وزارة التعليم العالي و البحث العلمي

BADJI MOKHTAR-ANNABA UNIVERSITY
UNIVERSITE BADJI MOKHTAR-ANNABA

جامعة باجي مختار - عنابة

Faculté des Sciences de l'Ingénieur **Année :2010**

Département d'Informatique

MEMOIRE

Présenté en vue de l'obtention du diplôme de **MAGISTER**

Problème d'optimisation de ressources pour systèmes embarqués

Option :

Informatique Industrielle

Par :

Haouari Aouatef

DIRECTEUR DE MEMOIRE : Mohamed BENMOHAMED Pr Université de Constantine

DEVANT LE JURY

PRESIDENT : **Rachid BOUDOUR** MCA Université d'Annaba

EXAMINATEURS :

Mohamed Tahar KIMOUR MCA Université d'Annaba

Résumé

Les systèmes embarqués sont des systèmes critiques et complexes. Leur optimisation devient donc une nécessité afin de satisfaire les contraintes strictes imposées par ce type de systèmes. L'optimisation peut porter sur trois aspects différents : l'optimisation mémoires, l'optimisation énergétique, et l'optimisation de la vitesse d'exécution. Afin d'améliorer ce dernier critère, plusieurs outils de conception exploitent le parallélisme potentiel de l'application, le parallélisme effectif de l'architecture matérielle, et faire une adéquation entre les deux, et cela lors de l'étape d'ordonnancement.

Parmi ces outils l'outil SynDEx. Il permet d'implanter les applications orientées données, spécifiés avec les graphes de flot de données, sur des architectures multiprocesseur hétérogènes en optimisant la durée totale d'exécution lors de l'étape d'adéquation et ordonnancement, ces deux étapes sont réalisées simultanément par un algorithme basé sur une heuristique.

En effet, une description flot de données d'une application par l'utilisateur permet de détecter et d'exploiter le parallélisme de l'application (en définissant une sémantique aux accès aux données), de garantir les performances à l'exécution (en utilisant de manière implicite un ordre total d'exécution des tâches), et de fournir une information précise sur l'application à l'ordonnanceur.

Pour atteindre les meilleures performances des applications flot de données, nous proposons d'introduire la technique de resynchronisation dans l'étape de modélisation. Cette technique est une méthode de transformation de graphe qui permet de réduire le temps de cycle des applications récursives et itératives comme les applications destinées à être exécutées sur les processeurs de traitement de signal (DSP).

Mot clés : Spécification, SynDEx, graphe flot de données, resynchronisation, optimisation.

Abstract

Many common iterative or recursive DSP applications can be represented by data-flow graphs (DFG). Dataflow model is used for high-level specification of several types of signal and image processing applications. Their use is justified by their ability to highlight the potential parallelism of the application, data dependencies, and the order of operations. In addition, this model is formal, which makes it interesting because it allows complex mathematical reasoning.

SynDEx is a tool for implementing data-driven applications on heterogeneous multiprocessor architectures, with an optimization of the total time execution at the implementation stage. We will propose to improve the total execution time but in the stage of specification, in order to achieve the best performance. We will use retiming method that minimizes the cycle time of the application modelling with DFG. This technique is used to reduce the clock period in synchronous circuits, and cycle time in signal processing algorithms, by reducing the computation time of the critical path, while the application's function remains unchanged.

Key words: Specification, SynDEx, data flow graph, retiming, optimisation.

ملخـــــص

إن الأداة *SynDEx* هي أداة تستعمل في تصميم الأنظمة المضمنة (*Embedded System*)، هذه الأداة تقوم بتحسين الوقت الكلي لعمل النظام و ذلك في مرحلة دمج التطبيقات مع البنية العتادية للنظام , و تعتمد على منحنيات تدفق البيانات (*Data Flow Graph*) كنموذج وصف لمختلف البرامج.

و من أجل تحسين و الوصول إلى الوقت الأمثل لعمل التطبيقات الخاصة بمعالجة الصورة و الإشارات الرقمية *(Digital Signal Processing)* التي تتكون غالبا من حلقات التكرار، و التي تمثل غالبا بواسطة منحنى تدفق البيانات (*DFG*). وباعتبار أن تحسين الوقت يعتبر من أهم أهداف تصنيع الأنظمة المضمنة، قمنا بتحسين الوقت الكلي لعمل البرنامج قبل مرحلة دمجه بالعتاد وبالضبط في مرحلة وصف البرامج، و ذلك باستعمال تقنية إعادة المزامنة (*Retiming*)، التي تتمثل في تحسين وقت البرنامج دون تغيير الوظيفة الأساسية لهذا الأخير.

كلمات المفتاح: منحنى تدفق البيانات , تقنية إعادة المزامنة (*Retiming*), الأداة *SynDEx*, تحسين وقت البرنامج

Remerciements

Je tiens tout d'abord à remercier monsieur Mohamed Benmohamed, qui m'a encadré tout au long de cette thèse, et pour ces conseils avisés. Je lui suis reconnaissante pour la disponibilité dont il a fait preuve et pour la confiance qu'il m'a accordée.

Je tiens aussi à remercier monsieur Rachid Boudour, maitre de conférence, pour m'avoir fait l'honneur de présider le jury de cette thèse.

J'adresse mes remerciements également à monsieur Mohamed Tahar Kimour, maitre de conférence, et monsieur Azzedine Bilami, maitre de conférence, pour avoir accepté de juger mon travail en tant que rapporteurs.

Ces remerciements ne seraient pas complets sans mes pensées pour ma famille, pour m'avoir soutenue et m'avoir permis de réaliser mes études dans les meilleures conditions.

Table Des Matieres

Table des figures

Introduction :

Les systèmes embarqués sont des composants qui intègrent du logiciel et du matériel et assurent des fonctionnalités critiques. Ils occupent une place de plus en plus importante dans le monde qui nous entoure. Ils se caractérisent par une interaction continue avec leur environnement physique. Ils trouvent leur application dans de nombreux domaines comme les équipements industriels lourds (centrale nucléaire, chaîne de fabrication, avionique, systèmes d'armes), ainsi que des produits grands publics (automobile, téléphone, domotique, équipements hi fi et vidéo).

Ces systèmes sont composés de deux parties qui interagissent. La première correspond à un système informatique lui-même composé d'un calculateur qui exécute un ensemble de programmes. Ces derniers forment le logiciel du calculateur, ils renferment les algorithmes de l'application. La seconde partie d'une application correspond à son environnement physique, dont les changements d'état, sont perçus par le calculateur au moyen de capteurs.

La conception des systèmes embarqués exige des méthodes et outils permettant de prendre en compte dans ses phases amont, non seulement des exigences fonctionnelles concernant la correction du calcul mais également des exigences extra- fonctionnelles relatives à l'utilisation optimale de ressources telles que le temps, la mémoire, l'énergie,... À ces exigences s'ajoutent celles de l'autonomie, la réactivité et la robustesse. L'objectif des outils de conception des systèmes embarqués est d'aider le concepteur à spécifier et vérifier rapidement, générer automatiquement des exécutifs conformes à la spécification, et assurer une grande maîtrise de la sûreté de fonctionnement. Dans ce domaine on trouve la méthodologie AAA (adéquation algorithme architecture) et l'outil SynDEx. Cette méthodologie a été conçue et utilisée avec succès pour la réalisation de systèmes temps réel dont l'architecture matérielle est de type multiprocesseurs. Elle autorise la spécification des algorithmes à l'aide de graphes flots de données, et une spécification de l'architecture par des graphes. L'optimisation de temps d'exécution dans la méthodologie AAA se fait lors d'implantation de l'application sur l'architecture matérielle, et plus précisément dans l'étape de distribution et ordonnancement.

Les applications embarquées, sont des applications qui reposent principalement sur des algorithmes de traitement du signal et d'images qui nécessitent d'importantes quantités de calculs, lorsqu'ils doivent être effectués en un temps court.

Dans le but d'atteindre les meilleures performances, pour ce type d'application, on va optimiser le temps d'exécution des applications embarqués modélisé par SynDEx, lors de l'étape de spécification. Pour cela on va introduire la technique de resynchronisation dans la phase de modélisation. Cette technique est une méthode d'optimisation des applications de traitement de

données, elle repose sur la mobilisation des éléments retards dans le graphe, et cela sans affecter le fonctionnement principal de l'application.

Plan du mémoire

Dans le premier chapitre, nous introduirons les systèmes embarqués et les enjeux de conception de ce type de système. Ensuite on va présenter les divers modèles mathématiques de modélisation des systèmes embarqués dans le chapitre deux, qui se conclura par une motivation sur le choix du modèle graphe flot de données.

Le chapitre trois constituera une présentation (non exhaustive) des outils de conception qui utilise le modèle de graphe de flot de données comme un modèle de spécification.

Le chapitre quatre est une introduction à notre proposition, il s'agit d'une présentation de la technique de resynchronisation.

Le dernier chapitre de ce mémoire présente des exemples pratiques modélisés par SynDEx, et les résultats de l'application de cette méthode sur quelques exemples pratiques.

Les systèmes embarqués

Dans cette première partie, nous définisserons le domaine d'étude des systèmes embarqués temps réel. Les systèmes sont vus au travers de leurs contraintes qui influencent fortement leur processus de conception.

1. Introduction

Du fait des avancées technologiques qui permettent une plus grande miniaturisation des systèmes, les systèmes embarqués temps réel, souvent cachés aux utilisateurs, sont de plus en plus présents dans notre environnement et de plus en plus complexes. Les applications les plus connues des systèmes embarqués temps réel sont les systèmes de transport (voiture, avion, train) et les systèmes mobiles autonomes (robot, fusée, satellite). De même, les systèmes liés à la gestion d'un périphérique (imprimante, souris sans fil), à la mesure (acquisition en temps réel) et les systèmes domotiques (électroménager) sont des systèmes embarqués pouvant posséder des contraintes temps réel liées aux capteurs ou actionneurs utilisés. La notion d'embarqué peut être étendue aux objets portables grand public (cartes à puce, assistants personnel, téléphones mobiles, lecteurs, vidéo, consoles de jeu). Le point commun de tous ces systèmes porte sur la spécificité de leurs contraintes [Jea05]**.**

L'attrait principal des systèmes embarqués vient du fait qu'ils permettent d'implémenter à faible coût des fonctions complexes dont la réalisation était inimaginable il y a quelques années seulement. Aujourd'hui, la tendance générale est d'utiliser les systèmes embarqués pour incorporer des fonctionnalités complexes, précédemment considérées comme exotique, dans les produits de tous les jours. En plus, les systèmes embarqués possèdent des caractéristiques spécifiques.

Par exemple, l'interface IHM d'un système embarqué, contrairement à un PC, peut-être aussi simple qu'une diode électroluminescente (LED) qui clignote ou aussi complexe qu'un système de vision de nuit en temps réel ; les afficheurs à cristaux liquides (LCD) de structure généralement simple sont couramment utilisés. Afin d'optimiser les performances et la fiabilité de ces systèmes, des circuits numériques programmables (FPGA), des circuits dédiés à des applications spécifiques (ASIC) ou des modules analogiques sont en plus utilisés. Le logiciel a une fonctionnalité fixe à exécuter qui est spécifique à une application. L'utilisateur n'a pas la possibilité de modifier les programmes.

2. Caractéristiques des systèmes embarqués

Les systèmes embarqués ont pour but de permettre aux objets usuels de réagir à l'environnement. Ils peuvent aussi apporter une interface avec l'utilisateur. La structure de base de ces systèmes est présentée dans la figure 1.1 : l'environnement est mesuré par divers capteurs. L'information des capteurs est échantillonnée pour être traitée par le cœur du système embarqué. Puis le résultat du traitement est converti en signaux analogiques qui génèrent les actions sur

l'environnement (afficheur d'informations pour l'utilisateur, actionneurs, transmission d'information, etc.).

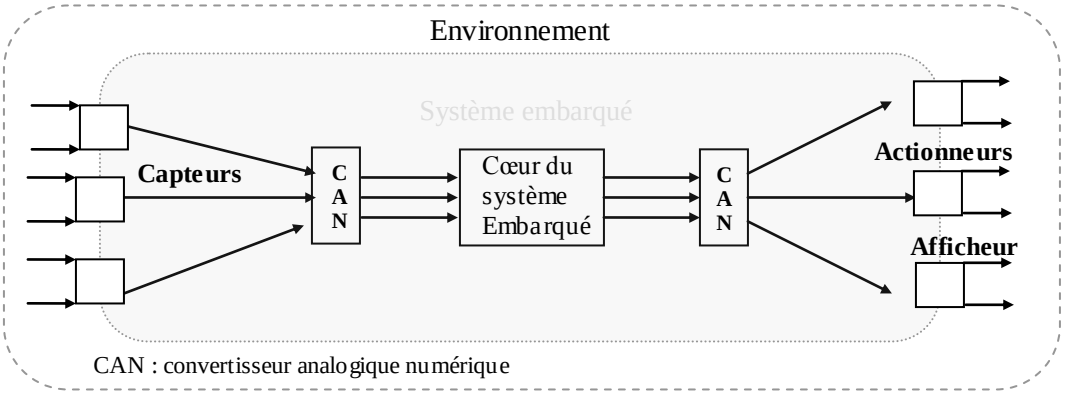


Figure 1.1 : Un système embarqué dans son environnement.

3. Les contraintes des systèmes embarqués

Les systèmes embarqués ont pour caractéristique commune d'être contraints. Il s'agit aussi bien de contraintes sur le support d'exécution que de contraintes de Qualité de Service (QdS) [Bab05]. Selon la machine sur laquelle un programme embarqué est destiné à être exécuté, il peut avoir à satisfaire différentes de ces contraintes :

- **L'encombrement** : ils doivent souvent être transportés et doivent donc être de taille réduite ;
- **L'utilisation mémoire :** La miniaturisation du matériel entraîne dans certains cas une réduction des capacités mémoires de l'appareil. Le programme embarqué doit donc tenir compte de cette contrainte et, par exemple, ne pas dépasser certaines limites d'occupation mémoire statique et d'utilisation mémoire lors de son exécution.
- **Le temps de calcul :** L'utilisation de programmes embarqués se fait souvent dans des contextes où le temps est un paramètre essentiel du système. Les délais d'exécution sont alors connus et bornés. De tels systèmes ont des propriétés temps réel dur ou mou.
- **La consommation d'énergie :** Le caractère embarqué d'un système fait que ce dernier n'est pas toujours relié à une source infinie d'énergie. Le système doit alors gérer une certaine quantité d'énergie fournie par une batterie autonome (panneaux solaires, pile).
- **La sûreté/sécurité :** Certaines pannes de systèmes embarqués peuvent avoir des conséquences désastreuses tant d'un point de vue humain (train d'atterrissage d'un avion, appareillage médical) que d'un point de vue économique (système d'exploitation d'un distributeur d'argent). De tels systèmes sont dits critiques [Ven08].

- **Forte contraintes :** En raison de la nature des systèmes embarqués, la conception des indicateurs tels que la taille, la performance et la consommation imposent de fortes contraintes [all05].

Vu la diversité des domaines d'application et des contraintes à considérer, un système embarqué peut avoir à satisfaire toutes ou parties de ces contraintes bien qu'elles puissent être contradictoires. L'exemple type est le gain de temps d'exécution d'une boucle ordonnancée par pipeline logiciel. Cette optimisation augmente la taille du code généré et donc son occupation en mémoire. Si un programme devait être optimisé de manière à augmenter sa vitesse et diminuer sa taille, le pipeline logiciel demanderait de faire des compromis. La demande d'optimisation du programme à exécuter est très forte, voire primordiale.

Application	Sensibilité au coût	Mobilité	Rudesse de l'environnement	Haute fiabilité	Criticité
Électroménager et audio-visuel	√			√	
Téléphonie mobile	√	√		√	
Transport					
Automobile	√	√	√	√	√
Ferroviaire		√	√	√	√
Aérien		√	√	√	√
Maritime		√	√	√	√
Aérospatial					
Fusée navette	√	√	√	√	√
spatial	√	√	√	√	√
Carte à puce		√		√	

Tableau 1.1 : Propriétés des applications incorporant les systèmes embarqués. [Sim05]

Le tableau 1.1, présentes quelques applications embarquées avec les différentes contraintes qu'elles exigent.

4. Conception des systèmes embarqués

La conception des systèmes embarqués doit être capable d'implémenter une application avec une garantie des propriétés des systèmes critiques, mais elle doit également satisfaire la complexité et l'hétérogénéité de l'architecture. Dans la tendance actuelle de conception industrielle, le coût de la vérification par rapport au coût total de la conception est en

augmentation. Les conceptions actuelles dépendent fortement sur les techniques de simulation (qui seront très important dans le futur), mais qui sont besoin de méthodes formelles de vérification.

Mais dans la pratique il est très difficile de capturer les fonctionnalités du système et vérifier les propriétés au niveau système à celle au niveau composant de l'architecture, la conception du système doit commencer dans un niveau d'abstraction plus élevé. Malheureusement, plus le niveau d'abstraction est élevé, plus que les détails de l'implémentation sont manqués, et qui seront nécessaire pour une implémentation efficace. Le processus de conception doit être capable d'ajouter les détails de l'implémentation.

La conception système commence par le développement d'un modèle de spécification. Dans cette phase le concepteur formule le modèle selon les exigences données par la spécification, qui est généralement écrite dans le langage naturel, par exemple en français. Il est important que le modèle de spécification soit exprimé dans un langage formel. Un langage formel est un langage qui a une syntaxe et une sémantique formelle, et qui permettent aux outils et la manipulation formelle de détecter les inconsistances, ambiguïtés ou l'incomplétude dans le modèle de spécification.

Plus que le niveau d'abstraction est élevé dans l'étape de spécification, moins que les détails de l'implémentation sont inhérents dans le modèle d'implémentation et l'espace de conception est plus large. L'espace de conception est défini comme l'ensemble des implémentations possibles qui répond à un modèle de spécification donné (Figure 1.2).

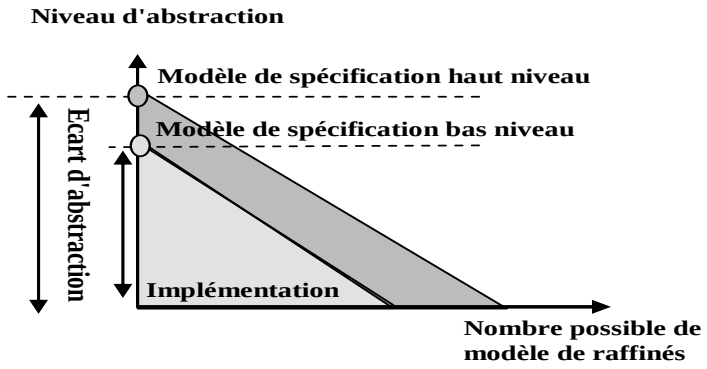


Figure 1.2 : Le niveau d'abstraction est élevé, l'espace de conception est plus large [San03]

Le niveau d'abstraction élevé signifié également que la description du système peut être exprimée d'une manière claire et simple, car les détails de l'implémentation ne sont pas pris en considération. Un modèle simple aide le concepteur à comprendre et formuler le fonctionnement du système, avec moins de détails. En plus, un modèle simple permet aussi une vérification efficace du système, car ce modèle de spécification répond à toutes les contraintes imposées à ce dernier.

D'autre part un niveau d'abstraction élevé du modèle système, rend le processus de la synthèse plus difficile, car le modèle de spécification contenant un nombre restreint de détails système, doit être transformé en une implémentation sur une architecture complexe et hétérogène, qui nécessite un nombre important de détails. Cela va induit un large écart dans l'abstraction (figure 1.3), qui doivent être comblé dans la synthèse.

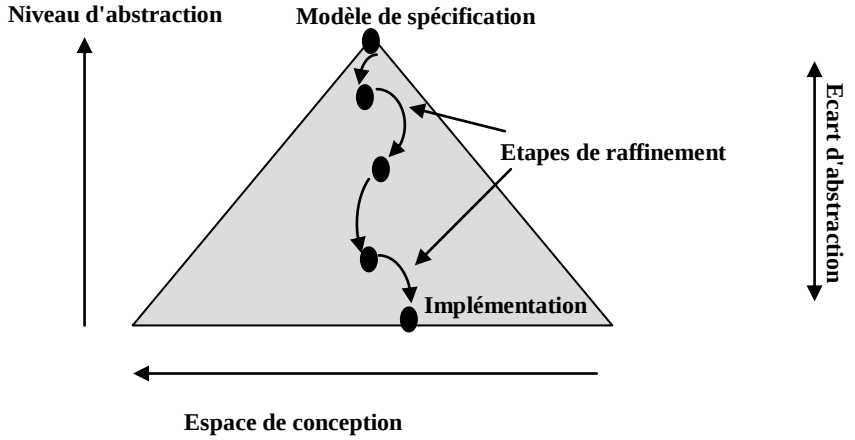


Figure 1.3 : La synthèse est un processus de raffinement par étapes à partir d'un niveau élevé du modèle de spécification en une implémentation finale.

En résumé, une méthodologie de conception doit satisfaire les besoins suivants :

- La possibilité de modéliser le système dans un niveau d'abstraction élevé ;
- Un processus de synthèse qui permet de combler l'écart d'abstraction à fin d'assurer une implémentation efficace.

Ces objectifs peuvent être définis comme le défi d'une méthodologie de conception réussie des systèmes embarqués.

5. Les types des systèmes embarqués

En général, Les fonctionnalités des systèmes temps réel embarqués combinent les aspects contrôle et les aspects traitement de données, ce type de système peut être classifié selon l'aspect qui dominent leurs fonctionnalités, ils peuvent être soit des systèmes orientés traitement de données ou des systèmes orientés contrôle. Les fonctionnalités de traitement de données consistent en un calcul massif sur des données, alors que le contrôle consiste à séquencer ces calculs en choisissant lequel effectuer parmi différentes alternatives. Il est important de noter que le contrôle n'implique pas la notion d'état [Per05]. En effet, les choix effectués par le contrôle peuvent l'être sans connaissance des calculs antérieurs. Le cas échéant, c'est l'obligation de mémoriser des résultats de calculs précédents qui introduisent la notion d'état.

Mais on peut aussi classer les systèmes embarqués selon leurs usages en quatre types :

- L'informatique générale (General Computing) : Application similaire à une application de bureau mais empaquetée dans un système embarqué exemple : jeu vidéo, set- top box.

- Les systèmes de contrôle (Control Systems) : inclus les applications de contrôle du système temps réel. Exemple : Moteur d'automobile, processus chimique, processus nucléaire, système de navigation aérien.

- Traitement du signal (Signal Processing) : qui se caractérise par le calcul sur des quantités importantes de données. Exemple : Radar, Sonar, compression vidéo.

- Réseau et communication (Communication & Networking) : Ce type concerne les applications de transmission d'information et commutation. Exemple : Téléphone, Internet.

6. Conclusion :

Les systèmes embarqués sont des systèmes qui ont des contraintes spécifiques, et leurs prises en considération lors de la phase de développement sont nécessaires. Ces systèmes doivent être optimisés en termes de plusieurs critères. En raison de leur caractère critique, leur complexité ou encore leur inaccessibilité pendant leur utilisation, la conception de ces systèmes nécessite l'utilisation des méthodes, des modèles et d'outils bien spécifiques.

Spécification et modélisation des systèmes embarqués

Dans ce chapitre, on va présenter les principaux langages de spécification et de modélisation, utilisés pour les systèmes embarqués. On va mettre l'accent sur le modèle qu'on va exploitait lors de ce mémoire.

1. Introduction

La conception d'un système embarqué passe généralement par trois étapes : modélisation, spécification et implantation. Ce cycle de conception fait appel à différents domaines de compétences et divers métiers (automatique, informatique, temps réel,...). Ainsi, un système embarqué se trouve décrit, au cours de ses différentes étapes de conception, avec des langages différents et des formalismes variés, adaptés chacun à un métier donné.[S01]

La spécification est une description haut niveau des aspects logiciels et matériels du système sous une forme simplifiée sans en donner les détails. Dans un système embarqué temps réel, les aspects matériel et logiciel sont fortement liés. Pour réaliser le logiciel il est nécessaire de connaître non seulement les algorithmes applicatifs (par opposition aux algorithmes systèmes par exemple d'ordonnancement) ainsi que les contraintes temporelles sur l'exécution de ces algorithmes, mais il est aussi nécessaire de connaître l'architecture matérielle afin de générer du code exécutable adapté à l'architecture (jeu d'instructions des processeurs, accès capteurs et actionneurs, protocoles des médias de communication, etc.).

La spécification sert de point de départ à la conception du logiciel. Elle doit donc décrire à un haut niveau les algorithmes, l'architecture matérielle et les contraintes temporelles d'exécution des algorithmes sur le calculateur.[Koc00]

2. Propriétés des langages de spécification des systèmes embarqués

Parmi les caractéristiques les plus importantes que doit représenter un langage de spécification, on va citer quelques-uns :

- ◊ **La hiérarchie** : Les êtres humains ne sont pas généralement capables de comprendre les systèmes, qui contiennent de nombreux objets (états, composants) ayant complexes relations les uns avec les autres. La hiérarchie est le seul mécanisme qui aide à résoudre ce dilemme.

- ◊ **Temporisation** : les exigences temporelles sont parmi les caractéristiques explicites des systèmes embarqués, ils doivent être définis dans le cahier des charges.

- ◊ **Le comportement orienté état :** les automates peuvent fournir un mécanisme efficace pour la modélisation de systèmes réactifs. Par conséquent, le comportement des automates doit être facile à décrire. Toutefois, les modèles classiques d'automates sont insuffisants, car elles ne peuvent pas modéliser le temps ainsi que la hiérarchie n'est pas supportée.

◊ **La portabilité et la flexibilité :** La spécification doit être indépendante de la plate-forme matérielle, afin qu'elles puissent être facilement utilisées pour une variété des plates-formes cibles. Elle doit être aussi flexible, de façon que la modification des caractéristiques du système, exige également des changements à la spécification.

◊ **Event-Handling :** En raison de la nature réactive des systèmes embarqués, des mécanismes pour décrire des événements doivent exister. Ces événements sont des événements extérieurs (causée par l'environnement) ou internes (causée par des composants du système).

◊ **Propriétés non fonctionnelles :** Les systèmes actuels exposent un certain nombre des aspects non fonctionnelles, telles que la tolérance aux pannes, la taille, l'extension, la durée de vie, la consommation d'énergie, le poids, la disponibilité, la convivialité, la compatibilité électromagnétique (EMC), etc. Il n'y a aucune spécification qui permet de satisfaire tous ces propriétés, et les définies de façon formelle.

◊ **Des modèles de calcul (MOC[1])** : Dont le but est de décrire les calculs, des modèles de calcul sont nécessaires. Ces modèles seront décrits dans ce chapitre.

Sur la liste des exigences, il est déjà évident qu'il n'y aura pas de langage formel capable de répondre à toutes ces exigences. Par conséquent, dans la pratique, il faut faire des compromis. Le choix du langage utilisé pour une conception dépendra du domaine d'application, et de l'environnement dans lequel la conception doit être effectuée. Par la suite, on va présenter une étude des langages qui peuvent être utilisés pour les modèles réels.

3. Modèles orientés contrôle

3.1. Les réseaux de Petri

Les réseaux de Petri qui ont été conçus par C.A. Petri en 1962 [Pet62] sont largement référencés pour la modélisation des systèmes embarqués [sad07] [Lee06] [Cor03]. Classiquement, un RDP est composé de quatre éléments de base : un ensemble de places, un ensemble de transitions, une fonction d'entrée qui associe les transitions aux places "application d'incidence avant" et une fonction de sortie qui est aussi une application des transitions aux places "

[1] Models of computation

application d'incidence après", en fait les RDP sont un outil de modélisation efficace[Ata94] [Pet81].

Notons également deux propriétés intrinsèques intéressantes des réseaux de Petri qui sont leur possibilité d'exprimer la concurrence, et leur nature asynchrone. La première caractéristique (le parallélisme), signé que les événements peuvent se produire d'une manière indépendante s'ils sont rendus actifs. La propriété d'asynchronisme signifié qu'il n'y a pas de mécanisme d'horloge inhérent pour activer les transitions.

Certaines communautés prétendent, cependant, que l'inconvénient des réseaux de Petri de base provient du manque de décomposition hiérarchique [Kao04], ainsi que la quasi-absence de commodités pour décrire l'aspect communication de données. [Kar04].

Quelques modèles fondés sur les réseaux de Petri ont permis d'obtenir une description compacte des parties traitements et données. Ces modèles, dits de haut niveau, attachent une partie des données aux jetons. Parmi ces modèles on cite les réseaux de Petri colorés [Gra02], les réseaux Prédicat Transitions et les réseaux à objet.

Parmi les extensions qui ont été apportées aux RDP classiques, on cite : les ETPN (Extended Timed Petri Nets : Réseaux de Petri Temporisés étendus) qui sont des RDP étendus avec des informations temporelles pour faciliter l'évaluation des performances [Pen94]. Les réseaux de Petri hiérarchiques (HPNs [Dit95] : Hierarchical Petri Nets) dont la principale motivation est la difficulté de spécifier et de comprendre les graphes de réseaux de Petri de systèmes complexes par des représentations planes. Les HPN héritent des propriétés les plus importantes des réseaux de Petri (y compris la concurrence et l'asynchronisme).

3.2. Les machines à états finis

Les entités de ce modèle représentent les états, et les connexions représentent les transitions entre états. L'exécution est une succession strictement ordonnée d'états et de transitions. Les machines à états finis (FSM) sont des modèles bien adaptés pour exprimer la logique de contrôle et pour construire des modèles de modes (systèmes avec des modes d'opération distincts, où le comportement est différent pour chaque mode : intéressant pour exprimer différents modes de reconfiguration dans un système reconfigurable dynamiquement).

Les modèles à base de FSM se prêtent bien à une analyse formelle et peuvent, de ce fait, être utilisés pour vérifier l'absence de comportements inattendus du système. L'une des faiblesses de ce modèle réside dans le fait que le nombre d'états peut devenir rapidement excessivement grand en fonction de la complexité des systèmes.

À fin de décrire les modules complexes de contrôle dans les systèmes embarqués, une extension du modèle FSM a été introduite, appelée fFSM(Flexible FSM) [Kim05], [Par05]. Cette extension supporte la concurrence, la hiérarchie et les évènements internes comme les state charts. Mais l'inconvénient de ce modèle supporté par certaines approches telles que Ptolemy est l'absence d'une sémantique formelle causant certain problème comme la confidence de simulation, l'exactitude de la génération du code, et la validité de la spécification du système.

3.3. Modèles réactifs synchrones

Dans le modèle RS, les entités représentent des relations entre les valeurs en entrées et celles qui sont en sorties à chaque réaction du système. Ce modèle est caractérisé par leur syntaxe simple à utiliser et leur sémantique succincte et formelle.

Les modèles SR sont adaptés aux applications avec des logiques de contrôle concurrentes et complexes. Du fait que le synchronisme fort du modèle permet de travailler séparément sur les aspects de correction fonctionnelle et temporelle des systèmes, les applications temps réel à sécurité critique constituent de bons candidats [Lee98]. Cependant, à cause de cette propriété du synchronisme fort, certaines applications sont sur spécifiées avec ce modèle, ce qui limite les alternatives d'implantation et rend les systèmes distribués difficiles à modéliser. Un autre exemple intéressant d'application pour lequel le modèle réactif synchrone convient idéalement est celui de la gestion du protocole d'accès au média à jeton "token-ring" [Ste01].

De nombreux langages synchrones basés sur ce modèle ont été développés, nous distinguons dans la littérature deux grandes familles : la famille des langages synchrones flot de données tels que Signal et Lustre, et celle des langages synchrones flot de contrôle tels que Esterel et Argos.

3.4. Graphe de tâches

Les nœuds dans le graphe de tâche représentent les processus qui exécutent les opérations. Ces processus transforment les flots de données en entrée en flot de données sortant. Le processus typique contient un nombre infini d'itérations. Les arcs représentent les relations entre les processus.

Pour tenir compte des évolutions des systèmes embarqués qui intègrent généralement plusieurs processeurs et au moins un système d'exploitation, différentes approches considèrent un modèle de type graphe de tâches avec un ordonnancement statique en ligne préemptif, par exemple [Nat01] [Val03] pour les systèmes embarqués, [Yen95] [Kir97] [Dav97] [Oh99] pour les Soc[1] .Les tâches sont généralement de forte granularité et traitées comme des boites noires avec

[1]System On chip

des conditions de précédence. C'est le système d'exploitation qui gère l'exécution des tâches en utilisant ses propres structures de contrôle.

Le graphe de tâches est un modèle très utilisé pour la prédiction de performance et l'optimisation d'applications parallèles [Luo00] [Wud03] [Kua07]. Il présente cependant deux désavantages. Sa taille dépend de la valeur des paramètres de l'application qu'il modélise. La durée du calcul de l'ordonnancement ainsi que le coût mémoire dépendent de la taille du graphe et donc de la valeur des paramètres de l'application.

4. Modèles orientés traitement

4.1. Modèles à base de langages impératifs

4.1.1. Langage C

Les langages impératifs ne permettent pas de révéler un parallélisme potentiel aisément exploitable par les méthodes de conception du fait de leur nature séquentielle. Il existe des compilateurs permettant d'extraire le parallélisme au niveau instruction (ILP[1]), donc ayant une granularité très fine. Ces compilateurs ne sont cependant pas capables d'exploiter le parallélisme à gros grain qui est celui considéré par le concepteur dans une phase initiale de partitionnement ou de conception système. Le langage C est le langage le plus utilisé dans la pratique pour exprimer les comportements des systèmes embarqués.

Un langage de type C (qui peut être étendu pour représenter par exemple des processus concurrents) est ainsi utilisé dans les approches [Vah97] [Tei97] [Cor98] [Cal00]. D'autres approches comme [Kie00] et [Cos00] utilisent l'environnement Simulink de Matlab comme langage de spécification.

4.1.2. Java

Java est un autre langage impératif qui gagne du terrain avec C++ du fait de la réutilisation potentielle apportée par le concept d'objet. Java est devenu un des langages principales utilisé pour l'embarqué et les plates-formes mobiles [Tak01] [Ber05] [Rus07] [Ise08] [Suno6] en raison de son architecture de conception neutre, de sa portabilité et sa sécurité. Mais leur exécution dans les systèmes embarqués englobe spécialement les machines virtuelles Java (JVM) spécifié pour ce type de système, avec une configuration pour les environnements limités en mémoire.

[1] ILP : Instruction Level Parallelism

4.2. Processus communicants concurrents / séquentiels

D'un point de vue théorique, ce modèle a été formalisé sous forme d'algèbres de processus [Bae90]. Deux approches algébriques ont été particulièrement développées : le modèle CSP (Communicating Sequential Processes) de Hoare [Hoa83] et le modèle CCS (Calculus of Communicating Systems) de Milner [Mil82]. Les termes de l'algèbre permettent de décrire le comportement des processus. Les échanges de messages sont des actions élémentaires simultanées synchrones (on parle de rendez-vous) entre deux processus ou plus. Cette approche algébrique modélise un calcul réparti de façon suffisamment formelle, pour valider certaines propriétés de sûreté telles que l'interblocage.

Cependant, la complexité du raisonnement croît très rapidement avec le nombre des interactions par messages et rend difficile la validation de telles applications. Paradoxalement, bien que le formalisme de description algébrique soit très abstrait, deux obstacles se combinent et réduisent les possibilités offertes par cette approche : d'une part, les concepts modélisés sont eux très élémentaires et d'autre part, la communication est considérée comme synchrone et idéalement atomique, masquant ainsi bien des difficultés rencontrées dans les systèmes réels.

Les modèles basés sur ces processus permettent de décrire le parallélisme d'une application tout en étant indépendant d'une implémentation logicielle ou matérielle. Ces modèles sont bien adaptés à la description de systèmes de télécommunication. Dans [Jun09] propose un modèle CSP#, qui combine une modélisation haut niveau des opérateurs avec une modélisation bas niveau de code procédurales, pour permettre une vérification efficace du système.

4.3. Modèles à base de graphes de flots :

Dans ces modèles, ce sont les données qui fixent l'exécution.

4.3.1. Graphe de flot de données :

Les graphes de flot de données (DFG : Data Flow Graph), sont utilisés pour représenter les dépendances de données d'une application. Les DFG sont des graphes orientés acycliques dans lesquels les nœuds représentent les opérations et les variables, et les arcs représentent les dépendances de données entre opérations et variables (Figure 2.1). Au cours du temps, un nœud du graphe ne peut être exécuté que lorsque toutes ses données d'entrée sont présentes.

Chaque opération, à chacune de ses exécutions, consomme une donnée sur chacun de ses arcs d'entrée et combine ses entrées de manière à produire une donnée sur chacun de ses arcs de sortie. Cela induit une relation de dépendance d'exécution entre les différents nœuds du graphe. Cette relation implique qu'un nœud ne peut être exécuté qu'après complétion de l'ensemble de ses prédécesseurs (nœuds reliés à ses arcs présents en entrée), ce qui introduit un ordre partiel

d'exécution des opérations. Dans le cas d'opérations indépendantes (opérations sans dépendances), il n'existe pas de relation d'ordre d'exécution : ces dernières peuvent donc être exécutées dans n'importe quel ordre.

Les graphes de type DFG, permettent de mettre en évidence, par analyse des dépendances de données, les opérations pouvant être exécutées en parallèle.

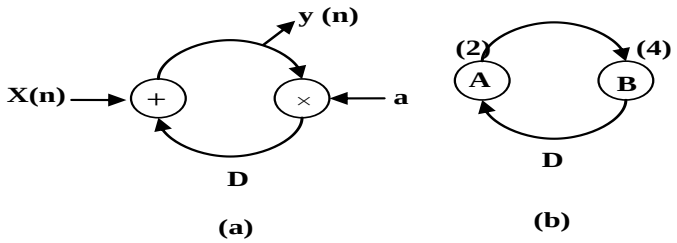


Figure 2.1 : y(n)= ay (n-1) +x(n) présenté (a) graphiquement et (b) DFG

Une des principales limitations des DFG réside dans leur incapacité à exprimer des comportements non déterministes, où le nombre d'itérations d'une boucle pourrait par exemple varier en fonction des données fournies au composant.

4.3.2. Graphes de flots mixtes de données et de contrôles

Les graphes de flots de données et de contrôles (CDFG[1]) [Mcf90] sont très appropriés pour décrire les manipulations de données, les communications ainsi que de simples contrôles locaux [Noo07]. Le contrôle est enfoui dans le flot de données au moyen de nœuds de branchement et de fusion et souvent au moyen de structures de boucles spéciales. L'expression du flot de contrôle est souvent limitée et n'est pas en mesure d'exprimer avec aisance les synchronisations (hors synchronisation par échange de données), ou les interruptions.

On parle aussi de graphes de flots de données et de contrôle hiérarchiques HCDFG (Hierarchical Control Data Flow) [Dig00] [Lem02], où le niveau de la hiérarchie le plus bas est le DFG avec des nœuds élémentaires de traitement et des liens de communication. À un niveau plus haut on retrouve les CDFG avec des nœuds test (if, case, loop...) et des appels à des DFG. Au niveau le plus haut, le HCDFG peut contenir des nœuds tests, d'autres HCDFG et des CDFG.

4.3.3. Flots de données synchrones (SDF)

---

1 Control Data Flow Graph

Les SDFGs (Synchronous Data Flow Graph) constituent un outil utile pour la modélisation et l'analyse des applications embarquées flot de données, tant dans le contexte d'un seul et multi processeur [Kan04], ou pour les applications concurrente temps réel [Gha06] . Le modèle SDF est un sous ensemble restreint des réseaux de processus de Kahn, où l'ordonnancement des exécutions, séquentielles ou parallèles, est calculé statiquement [Edw87]. Cette propriété fait de SDF un formalisme de spécification extrêmement utile pour le matériel [Lom04] [Pat05] et pour le logiciel temps réel embarqué.

Dans une implantation physique d'un SDF, les arcs représentent des tampons en mémoire. Un SDF doit en général s'exécuter de façon répétitive sur des données qui arrivent de façon continue dans les tampons. C'est le cas lorsqu'il modélise des applications de traitement du signal [Lee 87] [Mur02] [Hol06] [Gro07]. Alors, il faut pouvoir obtenir un ordonnancement des tâches du SDF qui puisse être exécuté dans une boucle infinie, tout en utilisant des tampons d'entrée et sortie de taille finie.

Si les SDF sont exécutés de manière répétitive. Les tampons mémoire représentés par les arcs peuvent être alloués une unique fois au moment de la compilation (ce qui n'est par toujours le cas pour les flots de donnée standard).

4.4.Modèles hybrides

Une approche de modélisation unifiée cherche à définir un modèle de calcul concurrent qui servirait à modéliser toutes les composantes d'un système. Ceci pourrait être possible en combinant tous les modèles précédents, mais une telle combinaison serait extrêmement complexe, difficile à définir et à utiliser (notons par exemple l'expérience *VCC* [1]de Cadence [Gar04]), et les outils de synthèse et de simulation seraient aussi difficiles à concevoir.

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 :

4.5.Les StateCharts et les CFSM

StateCharts est un type de modèle UML utilisé pour saisir les transformations du système à travers le temps. Il a été introduit en 1987 par David Harel [Har87], ce langage est utilisé traditionnellement dans le domaine de la conception de systèmes embarqués [Bus98] [Sch00] [Gri01] [Dal99] [Dar02] [Mue09], et permet de décrire la communication des machines à état finis. Il est basé, en termes de communication sur le concept de la mémoire partagée. Les nœuds

[1] Virtual Component Co-design

représentent les états, les arcs représentent les transitions d'état, les évènements sont représentés par des étiquettes sur les arcs.

En règle générale, les états transitions sont associés à des classes particulières (souvent, un ou plusieurs diagrammes d'état transition peuvent être dressés en vue de décrire complètement les états potentiels d'une classe).CFSM[1] sont développé à fin de modéliser les protocoles de communications. Chaque processus est représenté par un FSM et connecté entre eux par des listes de type FIFO.

4.6.Le modèle SDL[2]

L'utilisation de la mémoire partagée et le mécanisme de broadcast, les StateCharts ne peuvent pas être utilisés pour les applications distribuées. SDL est conçu pour les applications distribuées et se base sur le passage des messages asynchrones.

Le modèle SDL formel fournit deux types de format, le format textuel et le format graphique de processus qui sont les éléments de base de ce modèle. En général, la description SDL est un ensemble d'interaction de processus, ou FSMs. Les processus peuvent échanger des signaux. La communication se fait à travers des listes de type FIFO associées à chaque processus.

En effet, ce modèle est excellent pour la modélisation des systèmes de télécommunication [Dro01], et les systèmes embarqués [Kol98] [Mar02] [Alk02] [Die08], bien qu'il ne contienne pas un support de programmation complet, et aucune description pour les propriétés non fonctionnelles.

4.7.Le modèle PSM

Le modèle PSM[3], est une instance du modèle hétérogène, qui intègre un HCFSM (hierarchical concurrent finite-state machine) avec le paradigme du langage de programmation. Ce modèle se consiste essentiellement d'une hiérarchie des états de programmes, dont chaque état du programme représente une mode distincte de calcul. Dans n'importe quel moment, uniquement un sous-ensemble des états de programmes peut être actif, i.e., exécutant leurs calculs.

4.8.Modèles au niveau transactionnel

4.8.1. SystemC

---

1 Co-design Finite State Machine

2 Specification and Description Language

3 Program State Machine

SystemC permet la spécification des structures matérielles dans les langages de spécification du logiciel. La spécification peut être écrite en C ou C++, en faisant une référence appropriée aux bibliothèques des classes. SystemC permet la modélisation du temps, et des types de données pour tout le matériel commun. En général, Le comportement déterminé (Deterministic behavior) n'est pas garanti sauf qu'il soit utilisé par certain style de modélisation.

4.8.2. Verilog et SystemVerilog

Verilog[1] est un langage de description du matériel. Les processus sont utilisés pour modéliser la concurrence des composants matériels. Parmi ces points forts, ses interfaces avec C, C++, et les modèles SystemC, les facilités de simulation ainsi la validation de la vérification formelle.
SystemVerilog est un nouveau langage, considéré comme une extension du langage de description matériel Verilog afin de supporter de plus hauts niveaux d'abstraction, pour la modélisation et la vérification [Sys03].

4.9. Modèles orientés objet

En raison du nombre croissant de logiciels embarqués, UML a gagné une place importante dans la modélisation et la conception des systèmes embarqués. Plusieurs extensions ont été introduites pour introduire la notion du temps**.** Ces extensions ont été prises en compte lors de la conception d'UML 2.0 qui inclut 13 types de diagramme (9 diagrammes UML 1.4)**.**

Comme conséquence, une spécification exécutable peut être obtenue, seulement si UML est combiné avec un autre langage exécutable. Les outils de conception sont combinés avec SDL et C++.

5. Divers autres modèles

L'approche développée dans [Mou04] propose un formalisme pour décrire le concept d'asynchronisme dans le modèle RS Signal, ce qui permet de considérer le système comme étant globalement asynchrone.

Dans [Sil04], les auteurs utilisent un modèle basé sur les machines d'états finis synchrones pour modéliser l'architecture de communication d'un SoC (System on chip). Ce modèle se base sur le formalisme dit d'automates à protocoles synchrones.

Dans [Lee02], le modèle considéré est basé sur les machines à états finis hiérarchiques (HFSM) couplées à un modèle flot de données synchrone (SDF), pour modéliser un SoC reconfigurable et gérer l'ordonnancement de ses reconfigurations dans le temps.

---

[1] Very High Speed Integrated Circuit Hardware Description Language.

6. Choix d'un modèle de spécification

Le choix judicieux du modèle adopté pour la spécification d'un système est d'une importance primordiale, car il peut avoir une influence directe sur toutes les autres étapes de conception. Ce choix est souvent guidé par l'adéquation des propriétés et caractéristiques du modèle avec les spécificités du système à concevoir.

Dès lors, l'évaluation des différents modèles doit donc être établie en fonction des exigences des systèmes mixtes matériel/logiciel en termes de déterminisme et de sûreté de fonctionnement, de distribution et de parallélisme, ainsi que de la rapidité de la réaction (complexité et synchronisme). Ces notions, qui constituent quelques-unes des préoccupations parmi les plus importantes des concepteurs d'applications temps réel, sont précisément des concepts fondamentaux qui forment le socle sémantique des modèles de spécification.

Comme on peut le constater certains modèles sont plus appropriés à la modélisation des systèmes qui doivent manipuler de grandes quantités de données, d'autres modèles sont plus appropriés pour les systèmes comportant une forte dominante de contrôle. Ces modèles représentent donc plus exactement soit les données, soit le contrôle mais prennent rarement en compte les deux aspects : les graphes flots de données (DFG), par exemple, sont bien adaptés à décrire les dépendances de données dans un système de traitement du signal, mais ne sont pas aussi si bien adaptés pour la modélisation de la logique de contrôle associée et la gestion des ressources. Les automates à états finis (FSM) sont très bien adaptés pour la modélisation d'une logique de contrôle plus au moins simple, mais sont moins bien adaptés à modéliser les dépendances de données et le calcul numérique. Les processus communicants (CSP) sont adaptés pour la gestion de ressource, mais ils sur spécifient les dépendances de données. Ceci dit, il est rare qu'un système réel ait un traitement exclusivement orienté données ou exclusivement orienté contrôle, ce qui explique la continuité des travaux de recherche visant à proposer de nouvelles solutions pour une spécification complète des systèmes.

On constate aussi que parmi les principaux modèles de spécification cités, nombreux sont ceux qui ne prennent pas en compte la hiérarchie (FSM, PN,...). Cette insuffisance les rend inadéquats, dès que la taille et la complexité des systèmes à modéliser augmentent. Quelques modèles ne permettent pas de prendre en compte la concurrence (FSM, ...) et introduisent de ce fait des contraintes qui empêchent le système spécifié de profiter pleinement des ressources disponibles dans l'architecture cible (les processeurs logiciels et les circuits matériels) qui doivent fonctionner en parallèle.

Signalons aussi que la majorité des modèles sont basés sur une sémantique mathématique forte, offrant ainsi des possibilités de vérification formelle et d'optimisations poussées, d'ailleurs

c'est une propriété fortement exigée pour un modèle de spécification. On note également une prédominance de modèles asynchrones dans le sens où les transitions ne sont pas pilotées par un signal d'horloge.

En résumé, chacun des modèles présente des points forts pour la modélisation de certains concepts, mais aussi des points faibles pour la modélisation d'autres concepts, ce qui signifie qu'aucun modèle existant ne présente la totalité des capacités nécessaires à la modélisation de tous les types de systèmes à tous les niveaux d'abstraction. Ils offrent tous seulement des solutions partielles pour la spécification des systèmes actuels.

Actuellement, un modèle de spécification est essentiellement sélectionné en fonction des caractéristiques de l'application à implanter : pour un système qui répète périodiquement les mêmes transformations/opérations sur des paquets de données, comme un système de traitement du signal et des images, le modèle flot de données semble être le plus approprié. Par contre, pour un système qui n'exécute pas de calculs complexes, mais qui doit répondre à des séquences complexes d'événements externes, comme un système de contrôle commande le modèle FSM est approprié. Pour les systèmes qui exécutent des transformations complexes de données, plusieurs fois en parallèle, comme les bases de données client serveur ou les systèmes multi-tâches, le modèle CSP semble être le plus approprié.

Après avoir choisi le modèle approprié, il faut spécifier la fonctionnalité du système par l'intermédiaire d'un langage de spécification dont la sémantique est définie par le modèle choisi. VHDL et Verilog sont des standards très utilisés, permettant de décrire les modèles CSP et FSM. Esterel est également utilisé pour décrire les modèles CSP et FSM. StateCharts permet la description de FSM hiérarchiques et concurrentes. SpecCharts décrit les modèles CSP, FSM hiérarchiques et concurrents. SDL permet de décrire des GFD hiérarchiques contenant des FSM, Silage [Hil93] décrit des GFD.

7. Motivation du choix d'un modèle flot de données

En conclusion de cette étude de l'existant nous pouvons affirmer que malgré les nombreux modèles existants aucun n'a réussi à faire l'unanimité, et encore moins à s'imposer comme standard. Jusqu'à présent, aucun consensus ne semble se dégager sur un modèle de spécification universel (idéal) qui soit utilisable pour toutes les classes d'applications. Pour le moment, la sélection d'un modèle approprié est souvent guidée par les caractéristiques de l'application à implanter. Néanmoins, il est généralement admis qu'un modèle qui se voudrait efficace pour modéliser les applications dans un processus de conception conjointe, doit présenter un certain nombre de caractéristiques. Il doit, en particulier :

- Permettre de représenter aussi bien les flots de données que les flots de contrôle ;
- Préserver la cohérence des différentes parties du système à travers toutes les étapes du processus (continuité du modèle) ;
- Reposer sur un formalisme mathématique rigoureux, afin de permettre de recourir à la preuve formelle comme mécanisme de validation/vérification ;
- Permettre de modéliser au maximum la concurrence (parallélisme) entre calculs ;
- Prendre en compte la communication et la synchronisation ;
- Permettre l'implantation facile en matériel ou en logiciel ;
- Représenter la hiérarchie des systèmes complexes afin d'augmenter la lisibilité lorsque les systèmes modélisés sont de grande taille.

Le modèle flot de données permettant de décrire la concurrence, il s'adapte bien à décrire le parallélisme de l'application nécessaire à son implantation distribuée. Étant basé sur un modèle de graphe il permet naturellement de prendre en compte la hiérarchie : chaque opération du graphe peut être à son tour décrite par un sous graphe permettant une spécification hiérarchique de l'algorithme jusqu'aux "opérations atomiques" que l'on ne peut spécifier à l'aide d'un sous graphe. Cette "hiérarchie de graphes" offre des niveaux d'abstraction de la spécification permettant une souplesse dans la maîtrise de la complexité croissante des applications temps réels embarqués. De plus, ce modèle repose sur une théorie mathématique bien fondée "la théorie des graphes et des ordres partiels", ce qui permet de démontrer formellement les propriétés de la spécification et de les maintenir durant tout le cycle de conception à travers des transformations de graphes.

Outils de spécification de systèmes embarqués

Dans ce chapitre, on va présenter quelques outils de conception des systèmes embarqués les plus proches de notre objectif, ceux permettant la modélisation et l'optimisation des applications embarquées orientés traitement de données tels que les applications de traitement de signal et traitement d'images.

1. Introduction

La complexité des applications temps réel embarquées nécessite à la fois des outils de spécification de haut niveau et des méthodologies spécifiques de conception. Afin de réduire le nombre d'erreurs de spécification des algorithmes et de limiter au maximum les tests matériels, de nouvelles méthodes sont proposées. Ces méthodes permettent à l'utilisateur de se concentrer sur les aspects temporels qui sont cruciaux dans le domaine du temps réel (réactivité du programme et temps de réponse contraint), d'étudier les relations entre le parallélisme potentiel au niveau de l'algorithme et celui disponible au niveau de l'architecture, et être déchargé de la programmation de bas niveau (exécutifs) souvent fastidieuse.

La définition d'une méthodologie de conception effective selon l'article de Keutzer et Al, est une méthodologie qui commence d'un niveau d'abstraction élevé [Keu00] et permet la séparation des aspects de conception, pour permettre une exploration effective des solutions alternatives.

Plusieurs outils de conception sont apparus pour faciliter la spécification, la conception, et la génération de code de bas niveau. Ces outils de conception ont des objectifs et implémentent des approches différentes. Dans ce qui suit nous présentons quelques-uns. Notre choix a porté principalement sur des outils qui permettent une spécification haut niveau par des flots de données.

2. Gedae

Outil développé dans le cadre du projet RASSP (Rapid Prototyping of Application Specific Signal Processor) [Jam97], GEDAE (Graphical Entry, Distributed Applications Environment [Loc98]) est un logiciel commercial graphique, d'aide au développement d'applications de traitement du signal (systèmes temps réel) sur architectures multiprocesseur embarquées. Dans le domaine du traitement du Signal (TS), GEDAE est considéré comme l'outil le plus complet. En effet GEDAE, couvre tout ce domaine sans restriction apparente et permet de :

- Spécifier hiérarchiquement les applications à partir de bibliothèques de fonctions usuelles, développer des fonctions utilisateur, placer les traitements sur des architectures parallèles,
- Décrire les transferts de données en tenant compte du parallélisme des traitements, générer automatiquement du code C,
- Simuler l'exécution sur station de travail ainsi que le contrôle temps réel de l'exécution (visualiser la place mémoire occupée par chaque traitement et visualiser le temps passé à exécuter chaque traitement).

2.1. Algorithme

GEDAE repose sur un formalisme de graphe flot de données, conditionnel, acyclique, et fournit un environnement hiérarchique pour spécifier les algorithmes. On décompose l'application en boîtes fonctionnelles réutilisables. Ces boîtes donnent une structure modulaire aux graphes flot de données. Elles peuvent comprendre d'autres boîtes hiérarchiques, des boîtes primitives, des entrées, des sorties, des paramètres et des équations mais l'imbrication ne peut être récursive. Les boîtes primitives sont atomiques du point de vue du placement, c'est-à-dire qu'on ne peut les distribuer que sur un unique processeur.

2.2. Architecture

L'architecture est décrite par un unique fichier de configuration, elle contient 4 sections :

- La section processeur, la section communication
- La section mémoire, la section système

Pour insérer les paramètres d'initialisation. GEDAE ne fournit pas d'outils automatiques de distribution et d'ordonnancement de l'algorithme sur l'architecture. Le partitionnement et la distribution sont donc donnés graphiquement par l'utilisateur.

Les applications GEDAE peuvent être distribuées sur un réseau de stations de travail, sur un système de processeurs embarqués ou sur une station de travail multiprocesseur.

2.3.Génération de code

GEDAE génère un exécutif codé en C ANSI pour chacun des processeurs embarqués à partir du partitionnement et de la distribution utilisateur. Un noyau temps réel GEDAE porté sur la machine cible et résident sur chacun des processeurs embarqués exécute l'application auto codée.

En développant GEDAE, ses concepteurs ont cherché à rendre ses étapes principales de fonctionnement aptes à être réitérées (ces étapes sont visibles sur la figure 3.1). Cette itérativité évite le développement en cascade (Waterfall Model) [Mor01] et toutes les inefficacités qu'il engendre. Une des raisons qui permettent l'itération est la dissociation entre le développement et l'implémentation de l'application. Par ailleurs l'ensemble des architectures cibles pour cette implantation n'est pas extensible par l'utilisateur. De plus l'outil n'effectue aucune optimisation automatique, ni au niveau de la distribution et de l'ordonnancement, ni au niveau de l'optimisation de la mémoire.

2.4. Conclusion

GEDAE c’est un des outils les plus complets et les plus aboutis dans le domaine de traitement du signal. Cependant, si son aspect graphique permet de construire aisément des algorithmes, de

les implanter et d'étudier leur évolution, l'ensemble des architectures cibles pour cette implantation n'est pas extensible par l'utilisateur. De plus l'outil n'effectue aucune optimisation automatique, ni au niveau de la spécification, ni au niveau de la distribution ordonnancement de calculs.

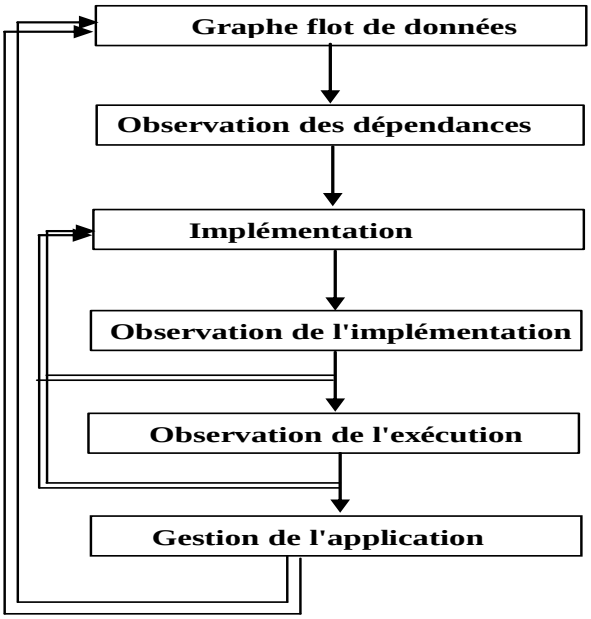


Figure 3.1 : Principales étapes de fonctionnement du logiciel GEDAE

3. PtolemyII

PtolemyII [Tsa00] [Dav99] est le successeur de Ptolemy 0.7 (appelé aussi "Ptolemy Classic" [Bha96]) dont il reprend de nombreuses caractéristiques. PtolemyII est un logiciel de recherche, non commercialisé, développé par l'université de Californie Berkeley, sous la direction de Edward A.Lee. PtolemyII vise la conception et la simulation, hétérogènes et concurrentes, de systèmes embarqués réactifs. Son but est de supporter la construction et l'interopérabilité de modèles exécutables selon des modèles d'exécution ("domaines") très variés. PtolemyII repose extensivement sur la technologie objet Java, aussi bien pour l'environnement de développement et son interface graphique "Diva", que pour le codage (et donc l'interopérabilité) des modèles de simulation, jusqu'à l'implantation sur un système réel (applets ou Java embarqué).

Cet outil vise le domaine des systèmes embarqués complexes, qui combinent plusieurs technologies, comme par exemple de l'électronique analogique et numérique, du matériel et du logiciel, des composants électroniques et mécaniques, et qui combinent également plusieurs types de traitements, comme le traitement du signal, les asservissements en boucle fermée, les automates de décision séquentielle, ou les interfaces utilisateur. L'accent est mis surtout sur la

simulation de ces systèmes, selon plusieurs niveaux de raffinement successifs ; le codage d'une implantation exécutable en temps réel par le système cible embarqué n'est vu que comme le dernier niveau de raffinement, qui nécessite, comme les autres, le support d'une machine virtuelle Java.

3.1. Algorithme

La variété des domaines d'application définit par PtolemyII nécessite une variété de formalismes de spécification, chacun bien adapté à un domaine spécifique. La décomposition d'un système complexe en sous-systèmes est soumise à des règles d'interaction entre sous-systèmes, qui définissent la sémantique du "modèle d'exécution" ou "domaine" du système. La décomposition récursive d'un système peut être hétérogène, c'est-à-dire que chaque décomposition peut avoir un modèle d'exécution différent. La décomposition récursive d'un système est décrite par un graphe hiérarchisé ("clustered hierarchical graph"), dont chaque sommet, appelé "entité", est représenté graphiquement par une boîte comportant des "ports" d'interface entre l'extérieur et l'intérieur de la boîte (qui peut contenir un sous graphe), et dont chaque arête, appelée "relation", est représentée graphiquement par une interconnexion en étoile entre ports.

PtolemyII définit les "domaines" suivants :

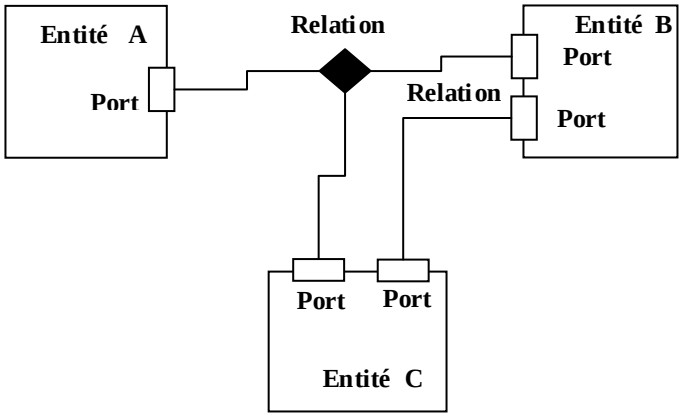


Figure 3.2 : Présentations graphiques des modèles dans PtolemyII

- **CT ("Continuous Time")** Chaque sommet représente un sous-système continu modélisé par un système d'équations différentielles, et chaque arc représente une variable continue.
- **PN ("Process Network")** Chaque sommet représente un processus séquentiel, et chaque arc représente un canal de communication monodirectionnel FIFO "asynchrone", c'est-à-dire les données sont reçues dans l'ordre où elles ont été émises, avec blocage du processus en réception si FIFO vide (modèle de Kahn).

- **DFM (”Design Flow Management”)** Chaque sommet représente un outil de conception, et chaque arc représente une interface entre outils (fichier, pipe, CORBA); DFM est une utilisation spécifique de PN pour automatiser le flot de conception.

- **CSP (”Communicating Sequential Processes”)** Restriction ”synchrone” du domaine PN, avec rendez-vous entre le processus émetteur et le processus récepteur d'une même communication : le premier prêt doit attendre l'autre (modèle de Hoare).
- **DE (”Discrete Event”)** Chaque sommet est un ”acteur”, qui reçoit et émet par ses arcs adjacents des ”jetons” (événements) datés lors de leur émission. En réaction à la réception d'un jeton, un acteur peut modifier son état interne et émettre à son tour d'autres jetons. Un ordonnanceur garantit un traitement chronologique des jetons.
- **SDF (”Synchronous DataFlow”)** Restriction déterministe du domaine DE, où chaque acteur doit recevoir un nombre prédéterminé invariable de jetons sur chacun de ses arcs d'entrée avant de réagir, et produit un nombre prédéterminé invariable de jetons sur chacun de ses arcs de sortie à chaque réaction.
- **OD (”Ordered Dataflow”)** Relaxation par rapport au domaine DE, de la notion de temps global : le temps est local à chaque acteur, qui doit en conséquence arbitrer l'ordre de traitement de jetons reçus portant la même date.
- **SC (”Star Charts”)** Domaine des machines à états finis (FSM) hiérarchiques.

Cette liste n'est pas exhaustive et peut s'enrichir dans le futur.

3.2. Architecture

L'architecture n'est pas modélisée par un graphe, elle est décrite par un module "target" (cible) dont le format n'est pas ouvert. Il existe différentes cibles : ordonnanceur monoprocesseur, multiprocesseur complètement connecté (fully connected), processeurs connectés par un bus (shared bus). Le modèle de l'architecture est donc difficilement formalisable et limité à quelques topologies particulières.

3.3. Adéquation

La distribution spatiale de l'algorithme n'est pas automatique, elle doit être spécifiée par l'utilisateur. PtolemyII n'automatise donc pas cette tâche. L'ordonnancement des calculs et des communications est effectué dynamiquement par l'ordonnanceur de chaque sous domaine.

3.4.Génération d'exécutif

PtolemyII crée et interconnecte des objets Java, en interaction avec l'utilisateur pour construire le graphe hiérarchisé représentant son application. Ces objets sont les acteurs directs de la simulation, leur ordre relatif d'exécution est déterminé par l'ordonnanceur de chaque sous domaine en fonction de leurs interconnexions. Le type et la sémantique des communications sont aussi fonction du domaine. Leur exécution est supportée par la machine virtuelle Java. Pour toute partie de l'application réelle qui ne serait pas supportée par une machine virtuelle Java, la traduction du code Java de simulation en code cible (assembleur, VHDL, plans mécaniques...), ainsi que la vérification de leur équivalence, sont à la charge de l'utilisateur.

3.5. Conclusion

PtolemyII s'intéresse surtout à la simulation d'applications complexes très hétérogènes, mais apparemment pas à l'optimisation ni à la génération automatiques de leur implantation.

4. ForSyDe

ForSyDe (Formal Design System) est une méthodologie de conception des systèmes embarqués hétérogènes [Jan04]. Le point de départ de l'élaboration de cette méthodologie a été la conviction des futures méthodologies de conception de systèmes qui doivent :

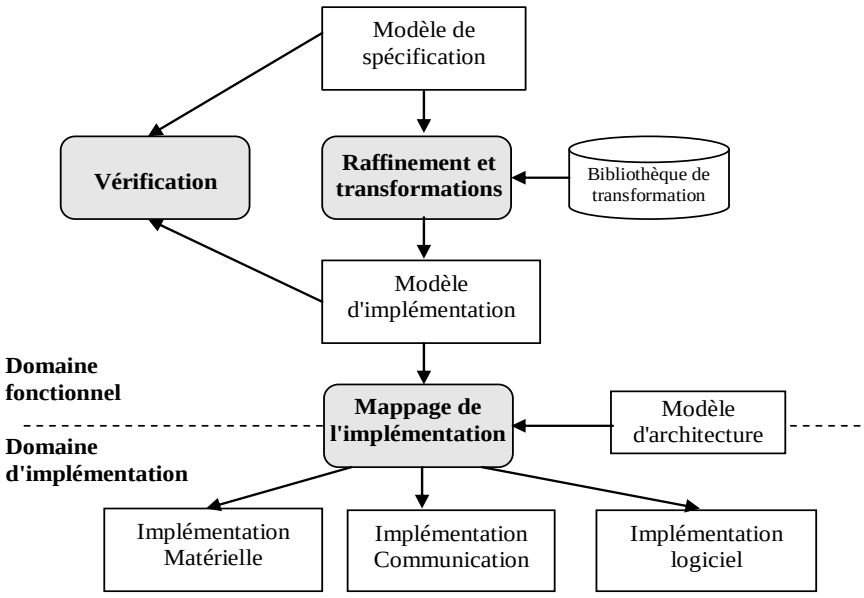


Figure 3.3 : Flot de conception de ForSyDe

- Commencer à partir d'un haut niveau d'abstraction afin d'être en mesure de faire face à l'extrême complexité des applications, ainsi celle de l'architecture ;

- Donner une base solide pour l'intégration future des méthodes formelles, puisque la simulation seule n'est pas suffisante pour vérifier les futurs systèmes à différents niveaux d'abstraction.

À partir d'un modèle de spécification formelle, qui capture les fonctionnalités du système à un niveau d'abstraction élevé, ForSyDe fournit des méthodes formelles de transformation qui permettent un raffinement transparent des processus du modèle système, en un modèle d'implémentation qui est optimisé pour la synthèse.

4.1. Algorithme

Le modèle de spécification est basé sur le modèle de calcul synchrone et modélise le système avec un réseau hiérarchique de processus connectés par des signaux. Les processus sont construits par des constructeurs de processus, qui permettent la séparation des communications et des opérations de calcul, et donnent une sémantique à l'implémentation.

ForSyDe supporte les modèles de calcul (MOC) suivants ;

- Les flots de données synchronisées (Synchronous Data Flow SDF).
- Les machines à état fini (FSM).

Cet outil permet aussi une combinaison séquentielle, parallèle, et retour arrière (feed-back) des processus.

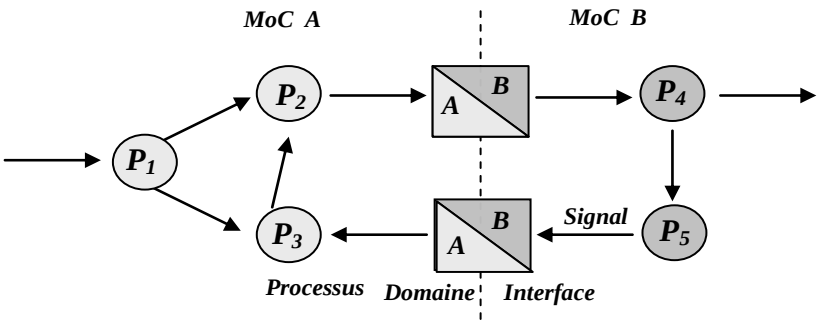


Figure 3.4 : Un modèle ForSyDe un réseau de processus concurrents. Les processus de différents modèles de calcul peuvent communiquer via un domaine d'interface.

4.2. Architecture

Chaque constructeur de processus (Process Constructor) a une structure matérielle et une sémantique logicielle, qui est utilisée pour la transformation du modèle d'implémentation en une implémentation matérielle / logicielle.

4.3.Génération de code

Le résultat du raffinement du modèle système par les différentes étapes de transformation, est un modèle d'implémentation, qui serve comme un point de départ pour la synthèse [Zlu02]. Après l'étape de partitionnement, un code VHDL pour les composants matériels sera généré ainsi un code C pour le logiciel.

4.4. Conclusion

L'idée principale de cet outil est de combler l'écart entre le modèle de spécification et celui de l'implémentation par les étapes de raffinement transformationnelles. Mais ForSyDe est limitée à un petit nombre de transformations, aussi un ensemble restreint de règles de mappage logiciel / matériel, qui sont insuffisants pour cibler la conception industrielle [San03].

5. GrapeII

GRAPE II [Lau95] (Graphical RApid Prototyping Environment), est un environnement de conception conjointe pour l'émulation temps réel de systèmes synchrones à base de DSP. GRAPE II permet la spécification, l'estimation de ressources, la resynchronisation (retiming), le partitionnement, la distribution, l'ordonnancement, le routage, la génération de code, la compilation, le débogage, la simulation et l'émulation des applications synchrones multi vitesse (multi rate) sur des architectures cible hétérogènes, constituées par de multiples processeurs DSP et des FPGAs.

5.1. Algorithme

Dans cet environnement, l'application est spécifiée d'une façon totalement indépendante de l'architecture cible qui va l'implanter : le concepteur peut utiliser son éditeur graphique hiérarchique pour dessiner le graphe de dépendances de données où il peut se servir de l'interface pour l'environnement DSP Station de Frontier Design, qui lui permet de spécifier l'application dans le langage flot de données DFL (dérivé du langage Silage [Hil93]).

5.2. Architecture

L'architecture est spécifiée de façon similaire, tout en utilisant l'éditeur graphique. On peut spécifier la fréquence d'horloge, le taux et le protocole de communication, la quantité de mémoire disponible, etc. Une fois l'architecture définie, GRAPE II estime la quantité de ressources nécessaires à l'exécution de chaque tâche de l'application. Pour des cibles microprocesseurs, il estime la latence, la taille du code et de la mémoire, et des ressources dédiées comme les

convertisseurs analogique numérique. Pour des cibles FPGA, il estime les ressources en termes du nombre de blocs logiques configurables (CLB Configurable Logic Block), d'entrée/sorties, etc.

À partir de cette estimation de ressources des tâches de calcul de l'application et des ressources disponibles décrite dans la spécification de l'architecture matérielle cible, GRAPEII effectue le partitionnement automatique, l'affectation, le routage et l'ordonnancement pour des cibles hétérogènes en utilisant des heuristiques de type "branch and bound".

5.3.Génération de code

Finalement, la dernière phase produit du code C ou VHDL pour chacun des composants de l'architecture mixte. GRAPE II génère automatiquement du code VHDL pour les outils de synthèse matérielle et du code C ou assembleur pour les compilateurs DSP. Notons que pour assurer une mise en correspondance efficace de l'algorithme de l'application sur l'architecture cible, GRAPEII effectue une exploration de l'espace de conception basée sur des transformations de déroulage ("folding", "unfolding", "retiming", "clustering", etc.) appliquées au graphe algorithmique de l'application.

5.4. Conclusion

Enfin, GRAPE II est un environnement ouvert. Le concepteur peut ainsi y ajouter ses propres outils. GRAPE II est commercialisé sous le nom Virtuoso Synchro par Intelligent Systems International.

6. SynDEx

Le logiciel SynDEx correspond à la mise en œuvre de la méthodologie AAA (adéquation algorithme architecture). Il permet la spécification d'algorithmes et d'architectures au moyen d'une interface graphique. Après d'éventuelles contraintes de placement (association spécifique d'une opération et d'un processeur), il permet le calcul d'un ordonnancement valide ainsi que sa durée d'exécution. Il offre ensuite la possibilité de générer un ensemble de programmes (un programme par processeur constituant l'architecture), l'exécution de ces programmes sur leurs processeurs associés correspondant à la mise en œuvre de l'algorithme spécifié.

6.1. Présentation

SynDEx est un logiciel de CAO (conception assistée par ordinateur) basé sur l'interactivité avec l'utilisateur grâce à une interface graphique qui permet de spécifier le graphe d'algorithme, le graphe d'architecture ainsi que les caractéristiques de ces graphes. SynDEx est ensuite capable de construire et d'afficher automatiquement le diagramme temporel d'implantation optimisé de

l'application grâce à l'heuristique qu'il renferme. Si les caractéristiques de ce graphe sont conformes avec les exigences temps réel de l'application, SynDEx est ensuite capable de générer automatiquement l'exécutif de chacun des processeurs de l'application ainsi que le "makefile" qui va automatiser les différentes phases de substitution, compilation, téléchargement et lancement de l'exécutif de l'application. La figure 3.5 indique ces différentes fonctionnalités offertes par SynDEx et les interactions utilisateur qu'il permet.

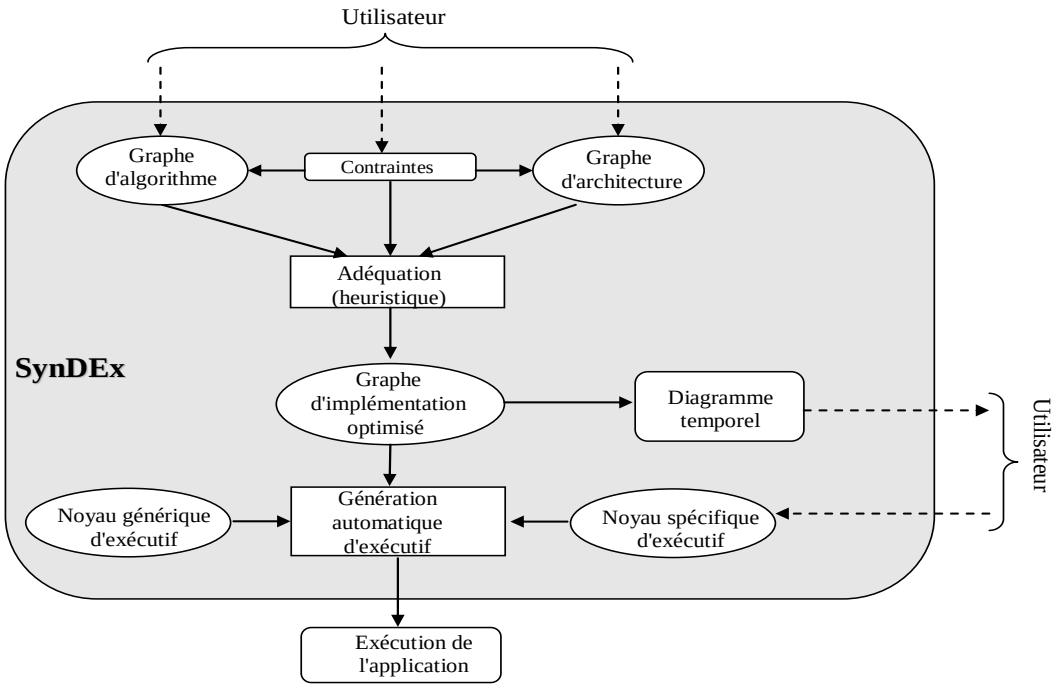


Figure 3.5 : Les fonctionnalités de SynDEx

6.2.Méthodologie AAA

La méthodologie "Adéquation Algorithme Architecture" (AAA) proposée par l'INRIA-OSTRE vise le prototypage rapide et l'implantation optimisée d'applications distribuées temps réel embarquées, telles celles rencontrées en contrôle commande de systèmes complexes comprenant du traitement du signal et des images. Cette méthodologie est fondée sur une approche globale formalisant l'algorithme et l'architecture à l'aide de graphes, et l'implantation de l'algorithme sur l'architecture à l'aide de transformation de ces deux graphes. Le graphe flot de données modélisant l'algorithme au niveau comportemental est transformé progressivement jusqu'à ce qu'il corresponde au graphe matériel modélisant l'architecture. Ces transformations représentent une distribution (allocation spatiale) et un ordonnancement (allocation temporelle) des calculs sur les processeurs et des communications sur les liaisons physiques inter-processeurs. Le résultat de ces

transformations est un exécutif temps réel distribué supportant l'implantation de l'algorithme spécifié sur l'architecture donnée.

L'adéquation revient ainsi à choisir parmi toutes les implantations possibles d'un algorithme sur une architecture, l'implantation dont les performances, déduites des caractéristiques des composants de l'architecture, respectent les contraintes temps réel, tout en minimisant la consommation de ressources, ce qui se traduit par un problème d'optimisation.

L'intérêt principal du modèle de graphe utilisé réside dans sa capacité à exprimer tout le parallélisme, non seulement dans le cas de l'algorithme (parallélisme potentiel que renferme l'algorithme, exprimé à travers un graphe d'algorithme : graphe flot de données) et de l'architecture (parallélisme disponible qu'offre effectivement l'architecture, exprimé à travers un graphe matériel : graphe de ressources de calcul, de ressources mémoire et de ressources de communication). Mais aussi dans le cas de l'implantation de l'algorithme sur l'architecture (réduction du parallélisme potentiel au parallélisme disponible exprimée par transformations de graphes, c'est-à-dire distribution et ordonnancement des calculs et des communications).

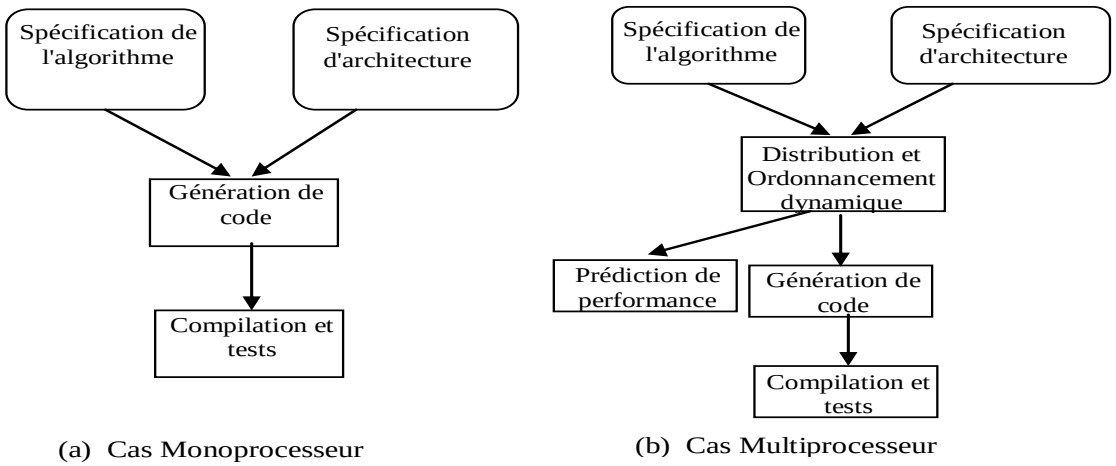


Figure 3.6 : Flot d'implantation AAA

Cette modélisation unifiée de l'algorithme, de l'architecture et de l'implantation à base de graphe offre un cadre formel bien adapté à la conception sans erreur et à la commande (pas de déformation ou perte d'information liée à une transcription du modèle) [Kao04]. Une telle unification et continuité du modèle permettent de vérifier facilement que toutes les transformations conservent les propriétés d'ordre partiel du graphe d'algorithme initial, que l'implantation finale correspond bien à la spécification initiale et que les optimisations effectuées seront conservées lors de l'exécution réelle. Ceci évite d'une part toute rupture entre la phase de

spécification et celle de réalisation et d'autre part la rupture entre le prototypage rapide de l'application et la réalisation industrielle de série.

6.3. Modèle d'algorithme

La représentation en graphe flot de données permet la description explicite des dépendances de données, et par conséquent la mise en évidence du parallélisme potentiel entre les opérations nécessaires à l'implantation parallèle de l'algorithme de l'application, alors que, dans le cas d'un graphe flot de contrôle, il faut extraire ces dépendances, implicitement décrites par le partage de variables entre opérations. Dans le cas du flot de données, les données sont dépendantes du contrôle et l'ordre d'accès aux données est ainsi directement imposé par l'ordre d'exécution des opérations. Ce type de spécification d'algorithme est moins sujet à des erreurs, ce sera au compilateur des langages qui supporte ce modèle flot de données d'assurer l'ordre d'écriture et de lecture dans les variables, qui pourront ainsi être réutilisées sans erreurs.

Les opérations atomiques que l'on peut définir sous SynDEx sont :

- Sensor : qui correspond aux entrées du graphe flot de données (capteur). Cette opération peut uniquement produire des données, elle ne contient que des ports de sortie ;
- Actuator : qui correspond aux sorties du graphe flot de données (actionneur). Cette opération peut uniquement consommer des données, elle ne contient que des ports d'entrée ;
- Constant : qui produit des données identiques lors de chaque exécution de l'algorithme. Il suffit donc de l'exécuter lors de la première itération du graphe d'algorithme ;
- Delay : qui permet de spécifier les dépendances inter itérations du graphe d'algorithme en les mémorisant d'une exécution à une autre. Il s'agit d'une dépendance de donnée entre chaque motif du graphe flot de données ;
- Function : qui est une opération atomique, quand elle contient uniquement des ports d'entrée et de sortie. Cependant cette opération peut elle-même contenir des opérations de calcul, de conditionnement et de répétition, elles-mêmes hiérarchiques.

Le graphe d'algorithme peut être issu d'une spécification effectuée dans les langages synchrones (Lustre, Esterel, Signal etc.) en important une spécification intermédiaire générée en DC [Hal95], le format commun des langages synchrones. SynDEx a aussi été interfacé avec d'autres outils parmi lesquels nous citerons AVS [Fre00], Scicos [Nio97] et Camlflow [Cha00].

En résumé, le modèle d'algorithme AAA est un graphe basé sur le modèle flot de données supportant la notion d'ordre partiel d'exécution. Il permet d'une part d'exprimer le parallélisme

potentiel pour pouvoir l'exploiter (à l'aide de transformations simples) lors de l'implantation distribuée, et d'autre part prend en compte l'interaction infiniment répétitive des applications cibles avec leurs environnements (systèmes réactifs) ainsi que l'aspect conditionnel des algorithmes. L'algorithme est donc spécifié par un hypergraphe orienté acyclique infiniment répété et conditionné [Lav97], appelé Graphe Factorisé et Conditionné de Dépendances de Données (GFCDD), dont les sommets sont des opérations partiellement ordonnées (parallélisme potentiel) par leurs dépendances de données. Ce modèle étendu introduisant le flot de contrôle à l'intérieur du flot de données en prenant en compte les spécificités des parties répétitives (boucles), ainsi que le cas de calculs exécutés conditionnellement (if..then..else).

6.4. Modèle d'architecture :

Le modèle d'architecture choisi dans AAA est hétérogène et multi-composants, car sa structure comprend en général des composants capteurs et actionneurs, des composants programmables (processeurs RISC, CISC, DSP, microcontrôleurs) et des composants non programmables (circuits intégrés spécifiques ASIC, circuits FPGA reconfigurables), interconnectés par des médias de communication. Ce modèle d'architecture est un graphe orienté, dont chaque sommet est une machine à états finis (machine séquentielle) et chaque arc une connexion physique entre deux machines à états finis. Dans AAA, il existe deux types de nœuds :

- Opérateur : Pour exécuter des opérations de calcul,
- Médium de communication : pour exécuter des opérations de communication (bus /mux /démux, mémoire).

La mémoire peut aussi être considérée comme une machine séquentielle. Dans AAA, il existe deux types de sommets mémoire :

- La mémoire RAM (à accès aléatoire) pour stocker les données ou programmes locaux à un opérateur,
- La SAM (à accès séquentiel).

L'hétérogénéité signifie non seulement que les sommets peuvent avoir chacun des caractéristiques différentes (durée d'exécution des opérations et taille mémoire des données communiqués par exemple), mais aussi que certaines opérations peuvent n'être exécutées que par certains opérateurs, ce qui permet de décrire tout autant des composants programmables (processeurs-FPGA) que des composants non programmables (ASIC).

6.5. Adéquation : Optimisation de l'implantation

La recherche d'une implantation optimisée d'un algorithme sur une architecture respectant les contraintes temps réel et minimisant la taille de l'architecture, correspond à une adéquation (" mise en correspondance efficace" entre cet algorithme et cette architecture). Quoique cette notion d'implantation optimisée est utilisée bien que l'on ne peut pas garantir l'obtention d'une solution optimale pour ce type de problème reconnu NP-difficile : en effet, pour un couple donné de graphes d'algorithme et d'architecture, il existe un nombre très vaste mais fini de graphes d'implantations possibles. Le nombre de cas à étudier étant très élevé, l'adéquation se contentera donc d'une solution approchée obtenue rapidement, plutôt que d'une solution exacte obtenue dans un temps rédhibitoire à l'échelle humaine à cause de la complexité combinatoire exponentielle de la recherche exhaustive de la solution exacte (optimale).

La méthodologie AAA utilise les heuristiques gloutonnes (c'est-à-dire sans remise en cause des solutions partielles intermédiaires conduisant à une solution finale ; elles sont aussi dites sans retour arrière) et plus particulièrement celles de liste car ce sont elles qui donnent le plus rapidement leur résultat tout en ayant une bonne précision [Lui93]. Celte heuristique a été formalisée et automatisée [Ker09] est basée sur un algorithme de liste glouton, associé à une fonction de coût qui prend en compte la durée d'exécution des opérations et des communications pour indiquer le degré d'urgence qu'il y a à distribuer et ordonnancer une opération sur un opérateur. Cet algorithme basé principalement sur les dates d'exécutions des calculs, requiert une phase préalable de caractérisation dans laquelle une durée d'exécution est associée à chaque opération de calcul ou respectivement de communication. Cette durée est fonction de l'opérateur ou respectivement du communicateur qui est capable de l'exécuter.

6.6. Génération de code

La génération automatique d'exécutif distribué se fait suivant des règles décrivant la transformation d'un graphe d'implantation optimisé en un graphe d'exécution. Pour chaque opérateur (resp. médium de communication) SynDEx construit un programme séquentiel formé de la séquence des opérations de calcul (resp. communications) qu'il doit exécuter. Les opérations de communication sont :

- des 'Send' et des 'Receive' de données transmises entre communicateurs via une SAM (communication par passage de messages),
- des 'Write' et des 'Read' quand les données sont transmises via des RAM (communication par mémoire partagée). Les principes majeurs de la génération d'exécutif de SynDEx sont très bien détaillés dans le document [Gran98].

6.7. Conclusion

La méthodologie AAA d'adéquation algorithme architecture, que nous avons présentée offre un cadre formel et un flot de conception unifié sans rupture. Permettant d'une part de faire des vérifications temporelles en termes d'ordre sur les événements qui entrent et sortent de l'algorithme, éliminant ainsi un grand nombre d'erreurs concernant la logique de l'algorithme, et d'autre part d'aider à la recherche, parmi toutes les implantations possibles que l'on peut faire d'un algorithme celle qui respecte les contraintes temps réel et minimise les ressources matérielles. Enfin elle permet de générer automatiquement des exécutifs taillés sur mesure pour chacun des processeurs de l'architecture, qui sont sans interblocage et dont le surcoût est très faible, en assurant que les vérifications faites précédemment restent valides.
Par rapport aux autres produits concurrents (CASH, GEDAE, Ptolemy II, TRAPPER), les atouts principaux de SynDEx, on peut citer les points suivants :

- Propose des modèles cohérents pour spécifier l'algorithme et l'architecture matérielle. Cette dernière est décrite de manière détaillée afin de prendre en compte les communications inter-processeurs qui sont critiques ;
- Offre la possibilité, avant la génération de code, de visualiser (simuler) un graphe temporel d'exécution des tâches ;
- Génère un code intermédiaire afin que l'utilisateur, suivant une méthodologie conseillée, puisse implémenter son algorithme sur n'importe quelle architecture.

Tout cela place AAA ainsi que le logiciel SynDEx qui la supporte, parmi les méthodes et outils logiciels d'aide à l'implantation les plus avancés dans le domaine de développement d'applications distribuées temps réel embarquées.

Resynchronisation

La méthode de resynchronisation est une méthode permettant de minimiser le temps d'exécution des applications de traitement du signal et de traitement d'image, modélisées par les graphes de flot de données. Dans ce qui suit on va présenter cette technique avec quelques exemples d'applications.

1. Les flots de données et la resynchronisation

Dans les applications traitement du signal DSP, traitement d'images et la simulation dynamique des fluides, les volumes de données traités sont très importants. La prise en compte du temps de calcul reste un élément majeur dans les algorithmes en dépit des progrès technologiques exponentiels des microprocesseurs, d'où il est important de minimiser le temps de cycle d'exécution. Les boucles consistent le chemin critique de ces applications : il est donc nécessaire d'exploiter le parallélisme imbriqué dans ces boucles afin de minimiser le temps de cycle. La description de ces applications par le modèle flot de données a été largement utilisée pour exposer le parallélisme potentiel tel que dans [Kar92, Par89, Sri97, Chi04, Cal98, Zho04]. Ce modèle est largement utilisé dans de nombreux domaines, y compris les circuits [Lei91] [Cal98] [Sov07], traitement des signaux numériques [Kun85] [Ziv96] [Den98] [Liu04] et la description des programmes [cha92] [Lam88].

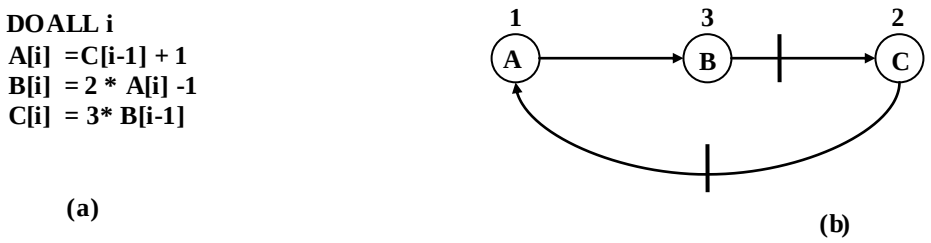


Figure 4.1 : (a) Exemple de programme ; (b) Le graphe de flot de données correspondant G_1.

La figure 4.1 présente un exemple d'un graphe de flot de données modélisant une boucle. Si on suppose que les additions prennent une unité de temps, tandis que les multiplications en demandent deux unités. Donc, le calcul de A[i] prend 1 unité de temps, celui de B[i] est 3, et celle de C[i] est de 2 unités. Chaque fois que la variable de la boucle est incrémentée, ces trois états s'exécutent une fois chacun. En d'autres termes, à chaque itération de la boucle, trois lignes du corps de la boucle, s'exécutent.

Notons que le calcul de *A* dépend de la valeur de *C* à l'itération précédente, et celle de *C* dépend de la valeur précédente de *B*. Ce sont deux exemples de dépendances d'inter-itérations, où une valeur étant calculée en cette itération nécessite des informations calculées dans une précédente. D'autre part, la valeur de *B* étant calculée dans la deuxième étape ne dépend que de la valeur de A calculée à l'étape précédente de la même itération. Ceci est un exemple d'une dépendance dans la même itération.

L'exemple du programme 4.1 (a) est modélisé par un DFG dans la figure 4.1 (b). Chaque état du corps de la boucle est représenté par un nœud, et un arc pour chaque dépendance entre les états. Le temps de calcul est représenté par un chiffre sur chaque nœud. Les bars qui coupent les

arcs, représentent les dépendances inter-itérations entre ces nœuds. Cette représentation d'une telle dépendance est appelée retards. Dans le cas de la DFG de la Figure 4.1 (b), il y a un seul retard sur les arcs (B, C) et (C, A).

2. La resynchronisation

La resynchronisation (retiming) est à l'origine une technique d'optimisation, connue dans le domaine de la synthèse de circuits (ou VLSI pour Very large scale Integration) sous le nom retiming. Cette technique a été introduite en 1991 par Leiserson et Saxe [Lei91] comme un outil théorique visant à minimiser la période d'horloge ainsi que le nombre de registres des circuits synchrones, modélisés sous forme de graphes dirigés pondérés [Hua01].

Cette méthode d'optimisation du graphe de flot de données, se base sur la distribution des retards sur les arcs de sorte que les fonctions de l'application restent les mêmes. Mais la longueur de plus long chemin avec zéro retard, appelée la période d'horloge noté *cl(G)* (qui représente aussi le temps de cycle), est diminuée. Cette technique a été introduite afin d'optimiser le débit de circuits synchrones [Dey92] [Ish93] [Lal95] [Pap94] [Soy94] [Sov07], et a été largement utilisée dans des domaines aussi variés que le pipeline logiciel [Pas94] [Xue06], le codesign [Cha98] [She96], la minimisation du coût de la mémoire [Liu09], et la minimisation d'espace [Zhu06].

La resynchronisation peut être utilisée aussi pour réduire la consommation énergétique, par la réduction des switchings, qui peuvent conduit à une dissipation dynamique de l'énergie dans les circuits CMOS statique. Le placement des registres dans les sommets d'entrée ayant une grande capacité peut réduire les activités de switching dans ces sommets, et qui peuvent conduire à des solutions qui consomment moins d'énergie [Mon93] [Cha04] [Fis05] [Ekp06].

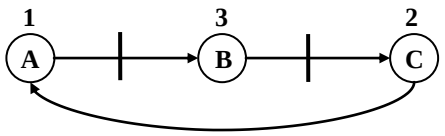


Figure 4.2 : G_1 resynchronisé avec cl (G_1r) =3

Dans la Figure 4.1(b) $cl(G_1) = 4$ (le temps de calcul du plus long chemin qui contient zéro retard). Un retard peut être supprimé de l'arc *(C,A)* est placé sur l'arc *(A,B)* à fin de créer une version resynchronisée de G_1, notée G_{1R} est illustrée dans la Figure 4.2, avec $cl(G_{1r}) = 3$. D'après l'algorithme polynomial de la resynchronisation qu'on va présenter dans la section suivante, la période d'horloge minimale que nous pouvons obtenir par la resynchronisation de G_1 est minimum trois unités de temps, tandis que dans la version original était de 4 unités de temps.

3. La minimisation du temps de cycle

3.1. Définition Formelle

Un graphe flot de données (DFG) formellement est un graphe *G* fini, dirigé, avec poids, noté $G = (V,E,d,t)$ où :

1. *V* est l'ensemble des sommets qui représente les nœuds de calcul ;
2. $E : \subseteq V \times V$ est l'ensemble des arcs, qui représente les relations de précédence entre les différents nœuds ;
3. $d : E \rightarrow Z$ est la fonction avec *d(e)* le nombre de retards sur l'arc *e* ;
4. $t : V \rightarrow Z$ est la fonction avec *t(v)*, le temps de calcul du nœud *v*.

Un chemin dans *G* est une séquence de nœuds et arcs connectés. On utilise la notation *T(p)* et *D(p)* pour présenter le temps de calcul total et le nombre de retards dans un chemin *p*, respectivement. Le chemin critique est le chemin le plus long qui contient zéro retard ; en d'autre terme : $cl(G) = \mathbf{max}\ \{T(p) : p \in G \text{ est un chemin avec } D(p) = 0\}$. Dans l'exemple de la Figure 4.1 (b) *cl du graphe* ***G*** *égal à 4.*

3.2. Algorithme de resynchronisation :

Une resynchronisation *r* d'un DFG *G* est une fonction de $V \rightarrow N \cup \{0\}$, qui spécifié une transformation de *G*. Elle marque chaque arc avec un nombre de retards retirés de ces arcs entrants et placés sur ses arcs sortants, en changeant le graphe G à un graphe resynchronisé $Gr = \{ V, E, d_r, t\}$, où $d_r(e) = d(e) + r(v) - r(u)$ pour chaque arc $e = (u,v)$.

La resynchronisation consiste à trouver un vecteur qui va minimiser la longueur d'un tel chemin. Par exemple, la fonction avec le vecteur (r(A) = 1, r(B) =0, r(C) = 0) resynchronise le graphe (b) de la figure 4.1 au graphe de la figure 4.2.

On trouve la période minimale *cl* obtenue par la technique de resynchronisation, en effectuant une recherche dichotomique sur toutes les périodes *cl* possibles. À chaque étape de la recherche, on tente de resynchroniser le graphe avec la valeur actuelle *cl* de la période. La plus petite période est retournée comme la meilleure période.

L'algorithme suivant présenté dans [Mah98] d'ordre $O(|V||E|)$ est utilisé, pour trouver une resynchronisation pour une période d'horloge donnée.

Algorithme FEAS : Avec un graphe $G = (V,E,d,t)$, et une période *c*, cet algorithme donne une resynchronisation *r* de *G*, dont $cl(G_r) \leq c$.

{

1. Pour chaque sommet $v \in V$, $r(v) \leftarrow 0$.

2. Répéter $|V|$ *-1* fois
   2.1. Calculer G_r *avec la valeur* actuelle de *r*.
   2.2. Exécuter l'algorithme CP sur le graphe G_r pour déterminer $\Delta(v)$ pour chaque sommet $v \in V$.
   2.3. Pour chaque v où $\Delta(v) > 0$, $r(v) \leftarrow r(v)+1$.
3. Exécuter l'algorithme CP sur le graphe G_r. *Si cl(G*r*) > c*, alors aucune resynchronisation faisable existe.

}

Algorithme CP : Cet algorithme calcule la période *cl(G)* pour un graphe *G = (V,E,d,t),*

{

1. Soit G_0 un sous graphe de *G* qui contient précisément les arcs *e* avec un poids *d(e) = 0.*
2. Effectuer un sort topologique sur G_0, en mettant les arcs en ordre de façon qu'aucun arc soit un lien entre le sommet *u* vers le sommet *v* dans G_0, puis *u* précède *v* dans l'ordre total.
3. Aller dans les sommets dans l'ordre est défini par le sort topologique. On calculant $\Delta(v)$ comme suit
   a. Si aucun arc est entrant au sommet v, alors $\Delta(v) \leftarrow d(v)$.
   b. Sinon, $\Delta(v) \leftarrow t(v) + \max \{ \Delta(v) : u \xrightarrow[e]{} v \text{ et } d(e) = 0. \}$
4. La période *cl(G)* est $\max \Delta(v)\ v \in V$.

}

Une fois qu'on a trouvé le temps de cycle minimum, on utilise la solution du système d'équations comme vecteur de resynchronisation pour déplacer les retards du graphe. Le temps total est en $O(|V|\ |E|\ lg(|V|))$ [Boy01].

Dans l'exemple de la Figure 4.2, on trouve que le temps de cycle minimum possible est 3. Le vecteur de resynchronisation est {1,0,0}.

4. Définition et propriétés

À fin de définir les propriétés de la resynchronisation. On va présenter deux exemples de circuits synchrones modélisés par les graphes de flot de données. Ces deux exemples sont tirés de l'article [Lei91].

4.1. Filtre FIR

La resynchronisation a plusieurs applications dans la conception de circuits synchrones. Ces applications incluent la minimisation de la période de l'horloge du circuit, réduire le nombre de

registres, réduire la consommation énergétique, et la synthèse logique. Par exemple, si on considère le filtre FIR dans la Figure 4.3 (a). Ce filtre est décrit par :

$W(n) = ay(n-1) + b\,y(n-2)$

$Y(n) = w(n-1) + x(n)$

$= ay(n-2) + by(n-3) + x(n).$

Le filtre dans la Figure 4.3 (b) est décrit par :

$W_1(n) = ay(n-1)$

$W_2(n) = by(n-2)$

$Y(n) = w_1(n-1) + w_2(n-1) + x(n)$

$= ay(n-2) + by(n-3) + x(n).$

Bien que les filtres présentés par les graphes (a) et (b) dans la Figure 4.3, aient des retards dans différentes locations, ces filtres ont les mêmes caractéristiques d'entrée/sortie. Ces deux filtres peuvent être dérivés de l'un à autre par l'utilisation de la resynchronisation.

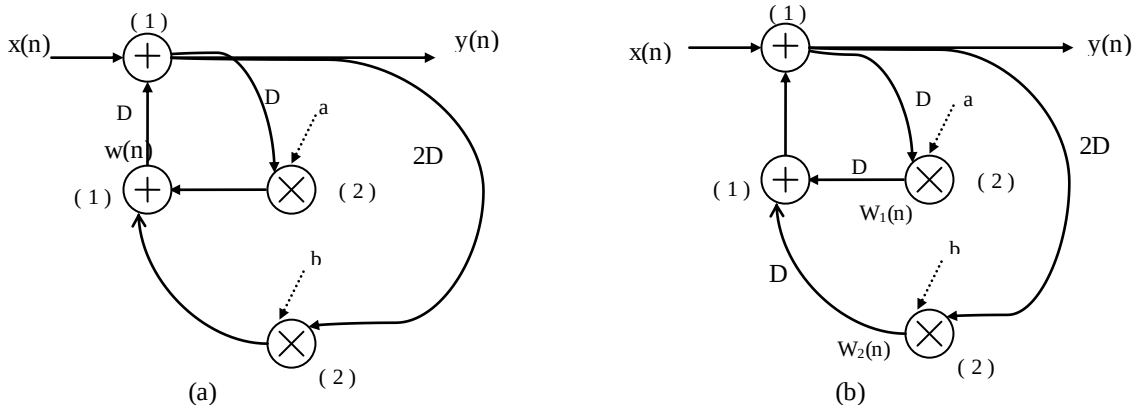


Figure 4.3 : Deux versions du filtre FIR.

Par exemple dans le graphe (a) de la figure 4.3 le chemin critique est le chemin le plus long qui passe à travers un multiplieur et un additionneur, avec un temps de calcul de 3 unités de temps. Le filtre après resynchronisation, présenté par le graphe (b), a un chemin critique qui passe par deux additionneurs, avec un temps de calcul égal à 2 unités de temps. La période d'horloge du circuit est réduite de 3 unités de temps à 2 unités de temps avec un gain de 33% (selon l'algorithme polynomial précédent).

La resynchronisation peut être utilisée pour réduire le nombre de registres dans un circuit [Hil04]. Le filtre dans la figure 4.3(a) utilise 4 registres pendant que le filtre dans la figure 4.3(b) utilise 5 registres. Depuis que la resynchronisation peut affecter la période de l'horloge et en même temps le nombre de registres, il est désirable de prendre ces deux paramètres en considération. Le graphe (c) de la figure 4.4, présente une resynchronisation du filtre FIR qui minimise le nombre de registre.

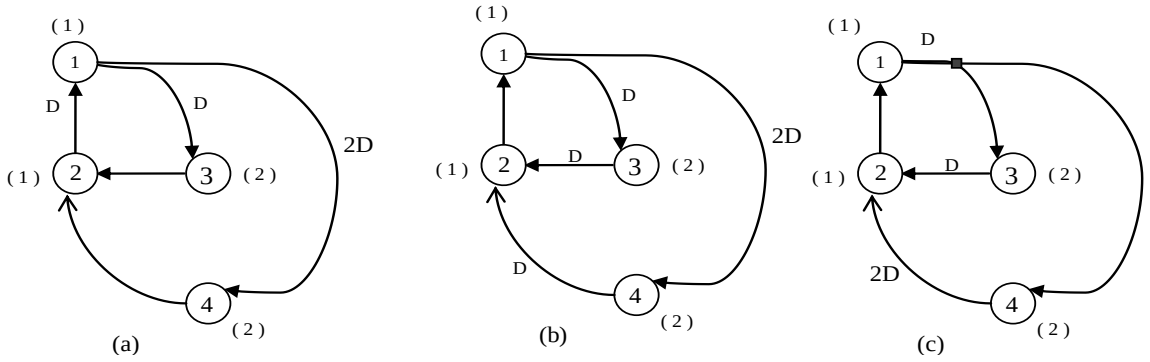


Figure 4.4 : (a) DFG. (b) DFG resynchronisé en utilisant r(1) =0, r(2) =1, r(3) =0, et r(4) =0. (c) DFG resynchronisé pour minimiser le nombre de registre

4.2. Une description quantitative de la resynchronisation

La solution de la resynchronisation est caractérisée par une valeur *r(V)* pour chaque sommet *V* dans le graphe. *d(e)* dénote le poids de l'arc *e* dans le graphe original *G*, et *dr(e)* dénote le poids de l'arc *e* dans le graphe resynchronisé *Gr*. Le poids de l'arc $U \xrightarrow{e} V$ dans le graphe resynchronisé, est calculé à partir du poids de l'arc dans le graphe orignal en utilisant :

$$dr(e) = d(e) + r(v) - r(u)$$

Afin de démontrer les aspects formels de la resynchronisation, le filtre dans la figure 4.3 (a) est redessiné dans la figure 4.4 (a), et le filtre dans la figure 4.3 (b) est redessiné dans la figure 4.4 (b). Les valeurs de resynchronisation r*(1) =0, r(2)=1, r(3)= 0*, et *r(4)= 0* peuvent être utilisées pour obtenir le DFG (b) à partir du DFG (a) de la Figure 4.4.

Par exemple, l'arc 3 $\rightarrow$ 2 dans le graphe resynchronisé DFG contient un seul retard.

$$dr(3 \rightarrow 2) = d(3 \rightarrow 2) + r(2) - r(3) = 0 + 1 - 0 = 1$$

Et l'arc 2 $\rightarrow$ 1 contient $dr(2 \rightarrow 1) = d(2 \rightarrow 1) + r(1) - r(2)$

$= 1 + 0 - 1 = 0$ Retards.

La solution de resynchronisation est faisable, si le poids $d_r(e) \geq 0$ pour tous les arcs. Cependant la solution qui trace le graphe (b) à partir du graphe (a) de la figure 4.4 est faisable, car tous les arcs du graphe ont des poids non négatifs. La solution *r(1) =0, r(2) = -1, r(3) =0, r(4) =0* n'est pas faisable, par exemple, l'arc $3 \rightarrow 2$ dans le graphe resynchronisé contiens -1 retard.

$$dr(3 \rightarrow 2) = d(3 \rightarrow 2) + r(2) - r(3) = 0 + (-1) - 0 = -1$$

4.3. Les propriétés de la resynchronisation

Les différentes propriétés de la resynchronisation peuvent être dérivées à partir de son équation. Un chemin est une séquence d'arcs et sommets :

$V0 \xrightarrow{e0} V1 \xrightarrow{e1} \ldots \xrightarrow{ek-2} Vk-1 \xrightarrow{K-1} V0.$

Le poids du chemin *p* est $d(p) = \sum_{i=0}^{k-1} d(ei)$ et le temps de calcul de ce chemin est $t(p) = \sum_{i=0}^{k} t(vi)$.

Un cycle est un chemin fermé $V0 \xrightarrow{e0} V1 \xrightarrow{e1} \ldots \xrightarrow{ek-2} Vk-1 \xrightarrow{K-1} V0.$

Le poids d'un cycle *C* est $d(c) = \sum_{i=0}^{k-1} d(ei)$ et le retard d'un cycle est $t(c) = \sum_{i=0}^{k-1} t(Vi)$.

Propriété 1: Le poids d'un chemin resynchronisé $p = V_0 \xrightarrow{e0} V_1 \xrightarrow{e1} \ldots \xrightarrow{ek-1} V_k$ est donné *par* $d_r(p) = d(p) + r(V_k) - r(V_0)$.

$$dr(p) = \sum_{i=0}^{K-1} d_r(e_i)$$

$$= \sum_{i=0}^{K-1} (d(e_i) + r(V_{i+1}) - r(V_i))$$

$$= \sum_{i=0}^{K-1} d(e_i) + (\sum_{i=0}^{K-1} r(V_{i+1}) - \sum_{i=0}^{K-1} r(V_i)$$

$$= d(p) + r(V_k) - r(V_0).$$

Par exemple, le chemin $2 \rightarrow 1 \rightarrow 3$ dans la figure 4.4 (a) contient deux retards, et ce chemin est resynchronisé dans le *DFG* de la figure 4.4 (b) contient un seul retard.

$$dr(2 \rightarrow 1 \rightarrow 3) = 2 + r(3) - r(2) = 1 \text{ retard.}$$

***Propriété 2* :** La resynchronisation ne change pas le nombre de retards dans un cycle.

Cette propriété est un cas particulier de la première propriété, où $V_k = V_0$. Le poids du cycle resynchronisé *c* est :

$$d_r(c) = d(c) + r(V_0) - r(Vj) = d(c).$$

Le cycle $1 \rightarrow 3 \rightarrow 2 \rightarrow 1$ contient deux retards dans le graphe original, et aussi dans le graphe resynchronisé, et le cycle $1 \rightarrow 3 \rightarrow 2 \rightarrow 1$ contient trois retards dans le graphe original et le même nombre de retards dans le graphe resynchronisé.

Propriété 3 : La resynchronisation ne modifie pas la limite d'itération (Iteration Bound) dans un DFG.

Propriété 4 : L'ajout d'une valeur constante à la valeur resynchronisée de chaque sommet ne change pas le mappage du graphe *G* au graphe *Gr*. Si on remplace *r(V)* avec *r(V)+j* pour chaque sommet, le poids de l'arc resynchronisé $U \xrightarrow{e} V$ dans *Gr* est :

$$d_r(e) = d(e) + (r(v) + j) - (r(u) + j)$$
$$= d(e) + r(v) - r(u)$$

Cette valeur est la même pour chaque valeur de j (j =0 est incluse).

Les valeurs de resynchronisation *r(1) = 0*, *r(2) = 1*, *r(3)= 0*, et *r(4) =0* sont utilisées pour obtenir le graphe DFG resynchronisé dans la figure 4.4(b) à partir du graphe original dans la figure 4.4(a).

En ajoutant, par exemple, la constante -38 à chaque valeur r(v). Les valeurs de la resynchronisation seront donc r(1) = -38, r(2) = -37, r(3) = -38, et r(4) = -38, et qui peuvent être utilisées pour obtenir le graphe resynchronisé (b) à partir du DFG (a) de la figure 4.4.

5. Aspect formel de la resynchronisation

La notion de resynchronisation dans le cadre de la transformation de code liée à la notion de déplacement des opérations d'un nid de boucle (boucles imbriqué) dans leur domaine d'itération. Appliquer une resynchronisation à une instruction d'un nid de boucles revient à appliquer une translation à son domaine d'itération. La notion de temps dans une boucle étant intimement liée à l'ordre lexicographique sur son domaine d'itération, appliquer une resynchronisation à une instruction revient donc également à déplacer ses instances dans le temps.

Plus formellement, une resynchronisation est une fonction $r : V \rightarrow Z^p$ qui associe à chaque sommet d'un graphe de dépendance, donc à chaque instruction d'un nid de boucles, un vecteur entier de la profondeur de l'instruction correspondante. Cette valeur de resynchronisation correspond à un retard qui sera appliqué à l'exécution de toutes les opérations associées au sommet concerné. Ce retard est exprimé en nombre d'itérations par lequel on souhaite reporter l'exécution des opérations (nombre d'itération sur chaque boucle du nid, à interpréter grâce à l'ordre lexicographique). Appliquer une resynchronisation *r* au graphe *G = (V, E)* signifie que chaque opération *(u, i)*, correspondant à l'instruction *u* dont la valeur de resynchronisation est *r(u)*, n'est plus exécutée à l'itération *i*, mais à l'itération $i + r(u)$[10]. Ainsi, une fois la resynchronisation appliqué, chaque instruction *u* de domaine $D_u = \{1,\ldots,n\}$ voit son domaine changé en $D'_u = \{1+r(u),\ldots, n+r(u)\}$ obtenu par simple translation de vecteur r(u), d'où le terme décalage d'instruction est utilisé au lieu du terme resynchronisation dans[Hua01].

L'application d'une resynchronisation *r* à une instruction *u* la fait s'approcher d'autant d'itération de ses successeurs et d'éloigner d'autant d'itération de ses prédécesseurs. En d'autres termes, après l'application d'une resynchronisation *r*, deux opération *(u,i)* et *(v,i')* ne s'exécutent plus respectivement aux itérations *i* et *i'*, mais respectivement aux itérations $i + r(u)$ et $i' + r(v)$. Ainsi la distance de dépendance entre ces deux opérations devient :

$$d_r(u,v) = (i' + r(v)) - (i + r(u))$$
$$= i' - r(v) - r(u)$$
$$= d(u,v) + r(v) - r(u)$$

Définition (resynchronisation d'un graphe) : Soit un graphe de dépendance $G = (V,E,d)$ de profondeur p. Une fonction r : V → Z^p est appelée une resynchronisation de G. Le graphe G_r = (V,E,dr) où $\forall e = (u,v) \in E,\ d_r(e) = d(e) + r(v) - r(u)$ est appelé graphe après resynchronisation de G et est noté G_r.

En termes de graphe, la modification de la distance de dépendance entre deux instructions peut être interprétée comme un " déplacement du poids des arcs à travers un sommet". Cette manière très intuitive de voir la resynchronisation permet de mieux comprendre l'effet de cette transformation sur un graphe de dépendance. La figure 4.5 représente ce déplacement du poids des arcs : sur cette figure chaque unité de poids est représentée par un rectangle noir. Comme nous pouvons constater, une resynchronisation de +1 appliquée sur un sommet enlève une unité de poids de tous ses arcs entrants pour l'ajouter à tous ses arcs sortants. C'est sous cette forme intuitive que la resynchronisation est présentée par Leiserson et Saxe dans [Lei91] ; rappelons toutefois que, dans leur présentation, la resynchronisation n'est pas une transformation de programme mais bel et bien un mouvement de registres (le poids des arcs) à travers des unités fonctionnelles (les sommets).

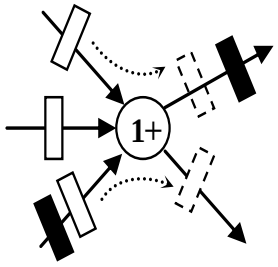


Figure 4.5 : Déplacement du poids produit par la resynchronisation

L'algorithme permettant de trouver une resynchronisation *r* d'un graphe *G* respectant la contrainte de poids minimal *m* ou de montrer qu'une telle resynchronisation n'existe pas, est présenté dans [Hua01]. Cet algorithme utilise l'algorithme de Bellman Ford pour calculer la longueur du plus court chemin d'un sommet *s* vers tous les autres, ou bien pour montrer que le graphe contient un circuit de poids négatif.

Algorithme 1 (algorithme de Bellman-Ford)

- Pour chaque sommet v$\in$ V définir $\pi(v) = +\infty$;
- Pour la source *s* des chemins cherchés définir $\pi(s) = 0$;
- Répéter $|V| - 1$ fois :
  - Pour tout arc $e = (u,v) \in E$, si $\pi(u) > \pi(v) + d(e)$ alors $\pi(u) = \pi(v) + d(e)$;

- S'il existe $e = (u,v) \in E$ tel que $\pi(u) > \pi(v) + d(e)$:
  - Alors G contient un circuit de poids négatif ;
  - Sinon π est la longueur cherchée.

Dans [Hua01], un autre algorithme de resynchronisation sous contrainte est présenté, qui trouve une resynchronisation *r* de *G* qui respecte la contrainte de poids minimal *m* ou conclut qu'il n'existe pas de telle resynchronisation :

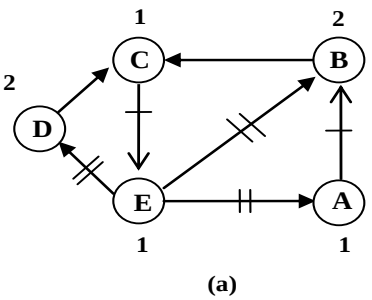
Algorithme 2 (resynchronisation sous contrainte)

- partir *G' = G – m(m le graphe de poids minimal)* ;
- Ajouter un sommet *s* à *G'* et un arc de poids nul de *s* vers chacun des autres sommets de G';
- Appliquer l'algorithme de Bellman-Ford pour trouver un plus court chemin de *s* vers chacun des autres sommets du graphe. Deux possibilités se présentent :
  - il existe un plus court chemin p_u de poids $\pi(u)$ de *s* vers chaque sommet *u* de *G'*, dans ce cas $r = -\pi(u)$ est la resynchronisation cherchée ;
  - Il existe un circuit de poids lexico négatif dans g, dans ce cas, il n'existe pas une re-synchronisation répondante à la contrainte.

6. Exemples de la resynchronisation

On va présenter quelques exemples d'application de la resynchronisation sur des boucles de programmes.

```
For  i =1 to n do
     A[i] = E[i-2]  + 9;
     B[i] = B[i-1]  * E[i -2];
     D[i] = E[i-2]  * 30;
     C[i] = B[i]  + D[i];
     E[i] = C[i - ]  + 5;
End
```



(a)

Figure 4.6 : Une boucle (a) le flot de données correspondant

La première boucle (Figure 4.6) est composée de cinq nœuds de calcul, avec un nombre total de retard égal à neuf, et un temps de cycle égal à trois unités de temps, après l'application de la

resynchronisation le temps de cycle est minimisé à deux unités de temps, avec un nombre de retards égal à sept (figure 4.7), en utilisant le vecteur de resynchronisation (1,1,1,1,0).

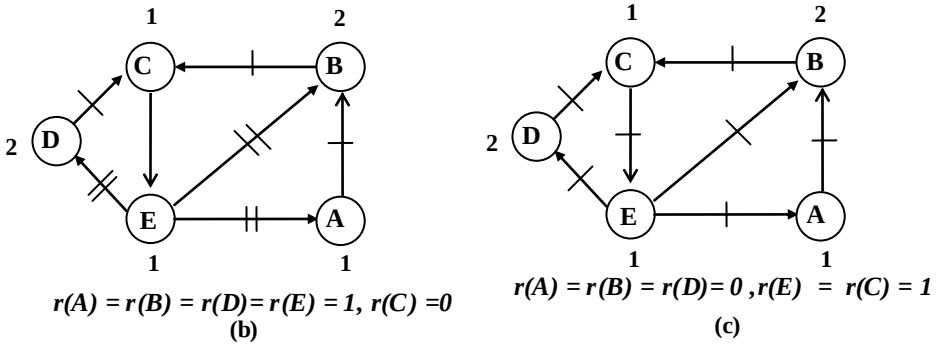


Figure 4.7 : DFG de la figure 4.7 resynchronisé cl = 2 (c) une autre resynchronisation *r'* (cl =2)

Un autre exemple d'une boucle, est présenté par le graphe flot de données dans la figure 4.8(a) et son DFG resynchronisé dans la figure 4.8 (b).

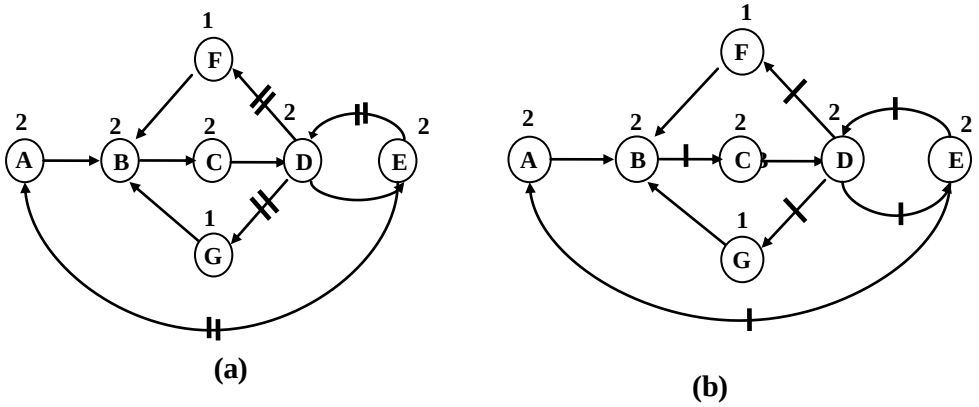


Figure 4.8 : (a) exemple de DFG avec cl(G) =8 (b) DFG resynchronisé avec cl(G) = 4

Dans cet exemple le nombre de retard de la boucle originale est huit, avec un temps de cycle égal aussi à huit unités de temps, avec l'application de la resynchronisation le temps de cycle est réduit par 4 unités de temps, tandis que le nombre de retards par deux retards seulement.

7. Conclusion

La technique de resynchronisation est une technique agissant sur les éléments retard dans un graphe de flot de données. Plus précisément, elle définit une redistribution de l'ensemble des retards du graphe correspondant directement à un décalage dans le temps d'exécution de ses différentes parties. Définir une resynchronisation consiste donc à assigner à chaque unité fonctionnelle une valeur représentant la quantité de temps, par laquelle son exécution est retardée

par rapport à l'exécution de l'ensemble du sommet du graphe, tout en maintenant le même comportement fonctionnel.

Dans le cas de la modélisation des systèmes embarqués, qui repose principalement sur des algorithmes de traitement d'images et traitement du signal, contenant un nombre important de boucles, l'application et l'intégration de cette technique dans les différents outils de conception se voit intéressante, car le gain en temps d'exécution est un facteur pertinent dans la conception de tels systèmes.

Implémentation

Ce chapitre est une illustration de notre proposition. On va présenter premièrement l'outil de conception SynDEx, puis notre proposition, avec la démarche de son intégration, et les résultats de l'application de la resynchronisation sur quelques exemples pratiques modélisés par le graphe de flot de données.

1. Introduction

SynDEx est un environnement logiciel graphique interactif de développement pour applications temps réel supportant la méthode AAA (Adéquation Algorithme Architecture). Il est basé principalement sur l'interactivité avec l'utilisateur grâce à une interface graphique qui permet de spécifier le graphe d'algorithme de l'application (sous la forme d'un graphe flot de données synchrone ou un langage synchrone), le graphe de l'architecture (sous la forme d'un graphe d'opérateurs "processeurs" décrivant l'architecture cible) ainsi que les caractéristiques de ces graphes. SynDEx est ensuite capable de construire et d'afficher automatiquement le diagramme temporel de l'implantation optimisée de l'application grâce à l'heuristique qu'il renferme (Figure 5.1). Si les caractéristiques de ce graphe de l'implantation optimisé sont conforme avec les exigences temps réel de l'application. SynDEx génère automatiquement l'exécutif permettant l'exécution de l'algorithme sur l'architecture, libérant ainsi l'utilisateur des tâches lourdes de programmation bas niveau.

Les architectures matérielles supportées actuellement par SynDEx sont des architectures de type "Multiprocesseurs hétérogènes" (architectures basées sur des composants programmables à jeux d'instruction figé de type microprocesseurs, microcontrôleur, DSP, station de travail, utilisation d'ASIC dédiés).

Afin d'arriver à une meilleure implantation de l'application sur l'architecture matérielle, nous étions amenés, à travers ce travail, à introduire la technique de resynchronisation en vue d'améliorer le temps d'exécution des applications récursives telles que les applications de traitement de signal et de traitement de l’image, qui sont de plus en plus utilisés dans les architectures temps réels embarquées. Cette technique (présentée dans le chapitre IV) permet de réduire le temps d'exécution par la mobilisation des éléments retard dans le graphe de flot de données.

On va montrer à travers ce chapitre, quelques démonstrations de l'utilité d’intégrer cette technique, soit au niveau plus bas de la conception (circuit logique), soit au niveau spécification algorithmique (graphe flot de données de l'application).

2. Présentation de l'environnement logiciel SynDEx

La version originale de SynDEx est développée par le langage CAML. Ce langage fonctionnel, développé à l'INRIA, est particulièrement adapté à la manipulation d'objets mathématiques comme les graphes qui constituent la structure de base de la méthode AAA.

L'IHM de SynDEx, est développée en Tcl/Tk fournit une interface graphique utilisateur pour manipuler le cœur Caml de SynDEx. Elle permet à l'utilisateur de spécifier graphiquement

l'algorithme et l'architecture, de lancer et d'interagir avec l'heuristique d'optimisation en permettant de visualiser les résultats (prédiction des performances) de l'heuristique, et d'afficher graphiquement le graphe de voisinage de l'implantation et enfin de lancer la génération automatique de code. Tout d'abord l'utilisateur doit spécifier, à travers l'interface graphique de SynDEx, le graphe d'algorithme de son application à l'aide d'un graphe flot de données (conditionné factorisé hiérarchique) de son architecture ainsi que la contrainte de performances souhaitée.

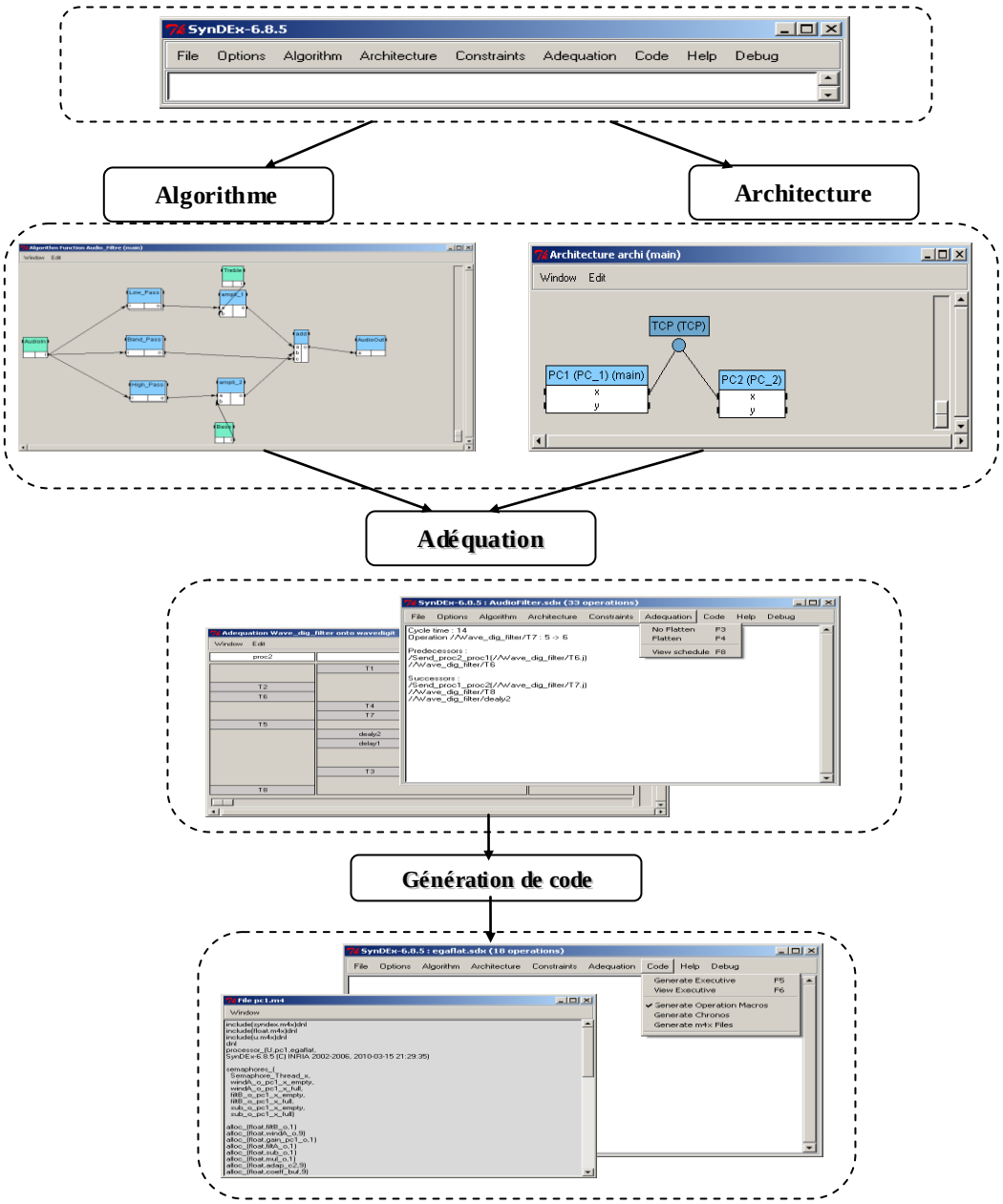


Figure 5.1 : Flot de conception SynDEx

3. Récapitulatif de notre proposition

Le modèle de notre proposition est représenté dans la figure 5.2. L'outil SynDEx sur lequel est implémentée la méthodologie AAA ne permet pas la resynchronisation, nous avons donc étendu la méthodologie AAA avec l'intégration de la méthode de resynchronisation, en basant sur la version SynDEx 6.8.5.

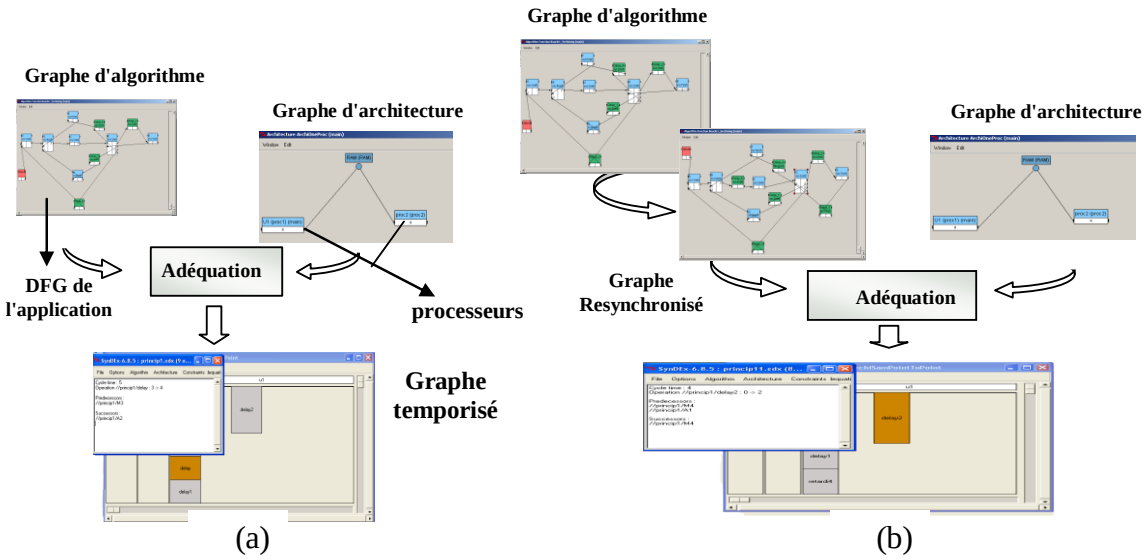


Figure 5.2 : (a) Flot de conception SynDEx (b) Intégration de la resynchronisation dans SynDEx

Comme le noyau de SynDEx n'est pas ouvert, on a donc spécifié directement des exemples avec leurs versions resynchronisés sous SynDEx, et on a construit une application (avec le langage OCAML, langage d'implémentation de SynDEx) qui permet de simuler l'exécution de l'algorithme de resynchronisation sur des graphes de flot de données, le code du programme est présenté dans l'annexe B.

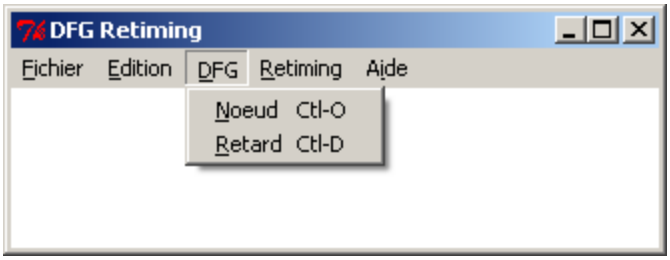
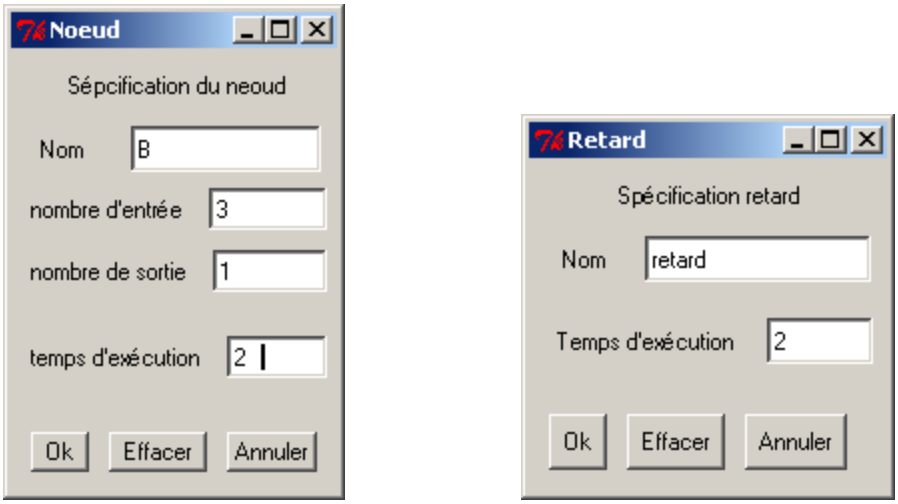


Figure 5.3 : Interface principale de l'application

L'interface graphique principale de l'application est présentée dans la figure 5.3, elle est constituée d'une barre de menu qui contient cinq boutons, dont trois (fichier, édition, aide) sont les boutons usuels d'une application Windows. A partir du menu « DFG », on peut dessiner un graphe flot de données, ce menu contient deux boutons « Nœud » et « Retard ». La fenêtre (a) de

la figure 5.4, correspond au bouton « Nœud », elle se compose de quatre champs, le premier pour donner un identificateur au sommet du graphe, le deuxième et le troisième champs pour spécifier le nombre de variables d'entrée et de sortie de chaque sommet, le quatrième champs pour introduire le temps d'exécution.

La fenêtre (b) de la figure 5.4 correspond au bouton « retard » du menu « DFG », elle est constituée de deux champs, le premier pour donner un identificateur au sommet retard, et le deuxième champ pour spécifier son temps d'exécution, (le nombre de variables d'entrée et de sortie pour un nœud retard est toujours à un).



(a) Nœud (b) Retard

Figure 5.4 : Spécification des nœuds et des retards

Après la confirmation par le bouton « ok » de la fenêtre nœud, un carré de couleur bleue est affiché dans la fenêtre principale, pour désigner un nœud de calcul. Par contre la spécification d'un nœud retard est présentée par un carré de couleur verte. Pour dessiner les arcs on clique deux fois sur le bouton droit de la souris pour activer la fonction de dessin qui permet de tracer et relier les différents nœuds et retards (figure 5.5). Pour exécuter l'algorithme de resynchronisation, on choisit à partir de la barre de menu le bouton « retiming », après l'exécution de l'algorithme de resynchronisation une deuxième fenêtre contenant le graphe resynchronisé est apparaît (figure 5.5). Dans cette dernière fenêtre, le temps de cycle minimum est affiché, pour indiquer à l'utilisateur le nouveau temps de cycle engendré par l'algorithme de resynchronisation.

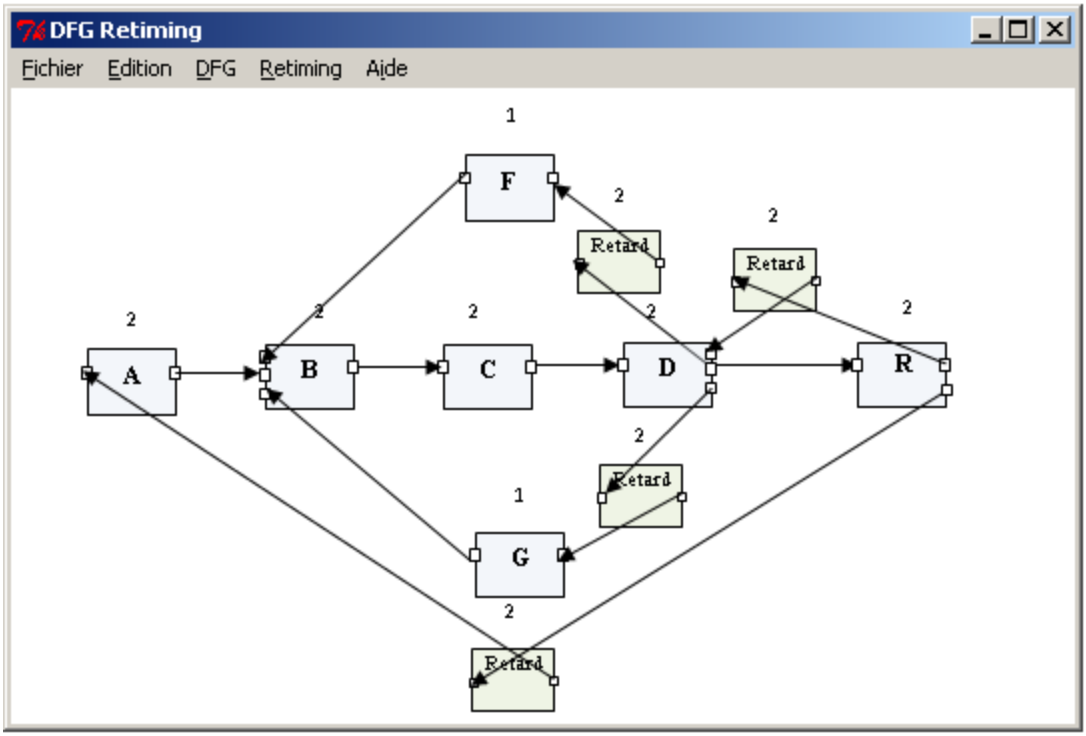


Figure 5.5 : L'exemple présenté dans l'application « DFG Retiming »

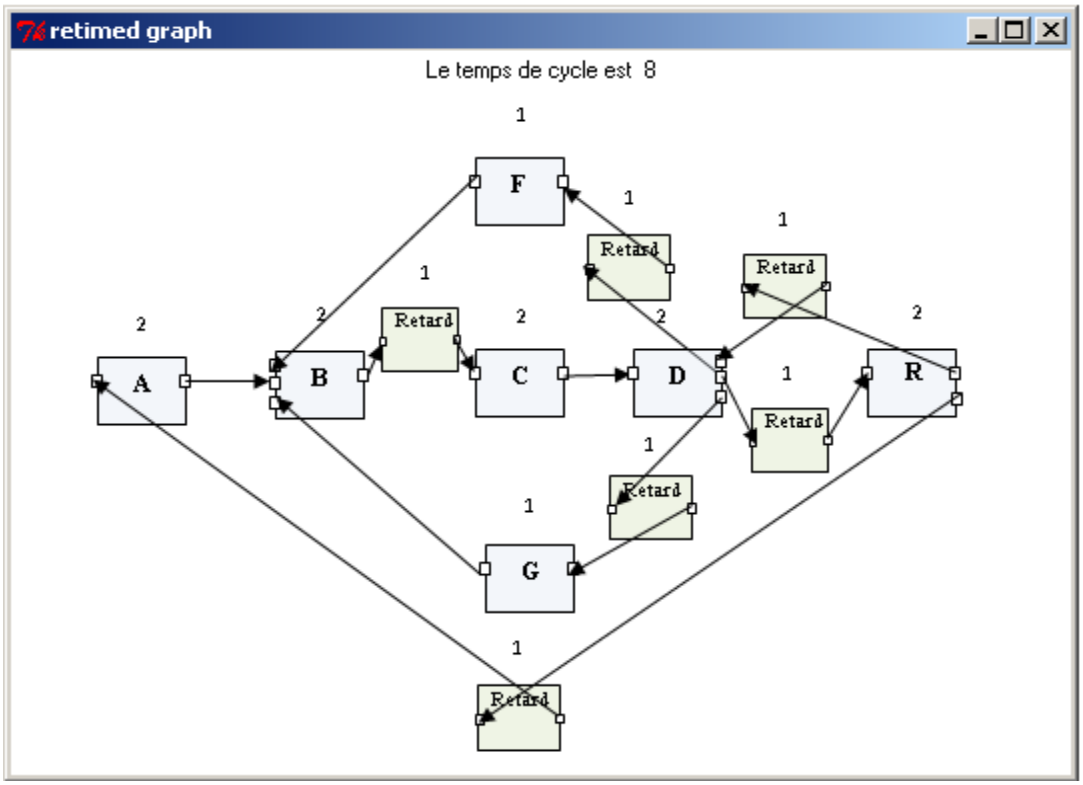


Figure 5.6 : l'exemple après l'exécution de la resynchronisation

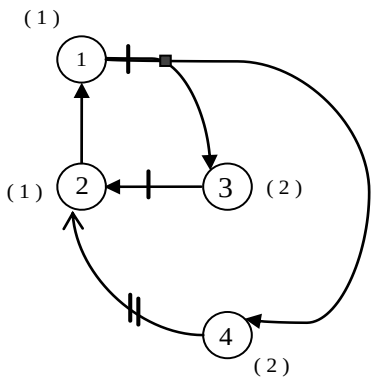
Sous SynDEx, cette technique peut être introduite comme un bouton à part dans la barre de menu, pour permettre aux utilisateurs de resynchroniser des applications récursives ou celles qui contiennent plusieurs éléments retard.

SynDEx intègre des librairies de fonctions les plus utilisées telles que les fonctions mathématiques d'addition, de multiplication etc. Dans notre cas, on peut ajouter les filtres resynchronisés directement dans les librairies de SynDEx à fin de faciliter leurs utilisations. Pour cela il suffit de les insérer dans la bibliothèque "int".

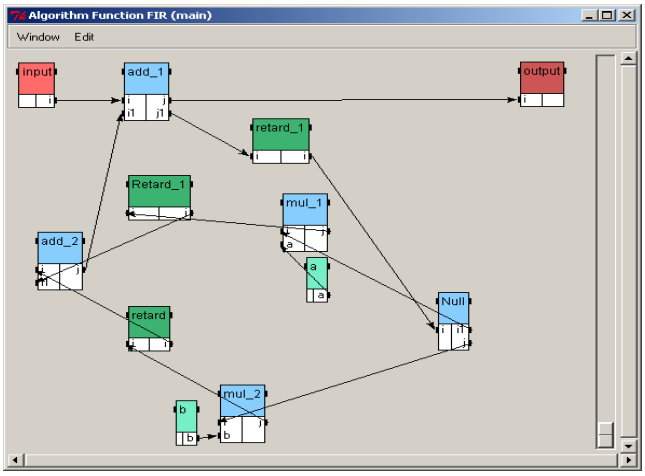
4. Exemples d'applications

4.1. Filtre FIR

Les filtres numériques constituent une alternative des filtres analogiques, ils travaillent sur des signaux numérisés et sont donc placés en aval des convertisseurs analogiques numériques, ils permettent de réaliser un nombre important d'opérations. Les filtres numériques sont implémentés dans des microprocesseurs DSP ou des composants spécifiques du type FPGA par des algorithmes de calcul. Ces algorithmes sont caractérisés par un ensemble de coefficients permettant de déterminer la valeur d'un échantillon de sortie en fonction des échantillons d'entrée et/ou de sortie précédents. Le filtre FIR est un filtre numérique qui est utilisé pour implémenter les fonctions de filtrage passe-haut, passe-bas, et passe-bande, le graphe (a) de la figure 5.7 présente l'équation de l'audio filtre modélisée par un graphe de flot de données.



(a) DFG du filtre FIR



(b) DFG sous SynDEx

Figure 5.7 : présentation du filtre FIR resynchronisé sous SynDEx.

La figure 5.7 montre une copie d'écran du graphe de flot de données du filtre sous SynDEx. Les sommets (de couleur bleue) sont les opérations d'addition et de multiplication à exécuter sur

les données. Les arcs sont les dépendances de données entre les opérations. Ce graphe est composé de quatre types de sommets : les sommets d'entrée (de couleur rouge) qui produisent des données, les sommets de calcul qui produisent et consomment des données, les sommets de sortie qui génèrent des données, et les sommets retards (en couleur verte).

Typiquement les sommets d'entrée sont les accès capteurs et ceux de sortie sont les accès actionnaires. Le graphe est exécuté itérativement ; les données traversent de la gauche vers la droite. Le temps d'exécution d'une itération du graphe correspond au temps de cycle.

Le temps d'exécution du filtre FIR sur une architecture contenant un seul élément de calcul est de 10 unités de temps, pendant qu'il est 9 unités de temps avec l'application de la resynchronisation.

4.2. Présentation de l'application Audio Filtre

L'efficacité de l'utilisation des filtres resynchronisés se voit dans le cas où l'application contient plusieurs filtres, comme celle de l'exemple suivant qui représente un audio filtre composé de trois filtres FIR.

La Figure 5.8 montre la structure de l'audio filtre tirée de [San03], qui est un sous système d'un égaliseur numérique. La tâche de ce sous-système est l'amplification des différentes fréquences du signal audio indépendamment, selon les niveaux du signal (Basse, Haute) assignés à chaque filtre. Le signal audio est découpé en trois signaux identiques, chaque signal pour une région de fréquence. Les signaux sont filtrés puis amplifiés selon le niveau d'amplification attribué. Comme l'audio filtre contrôle que les niveaux bas et hauts, les fréquences moyennes ne sont pas amplifiées. Le signal de sortie de l'audio filtre est l'addition des trois signaux filtrés et amplifiés.

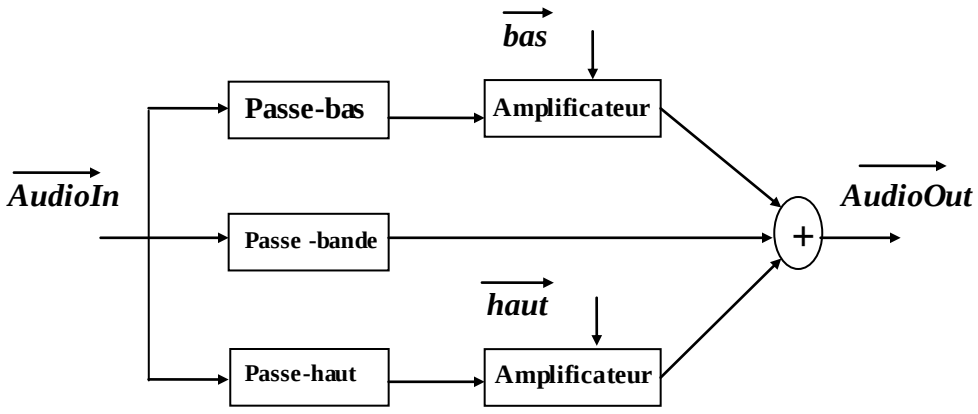


Figure 5.8 : Le sous-système système de l'audio filtre

On va modéliser cette structure par un graphe hiérarchique à partir de la figure 5.8. Le graphe de cet exemple sera composé de cinq nœuds de calcul. L'audio filtre a les coefficients des filtres passe-bas, passe-bande et passe-haut en tant que paramètres.

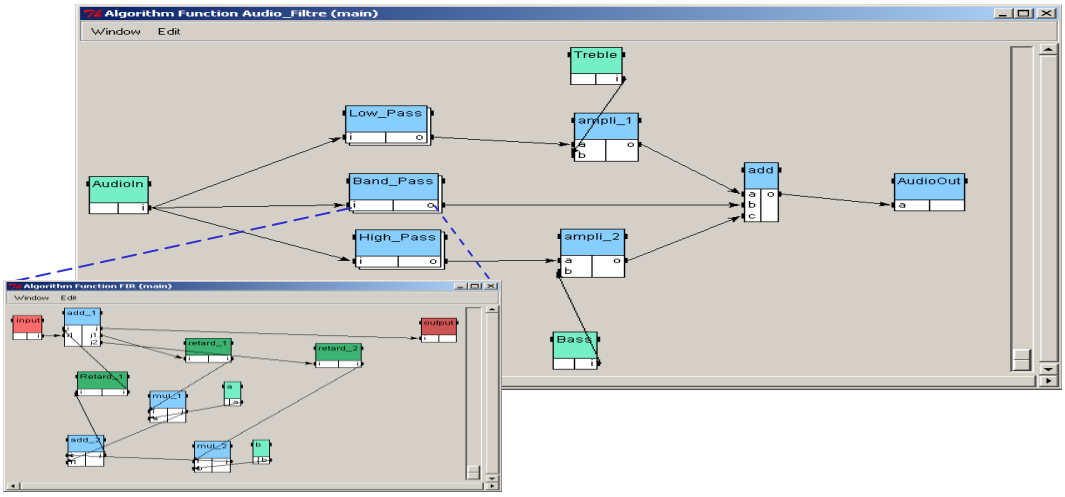


Figure 5.9 : Présentation du filtre sous SynDEx

Si on suppose que l'architecture matérielle soit composée d'un seul processeur, le temps d'exécution sans utilisation de la technique de resynchronisation est de 34 unités de temps (si on suppose que le temps d'exécution de l'amplificateur est 4 unités), par contre avec l'utilisation des filtres resynchronisés est de 31 unités de temps.

4.3.Exemple de boucle

La resynchronisation est utilisée aussi dans les algorithmes de traitement de signal et de traitement d'images, où les boucles constituent le chemin critique de ces applications, à fin de réduire le temps de cycle. Dans ce qui suit on va présenter quelques boucles et leurs spécifications sous SynDEx.

For i =1 to n do

A[i] = E[i-2] + 9;

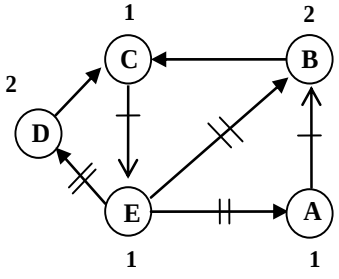
*B[i] = B[i-1] * E[i -2];*

*D[i] = E[i-2] * 30;*

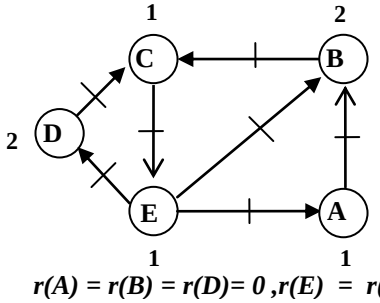
C[i] = B[i] + D[i];

E[i] = C[i -] + 5;

End



(a)

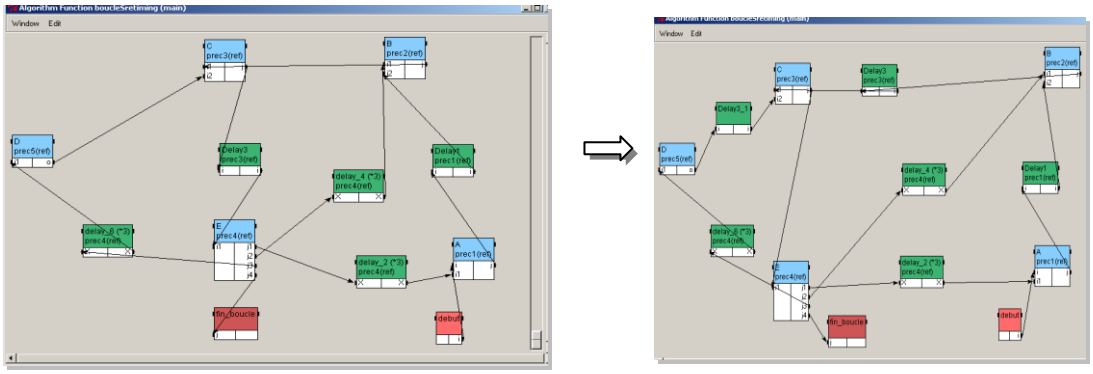


r(A) = r(B) = r(D)= 0 ,r(E) = r(C) = 1

(b)

Figure 5.10 : Une boucle (a) le flot de données correspondant (b) DFG resynchronisé cl =2

La boucle présentée par le graphe (a) de la figure 5.10, est composée de cinq nœuds de calcul avec un temps de cycle égal à trois unités de temps, après l'application de la resynchronisation, le temps de cycle est diminué à deux unités (Figure 5.10). La figure 5.11 présente la spécification de ce DFG et sa version resynchronisée sous SynDEx.



(a) (b)

Figure 5.11 : DFG de la figure 5.10 sous SynDEx

Dans le tableau suivant, on va présenter le temps d'exécution de cette boucle sur une architecture qui contient un seul élément de calcul (processeur), puis deux éléments de calcul.

	Avec un seul processeur	Avec deux processeurs
Sans resynchronisation	14	10
Avec resynchronisation	12	9

Tableau 5.1 : Tableau qui présente le temps d'exécution de la figure 5.11

L'application de la resynchronisation induit des améliorations significatives dans le temps d'exécution, les résultats de quelques autres exemples sont montrés dans le tableau suivant :

	Sans resynchronisation	Avec resynchronisation
Corrélateur	14	13
Boucle for1 (5 nœuds, 9 retards)	43	38
Boucle for2(7 nœuds, 6 retards)	36	30
Boucle (4 nœuds, 6 retards)	9	8

Tableau 5.2 : Résultats de l'application de la resynchronisation sous SynDEx

5. Conclusion

Nous venons de présenter dans ce chapitre quelques exemples pratiques de l'introduction de la technique de resynchronisation dans SynDEx. La technique de resynchronisation peut être utilisée pour optimiser les circuits contenant plusieurs registres, ainsi que les applications de traitement de signal et d'images où les boucles consistent le chemin critique de ces applications.

Une optimisation au niveau de la spécification combinée avec une heuristique d'optimisation au niveau d'ordonnancement qui est déjà implémentée dans SynDEx, nous a permet d'arriver à un meilleur temps d'exécution, sans changer ni les fonctionnalités du système, ni son architecture matérielle.

Cette technique peut être implémentée par l'introduction des composants resynchronisés dans une bibliothèque (par exemple FIR resynchronisé, corrélateur etc.), et par l'application directe de l'algorithme de resynchronisation sur les graphes de flots de données contenant plusieurs éléments retards.

Conclusions et perspectives

De plus en plus de champs d'applications (téléphone, radar, audio, multimédia ou encore radio logicielle) utilisent les technologies dédiées au traitement numérique du signal. Elles sont de plus en plus complexes et nécessitent des améliorations des fonctionnalités telles que les interactions avec l'utilisateur, la manipulation de plusieurs objets multimédia.

L'implantation de telles applications est dans la plupart des cas un exercice délicat à maîtriser, pour la simple raison que la satisfaction de cette demande croissante en puissance de calcul et des contraintes temporelles, nécessite l'adoption d'approches de conception efficace. Ces approches sont adoptées dans le but d'arriver à une implantation performante des applications embarquées sur des architectures multi composants.

De ce fait, il existe de nos jours un réel besoin en méthodologies de développement de haut niveau qui soient associées à des environnements logiciels efficaces afin d'aider les développeurs d'applications à implanter conjointement et à développer rapidement ces applications temps réel distribuées et optimisées (i.e. qui respectent les contraintes temps réels et minimisent la taille des architectures). Pour répondre à ces besoins, plusieurs méthodologies et outils de conception conjointe ont été développés pour assister les développeurs d'applications temps réels dans les phases de spécification et d'implantation.

Cependant l'état d'art entrepris sur les outils de conception nous a permis de constater que, parmi les différents outils universitaires et commerciaux de conception conjointe existants, aucun ne nous permet d'optimiser l'application par l'utilisation de plus d'une technique d'optimisation dans deux phases différentes.

Notre objectif était la combinaison de plusieurs méthodes d'optimisation de temps d'exécution à fin d'atteindre les meilleures performances. La méthodologie AAA et son outil SynDEx essaye d'optimiser les applications embarquées à l'aide d'une heuristique mais à l'étape d'ordonnancement et de partitionnement. Pour cela nous avons utilisé la méthode de resynchronisation à l'étape de modélisation afin d'arriver à une meilleur optimisation du temps de cycle de l'application. Cette technique minimise le temps de cycle par la mobilisation des éléments retard, sans affecter le fonctionnement principal de l'application.

Avec quelques exemples, l'introduction de cette technique nous a permet de diminuer le temps de cycle de l'application implémentée par rapport à leurs implémentation sans resynchronisation. Mais il faut signaler que l'utilisation de la resynchronisation sans répartition efficace des tâches peut entraîner une augmentation du temps de cycle.

Pour cela, nous envisageons une fenêtre d'aide à la répartition des tâches sur les composants de l'architecture, accompagnant la méthode de resynchronisation, ainsi une intégration de la méthode de resynchronisation dans le logiciel SynDEx-IC qui supporte les composants non programmables en particulier ceux qui sont reconfigurables tel que les FPGAs.

Références

[Alk02] A.Alkhodre, J.P Babau, J.J. Schwarz, "Méthodologie de développement des systèmes embarqués temps réel basée sur le langage SDL", La 3ème colloque de CAO de circuits et de systèmes intégrés, Lyon, France.2002

[All05] A. K. Allam," Power and memory optimization Techniques in embedded systems design", Thèse de doctorat, Université d'État de Louisiane, États-Unis d'Amérique, August 2005.

[Ata94] Y. Atamna, "Réseaux de Petri temporisés stochastiques classiques et bien formés : définition, analyse et application aux systèmes distribués temps réel", Thèse de Doctorat, Université Paul Sabatier, Toulouse, Octobre 1994.

[Bab05] J.P. Babau, "Formalisation et structuration des architectures opérationnelles pour les systèmes embarqués temps réel", Mémoire d'habilitation à diriger des recherches, Université Claude Bernard de Lyon, France, Décembre 2005.

[Bae90] J.C.M. Baeten, W.P. Weijland, "Process Algebra", Cambridge Tracts in Theoritical Computer Science, volume 18, Cambridge University Press, Cambridge, 1990.

[Ben04] K. Ben Chehida , "Méthodologie de Partitionnement Logiciel/Matériel pour Plateformes Reconfigurables Dynamiquement ", Thèse de doctorat, Université de Nice - Sophia Antipolis, France. Novembre 2004.

[Ber05] F.Bernichi, M. Mourlin, "Java Mobile Agents for Monitoring Mobile Activities", IEEE The International Conference on Computer as a Tool.page(s): 52 – 55, 2005.

[Bha96] S. Bhattacharya, K.P. Murthy, and E.A. Lee, "Software synthesis from dataflow graphs",

Kluwer Academic, 1996.

[Boy01] F.R Boyer," Optimisation lors de la synthèse de circuits à partir de langages de haut niveau", Thèse de Doctorat, Université de Montréal, Faculté des arts et des sciences, Canada, Avril, 2001.

[Bus98] R.Büssow , R.Geisler , M.klar,"Specifying safety-critical embedded systems with statecharts and Z : A case study", Lecture Notes in Computer Science, International Conference Fundamental approaches to software engineering, vol. 1382, pp. 71-87, 1998 .

[Cal00] J. Stockwood, R. Harr, T. Callahan, U. Kurkure, E. Darnell, Y. Li. "Hardware-software co-design of embedded reconfigurable architectures", the 37th Conference on Design Automation (DAC'00), Los Angeles, june 2000.

[Cal98] P.Y Calland et Y.Robert, "Circuit Retiming Applied to Decomposed Software Pipelining", IEEE Transactions on Parallel and Distributed Systems, 9(1):24-35,1998.

[Cha00] E. Chailloux, P. Manoury, and B. Pagano,"Développement d'applications avec Objective CAML" ,O'Reilly, 2000.

[Cha04] N.Chabini , W.Wayne ," Reducing dynamic power consumption in synchronous sequential digital designs using retiming and supply voltage scaling", IEEE transactions on very large scale integration (VLSI) systems, vol. 12, n°6, pp. 573-589, 2004.

[Cha92] L.-F. Chao, E. H.-M. Sha, "Retiming and unfolding data-flow graphs", In Proceedings of the International Conference on Parallel Processing, pages II 33–40, 1992.

[Cha98] K. S. Chatha, R. Vemuri. "RECOD: A retiming heuristic to optimize resource and memory utilization in HW/SW codesigns", In Proceedings of the IEEE International Workshop on Hardware/Software Codesign, pages 139–143, 1998.

[Chi04] T. Chien-Hsun, L.Stuart, "Full parallel process for multidimensional wave digital filtering via multidimensional retiming technique", In IEEE International Symposium on Circuits and Systems, Vol 5. pp. 209-212, 2004.

[Cor03] L.A. Cortés, P.Eles, Z.Peng ,"Modeling and formal verification of embedded systems based on a Petri net representation", Journal of Systems Architecture: the EUROMICRO Journal, Volume 49, Pages: 571 – 598, December 2003.

[Cor98] H. Corpooral, I. Karkowski, "Design space exploration algorithm for heterogeneous multi-processor embedded system design", The 35th Conference on Design Automation Conference (DAC'98), 1998DAC'98, San Francisco, June 1998.

[Cos00] P. Coste, F. Hessel, A.A. Jerraya, " Multilanguage Codesign Using SDL and Matlab", SASIMI'00, Kyoto, Japan, april 2000.

[Dal99] C. M. Dal, G. Huszerl, K. Kosmidis, "Transformation of guarded statecharts for quantitative evaluation of embedded systems", Proceedings of 10th European Workshop on Dependable Computing, Vienna, 1999.

[Dar02] A.Darvas , I.Majzik , B.Beny, " Verification of UML Statechart Models of Embedded Systems", In Proc. 5th IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop (DDECS),2002.

[Dav97] B. P. Dave, G. Lakshminarayana, N. K. Jha, "COSYN: Hardware-Software Co-Synthesis of Embedded Systems", Design Automation Conference, 34th Conference on (DAC'97), pp.703, 1997.

[Dav99] J. Davis, R. Galicia, M. Goel, C. Hylands, E.A. Lee, J. Liu, X. Liu, L. Muliadi, S. Neuendorffer, J. Reekie, N. Smyth, J. Tsay, and Y. Xiong, "Ptolemy II: Heterogeneous concurrent modeling and design in java", Technical Report Technical Report UCB/ERL No. M99/40, University of California, Berkeley, CA 94720, July 1999.

[Den98] T. C Denk , K. K Parhi , "Exhaustive scheduling and retiming of digital signal processing systems", IEEE transactions on circuits and systems Analog and digital signal processing, vol. 45, no7, pp. 821-838 1998.

[Dey92] S. Dey, M. Potkonjak, S.G. Rothweiler, "Performance optimization of sequential circuits by eliminating retiming bottlenecks", In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, pages 504–509, 1992.

[Die08] D.Dietterle, "Embedded system protocol design flow based on SDL: from specification to hardware/software implementation", In ACM journal, la conférence International on Simulation tools and techniques for communications, networks and systems & workshops, 2008.

[Dig00] J.P. Diguet, G. Gogniat, P. Danielo, M. Auguin, J.L. Philippe, "The SPF Model", Proceedings of International Forum on Design Languages of (FDL'00), Tübingen, Germany, September 2000.

[Dit95] G. Dittrich. "Modeling of Complex Systems Using Hierarchical Petri Nets", Article IEEE, Codesign : Computer- Aided Software/Hardware Engineering, pp. 128-144.1995.

[Dro01] C. Drosos, M. Zayadine, D. Metafas, "Embedded real-time communication protocol development using SDL for ARM microprocessor", In ACM journal, Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops, 2001.

[Edw87] E. A. Lee, D. G. Messerschmitt, "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing", IEEE transactions on computers, Vol. C-36, No. 1, pp. 24-35, January, 1987.

[Ekp06] M.Ekpanyapong, S.Kyu Lim, " Integrated retiming and simultaneous Vdd/Vth scaling for total power minimization", Proceedings of the international symposium on Physical design Pages: 142 – 148, 2006.

[Fis05] R.Fischer, K.Buchenrieder, U.Nageldinger, "Reducing the Power Consumption of FPGAs through Retiming", On the 12 IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05), pp.89-94, 2005.

[Fre00] V. Fresse, M. Assouil, and O. Deforges, "Rapid prototyping for mixed architectures", In 25th IEEE International Conference on Acoustic, Speech and Signal Processing (ICASSP), Istambul, Turkey, 05-09, June 2000.

[Gar04] P .Garg, A.Gupta, J.W.Rozenblit, "Performance Analysis of Embedded Systems in the Virtual Component Co-Design Environment", Proceedings of the 11th IEEE International Conference and Workshop on Engineering of Computer-Based Systems (ECBS), 2004.

[Gha06] A. H. Ghamarian, M. C. W. Geilen, S. Stuijk, T. Basten , A. J. M. Moonen, M. J. G. Bekooij, B. D. Theelen, M. R. Mousavi, " Throughput analysis of synchronous data flow graphs", IEEE Computer Society Press, Proceedings of the Sixth International Conference on Applications of Concurrency to System Design (ACSD'06), pp. 27-30, June 2006.

[Gra02] T. Grandpierre," Un nouveau modèle générique d'architecture hétérogène pour la méthodologie AAA", Actes des Journées Francophone sur l'Adéquation Algorithme Architecture (JFAAA'2002), pages 6-9, Monastir Tunisie, Décembre 2002.

[Gran98] T. Grandpierre, C. Lavarenne, Y. Sorel, " Modèle d'exécutif distribué temps réel pour SynDEx", Rapport de Recherche 3476, INRIA, August 1998.

[Gri01] W. Grieskamp, M.Heisel, H.Dorr, "Specifying embedded systems with statecharts and Z: an agenda for cyclic software components", Dans Science of Computer Programming, conférence international n°1 fundamental approaches to software engineering (FASE'98), Volume 40, pp. 31-57(27), 2001.

[Gro07] P.Grosse, "Gestion dynamique des tâches dans une architecture micro-électronique intégrée à des fins de basse consommation", Thèse de Doctorat de l'Ecole Normale Supérieure de Lyon, France, 2007.

[Hal95] N. Halbwachs, " The declarative code DC, version 1.2a", Vérimag, Grenoble, France, http://www.inrialpes.fr/bip/people/girault/Documentations/Dc1/index.html, October 1995.

[Har87] D.Harel, A.Pnueli, "Statecharts: a visual formalism for complex systems", Science of Computer Programming, Vol. 8, p. 231, 1987.

[Hil93] P. N. Hilfinger," Silage Reference Manual", DRAFT Release 2.0, Computer Science Division, EECS Dep., UC Berkeley, Berkeley. July 1989.

[Hil04] E. Hill, L.Su, "Digital Filter Design Space Exploration Tools", Rapport de projet final, Spring On-line projects presentation, 'Array Signal Processors For Digital Signal Processing', 2004.

[Hoa83] C.A.R. Hoare, "Communicating Sequential Processes", ACM, International Series in Computer Sience, vol. 26, no1, pp. 100-106, 1983.

[Hol06] M. Holzer, B. Knerr, P. Belanovic, M. Rupp, "Efficient Design Methods for Embedded Communication Systems", EURASIP Journal on Embedded Systems, Article ID 64913, Pages 1-18, 2006.

[Hua01] G. Huard, "Algorithmique du décalage d'instructions", Thèse de doctorat, Ecole normale supérieur de Lyon, Laboratoire de l'informatique du parallélisme, 2001.

[Ise08] C.Isen, John, L. Jung Pil Choi Hyo Jung Song, "On the representativeness of embedded Java benchmarks", IEEE International Symposium on Workload Characterization, page(s): 153-162, 2008.

[Ish93] A.T. Ishii," Retiming gate-clocks and precharged circuit structures", In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, pages 300–307. 1993.

[Jam97] J. E. Saultz, " Rapid prototyping of application-specific signal processors (rassp) ", in progress report, Journal VLSI Signal Process. Syst., 15(1-2):29.47, 1997.

[Jan04] Jantsch, A., "Modeling Embedded Systems and SoCs", Morgan Kaufmann, 2004.

[Jun09] S. Jun, Y. Liu, J. S. Dong , C. Chen, "Integrating Specifications and Programs for System Specification and Verification", IEEE International Conference on Theoretical Aspects of Software Engineering TASE '09, 2009.

[Kan04] U.Kanade, "Performance of Work Conserving Schedulers and Scheduling of Some Synchronous Dataflow Graphs", Article IEEE computer society, The 10th International Conference on Parallel and Distributed Systems (ICPADS'04), 2004.

[Kao04] L. Kaouane, "Formalisation et optimisation d'applications s'exécutant sur architecture reconfigurable", Thèse de Doctorat, Université de Marne-La-Vallée, France, Décembre 2004.

[Kar92] R.E. Karp, R.E. Miller, "Properties of a model for parallel computations: Determinacy, termination and queuing", SIAM Journal of Applied Mathematics, 14(6):1390–1411, 1966.

[Ker09]O. Kermia, "Ordonnancement temps réel multiprocesseur de tâches non-préemptives avec contraintes de précédence, de périodicité stricte et de latence", Thèse de Doctorat, Université Paris XI, Mars 2009.

[Keu00] K. Keutzer, S. Malik, A. R. Newton, J. M. Rabaey, A. Sangiovanni-Vincentelli, "System-level design: Orthogonolization of concerns and platform-based design ", IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, 19(12):1523–1543, December 2000.

[Kie00] B. Kienhuis, E. Rijpkema, E. Deprettere, "Compaan: deriving process networks from Matlab for embedded signal processing architectures", Proceedings of the eighth international workshop on Hardware/software codesign, SanDiego, California, Pages: 13 – 17.May 3-5, 2000.

[Kim05] D. Kim, S.Ha, "Static analysis and automatic code synthesis of flexible FSM model", Proceedings of the conference on Asia South Pacific design automation, ASP-DAC, pp. 161-165, 2005.

[Kir97] D. Kirovski, M. Potkonjak, "System level synthesis of low power hard real time systems", Dans IEEE/ACM Design Automation Conference (DAC'97), Anaheim, California, United States, june 1997.

[Koc00] R.Kocik, Y.Sorel, "De la modélisation à la réalisation : réduction du cycle de développement des applications temps réel distribuées", La 8ième conférence on Real-time and embedded systems(RTS) Paris, Mars 2000.

[Kol98] t .kolloch , g.färber, " Mapping an embedded hard real-time systems SDL specification to an analyzable task network : A case study", Lecture Notes In Computer Science, Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems. 1998.

[Kua07] H. P.Kuan ; H. Matin ; G.Soheil ," Joint throughput and energy optimization for pipelined execution of embedded streaming applications", Proceedings of the 2007 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems, San Diego, California, USA, 2007.

[Kun85] S.Y. Kung, J. Whitehouse, T. Kailath, "VLSI and Modern Signal Processing", Prentice Hall Professional Technical Reference, Pages: 448, 1985.

[Lal95] K.N. Lalgudi, M.C. Papaefthymiou, "DELAY: An efficient tool for retiming with realistic delay modeling", In Proceedings of the ACM/IEEE Design Automation Conference, pages 304–309, 1995.

[Lam88] M. Lam, "Software pipelining: An effective scheduling technique for VLIW machines", In Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, pages 318–328, 1988.

[Lau95] R.Lauwereins, M.Engels, M.Ade, J.A.Peperstraete, " Grape-II : a system-level prototyping environment for DSP applications Computer", Computer Journal, Volume : 28 Issue : 2, Page(s) : 35 -43, Feb 1995.

[Lav97] C. Lavarenne, Y. Sorel, "Modèle unifié pour la conception conjointe logiciel-matériel", Revue Traitement du Signal, 14(6) :569–578,, 1997.

[Lee02] S. Lee, S. Yoo, K. Choi, "Reconfigurable SoC Design with Hierarchical FSM and Synchronous Dataflow Model", Tenth International Workshop on Hardware/Software Codesign, Estes Park, Colorado, May 6-8, 2002.

[Lee06] J. Lee, M.Choi, C. Sung, "Modeling Technique Applying an Object-Oriented Petri Net for Embedded System ", Article IEEE, Conférence Hybrid Information Technology, Volume 2, Issue 9-11 Page(s):642 – 64, November 2006.

[Lee87] E. A Lee, D.G Messerschmitt, "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing", IEEE Transactions on Computers,Volume C-36. Page(s):24 – 35, 1987.

[Lee98] E. A. Lee, "Modeling Concurrent Real-Time Processes using Discrete Events", Rapport Technique, University of California, Berkeley, March 1998.

[Lei91] C.E. Leiserson, J.B. Saxe, "Retiming synchronous circuitry", In Springer, Algorithmica, volume 6 pages :5– 35, 1991.

[Lem02] Y. Le Moullec, J. P. Diguet, J. L. Philippe, "Design-Trotter: a Multimedia Embedded Systems Design Space Exploration Tool", Article IEEE, Workshop on Multimedia Signal Processing (MMSP'02), US Virgin Islands, December 2002.

[Liu04] M. Liu, Q. Zhuge, Z. Shao, K. Chen, E. H.-M. Sha, "Loop Fusion via Retiming for DSP Applications", In Proc. 17th International Conference on Parallel and Distributed Computing Systems (ISCA PDCS), pp. 403-408, San Francisco, California, September 2004.

[Liu09] M. Liu, E. H.-M Sha, C.Xue, " Loop Fusion Technique with Minimal Memory Cost via Retiming", Proceedings of the ISCA 24th International Conference on Computers and Their Applications (CATA), Louisiana,USA, April 2009.

[Loc92] B. Lockyear, C. Ebeling, "Optimal retiming of multi-phase, level-clocked circuits", In Proceedings of the Brown/MIT Conference on Advanced Research in VLSI and Parallel Systems, pages 265–280, 1992.

[Loc98] M.Lockheed, "Gedae technical paper",Technical report, Advanced Technology Laboratories. Malardalen University, 1998.

[Lom04] C.Lombriser , M.André, " Embedded Task Machine with BTnode and FPGA", Semester Project, Université Zürich, Suise, 2004 .

[Lui93] Z. Lui, C. Corroyer, "Effectiveness of heuristics and simulated annealing for the scheduling of concurrent task. an empirical comparison", Proc. of PARLE'93, 5th Int. PARLE conference, Munich, Germany, June 14-17, pages 452-463, Nov 1993.

[Luo00] J. Luo, N. K. Jha, "Power-conscious Joint Scheduling of Periodic Task Graphs and Aperiodic Tasks in Distributed Real-time Embedded Systems", IEEE/ACM international conference on Computer-aided design, pages 357--364, Nov 2000.

[Mah98] N.Maheshwari, " Fast algorithms for retiming large digital circuits", Thèse de doctorat. Lowa State University.Lowa , Inde ,1998.

[Mar02] C.A.M. Marcon, F. Hessel, A.M. Amory, L.H.L. Ries, F.G. Moraes, N.L.V. Calazans, "Prototyping of embedded digital systems from SDL language: a case study", In 8 ième IEEE International High-Level Design Validation and Test Workshop, 2002.

[Mcf90] M. C. McFarland, A. Parker, R. Camposano, " The high-level synthesis of digital systems", Proceedings of the IEEE, Vol. 78, No. 2, pp. 301--318, Feb. 1990.

[Mil82] R.Milner, "A Calculus of Communicating Systems", Springer Verlag, New York, p 206. USA, 1982.

[Mon93] J.Monteiro, S.Devadas, A.Ghosh, " Retiming sequential circuits for low power", IEEE /ACM international conference on Computer-Aided Design, 1993.

[Mor01] L.Morell, D.Middleton, "The software engineering learning facility", Journal Computer Small Coll., 16(3):299.307, 2001.

[Mou04] M.R. Mousavi, P. Le Guernic, J.-P., Talpin, S.K. Shukla, T. Basten, "Modeling and Validation of Globally Asynchronous Design in Synchronous Frameworks", Proceedings of the Conference on Design Automation and Test in Europe (DATE'04), pp. 384--389, Paris, France, February 2004.

[Mue09] P.Mueller," Statecharts can provide you with software quality insurance" , Article dans " http://www.embedded.com/design/opensource/219400531", 2009.

[Mur02] P.K Murthy, E.A Lee, "Multidimensional Synchronous Dataflow", IEEE Transactions on Signal Processing, volume 50, No 7, Aug 2002.

[Nat01] M. Di Natale, A. S.Vincentelli, F. Balarin, "Scheduling Reactive Task Graphs in Embedded Control Systems", Seventh IEEE Real-Time Technology and Applications Symposium (RTAS'01), 2001.

[Nio97] R. Nikoukhah, S. Steer, "SCICOS, a dynamic system builder and simulator user's guide" - version1.0, Technical Report RT 207, INRIA, 1997.

[Noo07] H. Noori, F.Mehdipour, M.S. Zamani, K.Inoue,K.Murakami ," Handling Control Data Flow Graphs for a Tightly Coupled Reconfigurable Accelerator", Dans ACM, Proceedings of the 3rd international conference on Embedded Software and Systems, Pages249-260, 2007.

[Oh99] H. Oh, S. Ha, "A hardware/software co-synthesis technique based on heterogeneous multiprocessor scheduling", Dans Seventh International Workshop on Hardware/Software Co-Design (CODES'99), Rome, Italy ,1999.

[Pap94] M.C. Papaefthymiou, " Understanding retiming through maximum average delay cycles", Proceedings of the 3rd ACM symposium on Parallel algorithms and architectures, pages 65–84, 1994.

[Par05] S.Park, G.Kwon, S.Ha , "Formalisation of fFSM model and its verification", Lecture Notes in Computer Science, Springer, ‘Embedded Software and Systems’ ICESS, Volume 3820, Pages 361-372, november 2005.

[Par89] K.K. Parhi , D.G. Messerschmitt," Fully-static rate-optimal scheduling of iterative data-flow programs via optimum unfolding", In Proceedings of the International Conference on Parallel Processing, pages 209–216, 1989.

[Pas94] N. L. Passos, E. H.-M. Sha. "Full parallelism in uniform nested loops using multi-dimensional retiming", In Proceedings of the International Conference on Parallel Processing (ICPP '94), pages 130--133, August 1994.

[Pat05] H.D.Patel, S.K.Shukla,"Towards a heterogeneous simulation kernel for system-level models: a SystemC kernel for synchronous data flow models", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume 24, Page(s):1261 – 1271, 2005.

[Pen94] Z. Peng, K. Kuchcinski, "Automated Transformation of Algorithms into Register-Transfer Level Implementations ", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 13(2) :150-166, Feb. 1994.

[Per05] N. Pernet, Y. Sorel," Spécification et implantation des systèmes distribués temps réel de contrôle et traitement de données", Dans Actes des Journées Francophones sur l'Adéquation Algorithme Architecture, Dijon, France, Janvier 2005.

[Pet62] C.A. Petri, "kommunikation mit Automaten", Thèse PHD , Institut for Instrumentelle Mathematik, Bonn, Germany, 1962.

[Pet81] J. Peterson, "Petri Net Theory and the Modeling of Systems", Prentice-Hall, Englewood Cliffs, NJ. 1981.

[Rus07] Russello, G. Changyu Dong Dulay, N. "Enforcing Fine-Grained Authorization Policies for Java Mobile Agents", The 21st International Conference on Advanced Information Networking and Applications Workshops, page(s): 489 – 496, May 2007.

[Sad07] N. Sadou, "Aide à la conception des systèmes embarqués sûrs de fonctionnement", Thèse de Doctorat, Université de Toulouse, France, November 2007.

[San03] I. Sander, "System Modeling and Design Refinement in ForSyDe", Thèse de doctorat, Royal Institute of Technology, Stockholm. Suède, Avril 2003.

[Sch00] S. Schulz, T.C. Ewing, J.W. Rozenblit, "Discrete Event System Specification (DEVS) and StateMate StateCharts Equivalence for Embedded Systems Modeling", Proceedings of 7ième IEEE International Conference and Workshop on the Engineering of Computer Based Systems, 2000.

[She91] N. Shenoy, R.K. Brayton, "Retiming of circuits with single phase transparent latches", IEEE International Conference on Computer Design on VLSI in Computer and Processors, pages 86–89, 1991.

[She96] M. Sheliga, N.L. Passos, E. H-M. Sha." Fully parallel hardware/software codesign for multi-dimensional DSP applications", In Proceedings of the IEEE international Workshop on Hardware/Software Codesign, pages 18–25, 1996.

[Sil04] V. D. Silva, S. Ramesh, “Synchronous protocol automata for modelling and verification of system-on-chip bus architectures”, Proceedings of the Conference on Design Automation and Test in Europe (DATE'04), Paris, France, pp. 384--389, February 2004.

[Sim05] E. Simeu, " Test et surveillance intégrés des systèmes embarqués", Mémoire d'habilitation à diriger des recherches, Université Joseph Fourier de Grenoble, France, Septembre 2005.

[Sov07] C. Soviani, O.S.A Tardieu, "Edwards. Optimizing Sequential Cycles Through Shannon Decomposition and Retiming", In IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, Volume: 26.On page(s): 456-467, 2007.

[Soy94] T. Soyata, E. Friedman," Retiming with non-zero clock skew, variable register and interconnect delay", In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, pages 234–241, 1994.

[Sri97] G. Srinivasa, N. Prasanna, " Compilation of parallel multimedia computations extending retiming theory and Amdahl's law", Proceedings of the sixth ACM SIGPLAN symposium on Principles and practice of parallel programming, Pages: 180 – 192,1997.

[Ste01] A.Stephen, E.A. Lee, "The semantics and execution of a synchronous block-diagram language", Science of Computer Programming, University of California, Berkeley, September, 2001.

[Sun06] SUN MICROSYSTEMS. " J2ME Building Blocks for Mobile Devices " Whitepaper on KVM and the Connected, Limited Device Configuration CLDC. Sun Microsystems, May 2000. http://java.sun.com/products/cldc/wp/KVMwp.pdf, Accessed 8/9-2009.

[Sys03] SystemVerilog 3.1, "Accellera's Extensions to Verilog, available at www.vhdl.org/sv/SystemVerilog_3.1_final.pdf".2003.

[Tak01] Takino, S.Dawn Corp, Kobe, Japan, " 'GIS on the fly' to realize wireless GIS network by Java mobile phone", Proceedings of the Second International Conference on Web Information Systems Engineering (WISE'01),volume 2, page(s): 76 - 81 ,2001.

[Tei97] J. Teich, T. Blickle, L. Thiele," An evolutionary approach to system level synthesis", The 5th International Workshop on Hardware/Software Co-Design (Codes/CASHE '97), Germany, 1997.

[Tsa00] J. Tsay, " A code generation framework for ptolemy II", Technical Report ERL Technical Report UCB/ERL N° M00/25, Dept. EECS, University of California, Berkeley, CA 94720, 2000.

[Vah97] F. Vahid, "Modifying Min-Cut for hardware and software functional partitioning", The 5th International Workshop on Hardware/Software Co-Design (Codes/CASHE '97), Germany. 1997.

[Val03] K. S. Vallerio, N. K. Jha, "Task Graph Extraction for Embedded System Synthesis", IEEE Computer Society, 16th International Conference on VLSI Design, pp.480, 2003.

[Ven08] H.Venturini,"Le débogage de code optimisé dans le contexte des systèmes embarqués", Thèse de doctorat, Université Joseph Fourier de Grenoble, France, Mars 2008.

[Wud03] D. Wu, B. M. Al-Hashimi, P. Eles, "Scheduling and Mapping of Conditional Task Graph for the Synthesis of Low Power Embedded Systems", IEEE Proceedings Computers and Digital Techniques, Volume 150, Issue 5, Page(s): 262-73, Sept. 2003.

[Xue06] C. Xue, Z. Shao, M. Liu, M. Qiu, E. H.-M. Sha, "Optimizing Parallelism for Nested Loops with Iterational and Instructional Retiming", Publication in Journal of Embedded Computing (JEC), 2006.

[Yen95] T. Y. Yen, W. Wolf, "Sensitivity-driven cosynthesis of distributed embedded systems", Dans la 8 ième IEEE/ACM, International Symposium on System Synthesis (ISSS'95), 1995.

[Zho04] H.Zhou C.Lin, "Retiming for wire pipelining in system-on-chip", In IEEE Transactions on Computer-aided Design of integrated circuits and systems, Volume: 23, Issue: 9 page: 1338- 1345, 2004.

[Zhu06] Q. Zhuge, C. Xue, Z. Shao, M. Liu, M. Qiu, and E. H.-M. Sha, " Design Optimization and Space Minimization Considering Timing and Code Size via Retiming and Unfolding", In Journal of Microprocessors and Microsystems, Vol. 30, Issue 4, pp. 173-183, June 2006.

[Ziv96] V. Zivojnovic, R. Schoenen, "On retiming of multirate DSP algorithms", Proceedings on IEEE International Conference on Acoustics, Speech, and Signal Processing icassp, vol. 6, pp.3310-3313, 1996.

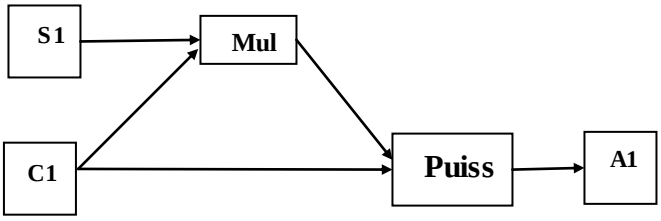
[Zlu02] Z. Lu, I. Sander, and A. Jantsch, "A case study of hardware and software synthesis in ForSyDe", In Proceedings of the 15th International Symposium on System Synthesis, Kyoto, Japan, October 2002.

Annexe A : les étapes de spécification sous SynDEx

1. Les étapes de spécification d'algorithme et d'architecture sous SynDEx :

1.1. Création d'algorithme :

On va considérer l'exemple d'un circuit simple avec :



S1 : Constante que l'on appellera ''constante'' et qui retourne un entier (la valeur de la constante est 2).

C1 : capteur que l'on appellera ''input'' et qui retourne un entier

Mul : Opération qui fait une multiplication entre 2 entiers, elle retourne un entier.

Puiss : Opération quelconque par exemple un composant dont la sortie i1 est puissance d'i2.

A1 : Actionneur que l'on appellera ''output'' et qui récupère un entier.

1.1.1. Création d'une fonction de multiplication sous SynDEx :

Pour spécifier une fonction sous SynDEx, on choisit à partir du menu principal :

Algorithm / New Local Definition / Function (Figure 1.1).

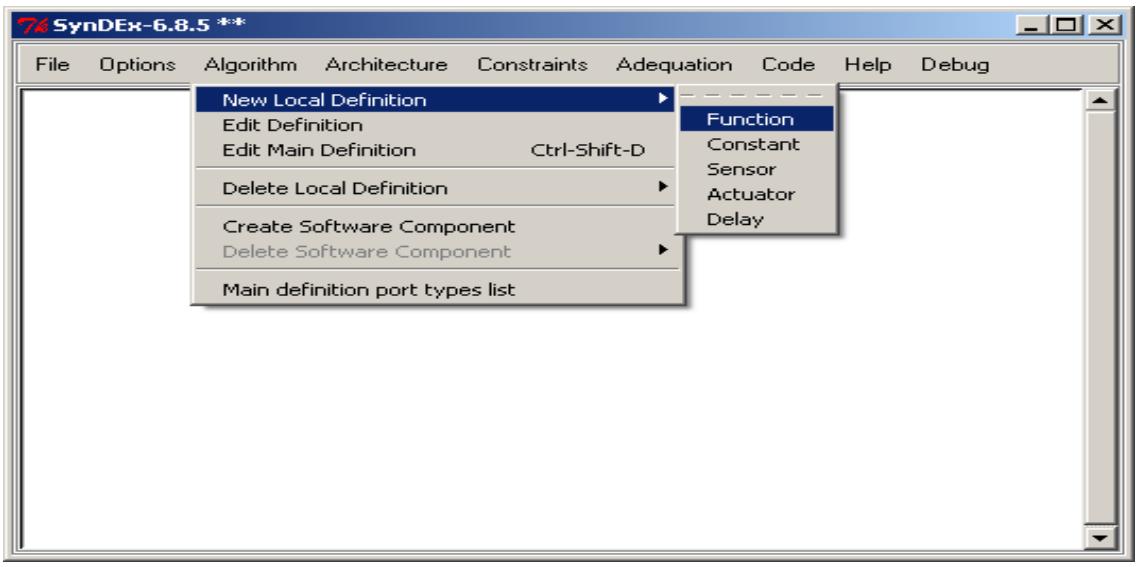


Figure 1.1: Algorithm / New Local Definition / Function

Cela va ouvrir une fenêtre de la figure 1.2, dont laquelle on introduit le nom de la fonction et la liste de ses arguments. Par exemple on écrit le nom « **Mul** » pour la fonction de multiplication, puis on clique sur Ok. Cela crée une définition de la fonction de multiplication nommé « **Mul** », et ouvre la fenêtre de définition correspondante figure 1.3.

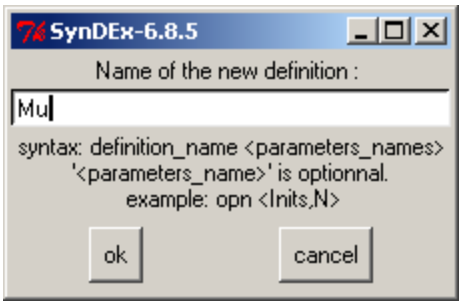


Figure 1.2 : Nom de la nouvelle définition

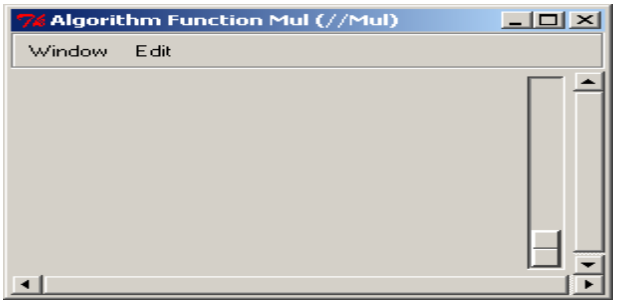


Figure 1.3 : Fenêtre de la définition

Dans la fenêtre définition, on introduit les ports qui représentent les variables d'entrée et de sortie de la fonction de multiplication :

Menu: Edit / Create Port: **"? int i1 ? int i2 ! int j ".** (Figure 1.4)

" ? int i1 " pour designer une variable d'entrée, **" ! int j "** pour désigner un variable de sortie.

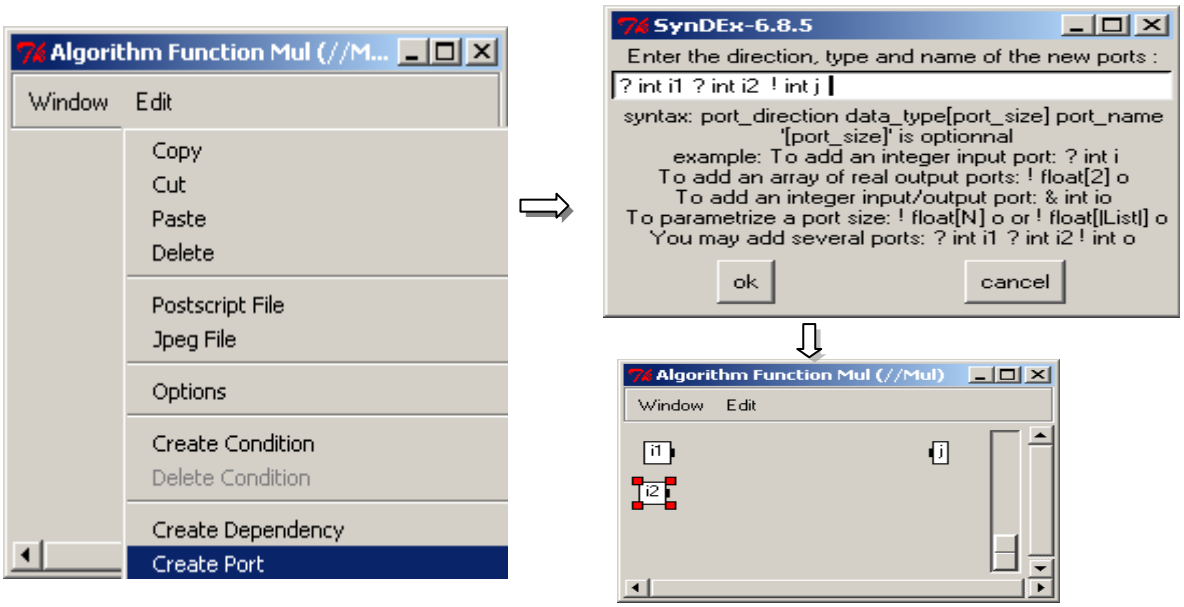


Figure 1.4 : Création des ports de la fonction de multiplication

De cette façon, la référence de la fonction multiplication est définit. On procède de la même manière pour définir la fonction « Puiss » :

- Création de la définition de la fonction '' Puiss ''

  Algorithm / New Local Definition / Function – on entre le nom de la définition (dans notre exemple, ce sera « Puiss »). Nous aurions pu aussi paramétrer la définition de ''Puiss'' en mettant en argument les variables à paramétrer : Puiss < N > (avec N la taille des éléments en entrée ou en sortie)

- Création des ports d'entrée (i1 et i2) et le port de sortie (o)
  - menu edit / Create Port
  - Définir le nom et le type du port, dans notre exemple, ce sera : ? int i1 ! int o

Pour créer la référence à la constante S1, on lance à partir du menu :

- Algorithm – New Local definition –Constant, puis on entre le nom de la constante, dans notre cas « S1» (Figure 1.5).

SynDEx possède des définitions standard : INPUT, OUTPUT, ADD, MUL…, qu'on peut les utiliser pour définir directement les fonctions C1 « INPUT » et A1 « OUTPUT », pour cela il faut introduire la bibliothèque « INT» (Figure 1.6).
On choisissant : File – include library - int

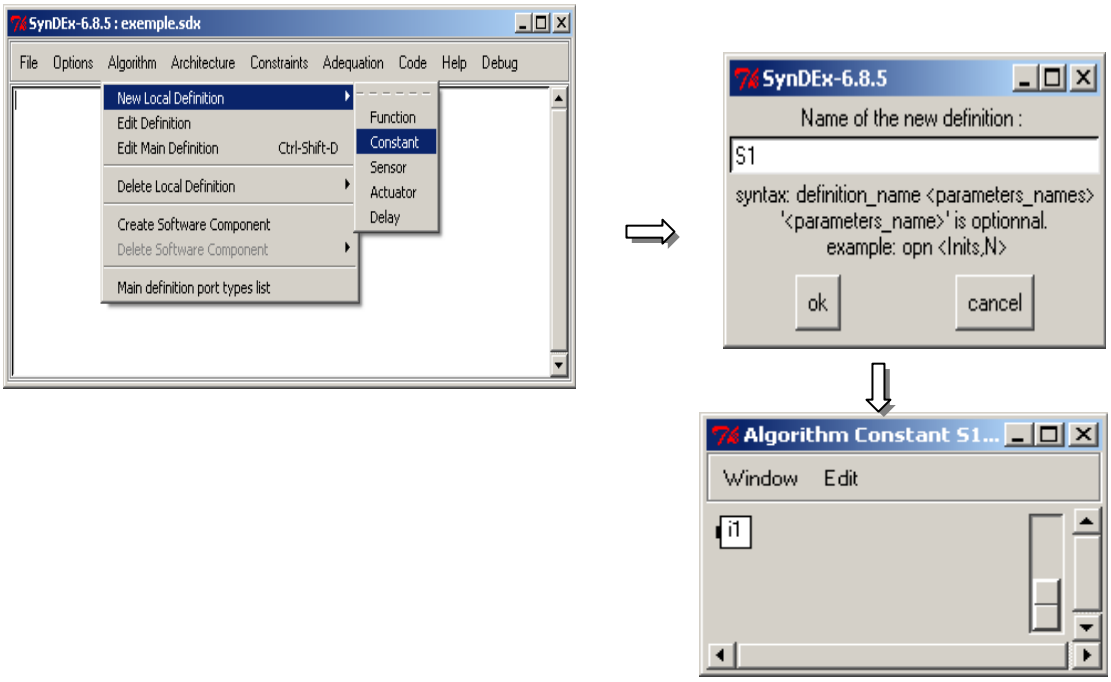


Figure 1.5 : création de la définition de la constante S1

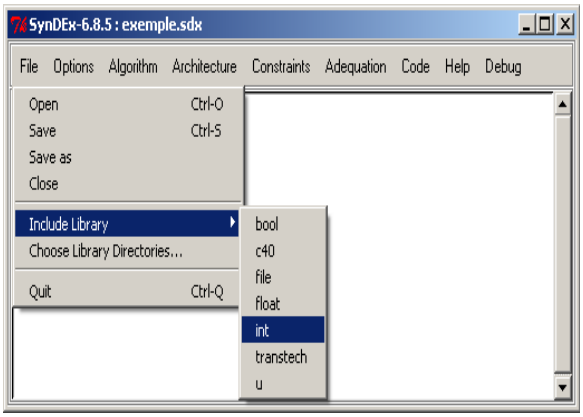


Figure 1.6 : Intégration de la bibliothèque int

1.2. Création de la définition de l'algorithme principale

- Dans la fenêtre principale on lance

  Menu: Algorithm / New Local Definition / Function / Dialog Window : on introduit le nom de l'algorithme principale "Main", OK.

- Dans la fenêtre de définition, pour définir l'algorithme comme principale

  Menu: Edit / Set As Main Definition;

- Dans la fenêtre de définition, pour créer une référence à la fonction de multiplication

  Menu: Edit / Create Reference / local, double click Mul / DialogWindow : "Mul " (figure 1.7)

- Pour créer une référence à la fonction puissance **"Puiss"**

  Menu: **Edit / Create Reference –** DialogWindow : click **local**, double click **Puiss /** DialogWindow : **"Puiss"** (Figure 1.7).

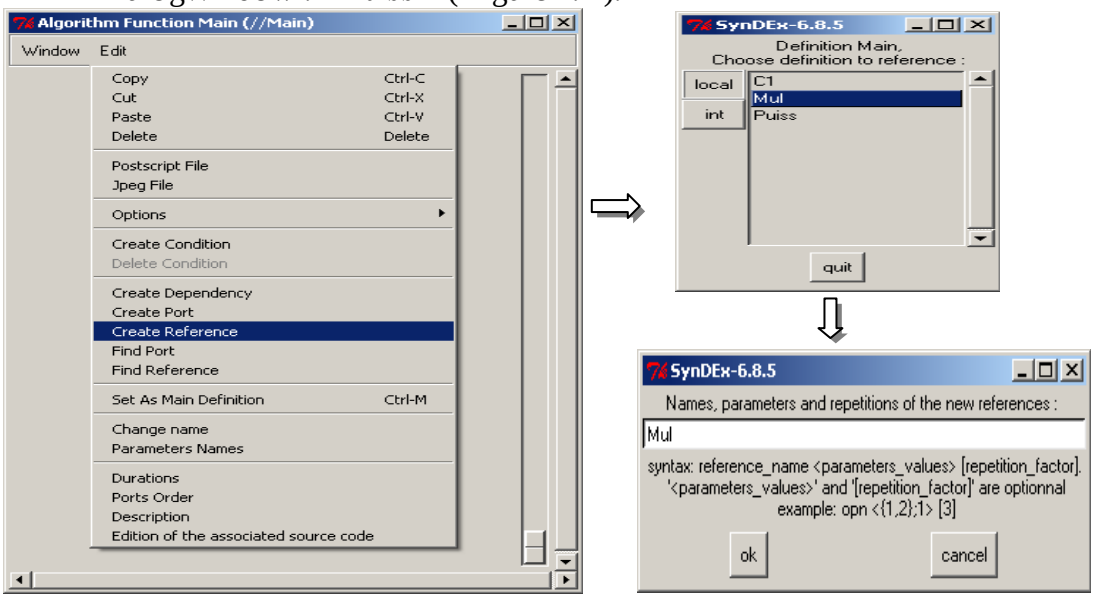


Figure 1.7 : Création de référence (Fonction de multiplication)

Pour les références A1 et C1, on va utiliser la bibliothèque « int » déjà intégrée :

- Pour créer une référence au capteur **"C1"**

  Menu: Edit / Create Reference – DialogWindow : click local, double click Input / DialogWindow : "C1".

- Pour créer une référence à l'actionnaire **"A1"**

  Menu: Edit / Create Reference – DialogWindow : click "Int", double click Output / DialogWindow : "A1" (Figure 1.8).

La fenêtre principale, après la création de toutes les références est présentée dans la figure1.9 (a).

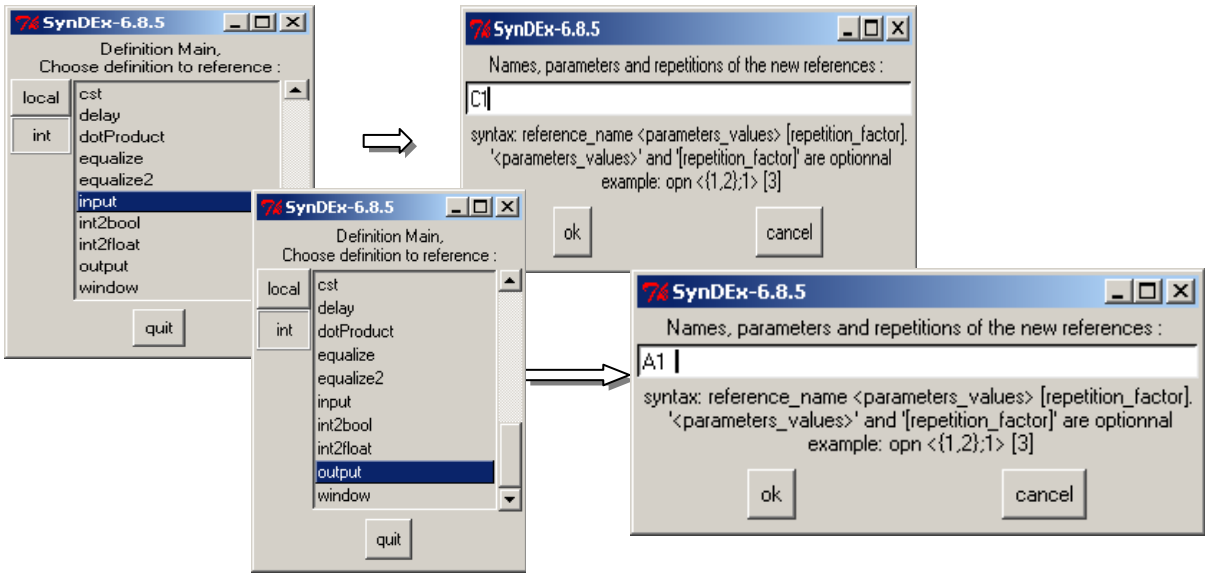
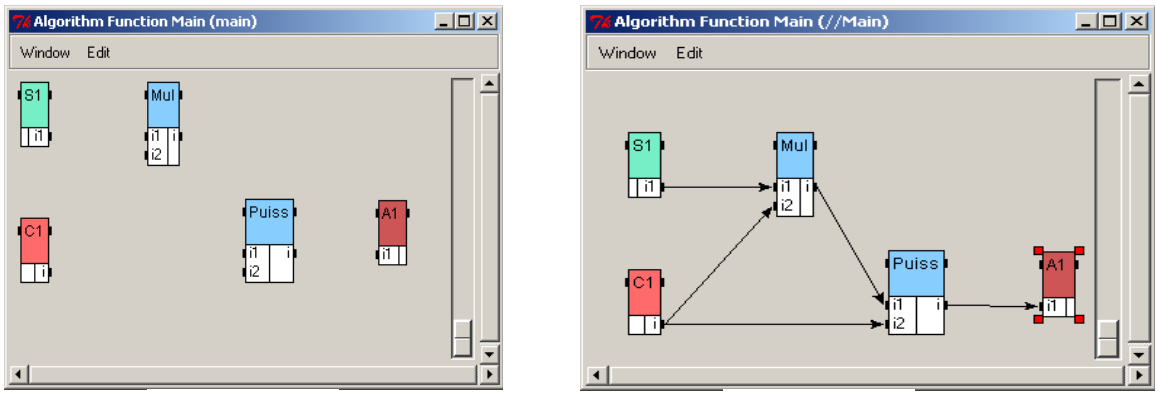


Figure 1. 8 : Création des références input et output à partir de la bibliothèque « int »



(a) **(b)**

Figure 1.9 : La fenêtre de l'algorithme principale

1.3. Création des arcs

Dans la fenêtre de l'algorithme principale, il faut ajouter des arcs entre les références. Les arcs sont faits avec le bouton du milieu de la souris. Ils représentent les liens de communication entre ces références. Ainsi, nous obtenons le graphe (b) de la figure 1.9.

2. Création de l'architecture avec deux opérateurs

2.1. création d'une définition d'un nouvel opérateur

- Dans la fenêtre principale :

  Menu: Architecture / New Local Operator Definition – DialogWindow : "Uinout", click OK /

- Dans la fenêtre de définition, pour associer à chaque opérateur, la durée d'exécution : on clique sur Modify durations – dans la fenêtre correspondante on introduit le temps d'exécution de chaque fonction (Figure 1.10) :

'' Mul = 2
Puiss = 4
S1 = 1
A1 = 3 ''

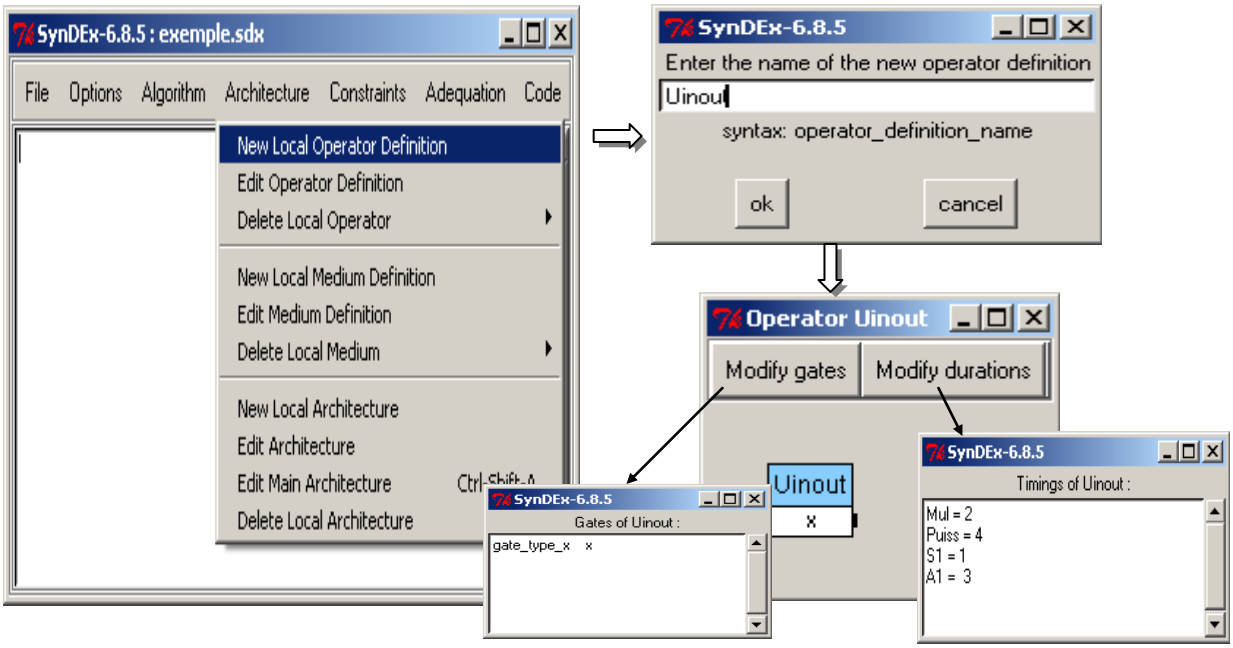


Figure 1.10 : Création d'un élément de calcul

Comme l'architecture est constituée de deux opérateurs, on crée un autre élément de calcul nommée « Proc1 », en suivant les mêmes étapes.

2.1.1. création d'une définition d'un nouvel médium

- Pour créer un médium entre les deux opérateurs de calcul on lance :

  Menu: Architecture / New Local Medium Definition – Dialog Window : on entre le nom du médium , dans notre « médium » (Figure 1.11) .

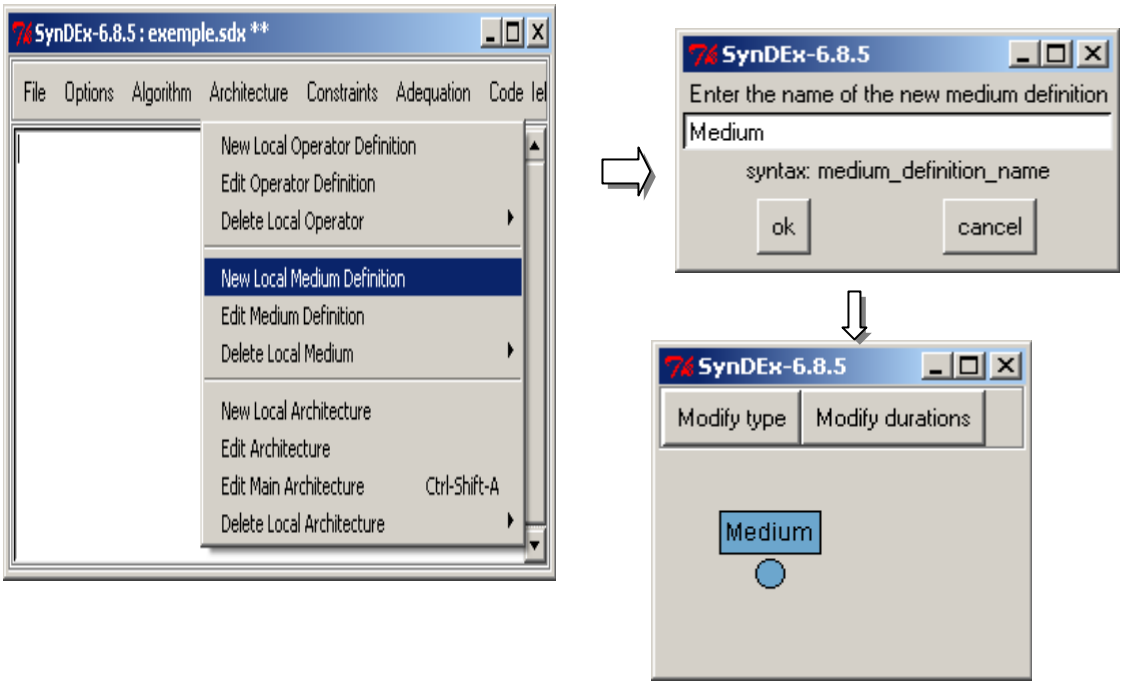


Figure 1.11 : Création d'un médium de communication

- Dans la fenêtre de définition (figure 1.12),
  - on lance Modify type / puis on choisit « SAM Point to Point»
  - On lance Modify durations – et dans la fenêtre on entre
    '' int = 2
    float = 2''

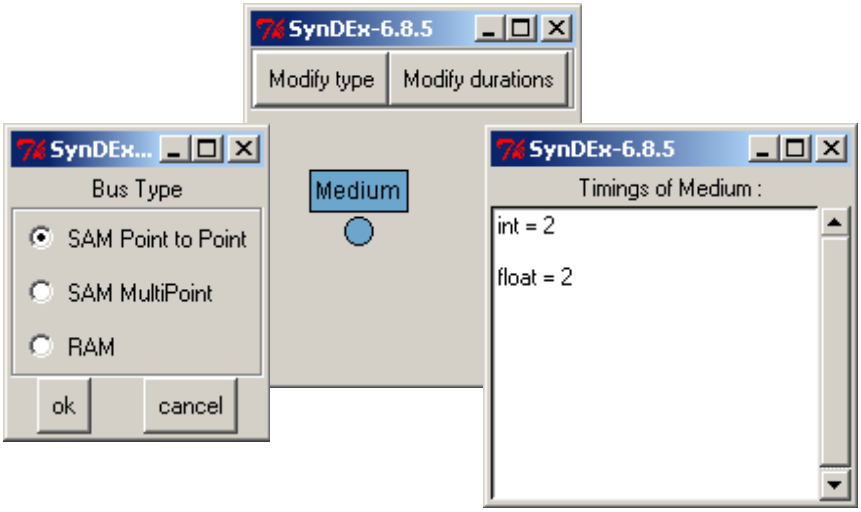


Figure 1.12 : La fenêtre de définition du médium de communication

2.2. Création de la définition de l'architecture principale

- Dans la fenêtre principale,

  Menu: Architecture / New Local Architecture / dans la fenêtre on entre le nom de l'architecture principale, dans notre cas "Main_Arch" (figure 1.13).

- Dans la fenêtre de définition, pour définir cette architecture comme 'architecture principale :

  Menu: Edit / Set As Main Architecture;

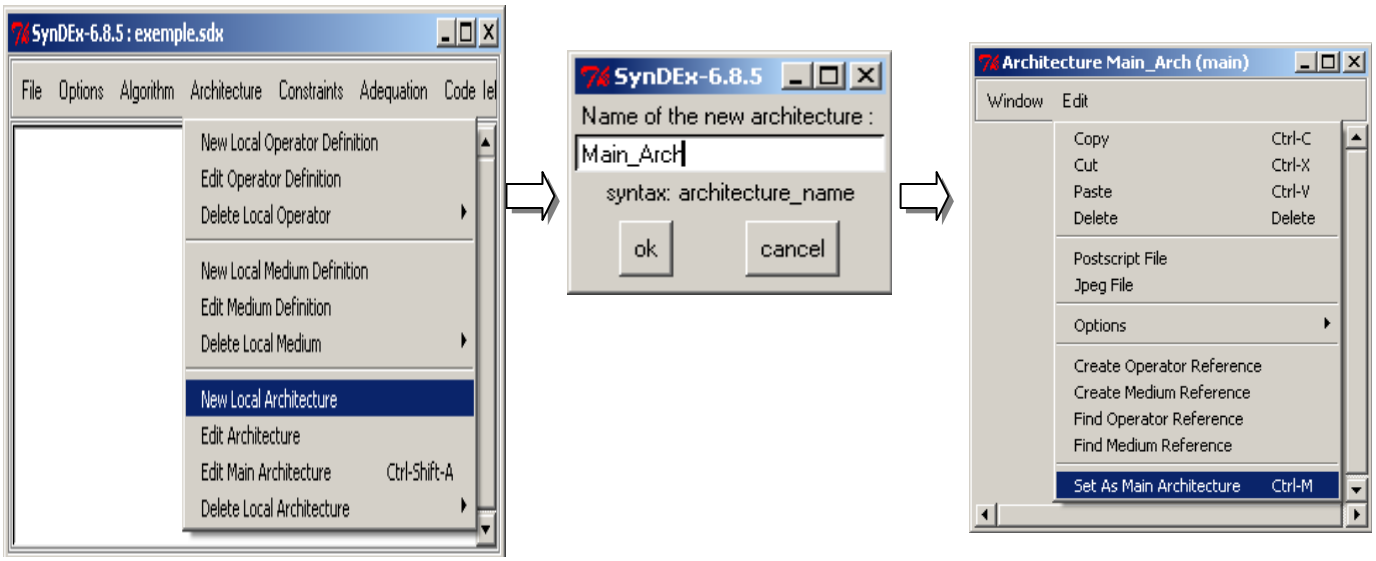


Figure 1.13 : création d'une architecture principale

- Dans la fenêtre principale de l'architecture, pour créer une référence à l'opérateur « Uinout »,

  Menu: Edit / Create Operator Reference / local - on sélectionne l'opérateur déjà créée "Uinout" – Dialog Window: "Main_Proc" (figure1.14)

De la même façon, on crée la définition au deuxième opérateur nommé « Proc1 »

- Dans la fenêtre principale de l'architecture, on crée une référence au médium de communication « medium »

  Menu: Edit / Create Operator Reference / local - on sélectionne l'opérateur déjà créée "Uinout" – Dialog Window: "Main_Proc" (figure1.15)

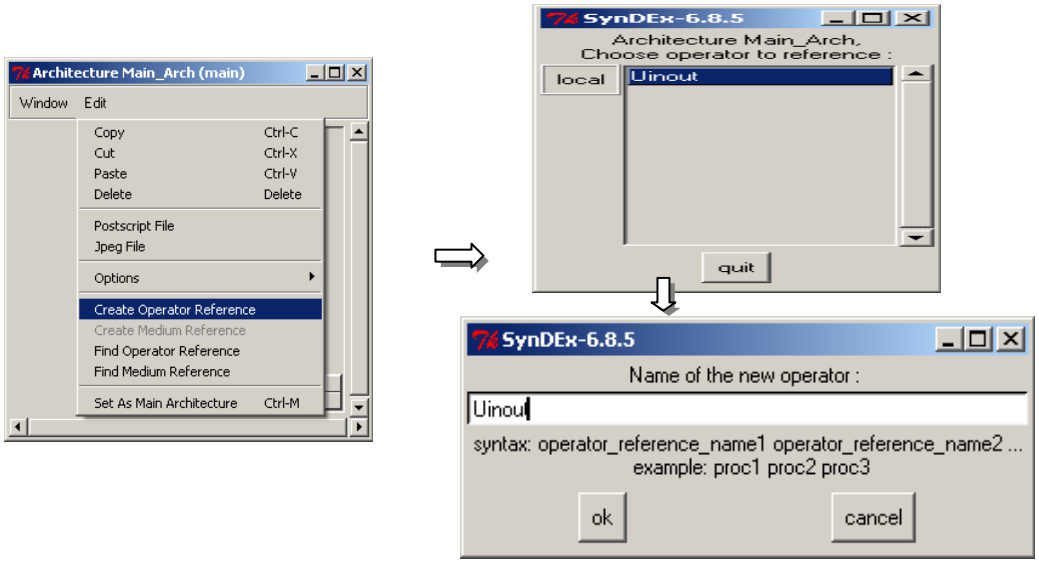


Figure 1.14 : Création d'une référence à l'opérateur de calcul

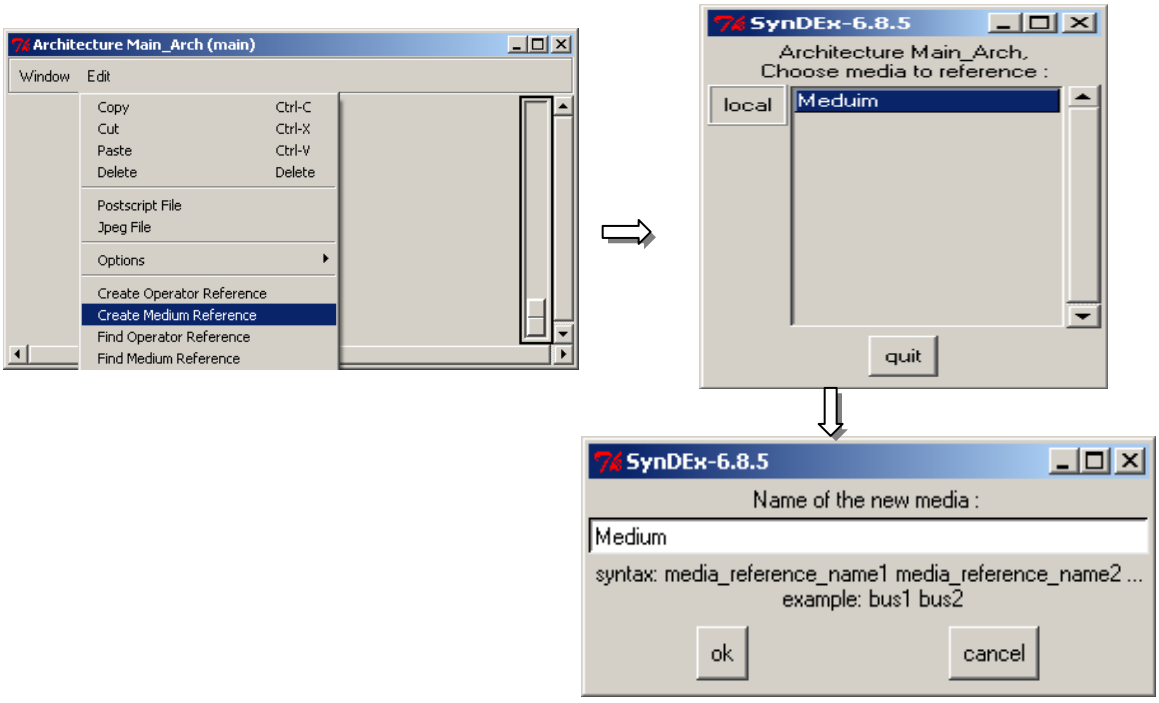


Figure 1.15 : création d'une référence au médium de communication

Après l'ajout des arcs entre les opérateurs de calcul et le médium de communication, l'architecture est présentée dans la figure 1.16.

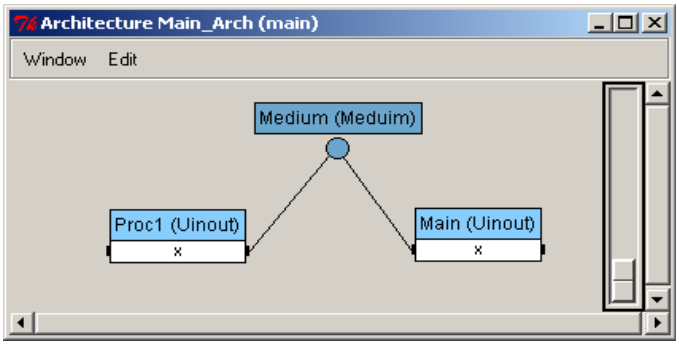


Figure 1.16 : architecture avec deux éléments de calcul

3. Performer une adéquation

Dans la fenêtre principale

- Menu: **Adequation / No Flatten** (L'hiérarchie est ignorée)
- Menu : **Adequation / Flatten medium** (L'hiérarchie est prise en considération)
- Menu: **Adequation / View Schedule**

Cela va ouvrir la fenêtre d'ordonnancement (Figure 1.17) dans laquelle on peut voir l'ordonnancement de l'algorithme sur l'architecture, et l'ordonnancement des différentes communications inter-opérateurs. Le temps de cycle d'exécution de toute l'application, le temps d'exécution de chaque opération et la communication inter-processeurs sont affichées dans la fenêtre principale de SynDEx.

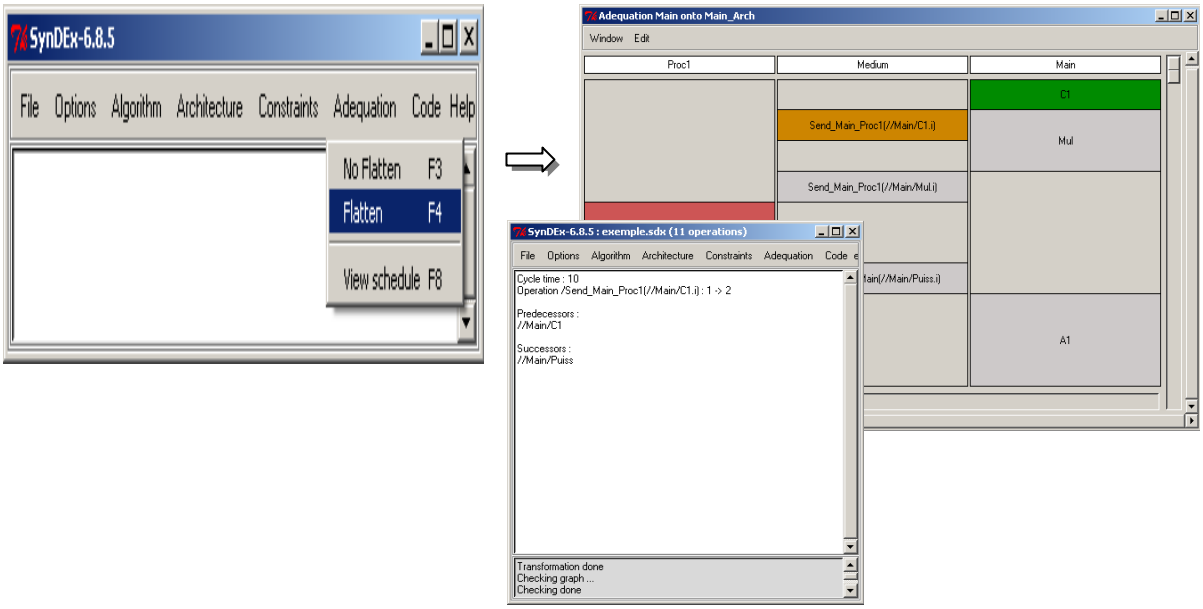


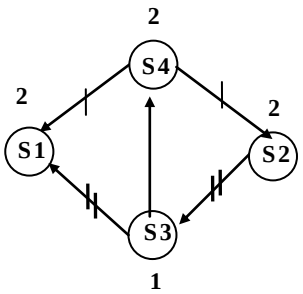
Figure 1.17 : La fenêtre graphique d'ordonnancement

Exemple de boucle :

For i =1 to n do

S1 *: a[i] = 2 * d[i-1]*
S2*: b[i] = 3 * d[i -1];*
S3 *: c [i] =a[i-2] + b[i -2];*
S4 *: d[i] = 2* C[i] ;*
End

(a)



Code transformé prologue
For i =2 to n -2 do

S4 *: d[i-1] = 2* C[i-1] ;*
S1 *: a[i] = 2 * d[i-1]*
S2*: b[i] = 3 * d[i -1];*
S3 *: c [i+2] =a[i] + b[i];*
End

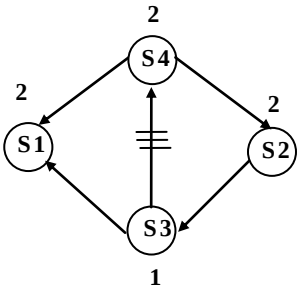


Figure 1.18 : Boucle resynchronisé avec le code prologue

Le graphe de la figure 1.18(a), est constitué de quatre nœuds de calcul, on suppose que l'opération de multiplication prend deux unités de temps, et l'opération d'addition est une seule unité de temps. Le temps d'exécution de cette boucle sous SynDEx sans resynchronisation est de 9 unités de temps par contre est de 8 unités de temps avec l'application de la resynchronisation.

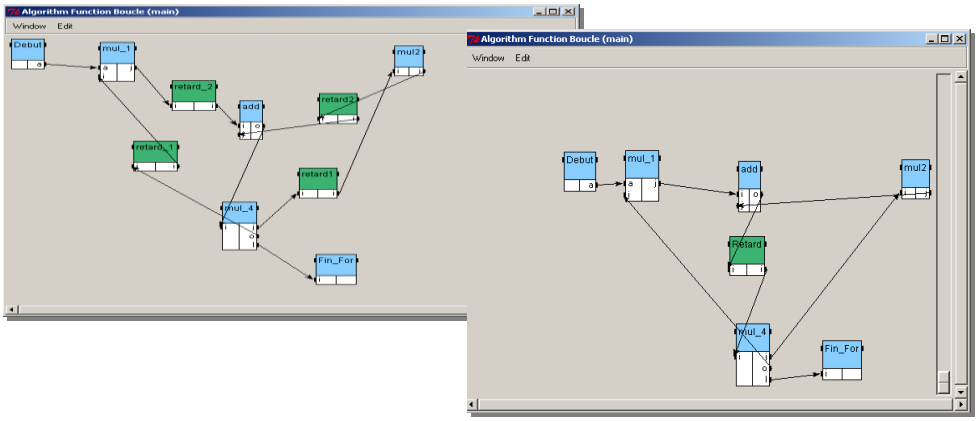


Figure 1.19 : Graphe flot de données resynchronisé

La figure 1.19 présente une capture d'écran du graphe de flot de données présenté sous SynDEx avec sa version resynchronisée.

Annexe B : Algorithme de resynchronisation en Ocaml

```
(** présentation du graphe***)
type noeud = ( string * string list * int * int *int ) list;;
type arcs = ( string * string * int ) list;;

(***  initialiser la valeur de R à zéro **)
let set r = (r =0);;
let rec setr noeud =
  match noeud with
  [] -> failwith "sommet vide"
  |(s, v, t, r, det)::reste  ->  if not( s = "") then
                                 set r
                                 else
                                 setr reste;;

(** Trouver la valeur de R d'un noeud "name"**)
let rec foundR noeud name  =
  match noeud with
  [] -> failwith "sommet vide"
  |(s, v, t, r, deta)::reste  ->  if ( s=name) then
                                  r
                                  else
                                    0+foundR reste name;;

(** Trouver la valeur de t d'un noeud "name"**)
let rec foundt noeud name  =
  match noeud with
  [] -> failwith "sommet vide"
  |(s, v, t, r, det)::reste  ->  if ( s= name) then
                                 t
                                 else
                                   0 + (foundt reste name);;

(** calculer la valeur w de tous les noeuds du graphe ***)

let rec calculw arcs noeud =
  match arcs with
  [] ->  failwith "sommet vide"
  |(a,b, w)::reste  ->
  if not(reste =[]) then (a,b, w+(foundR noeud a)-(foundR noeud b))::calculw reste noeud
  else (a,b, w+(foundR noeud a)-(foundR noeud b))::[];;

(**  liste des noeuds dont le poids w ==0 --**)
 let rec cp arcs =
  match arcs with
 []  -> failwith "sommet vide"
 |(a,b, w)::reste  ->  if  (w = 0 ) then
                       a::b::(cp reste)
                        else   if (reste=[]) then []
```

```
                    else cp reste ;;

(** liste des prédécesseurs d'un noeud 'nam_m' ****)
let rec list_pred arcs name_m =
match arcs with
[]  -> failwith " sommet vide"
|(a,b, w)::reste  -> if  (b = name_m) then
                  begin
                       if  not(reste = []) then a::( list_pred reste name_m)
                         else       a::[]
                       end
                   else
                  begin
                      if  not(reste = []) then  list_pred reste name_m
                     else []
                  end ;;

(***  Maj de la valeur deta de toute  la liste  ***)
let rec majdetv noeud sommet valeur =
match noeud with
 [] -> failwith " sommet null"
 |(s, v, t, r, det)::reste  ->  if ( s = sommet) then
                           begin
                             if  (reste =[]) then     (s, v, t, r, valeur)::[]
                             else    (s, v, t, r, valeur)::majdetv reste sommet valeur
                           end
                           else
                           begin
                              if  (reste =[]) then (s, v, t, r, det)::[]
                              else       (s, v, t, r, det)::majdetv reste sommet valeur
                           end ;;

(**  Trouver le maximum  de detV le temps cl***)
let max2 x y = if x>y then x else y;;
let rec maxdetv noeud =
  match noeud with
 []  ->  0
 |(s, v, t, r, det)::reste  ->  max2 det (maxdetv noeud);;

(** trouver la longueur d'une liste  **)
let rec size noeud =
match noeud with
 [] ->  0
 | a::q  ->  1 + (size q);;

(** Liste des prédécesseurs dont le poids de l'arcs w =0 **)

 let rec predwzero arcs head =

   match arcs with
    [] -> failwith "sommet vide"
```

```
                |(a,b,w)::reste  ->  if ((b = head) && (w = 0))  then
                                      if not(reste=[]) then   a::(predwzero reste head)
                                      else a::[]
                                    else
                                   if (reste=[]) then  []
                                   else predwzero reste head ;;

(** le max de la valeur du liste du prédécesseur **)
let rec maxt listpred noeud =
  match listpred with
  []  -> 0
  |fr::reste  -> max2 (foundt noeud fr) ( maxt reste noeud);;

(**** mettre à jour la valeur rv de toute la liste   ****)
let increm r = (r+1);;
let rec majrv noeud c =
  match noeud with

  [] -> failwith " sommet null"
  |(s, v, t, r, det)::reste  -> if (det >c) then
                          begin
                              if not (reste=[]) then   (s, v, t, increm r, det):: majrv reste c
                              else (s, v, t, increm r, det)::[]
                          end
       else
                     begin
                              if (reste =[]) then (s, v, t, r, det)::[]
                              else  (s, v,t, r, det):: majrv reste c
                           end;;
```

.

```
(*****************************************************************************)
(***                          L'algorithme principale                      ***)
(*****************************************************************************)
 let detv = 0 ;;

 let algo_principale noeud arcs  =
 for i=1 to ((List.length noeud)-1) do

  let arcs = (calculw arcs noeud )in
  let rec calcul noeud =
            match noeud with
            []  ->  failwith "sommet vide"
            |(s, t, v, r, det)::reste1  ->
              let sort = List.rev (cp arcs) in
                  let rec calculer_v sort =
                    match sort with

                []  -> failwith "aucun arcs avec poids null"
                      | head::reste2 -> if  ((list_pred arcs head) = [])  then

                        begin   (**** Misa jour du detv  ***)
                          let y = (foundt noeud head) in
                          (function (noeud,head) -> majdetv noeud head y)
                        end
                      else
                          begin      (**** Misa jour du detv  ***)
                [detv = ((foundt noeud head)+ (maxt (predwzero arcs head) noeud));
                           (function (noeud,head) ->  majdetv noeud head detv);
                      end;
                calculer_v reste2
                      in calculer_v [] ;

                        (**  calculer la longueur de cl(le maximum de detv) **)
                      majrv noeud (maxdetv noeud); (*** si detv > c , rv = rv +1 ***)
   calcul reste1
   in calcul noeud
 done;;
```