédemment, l'analyse statique de WCET analyse le code des programmes.

Nous allons maintenant présenter les différents types de code analyses et quelles sont les autres informations nécessaires à l'analyse. Puis nous décrivons la mise en forme de ces informations.

4.3.2.Langages sources

Pour pouvoir analyser statiquement le WCET d'un programme, celui-ci doit être écrit dans un langage analysable statiquement. Nous présentons ici les caractéristiques de langages analysables et les modifications à apporter à un langage pour le rendre analysable. Nous discutons aussi de l'influence des compilateurs optimisant sur l'analyse statique de WCET.

4.3.2.1.Type de langage analyse

Le code analysé peut être de plus ou moins haut niveau; il peut imposer différente structuration, ou encore introduire des abstractions. Nous allons étudier plusieurs possibilités de langages sources, utilisés dans diverses techniques d'analyse de WCET, en commençant par le langage source de plus bas niveau.

Le langage source de plus bas niveau analysable statiquement est le code assembleur lie à l'architecture dans lequel est codé le programme à analyser, soit directement (encore souvent le cas pour les systèmes embarques), soit après compilation d'un langage de plus haut niveau. Le code assembleur est également utile pour l'analyse de bas niveau.

Pour obtenir plus d'informations, il est également possible d'analyser le code source d'un programme écrit dans un langage de plus haut niveau (e.g. C, ADA, java etc.). Un tel langage introduit la notion de structure syntaxique et permet ainsi la construction d'une représentation du programme par arbre syntaxique qui est nécessaire pour l'analyse à base d'arbre.

L'analyse d'un langage de programmation à objets qui introduise d'autres abstractions telles que l'héritage, le polymorphisme, etc.

L'analyse statique de WCET de ce type de langage, étudiée dans [75], soulève plusieurs difficultés. Tout d'abord, le code résultant de la compilation de ce type de langage est trop dynamique. Un exemple est la possibilité de redéfinir des méthodes définies dans les classes parentes. Cela revient à dire que lorsqu'on exécute une méthode d'un objet on ne sait pas quel code sera vraiment exécuté. Un des derniers langages étudiées dans le cadre de l'analyse

statique de WCET est le langage Java, ou plus précisément le byte code Java (noté JBC, pour Java Byte Code). Java est un langage objet, on retrouve donc les problèmes liés au dynamisme de ces langages présenté précédemment.

Le JBC est un langage de programmation de machine à pile, et peut être obtenu par compilation d'un langage de plus haut niveau, dont Java (mais aussi ADA, C, etc.).

Le JBC est une représentation intermédiaire entre un code source de haut niveau et sa traduction en assembleur proche de l'architecture. Le but de l'analyse de WCET de JBC [76] [77] est d'être portable tout en restant précise grâce à une prise en compte de l'architecture matérielle sur laquelle il est porté. La portabilité est assurée par l'utilisation d'un modèle temporel dépendant de l'architecture cible. On dispose d'un modèle temporel pour chacune des architectures sur lesquelles peut s'exécuter le programme en JBC. Le WCET d'un programme est décrit de manière unique et indépendante de l'architecture. La combinaison de ce WCET portable et d'un modèle temporel d'architecture donne le WCET du programme sur cette architecture.

4.3.2.2.Restriction du langage source

Afin d'être analysable, le langage source choisi doit posséder certaines propriétés permettant son analyse statique. Si ces propriétés ne sont pas intrinsèques au langage, celui-ci doit être restreint pour respecter ces propriétés et ainsi permettre son analyse statique dans le but d'estimer son WCET.

La première propriété que doit respecter le code analyse est de permettre à l'analyseur de connaître statiquement tous les chemins d'exécution possibles. Pour ce faire, les branchements dynamiques (e.g. appel de fonction par pointeur) sont interdits, car ils introduisent de nouveaux chemins d'exécution dynamiquement. Dans le même but, on interdit souvent la récursivité (même indirecte) car elle introduit des chemins d'exécution de longueurs potentiellement infinies. Il est cependant possible de la prendre en compte pour l'analyse statique si on est capable de borner sa profondeur statiquement.

Ces premières restrictions s'appliquent quels que soit le langage analyse. On s'intéresse maintenant à un langage de haut niveau dont on peut extraire la structure syntaxique. Un tel langage nous permet de représenter le programme par un arbre syntaxique.

Si on souhaite pouvoir analyser statiquement le programme en utilisant conjointement les deux niveaux du langage source (langage de haut niveau et code objet), il faut que leurs représentations (l'arbre syntaxique et le graphe de flot de contrôle) soient en correspondance directe. Pour assurer cette correspondance entre les deux représentations, il est nécessaire de

restreindre l'expressivité du langage source. On cherche à avoir un langage de haut niveau ne permettant d'écrire que des programmes bien structurés, c'est a dire dont tous les chemins d'exécution apparaissent dans l'arbre syntaxique. Pour ce faire, on doit restreindre le langage afin de supprimer les possibilités de faire diverger le graphe et l'arbre. C'est pourquoi on peut interdire l'utilisation des commandes permettant des branchements non structurés tel que les GOTO, EXIT et CONTINUE du langage C, et parfois la sortie multiple des fonctions ou procédures.

4.3.2.3.Informations complémentaires

Le but de ces informations est de fournir à l'analyseur des informations sur le comportement dynamique du code à analyser. Ces informations permettent de restreindre le nombre de chemins d'exécution possibles, et donc le nombre de chemins à prendre en compte pour l'analyse. Ces informations sont le plus souvent utilisées pour:

1. Borner le nombre maximum d'itérations des boucles. Le WCET d'une boucle dépend non seulement de son code mais aussi de son pire nombre d'itérations. Cette information doit donc être associée aux boucles, le plus souvent sous forme d'une constante [74].

2. Contraindre le choix d'une branche dans une structure conditionnelle. Par exemple quand le choix d'une branche conditionnelle dépend d'un paramètre qui est constant.

3. Restreindre le nombre de chemins d'exécution possibles en indiquant les chemins infaisables. Par exemple, les informations complémentaires peuvent indiquer qu'une partie du code ne peut faire partie du pire chemin d'exécution (code mort ou cas d'erreur), ou encore, pour indiquer que deux branches s'excluent mutuellement.

Un exemple simple est:

if(x=0) then A

else B;

if (x>0) then C;

La valeur de la variable x implique que les branches A et C sont mutuellement exclusives.

On distingue deux possibilités pour obtenir ces informations. La première fait appel a l'utilisateur (i.e. le programmeur) qui doit fournir ces informations en les ajoutant au code source sous forme d'annotations [85], ou interactive ment [86]. La plupart des travaux concernant l'analyse statique de WCET utilisent cette méthode.

La deuxième possibilité [84] [75] permet dans certains cas d'obtenir automatiquement les bornes sur les nombres d'itérations des boucles en analysant le code et les données manipulées.

Bien que ces techniques soient efficaces pour des boucles relativement simples, l'annotation des boucles les plus complexes reste à la charge du programmeur.

Enfin, l'utilisation de techniques de preuves formelles a été envisagée dans [85] pour vérifier les annotations fournies par l'utilisateur.

4.3.2.4. Représentations logiques

Une fois le programme écrit dans un langage analysable, on cherche à construire sa représentation logique.

Le code objet est tout d'abord scindé en plus petites unités, les blocs de base. Puis, suivant le niveau du langage et les informations souhaitées, on va représenter les fonctions du programme par des graphes de flot de données contrôlé ou des arbres syntaxiques.

Enfin, les représentations des fonctions du programme sont assemblées pour ne former qu'une unique représentation logique du programme.

4.3.2.4.1.Décomposition du programme en blocs de base

Presque toutes les méthodes d'analyse statique de WCET utilisent le découpage du code

Objet en blocs de base. Un bloc de base est une suite d'instructions purement séquentielle ne contenant qu'un seul point d'entrée et un seul point de sortie.

- Un seul point d'entrée signifie que si une instruction de branchement hors du bloc de base effectue un saut vers une instruction dans le bloc de base, c'est obligatoirement la première instruction qui est visée.
- Un seul point de sortie signifie que les branchements d'un bloc de base sont soit des branchements inconditionnels vers l'instruction suivante dans le bloc de base, soit la dernière instruction du bloc.

L'exemple suivant va présenter le code C d'un programme constitué de deux fonctions.

La fonction principale main comporte une boucle, et pour chaque itération de la boucle la fonction impaire est appelée deux fois. Cette deuxième fonction comporte une structure conditionnelle.

Les deux fonctions constituant ce programme nous servit dans tous les exemples de ce chapitre.

```
int impaire(int x) {
int result;
if (x % 2) {
result = 1;
} else {
result = 0;
}
return(result);
}
```

```
void main() {
int tab[4] = {34,45,12,5};
int nb_impaires = 0;
int nb_paires = 4;
int i;
for(i=0;i<4;i++) {
nb_impaires +=
impaire(tab[i]);
nb_paires -= impaire(tab[i]);
}
```

Figure 4.2: Exemple de code source d'un programme analysée (langage de haut niveau)

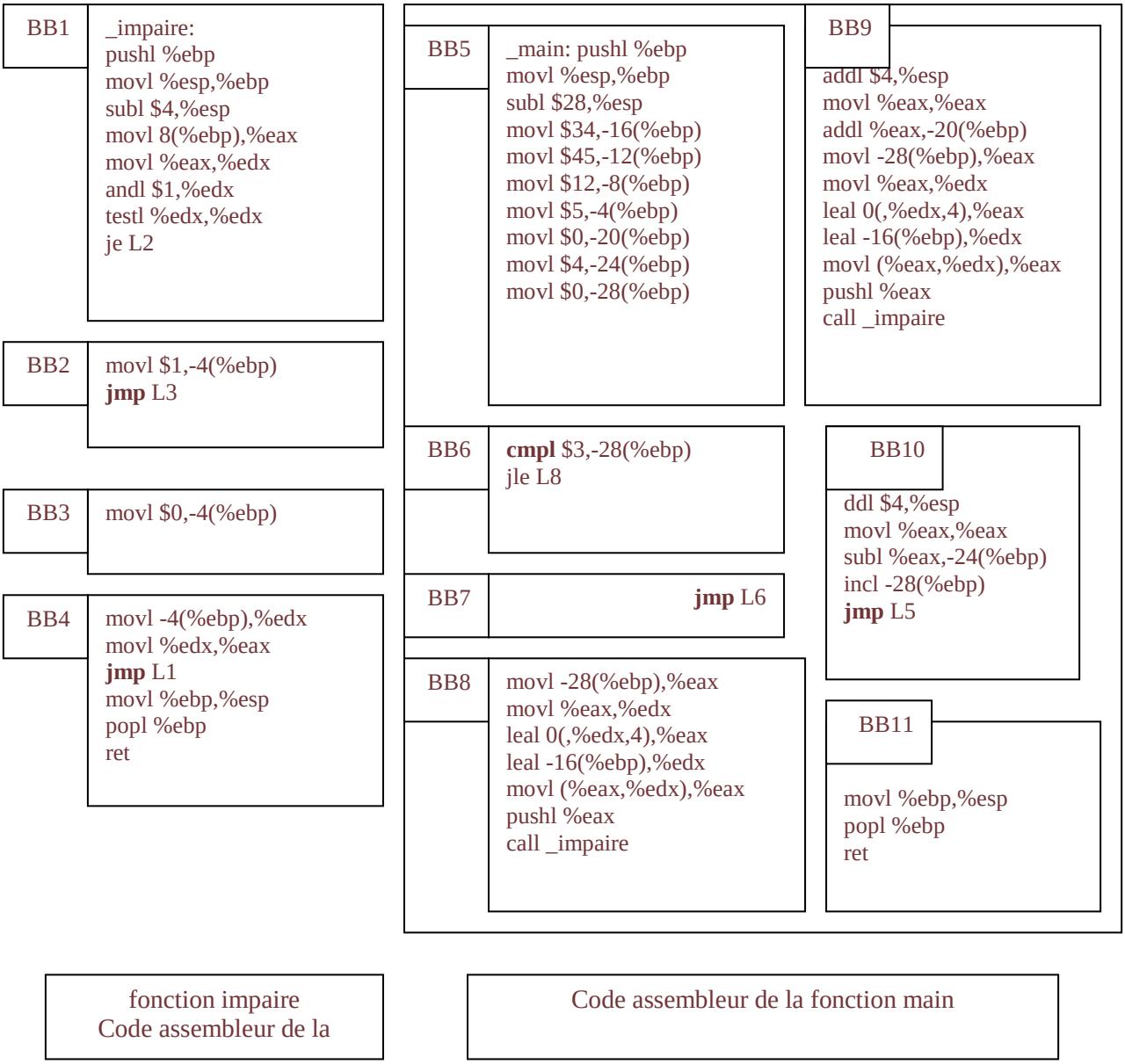

Figure 4.3:Exemple de code source d'un programme analysé (langage assembleur)

Ce programme, une fois compilé sans optimisations, le code assembleur est présenté ci-dessus. Les blocs de base sont décrits par les zones de couleurs différentes sur la figure.

La construction des blocs de base par regroupement d'instructions obéit aux règles précitées: instructions qui se suivent, un seul point d'entrée, et un seul point de sortie. Notons que le bloc de base BB4 contient deux branchements, ce qui est possible car le premier branchement saute vers l'instruction suivante de façon inconditionnelle. Les blocs de base

BB1 à BB11 ainsi obtenus sont les éléments de base des deux représentations décrites dans les paragraphes suivants.

4.3.2.4.2. Le graphe de flot de contrôle

Un graphe de flot de contrôle décrit tous les enchaînements de blocs de base possibles pour chaque fonction du programme. Il y a un graphe de flot de contrôle par fonction. Il est obtenu soit par analyse statique du code bas niveau en utilisant un compilateur modifié [83], soit par un outil dédié à la manipulation du code bas niveau comme par exemple SALTO [82].

On distingue deux types de graphes équivalents et couramment utilisés:

La première catégorie est celle des graphes dont les nœuds sont des blocs de base et ou les arcs représentent les relations de précédences entre blocs. On peut alors distinguer deux types d'arc: les arcs \pas de saut" qui représentent l'exécution de deux blocs qui se suivent sans saut (absence de branchement, ou branchement non pris) et les arcs \saut" qui représentent les branchements pris (e.g. arc de BB10 à BB6). Cette première catégorie de graphe de flot de contrôle est utilisée dans de nombreux travaux d'analyse statique basés sur les graphes [80] [81].

La deuxième catégorie de graphe, baptisée T-graph par P. Puschner [78], est une représentation duale de la première. Les arcs y représentent les blocs de base et sont values par le WCET de ces blocs, et les nœuds du graphe représentent les points dans le code ou le flot de contrôle du programme converge et/ou diverge, cette représentation est utilisée dans les travaux [78] et [79].

Le graphe de flot de contrôle est le plus souvent construit par analyse statique de code de bas niveau: du code assembleur le plus souvent, mais aussi du byte code [76].

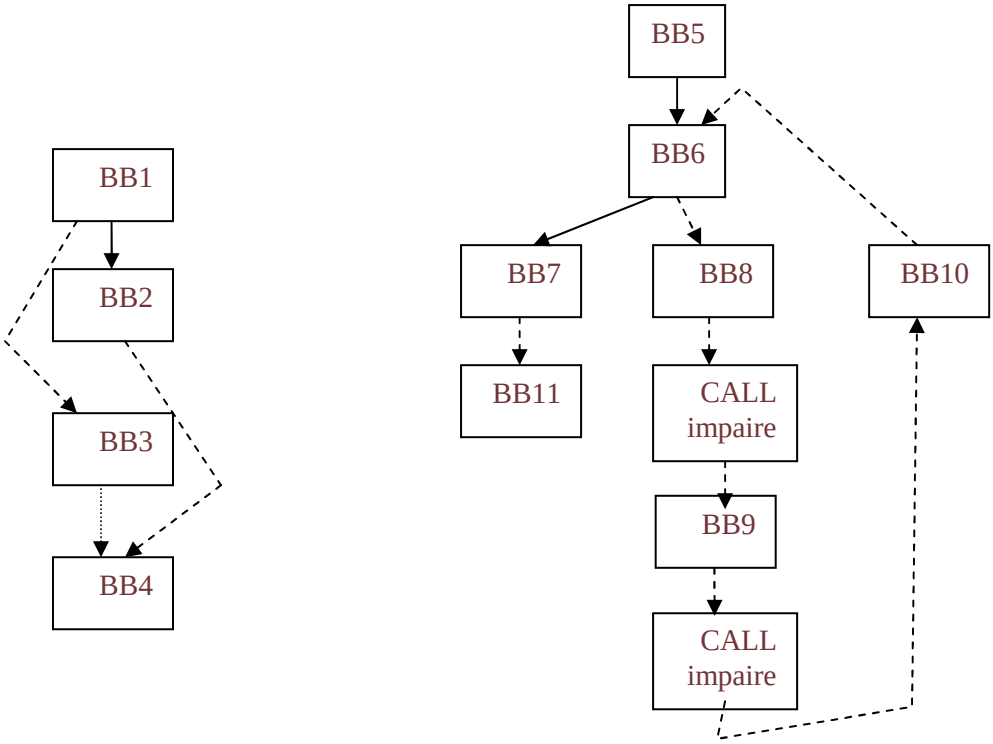

Figure 4.4:flot de données de l'exemple

4.3.2.4.3.L'arbre syntaxique

Un arbre syntaxique est un arbre dont les nœuds représentent les structures du langage de haut niveau et dont les feuilles sont les blocs de base.

Une représentation simple du programme par un arbre syntaxique peut être basée sur trois types de nœuds et deux types de feuilles.

- Les nœuds de type SEQ possèdent au moins un fils. Ils représentent la mise en séquence de leurs sous arbres fils.
- Les nœuds de type LOOP ont deux fils: le sous arbre test et le corps de la boucle. Le nombre d'itérations autorisé est borné.
- Les nœuds de type IF sont constitués d'un sous arbre test et de deux sous arbres " then" et " else".
- Les feuilles de type CALL représentent des appels de fonction.
- Enfin, les autres feuilles de l'arbre syntaxique sont les blocs de base du programme.

On peut imaginer d’autres types de nœuds pour représenter par exemple différents types de boucle.

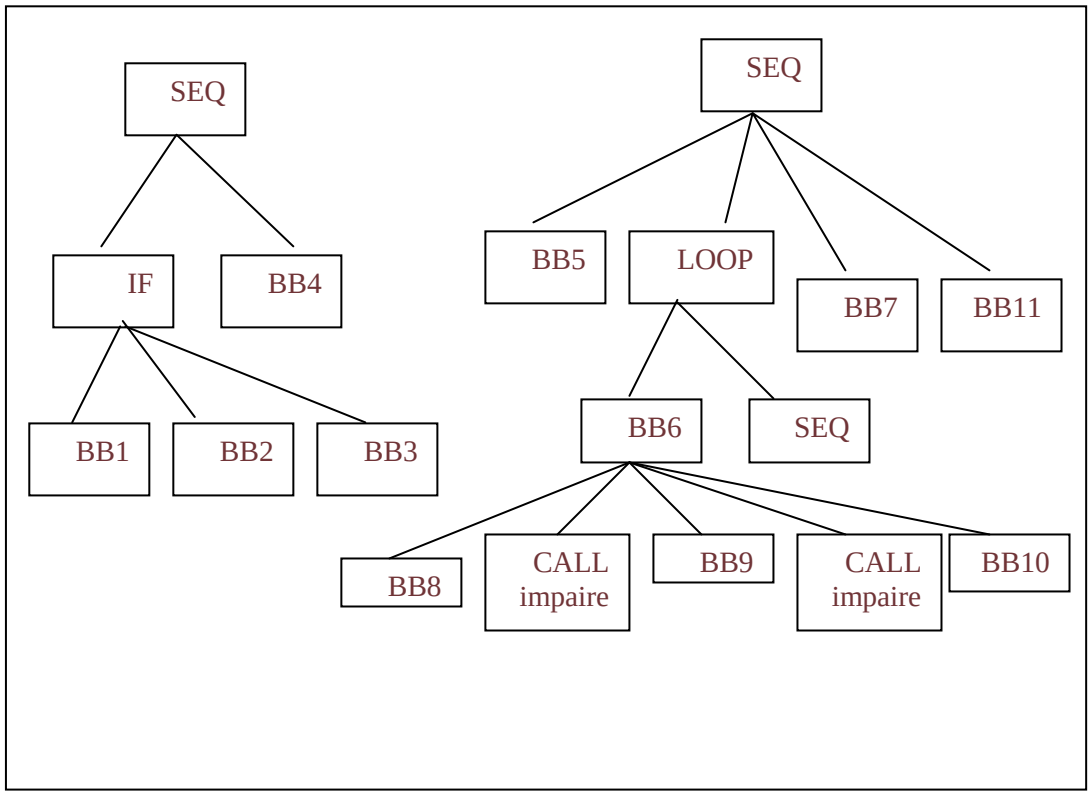

Figure 4.5:représentant l'arbre syntaxique du l'exemple

L'arbre syntaxique est obtenu par analyse statique d'un langage de haut niveau. Les feuilles de l'arbre syntaxique coïncident avec les nœuds du graphe de flot de contrôle décrit au paragraphe précédent. Les structures du langage (séquence, conditionnelle, boucles) représentées dans l'arbre syntaxique induisent des relations de précédence entre leurs fils comme le montre la figure 4.5.

4.4.L'analyse statique de WCET de haut niveau

Une fois la représentation du programme à analyser est obtenue, on va chercher le pire chemin d'exécution dans cette représentation, c'est à dire le plus long chemin dans le graphe de flot de contrôle, ou encore le plus long parcours de l'arbre syntaxique correspondant à une exécution possible.

La méthode utilisée pour la recherche du plus long chemin permet d'identifier les différentes méthodes d'analyse statique de WCET. On suppose ici que les WCET élémentaires des blocs de base sont connus, dans un premier temps, on pose deux hypothèses simplificatrices.

- H1: On suppose que le WCET d'une séquence de deux blocs de base est égal à la somme des WCET des blocs de base pris séparément.

- H2: On suppose que le WCET d'un bloc de base est constant quelque soit son contexte d'exécution (i.e son WCET ne dépend pas de l'enchaînement du blocs de base le précédant dans un chemin d'exécution).

Ces hypothèses simplificatrices, qui sont à la base des premiers travaux du domaine, nous permettent de ne pas nous préoccuper pour l'instant du niveau bas de l'analyse statique de WCET [5].

4.4.1.Techniques utilisant l'algorithmique des graphes (Path-based)

La connaissance des WCET individuels des blocs de base permet d'associer des WCET aux nœuds du graphe de flot de contrôle, ou aux arcs du T-graph. On obtient alors un graphe valué avec un seul point d'entrée et un seul point de sortie. On peut chercher le pire chemin d'exécution dans le graphe de flot de contrôle [81] en utilisant les algorithmes traditionnels de l'algorithmique des graphes (e.g. algorithme de Dijkstra [57]) recherchant le plus long chemin dans un graphe. Puis on vérifie que le chemin trouvé est un chemin d'exécution possible, et si ce n'est pas le cas, on l'exclut du graphe et on recommence la recherche. Les informations du nombre maximum d'itérations limitent le nombre d'occurrences d'un bloc de base ou d'un arc dans un chemin d'exécution.

4.4.2.Techniques IPET

Cette méthode, dite d'énumération implicite des chemins (note IPET - Implicite Path Énumération Technique) est utilisée dans de nombreux travaux d'analyse statique de WCET [102, 101, 79]. Cette approche ne s'appuie que sur la représentation du programme sous forme de graphe de flot de contrôle, qu'elle transforme en un ensemble de contraintes devant être respectées. Un premier jeu de contraintes décrit la structure du graphe, et un deuxième permet de prendre en compte les informations de bornes des boucles.

L'ensemble des contraintes a pour rôle d'éliminer les chemins infaisables, et de restreindre l'ensemble des chemins possibles. On cherche ensuite à maximiser l'expression du WCET en respectant les deux jeux de contraintes. Les techniques IPET consistent donc en une énumération implicite des chemins d'exécution susceptibles de maximiser le WCET.

Les techniques de calcul IPET ne font appel qu'au graphe de flot de contrôle du programme. Comme l'arbre syntaxique n'est pas utilisé conjointement au graphe de flot de contrôle.

En levant cette restriction, les méthodes IPET permettent d'analyser plus de programmes que les méthodes à base d'arbre présentées ci-après. De plus, elles permettent d'ajouter d'autres contraintes que celles liées aux boucles pour, par exemple, prendre en compte l'exclusion mutuelle de deux blocs de base.Cette possibilité est exploitée par exemple dans [101].

La maximisation de l'expression du WCET ci-dessus fournit les valeurs des ni (et évidement le WCET). Les méthodes de résolution utilisées sont similaires a celles utilisées pour résoudre les problèmes de programmation linéaire (Integer Linear Programming) ou de satisfaction de contraintes. Le problème standard de la programmation linéaire entière (ILP) est la recherche de l'extremum d'une fonction linéaire à plusieurs variables, ces variables étant assujetties à un ensemble de contraintes fonctionnelles (i.e. elles doivent vérifier un système d'équations et/ou d'inéquations). Un tel problème peut être résolu en utilisant un outil tel que « lp solve » (Université de Technologie d'Eindhoven), ILOG CPLEX (société ILOG) ou encore Maple [100].

Ces méthodes comportent toutefois un inconvénient majeur: la complexité de la résolution du système impose parfois des temps d'analyse importants (on parle de quelques secondes à plusieurs heures dans [94]).

A noter, les récents travaux de P. Puschner [127] sur l'énumération implicite des chemins dans un arbre syntaxique. Il s'agit alors d'appliquer les mêmes techniques (traduction en système de contraintes, puis résolution) à un arbre syntaxique au lieu d'un graphe de flot de contrôle.

4.4.3.Techniques basées sur l'arbre syntaxique (Tree-based)

Cette classe de méthodes, proposée initialement par P. Puschner et C. Koza dans [124], s'appuie sur la représentation du programme en arbre syntaxique pour calculer récursivement son WCET. Un ensemble de formules nommé schéma temporel (timing schéma), permet d'associer à chaque structure syntaxique du langage source (un nœud de l'arbre syntaxique) un WCET, et ce en fonction des sous- arbres qui la composent (les fils du nœud).

Et ainsi de suite, les WCET des sous- arbres étant eux-mêmes calculés à partir des WCET de leurs fils jusqu'à arriver aux feuilles de l'arbre que sont les blocs de base dont on suppose les WCET connus.

On effectue donc un parcours de bas en haut (bottom-up) de l'arbre en partant des feuilles porteuses de l'information de WCET, pour obtenir le WCET de la racine.

Type	Formules pour le calcul du WCET: W(S)
S = S1;:::; Sn	WCET(S) = WCET (Si) +::: +WCET (Sn)
S = if (test) then S1 else S2	WCET(S) =WCET (Test) +max (WCET (S1); WCET (S2))
S = loop (tst) S1	WCET(S) =maxiter (WCET (Test) +WCET (S1)) +WCET (Test) où maxiter est le nombre maximum d'itérations.
S = f (exp1; _ _ _; expn)	W(S) = W (exp1) + _ _ _ +W (expn) +W (f ())
S = bloc de base BBx	WCET(S) = WCET du bloc de base BBx

Tableau 4.1:Formules de calcul du WCET [124]

A chaque nœud de l'arbre où un choix est possible, on choisit le chemin qui maximise le temps d'exécution. Par exemple, le WCET d'une séquence est simplement la somme des WCET des structures qui la composent (cf. première ligne du tableau 2.1), et le WCET d'une conditionnelle implique l'utilisation de l'opérateur max pour choisir la branche conditionnelle dont le temps d'exécution est le plus important.

4.4.4.Comparaison des techniques d'analyse de haut niveau

Les méthodes tree-based se distinguent des autres méthodes (IPET et graph-based) par l'utilisation de l'arbre syntaxique au lieu du graphe de flot de contrôle. Les principales différences qui en découlent sont:

- l'obligation d'analyser des programmes bien structurés.
- la difficulté à ajouter des contraintes sur les chemins d'exécution.
- les avantages liés à l'utilisation d'une représentation hiérarchique (emboîtement de boucles, blocs conditionnels, etc.).
- la possibilité d'utiliser un système d'annotations des boucles basées sur leur emboîtement.

- la connaissance du rôle de chaque branchement (test de boucle, test conditionnelle, etc.).
- la possibilité d'utiliser le calcul symbolique pour l'estimation du WCET.

Les méthodes IPET et graph-based utilisent le graphe de flot de contrôle. La principale différence entre ces deux classes de méthodes est que les méthodes IPET ne cherchent pas le pire chemin d'exécution mais donnent le pire nombre d'exécutions de chacun des nœuds du graphe de flot de contrôle. Elles ne donnent pas d'information précise sur l'ordre d'exécution.

Les approches path-based, quant à elles, calculent explicitement le plus long chemin d'exécution dans le programme, mais elles requièrent de connaître le nombre d'exécutions maximum des nœuds avant de commencer la recherche.

4.5.L'analyse statique de WCET de bas niveau

Les hypothèses simplificatrices H1 et H2 ne sont vérifiées que si on ignore l'effet de certains éléments de l'architecture matérielle qui permettent d'améliorer les performances moyennes du système (par exemple les caches et pipelines). En effet, le cache d'instruction introduit une variation du temps de traitement d'une instruction en fonction du contexte, ce qui contrarie l'hypothèse H2 de temps de traitement constant. De même, le pipeline, en introduisant du parallélisme dans le traitement d'une suite d'instructions, remet en cause l'hypothèse H1 de composition par simple somme du WCET des séquences de blocs de base.

Si on considère que l'effet de ces éléments d'architecture est nul (hypothèses H1 et H2), le WCET obtenu est une approximation très pessimiste du WCET réel. Par exemple, F.Mueller fait état dans [105] d'un facteur de surestimation des WCET allant de 4 à 9 pour les programmes de son jeu de test sans prise en compte du cache d'instructions, facteur qui est ramené entre 1 et 2 par sa méthode de prise en compte. Ce pessimisme de l'estimation entraîne une sous-utilisation importante du matériel. La prise en compte de ces éléments permettrait de réduire cette sous-utilisation et donc d'améliorer le taux d'utilisation d'un système temps réel strict sans modification du matériel.

La modélisation du comportement de ces éléments d'architecture n'est pas triviale, et leur prise en compte introduit une dépendance au contexte. Par exemple, la durée d'exécution d'une instruction dans le pipeline dépend des instructions précédentes, les durées des accès mémoire en lecture et écriture dépendent de l'état des caches et donc de l'historique de l'exécution (instructions et données accédées). Si on veut lever les hypothèses trop pessimistes

H1 et H2, il faut adapter les méthodes de calcul du WCET des instructions et donc des blocs de base [5].

Malgré ces difficultés, l'intégration de ces éléments dans l'estimation du WCET permet d'améliorer considérablement sa précision tout en garantissant la sûreté des estimations.

C'est pourquoi de nombreux travaux concernent la prise en compte de ces éléments, principalement les caches [103] [101] et les pipelines [104] et c'est le plus souvent pour des architectures de type RISC.

Dans les paragraphes qui suivent, nous donnons une vue d'ensemble des méthodes de prise en compte de l'effet de l'architecture matérielle sur le WCET. Les deux éléments les plus couramment considérés pour l'analyse statique de WCET sont le cache d'instructions et le pipeline.

4.5.1.Prise en compte des caches d'instructions

Les évolutions techniques récentes ont eu pour effet une augmentation considérable de la différence entre les vitesses des processeurs et celles des mémoires. La mémoire est devenue un élément limitant les performances de l'architecture. Or, les mémoires rapides sont d'un coût trop élevé pour que l'on puisse en disposer d'une quantité importante. Une solution à ce problème est l'installation d'une hiérarchie de caches organisée en plusieurs niveaux. Le niveau le plus bas, le niveau 1, est rapide mais de taille réduite. Quand un bloc de données auquel on veut accéder est absent (défaut de cache), il y est recopie depuis le niveau supérieur.

Grâce à ce mécanisme on dispose d'une quantité importante de mémoire presque aussi rapide que le cache à un coût presque aussi faible que celui de la mémoire centrale.

L'usage d'une mémoire cache introduit de l'indéterminisme dans l'architecture, car en ajoutant cet intermédiaire entre la mémoire centrale et le processeur, la durée des accès mémoire n'est plus constante. En effet, deux durées correspondant à deux cas d'accès au cache sont à considérer. Une première durée correspond à l'accès à un bloc mémoire qui est dans le cache (succès, ou hit). Une deuxième durée correspond au cas ou le bloc mémoire référence est absent du cache (échec, ou miss), il est alors recopie depuis le niveau supérieur. La durée d'un accès miss est évidemment bien plus importante que celle d'un accès hit.

Il faut donc pouvoir estimer de façon sure le résultat (hit/miss) des accès au cache et pour cela connaître statiquement le pire comportement des accès mémoire vis à vis du cache lors de l'exécution d'un programme. Ainsi, si on peut assurer qu'un accès mémoire sera un succès, on estime que le résultat est hit, et s'il y a un doute on garde l'estimation la plus pessimiste: miss.

Le cache d'instructions est un cas particulier pour deux raisons. D'une part, les seuls accès qui le concernent sont des accès en lecture car il ne contient que des instructions. D'autre part, toutes les références mémoire (i.e. les adresses des instructions) peuvent être connues statiquement, il suffit pour cela de connaître l'adresse d'implantation du programme en mémoire.

Plusieurs méthodes ont pour but d'intégrer le comportement du cache d'instructions dans le calcul du WCET [99]. Leur objectif commun est d'effectuer une classification de toutes les instructions d'un programme en fonction de leur comportement pire cas par rapport au cache.

4.5.2.Prise en compte de l'exécution pipeline

Le pipeline est une technique dans laquelle plusieurs instructions se recouvrent au cours de leur exécution. C'est la technique fondamentale utilisée pour réaliser des processeurs rapides.

Afin d'expliciter brièvement le principe du pipeline, on rappelle que le cycle d'exécution d'une instruction se décompose en plusieurs étapes, par exemple: lecture d'instruction, décodage d'instruction, exécution, accès mémoire, écriture du résultat. Le but du pipeline est d'introduire du parallélisme entre les traitements de différentes instructions. Pour ce faire, il comporte plusieurs étages qui lui permettent de traiter en parallèle les différentes étapes des instructions.

Le parallélisme d'instruction n'est pas maximum car l'occupation du pipeline par une séquence d'instructions peut être perturbée par des aléas (hazard):

- dépendance de donnée (data hazards) quand une instruction utilise le résultat d'une instruction précédente dont l'exécution n'est pas encore terminée.
- aléa de contrôle (control hazards) lorsque le résultat d'une instruction est un branchement pris (rupture de séquence par un saut).
- aléa de structure (structural hazards) lorsqu'il y a un conflit d'accès à une ressource entre plusieurs instructions dans différents étages du pipeline.

Les effets du pipeline sur le calcul du WCET se retrouvent à deux niveaux:

- inter blocs de base pour la prise en compte du chevauchement entre instructions et des aléas de données.
- inter blocs de base ou les aléas de contrôle sont pris en compte.

4.5.3.Influence sur le WCET des blocs de base

De par la définition d'un bloc de base, on sait qu'aucun aléa de contrôle ne peut avoir lieu (pas de rupture de séquence), mais la présence de dépendances de donnée et d'aléas de structure est possible. Le calcul du WCET des blocs de base présenté dans [98] repose sur la représentation de l'occupation du pipeline pendant l'exécution de ce bloc. Connaissant les occupations des étages du pipeline par les instructions, un ajustement est effectué afin de prendre en compte les aléas; la table de réservation ainsi obtenue fournit le WCET du bloc de base.

4.5.3.1.Effet inter bloc de base

À cause des dépendances de donnée et des aléas de contrôle, la durée d'exécution d'un bloc de base dépend du bloc exécute avant. Dans l'hypothèse de l'absence de tout mécanisme de prédiction de branchement, le calcul du WCET global par composition des WCET des blocs de base est réalisé de la manière suivante.

Si le long du chemin d'exécution considère, deux blocs de base en séquence se suivent réellement dans le programme (instructions contiguës en mémoire), alors une composition des descripteurs d'occupation du pipeline par recouvrement permet un calcul plus précis du WCET. Cette composition a pour but de réduire la durée d'exécution estimée de la séquence constituée des deux blocs de base.

4.5.4.Prise en compte de quelques autres éléments d'architectur

Les travaux de recherche visant à prendre en compte l'effet de l'architecture matérielle sur l'analyse de WCET se sont principalement intéressés au cache d'instructions et au pipeline.

Mais d'autres aspects architecturaux ont été étudiés au nombre desquels: le cache de données [94] [95] [93], le mécanisme de prédiction des branchements [88] et les processeurs super scalaires [96] [97]. Nous allons maintenant décrire brièvement ces travaux.

4.5.4.1.Cache de données

La prise en compte du cache de données est plus complexe que celle du cache d'instructions, car d'une part il est difficile ou même parfois impossible de connaître statiquement la plupart des adresses des variables d'un programme, et d'autre part ce cache peut accéder aussi bien en lecture qu'en écriture. Plusieurs problèmes se posent donc: (i) déterminer pour quelles variables les adresses peuvent être connues statiquement, (ii) calculer leurs adresses lorsque c'est possible, et (ii) gérer les accès en écriture.

Identification statique des variables utilisées L'analyse de programme utilisée dans [94] permet de différencier les variables statiques (qui ont une adresse fixe) des autres variables, dites dynamiques (dont l'adresse n'est connue qu'a l'exécution).

Dans [93], une référence dynamique est toujours considérée comme un accès miss et induit deux pénalités de cache: une pour l'absence de cette référence dans le cache et une deuxième pour la prise en compte du préjudice éventuel cause par le remplacement d'un bloc mémoire présent dans le cache. Dans ce cas, la distinction entre variables statiques et dynamiques est très importante.

Une autre possibilité est de n'autoriser à être mises en cache, que les variables statiques. La solution proposée dans [95] permet de connaître statiquement les adresses d'un sous-ensemble des variables dynamiques: les variables locales (allouées dynamiquement dans la pile d'exécution). Cette méthode se base sur la simulation statique du niveau de la pile d'exécution pour estimer à tout moment l'adresse du sommet de pile est donc l'adresse des variables locales allouées en début de fonction.

4.5.4.2.Gestion des accès en écriture

Dans le cas du cache d'instructions, seules les lectures dans le cache ont été considérées. Dans le cas du cache de données les écritures dans le cache doivent être prises en compte. Les différentes techniques de gestion du cache de données lors des écritures (write-back, writethrough) induisent différents temps pour les accès en écriture (cf. [92]). Ces différents temps doivent être pris en compte par l'analyse statique de WCET.

4.5.4.3.Processeurs super scalaires

Une autre technique d'accélération consiste à disposer plusieurs pipelines en parallèle de Manière à augmenter le débit de traitement des instructions. La modélisation de l'exécution en parallèle de plusieurs instructions a été étudiée dans [91]. Les nouveaux problèmes poses par l'aspect super scalaire des processeurs sont: (i) l'appariement des instructions pour savoir quelles instructions peuvent être exécutées en parallèle, (ii) les dépendances structurelles entre instructions (conflits de ressource) et (iii) les dépendances de donnée.

4.5.4.4.Prédiction de branchement

Un des derniers éléments d'architecture à avoir été pris en compte est le mécanisme de prédiction des branchements [88] [89] qui permet d'augmenter encore le taux d'occupation des pipelines en réduisant le nombre de passages à vide dans le pipeline dus aux aléas de contrôle.

4.5.5. Récapitulatif

Nous avons présente dans ce chapitre un large panorama des techniques d'estimation du WCET de code pris en isolation. Les techniques d'analyse statique de WCET peuvent être différenciées selon cinq caractéristiques principales:

- Le niveau du langage source analyse (langage de haut niveau, langage objet).
- la méthode d'obtention des informations sur les chemins d'exécutions (annotations manuelles ou obtention automatique).
- la ou les représentations de programme utilisées (graphe de flot de contrôle, T-graph ou arbre syntaxique).
- la méthode d'énumération des chemins d'exécution (path-based, IPET ou tree-based).
- les éléments d'architecture pris en compte.

Les différentes techniques de prise en compte du matériel par l'analyse bas niveau peuvent être classées en deux catégories. La première est l'analyse bas niveau globale, qui considère l'impact des éléments d'architecture ayant un effet non local. C'est le cas par exemple des caches et la prédiction de branchement. La deuxième catégorie est l'analyse bas

niveau locale, qui s'intéresse aux éléments n'ayant qu'un effet local (i.e. sur une instruction et son voisinage immédiat). C'est le cas par exemple du pipeline.

Le tableau 4.2 dresse un panorama des principaux travaux de recherche concernant l'analyse de WCET dans le monde.

Auteur	**Référence**	**II**	**III**	**V**	**V**
Altenbernd	[106] [81]	A	I	P	P
Bernat	[107] [108]	M	-	P	T
Chapman	[85]	A,m	-	-	P
Colin	[88] [89]	M	I,p	P	T
Ferdinand	[109] [91]	M	I	P,s	I
Li	[112] [94]	M	I	P	I
Lim	[113] [114] [97]	M	I, d	P, s	T
Liu	[111]	A	-	-	T
Lundqvist	[110]	A	I	P	P
Park	[116] [115]	M	-	-	T
Persson	[123]	M	-	-	T
Puschner	[124] [117]	M	-	-	T, i
Whalley	[124] [119] [120]	A, m	I, d	P	P
Antoine	[5]	A, m	-		I
Engblom	[78] [118] [121] [122]	A, m	i	p	I

Tableau 4.2:panorama des principaux travaux de recherche concernant l'analyse de WCET

La colonne II indique la méthode d'obtention des informations complémentaires sur les chemins d'exécution (a = automatique, m = manuelle). La colonne III indique la listes des éléments architecturaux pris en compte par l'analyse bas niveau globale (i = cache d'instructions, d = cache de données, b = prédiction de branchement). La colonne IV liste les éléments architecturaux pris en compte par l'analyse bas niveau locale (p = pipeline simple, s = super scalaire).

Enfin, la colonne V indique la méthode de recherche du plus long chemin d'exécution utilisée (P = path-based, T = tree-based, I = IPET).

Toutes ces méthodes d'estimation du WCET basées sur l'analyse statique n'étaient, il y a quelques années encore, étudiées et envisagées avec intérêt que dans un milieu académique. Le monde industriel n'utilisait que des techniques à base de tests et mesures.

4.6.Autres méthodes d'analyse

Plusieurs chercheurs ont proposé un certain nombre d'approches pour caractériser I'exécution des logiciels [134].

Une méthode simple d'estimation du temps d'exécution est présentée dans [140]. Dans cette étude, les temps d'exécution sont calculés en effectuant le produit du nombre d'instructions exécutées par le temps moyen d'exécution. Le traitement est effectué sur un processeur MIPS. L'étude se limite à caractériser Ie temps d'exécution sur ce processeur et n'évalue en aucun temps la pertinence de transposer des segments de code sur un coprocesseur de type FPGA.

Dans [133] et [137], on propose des méthodes statistiques pour modéliser l'exécution sur l'unité centrale de traitement de sorte que plusieurs types de CPU peuvent être évalués en regard du programme devant être exécuté, Parallèlement, [130] propose un système de synthèse logiciel, où toutes les primitives pour construire un programme sont définies dans une séquence d'instructions fixe. Le temps d'exécution requis pour effectuer un bloc d'instructions donné est pré calculé. Par conséquent. Ces primitives peuvent être employées pour effectuer des prévisions précises d'exécution.

Un modèle proposé dans [132] estime le temps d'exécution d'un programme par le nombre de cycles d'exécution nécessaires pour chaque instruction dans le programme, la quantité de mémoire en lecture/écriture, et le nombre de cycles pour chaque accès mémoire, Cependant, ce modèle ne tient pas compte des caractéristiques intrinsèques de l'architecture matérielle cible, ce qui est essentiel pour optimiser I'exécution via un partitionnement efficace.

Dans [129], la méthode présentée, et est synthétisée. L' approche consiste d'évaluation employée dans le système COSYMA et le système cible exécutant le programme est également à "exécuter" le code sur un modèle du système cible au niveau RT (Register-Transfer) pour en extraire des caractéristiques de synchronisation à partir des résultats de simulation. Cette méthode d'évaluation considère Ie code source dans son ensemble et non pas bloc de base.

Une autre variante intéressante pour calculer Ie temps d'exécution du logiciel est d'utiliser un modèle d'estimation- Il peut être spécifique à un processeur ou générique, de façon à pouvoir être indépendant du processeur choisi [131] [136]. Un profilage [142] peut être fait sur les différents blocs du logiciel, de façon à déterminer le nombre de fois où chacun des blocs sera exécuté, par exemple. Ce profilage peut se faire de façon dynamique en faisant exécuter plusieurs fois Ies blocs sur le processeur cible ou en utilisant un outil tel que QPT2 [141]. Le profilage statique n'utilise pas les données pour déterminer le nombre de fois où chacun des blocs sera exécuté.

Il doit donc être capable de déterminer les bornes supérieures et inférieures. À cet effet, l'outil Cinderella [135] peut être utilisé pour ce type de profilage.

Une étude dans [138] compare PP à QPT2, PP est un outil de profilage qui emploie la bibliothèque « EEL & AR951 », pour insérer une instrumentation dans les programmes exécutables, ainsi que QPT2 est un autre outil de profilage construit avec EEL, qui utilise un algorithme [139] profilant les effets de bord (les débuts et Ies fins de boucles). QPT2 requiert habituellement moins de temps système, mais les deux systèmes donnent des résultats similaires.

Encore une fois, ces outils de profilage ne considèrent pas les caractéristiques de l'architecture matérielle cible, ce qui est un handicap important lorsque l'on désire effectuer un partitionnement matériel/logicie1 pertinent et efficace.

4.7.Les outils d'analyse du WCET

On voit cependant apparaître depuis peu quelques outils, commerciaux ou non, d'analyse statique de WCET.

Ces outils représentent les premières applications industrielles d'une quinzaine d'années de recherche sur le domaine de l'analyse statique de WCET.

De nombreux outils de calcul du WCET exploitant ces deux familles d'analyses existent, le plus connu étant certainement Heptane. Heptane [147] est un outil de calcul de WCET composée de plusieurs modules.

Les deux modules principaux sont le module d'analyse de haut niveau exploitant l'arbre syntaxique du programme pour obtenir une formule symbolique du WCET et le module qui calcule le coût des blocs de base au niveau du langage machine.

Le découpage retenu dans Heptane lui permet de supporter plusieurs processeurs [77]. Le module d'analyse bas niveau correspondant. Heptane a notamment été utilisé pour analyser le code complet [89] du système d'exploitation temps réel open source RTEMS

Nous pouvons aussi citer les travaux de « Engblom et al » visant à définir un outil de calcul de WCET adapte à l'informatique embarquée [146]. Leur but est de proposer un outil capable d'analyser du code C quelconque pour une utilisation dans un contexte industriel de développement de logiciel embarqué.

En effet, pour bénéficier d'Heptane pour un nouveau microprocesseur il suffit d'écrire le module d'analyse bas niveau correspondant. Heptane a notamment été utilisé pour analyser le code complet [89] du système d'exploitation temps réel open source RTEMS [148].

Nous pouvons aussi citer les travaux de Engblom et al visant à définir un outil de calcul de WCET adapté a l'informatique embarquée [146]. Leur but est de proposer un outil capable d'analyser du code C quelconque pour une utilisation dans un contexte industriel de développement de logiciel embarque.

Les analyseurs statiques de WCET développes pour une utilisation industrielle à notre connaissance sont:

L'analyseur de temps d'exécution Bound-T de la société (Space Systems Finland Ltd1),cet analyseur n'utilise que le graphe de flot de contrôle du programme (pas l'arbre syntaxique), ce qui lui permet d'être indépendant du langage de haut niveau (il analyse le code objet). L'obtention des informations complémentaires sur les chemins d'exécution est semi-automatique (l'obtention des bornes de boucles est automatique pour les boucles dites à compteurs). La recherche du pire chemin d'exécution est basée sur une méthode IPET. Les différents processeurs cibles de cet outil sont: l'Intel-8051 (8 bits), deux DSPs 32 bits et le SPARC V7. Un point faible de cet analyseur est l'absence de prise en compte de l'architecture matérielle.

L'environnement d'analyse statique de la société AbsInt 2 (fondée en 1998 et issue de l'Université des Saarlandes) offre des outils d'analyse statique du cache d'instruction, de simulation de pile d'exécution, et un générateur d'analyseur statique de programme.

AbsInt ne propose donc pas d'analyseur statique de WCET mais propose les outils pour en construire un.

L'analyseur statique de WCET Cinderella 3.0 de l'université de Princeton 3 est l'outil académique le plus avancé. Il utilise une méthode IPET et prend en compte les effets du cache d'instructions et du pipeline des processeurs Intel I960KB et Motorola 68000. Les informations complémentaires sur les chemins d'exécution (e.g. les bornes sur le nombre d'itérations des boucles) sont demandées interactivement à l'utilisateur.

4.8.Conclusion

Dans ce chapitre nous avons présenté un état de l'art sur l'analyse de WCET dans le logicielle une analyse de haut niveau et une analyse de bas niveau.

Après avoir présenté la problématique de l'estimation du temps d'exécution au pire cas, nous avons comparé deux approches: l'estimation par test et mesure, et l'analyse statique de WCET. Nous avons ensuite étudié les différentes méthodes d'analyse statique de WCET existantes sous plusieurs aspects: les différents types de données nécessaires à ces méthodes d'analyse statique de WCET, et les techniques mises en œuvre par ces méthodes.

De cet effet, nous avons distingué deux niveaux d'analyse, le niveau haut qui regroupe les techniques de recherche du pire chemin d'exécution, et le niveau bas qui est chargé de la modélisation du comportement de l'architecture matérielle. Enfin, nous avons présenté les différentes techniques existantes permettant de réaliser ces deux niveaux de l'analyse statique.

le chapitre suivant va présenter comment analyser le temps pour le hardware.

CHAPITRE 5

ANALYSE DU TEMPS POUR LE HARDWARE

5.1.Introduction

Un circuit intégré (*integrated circuit*, IC) réalise une fonction électronique sous la forme d'un ensemble de composants électroniques miniaturisés assemblés sur un même substrat, usuellement de silicium.

Les progrès constants des techniques de fabrication permettent aujourd'hui de placer plusieurs dizaines de millions de transistors sur une surface de silicium plus petite qu'un timbre poste. Même si le nombre total de transistors est souvent dû pour une grande part à des structures régulières (mémoires, chemins de données), la conception du circuit dans son ensemble pose de redoutables problèmes lorsqu'il s'agit de satisfaire des contraintes de performances (surface, délais, partition matériel logiciel, partition logique- analogique) et de marché (domaine d'application, disponibilité et obsolescence rapide du produit).

Plusieurs objectifs doivent ainsi être absolument atteints:

• Une exploration efficace de l'espace des solutions de manière à prendre les bonnes décisions le plus tôt possible.

• Une réutilisation optimum de l'expertise acquise de manière à éviter de repartir systématiquement de zéro à chaque nouveau produit. Ceci requis, entre autres, une gestion rigoureuse des données de conception.

• Une grande flexibilité dans les technologies de réalisation possibles de manière à pouvoir rapidement tirer partie de nouvelles performances technologiques sans devoir nécessairement remettre en cause tous ou partie des choix de conception. Un exemple typique est la réalisation d'un prototype au moyen d'un circuit programmable du type FPGA (*Field Programmable Gate Array*) et de la version optimisée au moyen d'un ASIC (*Application Specific Integrated Circuit*).

Le mode de fonctionnement du circuit joue aussi un rôle très important. Un *circuit logique* (*digital circuit*) travaille selon un mode discret qui ne considère qu'un nombre limité d'états. Le comportement du circuit consiste principalement à passer d'un état à un autre et peut être décrit sous la forme d'un programme. Un microprocesseur est un exemple type de circuit logique.

Un circuit *analogique* (*analog circuit*) travaille selon un mode continu dont le comportement peut être décrit sous la forme d'équations. Un amplificateur audio de classe B est un exemple type de circuit analogique.

Un circuit *mixte* (*mixed-signal circuit*) incorpore des parties fonctionnant en mode logique et des parties fonctionnant en mode analogique. Un convertisseur logique-analogique (*digital-to-analog converter*) est un exemple type de circuit mixte.

D'une manière très générale, les circuits logiques se prêtent bien au traitement de l'information (*digital signal processing*), les circuits analogiques se prêtent bien à la mise en forme de signaux mesurés (capteurs (*sensors*)) et les circuits mixtes sont évidemment essentiels pour réaliser les interfaces entre les deux modes de fonctionnement.

5.1.1.Conception assistée par ordinateur

La complexité des fonctions réalisées sur une seule puce de silicium ne peut être maîtrisée que grâce à l'assistance d'outils logiciels appropriés et de méthodes de conception systématiques. Il existe trois grands types de méthodes de conception: Les méthodes descendantes, les méthodes montantes et les méthodes mixtes.

Les *méthodes descendantes* (*top-down*) sont basées sur une suite de raffinements successifs partant d'un cahier des charges pour aboutir à une description détaillée de la réalisation. Le cahier des charges définit le "quoi", c'est-à-dire principalement les fonctions à réaliser et les conditions dans lesquelles ces fonctions devront s'exécuter. A l'autre bout du processus, la réalisation décrit le "comment", c'est à- dire la manière qui a été retenue pour fabriquer un circuit satisfaisant les contraintes imposées par le cahier des charges. Les méthodes descendantes sont bien adaptées à la réalisation de circuits dont la structure peut être optimisée de manière très flexible à partir d'un ensemble de cellules standard (*standard cells*) ou des matrices de portes (*gate arrays*, *sea of gates*). Les contrôleurs (séquenceurs) sont des exemples typiques de tels circuits.

Les *méthodes montantes* (*bottom-up*) se basent sur l'existence de modules (primitives ou fonctions plus complexes) caractérisés, c'est-à-dire dont les fonctions et les performances sont connues. Une réalisation possible est alors construite par assemblage à l'aide d'un processus de sélection des modules. Le processus est tel qu'il doit garantir que les choix faits satisfont les contraintes imposées par le cahier des charges. Les méthodes montantes sont bien adaptées à la réalisation de circuits dont la structure est essentielle à leur bon fonctionnement. Les circuits réguliers (mémoires, chemins de données (*datapath modules*) sont des exemples types. Un additionneur, bloc de base très courant dans les circuits intégrés, peut par exemple

être réalisé selon différentes architectures (propagation de retenue, anticipation de retenue, etc.) avec différents niveaux de performances.

Les *méthodes mixtes* (*meet-in-the-middle*) sont une combinaison de méthodes descendantes et de méthodes montantes. Elle sont particulièrement adaptées à la réalisation de circuits à applications spécifiques (ASIC) possédant un grand nombre de composants personnalisés comme des multiplieurs, des unités de contrôle et de la mémoire.

Le processus de conception passe ainsi par un certain nombre d'étapes, chacune d'elles nécessite une description de l'état du système sous forme graphique (diagrammes, schémas, etc.) ou textuelle (algorithmes, liste de pièces et connectivité (*netlist*), etc.). Ces descriptions peuvent être fournies par le concepteur ou produites par des outils logiciels. On peut distinguer deux types de descriptions: Les formats d'échange et les langages de description de matériel.

Les *formats d'échange* (*interchange format*) sont des descriptions qui ne sont destinées à être lues et comprises que par des outils logiciels. On trouve par exemple dans cette catégorie les formats CIF et GDSII pour le layout et EDIF pour le schéma et le layout.

5.2.Architecture matérielle:

Cette section présente une synthèse des différentes plates-formes les plus utilisées lors de la conception de systèmes matériel/logiciel. Principalement, on y retrouvera les circuits reprogrammables ainsi que quelques systèmes embarqués.

5.2.1. Les FPGA

Bertin, Roncin et Vuillemin @ER921 enregistrent des vitesses impressionnantes pour dix algorithmes réalisés sur des coprocesseurs FPGA. Les algorithmes utilisés ont été soigneusement conçus pour convenir, de façon optimale, à l'architecture matérielle.

Certains de leurs exemples présentant des accélérations du traitement considérables sont à l'origine d'une reformulation des algorithmes, afin d'orienter leurs mises en œuvre de façon très étroite avec les caractéristiques intrinsèques de l'architecture ciblée. Leurs exemples sont simples et ils décrivent des applications spécifiques (non génériques), par opposition aux programmes classiques qui seront étudiés dans ce mémoire.

Luk, Lok et Page [62] ont effectué une étude qui montre comment une application logicielle peut être accélérée, en utilisant des FPGA qui lui sont disponibles, lors d'un partitionnement matérieMogicie1 soigneux qui adapte la partition matérielle en fonction de

l'architecture cible. Cependant, l'avantage principal repose essentiellement sur un choix judicieux ou une conception soigneuse de l'algorithme en fonction de l'architecture logicielle et matérielle disponible. En d'autres termes, tous ces facteurs doivent être connus par l'utilisateur. Malheureusement, cette approche n'est pas faisable pour un partitionnement matériel / logicie1 automatique d'une application donnée quelconque. Notre but est d'automatiser *ce* processus, en utilisant des estimateurs de performance afin de prédire la pertinence d'un éventuel partitionnement matériel/logicie1, Koch et Golze décrivent un système embarqué de coprocesseurs [63], qui est utilisé dans la conception, et le but de son utilisation est semblable à [64].

Cependant, ils se concentrent sur la conception de systèmes embarqués plus que les méthodologies de partitionnement, De plus, ils ne mettent en application aucun algorithme de partitionnement.

5.2.2. Les ASICs

Dans [65], la méthodologie de Codesign matériel/logicie1 est vue comme une approche rentable et prometteuse pour mettre en œuvre des systèmes complexes. La rentabilité est dérivée d'un partitionnement approprié des tâches du système parmi les composantes matérielles (applications spécifiques ASKs) qui sont rapides mais chères et les composantes logicielles (exécutées dans un ou plusieurs dispositifs, habituellement des processeurs standard) qui sont plus lentes mais peu coûteuse. En outre, certains comportements sont plus efficaces suite à une mise en œuvre matérielle ou logicielle.

Dans la phase de spécification, le système est décrit en utilisant le langage formel LOTOS [66]. LOTOS a les caractéristiques requises pour définir les spécifications au niveau du système. Les buts et les contraintes, guidant le processus de Codesign, sont dans la description même du système. Des transformations formelles sont appliquées pour l'optimiser, Après cette phase, la syntaxe et la sémantique sont analysées afin d'établir une forme intermédiaire avec laquelle le partitionnement matériel/logicie1 sera effectué L'architecture matériel logiciel cible se compose d'un processeur exécutant le logiciel, d'un ASIC mettant en application le matériel et d'une mémoire partagée consultée par un bus commun- Des modules d'interfaces sont utilisés pour relier le processeur et I'ASIC au bus, tout en tenant compte des transmissions et des transferts de données entre elles, Cette architecture est paramétrable en ce qui concerne le nombre de processeurs et/ou d' ASIC.

Nous venons de voir les différents modèles permettant de représenter un système, le simuler et l'analyser. La construction de ces modèles analysables et/ou exécutables se fait alors par l'intermédiaire d'un ou plusieurs langages qui permettent de capturer les spécifications du système.

Dans le domaine du co-design, différents projets ont vu le jour et ces projets ont choisi un ou des langages de spécifications différents selon le type de système qu'ils doivent étudier et la méthodologie qu'ils veulent établir.

Nous récapitulons dans cette section les langages les plus utilisés pour la spécification des systèmes matériels.

5.3.Langage de description de matériel

Un langage de description de matériel (*HDL – Hardware Description language*) est un outil de description, éventuellement formel, permettant la description du comportement et de la structure d'un système matériel. Un langage de description de matériel idéal a les propriétés suivantes:

• Il supporte la description d'une large gamme de systèmes à la fois logiques (numériques) et analogiques.

Pour les systèmes logiques, il supporte les systèmes combinatoires et séquentiels, synchrones et asynchrones. Pour les systèmes analogiques, il supporte non seulement des systèmes électriques, mais aussi mécaniques, thermiques, acoustiques, etc. Ce dernier aspect est très important pour la conception des systèmes car il s'agit de prendre en compte les interfaces avec l'environnement extérieur dans lequel le système conçu sera testé et dans lequel il fonctionnera.

• Il permet la description de l'état de la conception pour toutes les étapes du processus. Le fait d'utiliser un langage unique renforce la cohérence entre différentes représentations d'un même système.

• Il renforce aussi la cohérence des outils logiciels utilisés pour la simulation et la synthèse. Il peut être directement compris comme langage d'entrée pour de tels outils.

• Il est indépendant de toute méthodologie de conception, de toute technologie de fabrication et de tout outil logiciel. Il permet au concepteur d'organiser le processus de conception en fonction des besoins (conception descendante - *top-down design*, conception

ascendante - *bottom-up design*, séparation des parties logiques et analogiques, séparation des parties de contrôle et des parties opératives, etc.).

• Il supporte plusieurs niveaux d'abstraction et autorise des descriptions hiérarchiques. Il supporte des descriptions comportementales (fonctionnelles) aussi bien que structurelles. Pour chaque niveau d'abstraction, il supporte les primitives correspondantes et leurs caractéristiques et permet d'exprimer les contraintes de conception. Un aspect important est la possibilité de spécifier des caractéristiques temporelles comme le cycle d'horloge, des délais, des temps de montée, de descente, de pré positionnement (*setup time*) et de maintien (*hold time*).

• Il est extensible: il permet au concepteur de définir de nouveaux types d'objets et les nouvelles opérations correspondantes.

• Il est plus qu'un simple format d'échange entre outils logiciels. Il renforce la communication et la cohésion à l'intérieur des équipes de conception et entre les différentes communautés de concepteurs parce qu'il est lisible (format texte) et qu'une description écrite dans un tel langage contient beaucoup d'information sur l'expertise derrière la conception. Il améliore donc grandement les phases de spécification et de documentation.

• Il est standardisé par l'intermédiaire d'organisations reconnues comme l'IEEE, l'ANSI ou l'ISO. Ceci favorise une large acceptation à la fois de la part des fournisseurs d'outils EDA et des différentes communautés de concepteurs.

Nous récapitulons dans cette section les langages les plus utilisés pour la spécification des systèmes matériels.

5.3.1.Le langage de description matérielle VHDL

Le langage VHDL est un langage de description matériel des systèmes numériques. C'est un langage standard IEEE.

Le VHDL est basé sur l'assemblage de composantes nommées entités. Ces entités peuvent être définies par le langage sous la forme d'instructions séquentielles et/ou parallèles. Ces instructions peuvent soit correspondre à l'assemblage d'autres entités, soit correspondre à des instructions de base du langage.

Certaines variables peuvent être évaluées dans le temps.

La communication entre les entités se fait par l'intermédiaire de mémoires partagées représentées par des signaux.

C'est un langage qui permet de représenter la plupart des concepts nécessaires à la description des systèmes temps réel. Il n'a cependant pas d'instructions spécifiques aux transitions d'états et permettant de définir des exceptions.

Il est utilisé par plusieurs projet de Codesign notamment le projet Lycos (LYngby COSynthesis) [67]. Ce projet, de l'université technique du Danemark, traduit les spécifications VHDL en un graphe de flots de données et de contrôle sous un format intermédiaire qui peut ensuite servir à différents outils et notamment un outil de simulation de ces spécifications et un outil de partitionnement.

5.3.2.HardwareC

Le langage HardwareC est un langage de description matériel issu du langage C et étendu par l'adjonction d'instructions et de constructions utiles à la description matérielle des systèmes numériques. Il n'est cependant que très peu utilisé par les outils commerciaux de synthèse numérique.

Le projet Vulcan dirigé par R.K.Gupta utilise ce langage de spécification pour le traduire en un graphe de flots de données et de contrôle qui doit ensuite être implanté sur une architecture mixe comprenant un ASIC et un microprocesseur [13].

5.3.3.SystemC

Le langage SystemC est un ensemble de classes C++ dédié à la conception des systèmes électroniques. En fait, SystemC, tout en restant compatible avec le C Ansi, introduit de nouveaux concepts lui permettant de faire de la conception au niveau système d'un circuit:

- gestion des aspects temporels (description des horloges);
- bibliothèque de classes C++ pour la description de ports, d'interfaces, etc.;
- gestion du parallélisme;
- spécifications pour la vérification.

Ce langage a pour objectif de supplanter les langages VHDL en introduisant la spécification des parties logicielles [13].

5.4.Modèle

La création d’un modèle résulte d’un processus de structuration d’un ensemble de connaissances, parfois expérimentales, que l’on dispose à propos d’un phénomène ou d’un

système physique. Un modèle peut représenter le comportement et/ou la structure d'un système donné.

Le *comportement* (*behaviour*) d'un système se concentre sur la (les) fonction(s) du système en exprimant des relations de cause à effet. Les fonctions d'un système peuvent être organisées de manière hiérarchique (fonctions imbriquées ou récursives).

La *structure* (*structure*) se concentre sur la manière dont un système est organisé hiérarchiquement en sous-systèmes. La structure d'un système peut être de plus considérée sous deux aspects complémentaires:

Un aspect adimensionnel pour lequel les notions de dimension, de taille et de forme sont ignorées et un aspect géométrique qui tient compte de ces notions. Une description structurelle adimensionnelle est usuellement un schéma, éventuellement hiérarchique, représentant uniquement les liens topologiques entre éléments du système [68].

L'ensemble des modèles considérés ici est celui des *systèmes matériels* (*hardware systems*), c'est à dire des systèmes réalisés sous la forme de circuits intégrés. Ces modèles peuvent être classés en termes de *niveaux d'abstraction* (*abstraction levels*) et de *vues* (*views*). La Tableau 5.1 présente ces deux notions orthogonales. Chaque niveau d'abstraction est caractérisé par la nature des informations sur le modèle et sur le type de composant de base, ou *primitives*, qu'il considère [68]:

Niveaux d'abstraction	Vues			Primitives
	Comportementale	Structurelle	Physique	
Système	Spécification de Performances	Connexion topologique de processeurs, mémoires, bus	Partition en chips, modules, cartes, sous-systèmes	Processeur, mémoire, bus
Architecture (matériel)	Comportement concurrent déclaratif, transfert de données entre registre),	Connexion topologique de blocs fonctionnels (ALU, mémoires, multiplexeurs, registre)	Partition et placement de blocs fonctionnels (floorplan)	Blocs fonctionnels (ALU, mémoires, multiplexeurs, registre)
Architecture (logiciel)	Comportement séquentiel (impératif, algorithmes)	Structures de données, décomposition en Procédures	Placement et routage de cellules	Types, objets, Routines
Logique	Equations Booléennes	Connexion topologique de Portes	Placement et routage d'éléments électriques de base (layout)	Portes logiques
Circuit	Equations différentielles non linéaires	Connexion topologique d'éléments électriques (transistor, capacité, résistance, sources, etc.)		Eléments électriques (transistor, capacité, résistance, sources, etc.)

Tableau 5.1: Niveaux d'abstraction et domaines de description.

Le niveau *système* (*system level*), est le niveau le plus abstrait (le moins détaillé). Le système à concevoir est vu comme un ensemble de processus communicants, mais représentant des fonctionnalités de très haut niveau comme des processeurs, des mémoires, des canaux de communication.

La manière dont les processus communiquent entre eux est plus importante que le comportement des processus eux-mêmes. Le but principal est d'évaluer des réalisations de spécifications pour différentes décompositions fonctionnelles et différentes utilisations des ressources (débit de traitement, espace mémoire, etc.). Les modèles sont dits *non interprétés* (*uninterpreted models*).

• Le niveau *architecture* (*architecture*) prend en compte le *partitionnement matériel/-logiciel* (*hardware/software co-design*). La partie matérielle est décrite à un niveau appelé *transfert de registres* (*register transfer level*, RTL) sous la forme d'une représentation synchrone du système décomposé en une partie de contrôle et une partie opérative travaillant de manière concurrente. Les composants de base sont des modules logiques complexes, tels que ALU, multiplexeur, décodeur, registre, etc. L'information est constituée par des bits et des mots et le temps est réduit à des coups d'horloge. La partie logicielle est décrite à un niveau *algorithmique* (*algorithmic level*) sous la forme d'un programme constitué d'une séquence d'opérations. L'information peut être de n'importe quel type et le temps peut être explicite (réduit à des coups d'horloge) ou implicite (les événements sont gérés par la causalité).

• Le *niveau logique* (*gate level*) est basé sur l'algèbre de Boole avec quelques extensions pour introduire les aspects temporels (délais). La correspondance entre équations booléennes et portes logiques est immédiate. L'information est quantifiée sous la forme de valeurs 0 et 1, ou sous forme multi valuée (0, 1, X, Z, …). Le temps est représenté comme un multiple entier d'une unité de base, p.ex. La femtoseconde.

• Le *niveau circuit* (*circuit level*) est le niveau le moins abstrait (le plus détaillé). Les composants de base sont les éléments électriques traditionnels (transistors, capacité, résistance, sources, etc.) dont les comportements sont représentés par des équations mathématiques impliquant des fonctions du temps ou de la fréquence. Le temps et la fréquence sont des variables réelles.

A chaque niveau, il est en plus possible de représenter trois vues du système:

• La *vue comportementale*, représente ce que le système fait sous la forme d'un comportement entrée-sortie (boîte noire). Toute décomposition hiérarchique est purement fonctionnelle et n'implique pas forcément une structure.

• La *vue structurelle* représente comment le système est logiquement construit sous la forme d'une interconnexion de composants. Cette vue ne prend pas en compte l'aspect géométrique.

• La *vue physique (ou géométrique)* représente comment le système est réellement construit. Elle prend en compte les aspects de taille, de forme et de position des composants.

Pour illustrer la Table 1.1, on donne maintenant un exemple de description comportementale, structurelle et physique. Le Code 1.1 donne un extrait du comportement

d’une instruction d’appel conditionnel de sous-programme d’un processeur hypothétique en langage VHDL.

5.5.Les étapes de la synthèse

La synthèse d'un système matériel peut suivre différentes méthodes. La méthode dite ascendante (bottom-up) se base sur un ensemble de modules caractérisés par leurs fonctionnalités et leurs performances. La conception se fait alors par la construction d'une architecture constituée de ces modules et respectant les contraintes imposées par le cahier des charges [13].

La conception peut se faire aussi selon une méthode descendante (top down), qui est basée sur le raffinement progressif des spécifications identifiées à partir du cahier des charges. Le co-design se base sur une conception descendante en utilisant les spécifications fonctionnelles décrites au niveau système comme point de départ.

Dans le cadre des processeurs matériels, la synthèse est composée de deux étapes.

La première correspond à la synthèse architecturale, la seconde est nommée synthèse logique puis un synthèse physique [68].

5.5.1.Synthèse architecturale

La synthèse architecturale est aussi nommée synthèse comportementale ou synthèse de haut niveau. Le but de cette synthèse consiste à transformer la description fonctionnelle du système en une architecture numérique appelée communément: description au niveau registre (RTL). La description RTL représente le fonctionnement du système en utilisant les structures numériques fondamentales que sont les alu, les multiplexeurs, les décodeurs, les registres, etc.

Une méthodologie de co-design cherche à inclure des outils de synthèse architecturale automatique ou semi-automatique. Ce type d'outil retranscrit une spécification fonctionnelle en l'association de deux type d'entités: le chemin de données et l'unité de contrôle. Par analogie avec les microprocesseurs, le chemin de données correspond à L'ensemble {mémoire de données + registres de données + alu} et l'unité de contrôle a le même rôle que l'ensemble {unité de commande + registres d'adresses + mémoire programme [13].

Les chemins de données sont constitués de trois types de composants:

1. les unités de stockage.
2. les unités fonctionnelles comme les alu, les comparateurs, multiplieurs, etc.

3. les unités d'interconnexions.

L'unité de contrôle est une machine à état finis qui séquence l'exécution des opérations dans le chemin de données.

La synthèse de l'unité de contrôle et du chemin de données s'effectue en deux étapes:

- L'ordonnancement et l'allocation des ressources [69].
- L'ordonnancement permet de déterminer le séquencement des calculs. Une séquence pouvant comporter plusieurs opérations en parallèle.

L'allocation des ressources sélectionne le type et la quantité des modules matériels issus d'une bibliothèque, et les met en correspondance avec chaque opération. Elle accomplit également les allocations de registres et des connexions.

La synthèse se base sur l'optimisation d'une fonction coût qui est une combinaison du coût des ressources matérielles, du cycle d'exécution, et du nombre d'étapes de contrôle.

5.5.2. Synthèse logique

La synthèse logique détermine une architecture au niveau logique en identifiant les portes logiques nécessaires à la réalisation des blocs définis au niveau RTL et en déterminant une interconnexion optimale de ces portes, Deux tâches principales sont effectuées:

• Une phase d'*optimisation logique* (*logic optimization* ou *logic minimization*) cherche à réécrire les équations logiques (pour un circuit combinatoire) ou à minimiser le nombre d'états (pour un circuit séquentiel) représentant la fonction d'un bloc décrit au niveau RTL.

• Une phase d'*allocation technologique* (*library binding*) détermine la meilleure structure à base de cellules standard dans une technologie donnée ou la meilleure programmation d'un circuit FPGA.

La synthèse logique doit satisfaire les contraintes d'optimisation suivantes:

• La *surface* doit être minimum. La part due aux interconnexions doit être estimée à ce stade. Cette composante devient prépondérante lorsque l'on entre dans le domaine des technologies submicroniques (ð 0.5µ).

• Les *délais* doivent être minimums. La partie combinatoire du circuit exhibe toujours un *chemin critique* (*critical path*) dont le temps de propagation des signaux doit être ramené à une valeur minimum.

• Le *temps de cycle* (*cycle time*) doit être minimisé. Ceci concerne les circuits séquentiels dont les registres doivent être pilotés par une ou plusieurs horloges avec une fréquence maximum, à noter que le temps de cycle minimum est borné par le délai du chemin critique de la partie combinatoire.

La synthèse logique va tout d'abord compiler le modèle pour en dériver une représentation interne sous la forme d'un graphe.

Les opérateurs logiques utilisés dans le modèle sont traduits en portes logiques équivalentes.

On parle bien d'estimation de surface ici car la place effective prise par les interconnexions n'est pas encore connue. Un modèle de calcul se basant sur le nombre de fils associés aux noeuds a été utilisé [68].

La synthèse logique est principalement efficace pour de la *logique aléatoire* (*random logic*), c'est à dire de la logique non structurée ou peu structurée basée sur des cellules standard (NOT, NAND, MUX, flip-flops, latches, etc.) ou des composants programmables (EPLD, FPGA). La synthèse de blocs réguliers comme des opérateurs arithmétiques en tranches de bits (*bit slice operators*) ou des mémoires RAM et ROM requiert en général des outils spécialisés appelés générateurs de modules (*module generators*) qui possèdent une meilleure connaissance de leur structures particulières et qui permettent ainsi des optimisations plus poussées. Ces générateurs spécialisés produisent directement une description géométrique (*layout*) optimisée qui sera assemblée aux autres blocs lors de la synthèse physique.

Pour plus de détails sur la synthèse logique, consulter [70].

5.5.3. Synthèse physique

La synthèse physique a pour but de produire une description géométrique du circuit. Au cette description géométrique consiste en plusieurs couches de polygones, chaque couche correspondant à un masque utilisé pour la fabrication. Les positions relatives des polygones définissent des composants électroniques (transistors) et des interconnexions (*wires*).

La synthèse physique effectue deux tâches principales: le *placement* (*placement*) et le *routage* (*routing*). Le placement à pour fonction d'assigner des positions aux cellules faisant partie du circuit [68].

Ces cellules peuvent être de complexités variables allant du simple transistor, en passant par une porte logique, jusqu'à une macro cellule (mémoire, ALU). Les cellules complexes sont en général placées par blocs selon un *plan directeur* (*floorplan*). Chaque bloc est ensuite réalisé indépendamment selon plusieurs styles possibles.

Il est aussi question d'optimiser le placement des cellules de telle manière à ce que la surface soit minimum.

Un mauvais placement induira un grand nombre d'interconnexions trop longues qui prendront de la place et qui induiront des délais de propagation longs. Un autre aspect très important du placement est donc de minimiser les délais. Des techniques de *placement dirigé par les délais* (*timing driven placement*) ont été développées qui complémentent les techniques d'optimisation utilisées en synthèse logique.

5.6.Analyse et synthèse

Un modèle est une représentation abstraite d'une réalité physique dont on ne conserve que les aspects essentiels à une certaine utilisation. Un modèle ignore donc délibérément des détails, soit parce qu'ils sont inutiles ou peu importants, soit parce qu'ils ne sont pas (encore) connus. Dans le premier cas, le modèle sert de support à l'*analyse*, soit l'extraction et la vérification de propriétés. Dans le second cas, le modèle sert de support à la *synthèse*, soit la dérivation d'un modèle plus détaillé en fonction de contraintes de conception [68].

Le processus d'analyse peut être réalisé, ou supporté, par *simulation*, c'est-à-dire par un processus dans lequel le modèle est décrit sous une forme exécutable par un logiciel de simulation. Le modèle est soumis à un ensemble de stimuli et le logiciel calcule la ou les réponses à ces stimuli. Le logiciel de simulation est en général optimisé pour un type de modèle exécutable et utilise des méthodes numériques, logiques ou symboliques (résolution symbolique d'équations). Un exemple est la simulation logique qui permet de calculer l'évolution temporelle des signaux dans un circuit logique en fonction des signaux appliqués aux entrées primaires du circuit.

Le processus d'analyse peut aussi être réalisé, ou supporté, par des processus basés sur des techniques de *preuve* ou de *vérification*. Il n'est pas nécessaire d'appliquer des stimuli dans ce cas. Un exemple est le calcul des délais entrées-sorties d'un circuit logique par accumulation des délais des portes logiques et des interconnexions des chemins entre les entrées primaires et les sorties primaires du circuit.

Le processus de synthèse réalise des transformations sur un modèle de manière à en dériver un nouveau modèle plus détaillé et optimisé en fonction de contraintes imposées (p. ex. surface minimum, délais minimums). Un exemple est la dérivation d'un circuit logique capable de réaliser la ou les fonctions décrites par un algorithme tout en satisfaisant des contraintes sur les types et les nombres d'opérateurs disponibles (p. ex. N additionneurs et M multiplieurs) et sur le temps de cycle.

5.6.1.L'estimation des performances

Après avoir défini les différentes étapes de la synthèse du processeur matériel, on peut définir les différents niveaux d'estimations possibles.

Le premier niveau correspond au niveau des spécifications fonctionnelles, après avoir construit le graphe de flots de contrôle et de données. Puisque aucune synthèse n'a encore été exécutée, il n'est pas possible d'estimer correctement le nombre de portes logiques qui seront utilisées. Cela ne peut être fait qu'en limitant l'exploration de la synthèse architecturale (parallélisation maximale des calculs ou minimisation des ressources).

Dans ce cadre plus restreint, une analyse du CDFG peut permettre d'estimer le nombre de ressources utilisées et donc d'estimer approximativement la taille résultante.

L'estimation après la synthèse architecturale semble, quant à elle, relativement précise, en considérant une bibliothèque de composants fondamentaux caractérisés par le nombre de portes qu'ils utilisent et leurs temps de calculs. Il manque cependant l'ensemble des temps de propagation associés au routage [13].

L'estimation après synthèse logique correspond à l'estimation la plus précise et donne alors pour le temps de calcul, le temps de propagation du chemin critique.

Actuellement, l'estimation des performances après synthèse architecturale ou synthèse logique est couramment utilisée dans les outils commerciaux et ne pose pas de problèmes particuliers. La synthèse architecturale est, par contre, loin d'être un domaine maîtrisé.

5.7. Méthodes d'estimation

Il existe principalement deux méthodes permettant d'évaluer les performances d'une application lorsque celle-ci est exécutée sur une plate-forme matérielle. La première utilise une technique de synthèse. Cette technique réalise essentiellement une synthèse rapide sans aucune optimisation. Le bloc matériel est donc transformé depuis sa description comportementale à partir d'un graphe de flots de données. Une allocation d'opérateurs est

effectuée pour chacune des opérations- Ces opérateurs sont ensuite ordonnancés. Par la suite, une estimation du nombre d'étapes de contrôle est faite, suite à I'exécution du dit bloc matériel afin d'obtenir un temps d'exécution approximatif. Pour ce faire, un outil de synthèse comme Monet de Mentor Graphics peut être utilisé.

La seconde, consiste à réaliser une traduction du langage C vers un langage de description matérielle, tels que le VHDL ou le Verilog. Différents outils sont disponibles à cet effet [72] [71], cependant ceux-ci ne supportent qu'un sous-ensemble limité du langage et ils sont souvent orientés vers des architectures de type VLIW, par exemple. De nos jours, on retrouve des outils commerciaux de conception de systèmes sur puce (System-on-Chip, SOC), tel que SystemC [73].

L'apparition et la grande popularité de ce type de méthodologie de conception de système sur puce ont apporté avec lui une variété de suggestions pour un langage simple qui peut décrire toutes les conditions fonctionnelles pour ces conceptions fortement complexes. Ces systèmes favorisent la conception des blocs matériels qui peuvent être réutilisés éventuellement. Par la suite, différentes techniques peuvent être utilisées pour estimer le temps d'exécution du bloc matériel. Par exemple, le même programme peut être exécuté plusieurs fois et le temps maximal d'exécution pourra être choisi comme étant le temps d'exécution du bloc matériel. De cette façon, on obtient une quantité considérable de statistiques permettant ainsi d'évaluer les performances de chacun des blocs matériels analysés.

5.8.Conclusion

La conception de la partie matérielle est très compliqué comme nous avons vue et nécessite des ingénieurs spécialisé dans le domaine du hardware.

Dans la littérature nous avons constaté que l'analyse du temps du matérielle en utilisant des simulateur et peut de travaux base sur l'estimation nous présentant dans les chapitre suivant notre approche dédié pour l'estimation du temps d'exécution sur le matérielle et aussi sur le logicielle.

CHAPITRE 6

PROPOSITION DE L'APPROCHE D'ESTIMATION DU WCET

Dans ce chapitre, nous présentons l'approche que nous proposons pour estimer le WCET.

6.1.Le cas d'un processeur logiciel

De nombreux problèmes existent concernant le calcul de WCET. Le premier concerne les excès de pessimisme déjà signalé lors de la présentation des différentes méthodes. Quant au second, il est issu des analyses mixtes effectuées en parallèle sur les représentations de haut niveau extraites du code source et sur le code machine compilé. Les compilateurs effectuent souvent des optimisations qui détruisent la structure du code (injection du code des fonctions dans le code de l'appelant, compression de code...). Les outils de calcul de WCET sont souvent amenés à devoir simuler le comportement du compilateur pour éviter les divergences entre les analyses symboliques et les analyses de bas niveau.

Dans ce qui suit, nous allons présenter les différents démarches de notre approche afin d'estimer le temps d'exécution dans la partie logicielle d'un système.

6.1.1.Choix du niveau d'estimation du temps

Même si l'analyse statique du code source de haut niveau permet aisément d'accéder à la structure du programme, les temps d'exécution de fragments de code ne peuvent être estimés précisément qu'à partir du code assembleur.

Dans l'approche que nous proposons pour l'analyse statique du WCET, nous avons donc décidé d'utiliser en entrée une description en langage d'assemblage. Pour ce faire, il est nécessaire d'établir une correspondance étroite entre la structure syntaxique du langage de haut niveau et les fragments de code identifiés dans le code assembleur. Ceci reste simple si les schémas de compilation mis en jeu pour passer du langage de haut niveau à l'assembleur sont connus et constants. Cependant, l'établissement d'une telle correspondance peut être rendue très difficile si le compilateur ne se contente pas de traduire le code source en assembleur [5]. En effet, les compilateurs optimisants, qui ont pour but d'améliorer les performances moyennes du code compilé, réorganisent le code assembleur en déroulant par exemple certaines boucles et appels de fonctions (fonction inline), ou encore en les dupliquant et les spécialisant.

Donc la possibilité pour l'estimation du logiciel est de compiler le langage de haut niveau en un code assembleur. Ce dernier est associé au microprocesseur spécifique, il reproduit les instructions utilisées dans les microprocesseurs utilisés pour l'analyse comme l'addition, la soustraction, la multiplication, le chargement d'une instruction vers un registre, etc. Ce compilateur doit pouvoir réaliser certaines optimisations, toujours dépendamment d'un microprocesseur donné.

Le code assembleur générique est ensuite analysé par notre outil. Dans ce dernier, chaque instruction générique est étiquetée d'une information de temps de calcul, ces informations étant liées à un microprocesseur particulier.

Nous pouvons constater qu'un outil d'estimation peut être utilisé à chaque étape de la synthèse. Cependant, plus l'estimation est faite tardivement dans la synthèse du logiciel, plus les outils sont complexes et difficiles à mettre en œuvre. En contre partie, la précision des estimations augmente avec la prise en compte de plus en plus importante des technologies utilisées.

Le choix de l'estimateur utilisé devra donc être adapté à la précision recherchée et à l'exploitation des mesures. Il est notamment préférable d'augmenter la précision au fur et à mesure que l'on avance dans le cycle de conception afin de valider au mieux les choix antécédents.

6.1.2.Avantage de la compilation

Le rôle du compilateur est essentiel pour l'optimisation du code engendré. Un effort particulier est fait sur les points suivants [148]:

- allocation optimale des registres.

- élimination des redondances.

- optimisation des boucles, en ne conservant à l'intérieur que ce qui est modifié.

- optimisation du pipeline.

- optimisation du choix des instructions.

Le compilateur doit être capable d'exploiter au maximum les caractéristiques de l'architecture.

6.1.3.Le jeu d'instructions

Les ordinateurs sont capables d'effectuer un certain nombre d'opérations élémentaires. Une instruction au niveau machine doit fournir à l'unité centrale toutes les informations nécessaires pour déclencher une telle opération élémentaire: type d'action, où trouver l'opérande ou les opérandes, où ranger le résultat, etc. C'est pourquoi une instruction comporte en général plusieurs champs ou groupes de bits. Le premier champ contient le code opération. Les autres champs peuvent comporter des données ou l'identification des opérandes

Sur certaines machines les instructions sont toutes de même longueur. Sur d'autres, cette longueur peut varier avec le code opération ou le mode d'adressage.

On distingue six groupes d'instructions:

- transferts de données: de mémoire à registre, de registre à registre, de registre à mémoire.

- opérations arithmétiques: addition, soustraction, multiplication et division.

- opérations logiques: ET, OU inclusif, NON, OU exclusif, etc.

- contrôle de séquence: branchements conditionnels ou non, appel de procédure, etc.

- entrées/sorties.

- manipulations diverses: décalage, conversion de format, permutation circulaire des bits, échange d'octets, incrémentation, etc.

MOV	ADD
PUSH	INC
POP	SUB
PUSHF	DEC
POPF	ADC
REPE	SBB
REPNE	MUL
SCASB	CLD
SCASW	DTD
LODSB	MOVSB
LODSW	MOVSW
STOSB	REP

Figure 6. 1 Exemple d'un jeu d'instruction du INTEL 8086

6.1.4.Bibliothèque du temps d'exécution

La bibliothèque de temps sert à contenir tous les temps d'exécution de chaque instruction en assembleur.

L’outil est présenté dans ce chapitre. Nous allons montrer le format de cette bibliothèque ainsi que la manière de la remplir puisque le temps d'exécution de chaque instruction est déjà pris en compte.

Le temps d'exécution est estimé en supposant l'utilisation d’un cache, donc chaque instruction est présentée sur le cache pour être exécutée. L'exemple suivant présente une portion de la bibliologique d'un assembleur INTEL 8086.

Instruction	Mode d'adressage	Temps d'exécution (ms)
ADD	Immédiat	30
ADD	Indirect	35
STA	-	72
JNZ	-	25

Tableau 6.1: exemple de contenu de la bibliothèque du WCET des instructions assembleur

6.1.5.Mécanisme de l’estimation

Notre approche d'estimation du temps d'exécution suit les étapes suivant :

Elle est réalisée à l'aide de compilateurs qui traduisent la spécification du système (modèle abstrait ou langage de haut niveau) vers un langage de niveau plus bas (assembleurs par exemple). Ce langage de bas niveau est composé d'un jeu d'instructions supporté par le microprocesseur utilisé, c'est à dire celui qui est choisi pour exécuter la partie software du système.

6.1.5.1.Insertion d'un nouveau processeur

Dans le cas où on veut insérer un nouveau processeur logiciel on suive les étapes présentées dans la figure suivante:

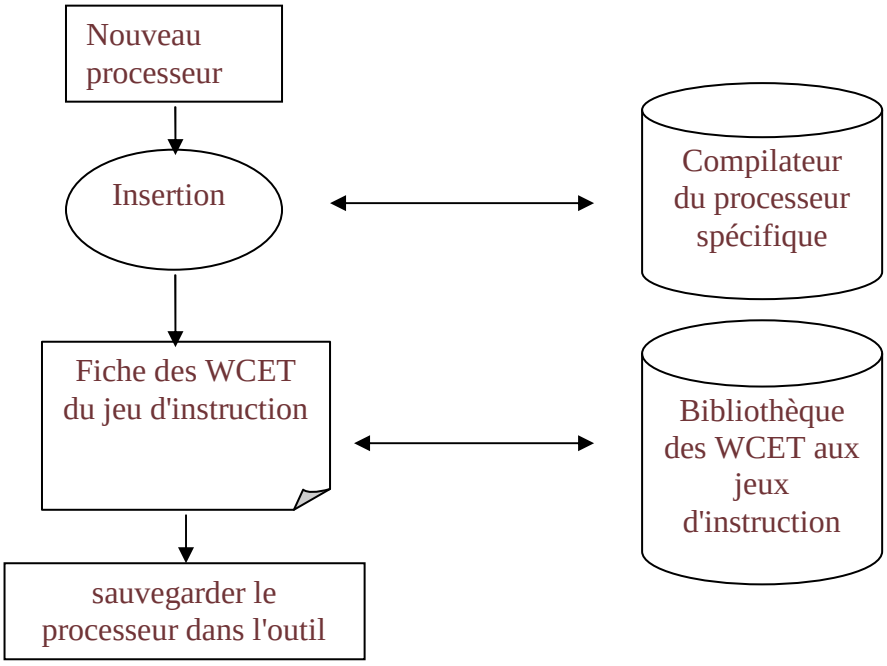

Figure 6.2. Insertion d'un nouveau processeur

Le nouveau processeur est d'abord introduit, puis toutes les informations possibles ainsi que les principales caractéristiques.

Dans l'étape suivante on introduit le jeu d'instructions du processeur: code opération de toutes les instructions ainsi que tous les types d'instructions possible et le mode d'adressage qui sont implémentés dans ce nouveau processeur.

Enfin, le temps d'exécution estimé pour chaque instruction est introduit. Rappelons que toutes les instructions sont exécutées avec prise en compte du cache d'instruction.

6.1.5.2.La méthode d'estimation

A fin de terminer le travail et d'obtenir les temps d'exécution, notre approche présentée exécute les étapes suivantes:

1- La première étape a pour but d'obtenir la spécification du système à réaliser. Cette spécification est en langage de haut niveau (C++). Elle est conçue en plusieurs entités, chaque entité peut être exécutée dans un processeur logiciel quelconque.

2- Au niveau ce cette étapes, on utilise le compilateur pour obtenir le code assembleur de chaque entité dans notre spécification du système selon le processeur logiciel utilisé.

3- Le but de cette étape est d'établir une correspondance entre le code assembleur d'une entité de la spécification et les instructions présentées dans la bibliothèque du processeur logiciel utilisé. Le résultat de cette étape permet d'aboutir au temps d'exécution estimé de chaque entité utilisée, en prenant en considération l'architecture du système et les propositions conçues dans l'analyse.

4- Cette étape permet de réaliser le même travail que les étapes précédentes mais en changeant les entités qui existent avec les différents processeurs logiciels qui existent.

5- A la fin de l'analyse, les résultats sont regroupés sous forme de tableau, fichier matrice, etc. La figure suivante présente notre méthode d'estimation.

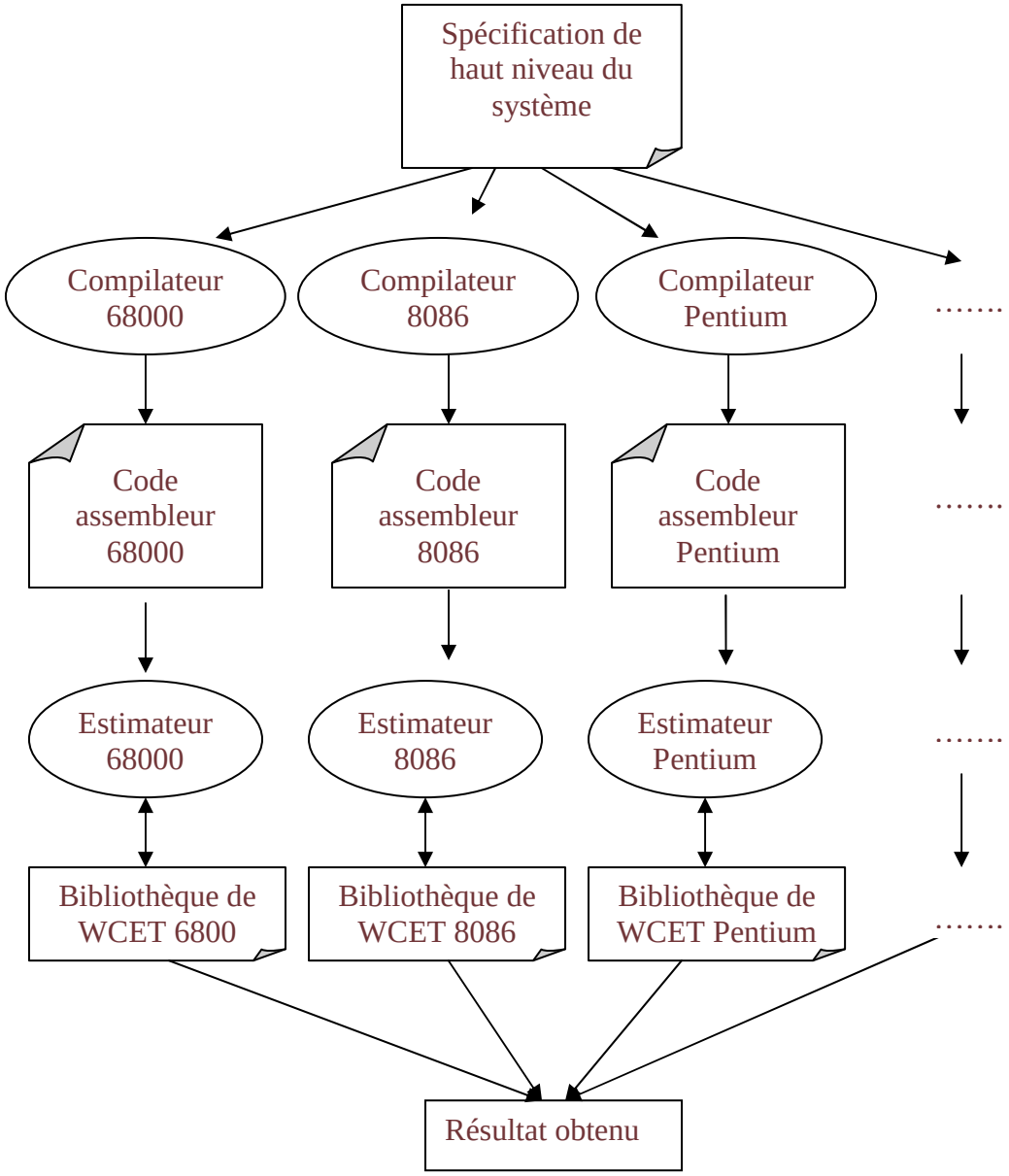

Figure 6. 2 l'estimation de la même spécification pour plusieurs types de processeurs

6.1.6.Prise en compte du mode d’adressage

Un champ adresse peut permettre de référencer un registre ou un mot en mémoire. Il peut contenir le numéro du registre ou l'adresse effective du mot mais ce ne sont pas les seules manières d'identifier un opérande. Selon le type de processeur il existe de plus ou moins nombreux modes d'adressage (le 68020 de Motorola dispose par exemple d'une cinquantaine de modes d'adressage). Le mode est défini soit par le code opération lorsque celui-ci impose un type déterminé, soit par un code faisant partie du champ adresse.

Code Opération	Mode d'adressage	adresse opérande

Il faut d'abord connaitre le mode d'adressage pour déduire le temps d'exécution depuis la bibliothèque. Il est possible de le faire à partir d'un analyseur de l'opérande de chaque instruction qui sera présenté dans ce qui suit.

6.1.7.Instruction CISC - RISC

Le concept RISC (Reduced Instruction Set Computer) est apparu en 1975 chez IBM (IBM801 de John Coke), avant d'être approfondi dans les années 80 par les universités de Stanford et Berkeley. Il repose sur la constatation que même les systèmes ou les applications les plus sophistiqués n'utilisent qu'une petite fraction du jeu d'instructions à leur disposition.

Des études statistiques, portant sur un grand nombre de systèmes d'exploitation et d'applications réels, ont montré que:

- Dans 80 % d'un programme on n'utilise les instructions RISC alors que 20 % du jeu d'instructions pour les CISC.

Les processeurs classiques sont appelés CISC (Complex Instruction Set Computer) par opposition au terme RISC. Le développement des compilateurs, la prise en charge par les concepteurs des systèmes, et le développement des architectures est très interdépendant.

La simplicité des processeurs RISC fournit au moins deux autres avantages: le coût du développement et la surface de silicium sont notablement réduits. Plus simple, un processeur RISC nécessite moins de temps et moins de main d'œuvre pour sa conception et sa mise au point.

Les instructions CISC ou RISC pour notre approche ne posent pas de problème, puisque toute instruction que ce soit CISC ou RISC est présente dans notre bibliothèque avec son temps estimé d'exécution.

6.1.8.Prise en compte du pipeline

Le traitement d'une instruction peut être découpée en plusieurs phases. A chacune d'elles il est possible d'associer une unité fonctionnelle. Considérons un microprocesseur disposant de trois unités :

- unité de recherche (R);

- unité de décodage (D);

- unité d'exécution (E).

Dans un tel microprocesseur de ce type avec pipeline, il faut toujours 3 cycles pour exécuter chaque instruction, mais dès qu'une instruction est terminée il suffit d'attendre un cycle pour que la suivante soit terminée, et non trois comme dans le cas d'un processeur sans pipeline. Dans ces conditions il ne faut que N+2 cycles d'horloge pour exécuter N instructions. On calcule presque 3 fois plus vite.

Il n'est cependant pas simple de bénéficier de ce gain de performance en totalité. Les performances d'un pipeline peuvent être dégradées par des aléas. On distingue trois types d'aléas:

- aléas structurels: deux instructions ont besoin simultanément d'une même ressource ou d'un même bloc fonctionnel;

- aléas de données: une instruction a besoin du résultat d'une instruction qui n'est pas terminée;

- aléas de contrôle: par exemple lors d'une rupture de séquence (branchement conditionnel ou non, fin de boucle, appel de sous-programme, etc.)

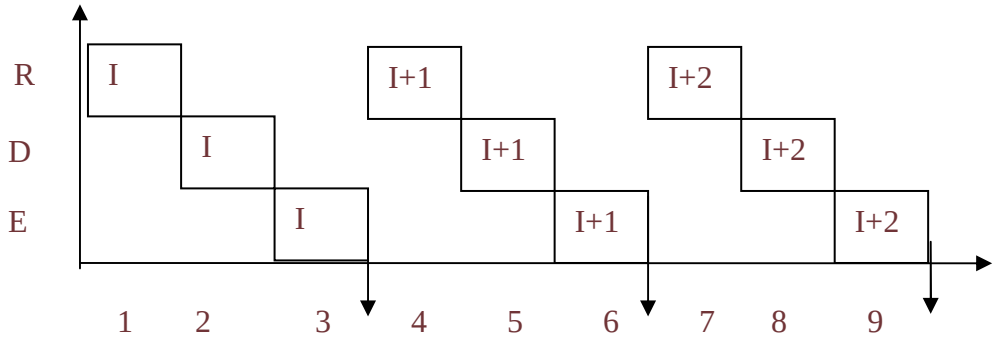

Figure 6. 3 exécution sans prise en compte du pipeline

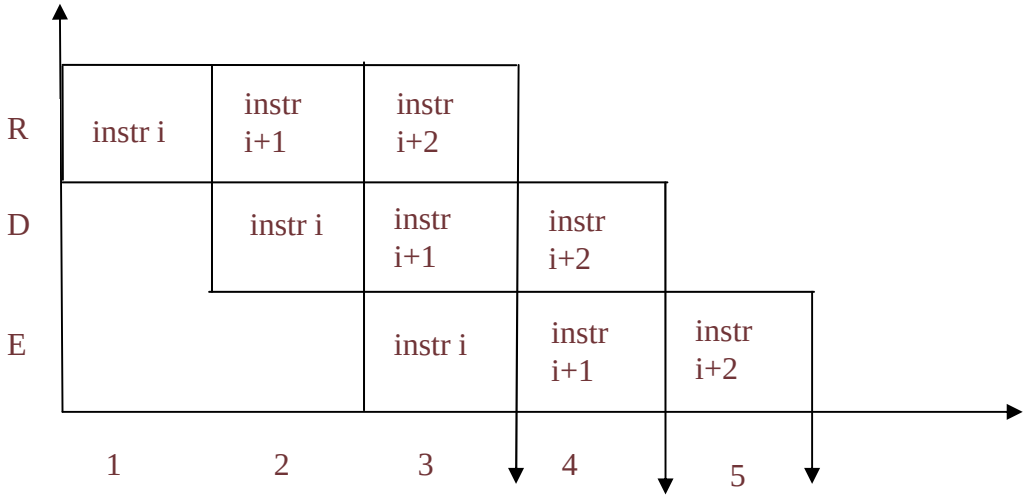

Figure 6. 4 exécution avec prise en compte du pipeline

La gestion de ces problèmes est généralement résolue par les compilateurs qui peuvent insérer des instructions NOP ou permuter des instructions du programme en tenant compte des spécificités du processeur. Cependant, il n'est pas facile d'effectuer une estimation tenant compte de ces cas. Une simulation est nécessaire.

A cet effet notre approche se base sur l'élimination de l'exécution pipeline pour ne pas tomber dans ces différents problèmes d'exécution. Donc on se base sur une exécution séquentielle qui peut surestimer le temps d'exécution mais permet une sûreté maximale.

Une autre solution qu'on peut l'implémenter dans notre approche, se base sur une probabilité d'estimation. L'utilisateur va donner à l'outil un pourcentage qui représente les instructions avec prise en compte du pipeline et le reste sans prise en compte. Exemple:

- 40% avec pipeline
- 60% sans pipeline exécution séquentielle

Le pourcentage de 60% englobe les différents aléas qu'on peut trouver dans l'exécution pipeline.

Donc le temps estimé de l'exécution c'est 60% du temps d'exécution globale sans prise en compte le pipeline (exécution séquentiel de tous le programme).

6.1.9.Exécution des boucles

Il peut être difficile de connaître le nombre de fois qu'une boucle est parcourue dans un programme donné. Ceci est principalement dû au fait qu'une boucle peut être conditionnée par des variables (Exemple : tant que A<B). Dans notre approche, la décision est laissée à l'utilisateur de fixer un nombre maximum de fois qu'une boucle donnée est parcourue. La valeur est stockée dans un registre CX. Cette valeur peur également être estimée:

- soit en fixant le nombre maximal de fois qu'elle peut être exécutée: 2^{16} CX avec 16 bits, 2^{32} CX avec 32 bits.
- Soit en fixant le nombre minimal = 1.
- soit par estimation d'une moyenne entre un maximum et minimum

Exemple d'une boucle:

```
MOV BX, 0
MOV CX, 100
REP: INC BX
ADD AX, BX
LOOP REP
```

Figure 6. 5: exemple d'une boucle

6.1.10.Prise en compte du branchement

Dans un programme de bas niveau, en plus des branchements insérées par le programmeur, il est courant que d'autres soient insérées automatiquement, par exemple pour matérialiser une conditionnelle ou une alternative. Notre approche permet de citer les différentes instructions de branchement conditionnel ou inconditionnel. La décision de prise

en compte du branchement ou non est faite par l'utilisateur. A chaque branchement trouvé, on interroge l'utilisateur pour donner une décision d'effectuer ce branchement ou pas, les types de branchement sont les suivants:

6.1.10.1. Branchement vers le haut du code

Cela se traduit comme pour l'exécution d'une boucle. L'utilisateur est informé qu'il y a un branchement conditionnel qu'on va prendre en considération ou non. Dans le cas où la réponse est oui, l'utilisateur doit donner une valeur estimative pour le nombre de fois que cette portion de code se déroule. L'outil prend cette valeur en considération et multiplie le temps de chaque instruction par la valeur estimée.

Exemple:

```
REP ADD CX,1
SUB BX, CX,
ADD AX,CX,
JMP REP /JNZ REP
```

Figure 6. 6 exemple d'un branchement en haut du code assembleur

6.1.10.2. Branchement vers bas du code

De la même façon l'outil détecte qu'il y a un branchement conditionnel vers le bas et donne la décision à l'utilisateur. Dans le cas où le branchement doit être pris, la portion de code se trouvant entre l'instruction et l'adresse de branchement est éliminée.

Exemple:

```
ADD AX, CX
SUB BX, CX
JMP REP /JNZ REP
SUB CX, AX
ADD AX
REP MOV CX, 20
```

Figure 6. 7:Exemple d'un Branchement en bas du code assembleur

6.1.11.Récapitulatif

Nous avons présenté dans ce chapitre notre approche proposée pour estimer le temps d'exécution. L'outil qui implémente cette approche sera présenté dans le chapitre mise en œuvre.

L'approche a des avantages et des inconvénients. L'intérêt principal est qu'il est possible de faire l'analyse pour un nouveau microprocesseur de manière rapide et simple. Il suffit en effet de créer le fichier de technologie contenant les informations sur le temps de calcul de chaque instruction.

L'inconvénient principal est lié au fait que le compilateur est totalement dissocié du processeur cible. Or il existe une étroite relation entre le jeu d'instruction du microprocesseur et le compilateur dédié qui permet de faire des optimisations particulières. De plus le code assembleur générique ne peut contenir que les instructions communes aux différents microprocesseurs, ce qui exclut les instructions qui font la particularité d'un microprocesseur. Cette technique sera par exemple peu efficace dans le cas de microprocesseurs CISC.

Par exemple, la compilation d'un module logiciel sur plusieurs processeurs nécessite une infrastructure coûteuse, composée de plusieurs compilateurs et de processeurs cibles, qui ne sont pas toujours disponibles. De cette façon, uniquement pour la détermination de performance des modules logiciels, même les systèmes les plus simples ont un grand coût de conception lié à l'achat des compilateurs et des processeurs cibles. La solution à ce problème est d'essayer de modéliser les étapes de la synthèse avec une approche automatique d'estimation du logiciel et du matériel au niveau système

6.2.Le cas d’un processeur matériel

La synthèse du hardware implique la réalisation des circuits du système à plusieurs niveaux. Par exemple, supposons que nous avons à synthétiser une opération de multiplication en hardware. Au niveau logique, le circuit de multiplication est réalisé à l'aide d'additionneurs. A un niveau plus bas, des portes logiques sont utilisées pour synthétiser les additionneurs. Le processus de synthèse se poursuit jusqu'à arriver au plus bas niveau hardware (bascules, portes, etc.).

6.2.1.La conception du matériel

La conception du circuit est représentée en figure 6.8. Elle se situe à trois niveaux: comportemental, structurel, et physique.

La conception démarre au niveau comportemental. On y trouve la conception du système et des algorithmes qui décrivent son comportement.

On passe ensuite à la description structurelle des unités arithmétiques et logiques au niveau transfert de registres. Les unités sont alors traduites au niveau portes logiques jusqu’à arriver à une description sous forme de transistors, en utilisant la spécification en langage de description de matériel (VHDL).

Le niveau physique termine la conception du système en permettant d’obtenir le circuit final.

On peut atteindre une bonne évaluation des performances du temps au niveau structurel. Notre approche se base sur cette description du circuit à concevoir au niveau des UAL et des portes logiques.

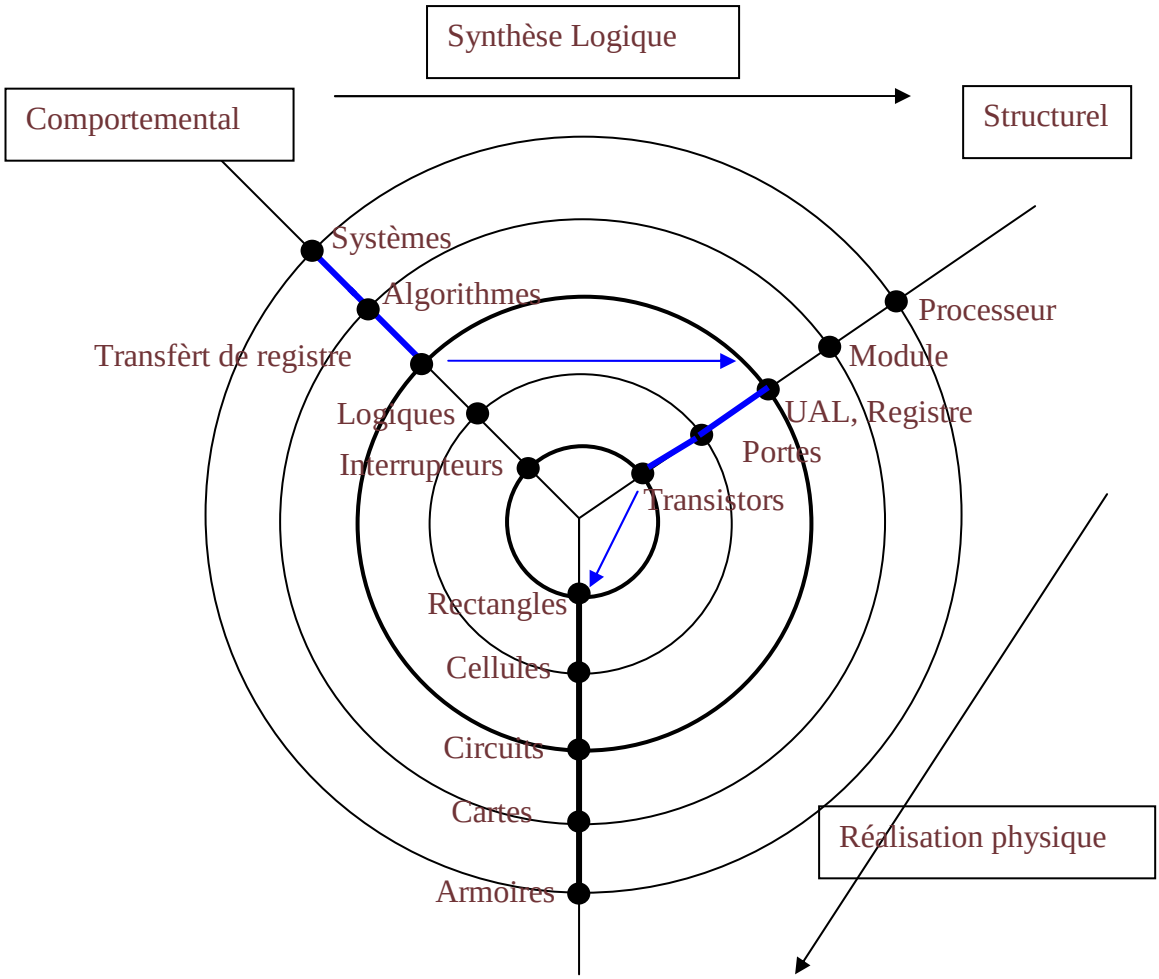

Figure 6. 8:La conception du circuit [13].

6.2.2.Le choix du langage matériel pour la synthèse

.Les fonctions numériques à réaliser ne cessent de prendre de l'ampleur. Il est dès lors indispensable d'utiliser un langage de haut niveau pour maîtriser la complexité grandissante des systèmes numériques. VHDL est l'un des langages modernes les plus puissants.

Le deuxième point fort de VHDL est d'être un langage de description comportementale de haut niveau. Les anciens langages, tel qu'ABEL, ne disposaient pas des mêmes fonctionnalités.

En fait, un langage est dit de haut niveau lorsqu'il fait le plus possible abstraction de l'objet pour lequel il est écrit. Dans le cas de VHDL, il ne fait jamais référence au composant ou à la structure pour lesquels on l'utilise. Ainsi, il apparaît une notion très importante: la portabilité des descriptions VHDL.

Il est également important de noter que le langage VHDL est normalisé. Il n'est pas la propriété d'un vendeur d'outils. Cette norme a poussé de grandes sociétés de logiciels à

l'utiliser comme langage de description de matériel pour leurs outils. Le langage est ainsi devenu un standard reconnu par une majorité des vendeurs d'outils de synthèse.

Le langage VHDL a été initialement utilisé pour la spécification et la simulation de systèmes complexes. Ce langage de haut niveau permet une décomposition hiérarchique et dispose de la notion de temps. Cela a permis de maîtriser la complexité des circuits, particulièrement dans le cas des ASIC. Le même langage est utilisé pour toutes les étapes du développement (spécification, simulation et synthèse). Il permet également la réalisation de bibliothèques de composants réutilisables d'un projet à l'autre.

On obtient notre spécification en langage VHDL en utilisant des traducteur qui traduisent le langage de haut niveau en entrée (langage C dans notre cas) vers le langage VHDL. Parmi ces traducteurs on trouve [13].

Flot de données (dfl1 et dfl2) et algorithmique (proc):

```
Entity and2 is
Generic (Tprop: time:= 0 ns);
port (A, B: in bit; Q: out bit);
End entity and2;
Architecture dfl1 of and2 is
Begin
Q <= A and B after Tprop;
END dfl1;
Architecture dfl2 of and2 is
Begin
Q <= ’1’ after Tprop when A = ’1’ and B = ’1’
Else ’0’ after Tprop;
END dfl2;
Architecture proc of and2 is
Begin
P_AND2: process (A, B)
Begin
If A = ’1’ and B = ’1’ then
Q <= ’1’ after Tprop;
Else
```

```
Q <= ’0’ after Tprop;
END if;
End process P_AND2;
END proc;
```

Figure 6. 9: exemple d'un flot de donnée en VHDL

6.2.3.Modélisation de circuits numériques

Le langage VHDL englobe un nombre important de modèles de composants matériels sauvegardés dans des bibliothèques.

Notre proposition utilise des modèles de composants matériels. Ces modèles sont définis et synthétisés en langage VHDL. On utilise ces modèles uniquement pour réaliser le système en matériel.

L'exemple suivant donne le modèle d’une porte AND à deux entrées sous la forme d’une description flot de données et sous la forme d’un processus équivalent.

```
entity and2 is
generic (Tprop: time:= 0 ns);
port (A, B: in bit; Q: out bit);
end entity and2;
architecture dfl1 of and2 is
begin
Q <= A and B after Tprop;
end dfl1;
architecture dfl2 of and2 is
begin
Q <= ’1’ after Tprop when A = ’1’ and B = ’1’
else ’0’ after Tprop;
end dfl2;
```

Figure 6. 10: Modèle d’un additionneur en VHDL synthétisé

6.2.4.Bibliothèque des WCET des modèles VHDL

Cette bibliothèque sert à sauvegarder les différents modèles utilisés en VHDL avec une estimation du temps d’exécution WCET. Ce dernier est le nombre de cycles pour exécuter ce modèle.

La bibliothèque des WCET de chaque modèle est organisée comme suit:

Type de modèles	Modèles	WCET (Nombre de cycles)
Porte logique	Inverseur	20
	NOR	5
	NAND	4
	XOR	2
	MUX	3
	AND	5
Circuit combinatoire	Comparateur	13
	Multiplexeur	15
	Encodeur	23
	Décodeur	22

Tableau 6.2:Exemple de Bibliothèque de modèles synthetisés en VHDL

6.2.5.Le mécanisme d'estimation

L'approche proposée suit le cheminement suivant pour aboutir au WCET estimé de la spécification:

La première étape permet d'aboutir à une bibliothèque globale qui contient les différents modèles utilisés en VHDL pour les constructions des circuits ainsi que leur temps d'exécution estimé ou simulé (nombre de cycle) dans les différents processeurs matériels cibles.

Cette étape est effectuée en utilisant des outils qui permettent d'aboutir au temps d'exécution des différents modèles présentés ci-dessus. Parmi ces outils, figure VIITAL. Une seconde manière d'obtenir ces temps d'exécution consiste à utiliser l'estimation.

L'un des modules de notre outil aide l'utilisateur à remplir la bibliothèque que les nombres de cycle d'exécution. Le schéma suivant montre l'utilité de cet outil:

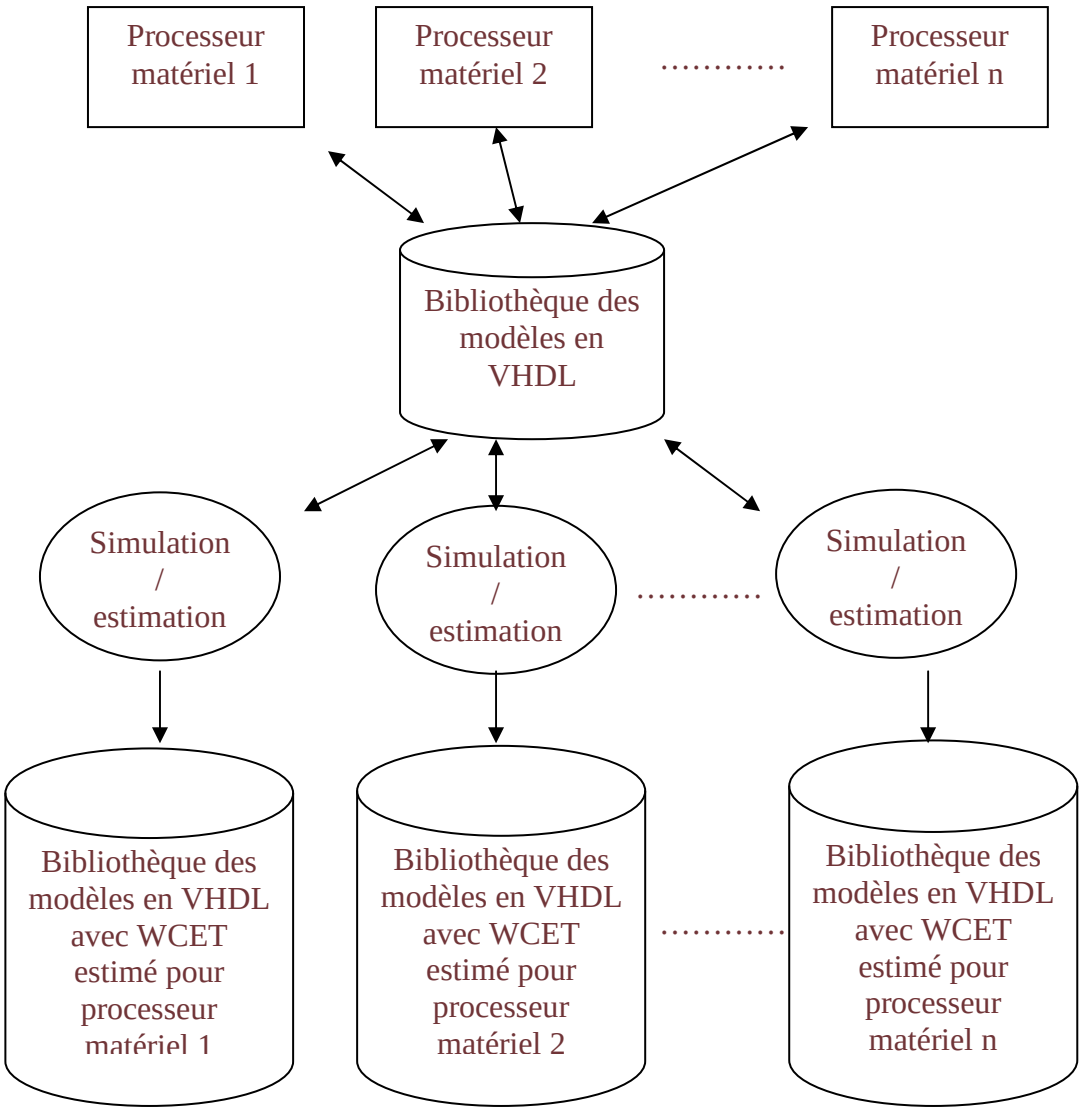

Figure 6. 11 Obtention de la bibliothèque des modèles VHDL

La deuxième étape de notre approche permet d'aboutir au calcul du temps d'exécution global.

L'outil que nous présentons permet de calculer le temps d'exécution à l'aide de la bibliothèque du WCET de chaque modèle.

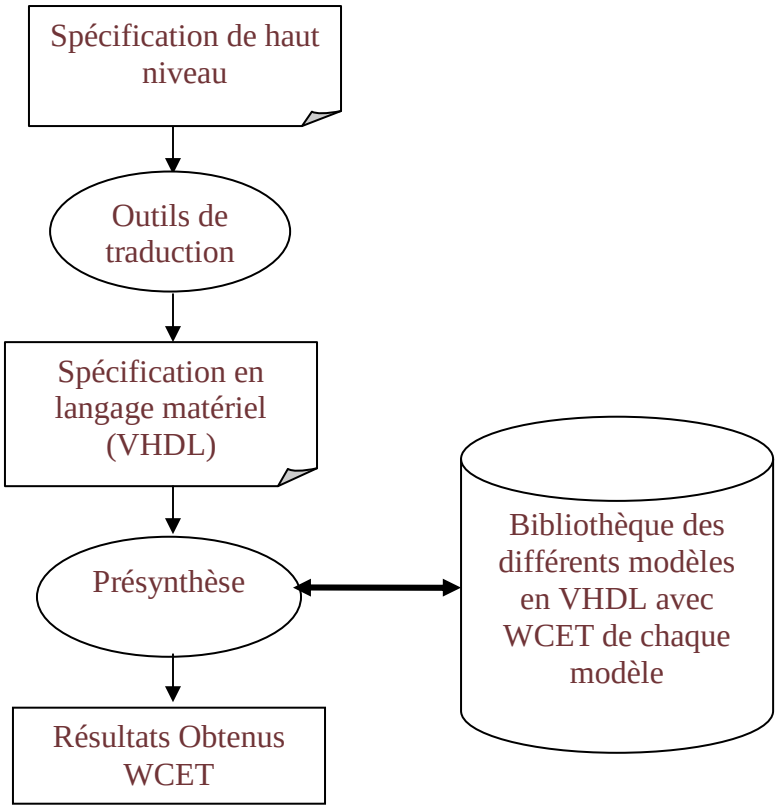

Figure 6. 12 Estimation du WCET de la spécification du matériel

Dans un premier temps, la spécification de haut niveau est traduite en VHDL en utilisant les différents outils qui existe pour traduire le code de haut niveau en VHDL.

La deuxième étape est la présynthèse. Elle permet d'établir une correspondance entre le langage VHDL de la spécification et les différents modèles.

Le but est de traduire la spécification VHDL en utilisant les différents modèles qui existent dans la bibliothèque.

Il reste alors à calculer les WCET de chaque modèle utilisé pour aboutir au WCET de la spécification sur le processeur matériel cible.

Présentons dans ce qui suit un exemple d'application:

Exemple:

```
Process EXAMPLE=
Entity TEST_CPATH_clk is
Port (clk: in BIT;
in1, in2: in INTEGER range 0 to 8;
b1, b2: in BOOLEAN;
Result: out INTEGER range 0 to 16);
END;
Architecture BEHAVIOR of TEST_CPATH_clk is
```

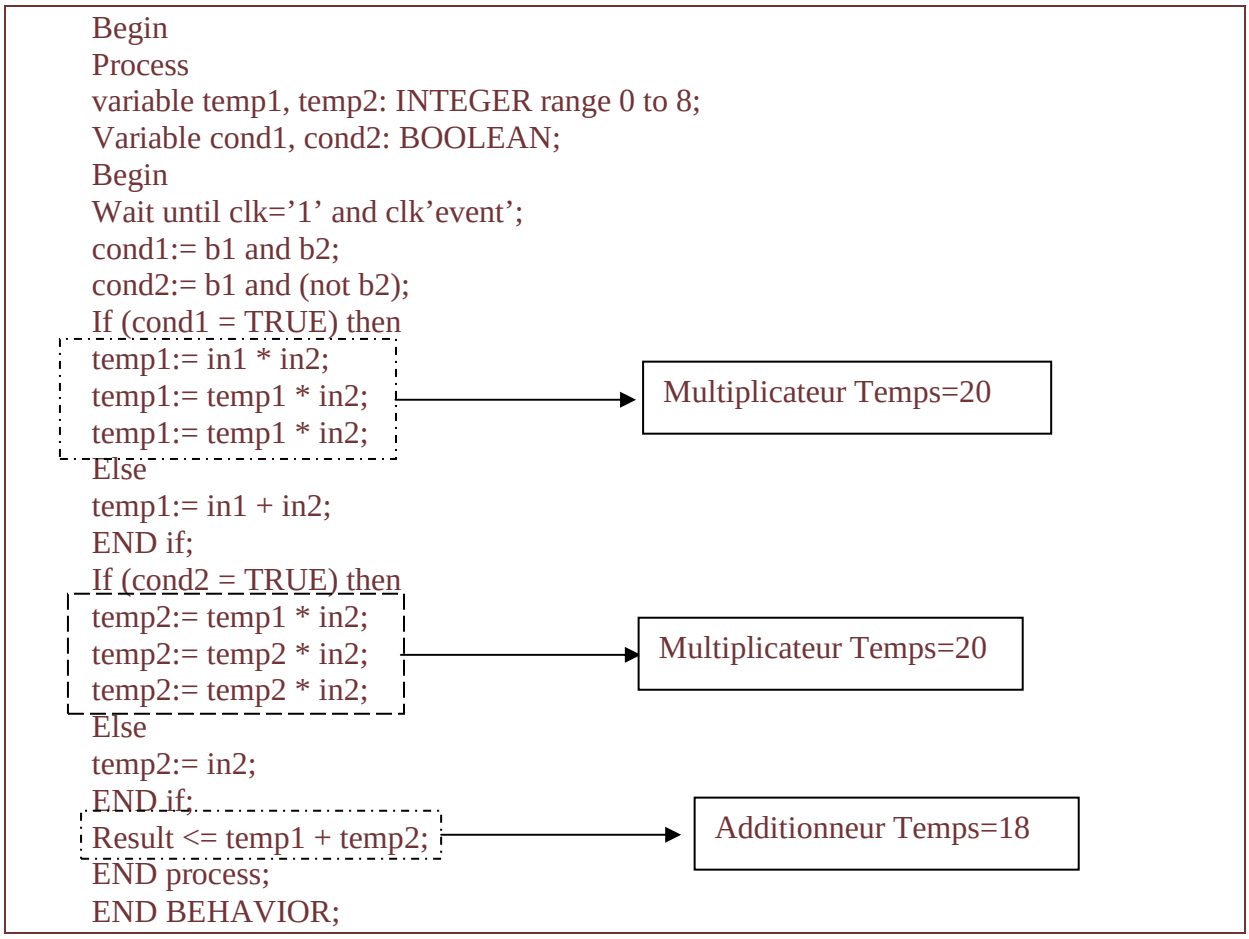

Figure 6. 13 Exemple de présynthèse du code VHDL

Cet exemple montre comment faire la correspondance entre la portion du code VHDL et les modèles de la bibliothèque. Cette correspondance se fait de manière manuelle. L'outil permet à l'utilisateur d'effectuer cette opération. Cet exemple présente le code VHDL d'une spécification qui est traduite en utilisant les modèles. Le résultat obtenu est une structure de modèle à exécuter pour cette spécification.

Chaque circuit conçu a une fréquence d'exécution propre. Cette fréquence est utilisée pour calculer les différents WCET qui sont étiquetés à chaque modèle utilisé dans notre bibliothèque. Cela permet d'obtenir le temps en multipliant le nombre de cycles trouvé par la fréquence du circuit.

6.3.Mise en œuvre

Toute approche proposée doit être suivie d'une mise en œuvre. Nous proposons dans ce qui suit les différentes étapes d'implémentation des différents outils proposés. Nous décomposons cette section en deux parties:

- la première concerne l'étape de construction de la plate forme en ajoutant les différents composants matériels et logiciels pour qu'on puisse effectuer le test.
- la deuxième consiste le test et la réalisation de l'approche sur un exemple.

6.3.1.Etape de construction

Le but de cette étape est d'acquérir une connaissance sur les différents composants qui existent. La première est consacrée à l'ajout des composants du système, qu'ils soient logiciels (processeur logiciel) ou matériels (processeur matériel).

6.3.2.Le cas de l'ajout d'un nouveau composant

Dans le cas où on veut ajouter d'autres composants pour le matériel ou pour le logiciel, notre outil permet à l'utilisateur d'ajouter un nouveau circuit (ASIC /FPGA) avec ses caractéristiques. C'est le même cas si on ajoute un composant logiciel (nouveau processeur) avec toutes ses caractéristiques.

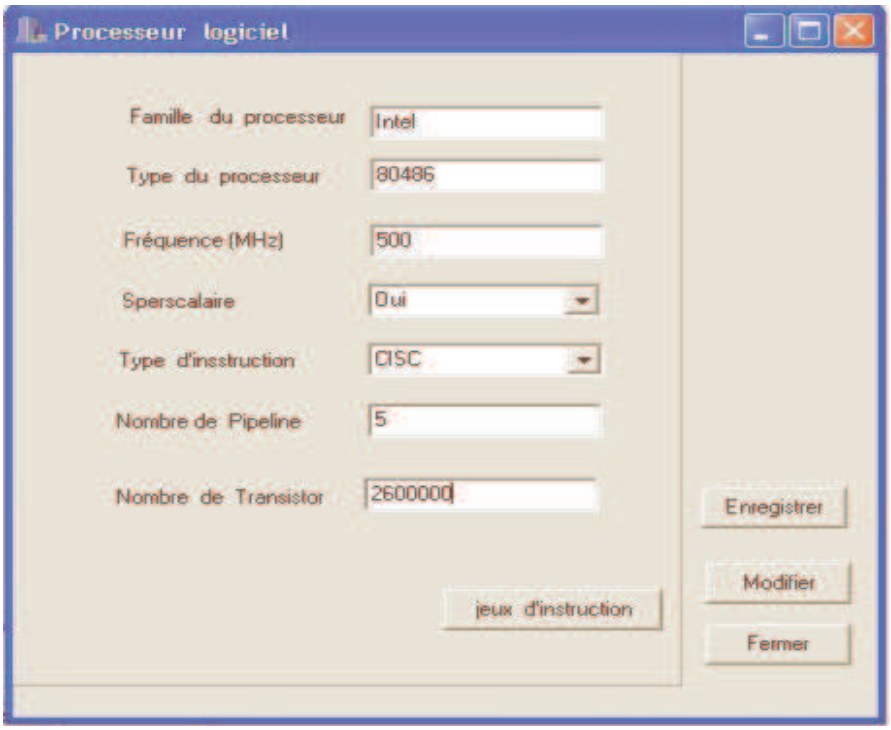

Figure 6.14 : Insertion d'un nouveau processeur logiciel

L'étape suivante consiste à fixer, pour chaque processeur logiciel ajouté, son jeu d'instructions. La Figure 6.15 illustre comment éditer ces instruction dans l'outil conçu, avec pour but de construire une bibliothèque globale des instructions et leur temps d'exécution estimé sur le même processeur logiciel.

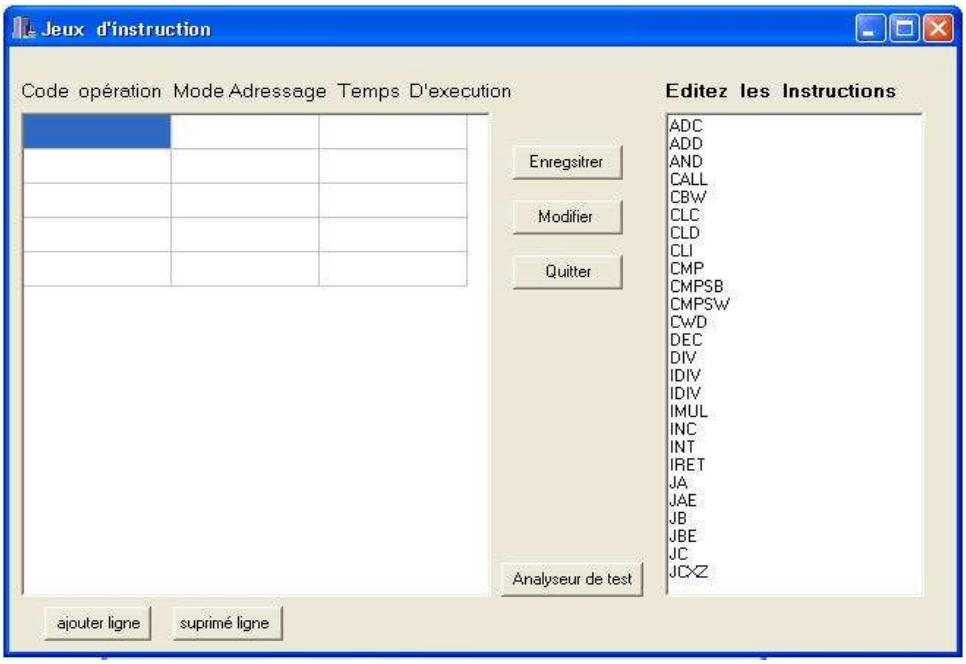

Figure 6. 16: Edition du jeu d'instruction

Dans le cas où le processeur regroupe plusieurs modes d'adressage, la Figure 6.17 montre comment présenter les différentes instructions répétitives et les instructions de branchements.

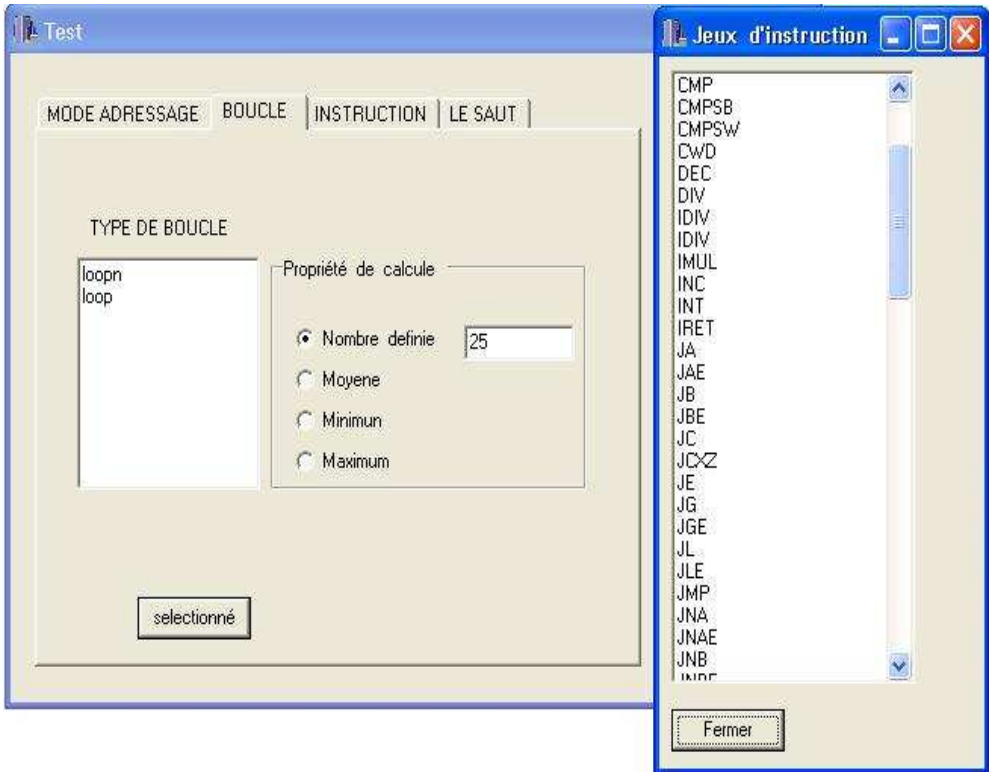

Figure 6. 18: Sélection des instructions

De la même façon, la figure 6.17 illustre la manière d'ajouter les différents modes d'adressage et leur format.

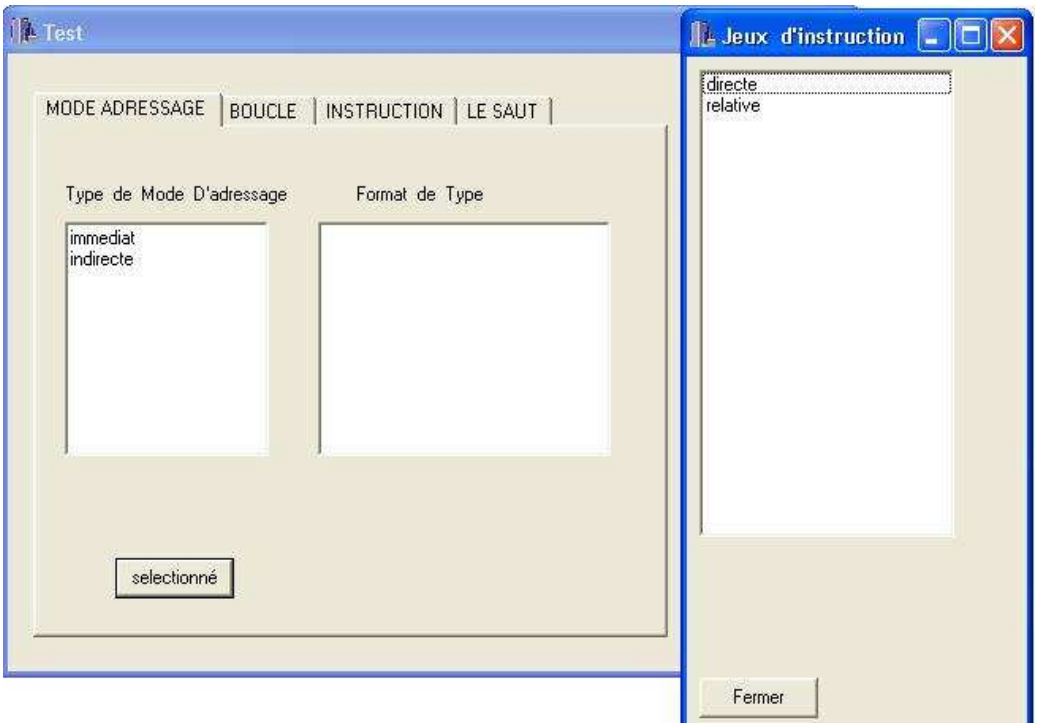

Figure 6.19 : Edition des modes d'adressage

6.3.3.Le temps estimé des instructions

Avant de commencer l'étape de test de la spécification en utilisant les processeurs logiciels insérés dans la bibliothèque, on donne le temps d'exécution estimé pour chaque instruction dans la bibliothèque pour tous les processeurs logiciels comme le montre la figure suivante.

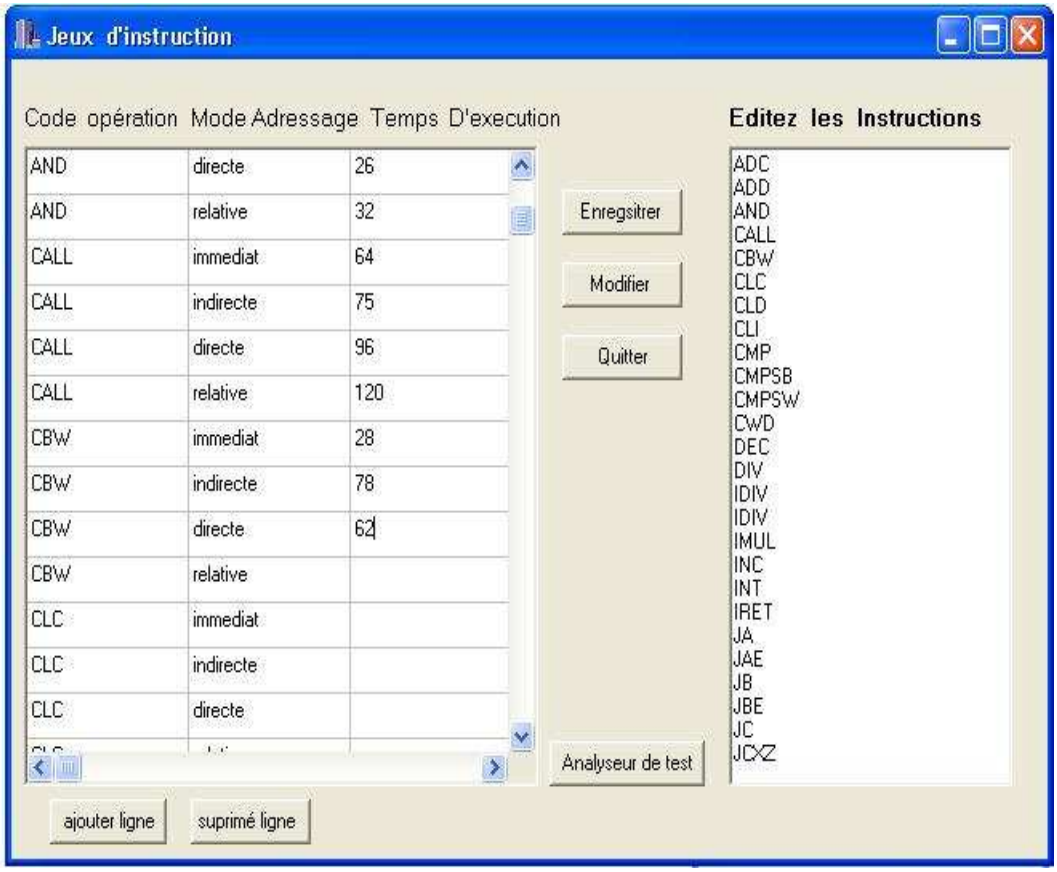

Figure 6.20 Insertion des WCET des instructions

A la fin de cette étape, les processeurs logiciels sont prêts à être utilisés pour estimer le WCET des parties logicielles du système à concevoir.

6.3.4.Le cas d'un composant Matériel

L'ajout d'un composant matériel est caractérisé par la fréquence d'exécution. Cette dernière peut être implémentée dans le code VHDL.

Exemple:

ASIC1 F=200 MHz

ASIC2 F=1500 MHz

Figure 6. 21: Exemple de description des fréquences d'exécution

6.3.5.Modèles exécutés par le processeur matériel

La bibliothèque du matériel est caractérisée par l'ajout des modèles utilisés dans le test. Ces modèles sont conçus et synthétisés en VHDL.

Parmi les modèles citons:

- Additionneur
- Soustracteur
- Les registres
- Les bascules
- Multiplicateur
- Comparateurs …….etc.

Figure 6. 22:Exemples de modèles

La figure 6.21 montre comment remplir la bibliothèque du matériel en décrivant les modèles par leur code VHDL, ainsi que le nombre de cycles que prend ce modèle pour s'exécuter. Ce nombre est estimé par l'utilisateur. Il est utilisé uniquement lors du test.

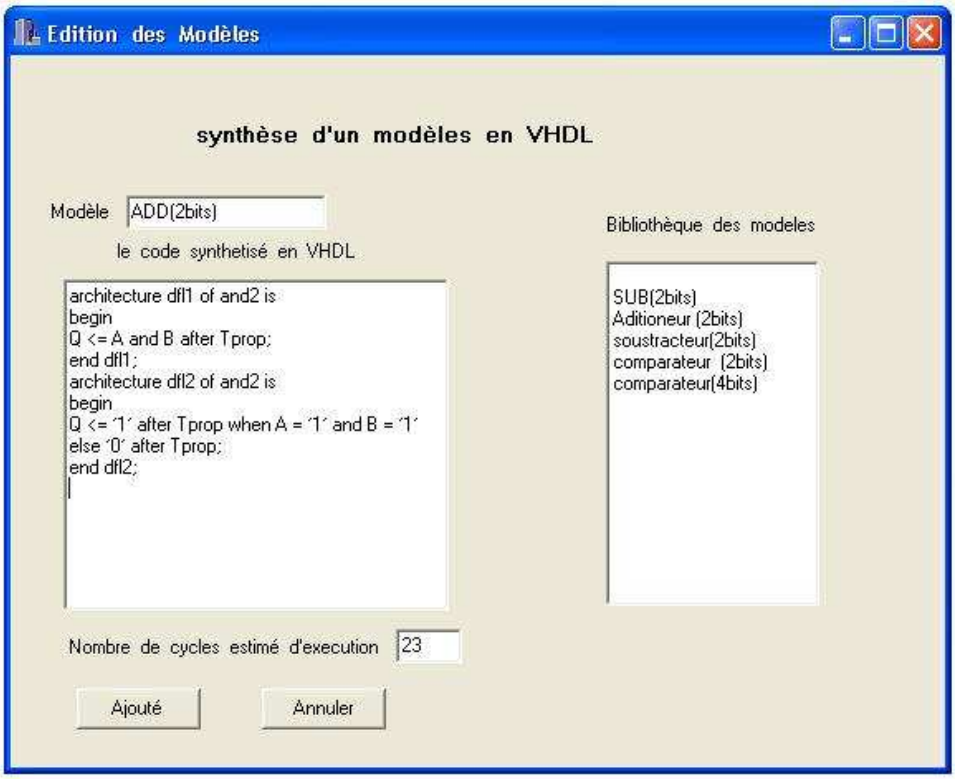

Figure 6. 23 La synthèse des modèles

6.3.6.Le choix de l'architecture cible:

A cette étape, l'utilisateur choisit les différents composants pour construire son architecture. Pour aider le concepteur à concevoir son système, un outil a été conçu. Il sert à aider l'utilisateur pour choisir l'architecture cible et les composants matériels pour concevoir son système.

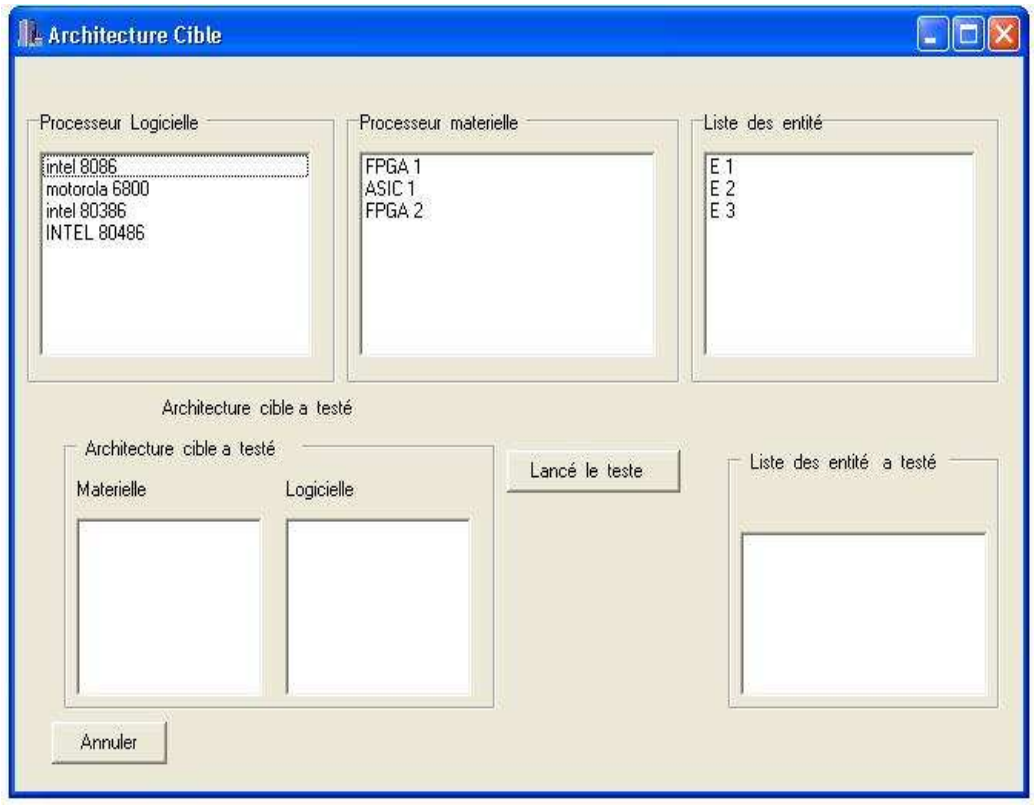

Figure 6. 24 Le test pour la partie logicielle

Un outil a été développé pour aider le concepteur à choisir les différents composants de l'architecture afin d'effectuer l'étape d'estimation du temps d'exécution en utilisant les composants choisis.

Nous présentons dans ce qui suit l'étape de test qui permet d'évaluer le temps d'exécution pour un composant matériel et un composant logiciel.

Voici un exemple de spécification:

```
int complex(int a, int b)
{
        while(a < 30)
        {
                while(b < a)
                {
                        if(b > 5)
                                b = b * 3;
                         else
                                b = b + 2;
                        if (b >= 10 && b <= 12)
                                a = a + 10;
                         else
                                a = a + 1;
                }
                 a = a + 2;
                 b = b - 10;
         }
         return 1;
}

int main()
{
        /* a = [1..30] b = [1..30] */
        int a = 1, b = 1, answer = 0;
        */if (answer)
                {a = 1; b = 1;}
        else
                {a = 30; b = 30;} */
        answer = complex(a, b);
        return answer;
}
```

Figure 6. 25 Exemple d'une spécification en langage haut niveau C++

Dans ce cas, le compilateur est utilisé pour aboutir au langage machine correspondant à la machine sur laquelle on veut effectuer le test.

Si on veut réaliser un test, par exemple, sur un INTEL 80386, il faut faire la compilation en utilisant ce processeur comme cible. Le code objet est ensuite traduit en langage VHDL à l'aide des outils de traduction qui existent.

Après cette étape, il est possible d'effectuer le test, en commençant par le processeur matériel. L'interface suivante présente cette étape:

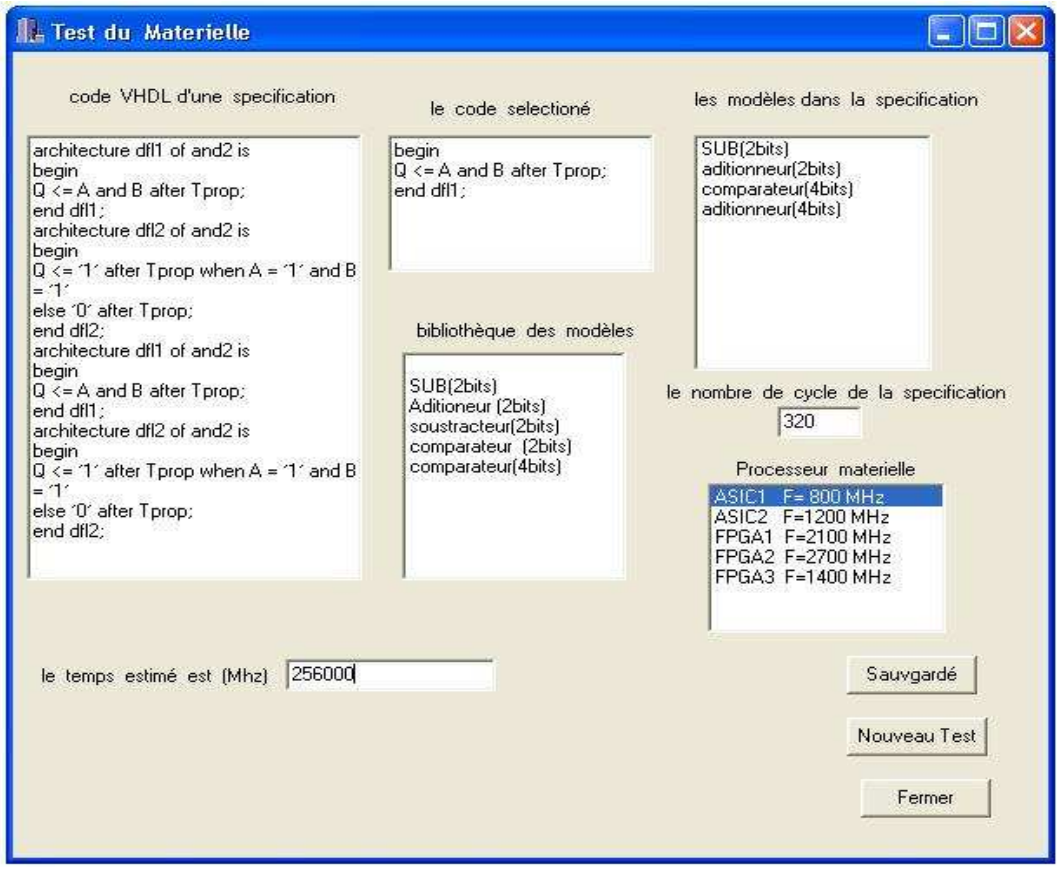

Figure 6. 26: La présynthèse du matérielle

Le code VHDL traduit est dans la fenêtre de gauche. Cette étape est appelée "présynthèse". Il est indispensable que la synthèse du code VHDL soit effectuée par un utilisateur spécialisé. Le but de cette étape est de traduire les différentes portions du code VHDL en différents modèles qui existe en VHDL.

Bien sûr, chaque modèle a son propre code VHDL détaillé et synthétisé. De plus, pour chaque modèle, le nombre de cycle est connu pour permettre son exécution.

À la fin de cette étape, nous aboutissons à un ensemble de modèles permettant de réaliser le circuit, et le nombre de cycles qu'il faut pour exécuter le code VHDL de la spécification est connu.

6.3.7.Résultat obtenu

Il reste à simuler ces résultats. La fréquence d'exécution de chaque circuit est connue.

Pour terminer le test, les résultats obtenus sont regroupés dans un tableau : les processeurs logiciels de l'architecture cible ainsi que les spécifications en assembleur.

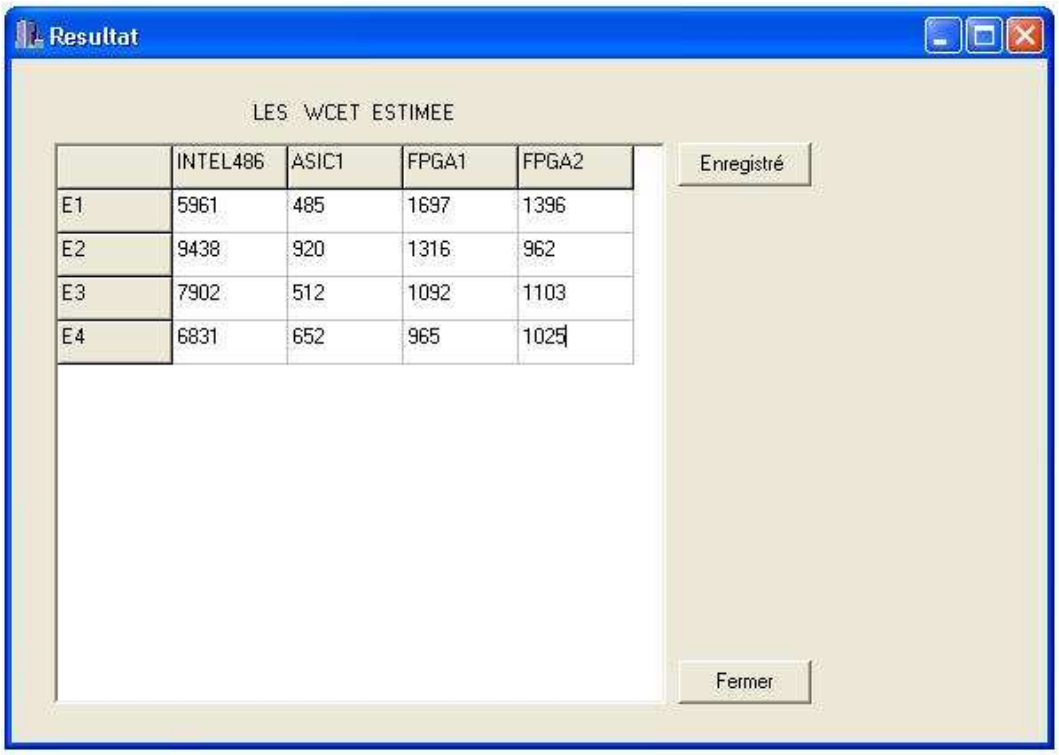

Figure 6. 27 Les résultats obtenus

6.4.Conclusion

L'approche proposée pour le logiciel, n'a pas pour but de résoudre définitivement les problèmes d'analyse ou d'évaluer les paramètre temporels, chaque méthode ou proposition se basé sur un paramètre bien précis ,langage orienté objet (C,C++,Java….ect) , niveau d'implémentation (haut niveau orienté objet) ou (bas niveau (langage machine),notre approché est basé sur le bas niveau (langage machine).

D'autre part, dans la proposition la partie hardware est intégrée dans l'outil conçu qui sert à aider l'utilisateur à effectuer une évaluation du temps, à l'aide du code VHDL qu'on obtient par la traduction de la spécification de haut niveau (C++). Des outils de traduction existent dans la littérature.

Par contre, dans l'approche du processeur logiciel, notre méthode est basée sur les langages de bas niveau (assembleur) qui permettent d'obtenir une meilleure évaluation du temps. Des compilateurs sont utilisés pour obtenir le langage de bas niveau de la spécification.

Nous avons proposé une méthode pour effectuer l'évaluation du temps. Cependant, dans notre approche, nous n'avons pas fait une proposition pour évaluer le temps de communication entre les processeurs logiciels et matériels. Nous avons basé notre approche sur le coté hardware seulement et le logiciel. Les prochaine travaux tiendront compte de la communication entre le logiciel et le matériel.

CONCLUSION

Avant de faire un découpage matériel-logiciel dans le processus de conception des systèmes mixtes, nous devons vérifier une étape d'estimation des performances.

Dans notre travail présenté dans ce mémoire, nous avons abordé les différentes étapes du processus CODESIGN et en particulier l'estimation des performances.

Les différents types de performances généralement évalués sont: le temps, l'espace, la communication et la consommation.

Notre mémoire met l'accent sur la performance temporelle. Nous avons détaillé un état de l'art en montrant les différentes méthodes utilisées pour estimer le temps d'exécution d'une spécification de haut niveau. Il montre comment estimer le temps d'exécution sur les composants logiciels et matériels.

Pour l'approche proposée sur le software:

Une spécification orientée objet d'un système permet d'obtenir une bonne estimation du temps et aussi d'éviter les différents problèmes concernant l'analyse d'un code objet.

La solution que nous avons proposée est que cette estimation soit réalisée sur une description en langage d'assemblage du système à concevoir. Pour cela, la spécification orientée objet est traduite en assembleur. Cette traduction est effectuée par le compilateur qui prend en considération les différents composants matériels des processeurs de l'architecture cible. A la fin de cette étape, nous obtenons un code assembleur équivalent à la spécification d'origine.

Un test complet de toute la spécification est alors effectué. Toutes les données extraites sont alors regroupées dans un tableau.

Pour l'approche proposée sur le hardware:

Nous avons proposé une méthode qui reste une proposition. Le temps à calculer pour le hardware est basé sur valeurs à calculer ou estimer: consommation d'énergie, surface du processeur, fréquence …etc.

Dans notre approche, notre proposition se base seulement sur le temps d'exécution en fonction de la fréquence du processeur hardware. Du travail reste à faire pour réaliser l'approche hardware et pour obtenir des résultats proches de la réalité.

Synthèse :

Notre travail présente comment utiliser le temps pour obtenir un bon partitionnement. Nous avons fait une proposition en tenant compte du temps d'exécution qui est estimé à travers le nombre de cycles utilisés. C'est une combinaison des cycles d'exécution en fonction de la fréquence du processeur matériel de l'architecture cible.

Un certain nombre de perspectives peuvent être suggérées à ce travail :

Il est possible d'ajouter une analyse sur le temps de la communication entre les différents composants da l'architecture suivant l'ordre d'exécution.

Un outil pourrait être réalisé, permettant la synthèse automatique des composants matériels.

Il serait intéressant de développer une plate forme en tenant compte de tous les cœfficients qui rentrent dans le calcul du temps (surface, temps, consommation d'énergie …).

Enfin, un outil d'aide à la décision permettrait de choisir plus facilement les composants matériels à mettre en œuvre dans une architecture cible.

RÉFÉRENCES

1 M. Auguin, "co-conception de systèmes spécialisés sur composant", les algorithmes / bâtiment euclide 2000.

2 Auguin M., Capella L., Cuesta F., Gresset E., "CODEF: a system level exploration tool", IEEE International Conference on coustics, Speech, and Signal Processing, Salt Lake City, p. 1031-1034, May 2001.

3 Ivan Augé, Rajesh K. Bawa, Pierre Guerrier, Alain Greiner, Ludovic Jacomme, and Frédéric Pétrot, "User guided high level synthesis", In Ricardo Reis and Luc Claensen, editors, VLSI: Integrated Systems on Silicon, pages 464–475, Gramado, Brazil, IFIP, Chapman & Hall, August 1997.

4 M. Azizi, "covérification des systèmes intégrés", thèse de doctorat, faculté des études supérieures, université de montréal, décembre 2000.

5 Antoine.Colin Estimation de temps d'exécution au pire cas par analyse statique et application aux systemes d'exploitation temps reel these de doctorat 2002

6 F. Balarin et al., "hardware-software codesign of embedded systems", the polis approach, kluwer academic publishers, 1997.

7 T. Ben ismail, m. Abid, k. O'brien, a. Jerraya, "an approach for hardware-software codesign", rsp'94, grenoble, france, june 1994.

8 T. Ben ismail, "synthèse au niveau système et conception de systèmes mixtes logiciels/matériels", thèse doctorat institut national polytechnique de grenoble 1996

9 FRANKE D.W. AND PURVIS M.K., "Hardware/Software codesign: a perspective",Proc 13th International Conference on Software Engineering, IEEE CS Press,Los Alamitos, California, Order N°2140-02, pp.344-352, 1991.

10 Stoy, E. A Petri Net Based Unified Representation for Hw/Sw codesign. Phd thesis, Departement of Computer and Information Science of Linköping University 1995.

11 Koudil, M Une approche orientée objet pour le codesign, Thèse d'état, Institut National d'Informatique d'Alger 2002.

12 Koudil, M. Cours intitulé -Méthode de conception conjointe des systèmes embarqués-. Institut National de formation en Informatique (Alger) 2004.

13 Réegis RUELLAND Apport de la co-simulation dans la conception de l'architecture des dispositifs de commande numerique pour les systemes electriques thèse de doctorat 2002.

14 Gajski d.d., narayan s., ramachandran l. And vahid f., "system design methodologies: aiming at the 100h design cycle", ieee trans. On vlsi systems, vol.4, n°1, pp.70-82, march 1996.

15 J. P. Calvez, "spécification et conception des système", editions masson, 1990

16 Koudil m., benatchba k., touati s.a., kherfi m.l. et tirane h., "cecooc:environnement de codesign à partir de co-spécifications c++ orientées objet", 4ème colloque africain sur la recherche en informatique, 1998,

17 A. Changuel, m. Abid, c. Valderrama, a. Jerraya, "environnement de conception et de prototypage de systèmes mixtes logiciels/matériels", journal technique des sciences informatiques (tsi), 1996.

18 Carlos alberto valderrama "prototype virtuel pour la generation des architectures mixtes" Logicielles/materielles 1998

19 M. Flynn, v. Milutinovic editor, "high-level language computer architecture", computer science press, 1989.

20 I. Bolsens et al., "hardware/sotware co-design of digital telecomunication systems,"proceedings of the ieee, vol. 85, pp. 391-418, march 1997.

21 C. Iseli, e. Sanchez, "spyder: a sure (superscalar and reconfigurable) processor", the journal of supercomputing, 9(3), pp. 231-252, kluwer academic publishers, 1995.

22 G. Boriello, k. Buchenrieder, r. Camposano, e. Lee, r. Waxman, w. Wolf, "hardware/software codesign", ieee design & test of computers, round table, pp. 83-91, 1993.

23 K. Keutzer, "hardware-software codesign and esda", proc. 31st design automation conference (dac), ieee cs press, pp. 435-436, 1994.

24 M. Voss, t. Ben ismail, a. Jerraya, k. Kapp, “towards a theory for hardware/software codesign”, proc. Third int’l workshop on hardware/software codesign (codes/cashe), ieee press, pp. 173-180, Grenoble, france, 1994.

25 M. A. Richards, “the rapid prototyping of application specific signal processors (rassp) program: overview and status”, proc. Int’l workshop rapid system prototyping, ieee cs press, order no. 5885, pp 1-6, los alamitos, calif., 1994.

26 W. Glunz, t. Kruse, t. Rossel, d. Monjau, ïntegrating sdl and vhdl for system-level hardware design”, proc. Ifip conf. Hardware description languages (chdl), plub. Elsevier science, ottawa, canada, april, 1993.

27 H. De man, i. Bolsens, b. Lin, k.van rompaey, s. Vercauteren, d. Verkest, “co-design for dsp systems”, nato asi hardware/software codesign, tremezzo, june 1995.

28 G. Cambon, c. Vial, l. Maillet-contoz, l. Torres, “environnement d’aide à la conception concurrente de systèmes dédiés matériel/logiciel”, workshop codesign, cnet, grenoble, 1995.

29 V. Madisetti, t. Egolf, s. Famorzadeh, l-r. Dung, “virtual prototyping of embedded dsp systems”, proc. Of ieee icassp , 1995.

30 A. Kalavade, e. A. Lee, “a hardware-software codesign methodology for dsp applications”, ieee design and test of computers, vol.10, no.3, pp.16-28, 1993.

31 A. Kalavade, e. A. Lee, “the extended partitioning problem: hardware/software mapping, scheduling, and implementation-bin selection”, proceedings of sixth international workshop on rapid systems prototyping, north carolina, pp 12-18, 1995.

32 R. Ernst, j. Henkel, th.benner, w. Ye, u. Holtmann, d. Herrmann, m. Trawny, “ the cosyma environment for hardware/software cosynthesis”, journal of microprocessors and microsystems, butterworth-heinemann, 1995.

33 A. Österling, t. Benner, r. Ernst, d. Herrmann, t. Scholz,w.ye, “the cosyma system”, hardware /software co-design: principles and practice, kluwer academic publishers, j. Staunstrup and w. Wolf edditors, pp. 263-305, 1997.

34 G. Goossens, i. Bolsens, b. Lin, f. Catthoor, “design of heterogeneous ics for mobile and personal communication systems”, proc. Int’l conf. On computer aided design (iccad), ieee cs press, pp. 524- 531, san jose, california,. 1994.

35 K. Van rompaey, d. Verkest, i. Bolsens, h. De man, “coware- a design environment for heterogeneous hardware/software systems”, in proceedings of the european design automation conference euro- dac’96, geneva, switzerland, p252,. 1996.

36 J-p. Calvez, "a system specification model and method," ciem current issues in electronic modeling, vol 4., high-level system modeling: specification and design

methodologies", kluwer academic publishers, editors j. Bergé, o. Levia, j. Rouillard, 1996.

37 J-p. Calvez, d. Heller, o. Pasquier, "uninterpreted cosimulation for performance evaluation of hw/sw systems,"proc. 4th international workshop on hardware/software codesign, pp. 132-139, codes/cashe'96, pittsburg, usa, 1996.

38 Zurawski, R., editor The Industrial Information Technology Handbook. CRC Press. 2005

39 Allara, A., Bombana, M., Fornaciari, W., and Salice, FA case study in design space exploration: The tosca environment applied to a telecommunication link controller. IEEE Design & Test of Computers, pages:60–72. . (2000).

40 Buchenrieder, K., Sedlmeier, A., and Veith, CHw/sw co-design with prams using codes. In Agnew, D., Claesen, L. J. M., and Camposano, R., editors, 11th IFIP Conference of the International Conference on Computer Hardware Description Languages and their Applications - CHDL , volume A-32, pages 65–78. North-Holland. . (1993).

41 Ernst, R., Henkel, J., and Benner, THardware-software cosynthesis for Microcontrollers. In IEEE Design and Test of Computers, volume 10, pages 64–75 .1993

42 Calvez, J., Pasquier, C., and Heller, D. Hardware/software system design based on the mcse methodology. In J.M. Bergé, O. Levia, J. R., editor, Current Issues in Electronic Modeling, volume 9, pages 1–36. Kluwer Academic Publishers 1997.

43 Niang, P., Grandpierre, T., Akil, M., and Sorel, Y.. Aaa and syndex-ic: A methodology and a software framework for the implementation of real-time applications onto reconfigurable circuits. In Becker, J., Platzner, M., and Vernalde, S., editors, conference of the 14th International Conference in Field Programmable Logic and Application, pages 1119–1123. 2004.

44 Thomas, D. E., Dirkes, E., Walker, R., Rajan, J., Nestor, J., and Blackburn, R. The system architect's workbench. In conference of the 25th ACM/IEEE Conference on Design Automation, pages 337–343, USA. 1988.

45 Grode, J., Voigt Knudsen, P., and Madsen, J. Hardware resource allocation For hardware/software partitioning in the lycos system. In Design, Automation and Test in Europe (DATE '98), pages 22–27. 1998

46 Gupta, R. And demicheli, G. Hardware-software cosynthesis for digital systems. In IEEE Design and Test of Computers, volume 10, pages 29–41 1993..

47 Gajski, D., Vahid, F., Narayan, S., and Gong, J. System-level exploration With specsyn. In conference of the 35th Conference on Design Automation, pages 812–817. ACM Press. 1998.

48 Eker, j., janneck, j. W., lee, e., liu, j., liu, x., ludvig, j., neuendorffer, s., sachs, J., and xiong, y.Taming heterogeneity - the ptolemy approach. Volume 91, pages

127–144. 2003.

49 Sedmak, R. M. And Evans, J. A hierarchical, desgin-for-testability (dft) methodology for the rapid prototyping of application-specific signal processors (rassp). In conference IEEE International Test Conference, pages 319–327. 1995

50 Rethman, N. And Wilsey, P. Rapid: A tool for hardware/software tradeoff analysis. In conference of IFIP Conference on Hardware Description Languages, Ottawa, Canada. Elsevier Science.1993

51 De Man, H., Bolsens, I., Lin, B., Van Rompaey, K., Vercauteren, S., and Verkest, D. Co-design of dsp systems. In NATO ASI Hardware/Software Co-Design.1995

52 J. M. Daveau, g. F. Marchioro, c. A. Valderrama and a. A. Jerraya, “vhdl generation from sdl specification”, in carlos d. Kloos and eduard cerny, editors, proceedings of chdl, pages 182--201. Ifip, chapman-hall, 1997.

53 A. Österling, t. Benner, r. Ernst, d. Herrmann, t. Scholz,w.ye, “the cosyma system”, hardware /software co-design: principles and practice, kluwer academic publishers, j. Staunstrup and w. Wolf edditors, pp. 263-305, 1997.

54 J. Madsen, j. Brage, “codesign analysis of a computer graphics application”, journal: design automation of embedded systems, vol.1, no.1-2, 1996.

55 N. Gehani, "c: an advanced introduction, ansi c edition", computer science press, 1988.

56 K. Van rompaey, d. Verkest, i. Bolsens, h. De man, “coware- a design environment for heterogeneous hardware/software systems”, in proceedings of the european design automation conference euro- dac’96, geneva, switzerland, p252,. 1996.

57 G. F. Marchioro découpage tranformationnel pour la conception de systémes Mixtes logiciel/matériel, thése de doctorat, institut national polytechnique de Grenoble, pages. 32, 35, 37 1998

58 R. Niemann et p. Marwedel an algorithm for hardware / software partitioning Using mixed integer linear programming, design automation for embedded Systems, pages. 36 1997

59 A. Osterling, t. Benner, r. Ernst, d. Herrmann, t. Scholz et w. Ye cosyma:cosynthesis for embedded architectures, http://www.ida.ing.tubs.de/projects/cosyma/.

60 R. K. Gupta et g. De micheli hardware-software co-synthesis of digital Systems, ieee design and test of computers, vol. 3, , pages . 29. 1993

61 F.P. Hessel, "Conception des systèmes hétérogènes multilangages", Thèse de Doctorat, UNIVERSITE JOSEPH FOURIER, 2000.

62 Luk, w., lok, v. Et, 1. Hardware acceleration of divide and conauer paradimm: a case studv, proceedings of the ieee workshop on fpgas for custom computing

machines. 1993

63 KOCH, A, et GO, An FPGA Based Co-Processor for sbus Workstations, 3* International Workshop on Field-Programmable Logic and Applications- 1993

64 Jantsch, a., ellervee, p., oberg, j. Et kemani, a. A case studv on hardware/software partitioning, proceedings of the ieee workshop on fpgas for custom computing machines, pp- 1 i 1- 1 18. 1994

65 Carreras, c-, lopez, j. C., lopez, m. L., delgado-kloos, c., martinez, n- et sanchez, l, A co-design methodologv based on formal specification and high-level estimation, international workshop on hardware/software co-design, vol- 4,pages28-35. (1996).

66 developpement d'estimateurs de performance pour des applications de co-design matérel~ogiciel lé vis thériault &moire présenté en vue de l'obtention du diplôme de ma~trise ès sciences appliquées (h4.sc.a.) (génie électrique) 2000

67 J. Madsen, j. Grode, p. Knudsen, m. Petersen et a. Haxthausen lycos: then lyngby co-synthesis system ¯, design automation for embedded systems, vol. 2, depatment of information technology, technical university of denmark, 1997.

68 Modélisation de Systèmes Intégrés Numériques Introduction à VHDL Alain Vachoux, Hiver 2002-2003

69 V. P. Vijayaraghavan { ® exploration des liens entre la synthése de haut niveau (hls) et la synthése au niveau transferts de registres (rtl) ¯, thése de doctorat, l'institut national polytechnique de grenoble pages 40, 1992.

70 Giovanni De Micheli, Synthesis and Optimization of Digital Circuits, Mac Graw-Hill, 1994.

71 RUNDENSTEINER, E- A- et GAJSKI, DA Desin Representation Mode1 for Hi oh-level-Svnthesis, Technicd report, University of Califomia, Irvine . 1990.

72 Composano, r. From behavior to structure: high-level svnthesis, ieee design & test of computers, pages. 8- 19. 1990.

73 SYSTEMC home page, httd://www.s~stemc.or.g~

74 N. Altman and n. Weiderman. Timing variations in dual loop benchmarks. Ada Letters, 8(3):98 106, 1989.

75 A. Ermedahl and j. Gustafsson. Automatic derivation of path and loop annotations In object-oriented real-time programs. Journal of parallel and distributed computing practices, 1(2):61 74, 1998.

76 I. Bate, g. Bernat, g. Murphy, and p. Puschner. Low-level analysis of a portable Wcet analysis framework. In proc. Of the 7th international conference on Real-time computing systems and applications, pages 39,48, 2000.

77 A. Colin and i. Puaut. Analyse de temps d'exécution au pire cas du systéme

d'exploitation temps-réel rtems. In seconde conférence fran_caise sur les systémes D'exploitation, pages 73, 84, paris, france, 2001.

78 P. Puschner and a. V. Schedl. Computing maximum task execution times a graph based approach. In proc. Of ieee real-time systems symposium, volume 13, pages 67;91. Kluwer academic publishers, 1997.

79 G. Ottosson and m. Sj odin. Worst-case execution time analysis for modern hardware architectures. In acm sigplan workshop on languages, compilers, and tools support for real-time systems (lctrts'97), 1997.

80 S. M. Petters and g. Farber. Making worst case execution time analysis for hard real-time tasks on state of the art processors feasible. In proc. Of the 6th international conference on real-time computing systems and applications, 1999.

81 F. Stappert and p. Altenbernd. Complete worst-case execution time analysis of straight-line hard real-time programs. Journal of systems architecture, 46(4):339 355, 2000.

82 F. Bodin, e. Rohou, and a. Seznec. Salto: system for assembly-language transformation and optimization. In proc. Of the sixth workshop on compilers for Parallel computers, 1996.

83 D. Macos and f. Mueller. Integrating gnat/gcc into a timing analysis environment. In work-in-progress of the 10th euromicro conference on real-time systems, pages 15,18, 1998.

84 C. Healy, m. Sj odin, v. Rustagi, d. Whalley, and r. Van engelen. Supporting timing analysis by automatic bounding of loop iterations. Real-time systems, 18(2-3):129,156, may 2000.

85 R. Chapman, a. Burns, and a.j. wellings. Combining static worst-case timing analysis and program proof. Real-time systems, 11(2):145,171, 1996.

86 L. Ko, d. B. Whalley, and m. G. Harmon. Supporting user-friendly analysis of Timing constraints. Acm sigplan notices, 30(11):99,107, 1995.

87 E. W. Dijkstra. A note on two problems in connexion with graphs. Numerische Mathematik, 1:269,271, 1959.

88 A. Colin and i. Puaut. Worst case execution time analysis for a processor with branch prediction. Real-time systems, 18(2-3):249, 274, 2000.

89 A. Colin and i. Puaut. A modular and retargetable framework for tree-based wcet analysis. In proc. Of the 13th euromicro conference on real-time systems, pages 37{44, delft, the netherlands, 2001.

90 A. Colin and i. Puaut. Worst-case execution time analysis of the rtems realtime operating system. In proc. Of the 13th euromicro conference on real-time systems, pages 191{198, delft, the netherlands, 2001.

91 J. Schneider and c. Ferdinand. Pipeline behavior prediction for superscalar processors by abstract interpretation. In acm sigplan workshop on languages, compilers, and tools for embedded systems, pages 35,44, 1999.

92 J. Hennessy and d. Patterson. Computer organization and design. The hardware/ software interface. Morgan kaufmann, inc., 1994.

93 S.-k. Kim, s. L. Min, and r. Ha. E_cient worst case timing analysis of data caching. In proceedings of the 1996 real-time technology and applications symposium, pages 230{240. Ieee computer society press, 1996.

94 Y.-t. S. Li, s. Malik, and a. Wolfe. Cache modeling for real-time software: beyond direct mapped instruction cache. In proceedings of the 17th ieee real-time systems symposium (rtss96), pages 254{263. Ieee, ieee computer society press, 1996.

95 R. T. White, f. Mueller, c. A. Healy, d. B. Whalley, and m. G. Harmon. Timing Analysis for data and wrap-around ll caches. Real-time systems, 17(2-3):209 233, 1999.

96 J. Schneider and c. Ferdinand. Pipeline behavior prediction for superscalar processors by abstract interpretation. In acm sigplan workshop on languages, compilers, and tools for embedded systems, pages 35, 44, 1999.

97 S.-s lim, j. H. Han, and s. L. Min. A worst case timing analysis technique for superscalar processors. Technical report, departement of computer engineering, seoul national university, 1997.

98 B.-d. Rhee, s.-s. Lim, s. L. Min, c. Y. Park, h. Shin, and c. S. Kim. Issues of advanced architectural features in the design of a timing tool. In proc. Of the 11th workshop on real-time operanting systems and software, pages 59,62, 1994.

99 S. Basumallick and k. Nilsen. Cache issues in real-time systems. In acm sigplan workshop on languages, compilers, and tools for embedded systems, 1994.

100 B. W. Char, k. O. Geddes, and g. H. Gonnet. Maple v language reference manual. Springer-verlag, 1991.

101 Y.-t. S. Li and s. Malik. Performance analysis of embedded software using implicit Path enumeration. Acm sigplan workshop on languages, compilers, and tools for embedded systems, 30(11):88, 98, 1995.

102 C. Ferdinand, f. Martin, and r. Wilhelm. Applying compiler technique to cache behavior prediction. In acm sigplan workshop on languages, compilers, and tools for embedded systems, pages 37,46, 1997.

103 J. V. Busquets-mataix, j. J. Serrano, r. Ors, p. Gil, and a.wellings. Adding instruction cache to schedulability analysis of preemptive real-time systems. In proceedings of the 1996 real-time technology and applications symposium, pages 204{212. Ieee computer society press, 1996.

104 N. Zhang, a. Burns, and m. Nicholson. Pipelined processors and worst case execution times. Real-time systems, 5(4):319,343, 1993.

105 R. Arnold, f. Mueller, d. Whalley, and m. Harmon. Bounding worst-case instruction cache performance. In proceedings of the 15th ieee real-time systems symposium (rtss94), pages 172,181, 1994.

106 M. Alt, c. Ferdinand, f. Martin, and r. Wilhelm. Cache behavior prediction by abstract interpretation. In sas'96, static analysis symposium, volume 1145 of lecture notes in computer science, pages 51/66. 1996.

107 G. Bernat, a. Burns, and a. Wellings. Portable worst-case execution time analysis Using java byte code. In proc. Of the 12th euromicro workshop of real-time systems, pages 81/ 88, 2000.

108 I. Bate, g. Bernat, g. Murphy, and p. Puschner. Low-level analysis of a portable wcet analysis framework. In proc. Of the 7th international conference on real-time computing systems and applications, pages 39,48, 2000.

109 C. Ferdinand, f. Martin, and r. Wilhelm. Applying compiler technique to cache behavior prediction. In acm sigplan workshop on languages, compilers, and tools for embedded systems, pages 37,46, 1997.

110 T. Lundqvist and p. Stenstrom. Integrating path and timing analysis using instruction-level simulation techniques. In acm sigplan workshop on languages, compilers, and tools for embedded systems, pages 1,15, , 1998.

111 Y. A. Liu and g. Gomez. Automatic accurate time-bound analysis for high-level languages. In acm sigplan workshop on languages, compilers, and tools for embedded, 1998.

112 Y.-t. S. Li, s. Malik, and a. Wolfe. E_cient microarchitecture modeling and path analysis for real-time software. In proceedings of the 16th ieee real-time systems symposium (rtss95), pages 298{307, pisa, italy, 1995.

113 S.-s lim, s. L. Min, m. Lee, c. Y. Park, h. Shin, and c.-s. Kim. An accurate instruction cache analysis technique for real-time systems. Ieee workshop on architectures for real-time applications, 1994.

114 S.-k. Kim, s. L. Min, and r. Ha. E_cient worst case timing analysis of data caching. In proceedings of the 1996 real-time technology and applications symposium, pages 230{240. Ieee computer society press, 1996.

115 C. Y. Park. Predicting program execution times by analyzing static and dynamic program paths. Real-time systems, 5(1):31/62, 1993.

116 C. Y. Park and a. C. Shaw. Experiments with a program timing tool based on source-level timing schema. Ieee computer, 24(5):48{57, may 1991.

117 P. Puschner and a. Schedl. Computing maximum task execution times with linear programming techniques. Technical report, technische universitat, institut fur technische informatik, wien, april 1995.

118 C. Healy, m. Sjodin, v. Rustagi, and d. Whalley. Bounding loop iterations for timing analysis. In fourth ieee real-time technology and applications symposium, pages 12{21, june 1998.

119 C. Healy, r. Arnold, f. Mueller, d. Whalley, and m. Harmon. Bounding pipeline and instruction cache performance. Ieee transactions on computers, 48(1), january 1999.

120 F. Mueller. Generalizing timing predictions to set-associative caches. In proc. Of euromicro workshop on real-time systems, pages 64{71, june 1997.

121 J. Engblom. Static properties of commercial embedded real-time programs, and their implication for worst-case execution time analysis. In proceedings of the 1999 real-time technology and applications symposium, pages 46{55, vancouver, ca, june 1999.

122 J. Engblom and a. Ermedahl. Modeling complex ows for worst-case execution time analysis. In proceedings of the 21th ieee real-time systems symposium (rtss00), orlando, florida, december 2000.

123 P. Persson and g. Hedin. Interactive execution time predictions using reference Attributed grammars. In d. Parigot and m. Mernik, editors, second workshop On attribute grammars and their applications, waga'99, pages 173/184, amsterdam,, the netherlands, 1999.

124 P. Puschner and c. Koza. Calculating the maximum execution time of real-time programs. Real-time systems, 1(2):159/176, september 1989.

125 Diagne a., "systèmes répartis et coopératifs: une approche multi-formalismes de spécification de systèmes répartis: transformation de composants modulaires en réseaux de petri", thèse de doctorat, université paris 6, 1997.

126 MINT GROUP PROJECT UPC, "System Level Specification and Verification ", 2000, http://mint.cs.man.ac.uk/Projects/UPC/Reviews/specificationandverification.html

127 P. Puschner. A tool for high-level language analysis of worst-case execution times. In proc. Of the 10th euromicro conference on real-time systems, pages 130/137, berlin, germany, june 1998.

128 P. Puschner and c. Koza. Calculating the maximum execution time of real-time programs. Real-time systems, 1(2):159{176, september 1989.

129 Ye, w., *ernst*, r., be-, t. Et kenkel, j. fast timing analysis for hardware/software co-synthesis, in proc. Of iccd, pp. 452, 457. 1993.

130 Smith, t. E. Et setliff, d- e. towards an automatic synthesis system for red-time

software, in proc. Of 12'h real-time systems symposium, pp. 34-42. 1991

131 Gajski, d. D., vahid, f., narayan, s. Et gong, j. specifications and desim of embedded svstems, ptr prentice hall, new jersey, 450 p. 1994

132 Gupta, r. K. Et de micheli, g. Constrained software generation for hardware-software svstems, in proc. Of int. Workshop on hardware/software codesign, pp. 56-63. 1994.

133 Hardt, w. Et camposano, r. Trade-offs in hw/sw codesien, in proc. Of h t. Workshop on hardware/software codesign, oct. 1993.

134 Gong, j., gajski, d. D. Et sanjn, . Software estimation from executable soecifications, journal of computer & software engineering, vol. 2, no. 3, pp. 239-258. 1994.

135 CINDERELLA home page, http://www.ee.~rinceton-edd-vauli/cinderelia-

136 Knfeser, M, J. Et PAPACHRISTOU, *C*- A. COMET: A Hardware-Software Codesign Methodoloav, EURO-DAC 1996.

137 Wolf, w. Et martdez, j. *C*- *C* program performance estimation for embedded svstems architecture sizinq, in proc, of int- workshop on hardwareisoftware codesign. 1993.

138 Ball, t. Et larus, j. R.- efficient path profiling;, ieee/acm 29" international symposium on microarchitecture, paris, france, pp. 45-57. 1996

139 Ball, t. Et larus, j- *r* Optirnallv ~rofiling: and tracinp prograns, acm transactions on programming languages and systems. Vol, 16, no. 4, pp. 1329-1360. (1994)

140 D'ambrosio, j. *G.* Et hu, x. confipuration-level hardware/software partitioning for real-time ernbedded systerns, in proc. Of int. Workshop on hardware/software codesign, pp, 34-41. 1994

141 Ball, t. Et larus, j. R. Optimdiv rofilinc and tracinq promams, in proceedings of the acm sigplan 92 conference on principles of programming languages, pp. 59-70, (1992).

142 Conte, t. M., menezes, k. N. Et kirsch, m. A Accurate and practice profile-driven compilation usina the profile buffer, proceedings of the 29& annual ieee/acm international symposium on microarchitecture, pp. 36-45. . (1996).

143 M.L. Kherfi, and S.-A.-A. Touati. CODOO: Un Environnement pour le Codesign hw/SW à Partir d'une Co-Spécification Orientée Objets. Computer Science Engineer Dissertation, Special field: Computer Systems. Institut National d'Informatique, Algiers, Algeria, September, 1997.

144 Hervé Marchand, Eric Marchand, Eric Rutten. « Speciofication et verification de système reactifs: experiimentation de la methodologie synchrone synchrone

SIGNAL » IRISA-INRIA Rennes-Université de Rennes. 1996.

145 Halbwachs n., capsi p., raymond p., pilaud d programmation et verification des systèmes reactifs le langage lustre, tsi-techniques et sciences informatiques, vol 10, n°2, 1991

146 J. Engblom, a. Ermedahl, and f. Stappert. A worst-case execution-time analysis tool prototype for embedded real-time systems. In *workshop on real-time tools (rt-tools 2001) held in conjunction with concur 2001*, aalborg, denmark,august 2001.

147 A. Colin. *Estimation de temps d'execution au pire cas par analyse statique et Application aux systµemes d'exploitation temps-réel.* Phd thesis, universit¶e de rennes i, octobre 2001.

148 Rtems. Rtems: real time operating system for multiprocessor systems. Http://www.rtems.com/.

149 Architecture des ordinateurs Sylvain Tisserant
Http://marpix1.in2p3.fr/calo/my-web/archi/chap7/page11.html