ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN GÉNIE ÉLECTRIQUE
M. Ing.

PAR
Maryem BENYOUSSEF

ÉLABORATION D'UNE MÉTHODOLOGIE DE CONCEPTION DES SYSTÈMES EMBARQUÉS BASÉE SUR LA TRANSFORMATION DU MODÈLE FONCTIONNEL DE HAUT NIVEAU VERS LE PROTOTYPE VIRTUEL

MONTRÉAL, LE 9 JUIN 2014

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Jean-François Boland, directeur de mémoire
Département génie électrique à l'École de technologie supérieure

Mme Gabriela Nicolescu, codirecteur de mémoire
Département génie logiciel à l'École Polytechnique

Claude Thibeault, président du jury
Département génie électrique à l'École de technologie supérieure

M. Guy Bois, examinateur externe
École Polytechnique

ELLE A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 13 MAI 2014

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

Je tiens à remercier sincèrement et sans compliment mon directeur de recherche Jean-François Boland et ma co-directrice Gabriela Nicolescus pour l'aide qu'ils m'ont apporté tout au long de la période de ma maîtrise, pour leurs remarques pertinentes, leurs renseignements précieux, leurs conseils, leurs encouragements et surtout leurs disponibilité. Ils m'ont été le meilleur support pour mener à bien un tel projet afin qu'il soit fructueux et profitable.

J'exprime aussi toute ma reconnaissance et ma gratitude à notre chef de projet le professeur Guy Bois, pour son soutien et sa collaboration.

Ensuite, je tiens à remercier l'École de technologie supérieure, le Consortium de recherche et d'innovation en aérospatiale du Québec (CRIAQ), le programme de stage du réseau de chercheurs en Mathématiques des technologies de l'information et des systèmes complexes (MITACS), ainsi que les partenaires industriels du projet AVIO 509, et tous les membres d'équipe du projet.

Finalement, j'aimerais remercier mes chers parents, mon cher mari, ceux qui n'ont jamais cessé de m'encourager et de me conseiller. Je remercie mes frères et sœurs, en témoignage de l'amour et de l'affection qui nous lient et toute ma famille.

Enfin je remercie les membres de jury de nous honorer de leur présence et de porter leurs jugements sur ce modeste travail.

ÉLABORATION D'UNE MÉTHODOLOGIE DE CONCEPTION DES SYSTÈMES EMBARQUÉS BASÉE SUR LA TRANSFORMATION DU MODÈLE FONCTIONNEL DE HAUT NIVEAU VERS LE PROTOTYPE VIRTUEL

Maryem BENYOUSSEF

RÉSUMÉ

La croissance rapide des progrès technologiques combinée aux demandes exigeantes de l'industrie entraîne une augmentation de la complexité des systèmes embarqués. Cette complexité impose plusieurs contraintes et critères à respecter pour produire des systèmes compétitifs et robustes. Aussi, les méthodologies de conception ont grandement évolué au cours des dernières années pour encadrer le développement de ces systèmes complexes et assurer leur conformité aux requis initiaux.

C'est ainsi que de nouvelles approches basées sur des modèles sont apparues, pour pallier à ces difficultés et maîtriser le niveau de complexité. Mais souvent ces approches basées sur des modèles traitent les aspects fonctionnels et logiciels du système sans prendre en considération les aspects d'exécution sur de réelles plateformes matérielles.

Les travaux développés dans le cadre de ce projet de recherche visent à mettre en œuvre une nouvelle méthodologie de conception des systèmes embarqués. Cette méthodologie permet d'établir un lien entre le niveau fonctionnel des modèles et la plateforme d'exécution matérielle de l'application en question. L'approche développée est basée sur l'utilisation du langage de modélisation AADL pour décrire le comportement logiciel du système embarqué à un haut niveau d'abstraction. Ensuite, une chaîne de transformation automatique convertit le modèle AADL vers un modèle SystemC. Finalement, l'environnement Space Studio est utilisé pour construire un prototype virtuel de la plateforme. Cet environnement permet l'exécution des aspects fonctionnels du système sur des ressources matérielles. Les performances du système peuvent ainsi être validées et raffinées en se basant sur une exploration architecturale de la plateforme matérielle.

Une application d'imagerie a été exploitée en tant qu'étude de cas pour expérimenter ce flot. Il s'agit d'une application de décodage vidéo MJPEG (Motion JPEG). Durant l'expérimentation, un modèle AADL de l'application MJPEG a été développé décrivant son comportement fonctionnel. Ensuite, la chaîne de transformation utilisée traduit automatiquement le modèle AADL en un modèle SystemC. Le modèle SystemC a servi comme élément de base représentant l'aspect logiciel dans l'environnement de prototypage virtuel et de conception conjointe Space Studio. L'outil Space Studio s'est montré utile en permettant la création rapide d'un prototype de plateforme matérielle d'exécution, le partitionnement des fonctions logicielles sur des ressources matérielles et la validation et raffinement des performances du système. Les résultats d'expérimentation obtenus furent concluants. La vitesse d'exécution a été visiblement augmentée et le temps pris pour achever

la simulation du système a été réduit de 81.86%. En ce qui concerne le taux d'occupation du processeur quant à lui a considérablement diminué, ce qui pourra ainsi diminuer le taux de puissance consommée par les ressources matérielles. Ainsi le traitement de données par unité de temps s'est amélioré 12 fois de plus après le raffinement porté sur l'assignement des fonctions logicielles sur la plateforme matérielle.

Dans le cadre de ce projet, un article scientifique a été publié (Benyoussef et al., Février 2014) à la conférence ERTS 2014 (Embedded Real Time Software and Systems). Ce travail présente le contexte et la problématique liée aux méthodologies basées sur des modèles, la nouvelle approche de modélisation développée ainsi qu'une preuve de concept avec une application de décodage MJPEG.

Mots clés: Ingénierie basée sur des modèles, AADL, SystemC, transformation, prototypage virtuel, exploration architecturale.

IMPLEMENTATION OF A NEW EMBEDDEDS SYSTEMS DESIGN APPROACH BASED ON TRANSFORMATION OF A FUNCTIONNAL HIGH LEVEL MODEL INTO VIRTUAL PROTOTYPE

Maryem BENYOUSSEF

ABSTRACT

The complexity of embedded systems continues to rise rapidly due to technological advances and industrial demand for electronics powerful enough to implement increasingly sophisticated software applications. High-quality designs are becoming increasingly necessary in order to meet the demand for embedded systems that are competitive in terms of reliability, safety and robustness.

In order to meet the challenge of improving embedded system design and managing the increasing complexity, engineers are turning to model-based engineering. But often validation of traditional model-based design generally considers functional and timing requirements only, and ignores target platform execution constraints.

In this paper a new modeling framework is proposed. It aims to simplify the design process by offering a trade-off between design abstraction and accuracy of results through performance analysis. High and low levels of abstraction are thus bridged in the same design flow so that architectural exploration and refinement can be performed at different levels.
The proposed approach combine the advantages of model-based engineering methodologies using the AADL language and virtual prototyping based on virtual platform environments Space Studio. Our methodology uses a tool chain transformation to bridge the gap between high-level models and the virtual execution platform to allow more accurate design space exploration.

The modeling language AADL is used to describe the system behavior of the embedded software at a high abstraction level, while the virtual platform environment is used to execute software system tasks on hardware resources, and allow performances validation and refinement based on an architectural exploration of the hardware platform.
The proposed framework is validated using an MJPEG video decoder application. Experimental results show how virtual prototyping allows the system architect to fine-tune performances. Many candidate architectures have been explored to find the better solution. So the execution speed has been visibly increased and the time taken to complete the simulation of the system was reduced by 81.86 %. Moreover the processor utilization rate has decreased significantly, which may thus reduce the rate of power consumption by the hardware resources. Finally system performance has improved in term of processing speed 12 times comparing to initial candidate architecture.

Within the framework of this project a paper was published for ERTS 2014 conference (Benyoussef et al., Février 2014) This work presents the research context and model based design problematic, the modeling methodology developed and also our experiment with an MJPEG video decoder application.

Keywords: Model Based Engineering, AADL, SystemC, transformation, virtual prototyping, architectural exploration.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

AADL	Architecture Analysis Design Language
AHB	Advanced High-performance Bus
AMBA	Advanced Microcontroller Bus Architecture
AMP	Asymmetric Multiprocessing
APB	Advanced Peripheral Bus
ATL	ATLAS Language
AXI	Advanced eXtensible Interface
BIP	Behavior, Interaction, Priority
CA	Cycle Accurate
CRIAQ	Consortium de Recherche et d'Innovation en Aérospatiale au Québec
DSL	Domain Specific Language
EDA	Electronic Design Automation
GALS	Globaly Asynchronos Localy synchronous
INCOSE	International Council on Systems Engineering
IRQ	Interrupt Request
MBE	Model Based Engineering
OSATE	Open-Source AADL Tool Environment
OSCI	Open SystemC Initiative
PIC	Programmable Interrupt Controller
RTL	Register Transfer Level
SDL	Space Direct Link
SME	Signal Meta under Eclipse
SPES	Software Platform Embedded Systems
TF	Timed Functional
TLM	Transaction-Level Modeling
UTF	Untimed Functional

INTRODUCTION

Les systèmes embarqués sont des systèmes électroniques incluant du logiciel et du matériel, dédiés à exécuter des fonctionnalités et des applications bien spécifiques. Ils sont d'un grand intérêt dans divers domaines qui font appel aux nouvelles technologies notamment en transport avionique et automobile, astronomie, télécommunication, électroménagers, multimédias, santé, etc.

En raison de la rapidité des évolutions technologiques, la complexité des systèmes embarqués ne cesse de croitre (Kopetz, 2008). Donc, il y a un besoin impératif de produire des systèmes embarqués compétitifs en termes de fiabilité, robustesse, rapidité de conception, rapidité de validation, faible coût et temps de mise en marché. Ces exigences industrielles (Henzinger et Sifakis, 2006), (Koopman, 1996) ont entrainé des difficultés au niveau des méthodologies de conception, en particulier aux aspects liés à la validation, l'optimisation et l'analyse des performances.

Dans le but de maîtriser cette difficulté, et répondre aux besoins de l'industrie, des approches de conception basées par des modèles (Model-Based Engineering, MBE) sont de plus en plus utilisées (Schmidt, 2006) (Wilson et Mantooth, 2013). Ces approches visent à simplifier le processus de conception en se basant sur la création de modèles pour décrire l'architecture, le comportement ainsi que les requis auxquels doit répondre un système. Ces approches de modélisation consistent à élaborer des modèles à un haut niveau d'abstraction en impliquant juste les éléments essentiels au contexte de conception. Cette abstraction de détails est adoptée dans le but de faciliter la représentation des systèmes conçus et accélérer les phases de développement.

De plus, les méthodologies de conception de type MBE offrent la possibilité de mener une étape de vérification et de validation hâtive pour s'assurer de la conformité aux spécifications, analyser les performances du système et détecter les erreurs aux premières phases de design.

Souvent, les techniques de validation utilisées dans une approche MBE sont liées aux aspects logiciels et fonctionnels du système sans tenir compte des considérations d'exécution sur de réelles plateformes matérielles (Fleurey, Steel et Baudry, 2004) (Hill, Tambe et Gokhale, 2007). Ceci peut engendrer des erreurs majeures lors de l'implémentation et ralentir la chaîne de production. Cet écart présent entre le modèle haut niveau du système et la plateforme d'exécution reste une lacune importante dans le processus de conception.

Ainsi, la difficulté de l'étape de partitionnement logiciel/matériel peut soulever des problèmes de retard des phases de design par rapport au cycle de vie d'un produit (Wolf, 2003). En effet le partitionnement logiciel/matériel nécessite plusieurs itérations de codage et évaluation de chaque solution, donc il a fallu réduire ce temps de développement et prendre des décisions rapides pour augmenter la compétitivité.

C'est dans ce cadre que s'inscrivent les travaux menés dans ce projet de recherche, qui visent à combler cette lacune en établissant une liaison entre le haut niveau d'abstraction du modèle système et l'environnement d'exécution sur des ressources matérielles concrètes. Ce lien permettra par la suite d'évaluer et d'améliorer les performances du système en se basant sur une exploration rapide de l'espace de conception.

Problématique

Dans ce contexte la problématique de recherche repose sur la question suivante : ¨Est-il possible d'élaborer une méthodologie de conception basée sur des modèles capable de prendre en compte les variantes de la plateforme matérielle d'exécution pour des fins de validation et d'amélioration des performances du système ? ¨

Le problème est motivé par la complexité croissante des systèmes embarqués à tous les niveaux du processus de développement (conception, validation, production). Ainsi, par le besoin grandissant en matière de méthodologie de développement basée sur des modèles capable de couvrir les processus d'évaluation de l'aspect logiciel en dépendance avec l'aspect matériel mis en jeu.

Dans le cadre du projet de recherche CRIAQ (Consortium de recherche et d'innovation en aérospatiale au Québec) AVIO-509 les partenaires industriels, CMC Électronique et CAE, ont exprimé leur intérêt pour développer un flot de conception permettant de gérer la complexité grandissante des systèmes embarqués et ainsi concevoir des produits compétitifs dans les temps et les budgets alloués. Ainsi, ce mémoire présente une méthodologie de modélisation fiable qui répond exactement à ces exigences.

Objectifs

Pour répondre à la problématique, les objectifs suivants ont été définis :

1- Le premier objectif est en premier lieu d'explorer les récents travaux de recherche dans le domaine de la conception des systèmes embarqués. Les lectures effectuées ont principalement porté sur l'étude des flots de conception basée sur des modèles d'une part, et sur les techniques de validation et d'analyse de performances d'un modèle de haut niveau d'abstraction d'autre part. Cet objectif nous a permis de mieux cerner la problématique de recherche et caractériser les limitations des travaux effectués pour bien orienter les champs d'action;

2- Le second objectif vise à proposer un flot de modélisation permettant la validation du système conçu à deux niveaux d'abstraction: fonctionnel et matériel;

3- Le troisième objectif consiste à développer un cas d'étude d'un système de traitement d'image pour mettre en évidence l'intérêt et l'apport de l'approche proposée ainsi que ses limitations.

Contributions :

Dans le but d'atteindre les objectifs mentionnés ci-dessus, les travaux de recherches effectués ont abouti aux contributions suivantes :

- élargir la base de connaissance sur les approches MBE (Model-Based Engineering) et développer l'expertise requise dans le domaine de la conception, modélisation et validation des systèmes embarqués au sein de notre équipe de recherche;

- prendre en main des outils de développement et de modélisation ainsi que les environnements de simulation utilisés pour mettre en évidence la méthodologie proposée dans ce projet. Parmi ces outils citons par exemple OSATE2, Space Studio, Ocarina, AADL2SYSTEMC, Simics, VisualSim, Cheddar;

- élaborer un flot de modélisation novateur à un haut niveau d'abstraction qui prend en considération les contraintes liées à la plateforme d'exécution matérielle. Le flot proposé a été réalisé en utilisant le langage de description d'architecture AADL (Architecture & Analysis Description Language) (Feiler, Gluch et Hudak, 2006) et une chaîne de transformation vers le langage SystemC;

- procéder à une étape d'exploration architecturale sur la base des résultats d'analyse de performances du système tout en raffinant itérativement le modèle de haut niveau pour optimiser l'utilisation des ressources de la plateforme d'exécution;

- faciliter le processus de partitionnement matériel-logiciel par l'utilisation d'un outil de co-conception;

- évoquer les limitations liées au flot proposé pour but d'amélioration et perfectionnement des futurs travaux dans le cadre de ce projet;

- publier un article scientifique (Benyoussef et al., Février 2014) à la conférence ERTS 2014 (Embedded Real Time Software and Systems) résumant l'ensemble des travaux développés dans le cadre de ce projet de recherche.

Organisation du mémoire :

Le corps de ce mémoire est structuré comme suit :

- le chapitre 2 présente la revue de littérature liée aux éléments de la problématique;
- le chapitre 3 présente l'approche de modélisation proposée;
- le chapitre 4 illustre le cas d'étude utilisé dans le contexte du flot proposé;
- le chapitre 5 englobe l'ensemble des résultats obtenus.

Finalement nous concluons avec une synthèse générale des travaux effectués dans le cadre de ce projet et les perspectives prometteuses envisagés pour les futures recherches.

CHAPITRE 1

REVUE DE LITTÉRATURE

Ce chapitre présente le principe d'ingénierie basée sur des modèles, ainsi qu'une revue de la littérature sur les différentes approches de modélisation et les langages associés. Finalement un bilan et quelques limitations seront présentés pour bien définir le contexte et le champ d'action des travaux développés dans le cadre de ce projet.

1.1 L'ingénierie basée sur les modèles

L'ingénierie basée sur les modèles (MBE) est une discipline émergente qui connait une forte croissance pour le développement des systèmes embarqués. Compte tenu de la complexité croissante et des défis technologiques associés au développement des systèmes embarqués modernes, plusieurs études et projets ont démontré que la MBE présente une perspective de grande envergure et d'efficacité pour adresser différents compromis et difficultés.

Deux études intéressantes (INCOSE et SPES) portant sur le thème de la MBE seront présentées par la suite, pour montrer la pertinence et l'efficacité des approches basées sur des modèles.

INCOSE (International Council on Systems Engineering)

INCOSE (Friedenthal, Griego et Sampson, 2007), un organisme international créé en 1991, comporte plusieurs équipes de groupes de recherche et des industries. Leur objectif est de mener une étude à long terme des dernières tendances et avancements dans le domaine de l'ingénierie des systèmes.

Suite à plusieurs travaux de développement et études de recherches, l'organisme INCOSE a fondé en 2007, une vision à long terme de l'ingénierie des systèmes complexes. Cette vision prévoit une croissance de l'utilisation des approches MBE à toutes les phases de cycle de

développement d'un système, ainsi qu'une évolution de la maturité de ces approches à long terme.

Cette évolution est à l'origine des multiples avantages offerts par MBE. En effet, la simplicité de représentation d'un système par des modèles réduit énormément la complexité de conception et facilite la capture, l'analyse, le partage et la gestion des informations associés aux spécifications d'un système. Ce qui permet :

- d'améliorer la communication entre des entités interdisciplinaires (clients, programme de gestion, ingénierie système, développements matériel et logiciel...);
- de gérer la complexité des systèmes tout en fournissant un modèle système global vu sous différents angles selon la perspective de conception, et par la suite d'analyser l'impact de chaque changement sur l'ensemble;
- d'améliorer la qualité des produits : tout en offrant la possibilité d'évaluer la consistance, l'exactitude et la cohérence du système dans les premières phases de design;
- de réduire les cycles de développement et les coûts de maintenance pour toute modification apportée au système tout en réutilisant les modèles déjà établis.

Donc l'organisme INCOSE présume que le futur des ingénieries système se base essentiellement sur les approches MBE à tous les niveaux de développement d'un produit.

SPES 2020 (Software Platform Embedded Systems)

Le projet SPES 2020 (Pohl et al., 2012), s'intéresse à élaborer un cadre méthodologique et technologique de développement basé sur les modèles et répondre à des défis spécifiques de développement industriel pour les futurs systèmes embarqués (réseautage, plateformes matérielles, architectures logicielles, intégration, spécification, implémentation, certification…). La
Figure 1.1 présente la structure principale du projet SPES 2020.

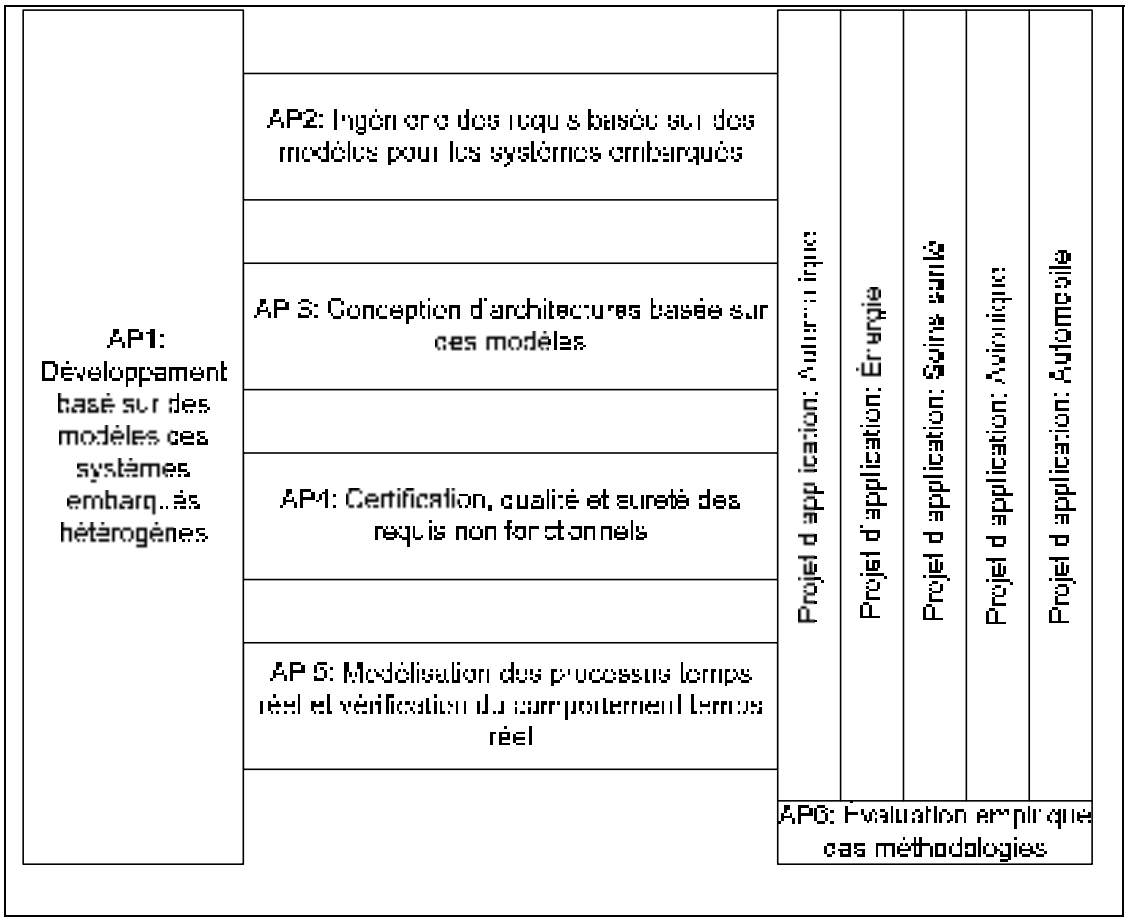


Figure 1.1 Structure principale du projet SPES 2020
Adaptée de Pohl et al. (2012)

Elle est fondée sur le développement des technologies de modélisation pour cinq domaines d'applications différents; à savoir l'automobile, l'avionique, l'énergie, l'automatique et la santé. Ensuite, le projet central est divisé lui-même en six sous parties traitant chacune un aspect bien précis :

1ère sous-partie : concerne l'intégration et le développement des processus et outils basés sur des modèles pour les systèmes embarqués hétérogènes;

2e sous-partie : concerne le développement de l'ingénierie des requis basée sur des modèles pour les systèmes embarqués;

3e sous-partie : concerne la conception d'architecture basée sur des modèles;

4e sous-partie : concerne l'approbation de sécurité, certification et assurance de qualité des requis non fonctionnels;
5e sous-partie : concerne la modélisation des processus parallèle et vérification du comportement en temps réel;
6e sous-partie : concerne l'évaluation empirique des différentes méthodologies de conception.

Donc le projet SPES couvre tous les aspects pertinents du cycle de vie d'un système embarqué. Le projet SPES vise à fonder une expertise de développement dans les différents domaines de l'industrie. Il se repose sur l'utilisation des approches de modélisation pour adresser les défis de développement en termes de conception, validation, évaluation, certification et sécurité.

D'après les deux études présentées, les approches MBE présentent un intérêt croissant pour le domaine de l'ingénierie des systèmes (plus spécifiquement les systèmes embarqués) et suscitent une attention particulière pour les futures technologies. Et pour tirer profit de ces opportunités offertes par MBE, l'orientation du cadre méthodologique de notre projet de recherche s'est dirigée vers l'utilisation des modèles. Donc on s'est inspiré de ces études pour fonder le processus de conception de système embarqué de ce projet sur une approche de modélisation.

Pour bien préciser les éléments de l'approche proposée, il était nécessaire d'explorer les différentes contributions de la littérature en tout ce qui concerne le domaine de la modélisation.

Donc les deux sections suivantes seront consacrées à la présentation des différents langages de modélisation utilisés et un survol sur quelques flots de conception basés sur des modèles.

1.2 Langages de modélisation

Les langages de modélisation servent à représenter un modèle selon des normes et règles de sémantiques spécifiques à chaque type de langage. Les prochaines sections présentent un survol sur les différents langages de modélisation existants et leur contexte d'utilisation.

1.2.1 ADL

Les langages de description d'architecture (ADL: Architecture Description Language) permettent de décrire précisément les applications et les architectures logicielles et matérielles d'un système donné. Les ADL sont en général conçus pour répondre à un certain nombre de fonctionnalités incluant:

- définir l'architecture d'un système comme composition d'éléments de base ainsi que les interactions entre eux;
- supporter une phase d'analyse architecturale qui se place entre l'analyse des besoins et la conception (supporter à la fois une approche descendante et ascendante);
- fournir une sémantique bien définie au plan architectural;
- aider à l'implantation du système, par exemple en permettant la génération automatique du code et des interactions entre les composants du système.

Pour présenter le ADL plus en détail, voici ses principaux éléments de base :

Composant : représente une entité de traitement ou de stockage répondant à un besoin de description fonctionnel, applicatif ou matériel;

Interface : comprend un ensemble de points d'interaction d'un composant ou d'un connecteur avec son environnement;

Connecteur : permet d'illustrer les interactions entre les différents composants ainsi que les règles auxquelles sont soumises ces interactions;

Configuration : comprend un graphe de composants et de connecteurs spécifiant l'architecture selon un certain point de vue;

Propriété: fournit une information sur un identifiant de modèle;

Flux: permet des analyses temporelles, de fiabilité, de propagation d'erreur, de qualité de service;
Mode : correspond à l'état opérationnel du système;
Paquet : permet d'organiser les descriptions en introduisant des espaces de nom.

Le langage de modélisation AADL (Architecture Analysis and Design Language) (Feiler et Gluch, 2012) est considéré comme l'un des ADL le plus utilisé dans le domaine de l'avionique; c'est un langage d'analyse et de conception d'architecture basé sur l'ADL Meta-H (Déplanche et Faucou, 2005) , développé dans les années 90 par Honeywell.

Ce langage permet la conception et la description de l'architecture temps-réel en considérant simultanément tous les aspects logiciels et matériels associés. C'est un langage textuel et graphique permettant de manipuler les flux de contrôle et de données et de décrire les aspects non fonctionnels importants liés à ces systèmes (exigences temporelles, fautes, propriétés de sûreté et de certification…).

1.2.2 SystemC

SystemC est une librairie de modélisation système (Liao et al., 2002). L'initiative OSCI (Open SystemC Initiative) a été créée en 2001 pour développer et affiner les spécifications du SystemC en partenariat avec plusieurs universités et entreprises. En décembre 2005 SystemC a été standardisé auprès de l'IEEE sous le nom de IEEE 1666™-2005.

SystemC comprend une bibliothèque de classes C++ étendue permettant la modélisation au niveau système. Cette extension a pour objectif de préciser la description de certains aspects qui ne sont pas supportés par le standard C++. SystemC peut décrire le matériel, le logiciel, les fonctionnalités concurrentes d'une application, ainsi que l'interface de communication à plusieurs niveaux d'abstraction et avec le même langage. SystemC supporte aussi la notion du temps dans la spécification d'un système et permet d'élaborer une simulation du comportement à partir d'un exécutable binaire.

La Figure 1.2 illustre l'architecture et les éléments de la bibliothèque SystemC.

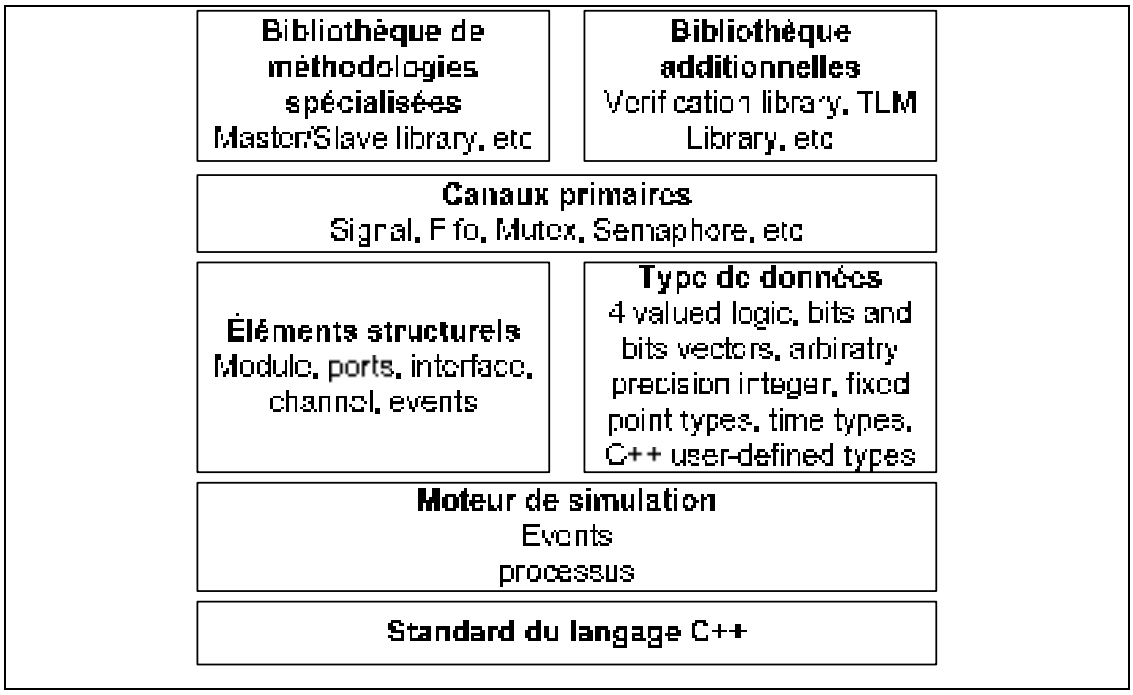


Figure 1.2 Bibliothèque des composants SystemC
Adaptée de Tech (2014)

D'après la Figure 1.2, la bibliothèque SystemC comprend des types de données adaptées pour représenter le matériel (modules, ports, signaux, fifo), d'autres pour le logiciel (sémaphores, mutex) et certains pour les deux (canaux, événements, interfaces ...). Les autres éléments servent soit à représenter la structure d'un système ou bien à décrire la communication entre processus et modules (événements, canaux).

Ce langage offre la possibilité de modéliser un système avec différents degrés de précision appelés niveaux d'abstraction. Ceci peut guider l'ingénieur dans son processus de conception et raffinement de son modèle. Ces niveaux d'abstraction sont :

TLM : modèle d'une représentation abstraite décrivant les interfaces, les fonctions et les aspects temporels des communications dans un système;

UTF : modèle qui décrit la fonctionnalité à un haut niveau sans mention de la notion de temps ou distinction entre le matériel et logiciel;

TF : modèle qui décrit la fonctionnalité avec insertion des délais (wait);

CA : modèle où la fonctionnalité et la communication sont décrits à chaque coup d'horloge;
RTL : modèle qui décrit la structure interne reflétant les registres et la logique combinatoire du système.

1.2.3 DSL

Les langages spécifiques DSL (Domain Specific Language) sont des langages conçus pour répondre à des besoins particuliers de modélisation et d'analyses (Hill, 2011). Parmi ces langages spécifiques le Fractal, Ptolemy et BIP. Cette section présente l'exemple du langage BIP.

Le langage BIP (Behavior Interaction Priority) est basé sur une théorie permettant le développement incrémental de systèmes à base de composants hétérogènes. Trois sources différentes d'hétérogénéité sont considérées : l'interaction, l'exécution et l'abstraction. Les principaux éléments de base du langage BIP sont :

Atome: pour préciser le comportement avec une interface composée de ports;
Connecteur: pour spécifier les coordinations entre les ports des composants et les actions;
Priorité: pour restreindre les interactions possibles basées sur des conditions, en fonction de l'état des composants intégrés;
Composite: pour spécifier la hiérarchie du système à partir des atomes avec l'utilisation des connecteurs des priorités;
Modèle: pour spécifier le plus haut niveau du système.

En BIP les composants sont la superposition de trois couches: comportement, interaction et priorité. Le modèle comportemental étend la théorie des automates temporisés avec l'expression de l'urgence. L'interaction concerne la synchronisation entre les composants, avec un éventuel transfert de données. La priorité est un mécanisme élémentaire de contrôle pour la résolution de conflits et permet la modélisation et la composition des politiques d'ordonnancement. La chaîne d'outils associée au langage BIP permet la génération d'une

implémentation exécutable sur une machine virtuelle supportant l'exécution du modèle BIP, ainsi que des vérifications formelles du modèle.

1.2.4 Langages synchrones

Les langages synchrones sont des langages spécialisés conçus pour la programmation sûre de systèmes réactifs et temps réel (Amagbégnon, Besnard et Le Guernic, 1995). Ils sont utilisés dans d'autres domaines comme la conception à haut niveau des circuits complexes et la programmation des systèmes embarqués. Ils s'appuient sur des modèles mathématiques et se prêtent à des vérifications formelles.

Les objets manipulés par les langages synchrones sont principalement les signaux et les actions à des instants définis :

- les signaux : constituent le moyen unique de communication aussi bien avec l'environnement qu'entre les unités de programmation du langage;
- les actions : accèdent aux objets du programme et peuvent les modifier;
- l'instant : chaque réaction d'un programme synchrone se produit à un instant particulier. La suite des instants définit un temps logique. La notion d'instant permet de définir clairement la simultanéité comme étant la présence de signaux au même instant.

Ces langages tirent partie de leurs fondements mathématiques pour offrir des possibilités étendues d'optimisation de code et de vérification formelle de propriétés. Parmi les langages synchrones les plus répandus, citons: Lustre, Signal et ESTEREL (Amagbégnon, Besnard et Le Guernic, 1995).

1.3 Flots de conception

Plusieurs processus et méthodologies sont proposés dans le cadre de MBE, pour représenter un système durant les différentes phases du développement de son cycle de vie. En effet les modèles peuvent être présents à tous les niveaux : spécification de requis, description des

fonctionnalités, développement d'architecture, analyse des propriétés, validation et vérification de la conformité aux exigences initiales.

Quelques approches de modélisations utilisées pour la conception des systèmes embarqués seront exposées pour présenter ces différents aspects.

1.3.1 Modélisation avec l'outil AADS

Les auteurs (Varona-Gómez et Villar, 2009) ont développé une approche pour pallier la complexité croissante de conception des systèmes embarqués. Cette approche repose sur la description d'architecture en se servant des principes de MBE, le langage AADL a été utilisé pour la modélisation des fonctionnalités logicielles, des composants matériels et l'analyse des contraintes et requis.

Par la suite, un outil AADS (AADL Simulation tool) a été développé pour supporter la simulation et l'implémentation des modèles AADL. Cet outil permet ainsi d'analyser les performances des spécifications AADL et d'effectuer des vérifications de contraintes fonctionnelles et non fonctionnelles du système (temps d'exécution, échéances manquées, détection des blocages ...). L'outil AADS a été développé en JAVA et intégré comme plugin en Eclipse. La Figure 1.3 présente le flot de modélisation proposé dans ce projet.

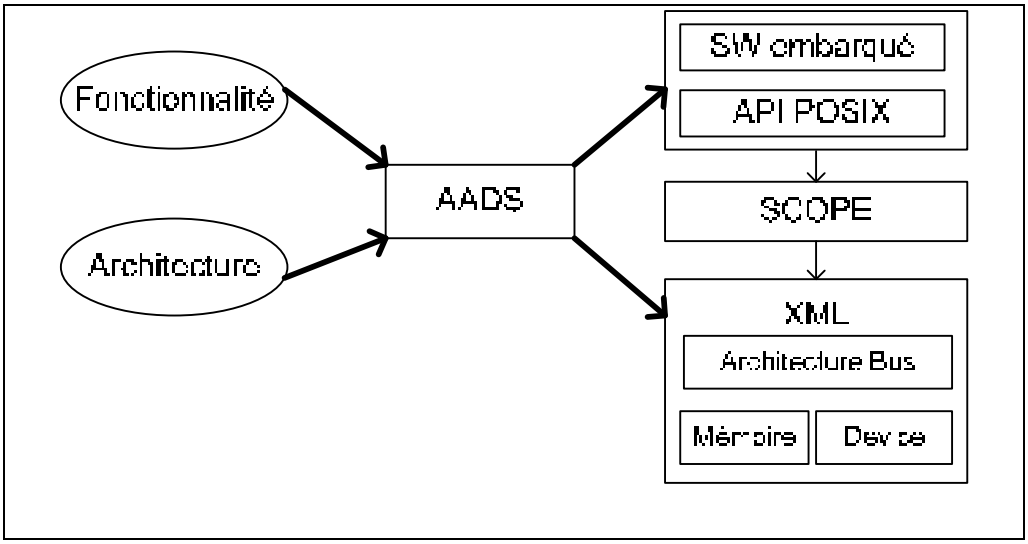


Figure 1.3 Flot de modélisation utilisant l'outil AADS
Adaptée de Varona-Gómez et Villar (2009)

Le flot consiste à décrire en langage AADL l'architecture matérielle et la fonctionnalité d'un système embarqué temps réel. Ensuite AADS effectue une analyse syntaxique (parsing) du modèle et génère un modèle SystemC approprié à l'environnement de simulation [1]Scope. La fonctionnalité du modèle AADL est traduite en un modèle POSIX équivalent tandis que l'architecture du système est représentée en XML. Les outils SCOPE et ADDS sont utilisés pour analyser les performances temporelles du système et requis non fonctionnelles (énergie consommée par l'ensemble du système).

Comme présentée, cette approche permet d'effectuer des simulations et des analyses de performance d'un modèle AADL haut niveau. Mais tél que remarqué, l'évaluation de l'aspect matériel du système est incomplet. Donc notre objectif c'est d'essayer d'orienter plus l'évaluation pour couvrir l'aspect matériel autant que le logiciel.

1.3.2 Approche de modélisation de l'environnement Polychrony

Dans ce travail (Yu et al., 2011) l'auteur présente une approche de modélisation reposant sur la composition, l'intégration et la simulation de modèles hétérogènes[2]. Elle assure un développement parallèle des aspects logiciel et matériel. Le principe de cette méthode consiste à :

1. Modéliser le comportement fonctionnel de l'application en utilisant l'outil Simulink (modèle synchrone);
2. Modéliser l'architecture du système en langage AADL (modèle asynchrone);
3. Traduire ces modèles de haut niveau synchrone et asynchrone (Simulink et AADL) en un modèle commun basé sur un modèle de calcul GALS (Globaly Asynchrone

[1] SCOPE : est un outil de co-simulation matérielle/logicielle des contraintes temporelles (http://www.teisa.unican.es/gim/en/scope/scope_web/scope_home.php)
[2] Modèles hétérogènes : modèles décrivant la fonctionnalité et l'architecture du système.

Locally Synchrone). Cette traduction se fait via une transformation automatique dans l'environnement Polychrony.

Démarches de la méthode :

La Figure 1.4 démontre le flot de l'approche proposée dans ce projet. Elle consiste à :

4. Distinguer la partie « comportement fonctionnel » et la partie de « composantes » constituant l'architecture matérielle;
5. Transformer les modèles AADL et Simulink en un programme Signal[3] (Besnard et al., 2010) via le modèle SME (Signal Meta under Eclipse);
6. Générer du code C/Java à partir du programme Signal;
7. Évaluer les architectures système en se basant sur les résultats de simulation de l'outil « Polychnrony ».

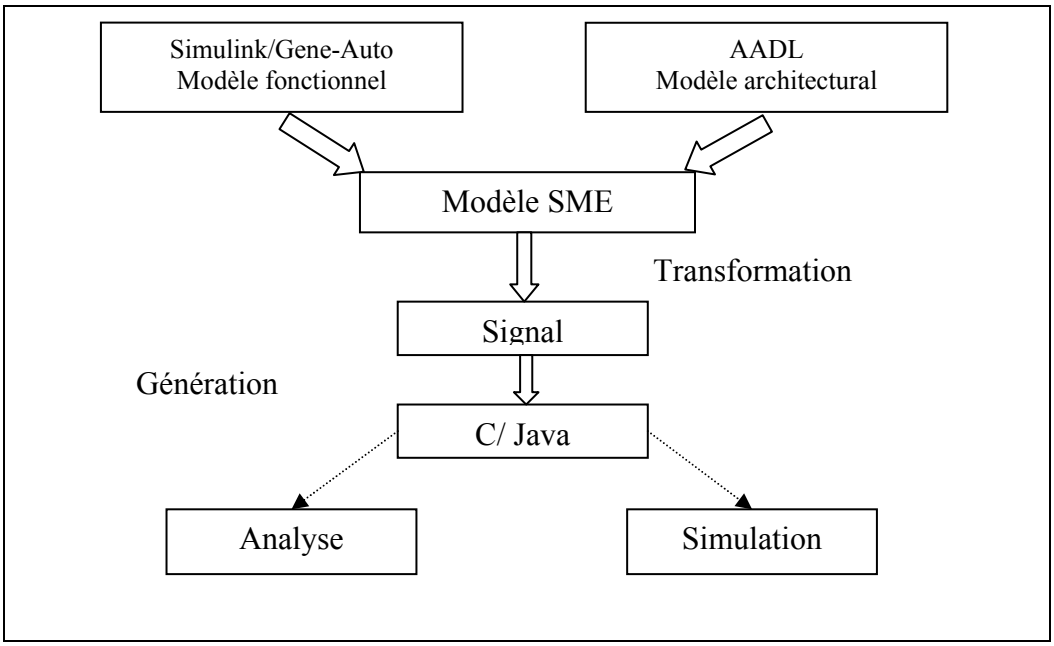


Figure 1.4 Flot de conception des modèles hétérogènes
Adaptée de Yu et al. (2011)

[3] Signal : un langage pour la programmation d'applications temps réel, suivant une approche synchrone.

Deux types de simulations sont effectués par l'outil Polychrony pour valider et évaluer le système, à savoir :

- profilage de code: c'est un type d'analyse dynamique des contraintes temporelles à partir des informations fournies lors de l'exécution de programme;
- vcd (Value Change Dump): permet la visualisation de changement de valeurs des différents signaux du système lors de l'exécution du programme, les fichiers VCD sont générés à partir d'un outil de type EDA (Electronic Design Automation).

1.3.3 Approche de modélisation AADL vers BIP

Le travail réalisé dans Chkouri et Bozga (2009) a donné lieu à une approche de modélisation des systèmes embarqués se basant sur le principe de simulation et de vérification des modèles AADL à l'aide du langage BIP.

Le langage AADL est utilisé pour décrire le comportement de l'application ainsi que les aspects fonctionnels et non fonctionnels d'un système. Le modèle AADL est ensuite traduit automatiquement en langage BIP qui permet d'établir des systèmes robustes et sûrs. La chaîne d'outil de traduction des modèles AADL vers BIP génère un modèle formel et une implémentation exécutable du système sur une machine virtuelle.

Après avoir généré le modèle formel du système, la chaîne d'outils associés au langage BIP permet d'effectuer des analyses et des vérifications. Par exemple la chaîne d'outils BIP assure la vérification de blocage, d'invariants d'états [4]et d'ordonnancement.

La Figure 1.5 montre le diagramme de la chaîne d'outils et les différents éléments permettant d'analyser le modèle BIP.

---

[4] Invariants d'états : contraintes d'exécution sur les interactions, telles que des valeurs de variables, d'attributs et d'états.

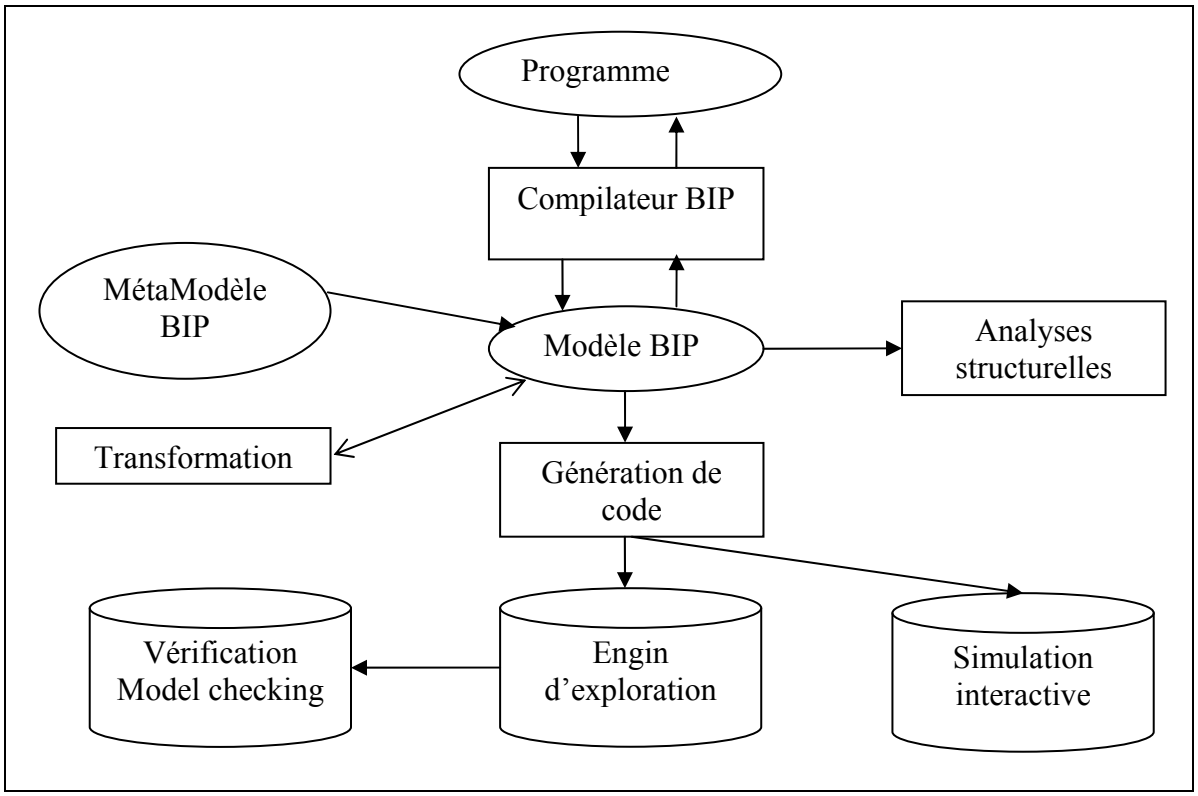


Figure 1.5 Chaîne d'outil associé au langage BIP
Adaptée de Chkouri (2010)

Cette chaîne contient :

- un éditeur pour la description textuelle;
- un parseur pour générer et analyser les modèles BIP;
- un reconstructeur de code pour fournir une description textuelle à partir d'un modèle BIP;
- un générateur de code C++ exécutable;
- un outil de transformation du modèle pour des fins d'optimisation;
- un outil d'analyse structurelle D-finder (2009);
- un outil de model checking de la chaîne IF (Bozga et al., 2004).

L'étude des différentes approches présentées nous a permis d'élaborer un cadre critique du contexte de recherche. Nous avons pu ressortir des conclusions et propositions pour mieux

orienter les travaux de ce mémoire. La section suivante présente les limitations observées et les directions prises pour la suite du projet.

1.4 Limites observées

Après avoir fait un survol sur les différentes approches de conception des systèmes embarqués et langages de modélisation, plusieurs observations ont pu être faites :

- la validation des modèles de haut niveau est souvent liée aux analyses des aspects logiciels du système. Par exemple dans les différents flots présentés, l'évaluation de performance d'un modèle porte sur l'analyse des contraintes non fonctionnelles (énergie), des spécifications temporelles (ordonnancement) ou bien sur la vérification des exigences d'exécution (blocage);
- les raffinements, effectués dans le cadre de l'exploration architecturale, agissent souvent au niveau de la définition de l'application, ou bien au niveau des propriétés et contraintes d'exécution du système. Par exemple dans (Varona-Gómez et Villar, 2009) le raffinement se fait au niveau de propriétés associées à la définition d'un composant de la librairie du langage de modélisation AADL;
- l'aspect matériel est absent des analyses de performance pour évaluer l'efficacité et la robustesse du système vis-à-vis la plateforme d'exécution;
- le langage de modélisation AADL est un élément fort présent dans les technologies de MBE, grâce à sa grande flexibilité et variabilité de description des différents éléments des systèmes embarqués (logiciel, matériel, propriétés non fonctionnelles, système d'opération, etc.). Mais le majeur inconvénient de ce langage demeure dans l'aspect des analyses effectuées pour des fins de validation et vérification. En effet le langage de modélisation AADL permet seulement d'analyser les contraintes fonctionnelles et non fonctionnelles du modèle haut niveau, indépendamment des ressources matérielles d'exécution du système.

D'après ces observations, on constate que la revue de littérature présentée confirme bien la problématique identifiée dans ce mémoire. Il y a actuellement (selon nos connaissances) une

absence d'outils et de méthodologies MBE de conception qui associent à la fois validation fonctionnelle des modèles haut niveau, vérification des contraintes non fonctionnelles, exécution de modèles et évaluation de performances du système en dépendance avec la plateforme matérielle d'exécution. Donc les travaux menés dans le cadre de ce projet visent à proposer une nouvelle approche de modélisation qui résout ces lacunes et répond à la problématique abordée.

CHAPITRE 2

APPROCHE DE MODÉLISATION PROPOSÉE

Suite à la problématique soulevée et aux conclusions de la revue de littérature, ce chapitre présente le flot de conception proposé. Ce flot de conception est basé sur l'utilisation conjointe de l'approche de modélisation MBE basée sur les modèles de haut niveau avec les outils de simulation de la plateforme d'exécution à bas niveau. Les prochaines sections décrivent les outils de développement utilisés, les détails des différentes étapes de la méthodologie proposée ainsi que les limitations observées.

2.1 Environnement et outils de développement

Cette section présente les différents outils et environnements de développement utilisés pour établir le flot de conception proposé, à savoir l'outil OSATE 2 (Team, 2006) utilisé pour le développement des modèles AADL (OSATE2) et l'environnement Space Studio (Moss et al., 2012) pour la conception du prototype virtuel et l'évaluation des performances du système.

2.1.1 OSATE2

OSATE 2.0 (Open-Source AADL Tool Environment) est un outil à code source libre intégré comme pluging à la chaîne d'outils Eclipse. Cet outil permet l'édition textuelle, la génération de code et l'analyse des modèles AADL. OSATE supporte ainsi la vérification des requis non fonctionnels d'un modèle AADL, par exemple la vérification de la sémantique, de la sécurité, de l'ordonnancement, de la latence, de la fiabilité, etc. OSATE offre aussi la possibilité de passer à une représentation intermédiaire XML des modèles AADL. Cette utilité permet ainsi d'assurer d'autres types d'analyses offerts par des engins d'analyse de fichiers XML.

Le flot de conception proposé dans le cadre de projet repose sur le développement de modèle AADL à haut niveau. Donc l'outil OSATE est choisi pour éditer les modèles AADL.

2.1.2 Space Studio

Space Studio est un outil logiciel destiné à la conception logiciel-matériel des systèmes embarqués sur puce. Il permet la configuration d'une plateforme virtuelle à partir d'une libraire de périphériques matériels. Le processus de partitionnement peut être accompli d'une manière semi-automatique en cochant, à l'aide d'une matrice de connexion, les périphériques matériels supportant l'exécution d'une telle ou telle tâche. Space Studio offre aussi une opportunité unique permettant la transformation automatique des modules logiciels en modules matériels. En outre, l'évaluation de performances du système offert par le volet *Space Monitor* de l'outil Space Studio supporte la simulation des différentes caractéristiques du prototype de la plateforme. En se basant sur ces résultats de simulation, le concepteur pourra ainsi réaliser une exploration architecturale pour étaler l'ensemble des solutions de conception proposées.

2.2 Description du flot de conception proposé

Durant le développement de l'approche proposée, des termes bien spécifiques seront employés selon le contexte de recherche et le principe évoqués dans ce projet. Il s'agit des niveaux d'abstractions d'un système. Deux notions d'abstraction seront abordées le long de ce mémoire : le haut niveau d'abstraction et le bas niveau d'abstraction.

- le haut niveau d'abstraction : concerne seulement les spécifications fonctionnelles et la description du modèle de comportement de l'application en faisant abstraction à la description d'architecture de plateforme;
- le bas niveau d'abstraction : inclut les modèles de composants matériels et les assignements des tâches logicielles sur la plateforme virtuelle.

Après avoir défini les niveaux d'abstraction selon le contexte de ce projet de recherche, présentons en détail l'approche de modélisation proposée. La Figure 2.1 illustre le diagramme descriptif du flot de conception :

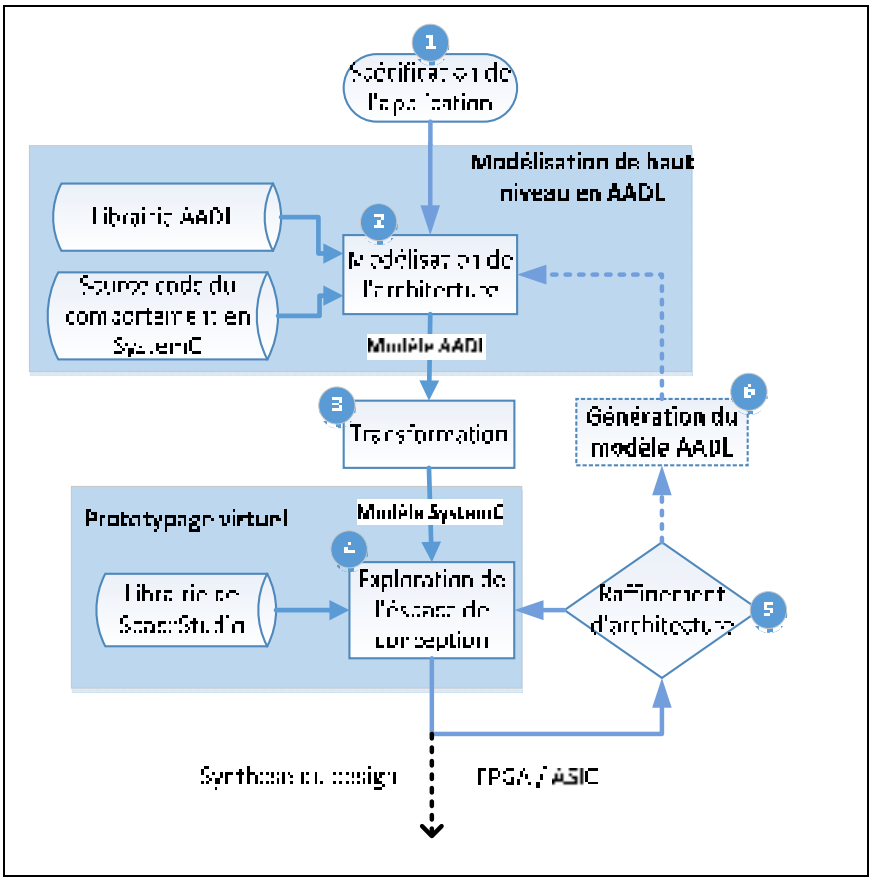


Figure 2.1 Flot de conception proposé

Selon la Figure 2.1, l'approche de modélisation est développée sous six différentes étapes :

1. Spécification de l'application;
2. Modélisation d'architecture à haut niveau;
3. Transformation via le langage ATL (ATLAS Language);
4. Exploration de l'ensemble des solutions de conception;
5. Raffinement d'architecture;
6. Génération du modèle AADL.

La flèche noire en pointillée indiquant la possibilité de faire une synthèse de la solution de design proposée, n'est pas abordé lors du développement des travaux de ce projet.

Mais mentionnons que l'outil Space Studio permet d'accomplir une synthèse du prototype virtuel proposé, et par la suite élaborer une implémentation sur un circuit FPGA via une extension d'outil associé à Space Studio.

Chacune des étapes du flot de conception proposé est détaillée dans les prochaines sections.

2.2.1 Spécification de l'application

La spécification de l'application consiste à définir les fonctionnalités du système et préciser les requis auxquelles il doit répondre. L'expression de la spécification d'une application peut prendre plusieurs formats, à savoir documents textes, chronogrammes, diagrammes graphiques, représentation schématique, tableaux, etc.

Le but de cette étape est de constituer un point de départ pour le concepteur. Elle permet d'illustrer clairement tous les éléments clés du système, ainsi que de disposer d'une source de référence à utiliser dans les différentes phases de conception. Par exemple, durant le processus de conception, la spécification de l'application peut servir à évaluer l'exactitude des fonctionnalités du système et comparer la conformité des résultats collectés aux requis initiaux.

2.2.2 Modélisation de l'architecture à haut niveau

À cette étape, les spécifications fonctionnelles sont prises en considération pour définir l'architecture logicielle du système à concevoir. Le langage de modélisation AADL a été choisi pour établir un modèle à un haut niveau d'abstraction traduisant le comportement de l'application. Le langage de modélisation AADL a été retenu pour plusieurs raisons. L'une des principales raisons est sa structure et sa sémantique pour représenter un système. En effet le langage de modélisation AADL permet d'établir des modèles à haut niveau d'abstraction via une syntaxe claire et simple à apprendre.

Une autre raison pour le choix du langage AADL concerne le concept d'analyse. En fait, l'outil de développement des modèles AADL assure plusieurs vérifications du modèle. Ces

vérifications portent principalement sur l'analyse des requis non fonctionnels (sécurité, fiabilité, coût, etc.), l'analyse de contrainte temporelle (ordonnancement, estimation de pire temps d'exécution) ou bien sur la vérification formelle en utilisant certaines extensions (Hladik, Peres et Shi, 2010). Donc le langage AADL est utilisé dans ce flot de conception pour élargir son champ d'analyse et assurer une évaluation de performance du modèle en liaison avec les contraintes de la plateforme matérielle d'exécution.

En plus, dans le cadre du projet CRIAQ AVIO-509 où le contexte avionique est fortement présent, le langage de modélisation AADL a été utilisé dans (Savard, 2012) pour proposer un environnement de modélisation et simulation conforme à la norme ARINC653. Le travail développé dans (Savard, 2012) a analysé juste les aspects logiciels des applications avioniques. Donc, au moment de la définition de notre projet de recherche, ces travaux ont été pris en considération pour proposer un flot de conception complémentaire qui aborde l'aspect matériel ainsi que le logiciel.

Dans la méthodologie proposée, l'architecture logicielle de l'application est définie en utilisant le langage AADL. Le modèle AADL ainsi créé utilise les composants suivants de la librairie du langage de modélisation AADL (la terminologie anglaise est utilisée par souci de clarté avec les termes utilisés dans la librairie AADL) :

- package : est utilisé pour stocker l'architecture, les modèles ou les composants décrits en AADL. Il permet de construire une bibliothèque d'éléments réutilisables pour la suite des développements sans avoir besoin de les réécrire;
- system : est utilisé pour instancier tous les types de composants d'un modèle. Il permet de construire une architecture et définir la structure interne du *package* tout en liant les différents éléments via leurs interfaces de communication. Il assure ainsi le déploiement des composants logiciels sur des composants matériels;
- interface : constitue le moyen de communication des différents composants AADL avec les autres structures externes de l'architecture décrite, tel que : les ports d'échange de données, les ports de transmission des événements, les groupes de ports, ou même des accès à des sous composants via des variables partagées;

- implementation : définit la structure interne des composants AADL. Il peut contenir les propriétés de ce type de composant, ses connexions avec d'autres interfaces, des appels de fonctions (call), des sous-composants, des extensions, etc;
- properties : attributs qui indiquent toute information nécessaire à la mise en œuvre d'un élément AADL ou à son analyse, par exemple la période d'un thread, les latences des connexions, la taille des données, etc;
- process : définit les plages mémoires occupées lors de l'exécution des threads. L'exemple suivant (Figure 2.2) illustre un cas d'utilisation d'un process. Le process x contient des sous composants thread et une connexion;

```
process x
end x ;
process implementation x. Impl
      subcomponents
            A : thread C ;
            B : thread D ;
connections
      data port A.outB-> B.inpA ;
end x.Impl ;
```

Figure 2.2 Exemple d'utilisation d'un processus

- thread : représente un flot de contrôle qui exécute un programme. Un ensemble de propriétés pourra être associé à chaque thread pour représenter quelques caractéristiques. Telle que la période, la politique de déclenchement, l'échéance, etc. L'exemple suivant (Figure 2.3) illustre un cas d'utilisation de *thread;*

```
thread x
      features
            inpA : in event port;
            outB : out event port ;
properties
      Dispatch_protocol=>Periodic;
      Period => 20ms;
end x ;
```

Figure 2.3 Exemple d'utilisation d'un fil d'exécution

Le thread x est périodique avec une période de 20 ms, il reçoit un événement à travers le port inpA et envoie une donnée à travers le port outB;

- subprogram : représente un extrait de code exécutable (Figure 2.4) qui pourra être intégré au modèle AADL. Il permet par exemple de décrire dans n'importe quel langage le comportement d'un système;

```
Subprogram sp
      features
            s : out parameter message;
      properties
            Source_Languages => "c";
            Source_Name => "code";
end sp;
```

Figure 2.4 Exemple d'utilisation d'un sous-programme

- data : définit les structures de données qui vont être utilisées par les éléments mentionnés ci-dessus;

```
data Person
end Person ;
data implementation Person.impl
      subcomponents
            Name : data string ;
            Adress : data string ;
            Age : data integer ;
end Person.impl ;
```

Figure 2.5 Exemple d'utilisation d'une structure de donnée

Comme évoquée précédemment, la modélisation de l'architecture du système à un haut niveau implique la description de l'architecture logicielle ainsi que le comportement des différentes fonctionnalités définies lors de la première étape.

2.2.3 Transformation de modèle

Une fois la description du modèle haut niveau est établie, une transformation de modèle est effectuée. Cette transformation consiste à traduire le modèle AADL de haut niveau en un modèle SystemC.

La chaîne d'outils de transformation utilisée est la chaîne de l'équipe Open People. Elle est développée dans le cadre du projet de recherche (Bomel et al., 2012). L'objectif de ce travail était d'associer la simulation événementielle du SystemC à la modélisation et analyse de propriétés non fonctionnelles du langage AADL. La chaîne de transformation permet de générer automatiquement un modèle SystemC à partir d'un modèle AADL. La transformation de modèle est basée sur la traduction syntaxique des déclarations réservées au langage AADL vers SystemC. Par conséquent, une librairie a été développée dans le cadre du projet (Bomel et al., 2012) contenant tous les types et classes équivalentes aux concepts de AADL et leurs équivalents en SystemC.

Notons qu'à ce niveau, l'interface de communication de l'architecture modélisée en AADL n'est pas encore supportée pendant cette étape de transformation automatique.

2.2.4 Conception conjointe matérielle et logicielle (Co-Design)

Dans l'objectif d'intégrer l'aspect matériel au flot de conception proposé, l'outil de co-conception Space Studio a été choisi. Space Studio est utilisé pour la création d'une plateforme virtuelle permettant l'exécution et la simulation du modèle haut niveau développé à l'étape 1.

À cette issue, la démarche de l'exploration des solutions de conception est développée en trois grandes lignes, à savoir le prototypage virtuel, l'assignement ou le partitionnement matériel/logiciel et l'interconnexion des composants.

2.2.4.1 Prototypage virtuel

Le prototypage virtuel consiste à constituer une plateforme virtuelle des éléments matériels à partir d'une librairie de composants. Le but est de mettre en place un environnement virtuel capable d'exécuter le modèle AADL de l'application, procéder à une validation du logiciel plus tôt dans les phases de design et mener une évaluation des performances du système tenant compte des variantes de la plateforme d'exécution.

L'outil Space Studio permet de diminuer considérablement le temps de développement. Il suffit d'instancier les composants désirés de la libraire de Space Studio et l'outil se charge de produire la configuration souhaitée.

2.2.4.2 Processus d'assignement

Le processus de partitionnement nous permet d'assigner les composants logiciels SystemC, obtenus lors de la transformation du modèle AADL, aux éléments du prototype virtuel de la plateforme matérielle d'exécution. L'insertion du modèle SystemC à l'environnement de Space Studio est effectuée via une importation de librairie.

Une fois la plateforme virtuelle configurée et le modèle SystemC (étape 3) importé à l'environnement Space Studio, le processus d'assignement est lancé pour attribuer l'exécution des fonctions logicielles aux composants matériels. Il faut juste cocher l'endroit où une tâche logicielle s'exécute et le code de partitionnement est généré automatiquement (Figure 2.6).

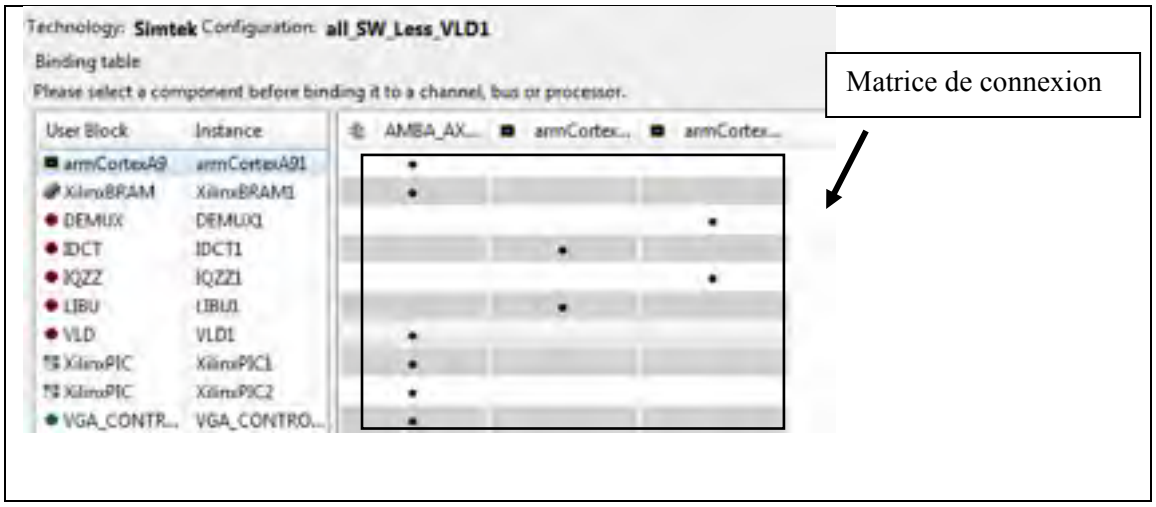


Figure 2.6 Matrice de connexion des composants du prototype virtuel

Space Studio offre l'opportunité de procéder à un processus de partitionnement assez particulier et rapide. En effet l'utilisateur aura la possibilité de transformer ses blocs logiciels en blocs matériels sans avoir besoin à reprogrammer ou reconfigurer la plateforme d'exécution. Par conséquent, les fonctionnalités d'une application peuvent s'exécuter soit en tant que blocs logiciels sur un processeur ou comme des composants matériels dédiés. Cette grande flexibilité d'assignement permet aux utilisateurs d'accélérer le processus de conception et d'exploration pour améliorer les performances du système.

2.2.4.3 Interconnexion des composants

Dès que la configuration de prototype et le partitionnement de modules sont achevés, Space Studio génère automatiquement une plateforme virtuelle en SystemC au niveau TLM-2.0.

Cette plateforme virtuelle générée comprend toutes les connexions des différents composants matériels (processeur, buses, mémoires, périphériques…).

Rendu à ce niveau de la méthodologie proposée, il reste valider le système conçu. Pour se faire, les performances du prototype sont évaluées et des raffinements sont effectués pour améliorer davantage ces performances. C'est l'objectif de la prochaine étape détaillée dans la prochaine section.

2.2.5 Raffinement d'architecture

Une fois que l'exécution de l'application logicielle sur le prototype virtuel est lancée, un ensemble de résultats de simulation pourra être collecté à partir du volet *Space Monitor* de l'environnement Space Studio. Ces résultats de simulation servent pour des fins de validation et évaluation de performances.

En utilisant *Space Monitor* il est possible de visualiser les communications entre modules, les transactions de bus, l'activité des différents composants ainsi que les accès mémoires. Par conséquent, le concepteur sera capable de détecter les goulots d'étranglements du système, valider les fonctionnalités et évaluer les limitations du système en tenant compte des performances des modules de la plateforme (latence de mémoire, charge de processeur, etc).

2.2.5.1 Évaluation des performances

L'évaluation des performances est une étape essentielle de l'approche proposée. Cette étape nous permet d'analyser et valider le système en question. Plusieurs métriques sont considérées pour évaluer les performances du système. Plus particulièrement les métriques liées à la plateforme matérielle. Parmi ces métriques, citons :

Accès à la mémoire : cette métrique est obtenue après simulation du système, et contient toutes les informations à propos des accès mémoires, par exemple :

- ordonnancement: ce graphique montre toutes les informations détaillées pour un seul accès mémoire, incluant le module de source, le type d'opération, la taille et la durée d'un accès et aussi l'adresse de destination;
- cadence: ce graphique permet d'avoir le nombre d'écritures et lectures d'un accès mémoire par période.

Bus de Communications : ce graphique permet la visualisation de toutes les transactions sur le bus et les canaux de communication. Par conséquent, le concepteur pourra déboguer facilement les accès au bus basé sur les priorités incluant la source de transaction, sa destination, sa taille;

Taux d'utilisation du processeur : cette métrique permet de montrer le taux d'occupation du processeur par les tâches exécutées ainsi que les informations d'ordonnancement correspondant à chacune d'eux;

Utilisation du SDL (Space Direct Link) : SDL sont des canaux FIFO utilisés lors des communications rapides entre les modules matériels. L'utilisation de SDL permet d'afficher des changements au niveau des FIFO internes, temps d'accès au FIFOs ainsi que le nombre maximum de FIFOs utilisés.

2.2.5.2 Exploration architecturale

Dans le but de déterminer des spécifications précises de la plateforme matérielle supportant l'exécution de l'application, une phase d'exploration architecturale a été effectuée pour améliorer les performances de façon optimisée, rapide et efficace.

Cette phase d'exploration architecturale permet d'ajuster et agir sur certains éléments au niveau matériel et fonctionnel. Les principales métriques qui peuvent influencer le niveau de performance du système sont :

- le nombre de composants de l'architecture matérielle;
- le choix de déploiement des modules sur la plateforme d'exécution;
- la configuration choisie de la plateforme d'exécution (type de périphériques constituant le prototype);

- les paramètres des modules matériels (capacité de mémoire, taille des FIFOs, type d'architecture de processeurs, plage d'adressage, fréquence d'horloge, etc.);
- le comportement du modèle haut niveau: modification, ajout ou suppression de certaines fonctionnalités des modèles AADL.

2.2.6 Génération du modèle AADL

Il s'agit ici d'une génération d'un modèle AADL à partir de la configuration finale de l'ensemble du système conçu (plateforme matérielle + architecture logicielle). Autrement dit la configuration finale sera modélisée en langage AADL.

Le but de générer le modèle AADL à ce niveau est de pouvoir procéder à d'autres types d'analyses non fonctionnelles non supportés par l'environnement Space Studio. Ces analyses peuvent porter sur la sécurité, fiabilité, consommation de puissance, robustesse, qualité, etc. L'intérêt de cette étape est de garantir une vérification additionnelle aux premières phases de design et réduire tout risque d'erreur ou de panne lors de l'exécution du système.

2.3 Limitations de l'approche proposée

Durant le développement des travaux d'expérimentations, les principales limitations de la méthodologie proposée ont été identifiées. Parmi ces limitations citons :

- le choix restreint de la librairie de périphériques matériels offerts par l'outil Space Studio. En effet, lors de la configuration du prototype virtuel, la librairie de space studio comprend un nombre limité de composants et donc un nombre limité des propositions de solutions de la plateforme matérielle;
- au niveau de l'outil Space Studio version 2.5, le seul système d'opération gérant le fonctionnement du processeur est μC-OS II. Le problème se pose lors de la conception par exemple des applications avioniques. Ce type d'application nécessite un système d'opération spécifique comme le *ARINC653* (Prisaznuk, 2008) pour gérer les problèmes

liés au partitionnement de plusieurs tâches fonctionnelles simultanément sur les mêmes ressources matérielles;

- au niveau de la transformation automatique de modèle AADL vers un modèle SystemC, les liens de communication entre les différents éléments du modèle AADL ne sont pas supportés. Par conséquent la définition de l'interface de communication entre les différents modules est effectuée selon les choix de partitionnement au niveau de l'outil Space Studio et suivant des règles de connexions spécifiques à cet environnement;
- absence d'analyse et évaluation des aspects non fonctionnelles du système, tel que la sécurité, fiabilité, coût, etc.

CHAPITRE 3

ÉTUDE DE CAS

Ce chapitre présente les travaux expérimentaux entrepris dans le cadre de ce projet de maîtrise. Une application de traitement vidéo MJPEG sera utilisée dans ce cas d'étude pour valider la méthodologie proposée. Cette application sera illustrée à travers les quatre premières étapes du flot de conception, à savoir la spécification de l'application, le modèle AADL, la transformation du modèle AADL vers SystemC et le modèle de la plateforme matérielle. Les différents modèles et documents générés pour la réalisation de l'approche proposée seront aussi présentés.

3.1 Spécification de l'application

Un exemple d'application fourni par la compagnie Space Codesign a été exploité pour la suite des expérimentations de ce travail de recherche. Il s'agit ici d'une application multimédia MJPEG (Motion JPEG) pour le décodage d'un flux vidéo en images JPEG (Keinert et Teich, 2011). Le diagramme fonctionnel de l'application MJPEG est présenté à la Figure 3.1. Ce diagramme contient les principaux blocs et éléments pour développer une telle application.

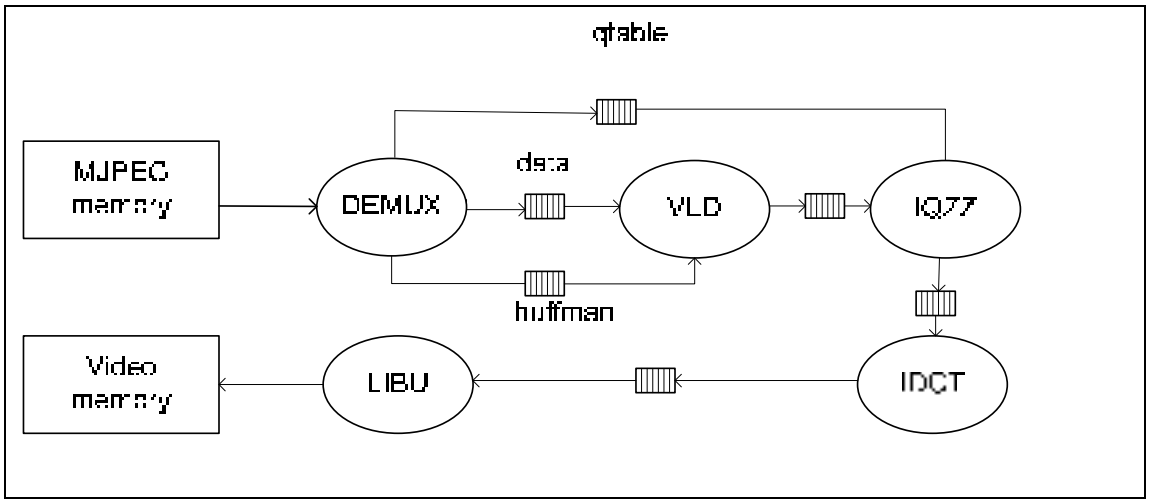


Figure 3.1 Diagramme fonctionnel de l'application MJPEG

Le diagramme fonctionnel de l'application MJPEG contient plusieurs blocs présentés en détail ci-dessous :

- fichier MJPEG : stocké dans un élément mémoire, ce fichier contient les marqueurs nécessaires au décodage par le module Demux;
- bloc Demux : se charge du démultiplexage du flux de données. Ce bloc lit le contenu de la mémoire et analyse les marqueurs qui délimitent les sections du fichier. Ensuite, la table de dé-quantification (*qtable*) est envoyée au bloc IQZZ. La table de Huffman et les données du fichier sont quand à elles envoyées au module VLD;
- bloc VLD : se charge de la décompression du flux compressé. Pour se faire, ce bloc décode les données envoyées par le bloc Demux avec les tables de Huffman, puis envoie le résultat du décodage au bloc IQZZ;
- bloc IQZZ : se charge quant à lui de la dé-quantification en appliquant la transformée ZigZag (wiseGeek, 2014). Cette transformation est effectuée via un parcours en dents de scie (En anglais unzig zag). Ensuite, les résultats obtenus sont transférés au bloc IDCT;
- bloc IDCT : applique une transformée cosinus inverse sur les données reçues et envoie le résultat au module LIBU;

- bloc LIBU : construit des lignes complètes d'images à partir des données reçues de l'IDCT et envoie ensuite les pixels vers la mémoire vidéo.

Le but de cette première étape est de faciliter la collecte d'information aux concepteurs et associer tous les éléments clés et requis d'un système en un seul document pour ainsi réduire le temps de développement.

3.2 Modèle AADL

La deuxième étape du flot de conception consiste à traduire les spécifications de l'application de l'étape 1 en un modèle AADL de haut niveau d'abstraction. La Figure 3.2 illustre sous forme graphique le modèle AADL de l'application MJPEG.

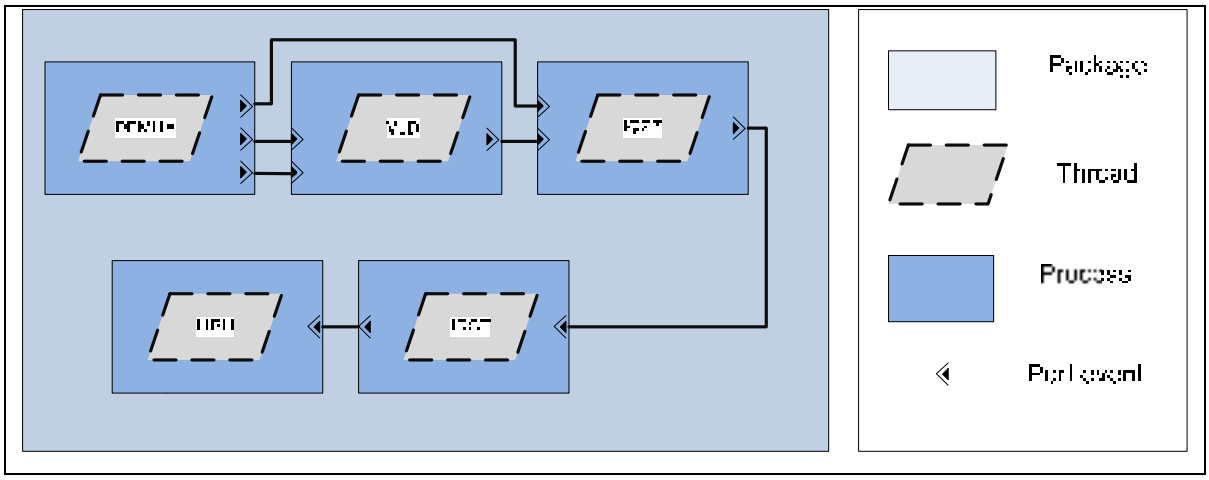


Figure 3.2 Représentation graphique du modèle AADL de l'application MJPEG

Le modèle AADL de la Figure 3.2 représente l'architecture fonctionnelle de l'application MJPEG (voir ANNEXE I). Cette architecture décrit le comportement ainsi que les spécifications logicielles du MJPEG. Le modèle a été établi en utilisant différents composants logiciels de la librairie du langage AADL :

- paquet (package) : englobe l'ensemble des éléments et décrit la structure globale de l'architecture logicielle du modèle AADL;

- processus (process): définit les différentes connexions entre les fils d'exécution de l'architecture fonctionnelle;
- port d'événement (port event) : permet l'émission et la réception d'un signal.
- fil d'exécution (thread) : décrit le comportement des blocs de l'application MJPEG. Le langage AADL permet d'intégrer un sous-programme, écrit dans un autre langage, dans la déclaration du fil d'exécution. Ce sous-programme consiste en un code source représentant le comportement détaillé des différents modules de l'application. Le langage utilisé dans notre cas d'étude pour décrire le comportement des fonctions logicielles est le SystemC.

Le SystemC a été choisi pour des fins de compatibilité avec la chaîne de transformation Open People ainsi qu'avec l'outil de prototypage virtuel Space Studio. Ces deux outils utilisent le langage SystemC comme base pour la structure des modèles employés. Prenons en exemple le modèle AADL du fil d'exécution du module IDCT pour illustrer le code créé.

La Figure 3.3 présente l'extrait de code du modèle AADL montrant clairement la description du fil d'exécution (thread) IDCT.

```
subprogram IDCT_Function
 properties
         Source_Language => (System_C);
      Source_Text    => ("IDCT.cpp");
      Source_Name    => "IDCT";
 end IDCT_Function;

 thread IDCT
 features
      IQZZ_In: in event data port IQZZ_Data;
      IDCT_Out: out event data port IDCT_Data;
 properties
  Compute_Entrypoint=> classifier (IDCT_Function);
 end IDCT;
```

Figure 3.3 Extrait de la description textuelle du modèle AADL

Un sous-programme est déclaré incluant toutes les informations sur le code source utilisé. L'interface de communication est définie par des *port event*. C'est à travers cette interface que le fil d'exécution va échanger l'ensemble des données et des informations avec les autres éléments de l'architecture logicielle selon la structure du système. En plus, une propriété propre au langage de modélisation AADL a été déclarée pour faire un appel des fonctions définies au code source.

3.3 De AADL vers SystemC

Une fois que le modèle AADL est établi, la chaîne de transformation Open People est utilisée. Cette étape permet de générer automatiquement un modèle SystemC du modèle AADL précédemment décrit. Cette transformation va constituer le point de liaison qui associe le haut niveau du modèle AADL avec la plateforme d'exécution. Ce lien permet d'entretenir une structure commune (modèle SystemC) entre l'environnement de prototypage virtuel et l'environnement de modélisation à haut niveau d'abstraction. Cette chaîne de transformation est introduite comme une extension au plugin OSATE2 de l'outil Eclipse.

Un extrait du fichier généré lors de la transformation est représenté sous Figure 3.5.

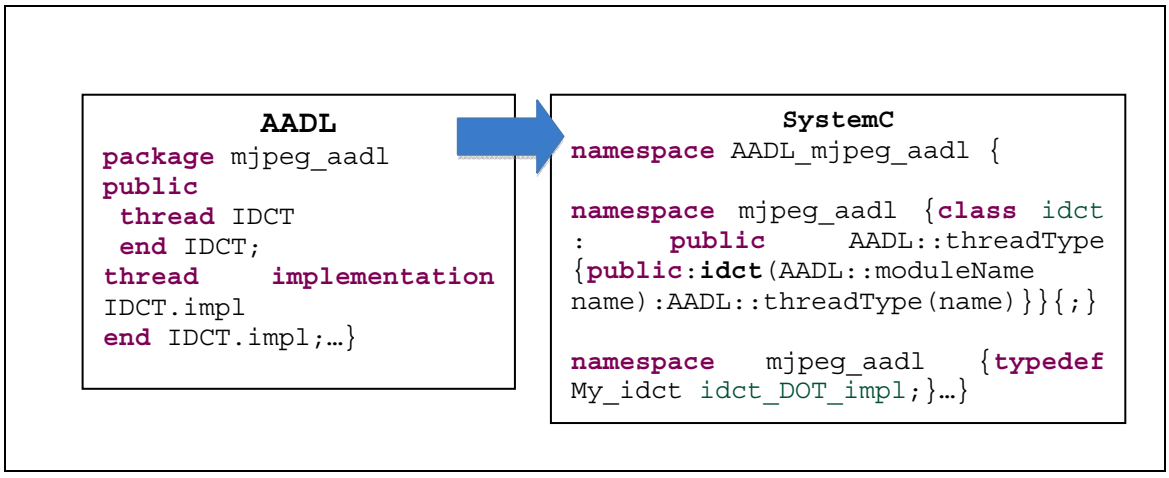


Figure 3.5 Transformation d'un extrait du modèle AADL vers SystemC

Pour chaque type de composants AADL, une classe C++ est attribuée pour reconstruire la structure du modèle AADL en SystemC. Prenant par exemple le type de composant *Packag*e, il est traduit en une classe *namespace* C++ qui contient toutes les informations de l'architecture fonctionnelle définie en AADL. Puisque la chaîne de transformation est basée sur la traduction de la sémantique, tous les éléments du modèle AADL sont conservés dans le modèle SystemC et sont remplacés par des classes C++ qui ont pratiquement les mêmes noms que les composants AADL pour ne pas créer une confusion après transformation. Et c'est pour cette raison qu'une librairie de classes de correspondance entre les composants AADL et modèle SystemC a été développée par l'équipe Open People. Notons qu'à ce niveau du flot, la transformation effectuée ne supporte pas les interconnexions décrites au niveau du modèle AADL. L'interface de communication entre les différents blocs de l'application sera décrite manuellement au niveau de l'environnement de prototypage virtuel Space Studio.

3.4 Modèle de la plateforme matérielle

Cette étape consiste à construire le modèle du prototype virtuel (via l'outil Space Studio) servant de plateforme d'exécution au modèle fonctionnel de l'application MJPEG. Le modèle SystemC obtenu après transformation, décrit l'application MJPEG à travers des fonctions concurrentes. Pour faciliter la communication entre ces fonctions, une interface explicite spécifique à l'outil Space Studio est définie pour chacune de ces tâches concurrentes. La librairie de l'outil Space Studio fournit un ensemble de composants matériels, d'où la possibilité de construire plusieurs solutions d'architecture candidate de plateforme matérielle. Plusieurs éléments peuvent être considérés pour déterminer la meilleure architecture pour implémenter l'application. Citons à titre d'exemple :

- le nombre de processeurs;
- le nombre de cœurs;
- le nombre de bus de communications;
- le partitionnement matériel/logiciel;

- la configuration choisie.

Pour chaque architecture candidate, Space Studio génère automatiquement un modèle de plateforme virtuelle en SystemC compatible avec le standard TLM2.0 (DOULOS, 2014) et des fichiers binaires pour chaque cœur de processeur de la plateforme. En procédant ainsi, une simulation du système entier peut être lancée pour évaluer les performances de la configuration choisie. Comme évoqués précédemment au chapitre 2, les différents axes de cette étape sont effectués suivant trois grandes lignes : la configuration du prototype virtuel, la mise en correspondance de l'application sur la plateforme et les interconnexions des composants de la plateforme matérielle. Les prochaines sections présentent en détails ces trois sous étapes.

3.4.1 Configuration du prototype virtuel

La configuration de la plateforme d'exécution se fait à travers le choix des différents composants matériels de la librairie Space Studio. Aucune programmation manuelle n'est nécessaire, car il suffit de désigner le type du module et de faire la sélection (Figure 3.6). Le prototype virtuel de la plateforme d'exécution est construit en utilisant les composants matériels disponibles dans la librairie de Space Studio.

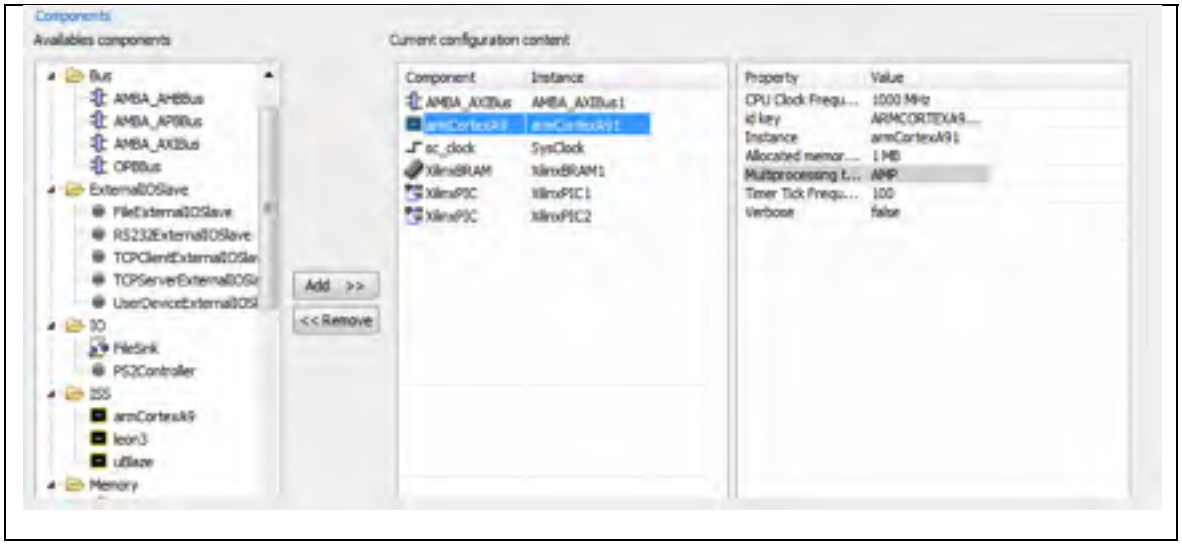


Figure 3.6 Création du prototype virtuel à partir de la librairie de Space Studio

La Figure 3.6 montre un exemple de configuration de prototype virtuel. Ce modèle de prototype contient les composants matériels suivants : un bus de communication de type AMBA_AXI, un processeur ARM Cortex-A9, une mémoire de type BRAM, un contrôleur d'interruption Xilnix PIC et finalement une horloge. Tous ces périphériques matériels ont été choisis à partir de la librairie Space Studio au volet *Available Component.*

3.4.2 Mise en correspondance de l'application sur la plateforme

Une fois que la plateforme virtuelle est configurée, le modèle SystemC généré à l'étape de la transformation automatique est importé à l'environnement Space Studio. Comme présenté sur la Figure 3.7, les cinq blocs compris dans l'application de décodeur MJPEG sont maintenant disponibles sous le volet *User Block* du tableau *Binding Table* de l'outil Space Studio.

Le partitionnement des modules logiciels est effectué à l'aide de la matrice de connexion (Figure 3.7). Chaque module est attribué à l'architecture proposée pour s'exécuter soit comme une tâche fonctionnelle sur le processeur ou bien comme un module matériel bien dédié. À titre d'exemple, à la Figure 3.7 les deux modules DEMUX et LIBU sont attribués au cœur 0 du processeur Arm Cortex-A9, le module IQZZ est attribué au cœur 1 du processeur Arm Cortex-A9 tandis que les autres modules IDCT et VLD sont connectés comme des modules matériels dédiés au bus de communication AMBA_AXI.

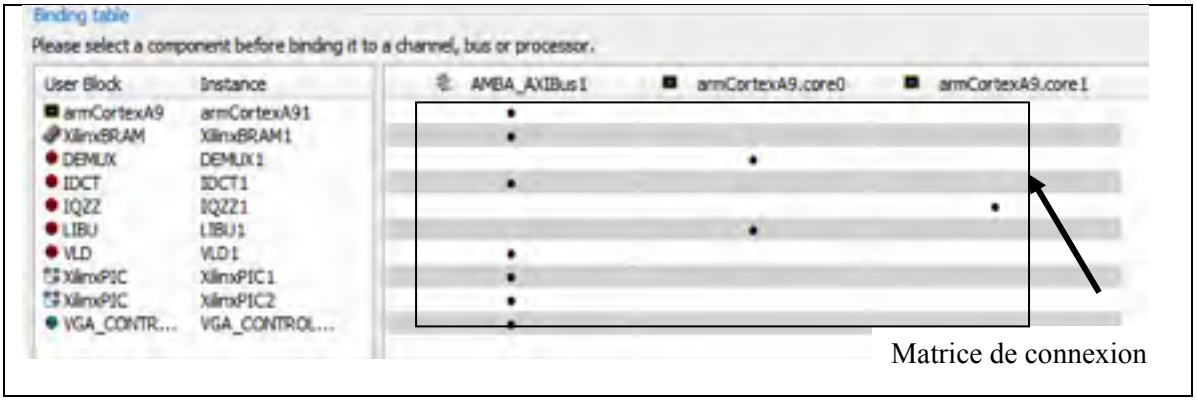


Figure 3.7 Assignement des composants logiciels à la plateforme matérielle

D'autres solutions de correspondance entre l'application et la plateforme peuvent être adoptées en modifiant l'assignement de la matrice de connexions.

La Figure 3.8 présente l'architecture de la plateforme d'exécution proposée. Cette configuration comprend un processeur Arm Cortex-A9, un bus de communication de type AMBA_AXI, une mémoire XilinixBRAM, un contrôleur VGA et d'autres périphériques. Les détails et caractéristiques liés à chacun de ces composants seront présentés ci-dessous :

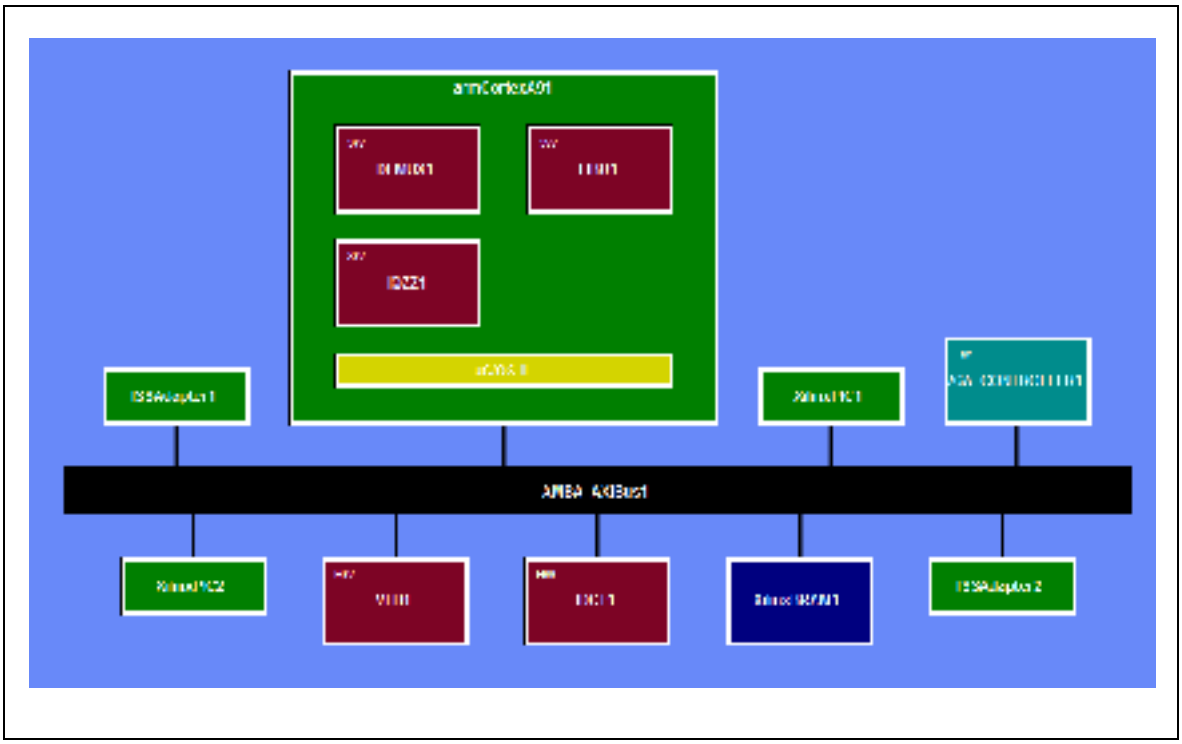


Figure 3.8 Architecture matérielle de la plateforme d'exécution

Processeur Arm Cortex-A9 : le type d'architecture du processeur utilisé est AMP, contient 2 cœurs avec une fréquence de 800MHZ, mémoire alloué 1MB et roule le système d'opération µCII-OS;

ISSadapter : permet d'accélérer la communication entre les modules logiciels attribués au processeur et agir sur le déroulement de la simulation, par exemple il permet aux modules logiciels de demander l'arrêt de la simulation. Chaque cœur de processeur possède son adaptateur pour l'interfacer avec les tâches fonctionnelles;

Xilinx PIC : c'est un composant utilisé pour générer le signal des interruptions au processeur, Space Studio connecte ce composant à l'entrée IRQ du processeur, il est automatiquement instancié au moment du choix d'un processeur;

VGA controller : est un composant défini par l'utilisateur, il sert à recevoir et afficher les images envoyées par le module LIBU. Il est considéré comme un composant externe de la plateforme, mais il est nécessaire pour effectuer la simulation sur la machine hôte;

IDCT et VLD : blocs fonctionnels de l'application MJPEG, ils sont considérés comme des modules matériels spécifiques pour exécuter l'ensemble du système;
LIBU, IQZZ et IDCT : sont des tâches fonctionnelles de l'application MJPEG exécutées par le processeur ARM.

À titre de récapitulation, pour construire le modèle de l'architecture matérielle, Space Studio fait appel au volet *Configuration Manager*. Ce volet permet au concepteur de choisir les périphériques de sa plateforme à partir de la librairie de composants de Space Studio, il peut également modifier les paramètres des périphériques ou bien les décisions de partitionnement pour générer d'autre choix de configurations.

3.4.3 Interconnexion des composants de la plateforme

Dès que le modèle du prototype est construit, Space Studio génère automatiquement une plateforme virtuelle compatible au standard SystemC TLM 2.0 et des fichiers binaires spécifiques à chaque configuration d'architecture.

Le fichier bus_mapping.cpp est l'un des fichiers générés par l'outil Space Studio (voir ANNEXE III). Ce fichier associe une plage d'adresse mémoire aux différents modules de la plateforme matérielle pour les connecter au bus de communication. L'interconnexion des différents modules au bus de communication se fait via leurs adresses mémoires attribuées au niveau du fichier bus_mapping.cpp. L'exemple suivant Figure 3.9 montre un extrait du code bus_mapping; à chaque module de plateforme, Space Studio spécifie une plage d'adressage.

```
AddressBinding AMBA_AXIBus1_address[] =
}#5,{11,0x42005000,0x42005fff},0},// AMBA_AXIBus1_SlaveAdapter_XilinxPIC1
```

```
{10,{4,0x42003000,0x42003fff},0},//
AMBA_AXIBus1_SlaveAdapter_adap_slave_VLD1_AMBA_AXIBus1_WriteInterface
{1,{10,0x40000000,0x40ffffff},0},// MBA_AXIBus1_SlaveAdapter_ISSAdapter1
{3,{5,0x43000000,0x43ffffff},0},// AMBA_AXIBus1_SlaveAdapter_VGA_CONTROLLER1
{9,{4,0x42002000,0x42002fff},0},//
AMBA_AXIBus1_SlaveAdapter_adap_slave_VLD1_AMBA_AXIBus1_ReadInterface
{8,{1,0x42001000,0x42001fff},0},//
AMBA_AXIBus1_SlaveAdapter_adap_slave_IDCT1_AMBA_AXIBus1_WriteInterface
{7,{1,0x42000000,0x42000fff},0},//
AMBA_AXIBus1_SlaveAdapter_adap_slave_IDCT1_AMBA_AXIBus1_ReadInterface
{0,{15,0x42004000,0x42004fff},0},// AMBA_AXIBus1_SlaveAdapter_DebugModule1
{6,{13,0x42006000,0x42006fff},0},// AMBA_AXIBus1_SlaveAdapter_XilinxPIC2
{4,{14,0x42010000,0x4201ffff},0},// AMBA_AXIBus1_SlaveAdapter_XilinxBRAM1
{2,{12,0x41000000,0x41ffffff},0}// AMBA_AXIBus1_SlaveAdapter_ISSAdapter2};
AddressInfo AMBA_AXIBus1_addressinfo = { 11, AMBA_AXIBus1_address};
```

Figure 3.9 Plage d'adressage attribué à chaque composant

Le fichier main.cpp comprend tous les paramètres associés aux modules lors de la configuration du prototype, les instanciations des modules ainsi que les informations liées aux différentes interconnexions (voir ANNEXE IV).

À ce niveau, notre plateforme contient des composants matériels ainsi que des composants logiciels définis au modèle AADL de haut niveau. Donc une partie de l'objectif de recherche précédemment défini est atteinte, car les fonctionnalités du modèle AADL haut niveau ont été intégrées dans un environnement de prototypage virtuel. Cette intégration à pour but de compléter la conception par la définition de l'architecture matérielle qui supporte l'exécution des fonctionnalités logicielles du modèle AADL. Par conséquent, une liaison a été établie entre le haut niveau d'abstraction et le bas niveau de la plateforme. À ce niveau tous les éléments des systèmes sont associés dans un seul modèle de prototype virtuel prêt pour l'exécution et l'évaluation de performances.

Par conséquent le chapitre suivant sera consacré aux résultats de simulation du prototype virtuel, l'évaluation de ses performances ainsi que le raffinement architecturale effectué pour améliorer davantage ces performances.

CHAPITRE 4

RÉSULTATS ET DISCUSSION

Ce chapitre présente les résultats obtenus suite à la réalisation des travaux expérimentaux présentés dans le chapitre précédent. Après avoir effectué la transformation du modèle AADL en SystemC, un prototype virtuel de la plateforme d'exécution est créé pour explorer l'espace de conception (étape 5). Pour ce faire, plusieurs architectures candidates sont créées à l'aide de l'outil Space Studio. Ensuite, une évaluation des performances est réalisée pour chacune de ces architectures. Les solutions proposées sont comparées et raffinées (étape 6) dans le but de choisir la meilleure d'entre elles. Dans un premier temps les résultats d'évaluation de performances du modèle vu au chapitre 3 seront présentés, suivi d'une présentation du processus d'exploration architecturale effectué dans le cadre de l'étape de raffinement des performances.

4.1 Évaluation des performances

Une fois le modèle du système mis en place, l'exécution est lancée et une évaluation des performances peut être effectuée pour analyser et valider le système. Les résultats de simulation sont collectés à partir du volet *Space Monitor* de l'outil Space Studio. Parmi ces résultats, notons : le temps de simulation, le taux d'occupation des processeurs, les statistiques d'utilisation du bus de communication et les informations des accès à la mémoire.

4.1.1 Temps de simulation

Space Studio fournit des informations concernant le temps d'exécution pris par les différents modules fonctionnels sur le prototype virtuel. Deux spécifications temporelles caractérisent la simulation du prototype virtuel. La première spécification est le temps totale de la

simulation communément appelé le « Wall Clock time » et la deuxième spécification est le temps d'exécution.

Tableau 4.1 Tempsde simulation

Temps totale de simulation	746 secondes
Temps d'exécution	0.0539886 secondes

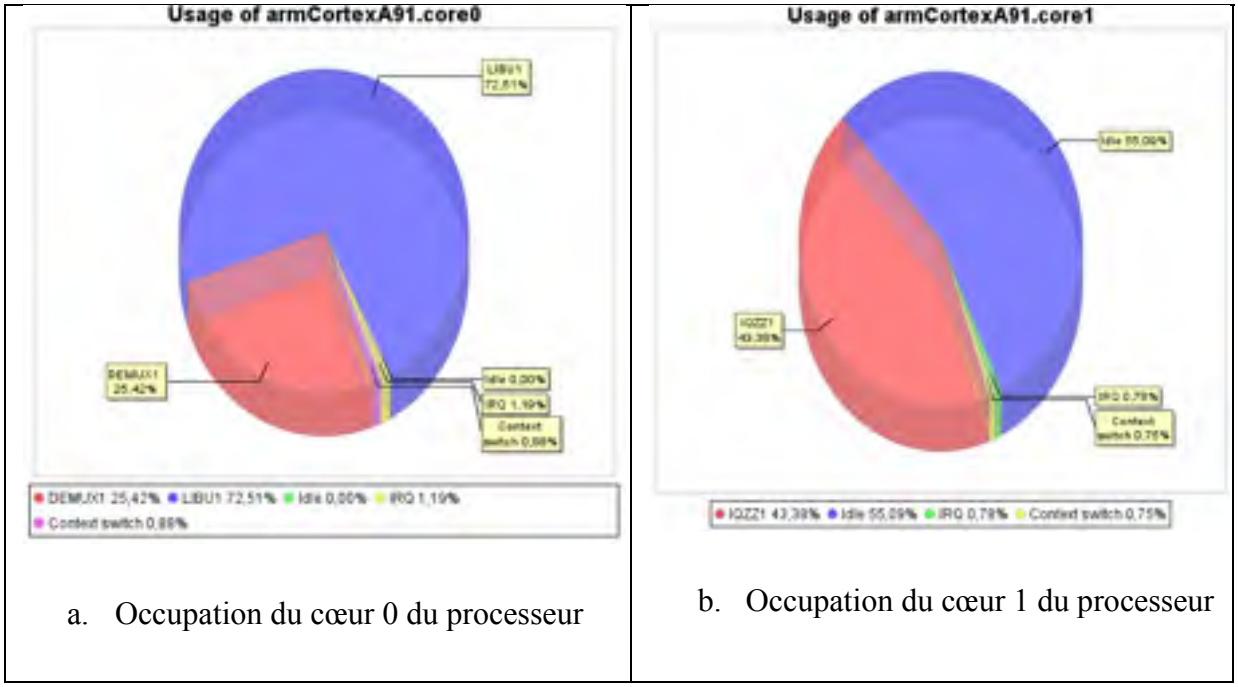
Le Tableau 4.1 présente les résultats temporels de simulation de la configuration présentée à la Figure 3.8. Il faut distinguer le temps total de simulation et le temps d'exécution. Le temps total de la simulation correspond au temps pris pour simuler l'application en question sur la machine hôte. Dans notre cas, la machine hôte est un Intel Pentium Dual Core cadencé à 2 GHz. À l'opposé, le temps d'exécution correspond au temps d'exécution estimé que devrait prendre l'exécution de l'application sur une plateforme réelle ayant les mêmes caractéristiques (fréquence, architecture, capacité de mémoire, etc.).

Dans un premier temps, on constate que le décodage d'une animation MJPEG comportant 25 images de 48*48 pixels chacune nécessite 12,4 minutes (746s) pour simuler 0.0539886 secondes du traitement vidéo en temps réel. Ceci est uniquement vrai pour l'architecture proposée à la Figure 3.8. Douze minutes peuvent sembler importantes en traitement de vidéo, mais il ne faut pas oublier qu'il y a trois tâches logicielles assignées au processeur ARM (qui est simulé via son ISS (Cmelik et Keppel, 1995)). Par exemple, si toutes ces tâches sont mises en matériel, il y a beaucoup moins de composantes à simuler ce qui aura comme conséquence de diminuer le temps réel de simulation. Également, plus la machine hôte sera performante, plus ce chiffre aura tendance à diminuer.

4.1.2 Taux d'occupation du processeur

Le taux d'occupation du processeur est le rapport entre la durée d'utilisation effective du processeur par l'une des tâches logicielles et la durée totale d'exécution.

L'occupation de chaque cœur du processeur par les différentes tâches fonctionnelles est présentée à la Figure 4.1.



a. Occupation du cœur 0 du processeur

b. Occupation du cœur 1 du processeur

Figure 4.1 Résultat d'utilisation du processeur ARM Cortex-A9

Sur le graphique de la Figure 4.1.a, les tâches LIBU et DEMUX qui sont exécutées sur le premier cœur (*core 0*) du processeur ARM, occupent respectivement 72.51% et 25.42% de la charge totale du processeur. Sur le graphique de la Figure 4.1.b. la fonction IQZZ exécutée sur le deuxième cœur du processeur (*core 1*) occupe 43.38% de la charge totale de ce processeur.

Tel que remarqué, le deuxième cœur du processeur ARM demeure inactif 55% du temps, puisque qu'il ne traite qu'une seule tâche logicielle. En ce qui concerne le premier cœur, les deux tâches LIBU et DEMUX occupent ensemble 100% du processeur.

Les résultats d'utilisation du processeur permettent de confirmer l'un des objectifs fixés au début de ce projet. En effet, le modèle haut niveau de l'application MJPEG s'exécute sur un prototype virtuel de la plateforme matérielle. Aussi, des métriques correspondant aux

caractéristiques de la plateforme matérielle sont proposées (dans ce cas l'utilisation du processeur) pour caractériser et valider le modèle haut niveau et ainsi pouvoir optimiser les ressources matérielles exécutant ce type de modèle. Autrement dit, si la configuration du prototype virtuel est modifiée, l'impact de cette modification sur l'exécution du modèle haut est déduit facilement. En procédant ainsi, plusieurs configurations sont explorées en peu de temps pour atteindre la meilleure performance d'exécution du modèle haut niveau de l'application de décodage MJPEG.

4.1.3 Statistiques d'utilisation du bus de communication

Le Tableau 4.2 présente un ensemble d'information sur les échanges effectués à travers le bus AMBA_AXI :

Tableau 4.2 Informations des transactions
du bus de communication

Information sur les communications	Valeur
Nombre d'octets	20 84 376
Nombre de transferts	521 094
Nombre d'écritures	240 707
Nombre de lectures	280 387
Débit du bus	345,65913 (Mbps)

Comme illustré sur le Tableau 4.1, Space Studio permet de collecter les statistiques concernant l'utilisation du bus de communication à savoir le nombre d'octets circulant sur le canal de communication, le nombre de transferts, le nombre d'écritures, le nombre de lectures et le débit du bus. Ces informations peuvent servir le concepteur à estimer le taux d'information et de données traitées par rapport aux capacités de ses ressources matérielles.

4.1.4 Accès mémoire

Le temps d'accès mémoire désigne le délai correspondant à l'accès par le processeur à la vidéo stocké en mémoire BRAM. Cette information sert à évaluer le temps nécessaire pris par la tâche logicielle DEMUX pour lire le contenu de l'information contenu dans la mémoire BRAM.

La Figure 4.2 montre les accès à la mémoire tout au long de l'exécution de l'application. L'accès est à son maximum au début de l'exécution. Ceci est dû au fait qu'à ce moment le système a besoin de la vidéo stockée en mémoire BRAM pour débuter le traitement des trames et procéder au décodage MJPEG.

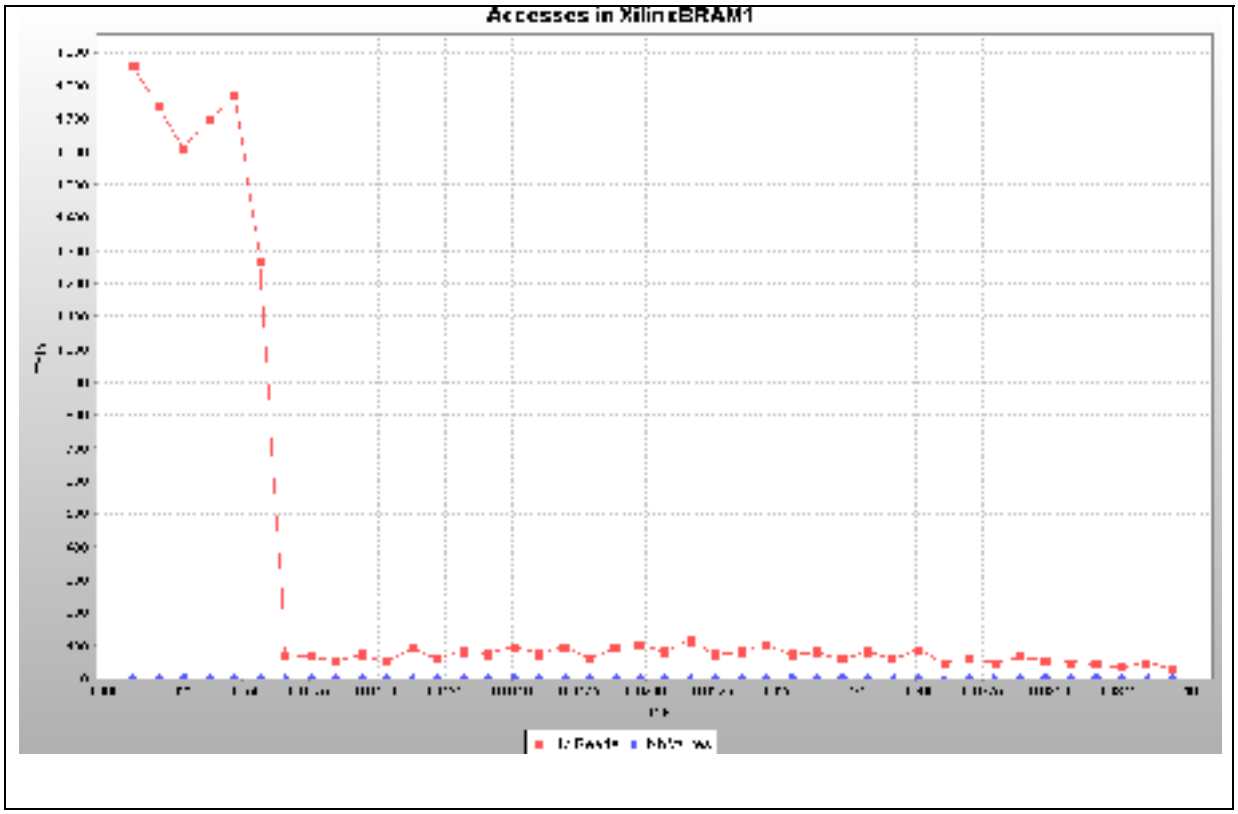


Figure 4.2 Les accès mémoires en fonction du temps

Cette étape d'évaluation des performances permet de mettre l'accent sur un aspect très important de la validation du modèle haut niveau. En effet les résultats des simulations obtenus permettant d'analyser le système dans son ensemble (logiciel et matériel). On peut quantifier l'impact de l'exécution des fonctions logicielles sur les ressources matérielles, telles que l'occupation de processeur par les différentes fonctions logicielles de l'application, les transactions et communication entre module à travers le bus de la plateforme, l'information d'accès au mémoire par le module DEMUX et finalement l'estimation du temps de l'exécution que va prendre ces fonctions logicielles sur une réelle plateforme avant l'implémentation. Vu qu'au départ ces fonctions logicielles sont décrites au niveau du modèle AADL, l'objectif de validation du modèle haut niveau en tenant compte des contraintes d'exécution sur les ressources matérielles est atteint. La prochaine étape vise le raffinement graduellement de la configuration de la plateforme matérielle pour améliorer les performances du système.

4.2 Raffinement d'architecture

Le raffinement d'architecture est effectué à travers une exploration architecturale, en se basant sur les résultats de simulation. Les performances du système peuvent être améliorées et de nouvelles configurations peuvent être mises en place rapidement.

Pour procéder à une exploration architecturale du système, trois facteurs ont été considérés, à savoir le type du processeur, la fréquence d'horloge et le choix du partitionnement.

4.2.1 1er facteur d'exploration : type de processeur

Cette section présente les différentes configurations de la plateforme matérielle établies avec différents types de processeur et les performances liées à chacune d'elle. Tenant compte de la restriction imposée par la librairie de Space Studio, trois types de processeur ont été explorés pour comparer les performances, à savoir ARM, Leon3 et µBlaze.

4.2.1.1 Processeur Leon3

Le prototype virtuel de la plateforme à base du processeur Leon3 est présenté sur Figure 4.3.

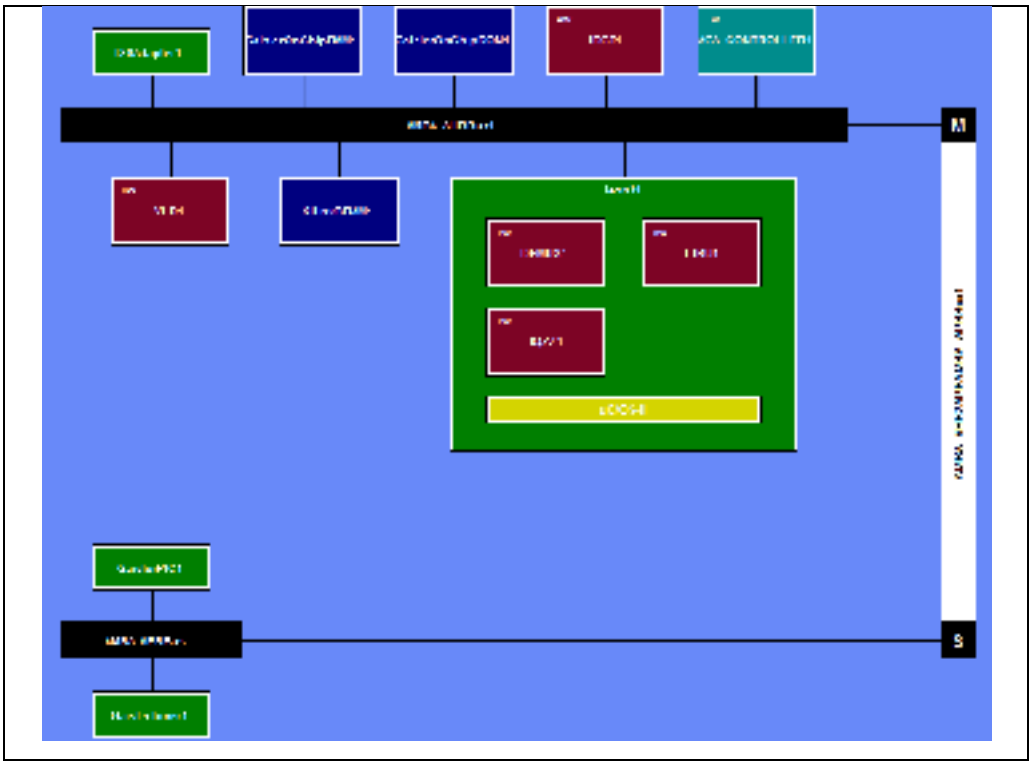


Figure 4.3 Le prototype virtuel à base de processeur Leon

Pour accéder aux données et instructions, le processeur Leon3 est connecté à deux principaux bus (AHB et APB). Le processeur est considéré comme un maître de plus haute priorité sur le bus AHB. Les composants nécessaires pour faire fonctionner ce processeur sont : une minuterie (GaislerTimer), un contrôleur d'interruption (GaislerPIC), un adaptateur d'ISS (ISSadapter), une mémoire ROM (GaislerOnChipROM) et une mémoire RAM (GaislerOnChipRAM). Ces composants sont instanciés automatiquement lors du choix du processeur Leon3 de la librairie de Space Studio. La minuterie et le contrôleur d'interruption sont connectés aux bus APB, qui est à son tour (Bus APB) connecté comme esclave au bus AHB.

Ayant finalisé l'exécution de l'application sur la plateforme virtuelle, les résultats de simulation sont prêts à être analysés.

Temps de simulation :

L'exécution de cette configuration de plateforme sur la machine hôte a pris 2274s, tandis que l'estimation du temps d'exécution sur de réelles plateformes prend 0.604427 s.

Utilisation du processeur :

L'occupation du processeur par les différentes tâches fonctionnelles est présentée sur la Figure 4.4 :

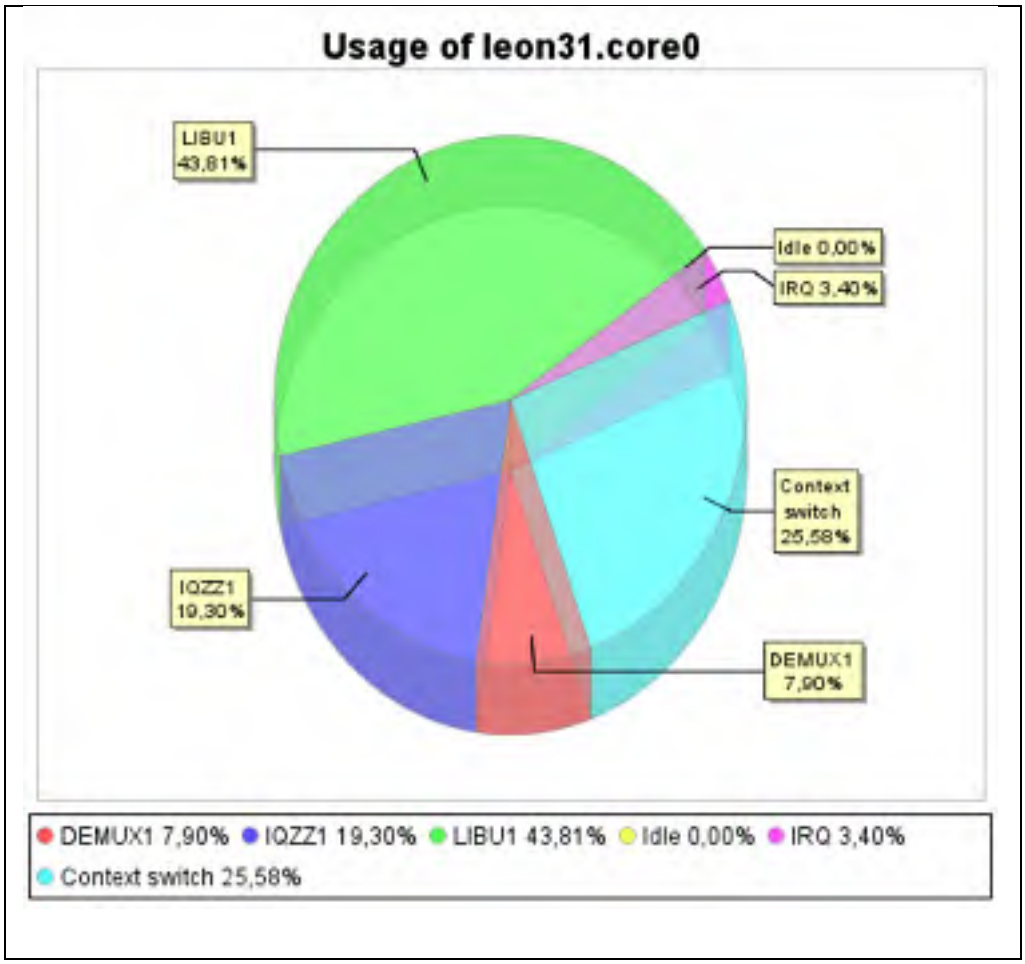


Figure 4.4 Taux d'utilisation du processeur Leon3

Le processeur Leon3 est occupé à 100% de sa capacité de traitement. Les trois threads LIBU, IQZZ et DEMUX couvrent respectivement 43,81%, 19,30% et 7,90% du pourcentage total. Quant au changement de contexte, il prend 25,58% de la capacité du processeur pour faire alterner l'exécution de ces 3 tâches.

Informations sur l'utilisation des bus de communication :

Les deux tableaux suivants présentent l'ensemble d'information sur les échanges effectués à travers le bus AMBA_AHB et AMBA_APB Tableau 4.3 et

Tableau 4.4:

Tableau 4.3 Information sur les transactions
pour le bus AMBA_AHB

Information sur les communications	Valeur
Nombre de bytes	1.1319616 E7
Nombre de transferts	3284029
Nombre d'écriture	2731370
Nombre de lecture	5526959
Débit du bus	149.82265 (Mbps)
Utilisation de la bande passante	4.68 (%)

Tableau 4.4 Charactéristiques des transactions
sur le bus AMBA_APB

Information sur les communications	Valeur
Nombre de bytes	176460
Nombre de transferts	44115
Nombre d'écriture	29410
Nombre de lecture	14705
Débit du bus	2.33557 (Mbps)
Utilisation de la bande passante	0.07 (%)

4.2.1.2 Processeur µBlaze :

Le deuxième type de processeur considéré dans cette exploration architecturale est le µBlaze.
Le modèle de la plateforme virtuelle associé est représenté sur la Figure 4.5.

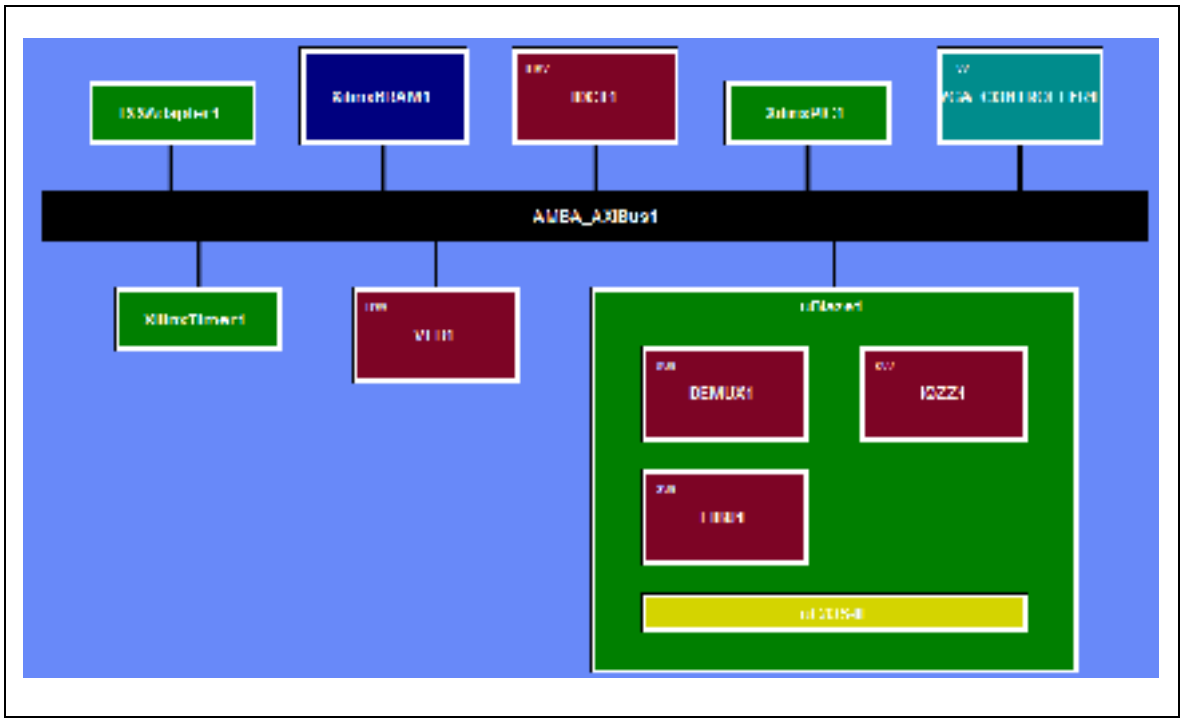


Figure 4.5 Prototype virtuel à base de processeur µBlaze

Comme tout type de processeur, le µBlaze nécessite des périphériques pour contrôler les interruptions et calculer le tic d'horloge selon l'ordre d'ordonnancement (xilinxPIC, xilinxTimer, ISS adapter). Dans cette configuration le processeur est connecté aux deux bus principaux. Le premier est le AMBA_AXI et le deuxième est un bus local à l'intérieur de l'architecture du processeur (LMB local bus memory). Le bus LMB sert à transférer les données et les instructions entre les différents composants de l'architecture.

XilinxTimer : génère des coups d'horloge après qu'un certain nombre de cycles s'est écoulé, il est utilisé généralement pour générer des coups d'horloge périodiques et permet aux tâches d'attendre leurs tours d'exécution.

Temps de simulation :

L'exécution de cette configuration de plateforme sur la machine hôte a pris 1758s, tandis que l'estimation du temps d'exécution sur de réelles plateformes prend 0.870606 s;

Utilisation du processeur :

L'occupation du processeur par les différentes tâches fonctionnelles est présentée sur la Figure 4.6:

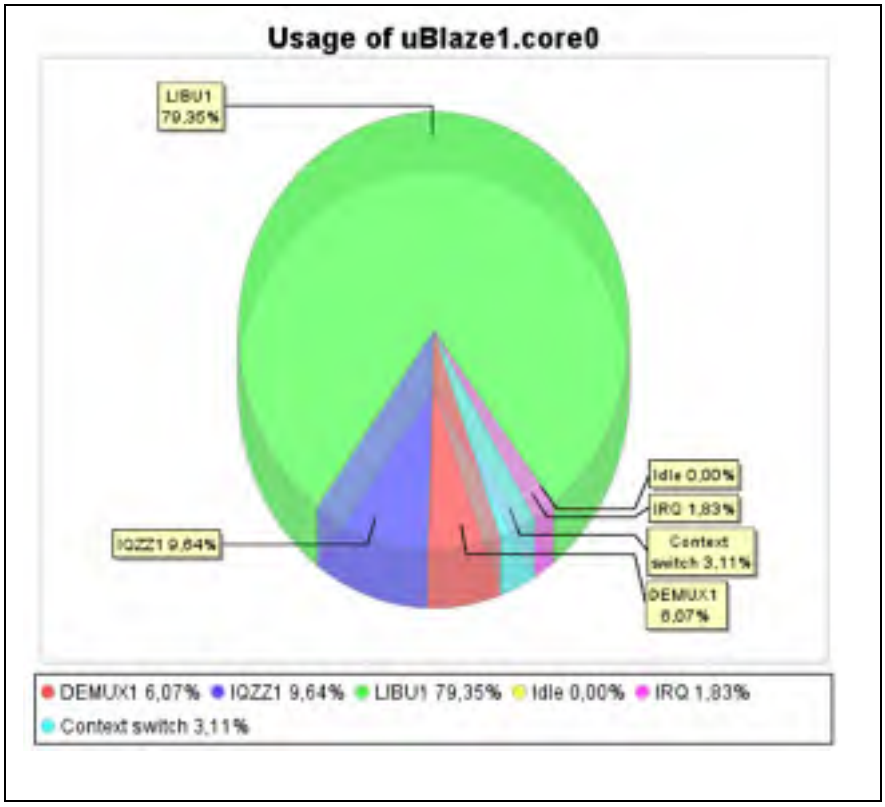


Figure 4.6 Taux d'utilisation du processeur µBlaze

La configuration proposée à base de processeur µBlaze montre que les tâches logicielles occupent toujours le taux maximal de la capacité du processeur.

Information sur l'utilisation du bus :

Le Tableau 4.5 présente l'ensemble d'information sur les échanges effectués à travers le bus AMBA_AXI :

Tableau 4.5 Informations des transactions sur le bus AMBA_AXI

Information sur les communications	**Valeur**
Nombre de bytes	2270052
Nombre de transferts	567513
Nombre d'écriture	316315
Nombre de lecture	251198
Débit du bus	20.85951 (Mbps)
Utilisation de la bande passante	0.65 (%)

D'après les résultats de simulation des deux différentes configurations à base du processeur Leon et μBlaze, la configuration à base du processeur ARM CORTEX-A9 (abordé à la section 3.4.2) présente la meilleure performance. En terme de vitesse de simulation l'architecture à base d'ARM CORTEX-A9 est la plus rapide, car elle prend seulement 0.0539886s, ce qui correspond à la plus basse valeur comparée aux deux autres types de processeur. En terme d'utilisation de ressource matérielle, l'architecture ARM passe moins de transactions à travers le bus de communication par rapport aux deux autres architectures (nombre d'écriture, de lecture, de transfert, etc.). Ainsi pour l'occupation du processeur, le taux d'occupation est le plus faible par rapport aux autres processeurs qui ont atteints leur maximum d'utilisation. Ce qui est tout à fait normal puisque le processeur ARM CORTEX-A9 comprend deux cœurs ce qui peut alléger le traitement de données et accélérer le processus de simulation.

4.2.2 2^{e} facteur d'exploration : fréquence

Ayant choisi le processeur ARM CORTEX-A9 comme composant matériel principal des futures configurations (vu ses performances élevées), une exploration architecturale basée sur

la fréquence du processeur est effectuée pour raffiner davantage les performances du système et accélérer le processus de traitement. Le choix des valeurs de fréquence pour l'exploration architecturale est effectué à travers la modification du paramètre de fréquence de processeur disponible au volet de *configuration manager* de Space Studio. Mentionnons que l'intervalle de valeurs de fréquence offert par Space Studio varie de 330 MHZ à 1000 MHZ. Le Tableau 4.6 montre les différents temps de simulation pour trois valeurs de fréquence :

Tableau 4.6 Comparaison des temps de simulation
en fonction de la fréquence du processeur

Fréquence	**Temps d'exécution**	**Temps total de simulation**
330 MHZ	0.094044 s	747 s
660 MHZ	0.0598991 s	482 s
1000 MHZ	0.0482412 s	338 s

D'après le tableau de comparaison, plus la fréquence est élevée plus le système est rapide en terme de vitesse d'exécution et de traitement de données.

Dans le but de minimiser le temps de simulation (ainsi le temps simulé) pour accélérer l'exécution de l'application du décodage vidéo MJPEG et par la suite augmenter la vitesse de traitement des données, la fréquence la plus adéquate selon cette exploration est 1000 MHZ.

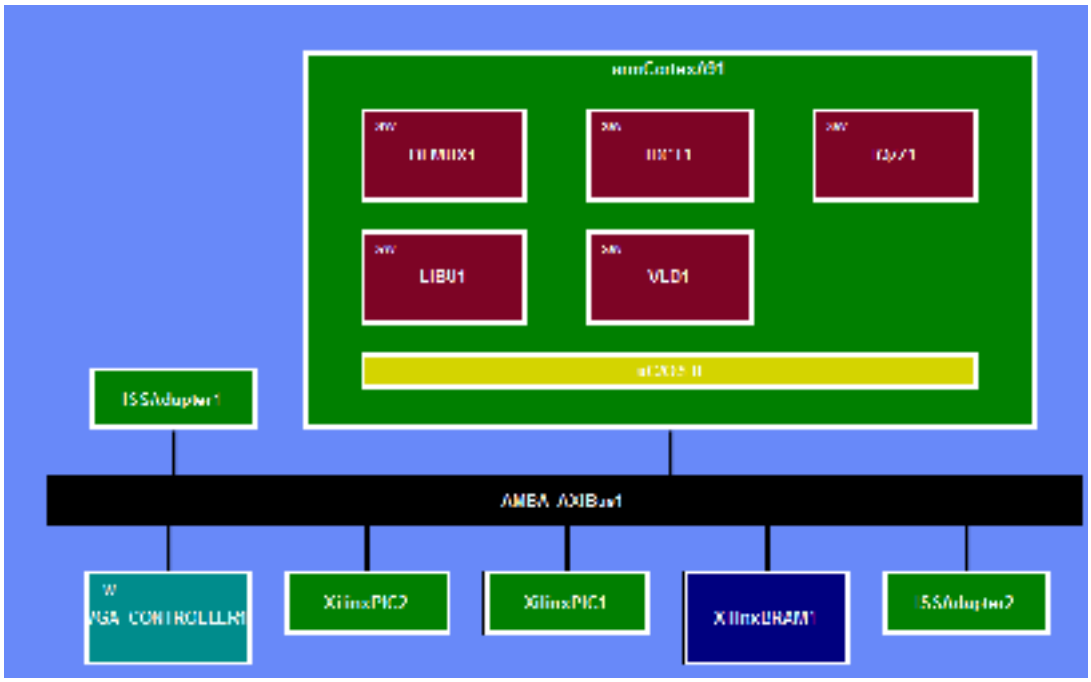
4.2.3 3e facteur d'exploration : partitionnement

Le troisième facteur d'exploration architecturale est celui du partitionnement. Cinq architectures candidates ont été proposées et évaluées dans le but du raffinement et amélioration de performances.

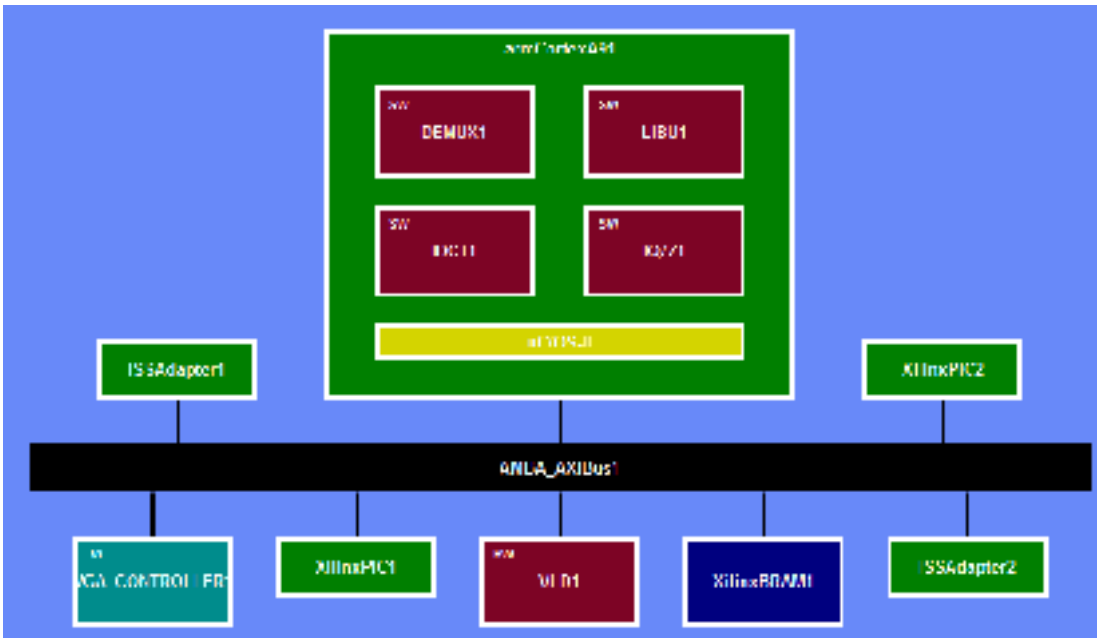
Tableau 4.7 Architectures candidates en fonction du partitionnement

Architecture candidate	Partitionnement logiciel	Partitionnement matériel
Candidat 1	Toutes les tâches	-
Candidat 2	DEMUX1, IQZZ1, LIBU1, IDCT1	VLD1
Candidat 3	DEMUX1, LIBU1, IQZZ1	VLD1, IDCT1
Candidat 4	DEMUX1, LIBU1	VLD1, IDCT1, IQZZ1
Candidat 5	DEMUX1	LIBU1, VLD1, IDCT1, IQZZ1

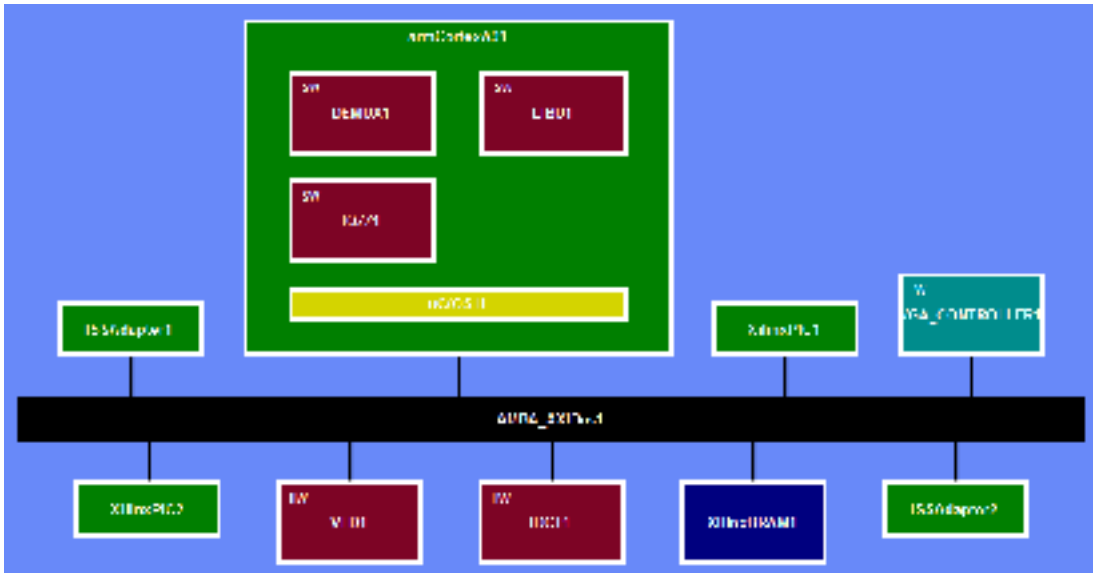
Le Tableau 4.7 présente les cinq solutions proposées de partitionnement. Lorsque les fonctions logicielles sont exécutées sur le processeur, il s'agit d'un partitionnement logiciel. Lorsque les fonctions logicielles sont considérées comme des composants faisant partie de la plateforme d'exécution, dans ce cas il s'agit d'un partitionnement matériel. La Figure 4.7 présente l'ensemble des prototypes virtuels associés à chaque architecture candidate.



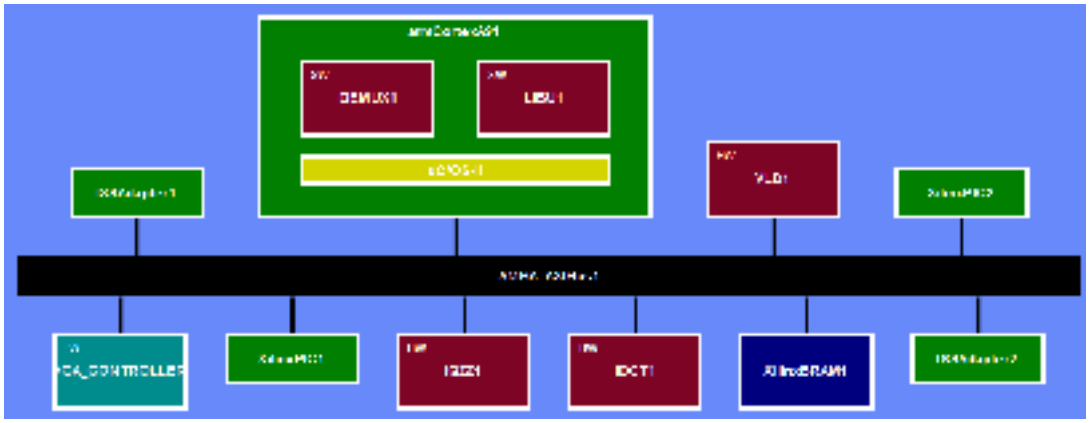
a. 1er architecture candidate



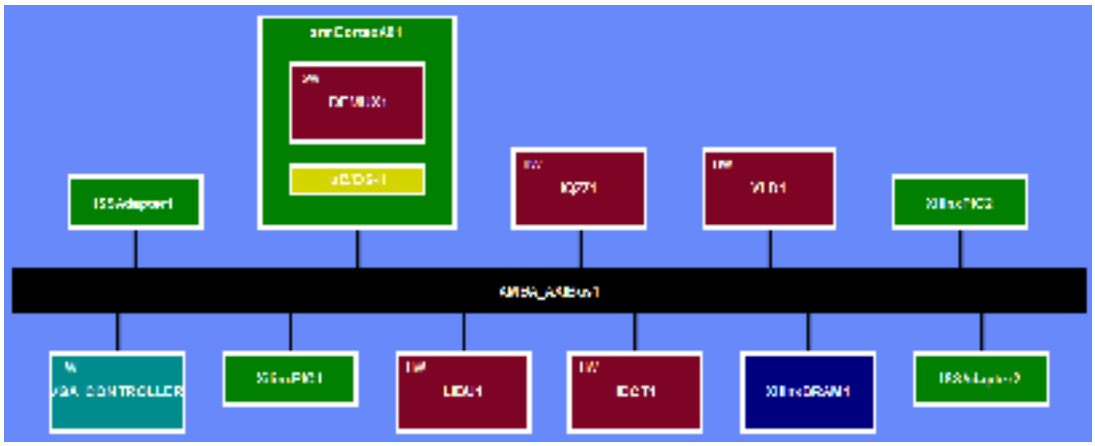
b. 2ème architecture candidate



c. 3ème architecture candidate



d. 4ème architecture candidate



e. 5ème architecture candidate

Figure 4.7 Les prototypes virtuels des différentes architectures candidates

Les différentes architectures candidates de la Figure 4.7 présentent plusieurs configurations de la plateforme du prototype virtuel. Ces configurations sont proposées en se basant sur la modification du partitionnement des tâches logicielles. La première architecture exécute les cinq tâches logicielles sur le processeur ARM Cortex-A9. À la deuxième architecture les quatre tâches logicielles DEMUX, IDCT, LIBU et IQZZ sont assignées au processeur ARM Cortex-A9 et la fonction VLD est désignée comme un module matériel connecté au bus de communication AMBA_AXI. Pour la troisième architecture seules les tâches DEMUX, LIBU et IQZZ sont assignées au processeur, les deux autres fonctions étant connectées en tant que modules matériels au bus de communication. Pour la quatrième architecture, uniquement les tâches DEMUX et LIBU sont attribuées au processeur et finalement pour la 5ème architecture, le processeur Arm Cortex-A9 exécute seulement la fonction DEMUX.

Dans le but de comparer les performances de chacune des cinq configurations proposées, le temps de simulation du décodage vidéo est utilisé pour calculer le nombre d'images traitées par seconde. La Figure 4.8 montre un graphe de traitement de nombre d'image par seconde pour chaque architecture candidate. D'après le graphe, on remarque un accroissement

progressif du nombre d'images traitées par seconde, en allant de la première candidate vers la cinquième candidate. Donc la meilleure performance en termes de capacité de traitement par rapport aux autres architectures est illustrée à travers la 5ème architecture.

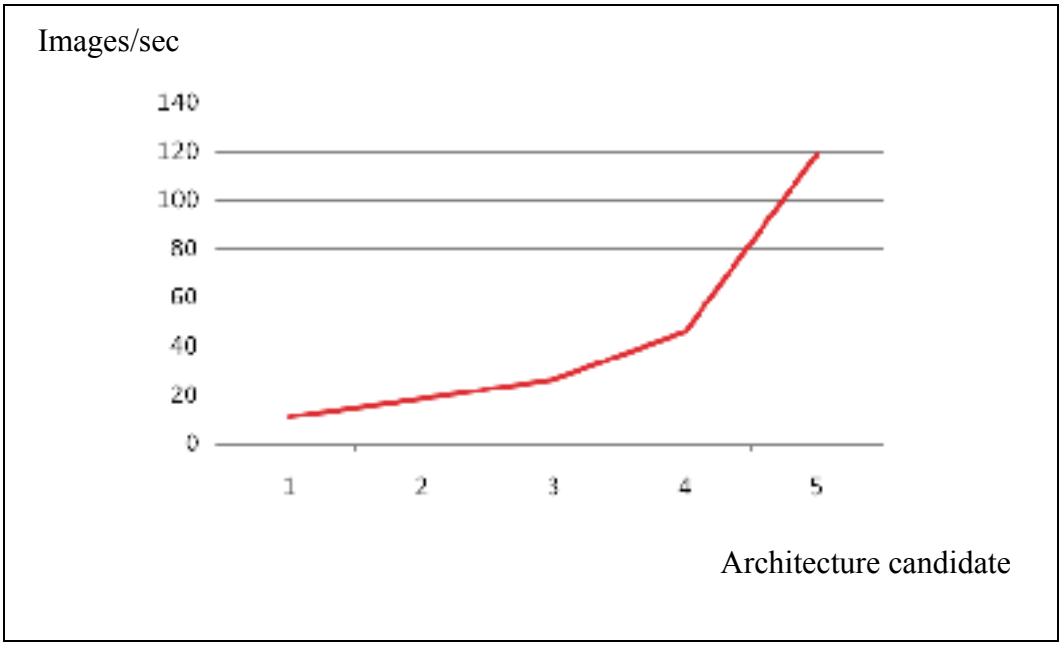


Figure 4.8 Performance de traitement de données
en fonction des architectures candidates

Un autre facteur a été pris en considération pour chacune des architectures candidates. Le Tableau 4.8 présente une comparaison du temps de simulation et du temps simulé des différentes architectures.

Tableau 4.8 Comparaison de temps totale d'exécution et de temps de simulation des architectures candidates

Architecture candidate	Temps d'exécution	Temps total de simulation
Candidat 1	0.0797722 s	914 seconds
Candidat 2	0.0600559 s	875 seconds.
Candidat 3	0.0539886 s	746 seconds.
Candidat 4	0.0334959 s	220 seconds
Candidat 5	0.00993938 s	63 seconds

D'après le Tableau 4.8 le temps de simulation ainsi que le temps simulé diminue graduellement de l'architecture candidate numéro 1 vers l'architecture candidate numéro 5.

Ainsi, le taux d'occupation du processeur a été relevé pour les différentes candidates. Le Tableau 4.9 présente l'utilisation des deux cœurs du processeur ARM CORTEX-A9 pour chacune d'entre eux.

Tableau 4.9 Le taux d'utilisation de processeur pour chaque architecture candidate

Architecture candidate	Utilisation du processeur ARM.core1	Utilisation du processeur ARM.core2
Candidat 1	77	97
Candidat 2	95	79
Candidat 3	100	45
Candidat 4	27	95
Candidat 5	83	0

Encore une fois, d'après le Tableau 4.9, la cinquième candidate présente le plus bas taux d'occupation des deux cœurs (83% - 0%) du processeur Arm Cortex-A9 par rapport aux autres candidates.

4.3 Discussion

Dans les différents cas d'évaluation de performance (nombre d'images traitées par seconde, le temps de simulation, le temps simulé et le taux d'occupation du processeur), la cinquième architecture candidate s'est avérée la plus efficace et performante par rapport aux autres candidates. Elle occupe le plus petit pourcentage du processeur et donc une consommation de puissance plus faible, en plus elle est plus rapide en termes d'exécution et de traitements de données.

En effet, plus le processeur a de nombreuses tâches logicielles à traités plus l'exécution de l'application prend plus de temps à s'exécuter et occupe un pourcentage plus élevé pour le taux total d'utilisation du processeur. Si par exemple la plupart des tâches sont mises en matériel (architecture candidate numéro 5), le processeur ne traitre qu'une seule fonction DEMUX et par conséquent le système est plus rapide. Ceci est dû au ralentissement causé par la simulation des détails de chaque instruction assembleur du modèle ISS du processeur Arm Cortex-A9.

Comme illustré à travers le cas d'étude, l'approche de conception proposée a permis de rencontrer les objectifs fixés au départ. Le flot de conception proposé a été validé par la preuve de concept. En effet une liaison est établie entre le haut niveau du modèle comportemental AADL et le bas niveau de la plateforme matérielle. Cette liaison est bâtie à travers une chaîne de transformation du modèle AADL vers un modèle SystemC. Étant donné que l'environnement Space Studio dispose d'une librairie de périphériques matériels défini en SystemC, l'intégration du modèle SystemC généré au niveau de la chaîne de transformation s'est effectuée facilement et sans aucun problème d'incompatibilité. Au niveau de cet environnement de prototypage virtuel un modèle de plateforme matérielle a été crée, ensuite les fonctions logicielles ont été assignées aux ressources matérielles de la

plateforme. Finalement, les performances du système ont été évaluées pour valider le modèle comportemental (qui se manifeste à ce niveau à travers des fonctions logicielles décrites en SystemC) et analyser ses performances après exécution sur la plateforme matérielle. Finalement, pour améliorer davantage les performances du prototype virtuel, on a procédé à une étape de raffinement pour choisir la meilleure configuration du prototype virtuel.

CONCLUSION

Trois principaux objectifs ont été fixés au début de ce projet de recherche. Il était en premier lieu crucial de développer une certaine compétence et un savoir-faire par rapport aux différents flots de conception et aux techniques d'analyse de systèmes embarqués employés par les approches de conception basées sur les modèles (MBE). Vient ensuite le second objectif où la visée était d'élaborer une approche de conception des systèmes embarqués capable de combler l'écart qui existe encore entre le haut niveau d'abstraction d'un modèle système et l'environnement matériel d'exécution. Finalement, le dernier et troisième objectif avait pour but de valider la méthodologie proposée par une étude de cas, mettant en évidence l'intérêt du concept développé et les différentes contributions.

Dans le cadre du premier objectif, la revue de la littérature a mis en évidence deux travaux de recherche démontrant la pertinence des approches basées sur les modèles pour les futures méthodologies de conception de systèmes embarqués complexes. Ainsi, les différents langages de modélisation, sur lesquels la description du modèle est fondée, ont fait l'objet d'une première exploration. L'importance accordée au langage AADL est due à sa flexibilité et sa simplicité d'utilisation pour la description des modèles et à son usage étendu dans le domaine du MBE. Ensuite, trois approches ont été présentées sur la conception basée sur des modèles qui utilisent principalement le langage AADL pour la modélisation. Différentes techniques d'analyse sont en effet mises en œuvre pour vérifier les fonctionnalités du système et valider le modèle haut niveau.

Pour le deuxième objectif, il s'agissait de proposer une nouvelle méthodologie de modélisation pour la conception des systèmes embarqués. L'approche proposée est suggérée comme une alternative propice pour le processus de validation fonctionnelle et architecturale du modèle haut niveau. À titre d'alternative, la méthodologie adoptée s'appuie sur le langage AADL pour la description du comportement du modèle à haut niveau d'abstraction. Ensuite, le dit modèle AADL est transformé au moyen de la chaîne de transformation automatique ATL en un modèle SystemC. C'est une transformation par le biais de laquelle le haut niveau

d'abstraction et l'environnement de prototypage virtuel Space Studio s'unissent. Ce dernier, qui est un outil se basant sur l'utilisation des modèles SystemC dans toutes les phases de développement, a été considéré comme un outil permettant de créer un prototype virtuel exécutable de la plateforme matérielle. Il permet également de procéder à l'assignation des fonctions logicielles sur les éléments matériels et d'évaluer les performances du système en considérant les paramètres d'exécution sur les ressources matérielles. La simulation du prototype virtuel permet de raffiner davantage certains critères de performances correspondants aux phases de design, à savoir le taux d'occupation du processeur, le temps de simulation et le taux d'information traitée par seconde. Cette étape d'exploration architecturale, cruciale dans le flot de développement, permet de vérifier l'ensemble du système avant sa fabrication et ainsi réduire les coûts de correction des erreurs.

Finalement, le dernier objectif a été atteint en mettant en pratique les différents concepts évoqués par la méthodologie proposée à travers un cas d'étude d'une application de traitement vidéo MJPEG. Un modèle haut niveau, décrivant les différentes fonctionnalités de l'application, est développé en langage de modélisation AADL. Puis, une transformation du modèle AADL vers SystemC est effectuée. Ce modèle SystemC est intégré à l'environnement Space Studio pour attribuer les tâches fonctionnelles de l'application à la plateforme virtuelle créée. Plusieurs résultats de simulation sont collectés pour évaluer les performances du modèle. Des graphiques permettent de visualiser certains critères d'évaluation comme le taux d'occupation du processeur par les différentes fonctions logicielles, les informations liées à l'échange de données sur le bus de communication et les accès mémoire. En phase d'exploration architecturale et grâce aux raffinements effectués, certaines performances se sont améliorées. Ces améliorations ont porté essentiellement sur l'accélération de la vitesse d'exécution, l'évolution de traitement d'images par seconde, la diminution du taux d'occupation du processeur.

En guise de bilan global sur l'approche de conception proposée, nous dirons que l'environnement de prototypage virtuel permet d'offrir aux modèles hauts niveaux un contexte de validation des contraintes d'exécution matérielle. Ainsi la phase d'exploration

architecturale peut apporter d'excellentes améliorations de performances en raffinant certaines spécifications du système. Cependant, il y a d'autres voies à explorer qui peuvent mener à des progrès supplémentaires et qui se présentent comme des perspectives de recherche.

Une de ces perspectives est de modifier la chaîne de transformation ATL du modèle AADL vers SystemC. L'objectif serait d'inclure la transformation automatique des interfaces de communication entre les fonctions logicielles du modèle AADL.

Une autre voie possible comme suite de ce travail consiste en la validation de la preuve de concept de l'approche proposée dans un contexte avionique. Plus particulièrement les systèmes modulaires avioniques puisqu'ils font appel à des techniques de conception et de partitionnement assez complexe pour gérer une grande variété de fonctions logicielles avec un nombre limité de périphériques matériels.

Des perspectives particulièrement prometteuses sont aussi envisageables. Il s'agit du développement d'une chaîne d'outils capable de générer automatiquement un modèle AADL à partir de la configuration finale de la plateforme virtuelle. Cette génération devrait permettre l'analyse des aspects non fonctionnels pour valider plus en détail le système avant sa fabrication.

Les résultats de ce projet de recherche ont fait l'objet d'une publication lors de la conférence ERTS 2014 qui s'est déroulée à Toulouse en France en février 2014. (Benyoussef et al., Février 2014). Cet article introduit les limitations liées aux méthodologies de conception basées sur des modèles. Le flot de conception proposé et les expérimentations associées avec une application de décodage vidéo MJPEG y sont présentés.

ANNEXE I

MODELE AADL

```
package mjpeg_aadl
public
 data Stream end Stream;
  data Huffman end Huffman;
  data QTable end QTable;
  data VLD_Data end VLD_Data;
  data IQZZ_Data end IQZZ_Data;
  data IDCT_Data end IDCT_Data;
subprogram Demux_Function
properties
    Source_Language => (System_C);
    Source_Text     => ("DEMUX.cpp");
    Source_Name     => "DEMUX";
end Demux_Function;
thread Demux
features
    Stream : out event data port Stream;
    Huffman : out event data port Huffman;
    QTable : out event data port QTable;
end Demux;
thread implementation Demux.impl
properties
    Dispatch_Protocol => Periodic;
    Compute_Entrypoint => classifier (Demux_Function);
end Demux.impl;

process Demuxp
 features
    Stream : out event data port Stream;
    Huffman : out event data port Huffman;
    QTable : out event data port QTable;
```

```
    end Demuxp;
 process implementation Demuxp.impl
 subcomponents
   P1: thread mjpeg_aadl::Demux.impl;
    connections
    C_Stream : port  P1.Stream->Stream;
    C_Huffman : port P1.Huffman->Huffman;
    C_QTable : port P1.QTable->QTable;
    end Demuxp.impl;

subprogram VLD_Function
properties
    Source_Language => (System_C);
    Source_Text     => ("VLD.cpp");
    Source_Name     => "VLD";
end VLD_Function;
thread VLD
features
    Stream : in event data port Stream;
    Huffman : in event data port Huffman;
    VLD_Out : out event data port VLD_Data;
end VLD;
thread implementation VLD.impl
properties
    Dispatch_Protocol => Periodic;
    Compute_Entrypoint => classifier (VLD_Function);
end VLD.impl;

process VLDp
 features
    Stream : in event data port Stream;
    Huffman : in event data port Huffman;
    VLD_Out : out event data port VLD_Data;
    end VLDp;
 process implementation VLDp.impl
 subcomponents
```

```
   P2: thread mjpeg_aadl::VLD.impl;
    connections
    cc_Stream : port  Stream-> P2.Stream;
    cc_Huffman : port Huffman->P2.Huffman;
    c_VLD_Out : port P2.VLD_Out->VLD_Out ;
 end VLDp.impl;

subprogram IQZZ_Function
properties
    Source_Language => (System_C);
    Source_Text     => ("IQZZ.cpp");
    Source_Name     => "IQZZ";
end IQZZ_Function;
thread IQZZ
features
    QTable : in event data port QTable;
    VLD_Out : in event data port VLD_Data;
    IQZZ_Out : out event data port IQZZ_Data;
end IQZZ;
thread implementation IQZZ.impl
properties
    Dispatch_Protocol => Periodic;
    Compute_Entrypoint => classifier (IQZZ_Function);
end IQZZ.impl;

 process IQZZp
 features
    QTable : in event data port Qtable;
    VLD_Out : in event data port VLD_Data;
    IQZZ_Out: out event data port IQZZ_Data;
    end IQZZp;
 process implementation IQZZp.impl
 subcomponents
   P3: thread mjpeg_aadl::IQZZ.impl;
    connections
    c_Stream : port  QTable-> P3.QTable;
```

```
    c_Huffman :port  VLD_Out-> P3.VLD_Out;
    c_IQZZ_Out: port P3.IQZZ_Out->IQZZ_Out;
 end IQZZp.impl;
subprogram IDCT_Function
properties
    Source_Language => (System_C);
    Source_Text     => ("IDCT.cpp");
    Source_Name     => "IDCT";
end IDCT_Function;
thread IDCT
features
    IQZZ_Out : in event data port IQZZ_Data;
    IDCT_Out : out event data port IDCT_Data;
end IDCT;
thread implementation IDCT.impl
properties
    Dispatch_Protocol => Periodic;
    Compute_Entrypoint => classifier (IDCT_Function);
end IDCT.impl;

process IDCTp
 features
   IQZZ_Out : in event data port IQZZ_Data;
    IDCT_Out : out event data port IDCT_Data;
    end IDCTp;
 process implementation IDCTp.impl
 subcomponents
   P4: thread mjpeg_aadl::IDCT.impl;
    connections
    c_IQZZ_Out : port IQZZ_Out-> P4.IQZZ_Out;
    c_IDCT_Out:port P4.IDCT_Out-> IDCT_Out;
 end IDCTp.impl;

subprogram LIBU_Function
properties
    Source_Language => (System_C);
```

```
    Source_Text     => ("LIBU.cpp");
    Source_Name     => "LIBU";
end LIBU_Function;
thread LIBU
features
    IDCT_Out : in event data port IDCT_Data;
end LIBU;
thread implementation LIBU.impl
properties
    Dispatch_Protocol => Periodic;
    Compute_Entrypoint => classifier (LIBU_Function);
end LIBU.impl;


process LIBUp
 features
   IDCT_Out : in event data port IDCT_Data;
 end LIBUp;
 process implementation LIBUp.impl
 subcomponents
   P5: thread mjpeg_aadl::LIBU.impl;
    connections
    c_IDCT_Out:port IDCT_Out-> P5.IDCT_Out;
 end LIBUp.impl;

SYSTEM mjpeg
END mjpeg;

SYSTEM IMPLEMENTATION mjpeg.impl
SUBCOMPONENTS

  P1 : PROCESS mjpeg_aadl::Demuxp.impl;
  P2 : PROCESS mjpeg_aadl::VLDp.impl;
  P3 : PROCESS mjpeg_aadl::IQZZp.impl;
  P4 : PROCESS mjpeg_aadl::IDCTp.impl;
  P5 : PROCESS mjpeg_aadl::LIBUp.impl;
```

```
CONNECTIONS

C_Stream :PORT P1.Stream-> P2.Stream;
C_Huffman: PORT P1.Huffman-> P2.Huffman;
C_QTable: PORT P1.QTable->P3.QTable;
c_VLD_Out: PORT P2.VLD_Out->P3.VLD_Out;
c_IQZZ_Out: PORT P3.IQZZ_Out -> P4.IQZZ_Out;
c_IDCT_Out: PORT P4.IDCT_Out-> P5.IDCT_Out;

end mjpeg.impl;
end mjpeg_aadl;
```

ANNEXE II

CODE DE TRANSFORMATION

```
<//AADL specification(mjpeg_aadl.aadl) translated into C++
(/mjpeg/aadl/packages\mjpeg_aadl.h)
#include "aadl.h"
namespace AADL_mjpeg_aadl {

  namespace mjpeg_aadl {
    class stream {
      public:
        stream {
        }
    };
  }

  namespace mjpeg_aadl {
    class huffman {
      public:
        huffman {
        }
    };
  }

  namespace mjpeg_aadl {
    class qtable {
      public:
        qtable {
        }
    };
  }

  namespace mjpeg_aadl {
    class vld_data {
      public:
        vld_data {
        }
    };
  }

  namespace mjpeg_aadl {
    class iqzz_data {
      public:
        iqzz_data {
        }
    };
  }

  namespace mjpeg_aadl {
```

```
  class idct_data {
    public:
      idct_data {
      }
  };
}

namespace mjpeg_aadl {
  class demux : public AADL::threadType {
    public:
      AADL::eventDataPort_out<mjpeg_aadl::stream> stream;
      AADL::eventDataPort_out<mjpeg_aadl::huffman> huffman;
      AADL::eventDataPort_out<mjpeg_aadl::qtable> qtable;
    public:
      demux(AADL::moduleName name) : AADL::threadType(name) {
      }
  };
}

namespace mjpeg_aadl {
  class vld : public AADL::threadType {
    public:
      AADL::eventDataPort_in<mjpeg_aadl::stream> stream;
      AADL::eventDataPort_in<mjpeg_aadl::huffman> huffman;
      AADL::eventDataPort_out<mjpeg_aadl::vld_data> vld_out;
    public:
      vld(AADL::moduleName name) : AADL::threadType(name) {
      }
  };
}

namespace mjpeg_aadl {
  class iqzz : public AADL::threadType {
    public:
      AADL::eventDataPort_in<mjpeg_aadl::qtable> qtable;
      AADL::eventDataPort_in<mjpeg_aadl::vld_data> vld_out;
      AADL::eventDataPort_out<mjpeg_aadl::iqzz_data> iqzz_out;
    public:
      iqzz(AADL::moduleName name) : AADL::threadType(name) {
      }
  };
}

namespace mjpeg_aadl {
  class idct : public AADL::threadType {
    public:
      AADL::eventDataPort_in<mjpeg_aadl::iqzz_data> iqzz_out;
      AADL::eventDataPort_out<mjpeg_aadl::idct_data> idct_out;
    public:
      idct(AADL::moduleName name) : AADL::threadType(name) {
      }
  };
}
```

```
namespace mjpeg_aadl {
  class libu : public AADL::threadType {
    public:
      AADL::eventDataPort_in<mjpeg_aadl::idct_data> idct_out;
    public:
      libu(AADL::moduleName name) : AADL::threadType(name) {
      }
  };
}

namespace mjpeg_aadl {
  class demux_DOT_impl : public mjpeg_aadl::demux {
    public:
      demux_DOT_impl(AADL::moduleName name) : demux(name) {
      }
  };
}

namespace mjpeg_aadl {
  class demuxp : public AADL::processType {
    public:
      AADL::eventDataPort_out<mjpeg_aadl::stream> stream;
      AADL::eventDataPort_out<mjpeg_aadl::huffman> huffman;
      AADL::eventDataPort_out<mjpeg_aadl::qtable> qtable;
    public:
      demuxp(AADL::moduleName name) : AADL::processType(name) {
      }
  };
}

namespace mjpeg_aadl {
  class vld_DOT_impl : public mjpeg_aadl::vld {
    public:
      vld_DOT_impl(AADL::moduleName name) : vld(name) {
      }
  };
}

namespace mjpeg_aadl {
  class vldp : public AADL::processType {
    public:
      AADL::eventDataPort_in<mjpeg_aadl::stream> stream;
      AADL::eventDataPort_in<mjpeg_aadl::huffman> huffman;
      AADL::eventDataPort_out<mjpeg_aadl::vld_data> vld_out;
    public:
      vldp(AADL::moduleName name) : AADL::processType(name) {
      }
  };
}

namespace mjpeg_aadl {
  class iqzz_DOT_impl : public mjpeg_aadl::iqzz {
```

```
    public:
      iqzz_DOT_impl(AADL::moduleName name) : iqzz(name) {
      }
  };
}

namespace mjpeg_aadl {
  class iqzzp : public AADL::processType {
    public:
      AADL::eventDataPort_in<mjpeg_aadl::qtable> qtable;
      AADL::eventDataPort_in<mjpeg_aadl::vld_data> vld_out;
      AADL::eventDataPort_out<mjpeg_aadl::iqzz_data> iqzz_out;
    public:
      iqzzp(AADL::moduleName name) : AADL::processType(name) {
      }
  };
}

namespace mjpeg_aadl {
  class idct_DOT_impl : public mjpeg_aadl::idct {
    public:
      idct_DOT_impl(AADL::moduleName name) : idct(name) {
      }
  };
}

namespace mjpeg_aadl {
  class idctp : public AADL::processType {
    public:
      AADL::eventDataPort_in<mjpeg_aadl::iqzz_data> iqzz_out;
      AADL::eventDataPort_out<mjpeg_aadl::idct_data> idct_out;
    public:
      idctp(AADL::moduleName name) : AADL::processType(name) {
      }
  };
}

namespace mjpeg_aadl {
  class libu_DOT_impl : public mjpeg_aadl::libu {
    public:
      libu_DOT_impl(AADL::moduleName name) : libu(name) {
      }
  };
}

namespace mjpeg_aadl {
  class libup : public AADL::processType {
    public:
      AADL::eventDataPort_in<mjpeg_aadl::idct_data> idct_out;
    public:
      libup(AADL::moduleName name) : AADL::processType(name) {
      }
  };
```

```
}

namespace mjpeg_aadl {
  class demuxp_DOT_impl : public mjpeg_aadl::demuxp {
    public:
      mjpeg_aadl::demux_DOT_impl p1;
    public:
    public:
      demuxp_DOT_impl(AADL::moduleName name) : demuxp(name), p1("p1") {
        p1.stream(stream);
        p1.huffman(huffman);
        p1.qtable(qtable);
      }
  };
}

namespace mjpeg_aadl {
  class vldp_DOT_impl : public mjpeg_aadl::vldp {
    public:
      mjpeg_aadl::vld_DOT_impl p2;
    public:
    public:
      vldp_DOT_impl(AADL::moduleName name) : vldp(name), p2("p2") {
        p2.stream(stream);
        p2.huffman(huffman);
        p2.vld_out(vld_out);
      }
  };
}

namespace mjpeg_aadl {
  class iqzzp_DOT_impl : public mjpeg_aadl::iqzzp {
    public:
      mjpeg_aadl::iqzz_DOT_impl p3;
    public:
    public:
      iqzzp_DOT_impl(AADL::moduleName name) : iqzzp(name), p3("p3") {
        p3.qtable(qtable);
        p3.vld_out(vld_out);
        p3.iqzz_out(iqzz_out);
      }
  };
}

namespace mjpeg_aadl {
  class idctp_DOT_impl : public mjpeg_aadl::idctp {
    public:
      mjpeg_aadl::idct_DOT_impl p4;
    public:
    public:
      idctp_DOT_impl(AADL::moduleName name) : idctp(name), p4("p4") {
        p4.iqzz_out(iqzz_out);
        p4.idct_out(idct_out);
```

```
        }
    };
  }

  namespace mjpeg_aadl {
    class libup_DOT_impl : public mjpeg_aadl::libup {
      public:
        mjpeg_aadl::libu_DOT_impl p5;
      public:
      public:
        libup_DOT_impl(AADL::moduleName name) : libup(name), p5("p5") {
          p5.idct_out(idct_out);
        }
    };
  }

  namespace mjpeg_aadl {
    class mjpeg : public AADL::systemType {
      public:
        mjpeg(AADL::moduleName name) : AADL::systemType(name) {
        }
    };
  }

  namespace mjpeg_aadl {
    class demux_function : public AADL::subprogramType {
      public:
        demux_function(AADL::moduleName name) : AADL::subprogramType(name)
{
        }
    };
  }

  namespace mjpeg_aadl {
    class vld_function : public AADL::subprogramType {
      public:
        vld_function(AADL::moduleName name) : AADL::subprogramType(name) {
        }
    };
  }

  namespace mjpeg_aadl {
    class iqzz_function : public AADL::subprogramType {
      public:
        iqzz_function(AADL::moduleName name) : AADL::subprogramType(name)
{
        }
    };
  }

  namespace mjpeg_aadl {
    class idct_function : public AADL::subprogramType {
      public:
```

```
        idct_function(AADL::moduleName name) : AADL::subprogramType(name)
{
        }
    };
  }

  namespace mjpeg_aadl {
    class libu_function : public AADL::subprogramType {
      public:
        libu_function(AADL::moduleName name) : AADL::subprogramType(name)
{
        }
    };
  }

  namespace mjpeg_aadl {
    class mjpeg_DOT_impl : public mjpeg_aadl::mjpeg {
      public:
        mjpeg_aadl::demuxp_DOT_impl p1;
        mjpeg_aadl::vldp_DOT_impl p2;
        mjpeg_aadl::iqzzp_DOT_impl p3;
        mjpeg_aadl::idctp_DOT_impl p4;
        mjpeg_aadl::libup_DOT_impl p5;
      public:
        AADL::channel<mjpeg_aadl::stream> c_stream;
        AADL::channel<mjpeg_aadl::huffman> c_huffman;
        AADL::channel<mjpeg_aadl::qtable> c_qtable;
        AADL::channel<mjpeg_aadl::vld_data> c_vld_out;
        AADL::channel<mjpeg_aadl::iqzz_data> c_iqzz_out;
        AADL::channel<mjpeg_aadl::idct_data> c_idct_out;
      public:
        mjpeg_DOT_impl(AADL::moduleName name) : mjpeg(name), p1("p1"),
p2("p2"), p3("p3"), p4("p4"), p5("p5") {
          p1.stream(c_stream);
          p2.stream(c_stream);
          p1.huffman(c_huffman);
          p2.huffman(c_huffman);
          p1.qtable(c_qtable);
          p3.qtable(c_qtable);
          p2.vld_out(c_vld_out);
          p3.vld_out(c_vld_out);
          p3.iqzz_out(c_iqzz_out);
          p4.iqzz_out(c_iqzz_out);
          p4.idct_out(c_idct_out);
          p5.idct_out(c_idct_out);
        }
    };
  }
} // end of namespace AADL_mjpeg_aadl
```

ANNEXE III

CODE BUS MAPPING

```
</////////////////////////////////////////////////////////////////
//
// Filename       : bus_mapping.cpp
//
////////////////////////////////////////////////////////////////
//  Copyright 2011 - Space Codesign Systems, Inc.
//  All rights reserved.
////////////////////////////////////////////////////////////////
#define BUS_MAPPING
#include "bus_mapping.h"


//////////////////////////////////////////////////////
///
///      Arrays   (those arrays are generated by SpaceStudio, please do not
modify)
///
//////////////////////////////////////////////////////
AddressBinding AMBA_AXIBus1_address[] =
{
         {  5, { 11, 0x42005000, 0x42005fff}, 0 },    //
AMBA_AXIBus1_SlaveAdapter_XilinxPIC1
         {  9, {  3, 0x42002000, 0x42002fff}, 0 },    //
AMBA_AXIBus1_SlaveAdapter_adap_slave_LIBU1_AMBA_AXIBus1_WriteInterface
         { 10, {  4, 0x42003000, 0x42003fff}, 0 },    //
AMBA_AXIBus1_SlaveAdapter_adap_slave_VLD1_AMBA_AXIBus1_WriteInterface
         {  1, { 10, 0x40000000, 0x40ffffff}, 0 },    //
AMBA_AXIBus1_SlaveAdapter_ISSAdapter1
         {  3, {  5, 0x43000000, 0x43ffffff}, 0 },    //
AMBA_AXIBus1_SlaveAdapter_VGA_CONTROLLER1
         {  7, {  1, 0x42000000, 0x42000fff}, 0 },    //
AMBA_AXIBus1_SlaveAdapter_adap_slave_IDCT1_AMBA_AXIBus1_WriteInterface
         {  8, {  2, 0x42001000, 0x42001fff}, 0 },    //
AMBA_AXIBus1_SlaveAdapter_adap_slave_IQZZ1_AMBA_AXIBus1_WriteInterface
         {  0, { 15, 0x42004000, 0x42004fff}, 0 },    //
AMBA_AXIBus1_SlaveAdapter_DebugModule1
         {  6, { 13, 0x42006000, 0x42006fff}, 0 },    //
AMBA_AXIBus1_SlaveAdapter_XilinxPIC2
         {  4, { 14, 0x42010000, 0x4201ffff}, 0 },    //
AMBA_AXIBus1_SlaveAdapter_XilinxBRAM1
         {  2, { 12, 0x41000000, 0x41ffffff}, 0 }     //
AMBA_AXIBus1_SlaveAdapter_ISSAdapter2
};
AddressInfo AMBA_AXIBus1_addressinfo = { 11, AMBA_AXIBus1_address};
```

```
ComponentInfo Global_address[] =
{
        { 10, -1,    SPACE_ISS_ADAPTER},    // ISSAdapter1
        { 12, -1,    SPACE_ISS_ADAPTER},    // ISSAdapter2
        { 15, -1,         SPACE_DEVICE},    // DebugModule1
        { 14,  0,         SPACE_DEVICE},    // XilinxBRAM1
        {  0,  0,         SPACE_MODULE},    // DEMUX1
        {  1, -1,         SPACE_MODULE},    // IDCT1
        {  2, -1,         SPACE_MODULE},    // IQZZ1
        {  3, -1,         SPACE_MODULE},    // LIBU1
        {  4, -1,         SPACE_MODULE},    // VLD1
        { 11, -1,         SPACE_DEVICE},    // XilinxPIC1
        { 13, -1,         SPACE_DEVICE},    // XilinxPIC2
        {  5, -1,         SPACE_DEVICE}     // VGA_CONTROLLER1
};


int IdByPriority[] =
{
        -1, -1, 3, 1, 2, 4, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -
1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -
1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -
1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -
1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -
1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -
1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1
};

int Global_nbComponents = 12;


#undef BUS_MAPPING
```

ANNEXE IV

CODE MAIN

```
<////////////////////////////////////////////////////////////////////////
///////
//
// Filename : main_all_HW_Less_DEMUX1.cpp
//
// Author : SpaceStudio generation engine
//
// Warning: This file content will be overwritten by the next generation
process.
//
////////////////////////////////////////////////////////////////////////
//////

#include "main_all_HW_Less_DEMUX1.h"
#include <getopt.h>
#include <cstdlib>
#include <time.h>

int sc_main(int arg_count, char **arg_value){

        sc_report_handler::suppress(SC_THROW);
        sc_report_handler::suppress(SC_ABORT);
        sc_report_handler::set_actions("object already exists",
SC_DO_NOTHING);
        sc_report_handler::set_actions(SC_ERROR, SC_DEFAULT_ERROR_ACTIONS
| SC_STOP);
        sc_report_handler::set_actions(SC_FATAL, SC_DEFAULT_FATAL_ACTIONS
| SC_STOP);

        // Variables for simulation time evaluation
        time_t simulation_time_begin = 0;
        time_t simulation_time_end = 0;

        // Variables for sorting the arguments
        int simulation_time = -1;
        int sim_units = 2; //SC_NS
        int DEBUGGER_PORT_KEY[] = { 0 };

        // Initialize the SpaceLib components and prepare for simulation
        if(SpaceLibInitialize(arg_count, arg_value, simulation_time,
sim_units, DEBUGGER_PORT_KEY ) == -1 )
                return(0);
        AXIMasterAdapter
AMBA_AXIBus1_MasterAdapter_adap_master_IQZZ1_ChannelIFPort("AMBA_AXIBus1_M
asterAdapter_adap_master_IQZZ1_ChannelIFPort", 10, SC_NS, IQZZ_ID, false);
```

```
        AXIMasterAdapter
AMBA_AXIBus1_MasterAdapter_adap_master_LIBU1_ChannelIFPort("AMBA_AXIBus1_M
asterAdapter_adap_master_LIBU1_ChannelIFPort", 10, SC_NS, LIBU_ID, false);
        AXIMasterAdapter
AMBA_AXIBus1_MasterAdapter_adap_master_VLD1_ChannelIFPort("AMBA_AXIBus1_Ma
sterAdapter_adap_master_VLD1_ChannelIFPort", 10, SC_NS, VLD_ID, false);
        AXIMasterAdapter
AMBA_AXIBus1_MasterAdapter_armCortexA91_DBusPort("AMBA_AXIBus1_MasterAdapt
er_armCortexA91_DBusPort", 10, SC_NS, ARMCORTEXA91_ID, false);
        AXIMasterAdapter
AMBA_AXIBus1_MasterAdapter_armCortexA91_IBusPort("AMBA_AXIBus1_MasterAdapt
er_armCortexA91_IBusPort", 10, SC_NS, ARMCORTEXA91_ID, false);
        AXISlaveAdapter
AMBA_AXIBus1_SlaveAdapter_DebugModule1("AMBA_AXIBus1_SlaveAdapter_DebugMod
ule1", 10, SC_NS, DEBUGMODULE1_ID, false);
        AXISlaveAdapter
AMBA_AXIBus1_SlaveAdapter_ISSAdapter1("AMBA_AXIBus1_SlaveAdapter_ISSAdapte
r1", 10, SC_NS, ISSADAPTER1_ID, false);
        AXISlaveAdapter
AMBA_AXIBus1_SlaveAdapter_ISSAdapter2("AMBA_AXIBus1_SlaveAdapter_ISSAdapte
r2", 10, SC_NS, ISSADAPTER2_ID, false);
        AXISlaveAdapter
AMBA_AXIBus1_SlaveAdapter_VGA_CONTROLLER1("AMBA_AXIBus1_SlaveAdapter_VGA_C
ONTROLLER1", 10, SC_NS, VGA_CONTROLLER_ID, false);
        AXISlaveAdapter
AMBA_AXIBus1_SlaveAdapter_XilinxBRAM1("AMBA_AXIBus1_SlaveAdapter_XilinxBRA
M1", 10, SC_NS, MJPEGRAM_ID, false);
        AXISlaveAdapter
AMBA_AXIBus1_SlaveAdapter_XilinxPIC1("AMBA_AXIBus1_SlaveAdapter_XilinxPIC1
", 10, SC_NS, XILINXPIC1_ID, false);
        AXISlaveAdapter
AMBA_AXIBus1_SlaveAdapter_XilinxPIC2("AMBA_AXIBus1_SlaveAdapter_XilinxPIC2
", 10, SC_NS, XILINXPIC2_ID, false);
        AXISlaveAdapter
AMBA_AXIBus1_SlaveAdapter_adap_slave_IDCT1_AMBA_AXIBus1_WriteInterface("AM
BA_AXIBus1_SlaveAdapter_adap_slave_IDCT1_AMBA_AXIBus1_WriteInterface", 10,
SC_NS, IDCT_ID, false);
        AXISlaveAdapter
AMBA_AXIBus1_SlaveAdapter_adap_slave_IQZZ1_AMBA_AXIBus1_WriteInterface("AM
BA_AXIBus1_SlaveAdapter_adap_slave_IQZZ1_AMBA_AXIBus1_WriteInterface", 10,
SC_NS, IQZZ_ID, false);
        AXISlaveAdapter
AMBA_AXIBus1_SlaveAdapter_adap_slave_LIBU1_AMBA_AXIBus1_WriteInterface("AM
BA_AXIBus1_SlaveAdapter_adap_slave_LIBU1_AMBA_AXIBus1_WriteInterface", 10,
SC_NS, LIBU_ID, false);
        AXISlaveAdapter
AMBA_AXIBus1_SlaveAdapter_adap_slave_VLD1_AMBA_AXIBus1_WriteInterface("AMB
A_AXIBus1_SlaveAdapter_adap_slave_VLD1_AMBA_AXIBus1_WriteInterface", 10,
SC_NS, VLD_ID, false);
        ISSAdapter ISSAdapter1("ISSAdapter1", 10, SC_NS, ISSADAPTER1_ID,
false);
        ISSAdapter ISSAdapter2("ISSAdapter2", 10, SC_NS, ISSADAPTER2_ID,
false);
```

```
        ModuleMasterAdapter adap_master_IQZZ1("adap_master_IQZZ1", 10,
SC_NS, IQZZ_ID, IQZZ_PRIO, (void*)&AMBA_AXIBus1_addressinfo, false);
        ModuleMasterAdapter adap_master_LIBU1("adap_master_LIBU1", 10,
SC_NS, LIBU_ID, LIBU_PRIO, (void*)&AMBA_AXIBus1_addressinfo, false);
        ModuleMasterAdapter adap_master_VLD1("adap_master_VLD1", 10,
SC_NS, VLD_ID, VLD_PRIO, (void*)&AMBA_AXIBus1_addressinfo, false);
        ModuleSlaveAdapter
adap_slave_IDCT1_AMBA_AXIBus1_WriteInterface("adap_slave_IDCT1_AMBA_AXIBus
1_WriteInterface", IDCT_ID, false);
        ModuleSlaveAdapter
adap_slave_IQZZ1_AMBA_AXIBus1_WriteInterface("adap_slave_IQZZ1_AMBA_AXIBus
1_WriteInterface", IQZZ_ID, false);
        ModuleSlaveAdapter
adap_slave_LIBU1_AMBA_AXIBus1_WriteInterface("adap_slave_LIBU1_AMBA_AXIBus
1_WriteInterface", LIBU_ID, false);
        ModuleSlaveAdapter
adap_slave_VLD1_AMBA_AXIBus1_WriteInterface("adap_slave_VLD1_AMBA_AXIBus1_
WriteInterface", VLD_ID, false);
        AMBA_AXIBus AMBA_AXIBus1("AMBA_AXIBus1", 10, SC_NS,
AMBA_AXIBUS1_ID, AXI_INTERCONNECT_SHARED_ADDRESS_CROSSBAR_DATA,
(void*)&AMBA_AXIBus1_addressinfo, false);
        DebugModule DebugModule1("DebugModule1", false);
        SDLRegister
DL_IDCT1_TO_LIBU1_SDLRegister1("DL_IDCT1_TO_LIBU1_SDLRegister1", 10,
SC_NS, DL_IDCT1_TO_LIBU1_SDLREGISTER1_ID, 64,
SDLREGISTER_LATENCY_HARDWARE, false);
        SDLRegister
DL_IQZZ1_TO_IDCT1_SDLRegister1("DL_IQZZ1_TO_IDCT1_SDLRegister1", 10,
SC_NS, DL_IQZZ1_TO_IDCT1_SDLREGISTER1_ID, 64,
SDLREGISTER_LATENCY_HARDWARE, false);
        SDLRegister
DL_VLD1_TO_IQZZ1_SDLRegister1("DL_VLD1_TO_IQZZ1_SDLRegister1", 10, SC_NS,
DL_VLD1_TO_IQZZ1_SDLREGISTER1_ID, 64, SDLREGISTER_LATENCY_HARDWARE,
false);
        ModuleSlaveAdapterFIFOWrite
fifo_adap_slave_IDCT1_AMBA_AXIBus1_WriteInterface0("fifo_adap_slave_IDCT1_
AMBA_AXIBus1_WriteInterface0", 10, SC_NS, 64, false);
        ModuleSlaveAdapterFIFOWrite
fifo_adap_slave_IQZZ1_AMBA_AXIBus1_WriteInterface0("fifo_adap_slave_IQZZ1_
AMBA_AXIBus1_WriteInterface0", 10, SC_NS, 64, false);
        ModuleSlaveAdapterFIFOWrite
fifo_adap_slave_LIBU1_AMBA_AXIBus1_WriteInterface0("fifo_adap_slave_LIBU1_
AMBA_AXIBus1_WriteInterface0", 10, SC_NS, 64, false);
        ModuleSlaveAdapterFIFOWrite
fifo_adap_slave_VLD1_AMBA_AXIBus1_WriteInterface0("fifo_adap_slave_VLD1_AM
BA_AXIBus1_WriteInterface0", 10, SC_NS, 64, false);
        armCortexA9 armCortexA91("armCortexA91", ARMCORTEXA91_ID,
DEBUGGER_PORT_KEY[0], false, 800000000, (void*)&AMBA_AXIBus1_addressinfo,
false);
        XilinxBRAM XilinxBRAM1("XilinxBRAM1", 10, SC_NS, MJPEGRAM_ID,
XILINXBRAM1_SIZE, "../../../../../import/txt/memoire.txt",
XILINX_BRAM_READ_LATENCY, XILINX_BRAM_WRITE_LATENCY, false, false);
        IDCT IDCT1("IDCT1", 10, SC_NS, IDCT_ID, IDCT_PRIO, false);
```

```
        IQZZ IQZZ1("IQZZ1", 10, SC_NS, IQZZ_ID, IQZZ_PRIO, false);
        LIBU LIBU1("LIBU1", 10, SC_NS, LIBU_ID, LIBU_PRIO, false);
        VLD VLD1("VLD1", 10, SC_NS, VLD_ID, VLD_PRIO, false);
        XilinxPIC XilinxPIC1("XilinxPIC1", 10, SC_NS, XILINXPIC1_ID,
false);
        XilinxPIC XilinxPIC2("XilinxPIC2", 10, SC_NS, XILINXPIC2_ID,
false);
        ResetManager reset_manager_1("reset_manager_1", 10, SC_NS, true);
        sc_clock SysClock("SysClock", 10, SC_NS, 0.5);
        sc_signal< bool > sReset;
        sc_signal< bool > sXilinxPIC1_IRQOutPort0;
        sc_signal< bool > sXilinxPIC2_IRQOutPort0;
        sc_signal< bool >
sfifo_adap_slave_IDCT1_AMBA_AXIBus1_WriteInterface0_IRQHasRoomOutPort0;
        sc_signal< bool >
sfifo_adap_slave_IQZZ1_AMBA_AXIBus1_WriteInterface0_IRQHasRoomOutPort0;
        sc_signal< bool >
sfifo_adap_slave_LIBU1_AMBA_AXIBus1_WriteInterface0_IRQHasRoomOutPort0;
        sc_signal< bool >
sfifo_adap_slave_VLD1_AMBA_AXIBus1_WriteInterface0_IRQHasRoomOutPort0;
        VGA_CONTROLLER VGA_CONTROLLER1("VGA_CONTROLLER1", 10, SC_NS,
VGA_CONTROLLER_ID, false);
        sReset = true;

        // Binding
        AMBA_AXIBus1_SlaveAdapter_DebugModule1.SlaveIFPort(DebugModule1);
        AMBA_AXIBus1_SlaveAdapter_ISSAdapter1.SlaveIFPort(ISSAdapter1);
        AMBA_AXIBus1_SlaveAdapter_ISSAdapter2.SlaveIFPort(ISSAdapter2);
        AMBA_AXIBus1_SlaveAdapter_VGA_CONTROLLER1.SlaveIFPort(VGA_CONTROLL
ER1);
        AMBA_AXIBus1_SlaveAdapter_XilinxBRAM1.SlaveIFPort(XilinxBRAM1);
        AMBA_AXIBus1_SlaveAdapter_XilinxPIC1.SlaveIFPort(XilinxPIC1);
        AMBA_AXIBus1_SlaveAdapter_XilinxPIC2.SlaveIFPort(XilinxPIC2);
        AMBA_AXIBus1_SlaveAdapter_adap_slave_IDCT1_AMBA_AXIBus1_WriteInter
face.SlaveIFPort(adap_slave_IDCT1_AMBA_AXIBus1_WriteInterface);
        AMBA_AXIBus1_SlaveAdapter_adap_slave_IQZZ1_AMBA_AXIBus1_WriteInter
face.SlaveIFPort(adap_slave_IQZZ1_AMBA_AXIBus1_WriteInterface);
        AMBA_AXIBus1_SlaveAdapter_adap_slave_LIBU1_AMBA_AXIBus1_WriteInter
face.SlaveIFPort(adap_slave_LIBU1_AMBA_AXIBus1_WriteInterface);
        AMBA_AXIBus1_SlaveAdapter_adap_slave_VLD1_AMBA_AXIBus1_WriteInterf
ace.SlaveIFPort(adap_slave_VLD1_AMBA_AXIBus1_WriteInterface);
        adap_slave_IDCT1_AMBA_AXIBus1_WriteInterface.FifoIFPort[0](fifo_ad
ap_slave_IDCT1_AMBA_AXIBus1_WriteInterface0.FifoIFExport);
        adap_slave_IQZZ1_AMBA_AXIBus1_WriteInterface.FifoIFPort[0](fifo_ad
ap_slave_IQZZ1_AMBA_AXIBus1_WriteInterface0.FifoIFExport);
        adap_slave_LIBU1_AMBA_AXIBus1_WriteInterface.FifoIFPort[0](fifo_ad
ap_slave_LIBU1_AMBA_AXIBus1_WriteInterface0.FifoIFExport);
        adap_slave_VLD1_AMBA_AXIBus1_WriteInterface.FifoIFPort[0](fifo_ada
p_slave_VLD1_AMBA_AXIBus1_WriteInterface0.FifoIFExport);
        AMBA_AXIBus1.master_sock(AMBA_AXIBus1_SlaveAdapter_DebugModule1.sl
ave_sock);
        AMBA_AXIBus1.master_sock(AMBA_AXIBus1_SlaveAdapter_ISSAdapter1.sla
ve_sock);
```

```
        AMBA_AXIBus1.master_sock(AMBA_AXIBus1_SlaveAdapter_ISSAdapter2.sla
ve_sock);
        AMBA_AXIBus1.master_sock(AMBA_AXIBus1_SlaveAdapter_VGA_CONTROLLER1
.slave_sock);
        AMBA_AXIBus1.master_sock(AMBA_AXIBus1_SlaveAdapter_XilinxBRAM1.sla
ve_sock);
        AMBA_AXIBus1.master_sock(AMBA_AXIBus1_SlaveAdapter_XilinxPIC1.slav
e_sock);
        AMBA_AXIBus1.master_sock(AMBA_AXIBus1_SlaveAdapter_XilinxPIC2.slav
e_sock);
        AMBA_AXIBus1.master_sock(AMBA_AXIBus1_SlaveAdapter_adap_slave_IDCT
1_AMBA_AXIBus1_WriteInterface.slave_sock);
        AMBA_AXIBus1.master_sock(AMBA_AXIBus1_SlaveAdapter_adap_slave_IQZZ
1_AMBA_AXIBus1_WriteInterface.slave_sock);
        AMBA_AXIBus1.master_sock(AMBA_AXIBus1_SlaveAdapter_adap_slave_LIBU
1_AMBA_AXIBus1_WriteInterface.slave_sock);
        AMBA_AXIBus1.master_sock(AMBA_AXIBus1_SlaveAdapter_adap_slave_VLD1
_AMBA_AXIBus1_WriteInterface.slave_sock);
        DebugModule1.DebuggerIFPort[0](XilinxBRAM1);
        IDCT1.ModuleAdapterReadIFPort[0](fifo_adap_slave_IDCT1_AMBA_AXIBus
1_WriteInterface0.SpaceModuleReadIFExport);
        IDCT1.SpaceInStreamIFPort[0](DL_IQZZ1_TO_IDCT1_SDLRegister1);
        IDCT1.SpaceOutStreamIFPort[0](DL_IDCT1_TO_LIBU1_SDLRegister1);
        IQZZ1.ModuleAdapterReadIFPort[0](fifo_adap_slave_IQZZ1_AMBA_AXIBus
1_WriteInterface0.SpaceModuleReadIFExport);
        IQZZ1.SpaceInStreamIFPort[0](DL_VLD1_TO_IQZZ1_SDLRegister1);
        IQZZ1.SpaceOutStreamIFPort[0](DL_IQZZ1_TO_IDCT1_SDLRegister1);
        LIBU1.ModuleAdapterReadIFPort[0](fifo_adap_slave_LIBU1_AMBA_AXIBus
1_WriteInterface0.SpaceModuleReadIFExport);
        LIBU1.SpaceInStreamIFPort[0](DL_IDCT1_TO_LIBU1_SDLRegister1);
        VLD1.ModuleAdapterReadIFPort[0](fifo_adap_slave_VLD1_AMBA_AXIBus1_
WriteInterface0.SpaceModuleReadIFExport);
        VLD1.SpaceOutStreamIFPort[0](DL_VLD1_TO_IQZZ1_SDLRegister1);
        adap_master_IQZZ1.ChannelIFPort(AMBA_AXIBus1_MasterAdapter_adap_ma
ster_IQZZ1_ChannelIFPort);
        adap_master_LIBU1.ChannelIFPort(AMBA_AXIBus1_MasterAdapter_adap_ma
ster_LIBU1_ChannelIFPort);
        adap_master_VLD1.ChannelIFPort(AMBA_AXIBus1_MasterAdapter_adap_mas
ter_VLD1_ChannelIFPort);
        armCortexA91.DBusPort(AMBA_AXIBus1_MasterAdapter_armCortexA91_DBus
Port);
        armCortexA91.IBusPort(AMBA_AXIBus1_MasterAdapter_armCortexA91_IBus
Port);
        IQZZ1.ChannelIFPort(adap_master_IQZZ1);
        LIBU1.ChannelIFPort(adap_master_LIBU1);
        VLD1.ChannelIFPort(adap_master_VLD1);
        AMBA_AXIBus1_MasterAdapter_armCortexA91_DBusPort.ChannelIFPort(AMB
A_AXIBus1.slave_sock);
        AMBA_AXIBus1_MasterAdapter_armCortexA91_IBusPort.ChannelIFPort(AMB
A_AXIBus1.slave_sock);
        AMBA_AXIBus1_MasterAdapter_adap_master_LIBU1_ChannelIFPort.Channel
IFPort(AMBA_AXIBus1.slave_sock);
```

```
        AMBA_AXIBus1_MasterAdapter_adap_master_IQZZ1_ChannelIFPort.Channel
IFPort(AMBA_AXIBus1.slave_sock);
        AMBA_AXIBus1_MasterAdapter_adap_master_VLD1_ChannelIFPort.ChannelI
FPort(AMBA_AXIBus1.slave_sock);
        armCortexA91.DebuggerIFPort(DebugModule1);
        XilinxPIC1.IRQOutPort(sXilinxPIC1_IRQOutPort0);
        XilinxPIC2.IRQOutPort(sXilinxPIC2_IRQOutPort0);
        fifo_adap_slave_VLD1_AMBA_AXIBus1_WriteInterface0.IRQHasRoomOutPor
t(sfifo_adap_slave_VLD1_AMBA_AXIBus1_WriteInterface0_IRQHasRoomOutPort0);
        fifo_adap_slave_IQZZ1_AMBA_AXIBus1_WriteInterface0.IRQHasRoomOutPo
rt(sfifo_adap_slave_IQZZ1_AMBA_AXIBus1_WriteInterface0_IRQHasRoomOutPort0)
;
        fifo_adap_slave_IDCT1_AMBA_AXIBus1_WriteInterface0.IRQHasRoomOutPo
rt(sfifo_adap_slave_IDCT1_AMBA_AXIBus1_WriteInterface0_IRQHasRoomOutPort0)
;
        fifo_adap_slave_LIBU1_AMBA_AXIBus1_WriteInterface0.IRQHasRoomOutPo
rt(sfifo_adap_slave_LIBU1_AMBA_AXIBus1_WriteInterface0_IRQHasRoomOutPort0)
;
        AMBA_AXIBus1_MasterAdapter_adap_master_IQZZ1_ChannelIFPort.ClockPo
rt(SysClock);
        AMBA_AXIBus1_MasterAdapter_adap_master_LIBU1_ChannelIFPort.ClockPo
rt(SysClock);
        AMBA_AXIBus1_MasterAdapter_adap_master_VLD1_ChannelIFPort.ClockPor
t(SysClock);
        AMBA_AXIBus1_MasterAdapter_armCortexA91_DBusPort.ClockPort(SysCloc
k);
        AMBA_AXIBus1_MasterAdapter_armCortexA91_IBusPort.ClockPort(SysCloc
k);
        AMBA_AXIBus1_SlaveAdapter_DebugModule1.ClockPort(SysClock);
        AMBA_AXIBus1_SlaveAdapter_ISSAdapter1.ClockPort(SysClock);
        AMBA_AXIBus1_SlaveAdapter_ISSAdapter2.ClockPort(SysClock);
        AMBA_AXIBus1_SlaveAdapter_VGA_CONTROLLER1.ClockPort(SysClock);
        AMBA_AXIBus1_SlaveAdapter_XilinxBRAM1.ClockPort(SysClock);
        AMBA_AXIBus1_SlaveAdapter_XilinxPIC1.ClockPort(SysClock);
        AMBA_AXIBus1_SlaveAdapter_XilinxPIC2.ClockPort(SysClock);
        AMBA_AXIBus1_SlaveAdapter_adap_slave_IDCT1_AMBA_AXIBus1_WriteInter
face.ClockPort(SysClock);
        AMBA_AXIBus1_SlaveAdapter_adap_slave_IQZZ1_AMBA_AXIBus1_WriteInter
face.ClockPort(SysClock);
        AMBA_AXIBus1_SlaveAdapter_adap_slave_LIBU1_AMBA_AXIBus1_WriteInter
face.ClockPort(SysClock);
        AMBA_AXIBus1_SlaveAdapter_adap_slave_VLD1_AMBA_AXIBus1_WriteInterf
ace.ClockPort(SysClock);
        ISSAdapter1.ClockPort(SysClock);
        ISSAdapter1.nResetPort(sReset);
        ISSAdapter2.ClockPort(SysClock);
        ISSAdapter2.nResetPort(sReset);
        adap_master_IQZZ1.nResetPort(sReset);
        adap_master_LIBU1.nResetPort(sReset);
        adap_master_VLD1.nResetPort(sReset);
        AMBA_AXIBus1.clk(SysClock);
        fifo_adap_slave_IDCT1_AMBA_AXIBus1_WriteInterface0.ClockPort(SysCl
ock);
```

```
        fifo_adap_slave_IDCT1_AMBA_AXIBus1_WriteInterface0.nResetPort(sRes
et);
        fifo_adap_slave_IQZZ1_AMBA_AXIBus1_WriteInterface0.ClockPort(SysCl
ock);
        fifo_adap_slave_IQZZ1_AMBA_AXIBus1_WriteInterface0.nResetPort(sRes
et);
        fifo_adap_slave_LIBU1_AMBA_AXIBus1_WriteInterface0.ClockPort(SysCl
ock);
        fifo_adap_slave_LIBU1_AMBA_AXIBus1_WriteInterface0.nResetPort(sRes
et);
        fifo_adap_slave_VLD1_AMBA_AXIBus1_WriteInterface0.ClockPort(SysClo
ck);
        fifo_adap_slave_VLD1_AMBA_AXIBus1_WriteInterface0.nResetPort(sRese
t);
        armCortexA91.ClockPort(SysClock);
        XilinxBRAM1.ClockPort(SysClock);
        IDCT1.nResetPort(sReset);
        IQZZ1.nResetPort(sReset);
        LIBU1.nResetPort(sReset);
        VLD1.nResetPort(sReset);
        XilinxPIC1.ClockPort(SysClock);
        XilinxPIC1.nResetPort(sReset);
        XilinxPIC2.ClockPort(SysClock);
        XilinxPIC2.nResetPort(sReset);
        reset_manager_1.ClockPort(SysClock);
        reset_manager_1.nResetPort(sReset);
        VGA_CONTROLLER1.ClockPort(SysClock);
        armCortexA91.SPI0InPort(sXilinxPIC1_IRQOutPort0);
        armCortexA91.SPI1InPort(sXilinxPIC2_IRQOutPort0);
        XilinxPIC1.IRQInPort[0](sfifo_adap_slave_VLD1_AMBA_AXIBus1_WriteIn
terface0_IRQHasRoomOutPort0);
        XilinxPIC1.IRQInPort[1](sfifo_adap_slave_IQZZ1_AMBA_AXIBus1_WriteI
nterface0_IRQHasRoomOutPort0);
        XilinxPIC1.IRQInPort[2](sfifo_adap_slave_IDCT1_AMBA_AXIBus1_WriteI
nterface0_IRQHasRoomOutPort0);
        XilinxPIC1.IRQInPort[3](sfifo_adap_slave_LIBU1_AMBA_AXIBus1_WriteI
nterface0_IRQHasRoomOutPort0);

        SPACE_MONITORABLE_SYSTEM( SPACE_MONITOR_USE_FILE,
"C:/SpaceCodesign/SampleApps/MJPEG_newversion/MJPEG/Simtek/all_HW_Less_DEM
UX1/build/monitoring");

        SPACE_MONITORABLE_BUS( AMBA_AXIBus1 );
        SPACE_MONITORABLE_ISS( armCortexA91 );
        SPACE_MONITORABLE_MEMORY( XilinxBRAM1 );
        SPACE_MONITORABLE_MODULE( IDCT1 );
        SPACE_MONITORABLE_MODULE( IQZZ1 );
        SPACE_MONITORABLE_MODULE( LIBU1 );
        SPACE_MONITORABLE_MODULE( VLD1 );
        SPACE_MONITORABLE_SDL( DL_IDCT1_TO_LIBU1_SDLRegister1 );
        SPACE_MONITORABLE_SDL( DL_IQZZ1_TO_IDCT1_SDLRegister1 );
        SPACE_MONITORABLE_SDL( DL_VLD1_TO_IQZZ1_SDLRegister1 );
```

```
        // The following section is a free section where user can
singlehandedly code.
        // All code put in between tags will be preserved and copied into
your newly generated main file
        // *Warning* All code put everywhere else will be erased without
possible undo operation.
        // _FREE_SECTION_START_ (Do Not Erase or Duplicate this Line)
                //       Empty
        // _FREE_SECTION_END_ (Do Not Erase or Duplicate this Line)

        // No buffer for the output.
        setvbuf ( stdout, NULL , _IONBF , 0);

        // Simulation section  -- Begin
        cout << "Starting simulation.\n";
        time( &simulation_time_begin );
        sc_start(simulation_time,sc_core::sc_time_unit(sim_units));
        time( &simulation_time_end );

        cout << endl << "Simulation has ended @" <<
sc_time_stamp().to_seconds() << " s";
        cout << endl << "Simulation wall clock time: " << (unsigned
long)(simulation_time_end - simulation_time_begin) << " seconds." << endl
<< endl;
        cout.flush();

        // Simulation section  -- End

    return 0; // No problem occurs
}
```

BIBLIOGRAPHIE

Amagbégnon, Pascalin, Loïc Besnard et Paul Le Guernic. 1995. « Implementation of the data-flow synchronous language signal ». *ACM SIGPLAN Notices,* vol. 30, n° 6, p. 163-173.

Bensalem, Saddek, Marius Bozga, Thanh-Hung Nguyen et Joseph Sifakis. 2009. « D-finder: A tool for compositional deadlock detection and verification ». In *Computer Aided Verification*. p. 614-619. Springer.

Benyoussef, M, JF Boland, G Nicolescu, G Bois et J Hugues. Février 2014. « Design Space Exploration: Bridging the Gap Between High-Level Models and Virtual Execution Platforms ». In *Embedded Real Time Software and System*. (Toulouse).

Besnard, Loïc, Thierry Gautier, Paul Le Guernic et Jean-Pierre Talpin. 2010. « Compilation of polychronous data flow equations ». In *Synthesis of Embedded Software*. p. 1-40. Springer.

Bomel, Pierre, Dominique Blouin, Mickael Lanoe et Eric Senn. 2012. « Functional validation of AADL models via model transformation to SystemC with ATL ». In *Proceedings of the 5th International Workshop on Model Based Architecting and Construction of Embedded Systems*. p. 13-18. ACM.

Bozga, Marius, Susanne Graf, Ileana Ober, Iulian Ober et Joseph Sifakis. 2004. « The IF toolset ». In *Formal Methods for the Design of Real-Time Systems*. p. 237-267. Springer.

Chkouri, Mohamed Yassin. 2010. « Modélisation des systèmes temps-réel embarqués en utilisant AADL pour la génération automatique d'applications formellement vérifiées ». Grenoble, Université Joseph-Fourier, 176 p.

Chkouri, Mohamed Yassin, et Marius Bozga. 2009. « Prototyping of distributed embedded systems using aadl ». *ACESMB 2009*, p. 65.

Cmelik, Bob, et David Keppel. 1995. *Shade: A fast instruction-set simulator for execution profiling*. Springer.

Déplanche, Anne-Marie, et Sébastien Faucou. 2005. « Les langages de description d'architecture pour le temps réel ». p. 31.

DOULOS. 2014. « SystemC TLM-2.0 ». < https://www.doulos.com/knowhow/systemc/tlm2/ >. Consulté le 20 Avril 2013.

Feiler, Peter H, et David P Gluch. 2012. *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. Addison-Wesley.

Feiler, Peter H, David P Gluch et John J Hudak. 2006. *The architecture analysis & design language (AADL): An introduction*. DTIC Document.

Fleurey, Franck, Jim Steel et Benoit Baudry. 2004. « Validation in model-driven engineering: testing model transformations ». In *Model, Design and Validation, 2004. Proceedings. 2004 First International Workshop on*. p. 29-40. IEEE.

Friedenthal, Sanford, Regina Griego et Mark Sampson. 2007. « INCOSE model based systems engineering (MBSE) initiative ». In *INCOSE 2007 Symposium*.

Henzinger, Thomas A, et Joseph Sifakis. 2006. « The embedded systems design challenge ». In *FM 2006: Formal Methods*. p. 1-15. Springer.

Hill, James H. 2011. « Measuring and reducing modeling effort in domain-specific modeling languages with examples ». In *Engineering of Computer Based Systems (ECBS), 2011 18th IEEE International Conference and Workshops on*. p. 120-129. IEEE.

Hill, James H, Sumant Tambe et Aniruddha Gokhale. 2007. « Model-driven engineering for development-time QoS validation of component-based software systems ». In *Engineering of Computer-Based Systems, 2007. ECBS'07. 14th Annual IEEE International Conference and Workshops on the*. p. 307-316. IEEE.

Hladik, Pierre-Emmanuel, Florent Peres et Xiaomu Shi. 2010. « Analyse d'un modèle AADL à l'aide de Pola ». *Actes des 10es journées Francophones sur les Approches Formelles dans l'Assistance au Développement de Logiciels*.

Keinert, Joachim, et Jürgen Teich. 2011. « Electronic System Level Design of Image Processing Applications with SystemCoDesigner ». In *Design of Image Processing Embedded Systems Using Multidimensional Data Flow*. p. 81-91. Coll. « Embedded Systems »: Springer New York.

Koopman, Philip. 1996. « Embedded system design issues (the rest of the story) ». In *Computer Design: VLSI in Computers and Processors, 1996. ICCD'96. Proceedings., 1996 IEEE International Conference on*. p. 310-317. IEEE.

Kopetz, Hermann. 2008. « The complexity challenge in embedded system design ». In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*. p. 3-12. IEEE.

Liao, Thorsten Grötkerand Stan, Grant Martin, Stuart Swan et Thorsten Grötker. 2002. *System design with SystemC*. Springer.

Moss, Laurent, Hubert Guérard, Gary Dare et Guy Bois. 2012. « Rapid Design Exploration on an ESL Framework featuring Hardware-Software Codesign for ARM Processor-based FPGA's ». *SAME 2012 Conference,* vol. 1.

Pohl, Klaus, Harald Hönninger, Reinhold Achatz et Manfred Broy. 2012. *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*. Springer.

Prisaznuk, Paul J. 2008. « Arinc 653 role in integrated modular avionics (IMA) ». In *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*. p. 1. E. 5-1-1. E. 5-10. IEEE.

Savard, Julien. 2012. « Intégration d'un simulateur de partitionnement spatial et temporel à un flot de conception basé sur les modèles ». École Polytechnique de Montréal.

Schmidt, Douglas C. 2006. « Model-driven engineering ». *COMPUTER-IEEE COMPUTER SOCIETY-,* vol. 39, n° 2, p. 25.

Team, SEI AADL. 2006. « An extensible open source AADL tool environment (OSATE) ». *Software Engineering Institute*.

Varona-Gómez, Roberto, et Eugenio Villar. 2009. « AADL simulation and performance analysis in SystemC ». In *Engineering of Complex Computer Systems, 2009 14th IEEE International Conference on*. p. 323-328. IEEE.

Wilson, Peter, et H Alan Mantooth. 2013. *Model-based engineering for complex electronic systems: techniques, methods and applications*. Access Online via Elsevier.

wiseGeek. 2014. « What Is a Zigzag Transformer? ». < http://www.wisegeek.com/what-is-a-zigzag-transformer.htm >. Consulté le 2 avril 2014.

Wolf, Wayne. 2003. « A decade of hardware/software codesign ». *Computer,* vol. 36, n° 4, p. 38-43.

Yu, Huafeng, Yue Ma, Yann Glouche, Jean-Pierre Talpin, Loïc Besnard, Thierry Gautier, Paul Le Guernic, Andres Toom et Odile Laurent. 2011. « System-level co-simulation

of integrated avionics using polychrony ». In *Proceedings of the 2011 ACM Symposium on Applied Computing*. p. 354-359. ACM.