UNIVERSITE SAAD DAHLAB DE BLIDA

Faculté des Sciences

Département d'Informatique

MEMOIRE DE MAGISTER

Option : Systèmes d'information et de connaissances

HARDWARE/SOFTWARE CODESIGN SELON UNE APPROCHE D'ARCHITECTURE LOGICIELLE

Par

Yasmine MANCER

Devant le jury composé de :

N. BENBLIDIA	Maître de conférences (A), U.S.D., Blida	Présidente
H. ABED	Professeur, U.S.D., Blida	Examinatrice
N. BOUSTIA	Maître de conférences (B), U.S.D., Blida	Examinatrice
A. HACHICHI	Maître de conférences (B), U.S.T.H.B., Alger	Examinatrice
D. BENNOUAR	Maître de conférences (A), U.S.D., Blida	Promoteur

Blida, Février 2012

RESUME

L'augmentation de la complexité des systèmes électroniques conduit à un dépassement du temps de conception alors que paradoxalement, la concurrence économique impose des temps de mise sur le marché de plus en plus courts. Pour tenter de limiter ce décalage, de nouvelles méthodologies de conception sont requises, le codesign matériel/logiciel peut faire face à ce problème. Dans le codesign, les systèmes sont composés d'une partie logicielle et d'une partie matérielle. La communication entre ces deux parties est assurée à travers l'utilisation d'une interface logicielle/matérielle.

Parmi les étapes du codesign les plus problématiques, la synthèse des interfaces de communication. Cette dernière, est la clé de l'étape d'intégration des divers composants du système. C'est une tâche complexe et difficile à mettre en œuvre. De plus, les interfaces ont des structures complexes, leur conception nécessite des compétences issues des domaines du logiciel et du matériel. Le défi de la synthèse des interfaces de communication est de trouver des modèles qui permettent de gérer le système à un haut niveau d'abstraction pour supporter la complexité croissante des interfaces.

Le présent mémoire introduit une nouvelle approche pour la cosynthèse des interfaces de communication dans le codesign matériel/logiciel. L'idée de base de l'approche proposée est de résoudre un certain nombre de problèmes de la cosynthèse des interfaces de communication en ramenant le problème de la communication dans un système de codesign à un niveau d'abstraction très élevé qui est le niveau architecture et en libérant le concepteur de la gestion des détails de bas niveau. Cette approche consiste à considérer le problème de la communication entre le logiciel et le matériel dans l'espace Architecture Logicielle à travers un modèle de composant adapté au codesign et d'utiliser un langage de description d'architecture logicielle pour la cospécification du système.

Mots clés : Codesign matériel/logiciel, Cosynthèse d'interfaces, Communication, Architecture Logicielle, Composant.

ABSTRACT

The increasing complexity of electronic systems led to exceeding the design time while, paradoxically, economic competition imposes a shorter time-to-market. In an attempt to reduce this gap, new design methodologies are required, the hardware/software codesign can cope with this problem. In codesign, systems are composed of a software part and a hardware part. The communication between these two parts is ensured through the use of hardware/software interface.

Among the most problematic steps of codesign, the synthesis of communication interfaces. This is the key for the step of integration of the various components of the system. It is a complex task and difficult to implement. Also, the interfaces have complex structures and their design requires skills from the fields of software and hardware. The challenge of the synthesis of communication interfaces is to find models that can manage the system at a high level of abstraction in order to support the growing complexity of interfaces.

This document introduces a new approach for the cosynthesis of communication interfaces in the hardware/software codesign. The basic idea of the proposed approach is to solve many problems of the cosynthesis of communication interfaces by bringing back the problem of the communication in a codesign system to a very high level of abstraction which is the architecture level and by liberating the designer for the management of low-level details. This approach considers the problem of the communication between the software and the hardware in software architecture space through a component model adapted to the codesign and uses a software architecture description language for the cospecification of the system.

Keywords: Hardware/software codesign, Interface cosynthesis, Communication, Software Architecture, Component.

ملخص

التعقيد المتزايد في الأنظمة الإلكترونية يؤدي إلى الزيادة في وقت التصميم في حين أن المنافسة الاقتصادية تفرض أن يكون الوقت للوصول إلى السوق أقصر فأقصر. للحد من هذه الفجوة مطلوب منهجيات جديدة للتصميم، التصميم المشترك معدات / برمجيات يمكن أن يحد من هذا المشكل. في التصميم المشترك، الأنظمة تتألف من جزء للمعدات و جزء للبرمجيات. التواصل بين هذين الجزأين يتم من خلال استخدام واجهة معدات / برمجيات.

من بين مراحل التصميم المشترك الأكثر إشكالا، استخلاص واجهات الاتصال. هذه الأخيرة تعد الخطوة الرئيسية لدمج مكونات النظام. إنها عملية معقدة وصعبة التنفيذ. بالإضافة إلى ذلك، الواجهات لديها هياكل معقدة وتصميمها يتطلب مهارات من مجالي المعدات و البرمجيات. التحدي الذي يواجه عملية استخلاص واجهات الاتصال هو العثور على نماذج لإدارة النظام على مستوى عال من التجريد من اجل مواجهة التعقيد المتزايد للواجهات.

هذه المذكرة تقدم منهاج جديد لاستخلاص واجهات الاتصال في التصميم المشترك معدات / برمجيات. الفكرة الأساسية من المنهاج المقترح هو حل عدد من المشاكل للاستخلاص المشترك لواجهات الاتصال عن طريق معالجة مشكل الاتصال في نظام التصميم المشترك من خلال مستوى جد عال من التجريد ألا و هو مستوى الهندسة وتخليص المصمم من عبء إدارة التفاصيل ذات المستوى المنخفض. هذا المنهاج ينص على اعتبار مشكل الاتصال بين المعدات و البرمجيات على مستوى الهندسة البرمجية من خلال نموذج مكون مخصص للتصميم المشترك واستخدام لغة لوصف الهندسة البرمجية من اجل التوصيف المشترك للنظام.

الكلمات الرئيسية: التصميم المشترك معدات / برمجيات، استخلاص الواجهات، الاتصال، الهندسة البرمجية، مكون.

TABLE DES MATIERES

RESUME

REMERCIEMENTS

TABLE DES MATIERES

LISTE DES ILLUSTRATIONS, GRAPHIQUES ET TABLEAUX

INTRODUCTION………………………………………………………………… 11

1. LE CODESIGN MATERIEL/LOGICIEL……………………………………… 21

1.1 Introduction………………………………………………………………….. 21
1.2 Définition du terme codesign……………………………………………….... 21
1.3 Origines du codesign……………………………………………………….... 22
1.4 Les domaines d'application du codesign……………………………………. 24
1.4.1 Les systèmes embarqués…………………………………………………... 24
1.4.2 Les systèmes de télécommunication………………………………………. 25
1.4.3 Les systèmes de traitement spécifiques…………………………………… 25
1.4.4 Les accélérateurs d'exécution logicielle…………………………………... 25
1.5 Les avantages du codesign………………………………………………….. 25
1.6 Comparaison entre la conception traditionnelle et le codesign…………………... 27
1.7 Processus général de codesign……………………………………………… 28
1.7.1 La cospécification…………………………………………………………. 29
1.7.1.1 La spécification homogène……………………………………………… 30
1.7.1.2 La spécification hétérogène…………………………………………….. 31
1.7.1.3 Les langages de spécification…………………………………………… 31
1.7.1.4 Exigences pour les langages de spécification………………………….. 33
1.7.2 La modélisation……………………………………………………………. 34
1.7.2.1 Les modèles de calcul…………………………………………………... 35
1.7.2.1.1 Les automates à états finis……………………………………………. 37
1.7.2.1.2 Les réseaux de Petri…………………………………………………… 38
1.7.2.1.3 Les systèmes à événements discrets…………………………………... 38
1.7.2.1.4 Les graphes flots de données…………………………………………. 39
1.7.2.1.5 Les modèles synchrones/ réactifs…………………………………….... 39
1.7.2.1.6 La modélisation par rendez-vous……………………………………… 39
1.7.2.1.7 Les modèles orientés objet……………………………………………... 39
1.7.2.1.8 Les modèles hétérogènes……………………………………………… 40
1.7.3 Le partitionnement matériel/logiciel………………………………………. 41
1.7.3.1 Méthodes de partitionnement matériel/logiciel………………………….. 42
1.7.3.2 Les algorithmes de partitionnement……………………………………. 42
1.7.3.3 Le niveau de granularité…………………………………………………. 44
1.7.3.4 L'analyse des propriétés d'un partitionnement………………………….. 44
1.7.3.5 La fonction de coût……………………………………………………… 44
1.7.3.6 L'ordonnancement………………………………………………………. 45

1.7.3.7 L'architecture cible………………………………………………………… 45
1.7.3.7.1 Les modèles de l'architecture cible……………………………………… 47
1.7.3.8 Travaux liés au partitionnement matériel/logiciel……………………………. 48
1.7.4 La covérification……………………………………………………… 49
1.7.4.1 La cosimulation………………………………………………………. 49
1.7.4.1.1 Approches de cosimulation…………………………………………. 50
1.7.4.2 La vérification formelle…………………………………………………. 51
1.7.4.3 Travaux liés à la covérification……………………………………….. 51
1.8 Conclusion………………………………………………………………... 52

2. LA SYNTHESE DES INTERFACES DE COMMUNICATION DANS LE CODESIGN MATERIEL/LOGICIEL……………………………………………… 54

2.1 Introduction……………………………………………………………….. 54
2.2 Les interfaces logicielles/matérielles…………………………………….. 54
2.2.1 Caractéristiques des interfaces de communication……………………... 55
2.3 La synthèse du matériel……………………………………………………... 56
2.4 La synthèse du logiciel………………………………………………… 56
2.5 La communication dans un système de codesign………………………… 57
2.5.1 La synthèse des communications…………………………………………….. 57
2.5.2 Les niveaux d'abstraction de la communication…………………………….. 58
2.5.2.1 Le niveau service……………………………………………………….. 58
2.5.2.2 Le niveau transaction…………………………………………………. 58
2.5.2.3 Le niveau macro-architecture ou message……………………………. 59
2.5.2.4 Le niveau RTL ou microarchitecture…………………………………. 59
2.5.3 Types de communication dans un système mixte………………………. 60
2.5.3.1 Communication par mémoire partagée……………………………….. 60
2.5.3.2 Communication par passage de message…………………………….... 60
2.5.3.3 Utilisation de mémoire virtuelle d'entrées/sorties……………………. 61
2.5.3.4 Accès direct à une mémoire distante………………………………. 61
2.5.4 Types de conception des interfaces de communication………………… 61
2.5.4.1 Synthèse manuelle des interfaces…………………………………….. 61
2.5.4.2 Synthèse automatique des interfaces…………………………………. 62
2.5.5 Les composants de la communication………………………………….. 62
2.5.5.1 Les adaptateurs……………………………………………………….. 62
2.5.5.2 Les ponts (bridges)…………………………………………………… 63
2.5.6 Topologies du réseau de communication………………………………. 63
2.5.6.1 Les bus partagés……………………………………………………. 63
2.5.6.2 Les connexions point à point………………………………………. 64
2.5.7 Les modèles de signalisations pour la synchronisation logicielle/matérielle…... 64
2.5.7.1 La scrutation…………………………………………………………… 64
2.5.7.2 Les interruptions matérielles………………………………………….. 65
2.6 Approches de cosynthèse d'interfaces de communication……………………… 65
2.6.1 Daveau……………………………………………………………… 65
2.6.2 Gogniat………………………………………………………………. 65
2.6.3 Hommais…………………………………………………………… 66
2.6.4 Coste………………………………………………………………. 66
2.6.5 Maalej et al……………………………………………………………… 66
2.6.6 Lyonnard…………………………………………………………….. 67
2.6.7 Gharsalli……………………………………………………………… 67

2.6.8 Paviot………………………………………………………………………… 67
2.6.9 Grasset……………………………………………………………………….. 68
2.6.10 Youssef…………………………………………………………………….. 68
2.6.11 Jerraya……………………………………………………………………… 68
2.6.12 Chavet……………………………………………………………………… 69
2.6.13 Khan……………………………………………………………………….. 69
2.6.14 Hao et Xie………………………………………………………………….. 69
2.6.15 Faes et al…………………………………………………………………… 70
2.6.16 Autres approches de cosynthèse…………………………………………… 70
2.7 Outils de cosynthèse d'interfaces de communication…………………………….. 71
2.7.1 POLARIS…………………………………………………………………….. 71
2.7.2 DTSE………………………………………………………………………… 72
2.7.3 PICO…………………………………………………………………………. 73
2.7.4 SPARK……………………………………………………………………….. 74
2.7.5 Streamroller………………………………………………………………… 74
2.7.6 xPilot……………………………………………………………………….. 75
2.7.7 Catapult C…………………………………………………………………… 76
2.7.8 SynDEx………………………………………………………………………. 77
2.7.9 GAUT……………………………………………………………………… 78
2.8 Bilan………………………………………………………………………………. 79
2.8.1 Caractéristiques des approches et des outils de cosynthèse…………………….. 79
2.8.2 Problèmes de la synthèse des interfaces de communication…………………… 83
2.9 Conclusion………………………………………………………………………... 84

3. L'ARCHITECTURE LOGICIELLE ET L'APPROCHE IASA…………………. 86

3.1 Introduction……………………………………………………………………….. 86
3.2 L'architecture logicielle……………………………………………………….…. 86
3.2.1 Avantages de l'architecture logicielle…………………………………………….. 88
3.2.2 Les langages de description d'architectures……………………………………. 88
3.3 L'approche intégrée d'architecture logicielle…………………………………….. 89
3.3.1 Le modèle de composant de l'approche intégrée………………………………. 89
3.3.1.1 Modélisation de la vue externe……………………………………………… 90
3.3.1.1.1 Les points d'accès…………………………………………………… 90
3.3.1.1.1.1 Les points d'accès aux données (DOAP)………………………………… 92
3.3.1.1.1.2 Les points d'accès de services (ACTOAP)……………………………… 94
3.3.1.1.2 Les ports……………………………………………………………………….. 94
3.3.1.1.3 L'enveloppe………………………………………………………………… 95
3.3.1.2 Organisation de la vue interne……………………………………………….. 95
3.3.1.2.1 La partie opérative…………………………………………………… 96
3.3.1.2.2 La partie contrôle…………………………………………………………… 96
3.3.1.3 Déploiement des composants……………………………………………… 97
3.3.1.3.1 Les cas de déploiement…………………………………………………….. 97
3.3.2 L'ADL de l'approche intégrée…………………………………………………….. 98
3.3.2.1 Description des balises du langage x3ADL…………………………………….. 98
3.3.2.1.1 La balise Ports……………………………………………………………… 99
3.3.2.1.1.1 La balise InDataPort……………………………………………………… 99
3.3.2.1.1.2 La balise OutDataPort…………………………………………………….. 100
3.3.2.1.2 La balise Connectors………………………………………………………... 101
3.3.2.1.3 La balise OperativePart……………………………………………….…. 101

3.3.2.1.4 La balise ControlPart…………………………………………………….. 102
3.3.2.1.5 La balise Properties………………………………………………………. 103
3.4 Conclusion…………………………………………………………………... 103

4. SYNTHESE DES COMMUNICATIONS SELON L'APPROCHE IASA……… 104

4.1 Introduction………………………………………………………………….. 104
4.2 Cospécification du système de codesign dans IASA…………………………….. 104
4.3 Représentation des composants d'un système de codesign dans IASA………… 105
4.4 Spécification du comportement du contrôleur……………………………………. 106
4.5 Spécification du déploiement………………………………………………… 108
4.6 Transformation d'une description IASA en une spécification de codesign……… 112
4.7 Les règles de transformation…………………………………………………… 115
4.7.1 Les règles de transformation dans le cas du codesign matériel/logiciel………... 116
4.8 Production du contrôleur de la communication……………………………….. 118
4.9 Conclusion………………………………………………………………. 119

5. IASASTUDIO : UN OUTIL POUR LA VALIDATION DE LA COMMUNICATION DANS UN SYSTEME DE CODESIGN SELON L'APPROCHE IASA………………………………………………………… 120

5.1 Introduction………………………………………………………………... 120
5.2 Présentation de l'environnement IASASTUDIO……………………………. 120
5.2.1 Conception de l'outil IASASTUDIO……………………………………… 121
5.2.2 Représentation des composants composites dans IASASTUDIO……………. 123
5.2.3 Interface graphique de l'outil IASASTUDIO……………………………… 124
5.3 Mise en œuvre de l'outil IASASTUDIO pour la réalisation d'un composant composite……………………………………………………………………... 127
5.3.1 Création d'un nouveau composant composite dans IASASTUDIO……………. 127
5.3.2 Définition de la vue interne du composant Codesign………………………... 129
5.3.2.1 La bibliothèque des composants logiciels…………………………………..... 129
5.3.2.2 La bibliothèque des composants matériels…………………………..….. 131
5.3.3 Définition de la vue externe du composant Codesign……………………..…… 135
5.3.4 Réalisation de la connexion interne……………………………………….……… 137
5.3.5 Réalisation de la communication……………………………………….…… 138
5.4 Conclusion…………………………………………………………………... 142

CONCLUSION…………………………………………………………………… 144

APPENDICE A : LISTE DES ABREVIATIONS

REFERENCES

LISTE DES ILLUSTRATIONS, GRAPHIQUES ET TABLEAUX

Figure 1.1 : Conception traditionnelle et codesign……………………………... 27
Figure 1.2 : Processus général de codesign…………………………………….. 29
Figure 1.3 : Flot de codesign pour une spécification homogène………………….. 30
Figure 1.4 : Flot de codesign pour une spécification hétérogène…………………. 31
Figure 1.5 : Exemple d'architecture cible………………………………………. 46
Figure 1.6 : Architecture monoprocesseur……………………………………… 47
Figure 1.7 : Architecture multiprocesseur……………………………………… 47
Figure 2.1 : Interfaces de communication……………………………………… 55
Figure 2.2 : Architecture d'interface entre trois composants……………………… 72
Figure 2.3 : La méthodologie DTSE……………………………………………. 73
Figure 2.4 : Fonctionnement de l'outil PICO………………………………….. 74
Figure 2.5 : Le flot de synthèse SPARK……………………………………….. 74
Figure 2.6 : Le flot de synthèse Streamroller…………………………………… 75
Figure 2.7 : Le flot de synthèse xPilot………………………………………….. 76
Figure 2.8 : Interface de l'outil Catapult C…………………………………….. 77
Figure 2.9 : Interface de l'outil SynDEx……………………………………….. 78
Figure 2.10 : Interface de l'outil GAUT………………………………………... 79
Figure 3.1 : Concepts de base de l'architecture logicielle………………………… 87
Figure 3.2 : Connexions utilisant les points d'accès……………………………. 91
Figure 3.3 : Diagramme de classes du point d'accès……………………………. 92
Figure 3.4 : Représentation graphique des DOAP………………………………… 94
Figure 3.5 : Représentation graphique des ACTOAP…………………………….. 94
Figure 3.6 : Vue interne d'un composant composite……………………………. 96
Figure 3.7 : Les balises du langage x3ADL…………………………………… 99
Figure 3.8 : La balise Ports…………………………………………………... 99
Figure 3.9 : Exemples de la balise InDataPort………………………………….. 100
Figure 3.10 : Exemples de la balise OutDataPort………………………………. 100
Figure 3.11 : La balise Connectors…………………………………………... 101
Figure 3.12 : Exemples de la balise OperativePart…………………………….. 102
Figure 3.13 : Exemple de la balise ControlPart………………………………… 102
Figure 4.1 : Architecture d'un composant de codesign composite dans IASA…... 106
Figure 4.2 : Un composant de codesign composite selon IASA………………….. 108
Figure 4.3 : Architecture du flot HS_COMMUNICATION……………………… 110
Figure 5.1 : Présentation de l'environnement IASASTUDIO……………………. 121
Figure 5.2 : Interface du plugin Eclipse Sigasi HDT……………………………. 122
Figure 5.3 : Le simulateur Modelsim…………………………………………… 123
Figure 5.4 : Architecture d'un composant de codesign composite dans IASASTUDIO……………………………………………………………….. 124
Figure 5.5 : La fenêtre principale de l'outil IASASTUDIO………………………. 125
Figure 5.6 : La barre d'outils de l'outil IASASTUDIO…………………………… 126
Figure 5.7 : Création d'un nouveau projet dans IASASTUDIO………………….. 128
Figure 5.8 : Conception du composant Codesign dans IASASTUDIO…………… 129
Figure 5.9 : Importation d'un composant logiciel dans IASASTUDIO…………... 130
Figure 5.10 : Description x3ADL du composant logiciel………………………. 131
Figure 5.11 : Importation d'un composant matériel dans IASASTUDIO………… 132

Figure 5.12 : Description x3ADL du composant matériel………………………… 133
Figure 5.13 : Description VHDL du composant matériel…………………………. 134
Figure 5.14 : L'entité du composant matériel……………………………………... 134
Figure 5.15 : Arborescence du composant Codesign……………………………… 135
Figure 5.16 : Détermination du sens de la communication……………………….. 135
Figure 5.17 : Les points d'accès du composant Codesign………………………… 136
Figure 5.18 : La vue externe du composant Codesign…………………………… 136
Figure 5.19 : Réalisation de la connexion interne du composant Codesign………. 137
Figure 5.20 : La vue finale du composant Codesign dans IASASTUDIO………... 138
Figure 5.21 : Les propriétés du composant Codesign…………………………….. 138
Figure 5.22 : Détermination des paramètres de la simulation…………………….. 139
Figure 5.23 : Exécution pas à pas dans IASASTUDIO…………………………… 140
Figure 5.24 : Le fichier de communication………………………………………... 140
Figure 5.25 : Le fichier VHDL READ…………………………………………… 141
Figure 5.26 : Description x3ADL du composant Codesign……………………….. 141
Figure 5.27 : Les fichiers du projet Codesign…………………………………….. 142

Tableau 2.1 : Les différents schémas de communication dans un système mixte… 57
Tableau 2.2 : Niveaux d'abstraction pour la communication……………………... 60
Tableau 2.3 : Tableau récapitulatif des approches et des outils de cosynthèse…… 82
Tableau 2.4 : Caractéristiques de l'approche proposée pour la cosynthèse……….. 84

INTRODUCTION

Les systèmes informatiques dont la conception nécessite une approche dite mixte matérielle/logicielle sont devenus omniprésents dans la vie quotidienne, que ce soit pour un usage professionnel ou personnel. Généralement ces systèmes sont représentés par un seul circuit intégré à haute échelle d'intégration (VLSI[1]).

Les contraintes imposées à de tels systèmes, en termes de fonctionnalité, performance, coût, fiabilité et temps de mise sur le marché sont de plus en plus strictes. Par conséquent, leur développement fait apparaître plusieurs défis. En effet, si d'un coté les applications sont de plus en plus complexes, de l'autre coté la pression du marché pour le développement rapide de ces systèmes rend la tâche de leurs conceptions un réel défi. Les différentes versions d'un même produit se succèdent à un rythme élevé, intégrant à chaque fois de nouvelles fonctionnalités ou des améliorations de celles qui étaient déjà présentes dans la version précédente. Les concepteurs se retrouvent ainsi avec des durées de conception de plus en plus courtes et des SoC[2] de plus en plus compliqués à concevoir.

Le marché des SoC connait depuis quelques années une évolution à un rythme exponentiel. Cette évolution s'est faite, non seulement, au niveau de l'intégration des circuits spécifiques, mais aussi au niveau des architectures des microprocesseurs. Toutefois, les flots de développement et les outils de conception n'ont pas réellement exploité toutes ces potentialités. Il en résulte aujourd'hui un gap de productivité accentué par les lois économiques.

Les SoC et plus spécialement les systèmes embarqués évoluent de manière à incorporer des puissances de calcul sans cesse croissantes. La complexité des algorithmes à embarquer d'une part et le besoin de flexibilité d'autre part, ajouté à cela, le caractère hétérogène des architectures embarquées, nécessitent l'adoption de nouvelles méthodologies de conception qui intègrent les différentes fonctionnalités des systèmes mixtes et assurent leurs exigences. Ces nouvelles méthodologies doivent effectuer un compromis entre la haute performance et la faible consommation des circuits câblés d'une part, la flexibilité et la facilité de développement de l'architecture câblée d'autre part.

---

[1] Very-Large-Scale Integration
[2] System on Chip

Dans les méthodes de conception classiques, la conception des parties logicielles et matérielles d'un système intégré s'effectuait séparément. Ensuite, le logiciel obtenu était exécuté sur le prototype matériel. Dans ce cas, si les contraintes de la spécification exigées par le système ne sont pas respectées, le processus de conception est réitéré dans l'espoir de retrouver un bon prototype. Cette technique s'est avérée lourdement coûteuse en termes de temps de réalisation et de coût global du système. De plus, ces méthodes présentent de nombreux problèmes dont voici les plus importants :

- La séparation entre l'équipe du matériel et celle du logiciel entraine un manque de dialogue et d'interaction entre les concepteurs des deux parties conduisant ensuite à des difficultés importantes découvertes tardivement pendant la phase d'intégration du logiciel et du matériel.
- Les difficultés rencontrées nécessitent parfois d'importantes modifications dans le matériel ou le logiciel.
- Les possibilités d'exploration des différentes solutions où certaines fonctionnalités pourraient migrer du logiciel vers le matériel ou vice-versa, sont limitées.
- Les changements de spécification tardifs obligent des fois les concepteurs à refaire le travail.
- L'absence d'une représentation matérielle/logicielle unifiée entraîne des difficultés à vérifier le système dans son ensemble.

En vue de faire face aux insuffisances des méthodes classiques, plusieurs chercheurs se sont penchés sur l'élaboration d'environnements de codesign matériel/logiciel. De Micheli [1] un des pionniers dans le domaine de la conception des circuits intégrés, définit le codesign matériel/logiciel comme étant « Un processus qui consiste à répondre aux objectifs du système en exploitant la synergie du matériel et du logiciel par leur conception concurrente ». Le point clé des méthodes de codesign est de retarder la séparation entre le matériel et le logiciel pour éviter les surprises rencontrées lors de la phase d'intégration puisque les erreurs de conception sont détectées beaucoup plus tôt.

Le codesign matériel/logiciel est appliqué dans de nombreux domaines : les systèmes embarqués, les systèmes de télécommunication et en général dans les applications qui contiennent du matériel et du logiciel et qui exigent des performances strictes. Ce qui caractérise ces différents domaines c'est qu'ils sont toujours soumis à des contraintes de plus en plus sévères en termes de coût, de temps et de performances. Toutefois, le codesign matériel/logiciel est confronté à de nombreux problèmes. Le grand problème du codesign, posé par Stoy [2], est en fait « Comment prendre deux disciplines très différentes non

seulement dans leurs aspects techniques, mais aussi, dans leurs cultures et les fusionner en un tout intégré et fonctionnel ? ».

Le processus du codesign matériel/logiciel implique plusieurs étapes aboutissant à la génération d'architectures optimisées. Ces étapes sont :

- La cospécification : elle consiste à décrire les fonctionnalités attendues d'un système ainsi que toutes les contraintes qu'il doit satisfaire, à un niveau de détails suffisant permettant de prévoir son comportement complet, sans ambigüité et sans préciser les implémentations.
- La modélisation : est le processus de conceptualisation et d'affinement des spécifications. Un modèle de calcul (MoC[3]) est une notation conceptuelle qui décrit le comportement du système désiré [3].
- Le partitionnement matériel/logiciel : est le problème de division des fonctionnalités d'une application en une partie qui s'exécute comme des instructions séquentielles sur des microprocesseurs (le logiciel) et une partie qui s'exécute en tant que circuits parallèles sur des ASIC[4] ou des FPGA[5] (le matériel), dans le but d'atteindre les objectifs fixés par la conception tels que la performance, la puissance, la taille et le coût [4]. Le partitionnement s'effectue habituellement en deux phases: la sélection d'une architecture matérielle et l'allocation des éléments du modèle fonctionnel spécifiant l'application sur les composants de cette architecture. Pour cela, il est nécessaire d'avoir une connaissance bien précise, d'une part des caractéristiques logicielles et matérielles des fonctions modélisant l'application et d'autre part du modèle d'architecture cible considéré.
- La cosynthèse : permet de transformer les descriptions fonctionnelles en descriptions directement implantables sur les processeurs matériels et logiciels de l'architecture cible. La cosynthèse concerne la synthèse des parties matérielles et logicielles et la synthèse des interfaces de communication. L'objectif de la synthèse des communications consiste à appliquer les stratégies de communication qui permettent d'effectuer les transferts de données à moindre coût tout en respectant les contraintes de la spécification.
- La covérification : avant et après la cosynthèse, une covérification fonctionnelle est effectuée généralement par une technique de cosimulation. La covérification

[3] Model of Computation
[4] Application Specific Integrated Circuit
[5] Field Programmable Gate Array

permet d'éviter la propagation des erreurs entre les différents niveaux d'abstraction et de s'assurer que les raffinements successifs conduisent au même fonctionnement et respectent les contraintes de la spécification.

1. Problématique

Malgré les efforts de recherche déployés dans le domaine du codesign, aucune approche n'a totalement réussi à automatiser l'ensemble du processus de la conception concurrente. Ce qui rend complexe le travail des codesigners, c'est que le codesign comporte les étapes classiques du génie logiciel et de la micro-électronique (conception de circuits intégrés à très grande échelle).

Parmi les étapes du processus de codesign les plus problématiques, la synthèse des interfaces de communication. Cette dernière, est la clé de l'étape d'intégration des divers composants du système.

Pour qu'un système mixte fonctionne, les éléments de son architecture devront s'échanger des informations. Ces échanges sont désignés par le terme communication. Pour cela, les divers composants logiciels et matériels doivent être interconnectés. Il est donc nécessaire de développer des interfaces qui assurent une bonne efficacité des communications entre ces composants.

Dans le codesign, trois types de communication peuvent être considérés : communication logiciel/logiciel, communication matériel/matériel et communication logiciel/matériel. Le dernier type pose plus de problèmes en raison de l'hétérogénéité des composants utilisés.

La conception des interfaces de communication entre les composants matériels et les composants logiciels est une tâche difficile, et surtout source d'erreurs. Cela est dû à plusieurs problèmes:

- Le problème majeur de la synthèse des interfaces de communication est le niveau d'abstraction considéré. Beaucoup de travaux utilisent la synthèse des interfaces à un niveau trop bas, ce qui rend difficile sa réalisation et parfois, la communication entre composants devient un véritable goulot d'étranglement.
- Les différents outils existants imposent généralement des restrictions qui peuvent décourager les développeurs à les utiliser. De plus, certains outils utilisent des langages non standards que les développeurs n'ont pas l'habitude d'utiliser.
- Les interfaces ont des structures complexes, leur conception nécessite des compétences issues des domaines du logiciel et du matériel.

- Pour pouvoir satisfaire les contraintes de la spécification, les concepteurs sont obligés d'explorer l'ensemble de l'espace des solutions possibles, ce qui n'est pas toujours évident.
- Les différentes approches reposent sur le fait que les solutions aux sous-problèmes de la synthèse des interfaces sont spécifiques à l'environnement de l'application et les hypothèses sous-jacentes.
- La majorité des approches existantes souffrent du problème de la validation des interfaces de communication.
- La plupart des approches de cosynthèse ne prennent pas en charge la réutilisation des interfaces de communication. En conséquence, la modification d’un composant du système force les concepteurs à concevoir à nouveau une interface de communication.
- Le temps de communication pose de plus en plus de problèmes pour la conception des systèmes mixtes.

2. Objectifs et éléments fondamentaux de notre approche

L’objectif principal de notre travail est de rechercher une technique pour la communication entre un composant matériel et un composant logiciel qui permet de surmonter les problèmes que nous venons de citer. Les objectifs de cette technique sont :

- ✓ La cospécification facile et aisée des composants matériels et des composants logiciels. Le concepteur n’aura pas à se soucier des détails de bas niveau qui sont souvent la source d'erreurs principale dans les systèmes mixtes. Ces détails seront en fait générés automatiquement par la technique recherchée en se basant sur des propriétés globales du système à réaliser.
- ✓ La spécification de propriété non fonctionnelle qui permettrait d’étudier tôt dans un processus de conception le composant de codesign composite à réaliser.
- ✓ La spécification de systèmes hétérogènes dans lesquels un composant de codesign pourrait être réalisé à la base d’un composant matériel et d’un composant logiciel.

Afin d’atteindre les objectifs cités, et puisque le problème majeur de la synthèse des interfaces de communication est le niveau d'abstraction considéré qui est trop bas pour gérer la complexité croissante des systèmes mixtes, nous nous sommes basés sur une idée principale qui consiste à ramener le problème de la communication dans un système de codesign à un niveau d’abstraction très élevé qui est le niveau architecture.

Il y a quelques années cette idée ne nous aurait menés nulle part car le niveau architecture n’existait que pour la décomposition informelle d’un système sous forme de sous

systèmes moins complexes. Aujourd'hui, la phase architecture dans un processus de conception s'est dotée de concepts, outils et méthodologies qui ont donné naissance à une nouvelle discipline du génie logiciel connue sous le nom d'Architecture Logicielle.

Le choix de l'architecture logicielle comme approche pour la communication dans un système de codesign repose sur le constat suivant : la conception de systèmes complexes prise en charge par l'architecture logicielle et la conception de composants de codesign composites utilisent le même principe général : c'est la conception par composition, appelée souvent assemblage de composant en architecture logicielle. De plus, le codesign et l'architecture logicielle utilisent un même concept concernant les briques de base : C'est le concept de composant.

Le raisonnement à divers niveaux d'abstraction est une technique largement utilisée en architecture logicielle pour isoler les aspects technologiques et réduire la complexité des logiciels. Par cette technique, l'architecture logicielle permet d'une part, une exploitation aisée et efficace des diverses innovations technologiques en génie logiciel et d'autre part, une prise en charge de la construction de systèmes très complexes utilisant des grains de diverses tailles provenant de diverses sources [5].

Si à sa naissance et durant plusieurs années l'architecture logicielle se basait sur des composants abstraits très complexes, ce qui correspond à la programmation dans le large (programming in the large), aujourd'hui plusieurs approches existent pour permettre le raffinement d'une architecture pour arriver ensuite à une spécification architecturale très proche du niveau implémentation. Ainsi, la spécification d'une architecture peut aujourd'hui être exprimée à base de composants très complexes et abstraits ou à base de composants très fins complètement définis au niveau implémentation. C'est dans ce dernier contexte que les composants de codesign primitifs peuvent se positionner.

Le choix de l'approche architecture logicielle pour la résolution du problème de la communication dans un système de codesign est dû aussi en grande partie au fait que l'approche architecture logicielle parait aujourd'hui unanimement acceptée comme voie prometteuse pour la production de logiciels de haute qualité.

La dernière version d'UML (UML2.3)[6] témoigne de l'importance grandissante de l'approche architecture logicielle et de son intérêt. La version UML2.0 [6] représente un pas important dans la prise en considération des concepts fondamentaux de l'architecture logicielle. C'est face à la faiblesse d'UML1.4 et UML1.5 à représenter une architecture

[6] http://www.omg.org/spec/UML/2.3/

logicielle [7] [8] et la nécessité de rendre plus efficace les approches MDA (Model Driven Architecture) [9] qu'UML2.0 fut défini. A travers UML 2.0, l'OMG a introduit un ensemble de spécifications concernant notamment les concepts de composants, ports et connecteurs. Ces derniers représentent les concepts fondamentaux sur lesquels, repose la spécification d'architecture logicielle. Les composants et les connecteurs sont décrits par les aspects suivants: l'interface, le type, la sémantique ou comportement, les contraintes d'exploitation et les propriétés non fonctionnelles [10].

La spécification d'architecture logicielle repose principalement sur deux concepts importants : le concept de composant et le concept de connecteur. Un composant exprime via des interfaces les ressources dont il a besoin (appelées généralement : services requis) et les ressources qu'il est capable de produire (appelées généralement : services fournis). Une interface est le lieu où sont spécifiées les conditions d'exploitation d'un service. Ces conditions sont représentées à deux niveaux : un niveau structurel et un niveau comportemental. Le niveau structurel sert à contrôler si les deux interfaces connectées correspondent correctement au niveau des noms des opérations et des types de données utilisées. Le niveau comportemental, spécifié souvent sous forme de pré-conditions et de post-conditions, expose les règles et les contraintes d'exploitation des services fournis.

La spécification de la correspondance entre un service requis et un service fourni se fait par la définition d'un connecteur entre ces deux services. En plus de cette correspondance, un connecteur est le lieu où sont définies les interactions entre les services connectés. Une interaction valide doit respecter le niveau comportemental spécifié au niveau d'une interface.

Une architecture logicielle est décrite à l'aide de langage de description d'architecture logicielle (ADL)[7]. Un ADL a pour objectif premier d'aider les concepteurs à structurer et composer leurs éléments logiciels pour former des applications. Il permet aussi d'analyser et de vérifier tôt dans le cycle de développement les propriétés que le futur système devra satisfaire.

Un des problèmes auquel nous étions confrontés est la détermination de l'approche architecture logicielle (donc l'ADL si celui-ci existe) qui serait la plus intéressante pour nos travaux. Dans ce contexte, il fallait déterminer l'approche qui propose le modèle de composant le plus adéquat et celle qui supporte les niveaux d'abstraction les plus intéressants. A titre d'exemple, les ADL Fractal et ArchJava sont trop proches du niveau implémentation, ce qui les rend inadaptés pour une spécification de plus haut niveau d'abstraction.

---

[7] Architecture Description Language

Dans notre cas, nous nous sommes basés sur le modèle de composant de l'approche IASA (Integrated Approach to Software Architecture) définie dans le contexte d'un travail de recherche entre l'ESI[8] et le laboratoire LINA[9] de l'Université de Nantes [5]. Ce modèle permet la spécification libre à un niveau d'abstraction très proche des modèles mentaux de topologies très variées. Cette capacité est due principalement au concept d'interface de ce modèle qui permet de manipuler les éléments structurels et comportementaux d'une interface contrairement aux autres approches qui considèrent qu'une interface est atomique.

L'approche IASA, si elle fournit une définition très poussée des modèles de composant et de connecteurs, ne s'est pas encore dotée d'un ADL officiel. La première version de l'ADL de l'approche IASA, appelée SEAL (Simple and eXtensible Architecture Language) [11] a été définie juste pour démontrer le concept de composant exécutable à un haut niveau d'abstraction. Or, le modèle nécessite un langage plus complet pour la spécification des divers caractéristiques du modèle, notamment la spécification orientée aspect d'architecture logicielle et la spécification des propriétés de déploiement. Ensuite, pour combler les défaillances de SEAL et pour permettre d'exploiter le modèle IASA de manière efficace, un nouveau langage appelé 3ADL (Architecture, Aspect and Action Description Language) a été défini pour l'approche IASA. La forme XML de ce langage a été aussi définie d'où l'appellation x3ADL (eXtensible Architecture, Aspect and Action Description Language) [12]. Dans le cadre de notre travail sur le codesign matériel/logiciel, nous avons enrichi le langage x3ADL pour lui permettre de prendre en charge à la fois les composants matériels et les composants logiciels.

Avec le modèle de composant IASA et x3ADL il est possible de définir des architectures à divers niveaux d'abstraction. L'approche IASA est dotée d'un processus de développement complètement basé sur la conception par assemblage de composant. Les gros composants abstraits sont raffinés et conçus par assemblage de composants moins complexes. Le raffinement s'arrête lorsque des composants dits primitifs sont utilisés. Un composant primitif est complètement défini au niveau implémentation.

Une autre caractéristique qui nous a guidés vers l'approche IASA, c'est la spécification explicite des propriétés de déploiement pour chaque instance de composant[10]. Ainsi, il est possible de spécifier pour une instance de composant la nature réelle que l'instance aura à l'exécution. Dans IASA, le processus de transformation d'une instance de

---

[8] Ecole nationale supérieure d'informatique http://www.esi.dz/

[9] Laboratoire d'informatique de Nantes Atlantique http://www.lina.univ-nantes.fr/

[10] Un composant est un type dans l'approche IASA.

composant dans la nature spécifiée par les propriétés de déploiement repose essentiellement sur le concept d'enveloppe qui est associé à chaque instance de composant.

La nature hétérogène des composants de codesign impose l'utilisation de technologies d'implémentations différentes. Notre objectif est de pouvoir réaliser la communication entre un composant logiciel implémenté en JAVA et un composant matériel implémenté dans un langage de description de matériel, en l'occurrence le langage VHDL[11] et cela selon l'approche intégrée IASA. Dans notre cas, un composant primitif IASA peut être soit un composant matériel, soit un composant logiciel. Le composant de codesign composite sera alors considéré comme un composant mixte.

3. Les qualités de notre approche

L'objectif principal de notre travail est de résoudre le problème de la communication dans un système de codesign, en se basant sur une approche d'architecture logicielle qui est l'approche IASA. Cette dernière permet de porter le raisonnement à un haut niveau d'abstraction. Notre approche sera ainsi caractérisée par les qualités d'une approche d'architecture logicielle dont les plus importantes sont :

- ✓ La facilité de la spécification et de la validation d'une architecture mixte. En effet, il est plus facile pour un concepteur de raisonner dans un espace très proche de ses premières idées sur le système que dans l'espace de la technologie d'implémentation qui est assez complexe à maîtriser.
- ✓ La spécification de la solution se fait par assemblage de composant. Ce dernier est un processus simple ne nécessitant pas de grandes compétences et une maîtrise des technologies d'implémentation comme ceci est le cas avec les techniques actuelles de synthèse des communications. La transformation d'une architecture IASA en un composant de codesign composite étant à la charge de l'ADL de l'approche IASA.
- ✓ La capacité de cacher les technologies d'implémentation et le déploiement des composants. Ainsi, il serait possible de réaliser des systèmes dont des parties sont réalisées en utilisant des composants matériels et d'autres parties en utilisant des composants logiciels, sans se soucier du processus de synthèse de la vue implémentation de chaque instance de composant.

[11] http://www.vhdl.org/

4. Organisation du mémoire

Notre mémoire est organisé de la manière suivante :

Le premier chapitre présente le domaine du codesign matériel/logiciel, puis passe en revue les différentes étapes du processus de la conception concurrente qui sont: la cospécification, la modélisation, le partitionnement matériel/logiciel et la covérification. Les éléments fondamentaux liés à chaque étape sont étudiés.

Le second chapitre expose un état de l'art sur la synthèse des interfaces de communication dans le codesign. Des outils et des approches issus du monde de la recherche et de l'industrie sont discutés. En dernier, ce chapitre donne un bilan sur les différentes démarches de cosynthèse ainsi que les problèmes de la communication entre le matériel et le logiciel.

Le chapitre 3 présente les éléments fondamentaux de l'architecture logicielle. L'accent est mis sur l'approche intégrée d'architecture logicielle (IASA) qui représente notre contribution pour résoudre le problème de la synthèse des interfaces de communication dans le codesign.

Le chapitre 4 est consacré à la synthèse des communications selon l'approche IASA. Les concepts et les techniques développés dans l'approche IASA afin de lui permettre de gérer un processus de communication entre un composant matériel et un composant logiciel d'un système de codesign sont expliqués.

Le chapitre 5 est dédié à la présentation de l'environnement de développement que nous avons développé pour valider notre travail. Ce chapitre est argumenté par des études de cas sur la communication entre un composant hardware et un composant software.

Nous terminons ce mémoire par une conclusion générale ainsi qu'un ensemble de perspectives des travaux en cours et à venir.

CHAPITRE 1
LE CODESIGN MATERIEL/LOGICIEL

1.1 Introduction

L'industrie électronique, avec ses indénombrables branches dans tous les secteurs (industriels et services), poursuit sa conquête de la complexité. Cette complexité, mesurée en millions de transistors par puce, double tous les 18 mois [13], entraînant derrière elle une énorme croissance des capacités de traitement de l'information et une croissance également très importante de la taille des logiciels. Concevoir de nouvelles applications en utilisant toutes les potentialités qu'ouvre cette croissance devient un défi permanent. Aussi, la complexité croissante des systèmes pour lesquels la réalisation résulte de l'association d'une partie matérielle et d'une partie logicielle, la diversité des choix technologiques, les contraintes de coûts et la concurrence sur les produits et les services qui impose à tous la sévère loi du "time to market" où il faut être capable de passer très rapidement de la spécification au produit, nécessitent l'utilisation de nouvelles approches pour diminuer le temps de conception tout en garantissant les performances exigées. Le codesign est l'une de ces approches.

L'approche de conception traditionnelle procède par la conception du matériel d'abord, ensuite les composants logiciels sont conçus après la conception et le prototypage du matériel. Cela laisse peu de flexibilité dans l'évaluation des différentes options de conception. Avec la séparation entre la conception du matériel et celle du logiciel, il devient difficile d'optimiser la conception dans son ensemble. Une telle démarche de conception est en particulier inadéquate lors de la conception de systèmes exigeant des performances strictes et un petit délai de conception.

Le but de ce chapitre est de se familiariser avec le vocabulaire du codesign matériel/logiciel, pour cela nous allons donner quelques définitions du codesign ainsi que les motivations et les domaines d'application. Nous présenterons ensuite les différentes étapes du processus de la conception concurrente à savoir la cospécification, la modélisation, le partitionnement matériel/logiciel et la covérification.

1.2 Définition du terme codesign

Le terme "Hardware/Software Concurrent Design" souvent abrégé par "Hardware/Software Codesign" et qui se traduit par "conception conjointe

matérielle/logicielle" représente un processus de conception complet permettant aux concepteurs de transformer correctement les spécifications d'un système en un produit industriel comportant une partie logicielle et une partie matérielle et satisfaisant les contraintes fonctionnelles et non fonctionnelles (temps de réponse, débit, taille, coût, délai de fabrication) de son cahier des charges[14]. Il doit également permettre d'accroître la qualité de conception et de réduire le temps de développement.

L'une des premières définitions du codesign fut donnée par Franke et Purvis [15] comme étant un « Processus de conception de système qui combine le matériel et le logiciel dès les premières étapes de conception pour exploiter la flexibilité de conception et une allocation efficace des fonctions ». Pour Wolf [16], « Le matériel et le logiciel doivent être conçus ensemble pour s'assurer que non seulement l'implémentation fonctionne correctement mais qu'elle répond aussi aux objectifs de performances, de coûts et de fiabilité » . Kumar [17] ajoute que « Le Hardware/software codesign regroupe des concepts et des idées provenant de trois disciplines principales au sein de la conception des systèmes: la modélisation au niveau système, la conception matérielle et la conception de logiciels ». Selon Kalavade [18], « La clé principale du codesign est d'éviter l'isolement entre la conception du matériel et celle du logiciel. Cette stratégie permet de procéder à la conception du matériel et des logiciels en parallèle, avec une rétroaction et une interaction entre les deux pour que la conception progresse ». D'après De Micheli [1], « Le Hardware/software codesign consiste à répondre aux objectifs du système en exploitant la synergie du matériel et du logiciel par leur conception concurrente ».

Les concepts les plus importants qui se dégagent de ces différentes définitions sont le développement du matériel et du logiciel en parallèle et l'interaction entre eux pour produire une conception qui assure les performances attendues et les spécifications fonctionnelles du système.

1.3 Origines du codesign

La complexité des systèmes embarqués pour les applications numériques en termes de fonctionnalités ne cesse d'augmenter, cette évolution est liée aux progrès technologiques dans le domaine de la micro-électronique où plusieurs millions de transistors sont intégrables au sein d'une même puce. Pour répondre aux exigences de ces applications l'ensemble des solutions à analyser est de plus en plus étendu. L'architecture cible pour ces applications numériques peut être entièrement logicielle, composée par exemple de processeurs spécialisés

de type DSP[1] et/ou de processeurs généraux de type RISC[2]. La solution duale consiste à réaliser un système embarqué à partir d'une architecture cible entièrement matérielle [19], composée par exemple d'ASIC[3]. Ces deux solutions possèdent chacune des avantages et des inconvénients:

1. Dans un modèle matériel, les éléments du circuit fonctionnent en parallèle. L'utilisation de plusieurs éléments de circuit, augmente la quantité de travail réalisé dans un seul cycle d'horloge. Le software fonctionne de façon séquentielle. En utilisant plusieurs opérations, un programme logiciel prend plus de temps pour se terminer. Ainsi, un concepteur de matériel résout les problèmes par décomposition spatiale et un concepteur de logiciels résout les problèmes par décomposition temporelle [20].
2. L'implémentation en matériel présente une performance supérieure, en termes de vitesse d'exécution, par rapport à l'implémentation logicielle.
3. Par rapport au matériel, l'implémentation logicielle demande un temps de développement plus court car le travail est centré sur l'écriture des programmes, ce qui réduit le nombre d'étapes dans le cycle de conception.
4. Le coût pour une solution entièrement matérielle est habituellement beaucoup plus élevé que celui pour une solution logicielle à cause de la taille du matériel utilisé, et le temps de développement.
5. L'implémentation en matériel ne permet pas une prise en charge facile des évolutions des spécifications dans la mesure où l'architecture est figée, tandis que le logiciel offre l'avantage de la flexibilité. Actuellement, l'emploi des technologies de matériel reconfigurable (dont les principaux représentants sont les FPGA[4]) [21], apporte la flexibilité aussi au matériel.

Une autre solution consiste à combiner les avantages des deux approches précédentes. L'architecture résultante est alors qualifiée d'hétérogène dans la mesure où le système numérique généré est composé d'une partie codée sur un ou plusieurs processeurs et d'une partie câblée sur une structure matérielle. Les performances désirées pour les systèmes embarqués sont ainsi atteintes tout en minimisant la surface et en garantissant une certaine flexibilité. La conception de systèmes utilisant une architecture logicielle/matérielle n'est pas

---

[1] Digital Signal Processor
[2] Reduced Instruction Set Computer
[3] Application Specific Integrated Circuit
[4] Field Programmable Gate Array

nouvelle. Cependant, la complexité des systèmes embarqués ne permet plus de séparer les phases de conception des parties logicielles et matérielles et d'utiliser des techniques manuelles pour aboutir à des solutions efficaces dans un temps raisonnable [19].

Le terme Hardware/Software codesign est apparu au début des années 1990 pour décrire des problèmes de conception de circuit intégré [22]. Une classe de concepteurs qui étaient en grande partie différents des concepteurs de circuits intégrés, intégraient les microprocesseurs avec les composants matériels standards. Il était clair que la conception des systèmes à base de microprocesseurs allait devenir une importante discipline de conception pour les concepteurs de circuits intégrés, et le travail des chercheurs sur le développement de quelques approches de base pour la conception des logiciels embarqués fonctionnant sur les processeurs, avait ouvert la route vers une méthodologie de codesign matériel/logiciel.

L'un des premiers travaux sur le codesign fut le système SOS de Prakash et Parker [23] de l'université de Californie, qui pouvait synthétiser une topologie arbitraire de multiprocesseur et allouer des processus sur le multiprocesseur. Environ un an plus tard, l'atelier CODES dans le Colorado et l'atelier CASHE en Autriche introduisaient plusieurs éléments dans des travaux de recherche qui ont évolué en parallèle [22]. De cela, le partitionnement matériel/logiciel est apparu comme une première étape importante dans la création des modèles et des algorithmes. Deux premiers systèmes, Vulcan de Stanford [24] et COSYMA de l'université technique de Braunschweig [25], proposaient des approches complémentaires pour ce problème fondamental. Dans les années suivantes, le codesign est passé d'une discipline émergente à une technologie courante.

1.4 Les domaines d'application du codesign

Les domaines d'application de la conception conjointe sont très nombreux et différents, puisqu'ils correspondent aux applications qui contiennent du matériel et du logiciel et qui exigent des performances strictes. Parmi ces domaines :

1.4.1 Les systèmes embarqués

Un système embarqué (Embedded system en anglais) est un système digital, réactif, autonome, dédié à une tâche bien précise, disposant de l'ensemble des éléments physiques utiles à son fonctionnement (processeur, mémoire, capteurs/actionneurs), soumis à diverses contraintes (temps de réaction, disponibilité, fiabilité, robustesse, coût), piloté par un logiciel et complètement intégré dans le système qu'il contrôle [26]. Le codesign est applicable dans le

domaine des systèmes embarqués car un système embarqué intègre du logiciel et du matériel conjointement pour assurer des fonctionnalités souvent critiques. Initialement, les systèmes embarqués ont été utilisés pour des applications temps-réel critiques comme le contrôle (des fusées, missiles, satellites) et la production d'énergie. Actuellement, les systèmes embarqués apparaissent dans des domaines grand public tels que les appareils électroniques.

1.4.2 Les systèmes de télécommunication

Avec la percée rapide de l'électronique et l'effervescence des réseaux sans fil, de l'internet et du multimédia, les systèmes de télécommunication appartiennent aujourd'hui à l'activité qui a la plus forte croissance industrielle. Ces systèmes nécessitent, par nature une spécification hétérogène et une mise en œuvre avec des styles architecturaux hétérogènes. À cause de l'accélération des délais de commercialisation et la complexité en croissance exponentielle, ces systèmes nécessitent des compromis entre le coût et les performances, la puissance et la flexibilité. De cela les systèmes de télécommunication sont un candidat par excellence pour le codesign [27].

1.4.3 Les systèmes de traitement spécifiques

Le traitement de toutes formes d'informations: signal, image, parole, etc. Dans le passé, le matériel avait la grande part dans la réalisation de ces systèmes. Ensuite, les performances des microprocesseurs spécifiques (DSP) ont donné au logiciel une part plus importante [14]. Les exigences liées à ces systèmes sont principalement : un faible coût, une faible consommation et pour certaines applications une vitesse élevée [19].

1.4.4 Les accélérateurs d'exécution logicielle

Ils consistent à améliorer les performances des applications décrites à l'aide d'un algorithme purement logiciel et qui nécessitent des puissances de calcul élevées, en implémentant des sections critiques du logiciel en matériel. Cela dans le but de satisfaire des contraintes de temps ou de réduire le temps d'exécution global d'un programme [28].

1.5 Les avantages du codesign

Parmi les avantages du codesign, citons les suivants [2]:

- **Flexibilité de conception :** qui permet de prendre en charge plus facilement les changements de spécification causés par une modification dans le cahier des charges,

suite à des besoins de l'utilisateur. Aussi, la flexibilité garantit l'évolutivité du système car la plupart des produits sont fabriqués en plusieurs versions.

- **Réduction du temps de mise à disposition sur le marché :** le codesign peut réduire considérablement le temps de conception des systèmes par le développement en parallèle du matériel et du logiciel, entrainant une réduction du temps de mise à disposition sur le marché "time to market". Cela implique que le produit est mis sur le marché avec peu de concurrence ou pas, garantissant ainsi un gain meilleur.
- **Exploration efficace de l'espace de conception :** les systèmes complexes sont en général répartis sur plusieurs sous-systèmes ce qui constitue plusieurs alternatives de conception, car chaque sous-système est conçu de manière isolée. L'un des points forts du codesign est que les décisions prises au sujet d'un sous-système peuvent être utilisées pour guider la conception des autres. Donc le codesign permet une exploration efficace de l'ensemble des solutions envisageables.
- **Simplification de l'intégration et de la phase de test :** le codesign permet de réduire les erreurs provenant de l'intégration des sous-systèmes et la vérification du système dans son ensemble. Cela minimise non seulement le temps nécessaire pour ces phases, mais aussi les risques d'avoir à faire d'importantes révisions à la conception. De plus, les techniques de cosimulation et de covérification peuvent également être utilisées pour réduire la nécessité d'un test d'intégration.
- **Meilleur rapport coût/efficacité :** la réduction du temps de conception des systèmes par l'utilisation du codesign, entraine une réduction dans le coût de développement tout en respectant les performances exigées grâce à l'exploration efficace de l'ensemble des solutions.
- **Prototypage :** un prototype permet de prédire les fonctionnalités d'un système avant qu'il ne soit effectivement produit. Le codesign permet de développer des prototypes plus facilement et plus rapidement que les méthodes de conception classiques, cela est dû au fait que le codesign modélise un système dans son ensemble.

1.6 Comparaison entre la conception traditionnelle et le codesign

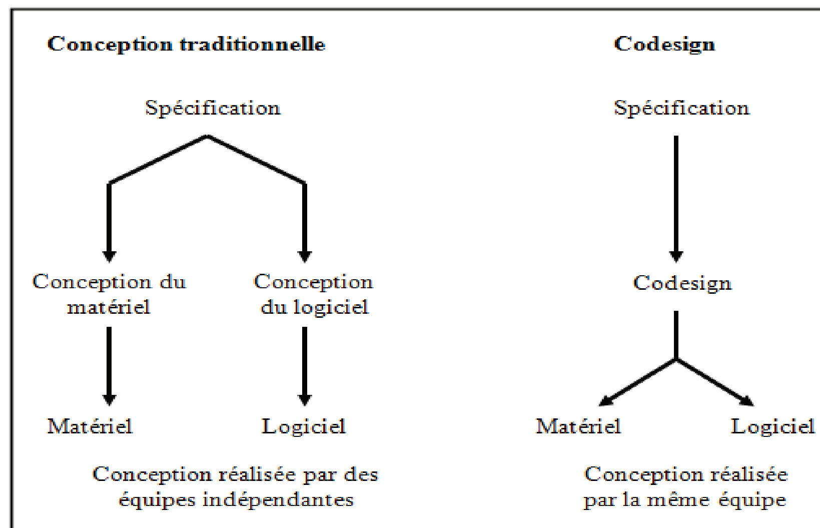


Figure 1.1 : Conception traditionnelle et codesign

Dans la méthodologie traditionnelle de conception de systèmes contenant du matériel et du logiciel, les parties logicielles et matérielles sont développées séparément dès les premières étapes du processus de conception. L'équipe du matériel (concepteurs d'architectures matérielles) réalise la conversion de la spécification dans un format exécutable (langage de description de matériel), puis elle réalise la synthèse et la génération des circuits intégrés. L'équipe du logiciel (informaticiens) est responsable de l'écriture du code qui va être compilé et exécuté. Ensuite, les deux équipes réalisent l'intégration physique des deux parties [29].

Cette séparation entraine un manque de dialogue et d'interaction entre les concepteurs des deux parties conduisant ensuite à des difficultés importantes découvertes tardivement pendant la phase d'intégration du logiciel et du matériel. Les difficultés rencontrées nécessitent parfois d'importantes modifications dans le matériel ou le logiciel. Ces problèmes émergent car cette approche restreint les possibilités d'exploration des différentes solutions où certaines fonctionnalités pourraient migrer du logiciel vers le matériel ou vice-versa, ce qui mène à une perte d'argent et de temps. De plus, les changements de spécification tardifs obligent des fois les concepteurs à refaire le travail.

Le codesign a été introduit pour faire face aux problèmes de la conception traditionnelle. La démarche du codesign consiste à éviter la séparation ente la conception du matériel et celle du logiciel. Cela évite les surprises rencontrées lors de la phase d'intégration

puisque les erreurs de conception sont détectées beaucoup plus tôt. De plus, le codesign répond au besoin de l'industrie, dans le secteur des hautes technologies, en développant des systèmes de plus en plus performants capables d'exécuter des applications de plus en plus complexes, dans des délais beaucoup plus courts. Le codesign permet d'obtenir un compromis performances/coûts très élevé contrairement aux méthodes de développement classiques. Aussi, il assure une productivité meilleure car la conception du matériel et la conception du logiciel sont effectuées simultanément.

1.7 Processus général de codesign

Le but d'une méthodologie de codesign est l'obtention d'un produit fini respectant les contraintes fixées par le cahier des charges pour un temps de développement et un coût beaucoup plus réduits. Mais, ce qui rend complexe le travail des codesigners, c'est que le codesign comporte les étapes classiques du génie logiciel et de la micro-électronique (conception de circuits intégrés à très grande échelle).

Il existe un certain nombre de projets de codesign, la plupart aux Etats-Unis. Parmi ces projets : Ptolemy [30], Polis/Metropolis [31], CoWare [32] et dans les différents travaux de recherche, les méthodologies de conception impliquent plusieurs étapes aboutissant à la génération d'architectures optimisées. Il s'agit principalement d'une spécification unifiée qui décrit les fonctionnalités attendues du système suivie de la modélisation qui fournit une représentation sous-jacente et formelle. Une fois les étapes de spécification et de modélisation achevées, il faut déterminer le choix de réalisation, logicielle ou matérielle, pour chacune des tâches composant l'application, c'est le rôle du partitionnement qui a pour objectif de trouver une répartition optimisée des tâches qui respecte les contraintes de la spécification et qui minimise une certaine fonction de coût (surface, consommation, fréquence). Vient ensuite, l'étape de la synthèse, la synthèse du logiciel, la synthèse du matériel, la conception d'interface permettant la communication entre le logiciel et le matériel et la dernière étape qui consiste à l'intégration finale des composants du système. La succession de ces étapes forme le flot typique d'une approche de conception conjointe schématisé dans la figure 1.2.

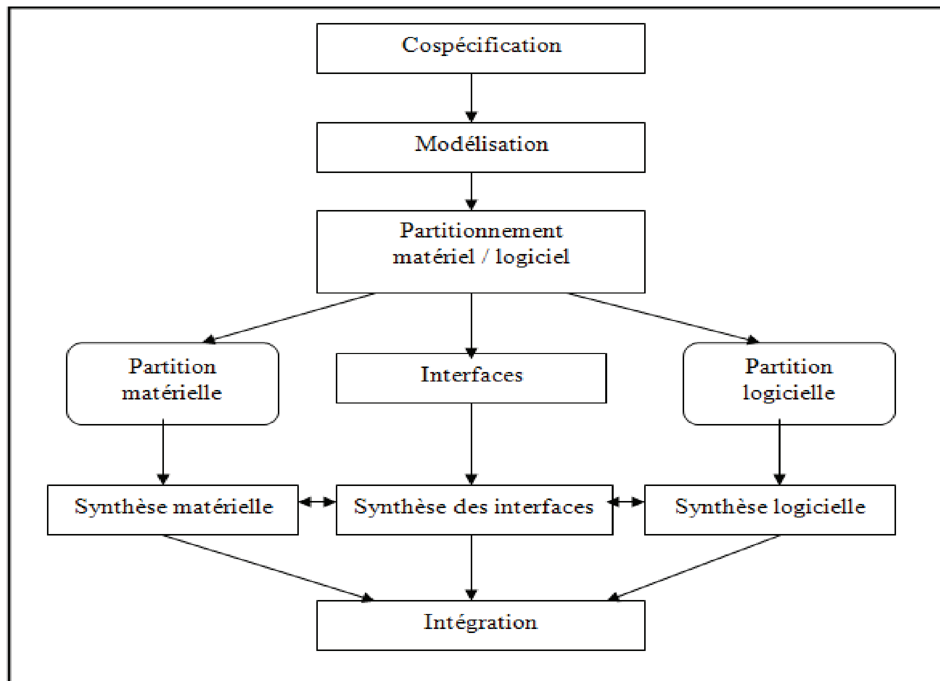


Figure 1.2 : Processus général de codesign

1.7.1 La cospécification

C'est la première étape dans le processus de codesign, elle consiste à décrire les fonctionnalités attendues d'un système ainsi que toutes les contraintes qu'il doit satisfaire, à un niveau de détails suffisant permettant de prévoir son comportement complet, sans ambigüité et sans préciser les implémentations. Cette spécification unifiée constitue donc le premier pas vers l'unification de la conception dans un processus de codesign. La cospécification est une étape déterminante car elle a un impact direct sur les résultats. De plus, elle sert de lien entre les besoins exprimés dans le cahier des charges et le développement du système par les concepteurs.

Dans la littérature, la spécification des systèmes complexes peut être différenciée selon plusieurs contextes, par exemple pour Gogniat [19] la spécification de l'application se situe au niveau système et se décompose en deux parties : la spécification fonctionnelle et la spécification comportementale. La spécification fonctionnelle correspond à la description des fonctions à réaliser par le système et peut être représentée par un ensemble de boîtes noires interconnectées. La spécification comportementale correspond à la description du comportement interne de chaque fonction du système. Hommais [33] parle de deux niveaux de spécification, la spécification fonctionnelle qui décrit le comportement de l'application (ce que le système doit être capable de faire) et les spécifications non-fonctionnelles qui sont

les contraintes externes que le système devra supporter (environnement, consommation, surface de silicium, débit). Plusieurs chercheurs [34, 35] distinguent deux méthodes de spécification des systèmes mixtes, la spécification homogène et la spécification hétérogène.

1.7.1.1 La spécification homogène

Utilise un langage de spécification pour spécifier les composants matériels et logiciels d'un système mixte. La figure 1.3 présente un environnement générique de codesign basé sur une spécification homogène [36]. Le principal défi d'un système de codesign utilisant l'approche homogène est d'analyser et de partager la spécification initiale entre des parties matérielles et logicielles où des concepts de haut niveau utilisés dans la spécification initiale doivent être mis en correspondance avec des langages de bas niveau (C et VHDL) pour représenter les parties matérielles et logicielles. Pour relever ce défi, la plupart des outils de codesign qui utilisent l'approche de la spécification homogène commencent avec un langage de spécification de bas niveau afin de réduire l'écart entre la spécification du système et les modèles matériels/logiciels [3]. Par exemple, le système Cosyma [37, 38] utilise une extension du langage C appelée C^x, prenant en charge les contraintes de temps et le processus de communication. Vulcan [24, 39, 40] utilise une autre extension du langage C appelée HardwareC permettant la spécification des composants matériels. Seuls quelques outils de codesign commencent avec un langage de spécification de haut niveau. Par exemple, Polis [31] utilise Esterel pour son langage de spécification.

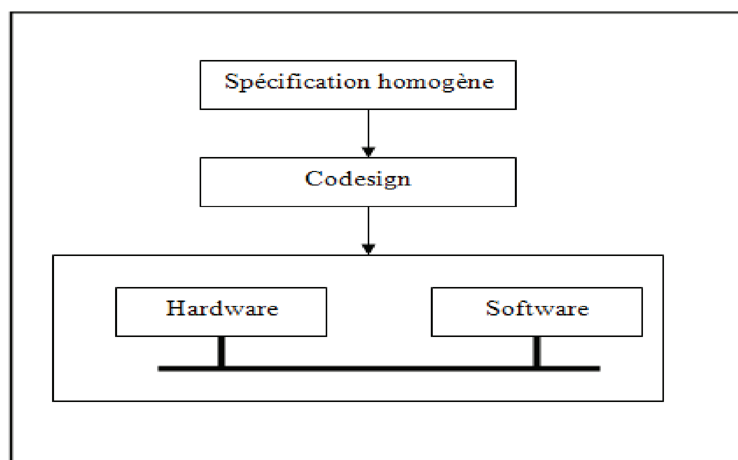


Figure 1.3 : Flot de codesign pour une spécification homogène

1.7.1.2 La spécification hétérogène

Utilise des langages spécifiques pour le matériel (par exemple, VHDL), et pour le logiciel (par exemple, C). La figure 1.4 présente un environnement générique de codesign basé sur une spécification hétérogène [35]. La spécification hétérogène permet d'avoir une spécification plus simple pour le logiciel et pour le matériel. Cependant, cette approche va à l'encontre de l'idée de base du codesign qui est la séparation entre le matériel et le logiciel le plus tardivement possible dans le processus de conception. De plus, cette approche suppose une affectation du matériel et du logiciel avant l'étape de partitionnement, ce qui peut éliminer des solutions potentielles de l'espace de conception. Aussi, elle rend l'étape de la validation et de la cosynthèse d'interfaces beaucoup plus difficiles. CoWare [32] est un exemple d'une méthodologie de codesign qui supporte la spécification hétérogène, en utilisant le VHDL pour le matériel et le langage C pour le logiciel.

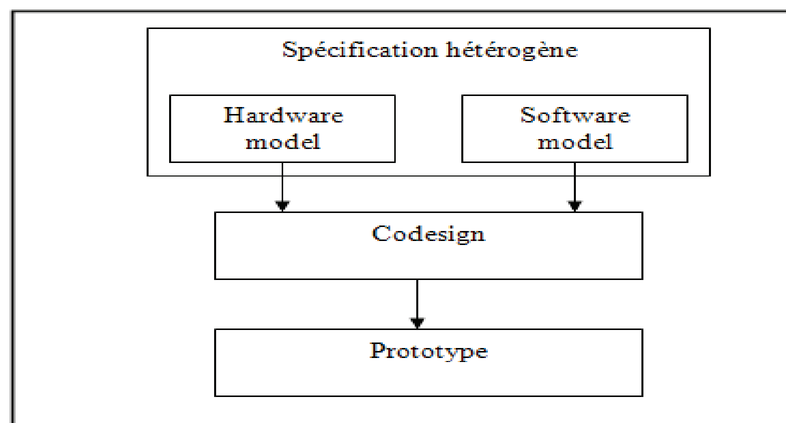


Figure 1.4 : Flot de codesign pour une spécification hétérogène

1.7.1.3 Les langages de spécification

L'objectif d'un langage de spécification est de décrire les fonctionnalités attendues d'un système d'une manière simple et précise. Un grand nombre de langages de spécification sont actuellement utilisés dans le codesign. Chaque langage revendique sa supériorité. Parmi ces langages :

- **Les langages de description de matériel (HDL[5]):** sont des langages qui supportent les concepts spécifiques aux systèmes matériels tels que les contraintes de temps, le parallélisme, la communication interprocessus et la réactivité. Les langages les plus utilisés sont VHDL, Verilog et HardwareC. VHDL est utilisé dans de nombreux travaux comme les travaux de [2, 41]. HardwareC est basé sur le langage C, conçu

[5] Hardware Description Languages

pour la synthèse du matériel. Il est utilisé dans les travaux de Gupta dans VULCAN [40].

- **Les langages de programmation :** la plupart des langages de programmation ont été utilisés pour la description du matériel. Il s'agit notamment de Fortran, C, Pascal, ADA, C + + et Java. Bien que ces langages fournissent des facilités pour la spécification au niveau système, ils ne prennent pas en charge des caractéristiques essentielles pour les systèmes matériels telles que les contraintes de temps, la concurrence et la communication. Beaucoup de travaux de recherche ont tenté d'étendre les langages de programmation pour la modélisation du matériel (par exemple: le langage C^x et HardwareC qui sont des extensions du langage C).
- **Les langages de description formelle :** basés sur la technique de la description formelle FDT[6], les langages formels ont été créés pour la description des protocoles afin de permettre leurs vérifications. Ils sont généralement utilisés pour la description des systèmes distribués. Les langages les plus importants dans ce domaine sont SDL, LOTOS et ESTELLE. SDL [42] est utilisé pour spécifier les systèmes de télécommunication. LOTOS [43] est basé sur une algèbre de processus et est utilisé pour la spécification des systèmes concurrents, distribués et pour la spécification des protocoles. ESTELLE [44] est un langage pour la spécification des protocoles.
- **Les langages des systèmes temps réel :** les systèmes temps réel sont des systèmes dont le comportement doit respecter des contraintes temporelles strictes. Esterel et StateCharts sont des exemples de langages pour les systèmes temps réel. Esterel [45] est un langage de programmation synchrone orienté flots de contrôle, il est utilisé pour spécifier des systèmes réactifs en temps réel. StateCharts [46] est un langage de spécification graphique pour les systèmes réactifs.
- **Les langages de conception au niveau système (SLDL[7]):** sont utilisés pour capturer les spécifications et les modèles intégrés au niveau d'abstraction système. SpecC et SystemC sont des exemples de SLDL. SpecC [47] est basé sur le langage C. SystemC [48] est une bibliothèque C + +, il a des points communs avec SpecC et est largement utilisé dans la modélisation au niveau système.
- **Les méthodes orientées objet :** ces méthodes sont destinées à la description des systèmes complexes à un très haut niveau d’abstraction. L’idée principale est la décomposition des systèmes complexes en sous-systèmes plus faciles à maîtriser.

---

[6] Formal Description Technique
[7] System Level Design Languages

1.7.1.4 Exigences pour les langages de spécification

Un langage de spécification doit garantir les aspects suivants:

- **La concurrence:** est souvent confondue avec le parallélisme dans le contexte du codesign matériel / logiciel, mais en réalité ces deux termes se rapportent à des concepts très différents. La concurrence est la capacité d'exécuter des opérations simultanées qui sont totalement indépendantes. Le parallélisme est la capacité d'exécuter des opérations simultanées qui peuvent s'exécuter sur différents processeurs ou des éléments de circuit [20]. Ainsi, la concurrence se rapporte à un modèle d'application, tandis que le parallélisme se rapporte à l'implémentation de ce modèle. La concurrence peut être au niveau de l'instruction ou au niveau processus suivant la granularité de la spécification.
- **La hiérarchie:** permet de décomposer un système en sous-systèmes afin de gérer sa complexité. Elle peut être comportementale ou structurelle [49]. La hiérarchie comportementale permet de cacher des sous-comportements locaux qui peuvent être séquentiels ou concurrents dans un module. Les procédures et les sous-états permettent d'exprimer la hiérarchie comportementale. La hiérarchie structurelle permet de décomposer un système en sous-systèmes interconnectés interagissant à travers des frontières bien définies. La communication peut être spécifiée à l'aide de signaux ou avec un niveau d'abstraction plus haut en utilisant des canaux encapsulant des protocoles de communication complexes.
- **La communication:** permet l'échange des données et des signaux de contrôle entre des modules concurrents [50]. Il existe deux types de communication: la communication par passage de messages où les processus échangent des données par l'intermédiaire de messages convoyés par des canaux de communication, et la communication par mémoire partagée où chaque communication nécessite la lecture ou l'écriture d'une partie de la mémoire pour échanger des données.
- **La synchronisation:** permet de coordonner l'exécution des différents modules d'un système. Il existe deux types de synchronisation, synchrone et asynchrone [29]. Dans le mode d'exécution synchrone, l'avancement de chaque module est lié aux autres. Les protocoles du type rendez-vous ou mémoire partagée avec des sémaphores sont des exemples du mode synchrone. A l'opposé, dans le mode asynchrone les états de l'émetteur et du récepteur ne sont pas liés. Le mode asynchrone peut être réalisé par un protocole du type FIFO ou par une mémoire partagée avec des moniteurs.

- **Les contraintes de temps:** qui caractérisent les systèmes embarqués et les systèmes temps réel. Par exemple les contraintes de fonctionnement qui représentent le temps consommé pour l'exécution d'un comportement.
- **Les transitions d'états:** la prise en charge explicite des transitions d'états est importante, car la transition d'état facilite le contrôle de la modélisation et la réactivité des systèmes embarqués.
- **La gestion des exceptions:** des exceptions telles que les interruptions, se produisent souvent dans les systèmes embarqués. Quand une interruption survient, le système doit faire la transition vers un nouvel état pour traiter l'interruption. Une fois que l'interruption est traitée, le système doit revenir à l'état initial. Les langages de spécification doivent être en mesure de traiter les exceptions.
- **La vérification formelle:** est souhaitable pour les langages de spécification car elle fournit un mécanisme permettant de vérifier la spécification en utilisant des méthodes mathématiques formelles.

Le choix du langage le plus adéquat pour une application donnée n'est pas une tâche facile car il existe une variété de langages de spécification. Cependant, aucun langage ne s'est imposé comme le meilleur langage de spécification pour soutenir tous les types d'applications. Cela, est dû au fait que chaque langage de spécification est plus efficace pour certaines caractéristiques alors qu'il présente des faiblesses pour d'autres. Par exemple certains de ces langages sont plus adaptés à la spécification sous forme d'automate d'états finis (SDL ou StateCharts), d'autres sont orientés flot de données (Lustre), des langages tels que (C, C + +, java) sont plus adaptés aux descriptions algorithmiques alors que des langages tels que (VHDL ou verilog) sont destinés à la description matérielle. La sélection d'un langage est généralement un compromis entre plusieurs critères tels que la puissance expressive du langage, les capacités d'automatisation fournies par le modèle sous-jacent et la disponibilité des outils et des méthodes permettant de soutenir le langage [36].

1.7.2 La modélisation

Afin de concevoir des systèmes qui répondent aux exigences demandées en termes de performances, de coût et de fiabilité, la conception doit être fondée sur des modèles de calcul formels pour permettre des améliorations dans toutes les étapes du processus de conception, de la spécification à l'implémentation [51]. La modélisation est le processus de

conceptualisation et d'affinement des spécifications. Un modèle de calcul (MoC[8]) est une notation conceptuelle qui décrit le comportement du système désiré [3]. En règle générale, les modèles de calcul sont représentés d'une manière formelle, en utilisant, par exemple, des fonctions mathématiques ou des notations de la théorie des ensembles. Cela, permet d'établir une sémantique bien définie et des techniques d'application formelles [52].

Les outils de codesign utilisent les langages de spécification comme entrée. Ensuite, pour permettre des affinements au cours du processus de conception, les spécifications initiales sont transformées en formes intermédiaires basées sur les modèles de calcul qui sont plus facile à transformer pour gérer les exigences et les contraintes de calcul. Idéalement, un modèle de calcul doit prendre en charge la concurrence, la hiérarchie, la réactivité, le temps et des méthodes de communication. Le formalisme utilisé pour la modélisation doit être indépendant des technologies de réalisation et ne doit pas prédéfinir une implémentation particulière afin de ne pas restreindre l'espace de recherche des solutions potentielles [19]. Pour Edwards et al. [53], un modèle formel d'une conception est défini par les éléments suivants:

1. Une spécification fonctionnelle, donnée comme un ensemble de relations explicites ou implicites qui impliquent des entrées, des sorties et des informations internes (états).
2. Un ensemble de propriétés que la conception doit satisfaire, données comme un ensemble de relations sur les entrées, les sorties et les états, qui peuvent être vérifiées par rapport à la spécification fonctionnelle.
3. Un ensemble d'indices de performances qui permettent d'évaluer la qualité de la conception en termes de coût, de fiabilité, de vitesse et de taille, donnés comme un ensemble d'équations impliquant aussi les entrées et les sorties.
4. Un ensemble de contraintes sur les indices de performances, spécifiées comme un ensemble d'inégalités.

1.7.2.1 Les modèles de calcul

Plusieurs modèles de calcul ont été développés pour représenter les systèmes hétérogènes. Les chercheurs ont classé les MoC en fonction de différents critères. Gajski et al. [54] classent les modèles de calculs en fonction de leur orientation en cinq classes:

[8] Model of Computation

- Les modèles orientés état: utilisent les états pour décrire les systèmes et les événements qui déclenchent les transitions entre les états. Exemple, les automates à états finis (FSM[9]).
- Les modèles orientés activité: décrivent un système comme un ensemble d'activités reliées par des données ou des dépendances d'exécution. Exemple, les graphes flots de données (DFG[10]).
- Les modèles orientés structure: sont utilisés pour décrire les modules physiques des systèmes et les interconnections entre eux.
- Les modèles orientés données: représentent un système comme une collection de données avec leurs attributs, les appartenances aux classes et les différentes interactions.
- Les modèles hétérogènes : fusionnent les caractéristiques des différents modèles dans un modèle hétérogène pour représenter un système complexe.

En plus des catégories décrites ci-dessus, Bosman [55] propose une classe de modèles orientés temps pour prendre en charge les notions de temps dans les modèles de calcul.

Pour Jerraya et al. [35], le modèle de calcul d'une spécification donnée peut être considéré comme la combinaison de deux notions orthogonales: 1) modèle de communication et (2) modèle de contrôle. Le modèle de communication peut être synchrone ou distribué. Le modèle de contrôle peut être classé en modèle flots de contrôle ou flots de données. Le modèle synchrone est souhaitable pour les applications monoprocesseurs, tandis que le modèle distribué est préférable pour les applications multiprocesseurs. Le modèle orienté contrôle met l'accent sur les séquences de contrôle plutôt que sur le calcul lui-même. Le modèle orienté données exprime le comportement comme un ensemble de transformations de données. Pour [52], les modèles de calcul peuvent être subdivisés en deux classes, modèles axés sur les processus et modèles axés sur les états. Les modèles basés sur les processus sont orientés données et sont généralement utilisés dans les modèles de comportement du système pour décrire les fonctionnalités désirées de l'application. Les modèles basés sur les états mettent l'accent sur la représentation du contrôle. Jantsch et Sander [56] classent les modèles de calcul en fonction de leur abstraction du temps. Ils définissent les classes de modèles suivantes:

---

[9] Finite State Machine
[10] Data Flow Graph

- Modèles à temps continu, le temps est représenté par un ensemble continu, généralement des nombres réels.
- Modèles à temps discret, où chaque événement est associé à un instant du temps et le temps est représenté par un ensemble discret, comme les nombres naturels.
- Modèles synchrones, le temps est également représenté par un ensemble discret, mais l'unité de temps primaire n'est pas une unité physique, mais plutôt abstraite où chaque événement se produit dans un cycle d'évaluation spécifique (intervalle de temps ou cycle d'horloge).
- Modèles sans notions de temps, la seule commande sur la présence des événements est déterminée par les relations causales. Par exemple, si un événement A dépend d'un autre événement B, l'événement A doit se produire après l'événement B.

Parmi les modèles de calcul qui sont utilisés dans les systèmes de codesign, nous pouvons citer:

1.7.2.1.1 Les automates à états finis (FSM)

Le modèle FSM est une machine séquentielle numérique qui se caractérise par un ensemble d'états, un ensemble d'entrées, un ensemble de sorties, une fonction de transition d'états et une fonction de sortie [20]. Le système est décrit comme un ensemble d'états qui représentent les entités de ce modèle, et de valeurs d'entrée qui peuvent déclencher une transition d'un état à un autre. L'exécution est une succession strictement ordonnée d'états et de transitions. Les FSM sont des modèles bien adaptés pour exprimer la logique de contrôle, étant donné que le comportement temporel de ces systèmes est plus facilement représenté sous forme d'états et de transitions entre les états. Les modèles à base de FSM se prêtent bien à une analyse formelle et peuvent, de ce fait, être utilisés pour vérifier l'absence de comportements inattendus du système [57]. Le principal inconvénient des automates à états finis est le développement d'une manière exponentielle du nombre des états avec l'augmentation de la complexité des systèmes en raison d'absence de la hiérarchie et de la concurrence. De plus, une faible variation de la spécification peut produire un grand changement de l'automate [51]. Pour remédier aux limitations des FSM classiques, des recherches ont proposé plusieurs dérivés des FSM classiques. Parmi ces extensions :

- **Codesign Finite State Machine (CFSM) :** permet d'opérer en termes d'événements survenant à un moment donné, plutôt que des valeurs, car le modèle en fonction d'un événement est plus adapté pour les systèmes de contrôle, et le comportement du

système est défini comme une séquence d'événements [58]. Les CFSM sont largement utilisés comme des formes intermédiaires dans les systèmes de codesign avec les langages de haut niveau. Par exemple, POLIS [31] est basé sur le modèle CFSM.

1.7.2.1.2 Les réseaux de Petri

Un réseau de Petri est un outil de modélisation graphique et mathématique composé d'un ensemble de places, d'un ensemble de transitions et d'un ensemble d'arcs [59]. Avec un réseau de Petri, il est possible de représenter un grand nombre de caractéristiques d'un système complexe comme la mise en séquence, le branchement, la synchronisation, le partage de ressources et la concurrence [60]. Les réseaux de Petri supportent la concurrence, mais leur principal inconvénient est le manque de décomposition hiérarchique. Un autre problème, surtout pour les systèmes temps réel, est que les réseaux de Perti ne permettent pas d'exprimer les contraintes de temps. Des variations de réseaux de Petri ont été conçues pour résoudre ces problèmes. Par exemple: les réseaux de Petri hiérarchiques qui sont appropriés pour la modélisation des systèmes complexes, car ils supportent la hiérarchie en plus de maintenir les propriétés des réseaux de Petri telles que la concurrence et l'asynchronisme [3].

1.7.2.1.3 Les systèmes à événements discrets (DE[11])

Un système à événements discrets est un système dynamique qui évolue avec l'apparition brutale, à intervalles irréguliers, éventuellement inconnus, d'événements causant un changement du système en déclenchant une transition d'un état à un autre [58]. L'interaction entre les processus se fait à travers des signaux, qui sont des collections d'événements [61]. La modélisation à événements discrets est souvent utilisée pour la simulation de matériel numérique. Par exemple, le langage Verilog a été conçu comme un langage d'entrée pour un simulateur à événements discrets. Le langage VHDL a également un modèle de calcul sous-jacent à événements discrets [53]. Même si ce type de représentation est utile pour des systèmes temps réel, le principal inconvénient de la modélisation à événements discrets est qu'elle est coûteuse car elle nécessite le tri de tous les événements selon leur temps d'apparition [51, 53].

[11] Discrete Event

1.7.2.1.4 Les graphes flots de données (DFG)

Les systèmes sont spécifiés à l'aide d'un graphe orienté où les nœuds, qui représentent les opérations, sont reliés par des arcs orientés qui représentent les dépendances (chemins de données) [3]. Les nœuds peuvent avoir des granularités différentes qui varient de la granularité de la simple instruction à celle de la fonction [57]. Ce modèle supporte la hiérarchie, et l'activité d'un nœud peut être un autre graphe flots de données [60]. Plusieurs variantes des graphes flots de données ont été proposées dans la littérature comme les flots de données synchrones (SDF). Dans un SDF, quand un acteur opère, il consomme un nombre fixe de jetons de chaque port d'entrée, et produit un nombre fixe de jetons à chaque port de sortie [62].

1.7.2.1.5 Les modèles synchrones/ réactifs (SR)

Les systèmes réactifs sont des systèmes qui interagissent continuellement avec leur environnement. Le modèle synchrone pour les systèmes réactifs est une représentation pour les systèmes temps-réel. Dans ce modèle de calcul, les réponses de sortie sont produites simultanément avec les stimuli d'entrée. Dans ce cas, il n'ya pas de délai observable dans la réaction. L'avantage de cette hypothèse (appelée hypothèse de synchronie) est que les systèmes sont plus faciles à décrire et à analyser [51]. Les modèles SR sont utilisés pour décrire des applications comprenant une logique de contrôle complexe et concurrente, et pour des applications temps-réel avec des contraintes de sûreté.

1.7.2.1.6 La modélisation par rendez-vous

Dans ce modèle, les acteurs représentent des processus qui communiquent par rendez-vous instantané. Les récepteurs gèrent les points de rendez-vous. Une tentative de mettre un jeton dans un récepteur ne sera pas complète jusqu'à ce que une autre tentative correspondante soit effectuée pour obtenir un jeton du même récepteur, et vice versa. En conséquence, le processus qui atteint un point de rendez-vous en premier doit attendre jusqu'à ce que l'autre processus atteigne aussi le même point de rendez-vous [61]. Ce modèle de calcul a été utilisé dans un certain nombre de langages concurrents comme Lotos.

1.7.2.1.7 Les modèles orientés objet

Le langage UML est en train de gagner l’attention de la communauté s’intéressant au logiciel embarqué qui le considère comme étant une solution possible pour travailler à un

niveau d'abstraction plus élevé [57]. Plusieurs travaux ont été réalisés ces dernières années pour étudier la possibilité d'utiliser UML comme langage de base pour la spécification des systèmes embarqués, ou plus particulièrement des plateformes embarquées. L'approche détaillée dans [63] combine les modèles UML et SDL[12]. La spécification au plus haut niveau est réalisée en UML et le comportement de chaque module est spécifié en SDL.

1.7.2.1.8 Les modèles hétérogènes

Beaucoup de recherches ont été menées pour combiner différents modèles de calcul. Cette approche a l'avantage, que dans un système complexe composé de plusieurs parties ayant chacune ses propres caractéristiques, le modèle de calcul approprié peut être utilisé pour chaque partie du système. Néanmoins, puisque le modèle du système est basé sur plusieurs modèles de calcul, la sémantique de l'interaction des différents modèles doit être définie, ce qui n'est pas une tâche facile. Cela amplifie encore le problème de validation, car le modèle du système n'est pas basé sur une sémantique unique [56]. Une des initiatives les plus connues pour la définition d'un environnement d'accueil de différents modèles de calcul est le système Ptolemy [30]. Ptolemy est un environnement de modélisation et de simulation qui définit des sémantiques précises pour interfacer les différents modèles comme dans [64].

Plusieurs études comparatives ont été réalisées entre les différents modèles de calcul [3, 51, 55], et à chaque fois le constat est le même: il n'existe pas un modèle idéal capable de soutenir tous les types de systèmes de codesign. En effet, certain formalismes de modélisation sont bien adaptés pour les systèmes orientés données, d'autre sont souhaitables pour les systèmes orientés contrôle, mais peu de représentations soutiennent les deux. Parfois, plusieurs modèles de calcul sont nécessaires pour la conception d'un système complet. Cependant, le meilleur modèle pour une application donnée est celui qui coïncide étroitement avec les caractéristiques du système qu'il modélise. Un modèle de calcul est différent d'un langage utilisé pour la spécification d'un système. Le même modèle de calcul peut être utilisé par plusieurs langages, et il existe des langages de description qui gèrent différents modèles. Ainsi, le modèle et le langage sont deux concepts différents. Cependant, il y a parfois une réelle confusion entre le langage de spécification et le modèle sous-jacent et il est parfois difficile de voir une frontière claire entre le modèle et le langage. C'est le cas par exemple des langages dits synchrones utilisés pour modéliser les systèmes réactifs [51].

[12] Specification and Description Language

1.7.3 Le partitionnement matériel/logiciel

Après les phases de spécification et de modélisation survient une phase de partitionnement ayant pour but d'assurer la transformation des spécifications du système en une architecture composée d'une partie matérielle et d'une partie logicielle. Le partitionnement matériel/logiciel est le problème de division des fonctionnalités d'une application en une partie qui s'exécute comme des instructions séquentielles sur des microprocesseurs (le logiciel) et une partie qui s'exécute en tant que circuits parallèles sur des ASIC ou des FPGA (le matériel), dans le but d'atteindre les objectifs fixés par la conception tels que la performance, la puissance, la taille et le coût [4]. Le partitionnement s'effectue habituellement en deux phases: la sélection d'une architecture matérielle et l'allocation des éléments du modèle fonctionnel spécifiant l'application sur les composants de cette architecture. Pour cela, il est nécessaire d'avoir une connaissance précise, d'une part des caractéristiques logicielles et matérielles des fonctions modélisant l'application et d'autre part du modèle d'architecture cible considéré. Mann [65] divise le problème du partitionnement matériel/logiciel en 5 sous-problèmes:

P1: Les contraintes matérielles et logicielles doivent être satisfaites.

P2: Essayer de satisfaire les contraintes matérielles tout en minimisant le coût du logiciel.

P3: Essayer de satisfaire les contraintes logicielles tout en minimisant le coût du matériel.

P4: Minimiser le coût des communications entre le matériel et le logiciel tout en satisfaisant les contraintes du matériel et du logiciel.

P5: Gérer les contraintes de partitionnement, où le but du processus de partitionnement est de minimiser à la fois le coût du matériel et du logiciel.

La sélection des différentes partitions est souvent guidée par l'ensemble des contraintes définies initialement par le concepteur (espace réservé au logiciel et au matériel, temps d'exécution, taux de parallélisme, taux de communication, consommation, temps de conception, coût industriel, poids,...). En effet, une fonctionnalité donnée peut avoir diverses implémentations (en matériel ou en logiciel), chacune avec ses propres caractéristiques de coût et de performance. Il est donc nécessaire d'explorer différentes possibilités d'implémentation en matériel et en logiciel avant de se décider sur la partition du système qui satisfait de façon optimale les contraintes de coût et/ou de performance imposées[66]. Les techniques de partitionnement décrites dans la littérature peuvent être classées en fonction des critères suivants:

- Le degré d'automatisation allant d'une démarche manuelle à une démarche entièrement automatique.
- Le choix de l'architecture cible figée ou libre.
- Le niveau de granularité.
- Les métriques d'estimation.
- Les objectifs.

1.7.3.1 Méthodes de partitionnement matériel/logiciel

L'exploration de toutes les architectures possibles pour un système donné est impraticable en réalité, car pour un nombre de composants de l'architecture C et une granularité de la spécification de A atomes, l'ensemble des solutions possibles est exponentiel : C^A [60]. Traditionnellement, le partitionnement été réalisé manuellement. Toutefois, comme les systèmes sont devenus de plus en plus complexes, cette méthode est devenue impossible, et beaucoup d'efforts de recherche ont été entrepris pour automatiser le partitionnement autant que possible [67]. Il existe alors deux approches différentes pour le partitionnement.

- **Le partitionnement interactif:** se base sur l'expérience et l'esprit d'analyse du concepteur qui décide des partitionnements à estimer afin de trouver l'architecture qui permet de respecter les contraintes. Le partitionnement interactif s'applique généralement sur une architecture hétérogène à définir et s'appuie sur des estimateurs de performances statiques et/ou dynamiques du système pour guider le concepteur dans le choix d'une répartition [29].
- **Le partitionnement automatique:** met en pratique une heuristique pour trouver une solution qui s'adapte aux contraintes. L'heuristique a pour objectif de réduire l'espace des solutions et par conséquence le nombre de réalisations à tester. Pour cela, les systèmes automatiques utilisent une fonction de coût pondérée pour évaluer les partitions. Le partitionnement automatique d'une spécification est un problème complexe du fait du grand nombre de paramètres à considérer [57]. De plus, ce type de partitionnement ne prend pas en compte l'expérience et le bon sens des concepteurs.

1.7.3.2 Les algorithmes de partitionnement

Les méthodes de partitionnement sont généralement construites de la manière suivante [60]: une première partition est élaborée manuellement ou automatiquement. Ensuite, l'algorithme de partitionnement va déplacer des atomes entre le processeur logiciel et le

processeur matériel afin de diminuer la fonction coût et de respecter les contraintes. Le déplacement des atomes n'est cependant pas aléatoire mais se base sur des critères de proximité afin de limiter le nombre de communications entre le logiciel et le matériel. Après chaque déplacement, la fonction coût est réévaluée en estimant les différents critères et un partitionnement est de nouveau effectué jusqu'à l'obtention d'un minimum de la fonction coût. Parmi les algorithmes de recherche utilisés par les systèmes existants dans le domaine de la conception conjointe, nous trouvons:

- **Le recuit simulé (SA):** accepte des déplacements générant une augmentation momentanée de la fonction coût quand la réduction n'est pas possible [29], ce qui permet d'éviter d'être piégé dans un optimum local.
- **La recherche taboue (TS):** exploite les structures de données pour l'historique de recherche en tant que condition des futurs déplacements. Elle impose systématiquement des contraintes afin de permettre l'exploration de régions interdites [68].
- **Les algorithmes génétiques (GA):** sont des algorithmes de recherche générale, ils sont efficaces pour résoudre les problèmes d'optimisation combinatoire.
- **La programmation linéaire entière (ILP):** réalise le découpage en utilisant un algorithme à base de formulations de programmation linéaire.

Bien que les algorithmes de partitionnement heuristiques sont généralement très rapides et produisent des résultats à une proximité optimaux ou même optimaux pour les petits systèmes, leur efficacité (qualité de la solution trouvée) se dégrade considérablement lorsque la taille du problème augmente. Cela est dû au fait que, afin d'être rapide, de telles heuristiques évaluent seulement une petite fraction de l'espace de recherche. Quand la taille du problème augmente, l'espace de recherche augmente de façon exponentielle (2^n différentes façons de partitionner n composants). Par conséquent, si le système qui doit être partitionné est grand et les contraintes sont bien serrées (et elles le sont généralement), alors le pourcentage que le partitionnement heuristique ne trouve pas de partition valide est élevé. Pire encore, le concepteur ne sait pas si cela est dû à la faible performance de l'algorithme de partitionnement ou parce qu'il n'y a pas de partition valide [69]. Ces problèmes peuvent être surmontés en utilisant des algorithmes exacts. Toutefois, puisque la plupart des formulations du problème de partitionnement matériel/logiciel sont NP-difficiles, ces algorithmes ont des temps d'exécution exponentiels ce qui les rend inappropriés quand le nombre d'instances est grand.

1.7.3.3 Le niveau de granularité

La granularité correspond à la finesse de description qui varie du niveau instruction ou groupe d'instructions, au niveau bloc système [70]. Trois niveaux de granularité du partitionnement sont habituellement utilisés [14]: le niveau tâche, le niveau procédure et le niveau instruction. Pour le niveau tâche (coarse-grain partitioning), l'unité d'allocation est la fonction qui est considérée indivisible et dont le comportement n'est pas obligatoirement séquentiel. Pour le niveau procédure, une fonction est décomposée en un ensemble de séquences d'instructions appelées procédures et qui peuvent être allouées sur des processeurs différents. Pour le niveau instruction (fine-grain partitioning), l'unité d'allocation est la plus petite possible puisqu'il s'agit d'une instruction. L'utilisation d'un niveau de granularité fin concerne plutôt des systèmes de faible complexité ou une conception architecturale avancée qui se situe relativement tard dans le cycle de développement.

1.7.3.4 L'analyse des propriétés d'un partitionnement

Dans les systèmes mixtes, les contraintes jouent un rôle important dans la réussite d'une conception. Toutefois, les contraintes strictes nécessitent un effort supérieur de conception et par conséquent un grand besoin d'outils automatisés pour guider le concepteur dans les décisions critiques [41]. Pour renseigner le concepteur sur la qualité des solutions trouvées, des estimateurs sont utilisés afin de prédire les résultats de la conception sans aller jusqu'à la réalisation du système. Cependant, pour la recherche d'une configuration optimale, il est intéressant de pouvoir estimer rapidement les différentes métriques d'une partition donnée. Ce facteur (vitesse d'estimation) permet d'augmenter, dans un temps donné, le nombre de solutions explorées dans le cadre de l'approche interactive, et d'accélérer la convergence des algorithmes, dans le cadre d'une recherche automatique. Mais le problème ici, est que plus un modèle d'estimation est précis, plus le temps de calcul associé est grand [60].

1.7.3.5 La fonction de coût

L'obtention du partitionnement matériel/logiciel optimal nécessite une fonction de coût efficace qui dirige le processus de recherche de la solution. Pour cela, la fonction de coût doit être capable de gérer les contraintes pour fournir une solution réalisable qui répond à toutes les spécifications de conception. En outre, l'estimation rapide de la fonction de coût est essentielle à la réussite de tout algorithme de partitionnement matériel/logiciel [41]. La

fonction de coût joue donc un rôle déterminant dans le processus de partitionnement. Son importance vient du fait que le problème de partitionnement repose sur des objectifs contradictoires et la fonction coût est essentielle pour déterminer quel terme(s) a (ont) la plus grande importance.

1.7.3.6 L'ordonnancement

L'ordonnancement consiste à ordonner les tâches partitionnées dans chaque élément de traitement de telle sorte qu'une bonne utilisation du processeur soit atteinte, et le temps de communication entre les tâches internes et les tâches inter-processeur soit optimisé [68]. Cela revient à déterminer l'instant début de chaque tâche. Quand une ou plusieurs tâches doivent être exécutées sur un ou plusieurs processeurs matériels et/ou logiciels, un algorithme d'ordonnancement doit être appliqué pour décider de l'ordre d'exécution. Cet ordre doit permettre de minimiser le temps d'exécution global de l'application.

1.7.3.7 L'architecture cible

Une fois que les spécifications ont été validées, il faut déterminer l'architecture sur laquelle seront développées les fonctionnalités de l'application. Cette architecture doit satisfaire un ensemble de contraintes et minimiser un certain nombre de paramètres comme le temps de calcul, la consommation, le poids et le coût. Le choix de l'architecture cible est d'une grande importance et les démarches diffèrent selon que l'architecture se trouve imposée ou que l'architecture et ses composants sont à déterminer. La première situation est la plus commune et la plus simple. L'architecture cible est généralement une architecture générique constituée d'un microprocesseur, d'un ensemble de circuits matériels programmables ou d'ASIC et d'une mémoire commune. Le problème du partitionnement se réduit alors à un problème de partitionnement binaire matériel/logiciel pour l'allocation des éléments fonctionnels sur les composants de l'architecture et peut se résoudre de manière automatique. La deuxième situation est plus proche de la réalité industrielle, mais elle est beaucoup plus complexe à cause de la nature hétérogène de l'architecture cible et de la diversité des contraintes à satisfaire [14]. Une architecture cible dans le domaine du codesign matériel/logiciel est constituée d'un ensemble de composants: matériels (ASIC, FPGA), logiciels (processeurs) et de communication. Les processeurs peuvent être des processeurs généraux (RISC), des processeurs parallèles (VLIW) ou des processeurs spécifiques (ASIP) [36].

- **RISC** (Reduced Instruction Set Computer): possède un jeu d'instructions[13] réduit où chaque instruction effectue une seule opération élémentaire. Toutefois, les instructions complexes nécessitent un compilateur très évolué dans le cas de programmation en langage de haut niveau.
- **VLIW** (Very Long Instruction Word): plutôt que d'essayer de lancer plusieurs instructions indépendantes vers les unités, un VLIW met plusieurs opérations dans une seule instruction très longue, d'où son nom: Very Long Instruction Word.
- **ASIP** (Application Specific Instruction-set Processor): offre la possibilité de personnaliser des instructions, ce qui permet d'enrichir le jeu d'instructions avec des instructions spécifiques à l'application. Un ASIP donne un bon compromis performance/flexibilité.
- **ASIC** (Application-Specific Integrated Circuit): intègre sur une même puce tous les éléments actifs nécessaires à la réalisation d'une fonction ou d'un ensemble électronique. Un ASIC assure un contrôle total du produit. Cependant, il augmente le temps de développement.
- **FPGA** (Field-Programmable Gate Arrays): sont des circuits intégrés qui contiennent des blocs de logique programmables, ainsi que des interconnexions configurables entre ces blocs. Ils sont devenus de plus en plus populaires et les tendances récentes indiquent une croissance plus rapide de leur densité de transistors que les processeurs à usage général.

La figure 1.5 illustre l'architecture cible utilisée dans l'outil CUPSHOP [41]. De nombreux composants matériels peuvent fonctionner en parallèle. De plus, chaque composant matériel peut avoir une mémoire locale dédiée pour réduire les goulots d'étranglement de la mémoire partagée.

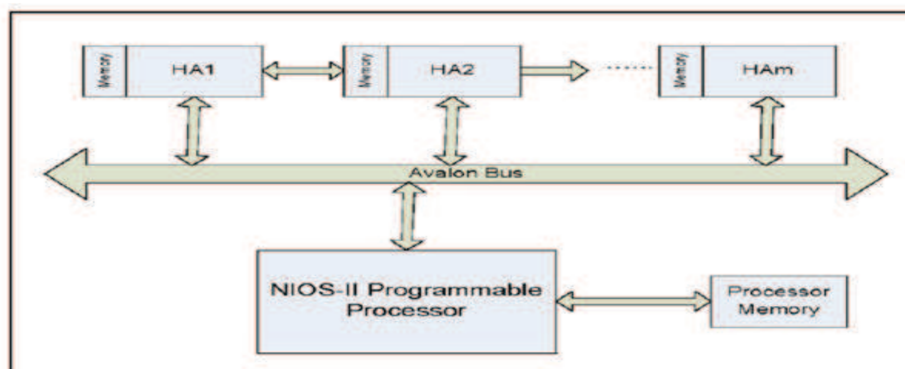


Figure 1.5 : Exemple d'architecture cible [41]

[13] Les opérations codées en langage machine qu'un processeur peut effectuer.

1.7.3.7.1 Les modèles de l'architecture cible

L'architecture cible utilisée pour implémenter une conception peut être monoprocesseur ou distribuée.

- **L'architecture monoprocesseur:** est constituée d'un processeur principal (agissant comme un contrôleur), d'un ensemble de composants matériels (ASIC, FPGA) et éventuellement d'une mémoire commune. La figure 1.6 présente un exemple d'une architecture monoprocesseur.

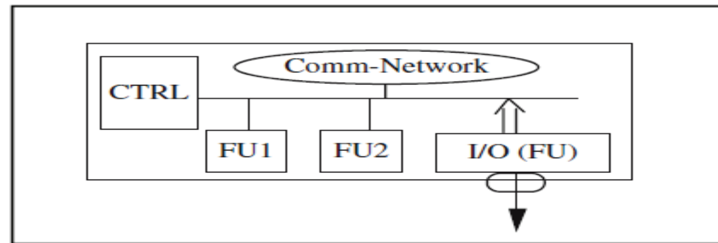


Figure 1.6 : Architecture monoprocesseur [35]

- **L'architecture distribuée:** exploite l'avantage du parallélisme en utilisant des éléments fonctionnant d'une manière concurrente. Elle est composée d'un ensemble de processeurs fonctionnant en parallèle. Plusieurs configurations de processeurs (monoprocesseur, multiprocesseur) et plusieurs modèles de communication (mémoire partagée, passage de message) peuvent être utilisés. La figure 1.7 présente un exemple d'une architecture multiprocesseur.

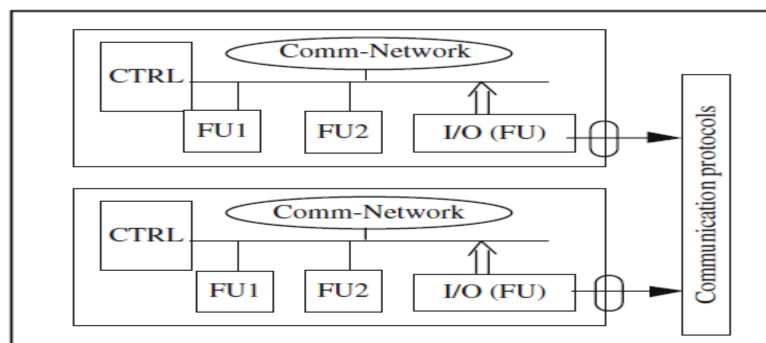


Figure 1.7 : Architecture multiprocesseur [35]

L'utilisation d'une architecture spécifique, bien adaptée à une application donnée, est plus efficace. Cependant, ce type d'architecture n'est pas assez flexible pour bien s'adapter dans un environnement de codesign à d'autres types d'application [36]. Plusieurs facteurs de coût et de performances dépendent directement du choix de l'architecture cible. Pour cela, il faut trouver le meilleur ensemble de composants pour implémenter les fonctionnalités d'un

système. Le problème de l'architecture cible est que le choix de ces composants doit se faire entre des composants matériels très rapides, spécialisés, mais très coûteux comme les ASIC, et des composants logiciels flexibles, moins chers mais moins rapides comme les processeurs généraux.

1.7.3.8 Travaux liés au partitionnement matériel/logiciel

Le partitionnement matériel/logiciel fait l'objet de nombreux travaux qui se différencient par la méthodologie de travail adoptée. Dans les premiers travaux, les algorithmes dépendaient fortement (coefficient de pondération de la fonction de coût) des caractéristiques de l'architecture cible choisie. Dans certains travaux, l'architecture cible est censée contenir une seule unité logicielle et une seule unité matérielle [24], tandis que d'autres n'imposent pas cette limitation. Pour l'approche COSYMA [25], les fonctions sont au départ toutes implantées en logiciel puis migrées vers le matériel jusqu'au respect des contraintes de performances. [57] propose une méthode basée sur un algorithme génétique couplé à une heuristique de Clustering. Ma et al [71] présentent une recherche taboue sur un réseau de neurones chaotique. Mann [69] donne un algorithme exact basé sur le branch-and-bound. Jigang et al [68] proposent un algorithme pour le partitionnement matériel/logiciel fonctionnel et l'ordonnancement. Stitt [72] donne une formulation du partitionnement matériel/logiciel qui considère plusieurs implémentations, obtenues à partir de l'exploration de l'espace de conception matériel de chaque région. Dans la méthodologie de [73], l'interface entre le matériel et le logiciel est transparente pour le concepteur de logiciels, et est basée sur des méthodes dynamiques d'interception. [74] travaille sur les techniques d'optimisation des problèmes multicritères pour améliorer l'efficacité des algorithmes. [41] présente un outil de partitionnement matériel/logiciel à un haut niveau d'abstraction (CUPSHOP), basé sur une méthodologie intégrée de haut niveau.

Bien qu'ils puissent fonctionner parfaitement dans leurs environnements de codesign propres, il est impossible de comparer les résultats obtenus, en raison des grandes différences dans les environnements de codesign et l'absence de critères de référence. Le problème du partitionnement matériel/logiciel est au cœur de l'activité de codesign. Trouver une partition optimale est difficile à cause du grand nombre des différentes caractéristiques des composants à prendre en considération et les frais généraux de communication entre le matériel et le logiciel. L'outil de partitionnement matériel/logiciel idéal doit produire automatiquement un

ensemble de partitions de haute qualité dans un temps de calcul réduit et prévisible. Malgré les efforts de recherche énormes dans les deux dernières décennies, actuellement, le partitionnement matériel/logiciel automatique n'est pas largement utilisé dans les outils industriels et commerciaux, et se limite à des outils académiques. Cette situation est le résultat de deux problèmes principaux, à savoir, les algorithmes de recherche inefficaces et le grand espace de recherche des modèles actuels [41].

Une fois le partitionnement matériel/logiciel terminé, les descriptions fonctionnelles sont transformées en code machine pour une implémentation en logiciel et en un ensemble de portes logiques pour une implémentation en matériel. C'est le rôle de l'étape de cosynthèse qui concerne la synthèse des parties matérielles et logicielles et la synthèse des interfaces de communication (la cosynthèse fera l'objet du prochain chapitre). Avant et après la cosynthèse, une covérification fonctionnelle est effectuée généralement par une technique de cosimulation.

1.7.4 La covérification

De nombreux chercheurs [19, 60, 75] considèrent qu'une phase de vérification est obligatoire après chaque étape du processus de conception des systèmes mixtes, afin d'éviter la propagation des erreurs entre les différents niveaux d'abstraction et de s'assurer que les raffinements successifs conduisent au même fonctionnement et respectent les contraintes de spécification. La vérification est définie comme le processus qui détermine que la conception, à différents niveaux d'abstractions, est correcte. La vérification des systèmes matériels/logiciels est appelée covérification. La covérification doit permettre de vérifier les parties matérielle et logicielle d'un système mixte, bien avant son implémentation physique [76]. Le problème de la vérification est particulièrement double pour les systèmes mixtes, qui sont composés d'un mélange hétérogène de modules matériels et logiciels, où la présence d'erreurs dans les premières phases du processus de conception peut conduire à un échec complet. De plus, la détection tardive demande des itérations et des investigations plus longues pour déterminer la source de l'erreur. En général, la covérification peut être effectuée par deux techniques principales qui sont la cosimulation et la vérification formelle.

1.7.4.1 La cosimulation

La cosimulation est la validation des fonctionnalités logicielles et matérielles dans une seule simulation [77]. Elle est utilisée pour observer le comportement du système complet

avant que la plateforme réelle ne soit disponible. Pendant la cosimulation, les fonctionnalités du modèle sont vérifiées en générant une séquence de stimuli d'entrée (jeu de test) qui est stimulée pour observer le comportement de la conception en cours de vérification, puis la séquence des sorties générées est comparée à la séquence des sorties prévues. Le processus de génération de test implique généralement une boucle dans laquelle la séquence de test est progressivement réévaluée et améliorée jusqu'à ce que les objectifs soient atteints. La cosimulation est ensuite réalisée en utilisant la séquence de test qui en résulte, et les réponses aux tests de cosimulation sont évaluées. Cette technique souffre essentiellement de la difficulté à trouver une séquence d'entrées appropriée pour mener le test, c'est-à-dire celle qui incite le système à générer une séquence de sorties riche en informations aidant à prendre des décisions sur la conformité du système [78].

Les problèmes de la cosimulation et du partitionnement matériel/logiciel sont souvent liés. En effet, la cosimulation peut permettre d'estimer le temps de calcul et l'utilisation dynamique des mémoires. Ces résultats peuvent alors confirmer ou réfuter le choix des composants avant la réalisation du prototype réel et aider à valider un partitionnement particulier. L'inconvénient de la simulation est qu'il est impossible d'appliquer une simulation pour tous les scénarios d'entrées possibles, pour cela, elle est considérée comme une approche incomplète. La cosimulation est très bonne dans la recherche des bugs, mais elle ne peut pas garantir leur absence. En conséquence, pour l'assurance de la qualité, la simulation est nécessaire mais pas suffisante. Le défi de la cosimulation est la gestion efficace et précise de l'interaction entre les composants hétérogènes. Les différences fondamentales dans les niveaux d'abstraction du matériel et du logiciel rendent les techniques de simulation difficiles à utiliser ensemble.

1.7.4.1.1 Approches de cosimulation

En dépit de la variété des architectures cibles, de l'efficacité des performances et des langages de description, les approches de cosimulation peuvent être classées en deux grandes catégories [75, 79] : cosimulation homogène et cosimulation hétérogène.

- **La cosimulation homogène :** Les environnements homogènes utilisent un moteur unique pour la simulation des composants matériels et logiciels. Ptolemy [30] et Polis [31] sont les travaux pionniers dans ce sens. Les environnements homogènes simplifient la modélisation de la conception et offrent de bonnes performances de

simulation. Toutefois, ils ne conviennent que dans une phase très initiale de la conception, avant le partitionnement matériel/logiciel.

- **La cosimulation hétérogène :** Les environnements hétérogènes assurent un réglage plus précis entre les composants matériels et logiciels par rapport aux environnements homogènes. Cela est fait par l'utilisation d'un niveau d'abstraction plus bas pour le comportement du matériel, et par l'évaluation du logiciel dans sa forme compilée qui est obtenue, de la description de haut niveau, avec des outils standards comme les compilateurs.

1.7.4.2 La vérification formelle

La vérification formelle consiste à prouver si le comportement d'un système, décrit selon un modèle formel, respecte des propriétés particulières, elles mêmes décrites selon un modèle formel [19]. La vérification formelle d'un système se fait à la manière des preuves mathématiques des théorèmes, sans se soucier des valeurs d'entrée. En effet, la vérification formelle est statique, elle est capable de travailler sans devoir énumérer ou simuler tous les états possibles du système. En employant ce type de méthode, il devient possible, dans certaines circonstances, de monter qu'une conception est correcte, sans appliquer des ensembles importants de jeux de test, puisque la preuve formelle est établie quelques soient les données en entrée. L'inconvénient de cette technique est qu'elle est limitée à des niveaux d'abstraction élevés, en conséquence, le concepteur ne dispose plus de détails de réalisation du système. La vérification formelle des systèmes mixtes dans l'industrie est généralement limitée à des techniques automatiques [75]. La condition préalable à l'application de ces techniques est la modélisation cohérente à un niveau d'abstraction adéquat.

1.7.4.3 Travaux liés à la covérification

Dans les différents travaux concernant la covérification matérielle/logicielle, la cosimulation reste la méthode la plus largement utilisée et cela quel que soit le niveau, mais l'importance des méthodes de vérification formelle augmente, en particulier pour les systèmes embarqués ayant des contraintes de sécurité critiques. Les travaux de Gupta et al. [39] utilisent une approche homogène typique de cosimulation. Cette approche s'appuie sur un simulateur unique personnalisé pour le matériel et le logiciel qui utilise une file d'attente d'événements. La stratégie de covérification de [78] est basée sur le concept du "multithreading". Dans cette approche, la partie logicielle est décrite par un ensemble de

threads communicants. [80] propose une validation des étapes de conception par cosimulation. La solution de cosimulation permet de connecter les simulateurs propres à chaque langage et ainsi profiter des possibilités de chacun. Cet environnement de cosimulation nommé MCI a été utilisé avec succès lors d'une expérience menée en collaboration avec le CENT (France Télécom). D'autres solutions utilisent des langages semi-formels comme UML suivi d'une compilation logicielle et d'une synthèse matérielle [76]. [79] présente une méthode de vérification par cosimulation, basée sur un modèle architectural consistant en une interaction du processeur avec des blocs de matériel via un bus commun. [81] donne une approche de covérification formelle basée sur une combinaison de model checking et des tests de conformité.

Toutes ces techniques présentent des avantages différents, mais aucune ne peut assurer la garantie de livraison de systèmes sans bug. La construction de systèmes mixtes réellement fiables nécessite une covérification matérielle/logicielle. Le défi majeur de la covérification concerne l'automatisation. Des tentatives de vérification au niveau système ont été menées pour vérifier tous les éléments de la conception d'une manière parallèle [82], permettant ainsi de découvrir les bugs tôt dans le cycle de développement lorsque le partitionnement du système est encore assez fluide pour permettre de faire des choix optimaux afin de régler le problème. Loghi et al. [75] pensent que l'avenir de la covérification matérielle/logicielle sera dominé par la combinaison de la cosimulation et la vérification formelle, car elles se complètent bien dans leur intention et leurs capacités. En particulier, pour la détection des traces des erreurs. Pour cela, de nouvelles techniques qui intègrent systématiquement des approches formelles et des approches basées sur la simulation sont nécessaires, afin de surmonter les limitations actuelles et arriver à des processus de vérification efficaces.

1.8 Conclusion

Dans ce chapitre, nous avons présenté le domaine du codesign matériel/logiciel et les différentes étapes du processus de la conception concurrente.

Le codesign s'impose de plus en plus dans le monde de la conception des systèmes contenant du matériel et du logiciel. Ce succès est dû au fait que le codesign tire profit des avantages du matériel et du logiciel en évitant leur séparation. Ainsi, cette démarche assure un meilleur compromis coût/performances et dans des délais beaucoup plus courts. Cependant, le codesign doit faire face à de nombreux problèmes, car il combine deux domaines très

différents non seulement dans le jargon des développeurs, mais aussi dans les pratiques de travail utilisées.

Malgré le nombre important des langages de spécification, aucun langage ne s'est imposé comme le meilleur langage de spécification pour soutenir tous les types d'applications. Afin de concevoir des systèmes qui répondent aux exigences demandées, la conception doit être fondée sur des modèles de calcul formels pour permettre des améliorations dans toutes les étapes du processus de conception. Ces modèles sont souvent confondus avec les langages de spécification où il est parfois difficile de voir une frontière claire entre eux. Trouver le partitionnement matériel/logiciel optimal est difficile à cause du grand nombre des différentes caractéristiques des composants à prendre en considération. Malgré les efforts de recherche énormes dans les deux dernières décennies, actuellement, le partitionnement matériel/ logiciel automatique est limité à des outils académiques. Avec la complexité croissante des systèmes mixtes, la covérification doit être exécutée en concurrence avec le codesign. En effet, pour garantir la fiabilité de ces systèmes, il est nécessaire que la covérification soit intégrée au codesign dès les premières phases.

Nous nous concentrons dans ce mémoire sur le problème de la synthèse des interfaces de communication entre le logiciel et le matériel, que nous allons présenter dans le prochain chapitre.

CHAPITRE 2
LA SYNTHESE DES INTERFACES DE COMMUNICATION DANS LE CODESIGN MATERIEL/LOGICIEL

2.1 Introduction

Un système mixte est composé d'un ensemble de modules logiciels et matériels qui ont besoin d'échanger des informations. Ces échanges sont désignés par le terme communication, chaque module est connecté aux canaux de communication par une interface. Pour garantir le bon fonctionnement du système global, il faut déterminer les protocoles et les interfaces requises par les différents sous-systèmes pour communiquer. Différents schémas et protocoles de communication peuvent être requis, de même que différentes topologies d'interconnexion selon les besoins du système. La cosynthèse permet de transformer les descriptions fonctionnelles en descriptions directement implantables sur les processeurs matériels et logiciels de l'architecture cible. L'objectif de la synthèse des communications consiste à appliquer les stratégies de communication qui permettent d'effectuer les transferts de données à moindre coût tout en respectant les contraintes de la spécification.

Dans ce chapitre, nous allons présenter les concepts de base de la synthèse des interfaces de communication tels que : les niveaux d'abstraction de la communication, les types de communication, les types de conception des interfaces de communication, les composants de la communication, les topologies du réseau de communication et les modèles de signalisations pour la synchronisation logicielle/matérielle. Nous présenterons ensuite quelques approches et quelques outils de synthèse. Enfin, nous terminerons ce chapitre par un bilan sur les différentes démarches de cosynthèse ainsi que les problèmes auxquels la communication entre le matériel et le logiciel doit faire face.

2.2 Les interfaces logicielles/matérielles

Une interface logicielle/matérielle est un adaptateur de communication entre le logiciel exécuté par le processeur et une ressource matérielle (unité de communication, mémoire, contrôleur) (Figure 2.1). L'implémentation de ces interfaces peut être faite en logiciel ou en matériel, et peut être aussi mixte (logiciel et matériel) [83].

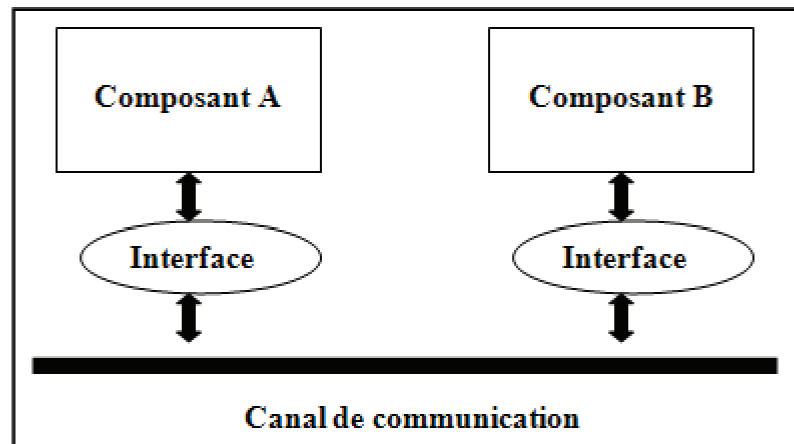


Figure 2.1 : Interfaces de communication

Une interface, qu'elle soit associée à un composant logiciel ou matériel possède un ensemble de ports qui représentent les points de communication du composant avec l'extérieur, et les opérations effectuées sur ces ports spécifient l'interaction entre le composant et le reste du système. Un canal de communication prend en charge l'échange des données entre les modules, il englobe tous les détails de communication et réalise l'échange des données entre les modules qu'il interconnecte [84].

2.2.1 Caractéristiques des interfaces de communication

[19, 84, 85] Soulignent que plusieurs paramètres doivent être pris en considération durant l'élaboration des interfaces de communication:

• **La topologie:** correspond à la structure des communications, elle peut être à base de bus, de connexions point à point ou d'une solution hybride.

• **Les supports de communication:** peuvent être des buffers, des FIFO, des bus, des mémoires partagées ou des mémoires à double ports.

• **Les types de transferts:** peuvent être synchrones ou asynchrones. Dans le cas synchrone, l'émetteur et le récepteur échangent directement les données sans mémorisation. A l'opposé, dans le cas asynchrone les données échangées sont mémorisées.

• **Les protocoles de communication:** correspondent aux mécanismes de transferts bloquants ou non bloquants. Dans le cas de la communication synchrone, le protocole doit être bloquant car le transfert ne peut se faire que lorsque l'émetteur et le récepteur soient prêts à échanger les données. Par contre, si la communication est asynchrone, l'émetteur et le récepteur accèdent aux données indépendamment l'un de l'autre.

• **Les modes de transferts:** représentent le style utilisé pour effectuer un transfert, qui peut être : un DMA[1] ou une gestion par le CPU.

• **Les types de données transmises:** par exemple des données génériques ou données avec représentation fixe.

• **Le comportement:** correspond à la description des détails de la communication.

• **Les caractéristiques de performance:** comme le débit de traitement des données offert par un composant.

2.3 La synthèse du matériel

La synthèse matérielle est la transformation d'un modèle décrit à un haut niveau d'abstraction (la spécification de l'application) en une description bas niveau (système physique) qui garantit les mêmes fonctionnalités. La synthèse du matériel comprend un nombre important d'étapes dont l'allocation des ressources, la synthèse logique, le plan de masse, le placement, le routage ou encore la génération des masques [19]. Des outils de synthèse de haut niveau peuvent être utilisés lors de la synthèse matérielle, ces outils prennent en entrée [33]:

- La spécification fonctionnelle de la tâche,
- Les contraintes non fonctionnelles (débit, fréquence, consommation) soit provenant des spécifications de départ, soit issues de l'évaluation du système.

2.4 La synthèse du logiciel

La synthèse logicielle est la génération d'un programme correct provenant d'une spécification de haut niveau et exécutable par le processeur cible. Si plusieurs tâches sont assignées à une même ressource matérielle, et dans le cas où elles ne peuvent être fusionnées, l'utilisation d'un système d'exploitation multitâche est obligatoire. Le code binaire exécutable peut être généré automatiquement à l'aide du compilateur associé à la cible. Malheureusement, les résultats rapidement fournis par cette technique sont souvent moins optimaux en termes de vitesse, de consommation ou d'optimisation mémoire, que le code assembleur obtenu manuellement [86].

[1] Direct Memory Access : pour accélérer les transferts, un périphérique peut devenir maître du bus et accéder directement à la mémoire.

2.5 La communication dans un système de codesign

Dans le codesign, trois types de communication peuvent être considérés : communication logiciel/logiciel, communication matériel/matériel et communication logiciel/matériel. Le dernier type pose plus de problèmes en raison de l'hétérogénéité des composants utilisés pour les deux parties et les différences de vitesses [87]. O'Nils [88] décrit en détails les différents schémas de communication qui peuvent exister dans un système mixte et plus spécialement dans un système embarqué, ces possibilités sont résumées dans le tableau 2.1.

Schémas de communication
Processus logiciel connecté à un autre processus logiciel sur le même processeur.
Processus logiciel connecté à un autre processus logiciel sur un processeur différent, avec le même ou un autre système d'exploitation.
Processus logiciel connecté à un processus matériel (coprocesseur).
Processus logiciel connecté à un périphérique, par exemple, module de la bibliothèque.
Processus logiciel connecté à une bibliothèque de module logiciel.
Processus matériel connecté à un autre processus matériel sur la même puce (partition).
Processus matériel connecté à un autre processus matériel sur une autre puce (partitions différentes).
Processus matériel connecté à un périphérique.
Processus matériel connecté à un processus logiciel.
Périphérique connecté à un processus logiciel.
Coprocesseur matériel connecté à une bibliothèque de module logiciel.

Tableau 2.1 : Les différents schémas de communication dans un système mixte

2.5.1 La synthèse des communications

La synthèse des communications consiste à réaliser physiquement les canaux permettant les échanges de données entre les tâches [33]. L'implémentation d'une architecture logicielle/matérielle hétérogène qui satisfait les contraintes de la spécification, est une tâche complexe. La difficulté augmente surtout lors de l'élaboration des mécanismes et de la structure des communications. La synthèse des communications s'effectue généralement après l'étape de partitionnement, après avoir assigné les fonctions de la spécification aux composants logiciels et matériels. Le problème revient alors à déterminer pour chaque transfert de données les modalités de la communication en termes de: structure, support, type de transfert et protocole. Cependant, dans certains cas les durées des communications augmentent considérablement le coût de l'implémentation entrainant ainsi une dégradation des performances. Cela oblige les concepteurs à itérer l'étape de partitionnement en intégrant dans la fonction de coût des contraintes qui renseignent sur le coût des communications. Pour

éviter ce problème, certains travaux [18] se sont intéressés au problème de l'évaluation des communications durant l'étape de partitionnement.

2.5.2 Les niveaux d'abstraction de la communication

L'utilisation de niveaux d'abstraction permet le raffinement progressif d'une spécification abstraite vers une réalisation finale du système. Le niveau d'abstraction le plus élevé qui peut être considéré pour un système mixte est le niveau système. À ce niveau, l'interface entre le logiciel et le matériel est complètement abstraite, ce qui permet de représenter la communication entre une tâche logicielle et une tâche matérielle par un canal abstrait de type FIFO ou buffer [85]. Ce modèle peut être considéré comme une spécification fonctionnelle du système hétérogène, car les interfaces logicielles et matérielles peuvent être réalisées par le langage de spécification utilisé. Des langages et des outils comme Simulink [89] et SystemC [48] peuvent être utilisés pour construire de tels modèles. Pour la suite, nous nous basons essentiellement sur les niveaux d'abstraction définis dans les travaux de thèse de Nicolescu [84], qui sont:

2.5.2.1 Le niveau service

Le comportement est décrit par un ensemble de tâches fournissant et utilisant des services. Les communications sont des appels et des réponses d'offres de services [90]. Les interfaces des modules sont composées de ports d'accès à des réseaux abstraits qui garantissent le routage et la synchronisation des connexions établies dynamiquement. Ces ports fournissent des services d'un certain type et les opérations sur les ports sont des requêtes et des services. Les tâches élémentaires sont des processus qui interagissent avec l'environnement via des requêtes et des services. Le temps de communication est non nul et peut ne pas être prévisible. Le temps global est abstrait à un ordre partiel entre les requêtes de services. A ce niveau, les processus peuvent contenir des flots d'exécution (threads) hiérarchiques. Ce niveau d'abstraction est utilisé dans les travaux de [91].

2.5.2.2 Le niveau transaction (TA[2])

Les différents modules du système communiquent via un réseau explicite de canaux de communication qui sont dits actifs. Ces canaux permettent de gérer la synchronisation et peuvent disposer d'un comportement complexe, comme par exemple la conversion des

[2] Transaction Accurate

protocoles spécifiques aux différents modules communicants. Les détails de la communication sont englobés par des primitives de communication de haut niveau et aucune hypothèse sur la réalisation des protocoles de communication n'est faite. Le comportement est décrit par des regroupements de tâches concurrentes inter-communicantes via une topologie explicite de média capable de transporter des données de type générique [90]. L'abstraction du temps est réduite à un ordre partiel des envois de messages. Ce niveau a été présenté dans plusieurs travaux antérieurs, citons [92, 93, 94].

2.5.2.3 Le niveau macro-architecture ou message

C'est le niveau spécifique à la communication par des fils abstraits, englobant des protocoles de niveau pilote des entrées/sorties. La dénomination macro-architecture provient de la vue macroscopique des unités de calcul, de communication et de mémorisation qu'offre une telle architecture. Un modèle de ce niveau implique par conséquent le choix d'un protocole de communication et la topologie des interconnexions. Au niveau macro-architecture, les ports composant les interfaces sont des ports logiques, qui assurent l'interconnexion des différents modules par des fils abstraits. Les opérations de communication peuvent durer un temps non nul mais prévisible. Le comportement des modules de base est décrit par des processus qui réalisent un pas de calcul et des opérations de communication.

2.5.2.4 Le niveau RTL[3] ou microarchitecture

La communication est réalisée par des fils et des bus physiques. La granularité de l'unité de temps est le cycle d'horloge et les primitives de communication sont du type set/reset sur des ports et l'attente d'un nouveau cycle d'horloge. Au niveau RTL ou microarchitecture, les interfaces des modules sont composées de ports physiques qui permettent de connecter les différents modules par des fils physiques. La communication étant concrétisée par des fils physiques, les données sont transmises instantanément en représentation binaire fixe. A ce niveau, la gestion des interruptions et le décodage des adresses sont explicites. Dans ce cas, l'unité temporelle devient le cycle d'horloge et les processus correspondent à des machines d'états finies où chaque transition est réalisée en un cycle d'horloge. Des travaux comme ceux de [95, 96] mettent en œuvre ce niveau d'abstraction.

---

[3] Register Transfer level

Le tableau 2.2 résume les principales caractéristiques des niveaux d'abstraction présentés [84].

Niveau d'abstraction	Média de communication	Primitive de communication typique
Niveau Service	Réseaux abstraits	Demande (Imprimer, dispositif, fichier)
Niveau Transaction	Canaux actifs	Send (fichier, disque)
Niveau Macro-Architecture	Fils Abstraits (canaux abstraits)	Write (donnée, port) Wait until x=y
Niveau Microarchitecture	Fils Physiques	Set (Valeur, port) Wait (clock)

Tableau 2.2 : Niveaux d'abstraction pour la communication [84]

2.5.3 Types de communication dans un système mixte

Pour l'élaboration des interfaces logicielles/matérielles, il faut décider de l'implémentation des communications entre les différents processus. Pour cela, il existe quatre principales manières de communication:

2.5.3.1 Communication par mémoire partagée

La communication est assurée par le fait que le processus expéditeur place les données à une adresse bien déterminée dans une mémoire partagée, à partir de cette adresse et par l'utilisation des instructions d'accès mémoire, le processus récepteur peut lire les données. L'avantage de ce modèle de communication est que le programmeur n'a pas besoin de réfléchir à l'allocation des données dans les mémoires locales des différents processus [97]. Cependant, cette solution suppose que le processeur et les périphériques matériels sont connectés au même bus mémoire. Ceci suppose aussi qu'il n'y a pas de problème d'adaptation entre l'interface de la mémoire et celle des périphériques utilisés, ce qui n'est pas toujours vrai dans le cas des systèmes mono-puces [83]. Un autre inconvénient de cette solution est le coût de la taille mémoire qui est assez important car les techniques standards sont basées sur l'utilisation de sémaphores ou de verrous pour gérer le partage de ressources. Cette technique augmente le coût des communications et implique une dégradation des performances.

2.5.3.2 Communication par passage de message

Les données sont transmises entre processus sous forme de messages. Un message est un ensemble de données associées à des informations permettant de les traiter et de les

identifier [97]. L'avantage de cette technique est qu'elle évite les problèmes d'accès à une mémoire commune aux processus. Elle permet aussi de gérer la synchronisation entre processus grâce à l'utilisation des primitives d'envoi de messages bloquantes.

2.5.3.3 Utilisation de mémoire virtuelle d'entrées/sorties

Cette technique consiste à utiliser un espace mémoire virtuel spécifique aux périphériques matériels d'entrées/sorties. L'adresse envoyée par le processeur contient une partie qui correspond à un périphérique donné. Cette adresse ne correspond à aucune adresse physique dans la mémoire du système [83]. Un comparateur logique est généralement utilisé pour décoder l'adresse envoyée par le processeur et sélectionner le périphérique approprié pour lui transférer les données. L'avantage de cette solution est qu'il n'y a pas un surcoût de mémoire durant le transfert des données entre l'émetteur et le récepteur.

2.5.3.4 Accès direct à une mémoire distante [98]

Les premières architectures à base de mémoire distribuée nécessitaient que le processeur soit interrompu pour chaque requête d'accès à la mémoire. Cette solution ne permet pas une bonne exploitation du processeur. Depuis, les nouvelles architectures contiennent un second processeur, ou coprocesseur, qui est responsable de la gestion des accès à la mémoire. Ce processeur gère le trafic et l'accès au réseau de communication.

2.5.4 Types de conception des interfaces de communication

La synthèse des interfaces de communication peut se faire d'une manière manuelle, automatique ou semi-automatique :

2.5.4.1 Synthèse manuelle des interfaces [97]

La synthèse manuelle des interfaces est une tâche difficile car elle demande une double compétence: celle de la conception numérique et celle du génie logiciel. L'implémentation est généralement répartie sur deux équipes de développement. La difficulté de dialogue pose donc un premier problème. Pour cela, il faut s'assurer de la coordination et de la bonne compréhension entre les deux équipes. Car une erreur de compréhension, qui se voit tard dans le flot de conception peut impliquer des efforts de recherche considérables dans les équipes et peut impliquer des modifications coûteuses dans les deux parties. Pour éviter cela, il faut qu'une coordination soit assurée par des personnes ayant une vue d'ensemble du

système ainsi qu'une bonne connaissance des détails propres aux implémentations logicielles et matérielles.

2.5.4.2 Synthèse automatique des interfaces

La synthèse des interfaces de communication dans les systèmes de codesign doit faire face à plusieurs problèmes. En effet, la spécification des protocoles de communication dans ces systèmes peut être considérée à plusieurs niveaux de granularité. Les primitives de communication décrivent une séquence de transferts depuis le niveau le plus bas jusqu'à la représentation d'un comportement de type processus communicant au moyen de primitives au niveau le plus abstrait [19]. De plus, chaque changement de scénario de communication oblige les concepteurs à redévelopper les interfaces. Pour ces raisons, il est difficile voir impossible d'explorer l'espace de conception complet par une démarche de synthèse manuelle. Il est alors nécessaire d'automatiser la synthèse des interfaces surtout pour les systèmes complexes et critiques. L'automatisation permet de considérer un plus grand nombre de solutions, et de se libérer de la gestion des détails de bas niveaux de l'implémentation. La synthèse des interfaces de communication peut être aussi faite d'une manière semi-automatique. Il s'agit dans ce cas d'une méthode interactive qui consiste à automatiser une partie du processus de synthèse et faire intervenir le concepteur.

2.5.5 Les composants de la communication

Les performances d'échange de l'information dépendent fortement des composants choisis pour l'élaboration de la communication, ces composants peuvent être :

2.5.5.1 Les adaptateurs [90]

Ce sont des blocs souvent matériels, rarement à base de processeur logiciel. Les performances de ces composants dépendent largement du type de l'application et du réseau de communication:

- Pour les applications orientées contrôle, le non-déterminisme des communications rend difficile leur modélisation et contraint les outils d'exploration d'architecture à les négliger.
- Pour les applications orientées calcul, bien que le déterminisme des communications permette de masquer les latences, le débit des communications doit cependant être suffisamment élevé pour suivre les cadences imposées par le cadencement des unités de calcul.

- Les ressources des réseaux de communication sont généralement partagées par un nombre massif de connexions logiques, l'utilisation trop longue de celles-ci peut empêcher une tâche de respecter les échéances qui lui étaient imposées.

2.5.5.2 Les ponts (bridges) [99]

Les ponts sont une application spécifique des adaptateurs de la connexion entre plusieurs bus supportant chacun plusieurs maîtres. Ils sont généralement implémentés en logiques câblées. Leur utilisation est requise lorsqu'il est nécessaire d'adapter deux sous-réseaux de communication. Cela se présente lorsque le réseau est hiérarchisé. En général, les ponts doivent assurer :

- L'attribution du bus: pour l'organisation, l'initiateur de la communication ou maître prend le contrôle de son bus et accède au pont se comportant alors comme un esclave. Le pont formule alors une requête pour obtenir l'accès au bus connecté à l'esclave et agit ainsi en tant que maître sur ce bus.
- La transformation de protocole dans le cas de bus hétérogènes: toutes les données échangées par un maître sur son bus suivent un protocole d'échange qui peut ne pas être supporté par le bus connecté à l'esclave ciblé. Le pont doit alors adapter ces deux protocoles.

2.5.6 Topologies du réseau de communication

Les critères de décision pour le choix d'un réseau sont la bande passante offerte, la latence des communications, le coût physique du réseau, sa consommation et sa flexibilité (possibilités de configuration) [100]. Le problème des réseaux de communication est que l'élévation de la bande passante augmente la latence au détriment des performances de la communication, pour cela le choix du réseau de communication le plus approprié pour un système est une opération délicate. Plusieurs topologies peuvent être employées pour le réseau: bus partagés, connexions point à point, réseau en anneau (token ring) ou réseau sur puce.

2.5.6.1 Les bus partagés

Il s'agit d'implémenter plusieurs liens logiques de communication par un unique médium de communication à accès partagés qui est le bus. Ce médium permet la mise en place de liens point à point ou multipoint entre un maître et un ou plusieurs esclaves [90]. Les inconvénients des bus sont une bande passante limitée, et un nombre limité de composants

pouvant être connectés. De plus, plus la charge du bus est importante, plus la latence moyenne des transactions est élevée. Pour cela, les systèmes utilisent généralement plusieurs bus connectés entre eux par des ponts. Mais le problème de cette technique est que les ponts apportent un temps de latence supplémentaire, ce qui peut dégrader les performances [100].

2.5.6.2 Les connexions point à point [90]

Reposent sur l'utilisation de ressources de transmission spécifiques et dédiées à l'implémentation d'un lien entre deux nœuds de calcul. Principalement utilisées pour l'implémentation des communications entre un processeur et un accélérateur. L'avantage de cette solution est qu'elle peut offrir des débits importants avec des latences faibles ce qui peut être très rentables pour les applications orientées traitement de données. Cependant, ces connexions sont non-extensibles et spécifiques à un protocole.

2.5.7 Les modèles de signalisations pour la synchronisation logicielle/matérielle

Les signalisations permettent d'informer le logiciel des événements générés par le matériel. Il existe deux mécanismes de signalisations principaux: la signalisation par scrutation et la signalisation par interruption matérielle.

2.5.7.1 La scrutation

La scrutation correspond à une attente active du processeur par lectures successives dans un registre ou une mémoire partagée afin de constater l'avènement d'une communication [97]. Les registres contiennent des drapeaux qui indiquent la fin d'une émission ou la réception d'une donnée. L'accès à ces registres peut se faire soit par accès direct à la mémoire physique, soit par accès à la mémoire virtuelle des E/S projetées dans l'espace mémoire du système d'exploitation ou/et dans l'espace utilisateur du processus [83]. L'inconvénient majeur de cette solution est que pendant la scrutation le processeur ne peut pas exécuter d'autres instructions. De plus, les accès aux périphériques de communication, pour l'interaction avec un grand nombre de canaux, requièrent l'intégralité des ressources du processeur. Cependant, il est possible de régler indépendamment pour chaque canal la période de scrutation, mais il faut pondérer cette solution par la latence de réponse du processeur à une communication, le nombre élevé de canaux et les contraintes temps-réel imposées à l'exécution des tâches [90].

2.5.7.2 Les interruptions matérielles

Les interruptions matérielles sont des requêtes émises par les périphériques matériels afin de détourner le processeur de son flot d'exécution courant. Lorsqu'un périphérique veut avertir l'ensemble logiciel d'un changement de son état, il peut émettre une telle requête afin que l'unité matérielle, embarquée au cœur du processeur en charge de leur traitement puisse automatiquement lancer l'exécution d'un logiciel dédié capable de prendre en compte cet événement extérieur [90]. Comparée à la scrutation, cette solution évite la surexploitation des ressources du processeur. Cependant, cette mise en œuvre coûte plus cher à cause du coût apporté par l'exécution du traitement de l'interruption.

2.6 Approches de cosynthèse d'interfaces de communication

2.6.1 Daveau [49]

Cette approche repose sur trois primitives interactives appelées par le concepteur et une bibliothèque de modèles de communication. Elle permet de transformer un système communicant à l'aide de schémas de communication de haut niveau en un ensemble de processeurs interconnectés à travers des bus et des signaux. Elle traite à la fois la sélection de protocole et la génération d'interface et est basée sur l'allocation d'unités de communication. Elle permet une large exploration de l'espace des solutions grâce à une sélection du protocole de communication et une synthèse du réseau d'interconnexion.

2.6.2 Gogniat [19]

Le modèle de communication considéré utilise une topologie à base de bus sans mécanisme de diffusion des données entre plusieurs émetteurs et récepteurs. Les communications peuvent être synchrones ou asynchrones, les protocoles bloquants ou non bloquants et le schéma d'exécution traitements/transferts peut s'effectuer en recouvrement ou de façon exclusive. La méthode de synthèse détermine, à partir d'un graphe partitionné et ordonnancé, les communications et les ressources associées nécessaires à l'exécution de l'application. Le flot de synthèse est décomposé en deux phases : la première permet de caractériser les communications de l'application, et la seconde conduit à l'implémentation de l'architecture finale. Le premier traitement de la phase de caractérisation consiste à estimer les durées des communications. Les modes de transfert sont ensuite évalués afin de définir les schémas d'exécution entre les traitements et les transferts. L'étape suivante représente le cœur

de la méthode de synthèse des communications, car elle influe profondément sur l'architecture finale.

2.6.3 Hommais [33]

La synthèse des communications est réalisée à partir de trois descriptions : la spécification parallèle de l'application, l'architecture cible choisie pour le système et le choix d'implémentation de chaque tâche en matériel ou en logiciel. La spécification parallèle est décrite à l'aide de la bibliothèque FSS (Functional System Specification). Cette bibliothèque réalise les canaux et les primitives de communication. Les primitives de communication bloquantes READ et WRITE sont implémentées en utilisant les moyens de synchronisation standard. Le comportement des tâches est décrit en langage C. Dans la spécification parallèle, les tâches communiquent entre elles par des canaux. Un canal contient une FIFO de profondeur finie.

2.6.4 Coste [80]

Le processus de synthèse choisit dans une bibliothèque le protocole de communication qui est capable d'exécuter les primitives de communication requises par le canal. Le protocole de communication est distribué entre les primitives de communication et le contrôleur. Dans le cas d'interconnexions de modules existants, le contrôleur sera responsable de l'adaptation des protocoles de communication utilisés par les modules interconnectés par le même canal. Lorsqu'il n'y a pas de contrôleur, le protocole est complètement distribué entre les primitives de communication. La complexité du contrôleur peut varier d'une simple file d'attente à un protocole complexe en couches, ou il peut être un module existant réutilisé.

2.6.5 Maalej et al. [101]

L'interface est définie par une description générale. Cette description est faite dans deux fichiers VHDL. Le premier fichier est un package qui décrit les types nécessaires pour l'interface. Il contient un type décrivant la taille du bus de donnés du processeur, un autre décrivant la taille du bus de données de l'accélérateur matériel, et deux autres décrivant les signaux entrées et sorties du bus du processeur. Le deuxième fichier constitue la description fonctionnelle de l'interface. Le fonctionnement de l'interface est assuré par 5 processus groupés dans un seul "process" qui est actionné par le changement du signal d'horloge. Chaque processus est contrôlé par des conditions propres à son fonctionnement.

L'interconnexion se fait par l'intermédiaire du bus interne du processeur. De ce fait, il s'agit plus d'intégrer des composants matériels au cœur du processeur que d'assembler un SoC habituel. Le principe de fonctionnement global ressemble plus à celui d'un ASIP.

2.6.6 Lyonnard [90]

Propose de générer des architectures spécifiques à une application. Il prend en entrée une spécification abstraite annotée de l'application, et produit en sortie des modèles d'implémentation de l'architecture et des adaptateurs de communication en assemblant et en configurant des composants contenus dans une bibliothèque. La bibliothèque est constituée de deux parties : une partie donnant la description des divers éléments, et une partie contenant leur code. La première partie contient de nombreux objets dont les principaux sont les modèles structurels. Ces éléments sont fournis avec des attributs indiquant leur compatibilité respective. La deuxième partie contient le code correspondant aux implémentations sous forme d'entrelacement de portions de code final.

2.6.7 Gharsalli [83]

Pour résoudre la complexité de la conception des interfaces, cette approche se base sur une abstraction de la communication qui permet de cacher les détails des interfaces logicielles/matérielles. Le niveau d'abstraction est élevé pour spécifier un modèle d'interface virtuel qui sépare l'interface de la mémoire du réseau de communication. Cette abstraction facilite l'implémentation générique des interfaces au niveau RTL d'une manière systématique. La génération automatique de ces interfaces raccourcit le temps de conception. Le découplage entre l'interface mémoire et celle du média de communication permet la séparation des responsabilités de conception. Le concepteur des mémoires et celui des réseaux de communication peuvent donc travailler séparément et en même temps.

2.6.8 Paviot [97]

Un modèle de représentation des services de communication permet au concepteur de décrire un service de communication à base d'unités fonctionnelles et d'unités de stockage de données. Le système de spécification permet de décrire des systèmes hétérogènes en utilisant des modules, des ports et le concept de modules virtuels. Ces spécifications sont traitées par deux outils permettant la génération des parties logicielles et matérielles des interfaces liant les processeurs au réseau de communication. Au lieu de spécifier un service de

communication et son implémentation par deux paramètres, les unités fonctionnelles et les unités de stockage de données composant le service de communication sont spécifiées. Cette solution est plus adaptée à l'exploration du partitionnement des services de communication.

2.6.9 Grasset [100]

La méthodologie de conception des interfaces de communication part d'une description des services de communication requis par l'application et des services offerts par le réseau. Le concepteur doit alors modéliser le protocole ou le choisir parmi une liste de protocoles déjà modélisés. La méthode prend en entrée une spécification obtenue à partir du découpage logiciel/matériel et permet d'obtenir un modèle RTL synthétisable. L'approche s'appuie sur un assemblage systématique de composants de base issus d'une bibliothèque.

2.6.10 Youssef [98]

Cette approche est basée sur l'utilisation d'un modèle de programmation parallèle de haut niveau d'abstraction et d'un modèle de l'architecture matérielle. Le modèle de programmation parallèle permet de commencer le développement du logiciel applicatif sans attendre que la partie matérielle ne soit prête. Il permet aussi d'avoir des estimations des performances du système final. Le modèle de l'architecture matérielle permet de commencer la conception de l'interface logicielle/matérielle avant même la fin de la conception de la partie matérielle. Toutefois, cette approche ne considère que les parties logicielles de l'interface et suppose que le matériel est figé.

2.6.11 Jerraya [35]

Dans COSMOS, la synthèse de la communication commence avec deux descriptions SOLAR (format intermédiaire pour les concepts du niveau système): un graphe du système et une bibliothèque de modèles de communication. Cette tâche est effectuée manuellement. Le système sélectionne une unité de canal de la bibliothèque pour l'exécution de chaque primitive de communication de la description. L'unité de canal choisie doit être en mesure d'exécuter les primitives utilisées par les processeurs communicants correspondants. Cette étape fixe le nombre et le type des protocoles à utiliser pour chaque unité de communication. La tâche de correspondance du canal remplace toutes les unités du canal en distribuant l'ensemble des informations contenues dans les unités de conception et les contrôleurs spécifiques de la

communication. Ces contrôleurs sont sélectionnés à partir d'une bibliothèque d'implémentation de canaux.

2.6.12 Chavet [102]

Propose une méthodologie de conception permettant de générer automatiquement un adaptateur de communication nommé Space-Time AdapteR (STAR). Le flot de conception prend en entrée des diagrammes temporels (fichier de contraintes) ou une description en langage C de la règle de brassage des données et des contraintes utilisateur (débit, latence, parallélisme…). Ce flot formalise ensuite ces contraintes de communication sous forme d'un graphe de compatibilité des ressources multi-modes (MMRCG) qui permet une exploration de l'espace des solutions architecturales afin de générer un composant STAR en VHDL de niveau transfert de registre (RTL) utilisé pour la synthèse logique. L'architecture STAR se compose d'un chemin de données (utilisant des FIFOs, des LIFOs et/ou des registres) et de machines d'états finies permettant de contrôler le système. L'adaptation spatiale (une donnée qui peut être transmise de n'importe quel port d'entrée vers un ou plusieurs ports de sortie) est effectuée par un réseau d'interconnexion adapté et optimisé. L'adaptation temporelle est réalisée par les éléments de mémorisation, en exploitant leur sémantique de fonctionnement (FIFO, LIFO).

2.6.13 Khan [103]

L'approche de cosynthèse produit des architectures multiprocesseurs composées d'éléments de traitement hétérogènes reliés par une structure de communication point à point. Le processus de cosynthèse se compose de quatre phases distinctes: la sélection des éléments de traitement pour l'ajout au système, la répartition des tâches en pipeline, l'ordonnancement et une interconnexion de topologie régulière. Initialement, une topologie irrégulière est générée et mappée à l'architecture régulière. Cette méthodologie effectue le partitionnement du système et produit une topologie irrégulière multiprocesseur du système. Le processus de cosynthèse commence par un monoprocesseur sur lequel est basée toute la solution logicielle de l'application. Les coûts généraux de l'ordonnancement des tâches sont ignorés.

2.6.14 Hao et Xie [104]

Proposent une approche pour la conception de l'interface matérielle/logicielle basée sur le concept d'élément du pont. Les interfaces sont considérées comme des composants du

pont. Les composants du pont permettent d'élever le niveau d'abstraction pour la conception des interfaces et de propager des événements à travers la frontière matérielle/logicielle. Une fois le système conçu, le matériel, le logiciel et les composants du pont seront traités comme des composants sur le même niveau. Tout nouveau système basé sur la même plate-forme peut réutiliser les composants du pont existant. Les composants du pont sont configurables selon l'abstraction de la plateforme.

2.6.15 Faes et al. [73]

L'interface entre le matériel et le logiciel est transparente pour le concepteur du logiciel, et est basée sur une méthode d'interception dynamique. Cette approche offre un protocole général qui assure la préservation de la sémantique de l'application. Ce protocole fournit une interface bidirectionnelle pour les appels de méthode à distance. Les objets sont passés par référence et les types de données primitifs sont passés par valeur, comme c'est le cas dans une pure implémentation logicielle.

2.6.16 Autres approches de cosynthèse

Narayan et Gajski [105] abordent le problème de la génération d'interface entre deux modules matériels d'une spécification partitionnée. L'approche nécessite un schéma d'arbitrage pour gérer les conflits des bus, et par conséquent un temps d'analyse supplémentaire. Ecker [106] présente une méthode de transformation et d'optimisation de protocoles. Dans [107] la communication est réalisée avec une mémoire partagée et le problème de l'interfaçage entre une mémoire et un coprocesseur ou un processeur d'entrée/sortie est abordé. Madsen [108] considère le problème de l'adaptation d'interface entre une interface fixe et un medium de communication choisi durant le partitionnement.

Kalavade [18] utilise une architecture avec un seul processeur pour exécuter la partie logicielle et plusieurs coprocesseurs pour implémenter la partie matérielle. Ces modules sont connectés à un bus unique. Un contrôleur central câblé gère les communications entre le processeur et les modules matériels. Le transfert d'une donnée entre le logiciel et le matériel est un simple accès mémoire. Srivastava [109] commence après le partitionnement avec un graphe de processus, un modèle d'architecture cible et transpose les communications sur les ressources de communication physiques disponibles.

Dans l'approche Vulcan [110], le système est décrit à l'aide du langage HardwareC comme plusieurs processus communicants. Les communications intra-processus sont gérées

par une mémoire partagée et les communications interprocessus par passage de messages. Cette approche impose d'avoir un seul processeur, le seul maître sur le bus, ce qui limite grandement les débits. Daveau et al. [111] proposent une approche qui décrit le système comme des processus communicants à travers des canaux abstraits. Pour cela, les processus utilisent des primitives prédéfinies. Dans cette approche, l'exploration spatiale est effectuée pour optimiser l'utilisation des ressources et pour minimiser le coût de la communication en résolvant le problème de la synthèse comme un problème d'allocation.

Passerone et al. [112] décrivent un algorithme pour la synthèse du code Verilog sous forme d'une machine à états finis qui assure les transferts des données cohérentes entre deux blocs IP avec différents protocoles de signalisation. La communication des processus en interaction est exprimée sous forme d'expressions régulières. L'approche optimise l'interface, mais les hypothèses considérées mettent trop de restrictions sur la communication. Vercauteren et al. [113] décrivent une approche de génération d'interfaces logicielles / matérielles basée sur une bibliothèque paramétrée. Pour réaliser la communication entre le matériel et le logiciel au niveau matériel, un processus logiciel est remplacé par un nouveau processus équivalent spécifié dans un langage de description de matériel (HDL). Cette approche est utilisée dans l'environnement CoWare [32].

IPCHINOOK [114] est un outil de synthèse pour les systèmes embarqués distribués. Il est orienté vers la réutilisation des composants. L'approche proposée permet de synthétiser les communications d'un système en utilisant une bibliothèque décrivant les protocoles de communication. Cependant, la description du système est très singulière et fort complexe. [115] gère le problème de l'hétérogénéité logiciel/matériel et permet une accélération matérielle du pilote, mais la modélisation reste manuelle. L'approche ROSES [116] repose sur une bibliothèque de processeurs et de modules de communication paramétrables. L'intégration automatique est faisable grâce à l'utilisation d'une bibliothèque de protocoles pour rassembler les simulateurs de matériel ainsi qu'une bibliothèque de logiciel contenant des blocs de construction pour un système d'exploitation. L'inconvénient de cette approche est la création manuelle et fastidieuse des bibliothèques.

2.7 Outils de cosynthèse d'interfaces de communication

2.7.1 POLARIS [117]

POLARIS est un outil de composition de matériel. Les protocoles de communication des composants du système sont gérés par des FSM spécifiques. Un FSM est synthétisé pour la correspondance entre le protocole de communication individuel et le protocole standard afin de former une partie d'une architecture d'interfaces générale (Figure 2.2). Cette méthode permet la conception et la génération d'une interface synchrone entre deux composants matériels pouvant fonctionner à des fréquences différentes. Toutefois, le mécanisme d'adressage de la communication et le mécanisme d'arbitrage ne sont pas traités automatiquement.

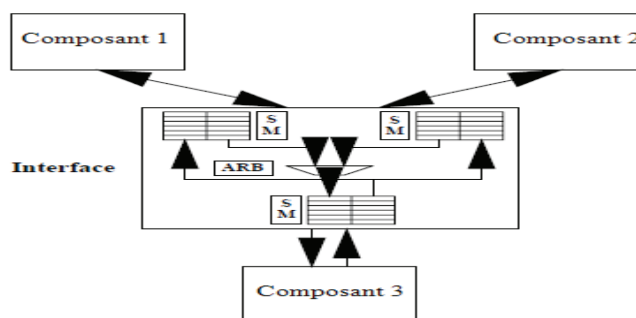


Figure 2.2 : Architecture d'interface entre trois composants [117]

2.7.2 DTSE [118]

La méthodologie DTSE (Data Transfert and Storage Exploration) repose sur la notion de position des données (Figure 2.3). Puisque les données sont souvent stockées dans des tableaux, beaucoup de calculs ne concernent qu'une petite partie de ces tableaux. Les caches des processeurs traditionnels ne sont pas en mesure d'exploiter pleinement ce potentiel. L'analyse du code programme à l'avance et la réécriture de ses parties, donnent une occasion pour réduire la consommation d'énergie. De plus, cette action peut potentiellement augmenter les performances. Cette méthodologie apporte de bons résultats puisqu'elle permet l'optimisation conjointe de l'architecture mémoire et du code de l'application. Cependant, le temps nécessaire à l'obtention d'une solution optimale est relativement important car de nombreuses étapes ne peuvent être automatisées et requièrent l'expertise des concepteurs.

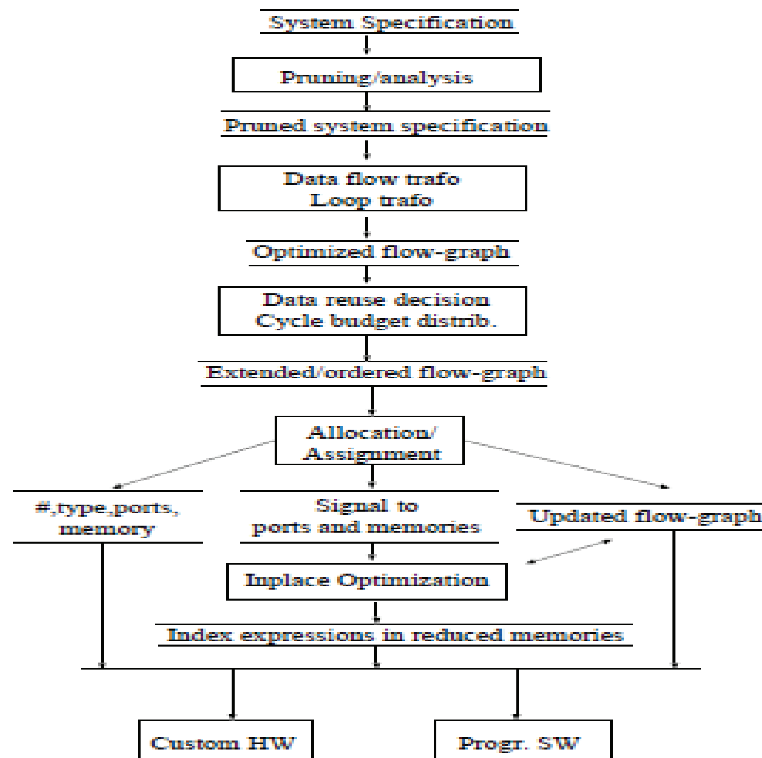


Figure 2.3 : La méthodologie DTSE [118]

2.7.3 PICO [119]

L'outil PICO (Program In, Chip Out) automatise la conception des systèmes optimisés spécifiques à l'application (Figure 2.4). Il utilise une application écrite en langage C pour un ensemble de conceptions de systèmes de haute qualité minimisant le coût des échanges pour plus de performances. Une conception du système PICO contient un EPIC / VLIW (explicitement calcul parallèle d'instruction / instruction très longue) du processeur et un accélérateur optionnel non programmable (NPA). Le système est composé d'un ou plusieurs NPA, tous reliés à un sous-système de cache à deux niveaux qui, à son tour, se connecte au bus système. Chaque NPA est personnalisé pour exécuter un calcul intensif de boucles. PICO génère les combinaisons les plus rentables des sous-systèmes pour fournir plusieurs conceptions systèmes de haute qualité à différents points selon le compromis coût/performance. Il émet du Verilog / VHDL structurel pour les composants matériels et modifie le code de l'application pour inclure les interfaces logicielles pour le matériel généré. Toutefois, l'architecture générée après la synthèse est décrite au niveau RTL.

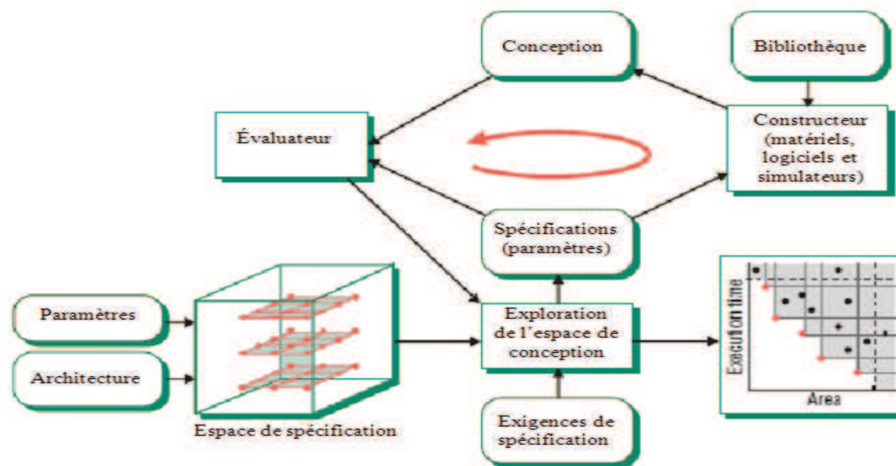


Figure 2.4 : Fonctionnement de l'outil PICO

2.7.4 SPARK [120]

SPARK est un outil de synthèse de haut niveau. Il prend un code ANSI-C de la description comportementale comme entrée et en utilisant le code abstrait et les transformations de boucles, génère une machine à états finis pour la conception graphique, pour produire en sortie un code VHDL synthétisable (Figure 2.5). Les contraintes acceptées par l'outil SPARK sont des contraintes matérielles (nombre et type des opérateurs disponibles). En fonction de ces paramètres, l'outil de synthèse va générer une architecture en optimisant sa latence. Pour optimiser le code source de la description, SPARK emploie des transformations de boucles diverses telles que le déroulage de boucles, l'élimination des calculs communs, les fusions de boucles et la boucle de décalage. Toutefois, SPARK ne produit pas la description du circuit, ce qui contraint les concepteurs à la faire manuellement.

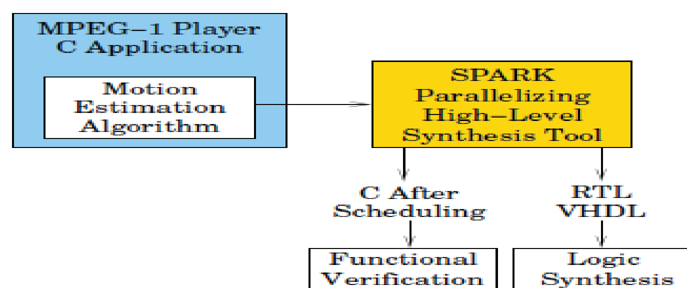


Figure 2.5 : Le flot de synthèse SPARK [120]

2.7.5 Streamroller [121]

L'outil prend en entrée l'application écrite en langage C exprimée comme un ensemble de noyaux de communication (Figure 2.6). Les performances et les contraintes de conception

telles que le débit global et la bande passante mémoire sont également précisées. L'interface du système effectue une analyse de dépendance des données sur l'application pour extraire le graphe de boucles, qui est une représentation de la structure des communications entre les noyaux. Un compilateur synthétise automatiquement les accélérateurs de boucles à des niveaux de performance différents. Le système synthétise un accélérateur pipeline avec un coût minimal pour répondre aux contraintes de performance. Le pipeline se compose d'un certain nombre d'accélérateurs de boucles pour l'exécution des noyaux dans l'application et des tampons de la mémoire pour communiquer les valeurs. Cependant, l'outil Streamroller est limité par le fait que l'architecture mémoire générée lors de la synthèse du chemin de données des accélérateurs, n'exploite que des registres ou des fichiers de registres à sémantique FIFO.

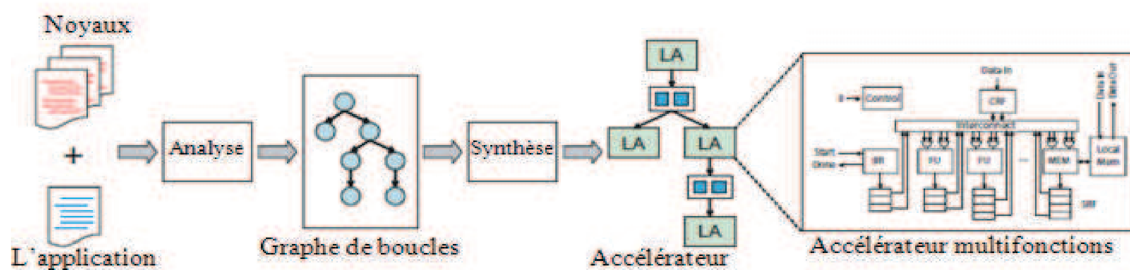


Figure 2.6 : Le flot de synthèse Streamroller

2.7.6 xPilot [122]

xPilot accepte comme entrée le langage C ou SystemC. La description comportementale est d'abord analysée et optimisée par un compilateur. Une synthèse de niveau système des données du modèle (SSDM) est alors construite à partir des représentations internes (Figure 2.7). Les blocs de base dans SSDM sont les processus et les canaux. Un processus décrit le comportement d'un module, et chaque processus utilise un graphe de flot de contrôle de données (CDFG) pour capturer son comportement. Un processus interagit avec les autres processus via des ports et des canaux. Chaque canal implémente une interface pour mettre en œuvre certains protocoles de communication. En général, une SSDM définit un réseau de processus pour modéliser le comportement concurrent d'un système complexe. En sortie, xPilot génère une implémentation RTL avec des fichiers de contraintes (par exemple, les contraintes de localisation physique). La faiblesse de l'outil xPilot est qu'il utilise un système de communication à base de FIFO ce qui limite l'exploration architecturale.

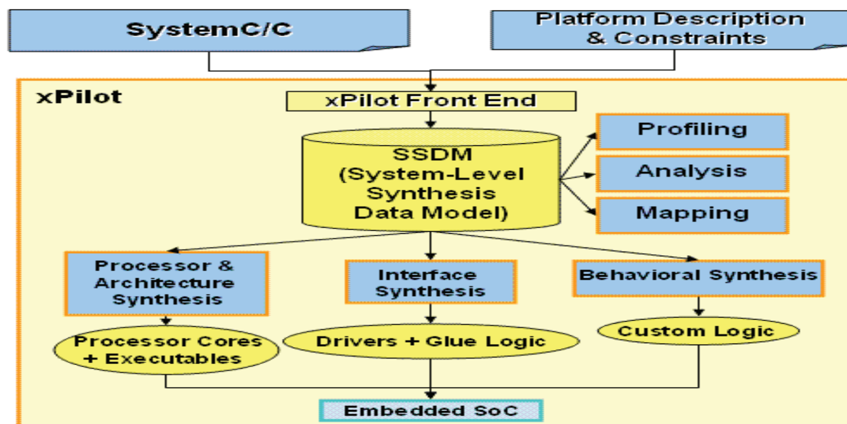


Figure 2.7 : Le flot de synthèse xPilot [122]

2.7.7 Catapult C [123]

Catapult C est un outil de synthèse de haut niveau pour les concepteurs de matériel ASIC et FPGA. Il accepte les langages C + + et SystemC et produit une description de niveau RTL pour les systèmes hiérarchiques complexes composés d'unités algorithmiques de contrôle logique. Le concepteur peut contrôler l'implémentation du matériel synthétisé par l'application de contraintes de haut niveau de gestion des aspects de conception telles que les protocoles d'interface, l'architecture mémoire, le débit, la latence et les transformations de faible puissance. Les concepteurs peuvent spécifier à l'outil avant la synthèse le type de mémoire à mettre en œuvre (Figure 2.8). Les différentes possibilités sont : file de registre multiplexée, mémoire mono-port ou multiports. L'outil utilise des méthodes permettant d'interfacer un grand nombre de ports d'entrées / sorties : streaming, mémoire simple ou double port, FIFO, bus ou tout autre composant de communication généré par le concepteur.

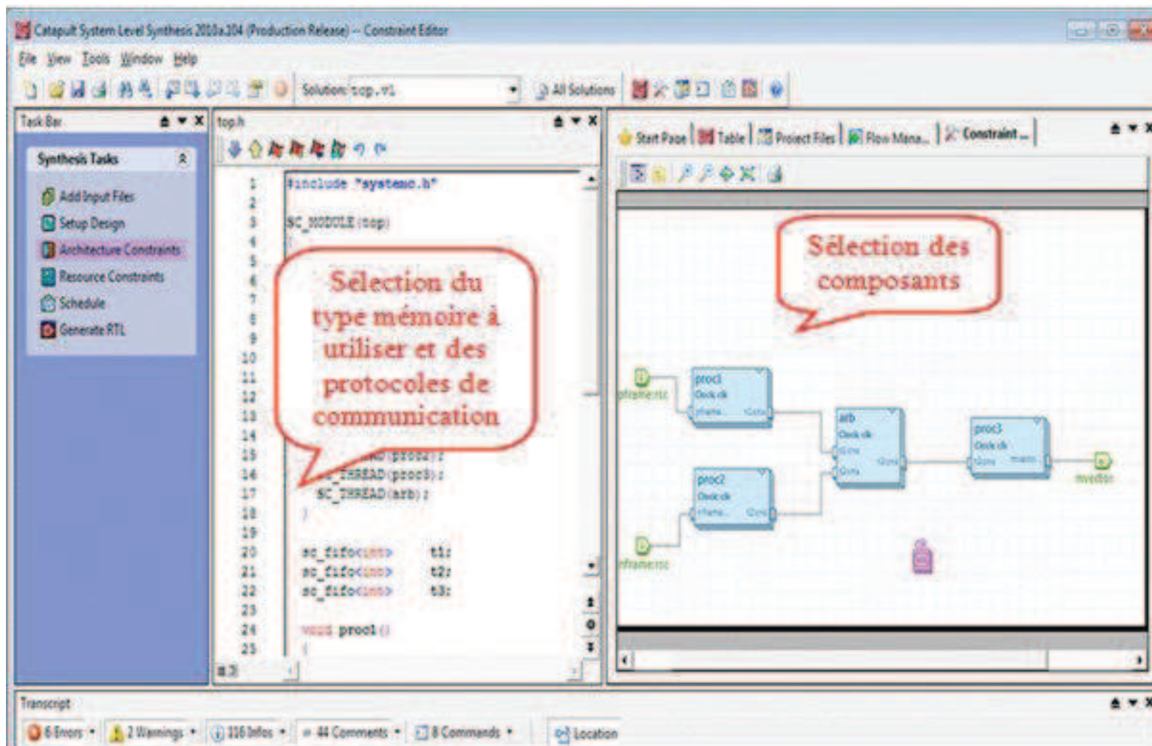


Figure 2.8 : Interface de l'outil Catapult C

2.7.8 SynDEx [124]

SynDEx est un outil de synthèse de niveau système basé sur la méthodologie AAA (Algorithm Architecture Adequation) pour le prototypage rapide et l'optimisation de l'implémentation d'applications embarquées temps réel. Il prend en entrée une spécification de l'algorithme de l'application sous forme d'un graphe factorisé et conditionné de dépendances de données ainsi qu'une spécification de l'architecture cible sous forme d'un graphe d'opérateurs et de média de communications. Il exécute des heuristiques d'optimisation, réalisant ainsi l'adéquation entre l'algorithme et l'architecture. La distribution et l'ordonnancement optimisé sont effectués de manière statique. Après la spécification de l'algorithme et la spécification de l'architecture (Figure 2.9), SynDEx mène à une implémentation sur une machine multiprocesseur. Un exécutif temps réel est généré automatiquement, après la visualisation des performances, permettant l'exécution de l'algorithme sur l'architecture.

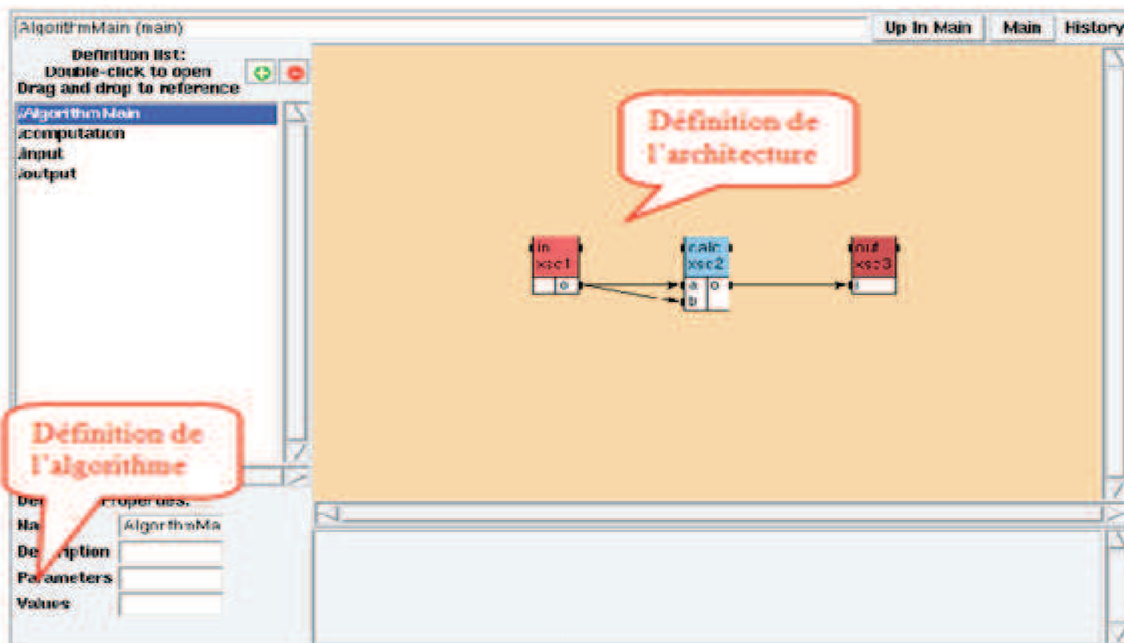


Figure 2.9 : Interface de l'outil SynDEx

2.7.9 GAUT [125]

GAUT est un outil de synthèse académique de haut niveau dédié à des applications numériques de traitement du signal (Figure 2.10). A partir d'une fonction C, GAUT extrait le parallélisme potentiel avant la sélection, l'allocation des opérateurs et la planification des opérations. Il repose sur des contraintes de conception obligatoires qui sont: le débit (l'intervalle d'initiation), le cycle d'horloge et la technologie cible et des contraintes de conception optionnelles qui sont : le chronogramme des entées/sorties et l'organisation de la mémoire. GAUT synthétise une architecture pipeline composée d'une unité de traitement, une unité de mémoire et une unité de communication et de multiplexage. Il permet le partage temporel et spatial des différentes ressources de calcul entre les opérations à effectuer, lorsque les contraintes temporelles le permettent.

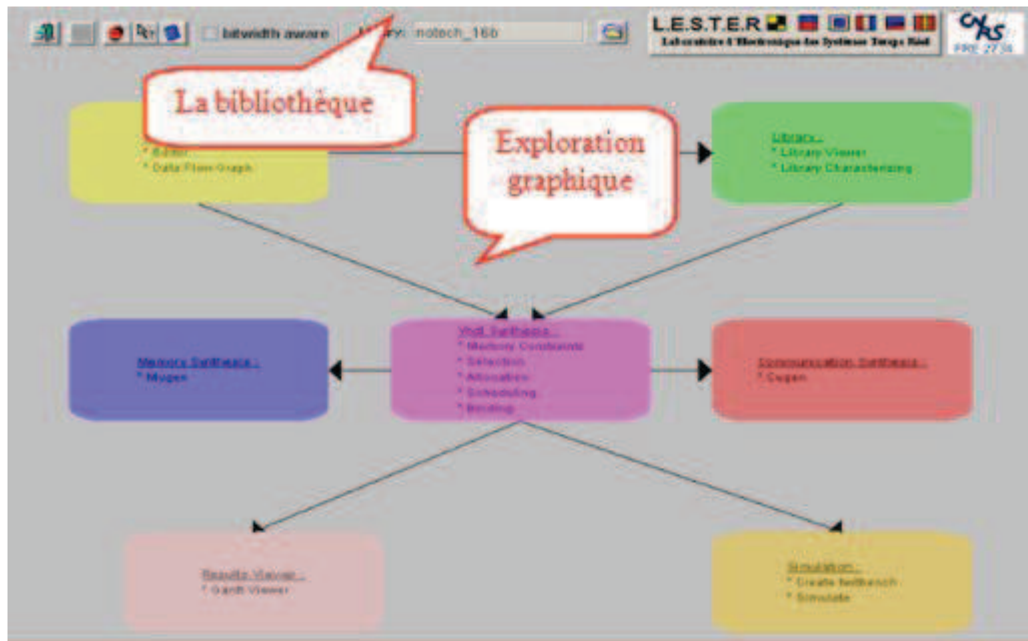


Figure 2.10 : Interface de l'outil GAUT

2.8 Bilan

2.8.1 Caractéristiques des approches et des outils de cosynthèse

Nous avons constaté que l'évolution des travaux dans le contexte de la synthèse des interfaces de communication dans les systèmes de codesign est marquée par deux périodes : la première période représente le codesign traditionnel ou le début du codesign matériel/logiciel, elle s'étend sur les années 90. Cette période est caractérisée par l'intégration d'un premier processeur à une puce. Au début de cette période, le logiciel était écrit à un bas niveau d'abstraction à travers le langage assembleur du processeur. Cela, ne demandait que la compilation du logiciel et son exécution sur le système embarqué, sans passer par une adaptation logicielle. A la fin de cette période, la complexité des systèmes embarqués a forcé les concepteurs à élever le niveau d'abstraction du logiciel et à utiliser les langages de programmation de haut niveau. D'où la nécessité de disposer d'interfaces pour la correspondance entre le matériel et le logiciel. La deuxième période est en cours actuellement. Elle est caractérisée par une complexité de plus en plus importante des systèmes embarqués mesurée en millions de transistors intégrables sur une même puce. Ces systèmes nécessitent des réseaux de communication complexes et des interfaces logicielles/matérielles fiables.

Beaucoup de facteurs et de caractéristiques différencient les approches de cosynthèse, car les domaines d'application et la nature de l'activité de conception varient

considérablement d'une approche à une autre. Dans ce qui suit, nous allons présenter quelques points qui différencient les approches et les outils de cosynthèse :

- Utilisation d'une bibliothèque de protocoles et/ou de composants où les exigences d'interface sont implémentées par des primitives de services appropriées et ces primitives sont sélectionnées à partir d'une bibliothèque. Parmi les approches basées sur les bibliothèques : [33, 35, 49, 80, 90, 100, 113, 116, 119, 120, 103, 123].
- La communication entre les différents composants dans un système mixte peut être réalisée à travers plusieurs stratégies selon les propriétés du système qui sont : la continuité, la nature bloquante ou non bloquante, le partage des canaux, la connexion point à point ou multidiffusion. [105] applique une connexion multidiffusion avec un canal partagé. [112] considère une connexion point à point, discontinue, bloquante et avec des canaux non partagés. [117] considère une connexion point à point, discontinue et qui repose sur un buffer et des canaux non partagés. [103] utilise une structure de communication qui repose sur les connexions point à point.
- Le fait d'avoir recours à une génération personnalisée pour synthétiser les interfaces directement de la description des composants interconnectés. Cette pratique est utilisée dans les travaux de [105, 112, 114, 117].
- Implémentation sur une architecture cible monoprocesseur [31, 110] ou sur une architecture distribuée multiprocesseur [32, 124].
- Le niveau de flexibilité des interfaces varie largement. En effet, certaines approches [105, 112, 117] traitent des interfaces rigides spécifiées par la description des fonctionnalités des composants, alors que d'autres [111] considèrent des interfaces plus flexibles où la mise en œuvre physique peut être choisie de manière appropriée.
- Certaines approches implémentent la communication entre les différents processus à travers une mémoire partagée [33, 107], d'autres approches utilisent le passage de messages [80]. [33, 80, 121, 122, 123] utilisent des FIFO, [102] utilise des LIFO avec des registres et [123] utilise une file de registre multiplexée avec une mémoire mono-port ou multiports.
- Le niveau d'abstraction de la communication est : le niveau message pour [111], le niveau transaction pour [105, 112, 117] et le niveau RTL pour [102, 119, 123].
- Certaines approches [18, 107] effectuent la synthèse des communications pendant l'étape de partitionnement en prenant en considération des paramètres tels que le délai

des communications, alors que la majorité des autres approches réalisent la synthèse après l'étape de partitionnement.

- Plusieurs approches utilisent des primitives de communication : [49] utilise des primitives interactives, [33] utilise des primitives de communication bloquantes READ et WRITE, [111] utilise des canaux abstraits alors que [104] met en pratique le concept d'élément du pont.
- Utilisation d'un système d'exploitation [33, 116].
- Les différentes possibilités d'entrées pour les approches et les outils de cosynthèse sont : une description des composants [117], des expressions régulières [112], les primitives du canal et une bibliothèque de communication [111], une description en langage C [33, 119, 120, 121, 122, 125], une description VHDL [101], une spécification abstraite [90], des diagrammes temporels [102], une description en langage SystemC [122, 123], C + + [123] et HardwareC [110].
- Les sorties sont : une synthèse d'un FSM [112, 117], bus avec sélection [105], sélection de protocole et implémentation [111], une génération en VHDL de niveau transfert de registre (RTL) [102], Verilog / VHDL [119], VHDL [120] et une implémentation RTL avec des fichiers de contraintes [122].

Afin de bien illustrer les différences entre les approches et les outils de cosynthèse les plus importants, nous avons élaboré le tableau récapitulatif qui suit :

	UB	Entrées	Sorties	GN	Niveau d'abstraction	Caractéristiques de la communication	Type de la spécification	Le modèle
POLARIS [117]	Non	Fichier de données formaté	Synthèse d'un FSM	Oui	Transaction	Connexion point à point, discontinue, buffer, canaux non partagés	FSM	Formel
PICO [119]	Oui	Langage C	Verilog/ VHDL	Non	RTL	EPIC / VLIW, NPA	Langage de programmation	Fonctionnel
SPARK [120]	Oui	Langage C	VHDL	Non	RTL	Description manuelle du circuit	Langage de programmation	Fonctionnel
Streamroller [121]	Non	Langage C	Fichier de registres	Non	RTL	Utilisation de FIFO	Langage de programmation	Fonctionnel
xPilot [122]	Non	Langage C/ SystemC	Fichiers de contraintes	Non	RTL	Utilisation de FIFO	CDFG	Formel
Catapult C [123]	Oui	SystemC/ C + +	Description RTL	Non	RTL	Utilisation de FIFO, file de registre multiplexée,	Langage de programmation	Fonctionnel

						mémoire mono-port /multiports		
GAUT [125]	Oui	Langage C	VHDL	Non	RTL	Utilisation de chronogramme des entées/sorties	Langage de programmation	Fonctionnel
Chavet [102]	Non	Langage C	VHDL	Non	RTL	Utilisation de FIFO, LIFO	Langage de programmation	Fonctionnel
Lyonnard [90]	Oui	Fichier de données formaté	VHDL	Non	Message	Utilisation de FIFO synchrone/ asynchrone	Fichier de données formaté	Pas de modèle
Hommais [33]	Oui	Langage C	Graphe de tâches	Non	RTL	Mémoire partagée, utilisation de FIFO, primitives de communication bloquantes, utilisation d'un système d'exploitation	Langage de programmation	Fonctionnel
Coste [80]	Oui	Fichier de configuration	VHDL	Non	Service/ message/ RTL	Passage de message, utilisation de FIFO	Fichier de configuration	Pas de modèle
Passerone [112]	Non	Expressions régulières	Synthèse d'un FSM	Oui	Transaction	Connexion point à point, discontinue, bloquante, canaux non partagés	FSM	Formel
Gharsalli [83]	Non	Fichier de données formaté	C/ VHDL	Non	Service	Utilisation de pilotes d'accès mémoire	Fichier de données formaté	Pas de modèle
Grasset [100]	Oui	Fichier de données formaté	Modèle RTL	Non	RTL	Utilisation d'un modèle de protocoles	Fichier de données formaté	Pas de modèle
Daveau [49]	Oui	Spécification SDL	C/ VHDL	Non	RTL	Utilisation de FIFO, primitives interactives	SDL	Fonctionnel

UB : Utilisation de bibliothèque, GP : Génération personnalisée

Tableau 2.3 : Tableau récapitulatif des approches et des outils de cosynthèse

Malgré l'existence de nombreux outils de synthèse au niveau RTL, il n'existe pas encore des outils de synthèse de haut niveau réussis avec un taux de fiabilité élevé. Cela est dû au fait que les différents outils existants imposent généralement des restrictions qui peuvent décourager les développeurs à les utiliser. De plus, certains outils utilisent des langages non standards que les développeurs n'ont pas l'habitude d'utiliser.

2.8.2 Problèmes de la synthèse des interfaces de communication

La conception des interfaces de communication entre les composants matériels et les composants logiciels est une tâche difficile, et surtout source d'erreurs. Cela est dû à plusieurs problèmes:

- Le problème d'interprétation des schémas de communication de haut niveau lors de la synthèse à partir de spécifications systèmes.
- La conception des interfaces de communication nécessite la bonne connaissance de plusieurs éléments: protocoles de communication, topologies d'interconnexion, supports de communication, types de transferts, modes de transferts, composants, processeurs et pilotes de communication.
- Souvent, le choix de la topologie d'interconnexion et le protocole de communication est basé sur des débits de communication moyens, négligeant ainsi les délais de communication, ce qui peut produire des solutions infaisables.
- Pour pouvoir satisfaire les contraintes de la spécification, les concepteurs sont obligés d'explorer l'ensemble de l'espace des solutions possibles, ce qui n'est pas toujours évident.
- Plusieurs approches de cosynthèse reposent sur l'utilisation de bibliothèques de composants. Ces approches supposent que le comportement de ces composants est déjà validé à l'extérieur de leurs flots. Mais, la construction d'un système à base d'assemblage de composants aboutit généralement à un échec [83]. De plus, l'intégration d'un nouvel élément externe à la bibliothèque pose des problèmes pour l'adaptation de cet élément avec le reste du système.
- Les différentes approches reposent sur le fait que les solutions aux sous-problèmes de la synthèse des interfaces sont spécifiques à l'environnement de l'application et les hypothèses sous-jacentes.
- La majorité des approches existantes souffrent du problème de la validation des interfaces de communication. En effet, les approches de validation valident le système dans sa globalité et négligent la validation des interfaces de communication entre chaque paire de composants du système. Entrainant ainsi, des problèmes de synchronisation entre les interfaces des composants.
- La plupart des approches de cosynthèse ne prennent pas en charge la réutilisation des interfaces de communication. En conséquence, la modification d’un composant du système force les concepteurs à concevoir à nouveau une interface de communication.

- Le temps de communication pose de plus en plus de problèmes pour la conception des systèmes mixtes.

Nous avons constaté que le problème majeur de la synthèse des interfaces de communication est le niveau d'abstraction considéré. En effet, beaucoup de travaux utilisent la synthèse des interfaces à un niveau trop bas, ce qui rend difficile sa réalisation et parfois, la communication entre composants devient un véritable goulot d'étranglement. Le défi de la synthèse des interfaces de communication est donc d'élever le niveau d'abstraction des flots de conception pour maîtriser la complexité des systèmes mixtes. Dans cette optique, et pour pouvoir gérer la synthèse des interfaces de communication à un niveau d'abstraction élevé, nous proposons d'appliquer une solution de l'architecture logicielle, à travers l'approche intégrée IASA à la communication entre un composant matériel et un composant logiciel. Les caractéristiques de l'approche proposée sont résumées dans le tableau 2.4.

	Utilisation de bibliothèque	Entrées	Sorties	Génération personnalisée	Niveau d'abstraction	Type de la spécification	Le modèle
IASA [126]	Oui	Le langage x3ADL	JAVA/ VHDL	Non	Architecture	ADL	Composant

Tableau 2.4 : Caractéristiques de l'approche proposée pour la cosynthèse

2.9 Conclusion

Nous avons passé en revue, dans ce chapitre, les notions de base relatives à la synthèse des interfaces de communication. Des outils et des approches issus du monde de la recherche et de l'industrie ont été présentés.

Bien qu'il y ait eu des progrès importants vers la synthèse du matériel et la synthèse du logiciel de manière indépendante, la synthèse d'interfaces de communication reste une question gênante pour les grands systèmes en raison de sa complexité. La conception des interfaces de communication est une tâche difficile à mettre en œuvre, et surtout source d'erreurs. En effet, les interfaces ont des structures complexes, leur conception nécessite des compétences issues des domaines du logiciel et du matériel. De plus, le processus de génération d'interface doit être capable d'explorer l'espace de conception en un temps minimal de telle manière à ce que l'interface qui en résulte réponde aux exigences de la communication des processus tout en minimisant les coûts de réalisation.

Dans le but de surmonter les problèmes de la cosynthèse des interfaces de communication dans le codesign, nous proposons dans ce travail de ramener le problème de la communication dans un système de codesign à un niveau d'abstraction très élevé qui est le niveau architecture.

Le chapitre qui suit sera consacré à l'architecture logicielle. Les concepts de base de l'architecture logicielle seront présentés, et plus particulièrement le modèle de composant de l'approche IASA qui constitue la base pour notre contribution.

CHAPITRE 3
L'ARCHITECTURE LOGICIELLE ET L'APPROCHE IASA

3.1 Introduction

Face à la croissance extraordinaire de la taille et de la complexité des systèmes informatiques, l'architecture logicielle s'impose aujourd'hui comme une solution prometteuse aussi bien pour la conception que pour la maintenance de ces systèmes. L'engouement pour l'architecture logicielle est d'autant plus motivé par l'évolution des technologies web et java qui ont contribué au développement des logiciels à base de composants (Component-Based Software Engineering). Dans ce chapitre nous allons présenter les éléments fondamentaux de l'architecture logicielle. Nous commencerons par exposer ses avantages et sa description à travers les langages de description d'architectures. Ensuite, nous mettrons l'accent sur l'approche intégrée d'architecture logicielle (IASA) qui représente notre contribution pour résoudre le problème de la synthèse des interfaces de communication dans le codesign.

3.2 L'architecture logicielle

La programmation orientée objet par ses concepts fondamentaux (l'encapsulation de comportements et de données, la notion d'instance, l'héritage et le polymorphisme) a apporté une nouvelle vision à la programmation et est devenue rapidement très populaire grâce au gain considérable en termes d'organisation et de réutilisation. Cependant, elle n'a jamais réussi à apporter une solution bien établie à la question de comment assembler le tout à partir des briques de base [127]. C'est là qu'intervient l'architecture logicielle pour résoudre ce problème.

Il existe de multiples tentatives de définition des architectures logicielles dans la littérature. Cette multiplicité vient du fait qu'il existe des points de vue variés. En effet, les architectures logicielles sont un sujet de recherche qui est apparu parallèlement dans différents domaines. Les différentes définitions sont donc fortement influencées par le domaine de leurs auteurs et convergent lentement [128].

L'objectif principal de l'architecture logicielle consiste à appliquer à la conception de logiciels des démarches similaires à celles de la conception d'architecture de processeurs et d'ordinateurs. Pour cela, elle se base sur des concepts similaires aux concepts d'architectures d'ordinateurs à savoir les concepts de composants et de connecteurs (Figure 3.1). Les

composants, peuvent provenir de sources variées et peuvent être réalisés dans des technologies diverses. La résolution du problème d'interopérabilité est un des objectifs de l'architecture logicielle [5]. Le raisonnement à divers niveaux d'abstraction est une technique largement utilisée en architecture logicielle pour isoler les aspects technologiques et réduire la complexité des logiciels. Par cette technique, l'architecture logicielle permet d'une part, une exploitation aisée et efficace des diverses innovations technologiques en génie logiciel et d'autre part, une prise en charge de la construction de systèmes très complexes utilisant des grains de diverses tailles provenant de diverses sources [5].

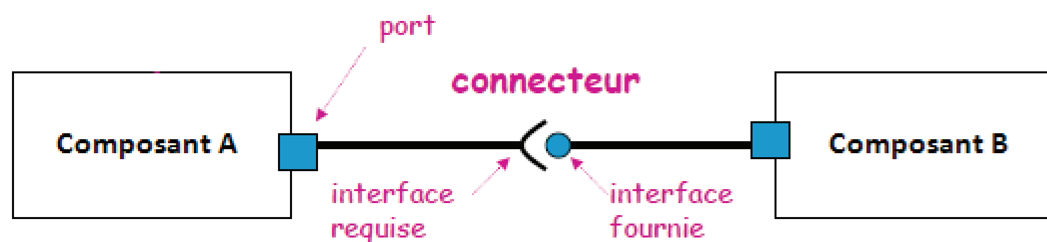


Figure 3.1 : Concepts de base de l'architecture logicielle

Un composant logiciel est une entité responsable de la réalisation d'une (ou plusieurs) fonctionnalité(s) bien précise(s) dans une architecture à un certain niveau d'abstraction. C'est une entité qui fournit des fonctionnalités de calcul et de stockage. Il interagit avec les autres composants pour réaliser un ou plusieurs objectifs d'une architecture. Un composant peut être très simple, comme une fonction C, un objet d'une classe C++ ou complexe comme un serveur HTTP ou un SGBD.

La structure externe d'un composant est caractérisée par ses interfaces. Les interfaces servent à décrire l'ensemble des services fournis par le composant et l'ensemble des besoins nécessaires à son fonctionnement et à la réalisation des services qu'il offre. C'est la partie visible d'un composant. Une interface est composée d'un certain nombre de points d'interaction appelés ports de communication dans Wright [129] et C2 [130] ou players dans UNICON [131].

Les connecteurs gèrent les interactions entre les composants. Un connecteur doit être indépendant de la logique de l'application. Cet aspect ouvre la voie vers les possibilités de changements statiques ou dynamiques des connecteurs sans aucun impact sur la fonctionnalité

de l'architecture du logiciel [5]. Un connecteur est doté d'interface qui indique les services fournis et requis par les composants que le connecteur est capable de transporter.

3.2.1 Avantages de l'architecture logicielle

Différents avantages de l'architecture logicielle ont été identifiés par [132, 133], repris par [128] :

- L'architecture donne une représentation d'un système à un haut niveau d'abstraction, offrant ainsi une compréhension meilleure. Cette vue du système met en valeur la plupart des décisions de conception ainsi que leurs conséquences.
- Identification plus facile des composants réutilisables d'un système.
- L'architecture offre une vue précise des dépendances entre les composants. Cette vue est nécessaire pour connaitre les impacts de la modification d'un composant sur les autres composants du système et donc les conséquences des différentes évolutions.
- La vue abstraite fournie par l'architecture permet de connaitre différentes caractéristiques telles que la consistance du système, son respect du style architectural ou encore d'autres attributs de qualité.
- En se basant sur les dépendances entre les composants, l'architecture garantit une gestion plus précise des coûts et des risques de modifications. Elle permet également une évaluation des qualités du système dans son ensemble, mais aussi des qualités de chaque composant.

3.2.2 Les langages de description d'architectures

Les langages de description d'architectures ADL (Architecture Description Language) constituent le moyen de décrire une vue architecturale d'un logiciel [127]. Ils ont pour objectif premier d'aider les concepteurs à structurer et composer leurs éléments logiciels pour former des applications. Ils permettent aussi d'analyser et de vérifier tôt dans le cycle de développement les propriétés que le futur système devra satisfaire. Toutefois, il n'existe pas de consensus à ce jour sur la vue architecturale qu'il faudrait décrire. Par exemple, pour [10], les ADL représentent les architectures logicielles en termes de structures et de comportement.

Les travaux sur les formalismes de description d'architectures logicielles ont donné naissance à plusieurs ADL, parmi ces ADL nous trouvons :

- Rapide [134] : est un langage de modélisation abstraite. Son objectif principal est de fournir un langage de prototypage qui permet de vérifier des propriétés de type synchronisation, concurrence et flots de données sur des assemblages de composants.

- Wright [129] : est une proposition ultérieure à Rapide pour la modélisation architecturale. Tout comme Rapide, Wright est un langage qui permet de prototyper des systèmes complexes. Mais contrairement à Rapide, il se concentre plus sur la vérification des aspects architecturaux comme la vérification de conformité des ports interconnectés.
- Darwin [135] : est un langage développé pour décrire l'organisation structurelle de systèmes répartis évoluant dans le temps. Il est fondé sur deux abstractions principales : les composants et les services. Les composants constituent les unités d'encapsulation de comportements et interagissent avec leur environnement via des services. Un service correspond à une opération implantée ou requise par le composant.

3.3 L'approche intégrée d'architecture logicielle

L'approche intégrée IASA (Integrated Approach to Software Architecture) est une approche définie dans le contexte d'une collaboration entre l'ESI[1] et le laboratoire LINA[2] de l'université de Nantes. IASA se distingue par des modèles homogènes, un langage de description d'architecture et une méthodologie de conception. Les modèles sont adaptés à la spécification sans contraintes d'architectures logicielles, à la mise en évidence et la prise en charge d'aspects spécifiques dans une architecture logicielle et à la spécification simultanée des aspects structurels et comportementaux. L'homogénéité des modèles est au centre de la méthodologie de conception de l'approche intégrée [5].

3.3.1 Le modèle de composant de l'approche intégrée

Le concept de composant est utilisé pour représenter n'importe quel élément rentrant dans la définition fonctionnelle d'une application. Cela veut dire que toute fonctionnalité faisant partie de la logique d'une application est explicitement prise en charge par un composant. Les connecteurs sont exclus de cet ensemble, vu qu'ils sont neutres vis-à-vis des fonctionnalités d'une application. Un composant est un type. La création d'instance est contrôlée par le type. C'est ainsi qu'il est possible de fixer le nombre total d'instances pouvant être crées. Cette technique nous permet de définir des situations où un composant qui paraît logiquement instancié plusieurs fois n'est en réalité qu'une instance partagée.

[1] ESI http://www.esi.dz/
[2] Lina http://www.lina.univ-nantes.fr/

Nous distinguons deux types de composants : les composants primitifs et les composants composites. La structure interne d'un composant primitif est inaccessible. Celle d'un composant composite possède une organisation bien précise. Elle est composée de deux parties. Une première partie, appelée partie opérative, comprend les composants réalisant les fonctionnalités de l'architecture. La deuxième partie, appelée partie contrôle, contient des composants qui réalisent les opérations de contrôle global sur les autres composants et des composants qui supportent les aspects techniques de l'application. L'instanciation d'un composant est réalisée dans le contexte du concept d'enveloppe. Une enveloppe permet d'isoler l'instance pure d'un composant de son environnement d'exploitation en fournissant à ce dernier les éléments nécessaires à l'exploitation de l'instance. L'enveloppe est l'endroit où seront solutionnés les divers problèmes liés au déploiement de l'instance du composant et à la spécification de topologies très variées, notamment celle mettant en œuvre directement les points d'accès de port.

Dans ce qui suit, nous allons présenter en détail les éléments fondamentaux du modèle de composant de l'approche intégrée. Nous commencerons par les éléments modélisant la vue externe, à savoir le concept de point d'accès et le concept de port. Nous présenterons ensuite le concept d'enveloppe et nous étudierons le modèle de la vue interne.

3.3.1.1 Modélisation de la vue externe

3.3.1.1.1 Les points d'accès

Le point d'accès est le plus petit élément manipulable dans une architecture. Dans notre modèle, les points d'accès représentent les concepts de base échangés entre deux ports de composants. En effet, toute information, quelque soit sa nature arrive ou part d'un composant à travers un point d'accès. Il permet de localiser les diverses ressources fournies ou requises à travers un port et de renseigner sur les éléments intervenant dans la réalisation d'un comportement observable sur un port. Pour l'instant, les concepts supportés par les points d'accès se réduisent à l'échange de données et au transfert du flux de contrôle. Ainsi, un paramètre d'une méthode peut être associé à un point d'accès vu qu'il véhicule une information.

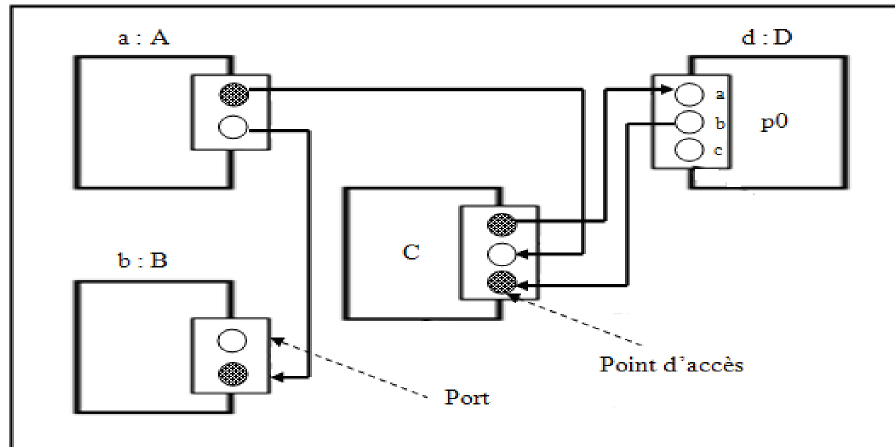


Figure 3.2 : Connexions utilisant les points d'accès

Les points d'accès sont tous représentés par le type de base IASA_AcessPoint. Le type AnyPoint possède un mode de communication bien précis, il est doté de propriétés spécifiques aux points d'accès et d'opérations de base notamment celles destinées à des opérations réflexives [136]. Un point d'accès peut être marqué ou non marqué (marked, unmarked). Un point d'accès marqué est un point d'accès correctement connecté. Cette caractéristique est très utile dans le processus de validation d'une architecture et est exploitée dans un processus de conception du général vers le particulier avec validation progressive de l'architecture durant les différentes phases de conception, sans nécessité que les composants utilisés soient effectivement réalisés. Un point d'accès est destiné soit au transfert de données (Data Oriented Access Point (DOAP)) soit à émettre ou recevoir des actions (Action Oriented Access Point (ACTOAP)) (Figure 3.3). Un ACTOAP indique la présence d'un service qui peut être initié à partir de ce point. Il est dédié aux transferts de flux de contrôle (invocation d'une action, reprise du flux de contrôle par une action).

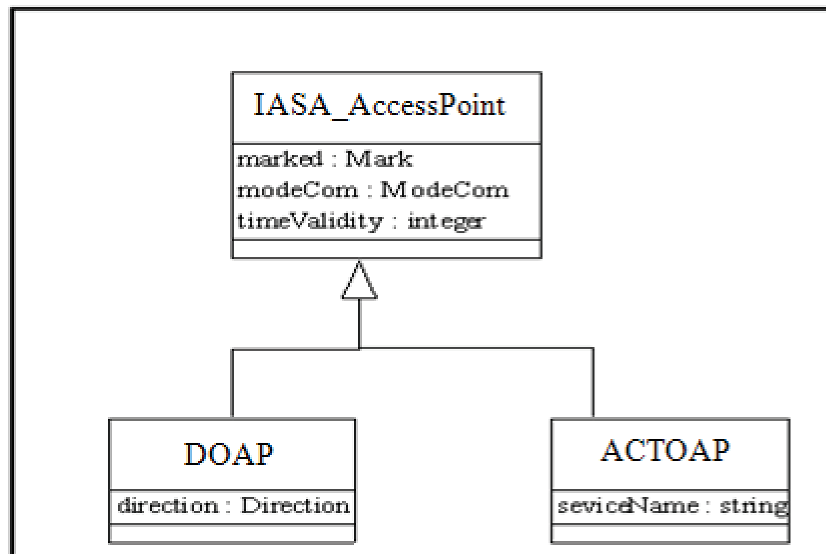


Figure 3.3 : Diagramme de classes du point d'accès

3.3.1.1.1.1 Les points d'accès aux données (DOAP)

Un DOAP est utilisé pour spécifier un transfert explicite de données. L'architecte peut utiliser un ensemble de DOAP prédéfinis ou définir un DOAP qui lui est spécifique. Les DOAP prédéfinis englobent les types de données primitifs que nous retrouvons dans les divers langages de programmation (entier, réel, caractère, booléen) ainsi que des types spécifiques à l'approche IASA, tels que les types de données renseignant sur l'état structurel d'un composant composite. Dans le contexte actuel où le langage Java est utilisé comme cible dans les diverses opérations de validation des modèles de l'approche intégrée, les types primitifs associés aux DOAP correspondent aux types primitifs Java. A titre d'exemple, IntDOAP, ByteDOAP, CharDOAP et BooleanDOAP sont successivement associés aux types primitifs int, byte, char et boolean. Les points d'accès SubCmpSetDOAP, ConSetDOAP, DConSetDOAP, OpPartPortSetDOAP, et CtrlPartPortSetDOAP sont associés à des types de données renseignant sur l'état structurel d'un composite (i.e. listes de composants, de connecteurs, de ports internes, etc..). La définition de nouveau DOAP spécifique doit suivre un style de nommage et une méthodologie de définition bien précise. Le style de nommage apparaît clairement dans les exemples précédents. Le nom du nouveau type commence par le nom du type associé au point d'accès et se termine par le terme DOAP.

Dans notre cas de travail sur le codesign matériel/logiciel, et pour pouvoir prendre en charge les composants matériels nous avons défini de nouveaux DOAP, qui sont :

- StlogicDOAP : pour le type signal.
- TimeDOAP : pour le type time.

- BitDOAP : pour le type bit (0,1).
- UnsignedDOAP : pour les entiers non signés.
- NatDOAP : pour les entiers naturels.
- PosDOAP : pour les entiers positifs.

Un point d'accès de type DOAP est doté d'attributs indiquant le sens des opérations de transfert de données (Figure 3.3). Trois valeurs sont possibles pour spécifier le sens des données : in, out et inout. L'attribut in indique la nécessité de pourvoir le point d'accès d'une donnée. L'attribut out indique que le point d'accès est une source de données. L'attribut inout spécifie que la donnée peut être transférée dans les deux directions. L'accès à un point d'accès inout est par défaut synchronisé.

Comme ceci est le cas dans plusieurs approches [137], les variables globales (mémoire partagée) sont considérées comme des implémentations possibles du concept de DOAP possédant l'attribut inout. Dans notre cas, un point d'accès inout spécifie que le concept associé possède une existence réelle et n'est pas réduit à une référence. Ainsi si deux points d'accès inout de deux composants sont connectés, il y aura alors deux copies de la donnée associée aux DOAP, chacune localisée dans son composant. Si deux points d'accès inout sont connectés, les deux données associées aux points d'accès auront toujours les mêmes valeurs.

La connexion de DOAP suit les règles suivantes : Un DOAPin (respectivement DOAPout) ne peut se connecter qu'à un DOAPout (respectivement DOAPin) ou DOAPinout. Un DOAPinout est connectable à un DOAPin, DOAPout et DOAPinout. Lorsqu'un point d'accès est correctement connecté, il est alors marqué (positionnement de l'attribut de marquage du point d'accès à la valeur marked).

Les DOAP peuvent être explicitement spécifiés avec des valeurs d'initialisation (Figure 3.4). Un point DOAPin initialisé est un point marqué qui peut être non connecté. Les DOAPin avec initialiseurs sont utilisés pour spécifier des valeurs par défaut pour des DOAP qui, éventuellement, ne seraient pas connectés. Si le DOAPin avec initialiseur est connecté, la valeur de l'initialiseur n'aura aucun sens. Un DOAPout avec initialiseur est un point représentant une valeur constante qui ne peut pas être altérée durant l'exécution.

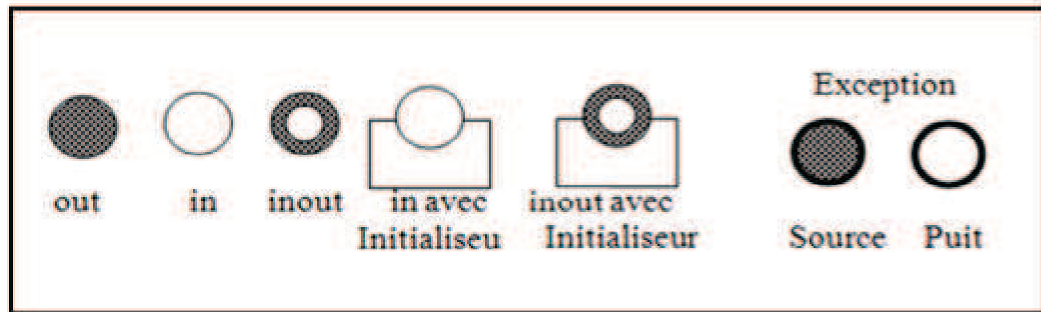


Figure 3.4 : Représentation graphique des DOAP

3.3.1.1.1.2 Les points d'accès de services (ACTOAP)

Un point d'accès de type ACTOAP indique qu'un service peut être initié à partir de ce point. Dans la version actuelle du modèle IASA, nous considérons qu'un ACTOAP correspond uniquement à un seul service et un service permet de réaliser plusieurs actions. Généralement le nombre d'actions que peut prendre en charge un service, est assez restreint et concerne un aspect bien précis d'un domaine d'application. C'est cette idée qui est à la base de la définition de la notion de contexte d'action au niveau du langage 3ADL. Vis-à-vis d'un service, un point d'accès de type ACTOAP ne peut être que fournisseur ou client. Un ACTOAP à travers lequel un service est fourni est représenté par le type spécifique ServerACTOAP (ACTOAPS). Lorsqu'un ACTOAP spécifie un besoin de service, il est alors représenté par le type spécifique ClientACTOP (ACTOAPC) (Figure 3.5).

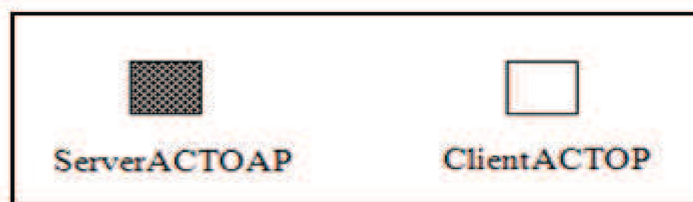


Figure 3.5 : Représentation graphique des ACTOAP

3.3.1.1.2 Les ports

Un port est un regroupement de points d'accès, étroitement associés dans le contexte de la réalisation d'un objectif commun. Tous les ports sont représentés par le type AnyPort. Un port possède un attribut renseignant sur le nom de l'instance. Le port représente un espace de nom pour les points d'accès. Chaque point d'accès est identifié de manière unique dans le contexte d'un port. Le composant représente un espace de nom pour un port. Ce dernier est identifié de manière unique dans un composant.

Les ports modélisent la vue externe d'un composant. C'est seulement à travers les ports qu'un composant est manipulable. Les ports divulguent les ressources (services et données) requises et fournies d'un composant ainsi que les aspects comportementaux du composant. Les comportements sont décrits dans le langage d'action 3ADL [138] qui est une des composantes essentielles de l'approche IASA. Un port est une entité à part, pouvant être ajouté ou supprimé à un composant. Il peut en outre être modifié par l'ajout ou la suppression de point d'accès, notamment les points d'accès de données. Cette situation est souvent rencontrée avec les composants représentant des opérateurs et les composants d'initialisation.

3.3.1.1.3 L'enveloppe

La vue externe de tout composant est formée d'une enveloppe. Le concept d'enveloppe a été introduit pour permettre d'atteindre les objectifs suivants [5]:

- L'isolation totale de la vue interne d'un composant du monde externe.
- Offrir le support nécessaire à la réalisation de topologies qu'il n'est pas possible de réaliser dans le contexte de ports typés par les interfaces, comme c'est le cas dans les divers ADL et UML.
- Offrir le support nécessaire au déploiement des instances d'un composant. Selon son environnement d'existence, une instance sera enveloppée par l'enveloppe adéquate[3].
- Offrir le support nécessaire à la réalisation des opérations de validation.
- La transformation de n'importe quel composant (COTS[4], anciennes applications) provenant de n'importe quelle source, en un composant prêt à être exploité dans les opérations d'assemblage selon le modèle de composant de l'approche intégrée.

3.3.1.2 Organisation de la vue interne

La vue interne est organisée en deux parties (Figure 3.6). Une partie opérative et une partie contrôle.

[3] Une instance s'habille adéquatement pour une situation particulière
[4] Component On The Shelf

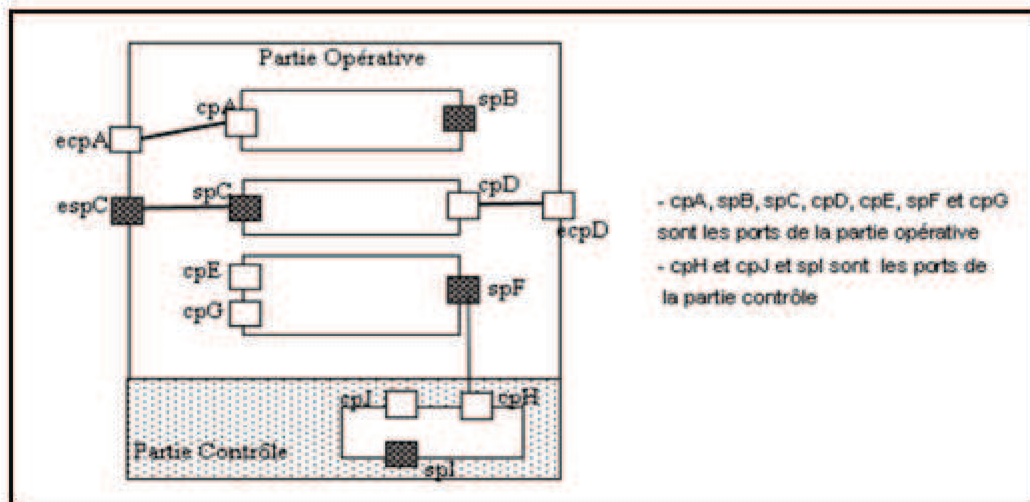


Figure 3.6 : Vue interne d'un composant composite

3.3.1.2.1 La partie opérative

Elle contient les composants représentant par leurs interconnexions la logique générale du composite à un moment bien précis. Les instances de composants et de connecteurs sont soit statiques soit dynamiques. Une instance statique est définie en tant que telle à la spécification. Elle est toujours présente dans la partie opérative et ne peut en être supprimée. Une instance dynamique peut disparaître et apparaître durant tout le cycle de vie de l'instance de son composite. Elle peut ainsi être créée, s'auto détruire ou être supprimée.

En plus des instances de types publiques de composants, la partie opérative peut comporter l'instanciation de type de composants internes. L'instanciation de ces types internes n'est possible qu'au niveau de la partie opérative du composant dans lequel les types internes ont été définis. Les types internes ont été introduits pour représenter des aspects très spécifiques à un composant. Les interfaces homme machine sont un exemple d'applications où il y a une mise en œuvre intense du concept de types internes de composant.

3.3.1.2.2 La partie contrôle

La partie contrôle réalise les diverses opérations de contrôle sur les composants de la partie opérative, telles que la gestion du flux de contrôle des divers services (arrêt, lancement en séquence ou en parallèle), le contrôle de l'évolution structurelle, la gestion des exceptions, l'exportation des états du composant et la génération de log. La partie contrôle est composée d'au moins un composant, appelé contrôleur. Ce dernier est un composant dit comportemental, sa spécification est faite complètement en langage d'action 3ADL.

3.3.1.3 Déploiement des composants

Le modèle de composant de l'approche IASA permet une haute flexibilité dans la spécification des propriétés de déploiement des composants. Ces propriétés sont utilisées pour produire le code, et éventuellement les descripteurs de déploiement effectif nécessaires. Ces codes et descripteurs sont utilisés pour charger les composants dans leurs environnements d'exécution. Le déploiement adressé dans IASA, ne concerne pas la méthode de chargement et de lancement des composants dans leurs environnements mais la production du code et des descripteurs qui seront utilisés par les outils de déploiement effectif des composants dans leurs environnements d'exécution. La spécification du déploiement consiste à définir pour un type de composant des cas de déploiement et des plans de déploiement. Les cas de déploiement renseignent sur les formes réelles que le composant pourra avoir une fois en exécution (i.e. tache, processus). Le plan de déploiement indique comment un cas de déploiement est appliqué (i.e. tâche dans un processus fonctionnant sur la machine locale).

3.3.1.3.1 Les cas de déploiement

Le cas de déploiement spécifie l'état du composant à l'exécution (i.e. tache principale, web service, processus, hardware, software, hardware/software). L'approche IASA a identifié plusieurs cas de déploiement direct parmi lesquels nous citons à titre d'illustration: PROCESS, THREAD, APPLET, SERVLET, EJB[5], et JAVASCRIPT. Il est clair que cette liste est extensible pour la prise en charge de cas de déploiement divers. Dans notre cas, il serait nécessaire d'introduire des cas de déploiement qui indiquent que la nature du composant à déployer est hardware, software ou hardware/software. Chaque cas de déploiement doit être spécifié avec son environnement. A titre d'exemple, pour le cas PROCESS, il est nécessaire d'indiquer le type du système d'exploitation et l'adresse Internet de la machine.

En plus de ces cas de déploiement, IASA définit trois autres cas de déploiement à travers lesquels est spécifiée la nature globale du flux de contrôle. Ces cas sont FULLY_CONTROLLED_FLOW, CONCURRENT_FLOW et FREE_FLOW. Ces cas additionnels peuvent être utilisés pour la synthèse de l'architecture finale, prête à l'exécution. Dans cette architecture, les connecteurs qui ne sont pas compatibles avec ces cas de

---

[5] Le choix de ces cas est dû à l'utilisation du langage JAVA qui dispose d'une grande variété de technologies d'implémentation.

déploiement seront inhibés durant l'exécution de l'architecture et aucune communication ne sera véhiculée par les connecteurs inhibés.

3.3.2 L'ADL de l'approche intégrée

La première version de l'ADL de l'approche IASA, appelée SEAL (Simple and eXtensible Architecture Language) [11] a été définie afin de démontrer le concept de composant exécutable à un haut niveau d'abstraction. Ensuite, pour combler les défaillances de SEAL et pour permettre d'exploiter le modèle IASA de manière efficace, un nouveau langage appelé 3ADL (Architecture, Aspect and Action Description Language) a été défini pour l'approche IASA. La forme XML de ce langage a été aussi définie d'où l'appellation x3ADL (eXtensible Architecture, Aspect and Action Description Language) [12]. x3ADL est défini pour être fortement extensible, flexible et adaptable afin de servir aux expérimentations au fur et à mesure de l'avancée des recherches. Le langage x3ADL rejoint la famille des langages de description d'architecture basés sur le langage XML, tels que XArch [139], xADL [140] et xAcme [141].

Dans le cadre de notre travail sur le codesign matériel/logiciel, nous avons enrichi le langage x3ADL pour lui permettre de prendre en charge à la fois les composants matériels et les composants logiciels.

3.3.2.1 Description des balises du langage x3ADL

Un document x3ADL est hiérarchisé en plusieurs niveaux, le premier niveau est illustré dans la figure 3.7. La balise <Component> possède les attributs name qui donne le nom du composant et deployedAs qui renseigne sur le cas de déploiement du composant. Les cas de déploiement possibles sont : HARD pour un composant matériel, SOFT pour un composant logiciel et HS pour un composant mixte.

```
- <Component name="" deployedAs="">
    <!-- DEFINITION DU COMPOSANT  -->
  - <Ports>
      <!-- DEFINITION DES PORTS  -->
    </Ports>
  - <Connectors>
      <!-- DEFINITION DES CONNECTEURS  -->
    </Connectors>
  - <OperativePart>
      <!--  DEFINITION DE LA PARTIE OPERATIVE  -->
    </OperativePart>
  - <ControlPart>
      <!--  DEFINITION DE LA PARTIE CONTROLE  -->
    </ControlPart>
  - <Properties>
      <!-- SPECIFICATION DES PROPRIETES  -->
    </Properties>
  </Component>
```

Figure 3.7 : Les balises du langage x3ADL

3.3.2.1.1 La balise Ports

Cette balise est une descendante directe de la balise Component. Elle représente des types de ports internes au composant, non-instanciables dans d'autres composants. Cette balise englobe les balises InDataPort et OutDataPort (Figure 3.8).

```
- <Ports>
  - <InDataPort>
      <!-- LES INPUTS DU COMPOSANT  -->
    </InDataPort>
  - <OutDataPort>
      <!-- LES OUTPUTS DU COMPOSANT  -->
    </OutDataPort>
  </Ports>
```

Figure 3.8 : La balise Ports

3.3.2.1.1.1 La balise InDataPort

Cette balise indique les entrées du composant composite, c'est-à-dire les DOAP ayant le sens IN (DOAPin). Elle est composée de plusieurs balises AccessPoint.

Pour les composants déployés en tant que hard, nous avons ajouté les attributs suivants (Figure 3.9) :

- L'attribut packedDimension qui donne le nombre de bit sur lequel est codé le signal.

- L’attribut kind qui introduit les vecteurs.
- L’attribut library qui renseigne sur la bibliothèque du point d’accès.

```xml
- <InDataPort>
  <!-- CAS D UN COMPOSANT DEPLOYE EN TANT QUE SOFT -->
  <Acesspoint name="a" type="intDataPoint" />
  <Acesspoint name="b" type="intDataPoint" />
  <Acesspoint name="c" type="intDataPoint" />
</InDataPort>
```

```xml
- <InDataPort>
  <!-- CAS D UN COMPOSANT DEPLOYE EN TANT QUE HARD -->
  <Acesspoint name="clk" type="StlogicDataPoint" kind="" />
  <Acesspoint name="reset" type="StlogicDataPoint" kind="" />
  <Acesspoint name="inalu" type="StlogicDataPoint" kind="" />
  <Acesspoint name="opalu" type="StlogicDataPoint" kind="" />
  <Acesspoint name="ldflag" type="StlogicDataPoint" kind="" />
  <Acesspoint name="entreea" type="StlogicDataPoint" kind="array" library="work" packedDimension="(7 downto 0)" />
  <Acesspoint name="entreeb" type="StlogicDataPoint" kind="array" library="work" packedDimension="(7 downto 0)" />
</InDataPort>
```

Figure 3.9 : Exemples de la balise InDataPort

3.3.2.1.1.2 La balise OutDataPort

Cette balise indique les sorties du composant composite, c'est-à-dire les DOAP ayant le sens OUT (DOAPout). Elle est composée de plusieurs balises AccessPoint (Figure 3.10).

```xml
- <OutDataPort>
  <!-- CAS D UN COMPOSANT DEPLOYE EN TANT QUE SOFT -->
  <Acesspoint name="s" type="FloatDataPoint" />
</OutDataPort>
```

```xml
- <OutDataPort>
  <!-- CAS D UN COMPOSANT DEPLOYE EN TANT QUE HARD -->
  <Acesspoint name="resultat" type="unsignedDataPoint" kind="array" library="work" packedDimension="(7 downto 0)" />
  <Acesspoint name="flag" type="StlogicDataPoint" kind="" />
</OutDataPort>
```

Figure 3.10 : Exemples de la balise OutDataPort

3.3.2.1.2 La balise Connectors

La balise Connectors regroupe un ensemble de balises connector, dans lesquelles les différents types de connecteurs sont définis (Figure 3.11 A). Chaque connecteur est identifié par son nom et est défini par ses rôles, son architecture et son comportement (respectivement les balises Roles, ConnectorArchitecture et Behaviour) (Figure 3.11 B).

A

```
- <Connectors>
  - <connector>
      <!-- Definition du type de connecteur 1 -->
    </connector>
  - <connector>
      <!-- Definition du type de connecteur 2 -->
    </connector>
  - <connector>
      <!-- Definition du type de connecteur n -->
    </connector>
  </Connectors>
```

B

```
- <connector name="">
  - <Roles>
      <!-- Declaration des roles -->
    </Roles>
  - <ConnectorArchitecture>
      <!-- Declaration du type de connecteur -->
    </ConnectorArchitecture>
  - <Behaviour>
      <!-- Definition du comportement -->
    </Behaviour>
  </connector>
```

Figure 3.11 : La balise Connectors

3.3.2.1.3 La balise OperativePart

Cette balise renseigne sur les composants primitifs ou composites qui composent le composant composite. De plus, pour les composants de type hardware, cette balise indique l'ensemble des bibliothèques nécessaires à l'implémentation du composant matériel grâce à la balise useClause qui utilise l'attribut path (Figure 3.12).

```
<OperativePart>
  <!-- CAS D UN COMPOSANT DEPLOYE EN TANT QUE HARD  -->
- <Components>
    <useClause path="ieee.STD_LOGIC_UNSIGNED.all" />
    <useClause path="ieee.std_logic_1164.all" />
  </Components>
</OperativePart>
```

```
<OperativePart>
  <!-- CAS D UN COMPOSANT DEPLOYE EN TANT QUE hardsoft  -->
- <Components>
    <import name="cpro1" type="soft" />
    <import name="alu8" type="hard" />
  </Components>
</OperativePart>
```

Figure 3.12 : Exemples de la balise OperativePart

3.3.2.1.4 La balise ControlPart

La partie contrôle est décrite par trois balises principales (Figure 3.13), qui sont :

- La balise DelegateConnectors: le connecteur de délégation se charge de faire la correspondance entre les points d'accès du composant composite et les points d'accès des composants qui le compose.
- La balise OPControler: le contrôleur de la partie opérative gère les connexions entre les divers composants.
- La balise DataFlow: renseigne sur le flux de données.

```
- <ControlPart>
  - <DelegateConnectors>
      <map var="a" to-cmp="cpro1" in="param0" />
      <map var="b" to-cmp="cpro1" in="param1" />
      <map var="c" to-cmp="cpro1" in="param2" />
      <map var="resultat" to-cmp="alu8" in="param3" />
    </DelegateConnectors>
  - <OPControler name="CoOpPartCtrler">
    - <Connecions>
        <connect cmp="cpro1.s" To_cmp="alu8.clk" />
      </Connecions>
    </OPControler>
  - <DataFlot>
      <fire name="cpro1" />
      <fire name="alu8" />
    </DataFlot>
  </ControlPart>
```

Figure 3.13 : Exemple de la balise ControlPart

3.3.2.1.5 La balise Properties

Cette balise définit les éléments de l'architecture et décrit le déploiement des composants à travers les balises architecture et deployment. La balise architecture donne des informations générales sur la machine comme son nom et son système d'exploitation. La balise deployment donne des informations sur le déploiement du composant.

3.4 Conclusion

Dans ce chapitre, nous avons survolé le domaine de l'architecture logicielle. Le but de notre travail consiste à résoudre les problèmes de la cosynthèse des interfaces de communication dans les systèmes de codesign par l'application du modèle de l'approche intégrée IASA. Pour cela, nous avons détaillé les concepts fondamentaux de cette approche intégrée.

L'architecture logicielle connaît actuellement un succès important dans le but de faire face, non seulement, aux difficultés rencontrées lors de la conception de gros logiciels et les caractéristiques de la programmation à grande échelle, mais aussi aux besoins de la réutilisation. C'est ainsi que l'architecture logicielle des systèmes est devenue, pendant la dernière décennie, un sous-domaine central du génie logiciel [142].

Par la vue abstraite qu'elle donne en termes de composants et de connecteurs, l'architecture logicielle permet de faciliter la compréhension du système et de l'analyser de manière efficace. Elle peut aussi évaluer la qualité d'un système, le documenter ou encore aider dans la phase de maintenance.

Dans le chapitre qui suit, nous allons présenter notre approche de synthèse des communications selon IASA.

CHAPITRE 4
SYNTHESE DES COMMUNICATIONS SELON L'APPROCHE IASA

4.1 Introduction

Après l'étude des différentes approches de cosynthèse dans le chapitre 2, nous sommes arrivés à la conclusion suivante: l'étape de la cosynthèse des interfaces de communication dans le codesign souffre du niveau d'abstraction considéré pour l'élaboration du système qui n'est pas assez élevé pour prendre en charge tous les aspects de la communication entre le matériel et le logiciel d'une manière efficace. C'est pour tenter de résoudre ce problème, que nous proposons dans ce travail d'appliquer une solution de l'architecture logicielle qui est l'approche intégrée IASA (décrite dans le chapitre 3) à la cosynthèse des interfaces de communication.

Dans ce chapitre, nous présentons les concepts et les techniques développés pour l'approche IASA pour gérer un processus de communication entre un composant matériel et un composant logiciel d'un système de codesign matériel/logiciel. Ce processus commence par une cospécification abstraite du système et s'achève par un processus de transformation de la vue abstraite en une vue d'implémentation dans une technologie de réalisation bien précise. Au niveau abstrait nous avons introduit un modèle de composant IASA spécifique pour la représentation des composants matériels, des composants logiciels et des composants mixtes à un haut niveau d'abstraction. Au niveau du processus de transformation, nous avons enrichi IASA par trois nouveaux cas de déploiement spécifiques aux systèmes de codesign. Concernant la production de la vue implémentation, nous avons utilisé le langage JAVA pour les composants logiciels el le langage VHDL pour les composants matériels. Actuellement l'approche IASA se base fondamentalement sur la technologie JAVA. Nous avons ainsi enrichi le processus de transformation de l'approche IASA par un ensemble de règles de transformation permettant de produire des composants logiciels réalisés totalement en JAVA et des composants matériels réalisés totalement en VHDL, à partir d'une cospécification en x3ADL.

4.2 Cospécification du système de codesign dans IASA

La cospécification dans le processus de codesign qui est une spécification unifiée de la partie matérielle et de la partie logicielle, consiste à décrire le système à concevoir d'une

manière abstraite. C'est une étape déterminante car elle a un impact direct sur les résultats. Dans le chapitre 1, nous avons présenté les deux méthodes existantes pour la cospécification qui sont la spécification homogène et la spécification hétérogène. Pour notre travail, nous avons opté pour une cospécification homogène. C'est-à-dire que nous avons fait le choix d'utiliser un seul langage de spécification qui est le langage x3ADL (décrit dans le chapitre3), pour spécifier à la fois les composants matériels et les composants logiciels.

Ce choix d'utiliser une spécification homogène plutôt qu'une spécification hétérogène est motivé par les raisons suivantes :

- ✓ La spécification homogène par le modèle unifié qu'elle donne, simplifie le développement d'applications complexes.
- ✓ La spécification hétérogène contredit l'hypothèse de base du codesign qui est la séparation entre le matériel et le logiciel le plus tardivement possible dans le processus de conception.
- ✓ La spécification hétérogène donne une affectation du matériel et du logiciel avant même l'étape de partitionnement, ce qui peut éliminer des solutions potentielles de l'espace de conception.
- ✓ La spécification hétérogène rend l'étape de la validation et de la cosynthèse d'interfaces beaucoup plus difficiles.

4.3 Représentation des composants d'un système de codesign dans IASA

Dans IASA, tout composant IASA est réalisé à l'aide de composant primitif représenté par des POJO[1] dont l'objectif est de représenter uniquement et purement le métier pour lequel le composant est destiné. Les fonctions techniques requises par un composant lui seront associées grâce à l'approche de conception par aspects supportée par IASA.

Afin de pouvoir gérer un système de codesign, nous avons défini dans le contexte de l'approche IASA un modèle de composant spécifique pour représenter les composants matériels, logiciels et mixtes. En effet, dans notre contexte, un composant primitif peut être soit matériel, soit logiciel. Un composant composite composé d'un composant logiciel et d'un composant matériel sera forcément un composant mixte. La figure 4.1 présente la notation graphique de ce modèle. La spécificité concerne uniquement la vue externe du composant et ne remet pas en cause la vue interne ni les capacités de l'approche IASA à manipuler la vue

[1] Plain Old Java Object

externe d'un composant notamment par l'opération de manipulation individuelle de points d'accès et l'injection des aspects. La vue externe d'un composant représentant un composant de codesign simple ou composite comporte deux ports : un port de service et un port de données. Le port de service qui est du type IASAServerPort [5] comporte en plus du point de service, des points d'accès aux données ayant le sens IN (DOAPin). Le port de données est le port à travers lequel le composant renvoie la réponse. Il contient un point d'accès aux données ayant le sens OUT (DOAPout).

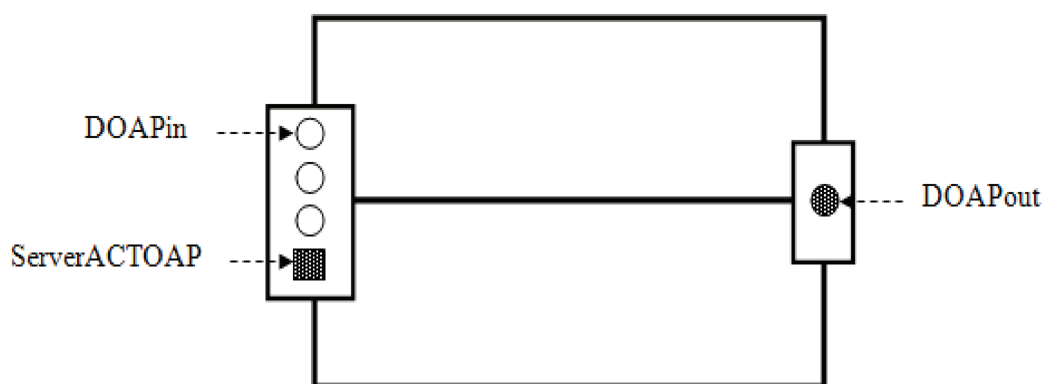


Figure 4.1 : Architecture d'un composant de codesign composite dans IASA

4.4 Spécification du comportement du contrôleur

Le contrôleur est le composant de base dans la spécification de n'importe quel composant composite. Dans IASA, ce composant doit être obligatoirement instancié dans la partie contrôle. Il représente une sorte de procédure principale du composant comme l'est la fonction main dans un programme C ou JAVA. Le contrôleur est un composant comportemental. Dans la version actuelle de l'approche IASA, il ne peut pas être à son tour un composant composite. Il possède un comportement défini en x3ADL. En règle générale, le comportement spécifié concerne l'initialisation du composite, la spécification du flot de contrôle global du composite et la gestion du dynamisme de l'architecture du composite (ajout / suppression de composants, de connecteurs, etc..).

Dans IASA, le flot de contrôle peut être totalement centralisé, totalement décentralisé ou mixte. La centralisation du contrôle se fait par sa spécification au niveau du composant de contrôle de la partie contrôle. La décentralisation est mise en œuvre par le concept d'enveloppe associée aux diverses instances du composant de la partie opérative. Dans la

spécification mixte, le flot de contrôle est géré simultanément par le concept d'enveloppe présent sur toutes les instances de composant et par le composant de contrôle.

Dans notre cas, nous nous sommes limités à la spécification centralisée du flot de contrôle et qui correspond à la communication entre un composant matériel et un composant logiciel. Nous verrons par la suite que cette forme de spécification est associée à un nouveau cas de déploiement que nous avons introduit dans IASA et qui s'appelle HS_COMMUNICATION. Ainsi, dans le contexte d'une communication dans un système de codesign, le contrôleur invoquera les composants (matériel et logiciel) un par un à travers leur port de service et récupèrera à la fin de chaque activation le résultat à partir du port de données. La forme générale en x3ADL de l'activation et la réception du résultat est comme suit :

```
<fire name="nom_du_composant_de_service" result="nom_du_port_de_service"
/>
```

Cependant, comme nous nous basons sur un modèle spécifique de composant IASA pour le codesign (un seul port de service et un seul port de résultats), nous utiliserons la forme simplifiée suivante :

```
<fire name="nom_du_composant_de_service"/>
```

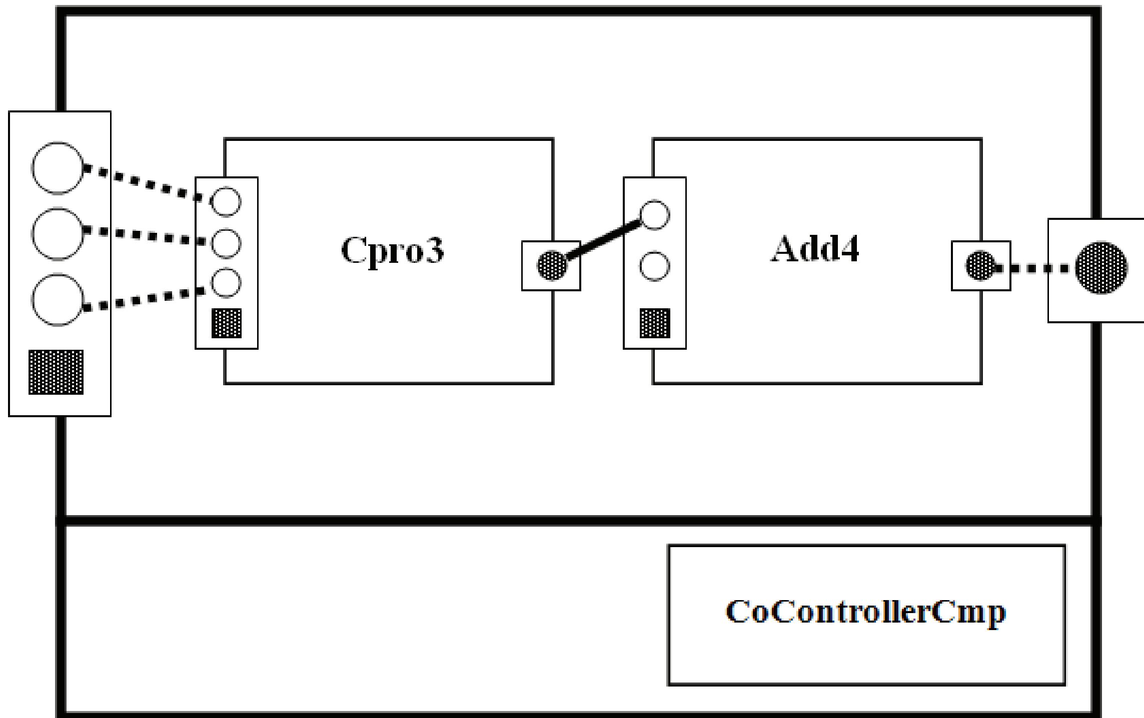


Figure 4.2 : Un composant de codesign composite selon IASA

Le code suivant montre un exemple de spécification simplifiée en x3ADL du flot de contrôle pour le composant composite de la figure 4.2 :

```
<DataFlot>
     <fire name="Cpro3" />
     <fire name="Add4" />
</DataFlot>
```

4.5 Spécification du déploiement

L'approche IASA définit plusieurs cas de déploiement à travers lesquels apparait clairement l'utilisation intensive de la technologie Java comme cible, dans un processus de transformation d'une description x3ADL vers une technologie d'implémentation. Parmi ces cas de déploiement, il y a ceux qui indiquent la nature exacte du composant lors de sa mise en œuvre et ceux qui indiquent la nature du flot de contrôle à appliquer sur l'architecture. Ces derniers se basent principalement sur le flot de contrôle défini au niveau du contrôleur instancié au niveau de la partie contrôle. Parmi les cas de déploiement à travers lesquels la nature du composant est indiquée, nous retrouvons le cas PROCESS, THREAD, EJB[2],

[2] Entreprise Java Beans

SERVLET, APPLET et JAVABEAN. Pour la précision de la nature du flot de contrôle dans une architecture, IASA a défini les trois cas de déploiement suivants : FULLY_CONTROLLED_FLOW, CONCURRENT_FLOW et FREE_FLOW. Actuellement les deux derniers cas de déploiement doivent être utilisés avec beaucoup de précaution car ils ne sont pas encore complètement gérés par IASA[3]. Ils font l'objet d'une recherche dans le cadre d'une thèse de doctorat.

Pour supporter de manière efficace la mise en œuvre des composants de codesign dans le contexte d'une communication entre un composant logiciel et un composant matériel, nous avons introduit trois nouveaux cas : HARD, HS et HS_COMMUNICATION. Le cas HARD indique que le composant doit être déployé en tant que composant matériel et le cas HS indique que le composant doit être déployé en tant que composant mixte matériel/logiciel. Le dernier cas permet d'agir au niveau de la gestion du flot de contrôle. Le cas SOFT existe déjà dans IASA : PROCESS, THREAD, EJB, SERVLET, etc. sont des cas SOFT.

Comme pour le cas FULLY_CONTROLLED_FLOW, le cas HS_COMMUNICATION indique que le flot de contrôle doit être totalement sous le contrôle du composant de contrôle. Dans les cas de déploiement FULLY_CONTROLLED_FLOW et HS_COMMUNICATION, le contrôleur peut remettre en cause la spécification d'une topologie lors de l'opération de transformation d'une spécification x3ADL vers une spécification dans une technologie d'implémentation, afin que celle-ci soit exempte de toute opération de toute activité concurrente non indiquée explicitement au niveau du contrôleur. Dans les cas CONCURRENT_FLOW, FREE_FLOW et HS_COMMUNICATION, les composants peuvent imposer leur flot de contrôle et évoluer indépendamment de la logique du contrôleur. Cependant, cette liberté est limitée pour le cas HS_COMMUNICATION qui doit respecter les exigences imposées par la communication entre le hardware et le software.

[3] Ces deux cas de déploiement sont en cours de traitement dans un travail de recherche dans le cadre d'un doctorat à l'ESI intitulé « Architecture Exécutable » et en préparation par Mr Saadi Abdelfettah sous la direction de Mr Henni et Mr Bennouar.

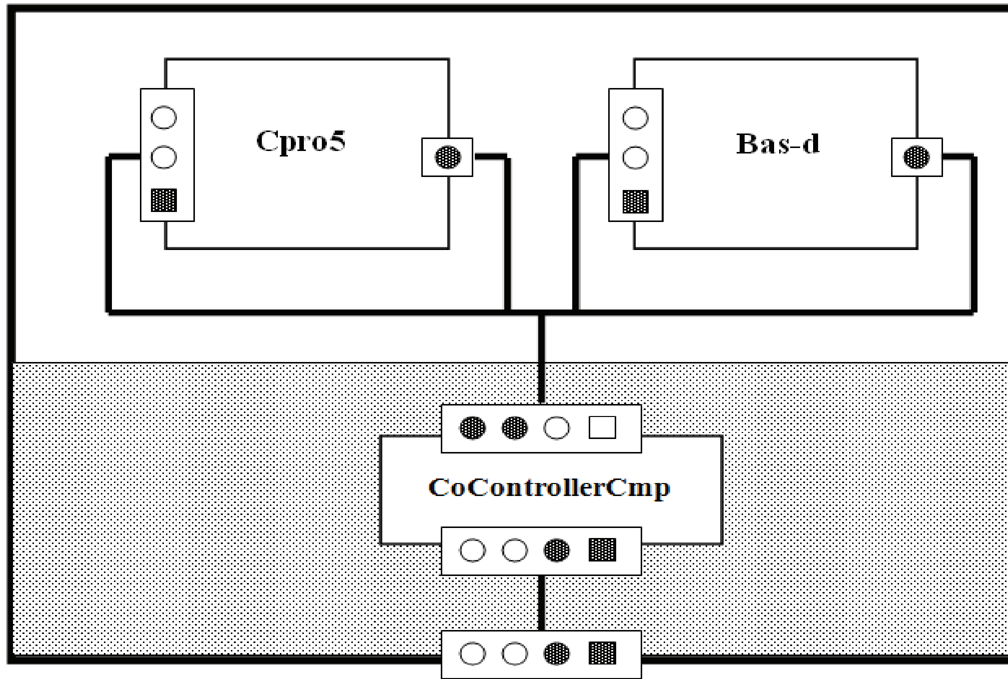


Figure 4.3 : Architecture du flot HS_COMMUNICATION

A titre d'exemple, nous présentons, dans ce qui suit, la description x3ADL des composants de la figure 4.3.

La description x3ADL du composant hardware Bas-d est la suivante :

```
<Component name="Bas-d" deployedAs="HARD">
<!--DEFINITION DU COMPOSANT-->
<Ports>
<!--DEFINITION DES PORTS-->
  <InDataPort>
  <!-- LES INPUTS DU COMPOSANT-->
    <Acesspoint name="clk" type="StlogicDataPoint" kind=""
library="work" />
    <Acesspoint name="d" type="StlogicDataPoint" kind=""
library="work" />
  </InDataPort>
  <OutDataPort>
  <!--LES OUTPUTS DU COMPOSANT -->
    <Acesspoint name="s" type="StlogicDataPoint" kind=""
library="work" />
  </OutDataPort>
</Ports>
<Connectors />
<OperativePart>
<!-- DEFINITION DE LA PARTIE OPERATIVE -->
   <useClause path="ieee.NUMERIC_STD.all" />
   <useClause path="ieee.std_logic_1164.all" />
</OperativePart>
<ControlPart />
<Properties />
</Component>
```

La description x3ADL du composant software Cpro5 est la suivante :

```
<Component name="Cpro5" deployedAs="SOFT">
<!--DEFINITION DU COMPOSANT-->
<Ports>
<!--DEFINITION DES PORTS-->
  <InDataPort>
  <!-- LES INPUTS DU COMPOSANT-->
     <Acesspoint name="a" type="intDataPoint" />
     <Acesspoint name="b" type="intDataPoint" />
  </InDataPort>
  <OutDataPort>
  <!--LES OUTPUTS DU COMPOSANT -->
     <Acesspoint name="r" type="FloatDataPoint" />
  </OutDataPort>
</Ports>
<Connectors />
<OperativePart />
<ControlPart />
<Properties />
</Component>
```

La description x3ADL du composant composite composé des composants Cpro5 et Bas-d est la suivante :

```
<Component name="Co" deployedAs="HS">
<!--DEFINITION DU COMPOSANT-->
<Ports>
<!--DEFINITION DES PORTS-->
     <InDataPort>
     <!-- LES INPUTS DU COMPOSANT-->
          <Acesspoint name="a" type="intDataPoint" />
          <Acesspoint name="b" type="intDataPoint" />
     </InDataPort>
     <OutDataPort>
     <!--LES OUTPUTS DU COMPOSANT -->
          <Acesspoint name="s" type="StlogicDataPoint" kind=""
library="work" />
     </OutDataPort>
</Ports>
<Connectors />
<OperativePart>
<!-- DEFINITION DE LA PARTIE OPERATIVE -->
     <Components>
          <import name="Cpro5" type="SOFT" />
          <import name="Bas-d" type="HARD" />
     </Components>
</OperativePart>
<ControlPart>
<!-- DEFINITION DE LA PARTIE CONTROLE -->
     <DelegateConnectors>
          <map var="a" to-cmp="Cpro5" in="param0" />
          <map var="b" to-cmp="Cpro5" in="param1" />
          <map var="s" to-cmp="Bas-d" in="param2" />
     </DelegateConnectors>
     <OPControler name="CoOpPartCtrler">
        <Connecions>
          <connect cmp="Cpro5.r" To_cmp="Bas-d.clk" />
        </Connecions>
     </OPControler>
     <DataFlot>
          <fire name="Cpro5" />
          <fire name="Bas-d" />
     </DataFlot>
</ControlPart>
<Properties />
</Component>
```

4.6 Transformation d’une description IASA en une spécification de codesign

Lors de la spécification des propriétés de déploiement des composants, l’approche IASA attache un composant à une technique d’implémentation et une nature bien précise du composant dans cette technologie. L’approche IASA permet la spécification libre de topologies à un haut niveau d’abstraction, totalement indépendante des divers mécanismes logiciels, notamment le mécanisme d’appel de procédure. Le processus de transformation de

la vue abstraite en une vue d'implémentation dépend de plusieurs facteurs tels que la mise en œuvre ou non de l'orienté aspect, la topologie spécifiée, le mode de communication des points d'accès (synchrone, asynchrone), la technologie d'implémentation ciblée, les cas de déploiement associés à chaque instance et les standards d'interconnexion utilisés. En règle générale, le processus de transformation d'une spécification x3ADL vers une spécification dans une technologie d'implémentation bien précise est composé de trois phases:

1. Le tissage des aspects
2. La normalisation
3. La production de la vue d'implémentation

Le tissage des aspects se base principalement sur le concept d'enveloppe. C'est au niveau de celle-ci que les types de ports de composant sont transformés par ajout de nouveau point d'accès appelé ASPOAP. La transformation de ces ports entraîne la création de nouveau type interne de ports. Le résultat de cette phase est une description x3ADL dans laquelle les opérations d'injection d'aspects sont totalement résolues. La phase de normalisation transforme une description x3ADL en une description basée sur les concepts ordinaires de port et d'interface. Nous rappelons que le port IASA, contrairement aux ports basés sur le concept d'interface, permet à un haut niveau d'abstraction, la manipulation individuelle de tout élément structurel ou comportemental le définissant. Ceci n'est pas le cas dans les autres approches d'architecture logicielle où le port, appelé aussi interface, est un concept atomique, ne permettant pas la manipulation de ses éléments constitutifs.

Le résultat de la phase de normalisation est une description qui permet une synthèse automatique et claire du niveau d'implémentation. Par exemple, la normalisation permettra d'obtenir une description basée sur les ports et interface UML, une description basée sur les interfaces Java ou une description utilisant les ports d'ArchJava. Dans notre cas, la vue externe d'un composant logiciel sera représentée par une interface Java, et la vue externe d'un composant matériel par une description XML appelée entité, extraite de la description VHDL du composant matériel.

A titre d'exemple, la vue externe du composant software Cpro5 de la figure 4.3 sera représentée par l'interface Java suivante :

```
package iasa.component.soft;
import iasa.port.*;
public interface SoftCpro5Port extends IASASoftSPort{
    public float Cpro5(int a,int b);
}
```

La vue externe du composant hardware Bas-d de la figure 4.3 sera représentée par l'entité XML suivante :

```
<interfaceDefinition version="1.2" language="vhdl" kind="component"
 name="Bas-d">
   <useClause path="ieee.NUMERIC_STD.all" />
   <useClause path="ieee.std_logic_1164.all" />
   <portList>
        <port name="clk" direction="in" type="std_logic" kind=""
 library="work" package="Bas-d" packedDimension="(1)" />
        <port name="d" direction="in" type="std_logic" kind=""
 library="work" package="Bas-d" packedDimension="(1)" />
        <port name="s" direction="out" type="std_logic" kind=""
 library="work" package="Bas-d" packedDimension="(1)" />
   </portList>
 </interfaceDefinition>
```

La phase de normalisation s'achève par la détermination de la topologie finale de l'application. La topologie produite par la phase de normalisation détermine les connecteurs qui seront effectivement mis en œuvre lors du fonctionnement de l'application. Cette phase dépend essentiellement du flot de contrôle spécifié au niveau du composant de contrôle et du cas de déploiement. Le flot de contrôle spécifié permettra de déterminer les connecteurs qui seront réellement mis en œuvre. Les voies de communication qui ne seront pas utilisées seront supprimées et les voies imposées par le flot de contrôle seront établies.

A titre d'exemple, lorsque le cas de déploiement est FULLY_CONTROLLED_FLOW et que le flot de contrôle spécifié au niveau du contrôleur est un flot synchrone adressant uniquement et explicitement les ports principaux de toutes les instances de composant de la partie opérative (pas de concurrence dans le flot de contrôle), alors l'architecture produite par la phase de normalisation est une architecture dans laquelle toutes les connexions inter composant seront inhibées. De plus, si les connecteurs entre contrôleur et ports principaux de composant n'ont pas été spécifiés, ces derniers seront établis lors du processus de production d'une architecture ordinaire.

Le cas HS_COMMUNICATION est spécifique par rapport au cas FULLY_CONTROLLED_FLOW par le fait qu'il impose un flot synchrone adressant uniquement et explicitement les ports principaux de toutes les instances de composant de la partie opérative comme l'indique la figure 4.2. La troisième phase du processus de transformation utilise l'architecture régulière produite par la phase de normalisation (Figure 4.3). Cette architecture est définie exclusivement à base d'interface ou port ordinaire (exemples : interface Java, port ArchJava). L'objectif principal de cette phase est la

production de la vue d'implémentation de l'enveloppe. C'est au niveau de cette phase, que sont installés dans l'enveloppe les adaptateurs pour solutionner le problème d'incompatibilité d'interface et le problème de distance entre ports interconnectés. Lorsque les ports interconnectés sont totalement compatibles (exemple : ports dotés d'opérations ayant exactement la même signature, ports standards tels que FTP, HTTP, SOAP), cette phase est réduite à la production d'une enveloppe dite transparente dont l'objectif principal est l'interception de toute information ou action.

4.7 Les règles de transformation

La phase de normalisation est la plus importante dans le processus de génération de la vue d'implémentation. Elle met en œuvre un ensemble de règles de transformation pour produire une architecture régulière à base d'interfaces ordinaires. Deux types de règles sont mis en œuvre dans cette étape : les règles générales et les règles spécifiques à chaque technologie d'implémentation. Les règles générales indiquent comment passer d'une description de port IASA à une description en termes d'interfaces. Les règles générales ne dépendent d'aucune technologie d'implémentation. La spécification du transporteur d'un DOAP qu'il soit utilisé indépendamment ou dans le contexte d'une action est une de ces règles. Ainsi, à titre d'exemple, un DOAP peut être cité dans plus d'une action. Dans ce cas, l'action citée représente le transporteur de l'information fournie ou requise par le DOAP.

Lorsque le DOAP est interconnecté individuellement, il doit être pourvu de transporteur spécifique représenté par les méthodes get ou set dépendamment de la manière dont le connecteur est tiré entre les DOAP interconnectés. Une étude poussée de ces règles de transformation est présentée dans [5]. Les règles spécifiques indiquent les techniques de génération de l'enveloppe dans la technologie d'implémentation choisie. Dans IASA le composant pur est protégé par l'enveloppe et n'est pas touché dans une opération d'instanciation. Ainsi, à titre d'exemple, si un composant est déployé comme une Servlet, l'enveloppe sera une classe qui étend l'interface HTTPServlet et qui dispose d'une référence sur l'objet représentant le composant instancié. Toutes les informations provenant des requêtes HTTP ou qui seront mises dans une réponse HTTP seront interceptées par l'enveloppe et aiguillées de ou vers l'instance enveloppée. Dans le cas d'un composant EJB3, l'enveloppe sera représentée par une classe dotée d'un ensemble d'annotation Java.

4.7.1 Les règles de transformation dans le cas du codesign matériel/logiciel

Notre objectif principal est de montrer comment une approche Architecture Logicielle pourrait être utilisée pour résoudre le problème de la communication entre un composant matériel et un composant logiciel dans un système de codesign. Nous avons opté pour l'approche IASA, vu que celle-ci offre plus de possibilité et de flexibilité que les autres approches d'architecture logicielle. Toutefois, dans les études de cas que nous présentons dans ce chapitre, nous n'avons pas utilisé toutes les capacités de l'approche IASA, notamment l'orienté aspect et la spécification libre de topologie dans laquelle il est possible d'établir des connecteurs entre point d'accès de ports distincts (nous n'avons pas eu besoin d'utiliser les points d'accès individuellement dans une spécification d'interconnexion).

Pour des raisons de validation de notre approche, nous nous sommes contentés d'une représentation standard de port (les points d'accès ne sont pas manipulés indépendamment). L'architecture que nous utilisons dans le contexte de l'approche IASA est une architecture ordinaire dans laquelle les composants sont instanciés dans une enveloppe qui a la charge de fournir les ports standards d'interaction. Les règles de transformation sont réduites ainsi aux règles spécifiques à la technologie d'implémentation qui est représentée dans notre cas par les composants hardwares, les composants softwares et les composants mixtes hardware/software. Le connecteur utilisé pour interconnecter les composants qui seront déployés en tant que HARD et les composants qui seront déployés en tant que logiciel (tâches, processus, EJB, etc.) est un connecteur standard. Les règles de transformation générales qui concernent la communication entre un composant matériel et un composant logiciel dans un système de codesign, sont fortement influencées par ce processus de communication. Les règles de transformation spécifiques sont celles qui permettent de générer des composants softwares écrits en Java et des composants hardwares écrits en VHDL.

Les règles de transformation mises en œuvre durant la phase de normalisation, qui permettent de générer un composant mixte déployé en tant que HS, à partir d'un composant déployé en tant que HARD et un composant déployé en tant que logiciel, se résument aux points suivants :

1. L'enveloppe est représentée par un composant mixte.
2. Le constructeur de l'enveloppe doit instancier le composant hardware et le composant software qui forment le composant enveloppé.

3. Tous les aspects permettant de transformer un composant hardware et un composant software, en un composant mixte feront partie de l'enveloppe. Ce sera le cas des fichiers entité, VHDL et JAVA.
4. Les connecteurs utilisés entre le composant matériel et le composant logiciel seront basés sur des fichiers XML.
5. L'enveloppe doit connaitre le sens de la communication, c'est-à-dire communication du hardware vers le software ou du software vers le hardware.
6. L'enveloppe du contrôleur doit disposer de l'interface du composant software et de l'entité du composant hardware.

Dans ce qui suit, nous allons présenter la description VHDL du composant Add4 de la figure 4.2, l'entité, l'interface du composant Cpro3 et sa description JAVA.
La description VHDL du composant Add4 est la suivante :

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity Add4 is
  port (
    a, b : in  unsigned(3 downto 0);
    s    : out unsigned(3 downto 0)
    );
end entity;
architecture arc of Add4 is
  signal resultat : unsigned(4 downto 0);
begin
  resultat <= ('0' & a) + ('0' & b);
  s        <= resultat(3 downto 0);
  cout     <= resultat(4);
end arc;
```

L'entité du composant Add4 est la suivante :

```
<interfaceDefinition version="1.2" language="vhdl" kind="component"
name="Add4">
      <useClause path="ieee.NUMERIC_STD.all" />
      <useClause path="ieee.std_logic_1164.all" />
       <portList>
          <port name="a" direction="in" type="UNSIGNED (3 downto 0)"
kind="array" library="work" package="Add4" packedDimension="(3 downto
0)" />
        <port name="b" direction="in" type="UNSIGNED (3 downto 0)"
kind="array" library="work" package="Add4" packedDimension="(3 downto
0)" />
          <port name="s" direction="out" type="UNSIGNED (3 downto 0)"
kind="array" library="work" package="Add4" packedDimension="(3 downto
0)" />
  </portList>
</interfaceDefinition>
```

L'interface Java du composant Cpro3 est la suivante :

```
package iasa.component.soft;
import iasa.port.*;
public interface SoftCpro3Port extends IASASoftSPort{
    public float Cpro3(int a,int b,int c);
}
```

La description Java (une partie) du composant Cpro3 est la suivante :

```
package iasa.component.soft;
public class SoftCpro3C implements SoftCpro3Port{
      public float Cpro3(int a,int b, int c){
            return Processor3 (a,b,c);
      // Processor3: l'ensemble des instructions du processeur
Cpro3
      }
```

4.8 Production du contrôleur de la communication

La production du contrôleur de la communication peut se faire par l'une des deux méthodes suivantes :

- Le contrôleur est représenté par un composant composite IASA pouvant être déployé en tant que HS. Dans ce cas, il doit être conforme au modèle spécifique représentant les composants mixtes.
- Le contrôleur est représenté par un POJO généré à partir de la description du flot de contrôle spécifiée pour le contrôleur.

Dans notre cas, nous avons opté pour la deuxième solution, car pour l'instant elle est amplement suffisante pour illustrer notre objectif qui est celui de l'exploitation d'une approche Architecture Logicielle pour la communication dans le codesign.

4.9 Conclusion

Nous avons présenté, dans ce chapitre, les concepts développés dans l'approche IASA afin de prendre en charge un processus de communication entre un composant matériel et un composant logiciel d'un système de codesign matériel/logiciel.

La complexité de la communication dans un système de codesign, nous a amenés à définir un modèle spécifique pour la représentation des composants mixtes et pour lequel les cas de déploiement introduits pourraient être appliqués. Toutefois, nous n'avons pas exploité toutes les capacités de l'approche IASA, notamment l'orienté aspects.

Dans le chapitre suivant, nous illustrerons les concepts présentés dans ce chapitre à travers un outil que nous avons réalisé pour traiter spécifiquement le problème de la communication dans un système de codesign matériel/logiciel et que nous avons appelé IASASTUDIO. Cet outil a été développé dans le but de valider l'approche proposée.

CHAPITRE 5
IASASTUDIO : UN OUTIL POUR LA VALIDATION DE LA COMMUNICATION DANS UN SYSTEME DE CODESIGN SELON L'APPROCHE IASA

5.1 Introduction

Ce chapitre est consacré à la mise en œuvre de l'approche IASA dans le contexte de la résolution du problème de la communication entre un composant hardware et un composant software d'un système de codesign matériel/logiciel. Dans le but de valider l'approche proposée par des études de cas, nous avons réalisé un environnement de développement d'applications selon l'approche IASA et nous l'avons appelé IASASTUDIO.

Dans ce qui suit, nous allons présenter l'environnement de développement IASASTUDIO, nous commencerons tout d'abord par introduire l'architecture générale de cet outil, ensuite nous nous intéresserons aux aspects conception et réalisation. Enfin, dans le contexte de la validation de notre approche, nous terminerons par l'utilisation de l'environnement IASASTUDIO dans une étude de cas sur la communication entre un composant hardware et un composant software qui sont réalisés dans des technologies d'implémentation différentes.

5.2 Présentation de l'environnement IASASTUDIO

IASASTUDIO est l'environnement de développement qui génère des composants composites selon l'approche IASA.

La version actuelle de l'environnement IASASTUDIO permet d'assurer la communication entre un composant hardware implémenté en VHDL et un composant software implémenté en JAVA. Le composant hardware et le composant software forment ensemble un composant mixte composite (Figure 5.1).

IASASTUDIO permet de concevoir les applications à un haut niveau d'abstraction, et génère automatiquement le code x3ADL.

A chaque fois qu'un événement se produit (création d'un composant, connexion entre composants, communication, etc.), le code correspondant s'ajoute automatiquement. Le résultat final est enregistré dans un fichier x3ADL décrivant le projet.

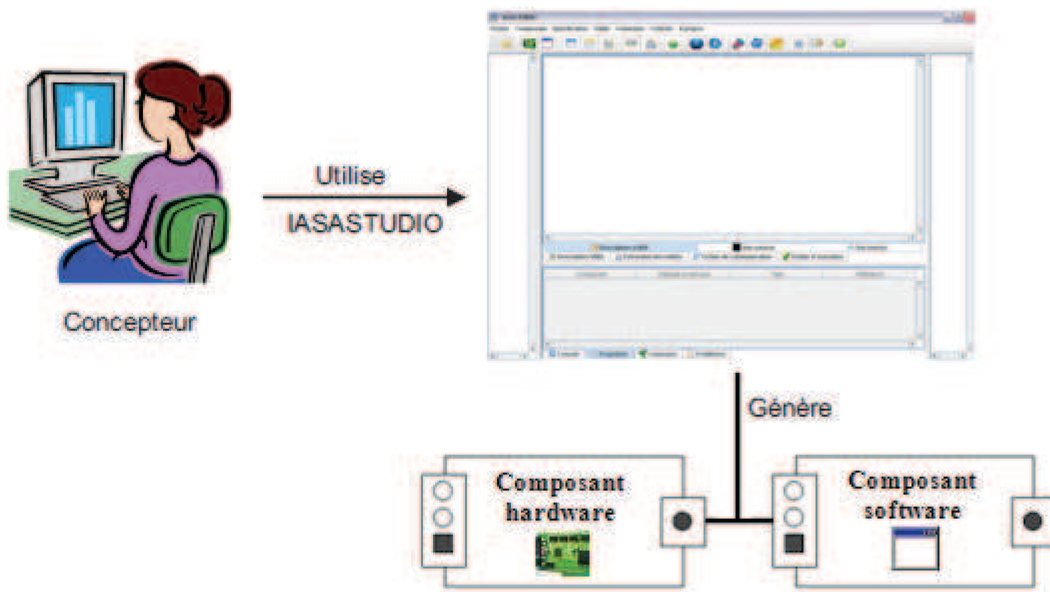


Figure 5.1 : Présentation de l'environnement IASASTUDIO

5.2.1 Conception de l'outil IASASTUDIO

Pour l'implémentation de l'outil IASASTUDIO, nous avons utilisé le langage JAVA. Ce dernier ne permet pas, tout seul, de prendre en charge des caractéristiques essentielles pour les systèmes matériels. Cela, nous a obligés à utiliser un langage de description de matériel qui est le langage VHDL[1] à travers le plugin Sigasi HDT[2] (Figure 5.2) pour la gestion des composants hardwares.

[1] http://www.vhdl.org/
[2] http://www.sigasi.com/

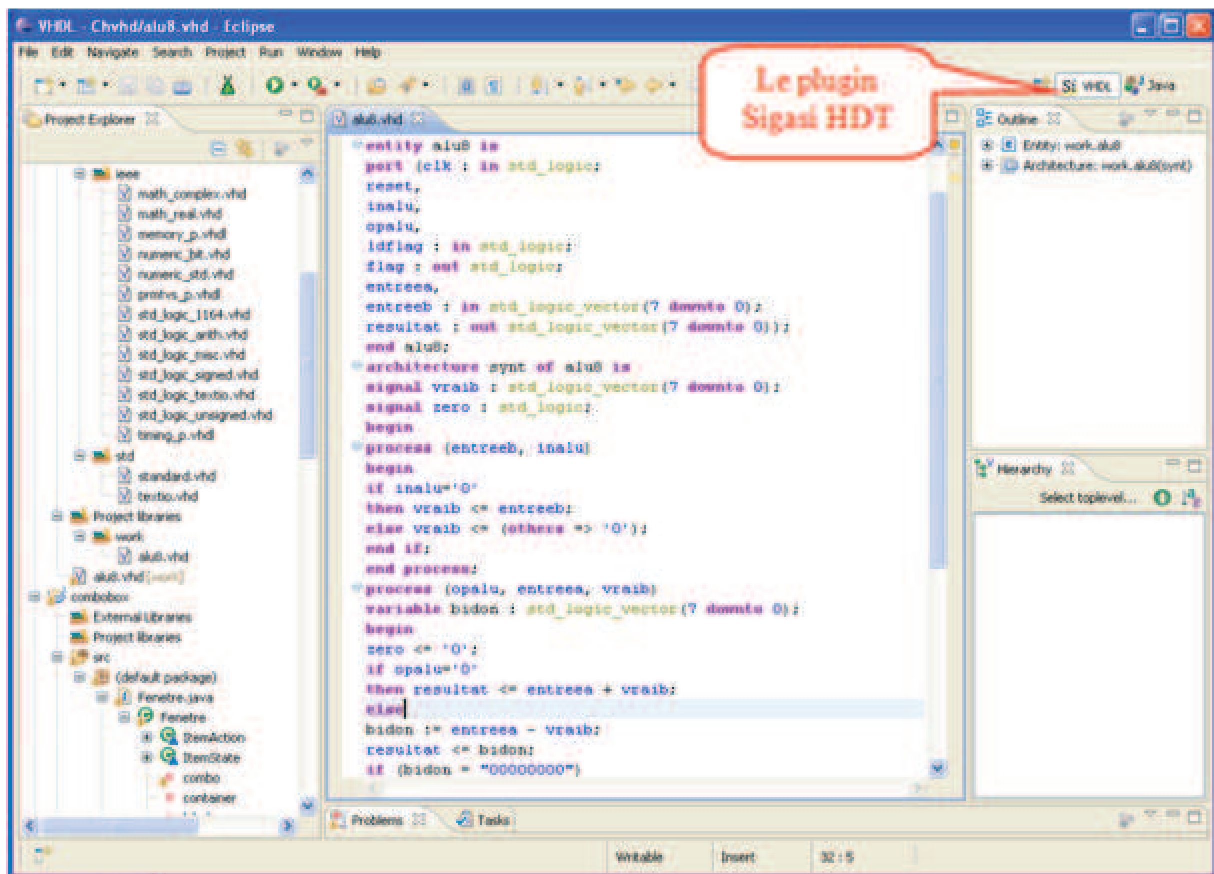


Figure 5.2 : Interface du plugin Eclipse Sigasi HDT

- **VHDL :** (**V**ery High Speed Integrated Circuit, **H**ardware **D**escription **L**anguage) a été conçu pour le département de la défense des USA (DoD) pour décrire les circuits complexes. Il représente le langage que tous les fournisseurs du DoD doivent adopter. Le langage VHDL est devenu un standard de l'IEEE[3] depuis 1987 sous la dénomination IEEE Std. 1076-1987 (VHDL-87). Il est sujet à révision tous les cinq ans. VHDL permet la description des aspects les plus importants d'un système matériel, à savoir son comportement, sa structure et ses caractéristiques temporelles. Le comportement définit les fonctions que le système devra remplir. La structure définit l'organisation du système en une hiérarchie de composants. Les caractéristiques temporelles définissent des contraintes sur le comportement du système.
- **Sigasi HDT :** Sigasi HDT est un environnement de développement avancé pour les concepteurs qui utilisent le VHDL. Toutefois, Sigasi HDT nécessite un environnement

[3] http://standards.ieee.org/

pour la compilation et la simulation. Pour faire face à ce problème, nous avons utilisé le simulateur Modelsim[4].

- **Modelsim :** Modelsim est un simulateur de circuits développé par Mentor Graphics. Modelsim supporte plusieurs langages (VHDL, Verilog, SystemVerilog, SystemC) (Figure 5.3).

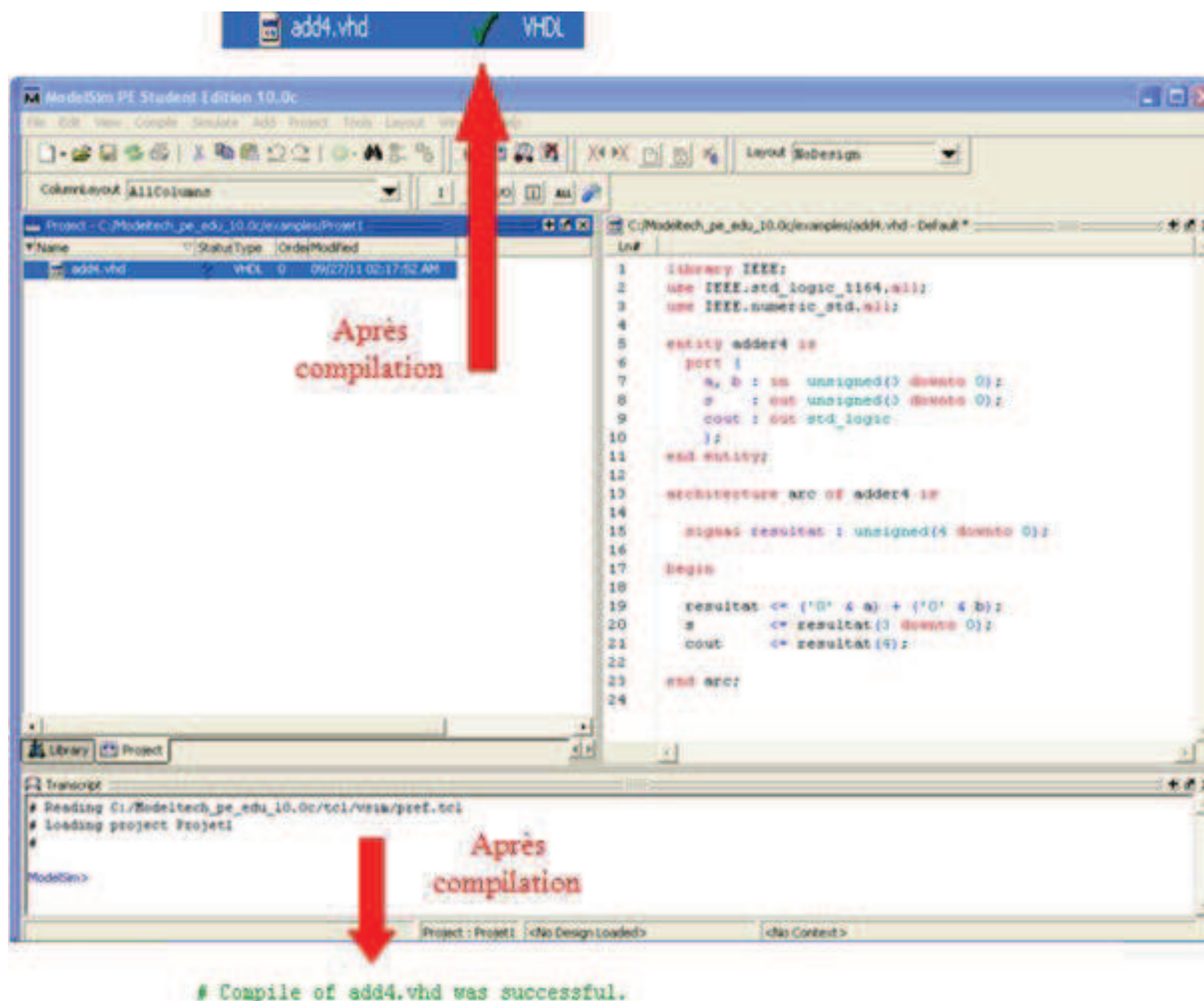


Figure 5.3 : Le simulateur Modelsim

5.2.2 Représentation des composants composites dans IASASTUDIO

Dans IASASTUDIO, un projet est un composant composite. La communication entre un composant hardware et un composant software est réalisée par un composant composite. Un composant composite représente un composant mixte matériel/logiciel composite. Ce

---
[4] http://model.com/

dernier est conforme au modèle IASA représentant les composants de codesign matériel/logiciel.

La vue externe du modèle IASA représentant un composant de codesign matériel/logiciel est caractérisée par les éléments suivants (Figure 5.4) :

- ✓ Un port représentant le service. Il est constitué de points d'accès aux données ayant le sens IN.
- ✓ Un port de données représentant le résultat. Il contient un point d'accès aux données ayant le sens OUT. Celui-ci retourne la réponse du composant.

Figure 5.4 : Architecture d'un composant de codesign composite dans IASASTUDIO

5.2.3 Interface graphique de l'outil IASASTUDIO

La fenêtre principale de l'outil IASASTUDIO (Figure 5.5) est composée de plusieurs zones ayant la charge de faciliter la spécification d'une architecture logicielle. Les zones principales sont :

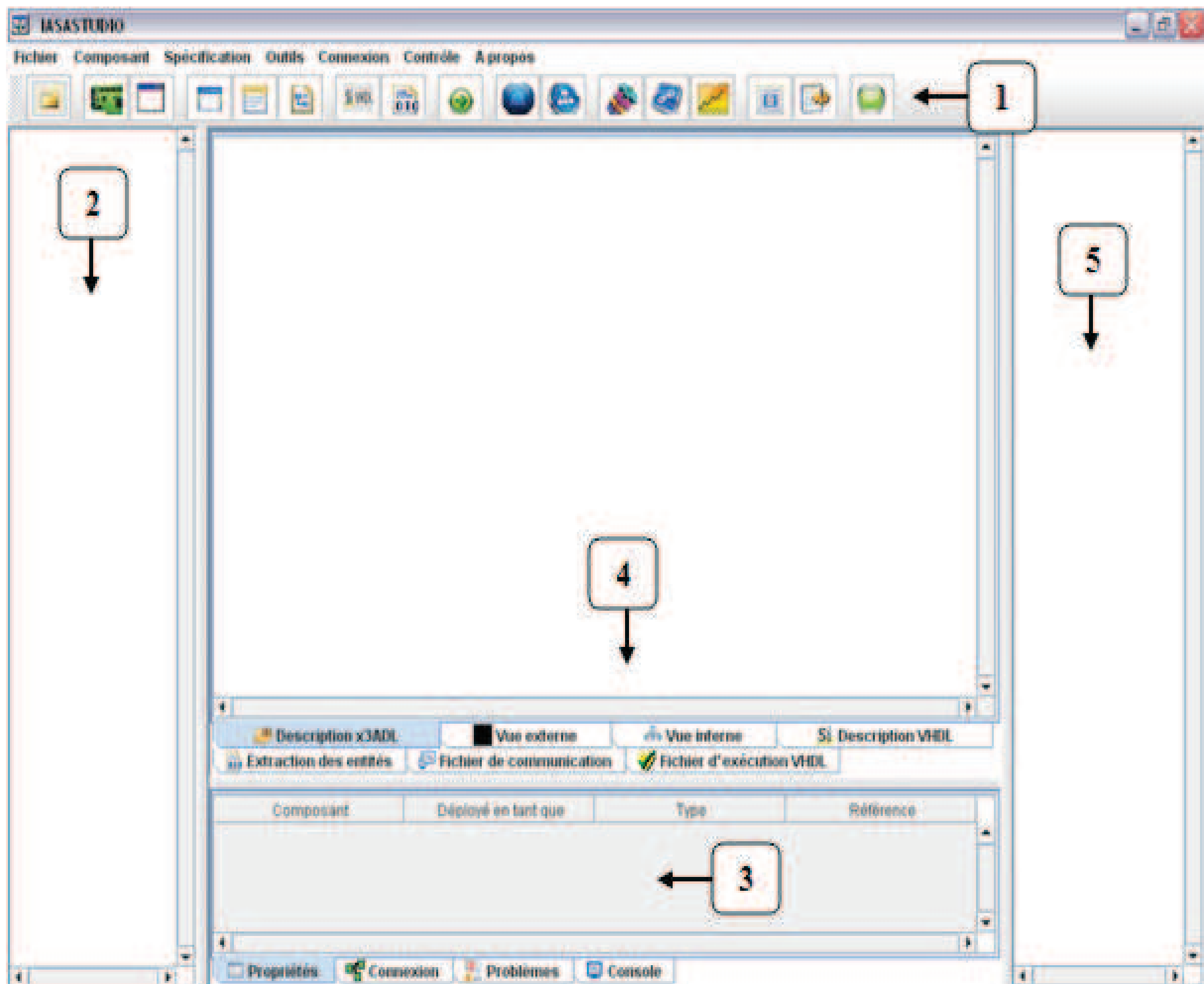


Figure 5.5 : La fenêtre principale de l'outil IASASTUDIO

- **Zone 1 :** représente la barre d'outils de l'outil IASASTUDIO (Figure 5.6), elle est constituée de l'ensemble des boutons suivant :
  - ✓ Bouton 1: pour la création d'un nouveau projet.
  - ✓ Bouton 2: permet d'importer un composant hardware de la bibliothèque des composants matériels.
  - ✓ Bouton 3: permet d'importer un composant software de la bibliothèque des composants logiciels.
  - ✓ Bouton 4: permet de donner la spécification matérielle.
  - ✓ Bouton 5: permet de donner la spécification logicielle.
  - ✓ Bouton 6: permet de donner la spécification mixte.
  - ✓ Bouton 7: permet de donner la description VHDL du composant matériel.
  - ✓ Bouton 8: permet d'extraire l'entité à partir de la description VHDL.

- ✓ Bouton 9: pour déterminer le sens de la communication.
- ✓ Bouton 10: pour l'ajout des ports externes du projet.
- ✓ Bouton 11: permet de réaliser la connexion interne.
- ✓ Bouton 12: pour établir les paramètres de la communication.
- ✓ Bouton 13: pour réaliser la communication.
- ✓ Bouton 14: pour le contrôle.
- ✓ Bouton 15: pour importer le fichier de communication.
- ✓ Bouton 16: pour importer le fichier d'exécution VHDL.
- ✓ Bouton 17 : pour fermer le projet.

Figure 5.6 : La barre d'outils de l'outil IASASTUDIO

- **Zone 2 :** représente l'arbre du projet. L'arbre affiche les noms des composants qui forment le projet (composant composite).
- **Zone 3 :** cette zone contient l'ensemble des onglets suivants :
  - ✓ Propriétés : affiche les noms des composants, l'état de déploiement, le type et la référence.
  - ✓ Connexion : affiche la connexion interne établie entre le composant hardware et le composant software.
  - ✓ Problèmes : signale les problèmes qui peuvent exister dans le projet, comme par exemple lors de l'établissement des paramètres de la communication (choix d'un codage qui ne peut pas être pris en charge par le composant hardware).
  - ✓ Console : affiche le résultat d'exécution pas à pas du projet.
- **Zone 4 :** cette zone est composée de l'ensemble des onglets suivants :
  - ✓ Description x3ADL : permet de visualiser la spécification matérielle, la spécification logicielle et la spécification mixte (spécification du projet).
  - ✓ Vue externe : affiche la vue externe du projet (composant composite).

- ✓ Vue interne : affiche les composants qui forment la partie opérative du projet, ainsi que la liaison interne.
- ✓ Description VHDL : permet de visualiser la description VHDL du composant matériel.
- ✓ Extraction des entités : permet d'extraire l'entité à partir de la description VHDL et de la visualiser.
- ✓ Fichier de communication : permet de visualiser le fichier XML de la communication.
- ✓ Fichier d'exécution VHDL : permet de visualiser le fichier VHDL de l'exécution.

- **Zone 5 :** représente la partie contrôle, cette partie contient le contrôleur du composant en cours de réalisation dans le contexte du projet ouvert.

5.3 Mise en œuvre de l'outil IASASTUDIO pour la réalisation d'un composant composite

Dans ce qui suit, nous allons montrer comment réaliser un composant composite en utilisant IASASTUDIO. Tout d'abord, nous présenterons les bibliothèques des composants hardwares et softwares que l'outil IASASTUDIO met à disposition de l'utilisateur. Ensuite, nous détaillerons les étapes permettant de réaliser la communication entre un composant matériel et un composant logiciel. En dernier, nous donnerons les différents fichiers qui ont été construits au fur et à mesure de l'avancée du projet.

5.3.1 Création d'un nouveau composant composite dans IASASTUDIO

La création d'un nouveau composant composite dans IASASTUDIO correspond à l'ouverture d'un projet dont le nom est celui du composant composite. Pour illustrer notre travail, nous allons créer un nouveau projet et nous lui donnerons le nom Codesign. La figure 5.7 montre la fenêtre affichée par IASASTUDIO suite à la demande d'ouverture d'un nouveau composant composite.

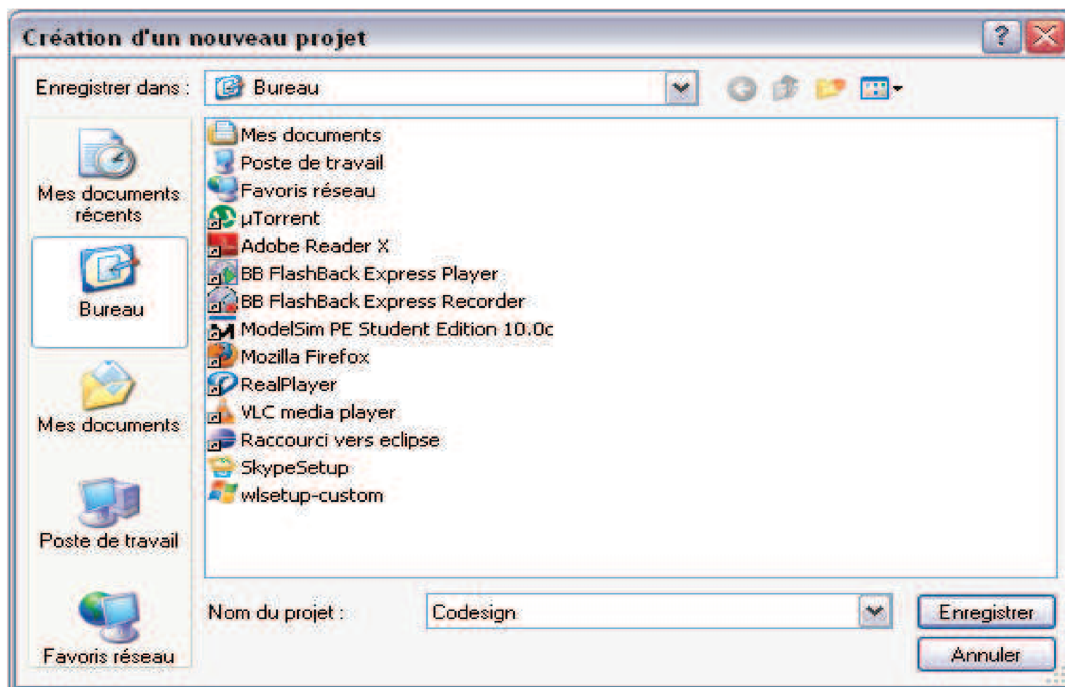


Figure 5.7 : Création d'un nouveau projet dans IASASTUDIO

La création d'un nouveau projet déclenche l'instanciation des éléments suivants (Figure 5.8) :

- ✓ L'arbre du projet dont le contenu est affiché dans la zone gauche.
- ✓ Un composant de contrôle est instancié dans la partie contrôle du composant composite.
- ✓ Le fichier x3ADL décrivant le nouveau composant composite.

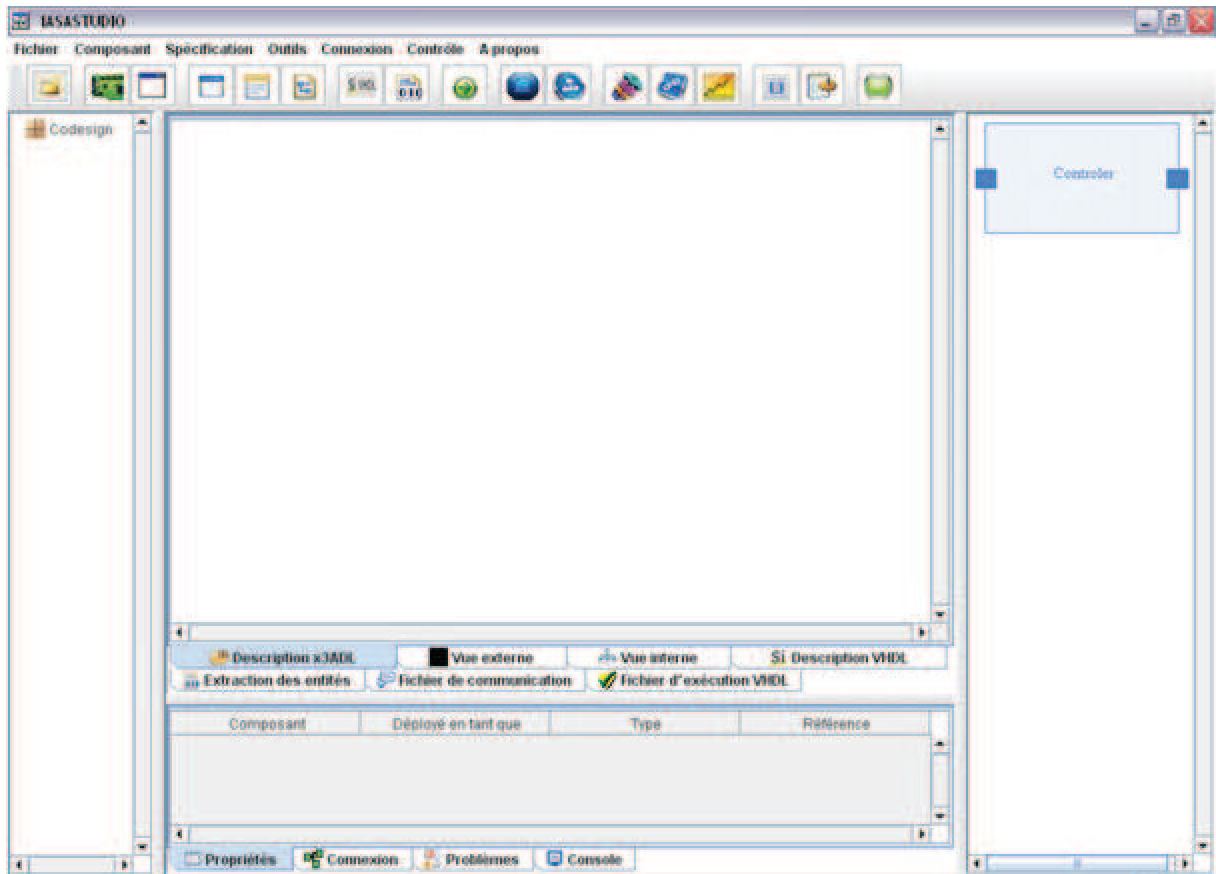


Figure 5.8 : Conception du composant Codesign dans IASASTUDIO

5.3.2 Définition de la vue interne du composant Codesign

La définition de la vue interne d'un composant composite consiste à définir ses composants primitifs. Dans notre cas, un composant composite est composé d'un composant logiciel primitif et d'un composant matériel primitif. IASASTUDIO est doté d'une bibliothèque de composants matériels prédéfinis et d'une bibliothèque de composants logiciels prédéfinis. Chaque composant primitif possède une vue d'implémentation qui permet son déploiement selon le modèle IASA.

5.3.2.1 La bibliothèque des composants logiciels

La bibliothèque des composants logiciels est composée d'un ensemble de processeurs généraux, chacun avec son propre jeu d'instructions. Ils sont déployés en tant que SOFT. Ces composants sont : cpro1, cpro2, cpro3, cpro4, cpro5 et cpro6 (Figure 5.9).

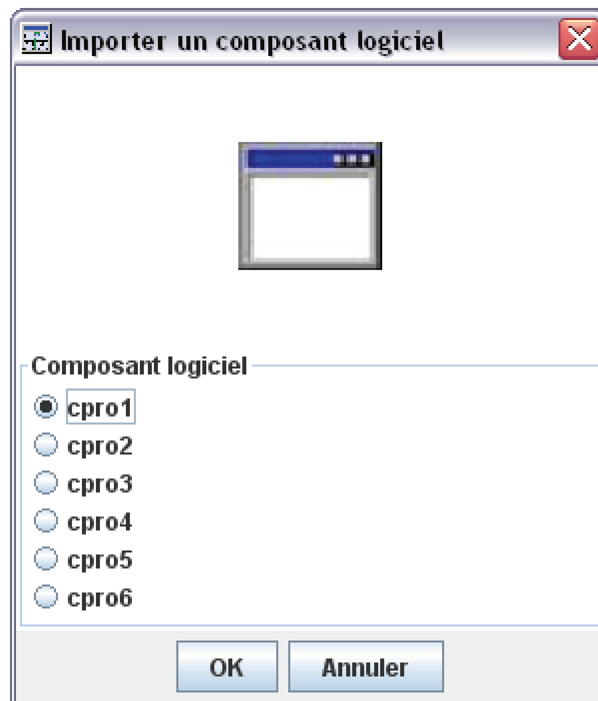


Figure 5.9 : Importation d'un composant logiciel dans IASASTUDIO

Une fois le composant logiciel importé, sa spécification x3ADL est instanciée et le fichier x3ADL correspondant est enregistré dans le projet Codesign. Ainsi, la spécification logicielle peut être alors visualisée dans l'onglet Description x3ADL (Figure 5.10).

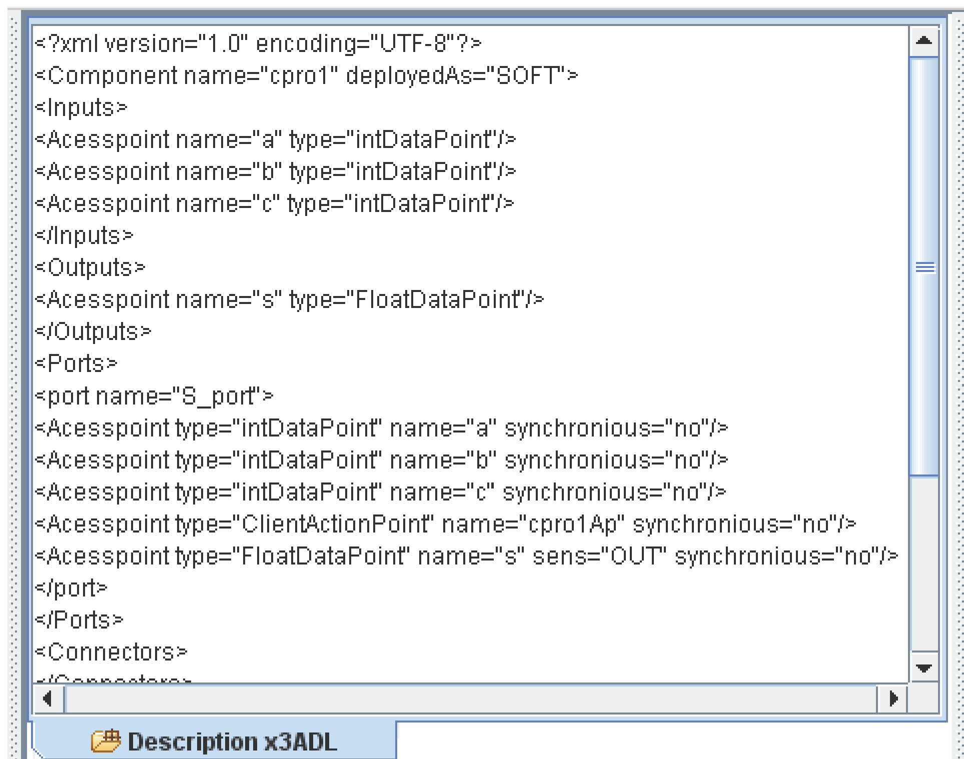


Figure 5.10 : Description x3ADL du composant logiciel

5.3.2.2 La bibliothèque des composants matériels

La bibliothèque des composants matériels est composée de l’ensemble des composants suivants (Figure 5.11):

- ✓ add-4: un additionneur 4 bits.
- ✓ alu-8 : une unité arithmétique et logique sur 8 bits.
- ✓ reg-d : un registre à décalage sur 8 bits.
- ✓ com-8: un compteur synchrone sur 8 bits, avec reset asynchrone actif à l'état bas.
- ✓ and-3 : une porte combinatoire AND à trois entrées.
- ✓ bas-d: une bascule D.
- ✓ add-1: un additionneur complet 1 bit.

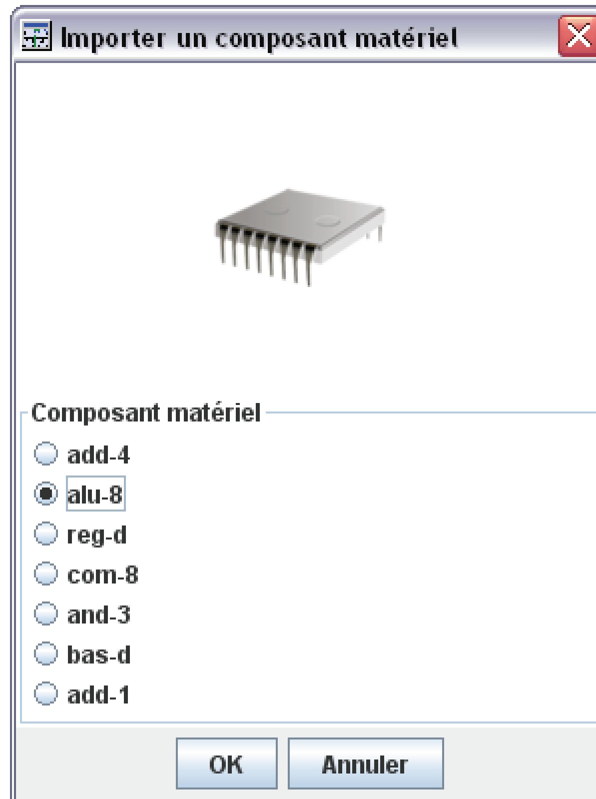
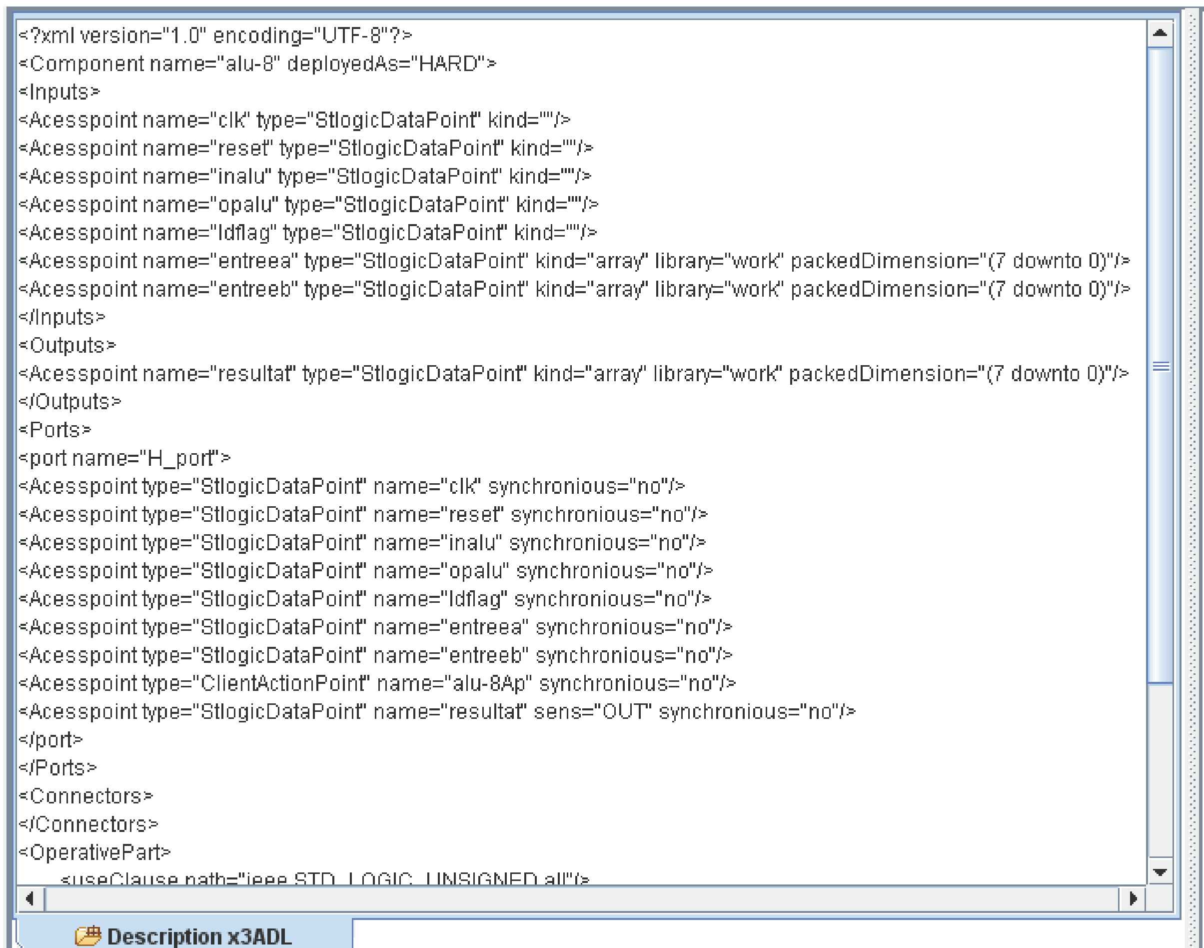


Figure 5.11 : Importation d'un composant matériel dans IASASTUDIO

Tous ces composants sont déployés en tant que HARD. L'importation du composant matériel déclenche l'instanciation des éléments suivants :

- ✓ Sa spécification x3ADL et le fichier x3ADL correspondant est enregistré dans le projet Codesign. La spécification matérielle peut être visualisée dans l'onglet Description x3ADL (Figure 5.12).
- ✓ Sa description VHDL et le fichier VHDL correspondant est enregistré dans le projet Codesign. La description VHDL peut être visualisée dans l'onglet Description VHDL (Figure 5.13).
- ✓ Extraction de l'entité à partir de la description VHDL et le fichier XML correspondant est enregistré dans le projet Codesign. L'entité peut être visualisée dans l'onglet Extraction des entités (Figure 5.14).



```
<?xml version="1.0" encoding="UTF-8"?>
<Component name="alu-8" deployedAs="HARD">
<Inputs>
<Acesspoint name="clk" type="StlogicDataPoint" kind=""/>
<Acesspoint name="reset" type="StlogicDataPoint" kind=""/>
<Acesspoint name="inalu" type="StlogicDataPoint" kind=""/>
<Acesspoint name="opalu" type="StlogicDataPoint" kind=""/>
<Acesspoint name="ldflag" type="StlogicDataPoint" kind=""/>
<Acesspoint name="entreea" type="StlogicDataPoint" kind="array" library="work" packedDimension="(7 downto 0)"/>
<Acesspoint name="entreeb" type="StlogicDataPoint" kind="array" library="work" packedDimension="(7 downto 0)"/>
</Inputs>
<Outputs>
<Acesspoint name="resultat" type="StlogicDataPoint" kind="array" library="work" packedDimension="(7 downto 0)"/>
</Outputs>
<Ports>
<port name="H_port">
<Acesspoint type="StlogicDataPoint" name="clk" synchronious="no"/>
<Acesspoint type="StlogicDataPoint" name="reset" synchronious="no"/>
<Acesspoint type="StlogicDataPoint" name="inalu" synchronious="no"/>
<Acesspoint type="StlogicDataPoint" name="opalu" synchronious="no"/>
<Acesspoint type="StlogicDataPoint" name="ldflag" synchronious="no"/>
<Acesspoint type="StlogicDataPoint" name="entreea" synchronious="no"/>
<Acesspoint type="StlogicDataPoint" name="entreeb" synchronious="no"/>
<Acesspoint type="ClientActionPoint" name="alu-8Ap" synchronious="no"/>
<Acesspoint type="StlogicDataPoint" name="resultat" sens="OUT" synchronious="no"/>
</port>
</Ports>
<Connectors>
</Connectors>
<OperativePart>
    <useClause path="ieee.STD_LOGIC_UNSIGNED.all"/>
```


Figure 5.12 : Description x3ADL du composant matériel

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity alu8 is
port (clk : in std_logic;
reset,
inalu,
opalu,
ldflag : in std_logic;
flag : out std_logic;
entreea,
entreeb : in std_logic_vector(7 downto 0);
resultat : out std_logic_vector(7 downto 0));
end alu8;
architecture synt of alu8 is
signal vraib : std_logic_vector(7 downto 0);
signal zero : std_logic;
begin
process (entreeb, inalu)
begin
if inalu='0'
then vraib <= entreeb;
else vraib <= (others => '0');
end if;
end process;
process (opalu, entreea, vraib)
variable bidon : std_logic_vector(7 downto 0);
begin
zero <= '0';
if opalu='0'
then resultat <= entreea + vraib;
```

Description x3ADL | Vue externe | Vue interne | Description VHDL

Figure 5.13 : Description VHDL du composant matériel

```
<interfaceDefinition version='1.2' language='vhdl' kind='component' name='alu8'>
 <useClause path='ieee.STD_LOGIC_UNSIGNED.all' />
 <useClause path='ieee.std_logic_1164.all' />
  <portList>

   <port name='clk' direction='in' type='STD_LOGIC' kind='' />
   <port name='reset' direction='in' type='STD_LOGIC' kind='' />
   <port name='inalu' direction='in' type='STD_LOGIC' kind='' />
   <port name='opalu' direction='in' type='STD_LOGIC' kind='' />
   <port name='ldflag' direction='in' type='STD_LOGIC' kind='' />

   <port name='entreea' direction='in' type='std_logic_vector (7 downto 0)' kind='array' library='work' package='al
   <port name='entreeb' direction='in' type='std_logic_vector (7 downto 0)' kind='array' library='work' package='al
   <port name='resultat' direction='out' type='std_logic_vector (7 downto 0)' kind='array' library='work' package='a

  </portList>
</interfaceDefinition>
```

Description x3ADL | Vue externe | Vue interne | Description VHDL
Extraction des entités | Fichier de communication | Fichier d'exécution VHDL

Figure 5.14 : L'entité du composant matériel

La figure 5.15 montre la partie de la fenêtre de conception chargée d'illustrer la hiérarchie de composition du composant composite.

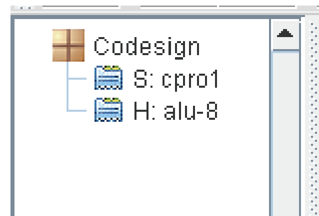


Figure 5.15 : Arborescence du composant Codesign

Après l'importation du composant matériel et du composant logiciel, il faut déterminer le sens de la communication (Figure 5.16).

Figure 5.16 : Détermination du sens de la communication

Le sens de la communication étant du software vers le hardware implique que le composant cpro1 va envoyer des données au composant alu-8. Dans ce cas, le port de service du composant composite Codesign sera le port de service du composant cpro1 et le port de données sera celui du composant alu-8.

5.3.3 Définition de la vue externe du composant Codesign

La vue externe consiste à définir la structure des ports du composant Codesign. Le premier port servant à l'invocation du service est représenté par un point d'accès orienté action (ACTOAP) et de 3 points d'accès orientés donnée (DOAP) a, b et c. Ces points d'accès sont du type intDOAP. Le port de données est représenté par un seul point d'accès ayant le

type StlogicDOAP et dont le nom est result (Figure 5.17). La vue externe complète du composant Codesign est illustrée dans la figure 5.18.

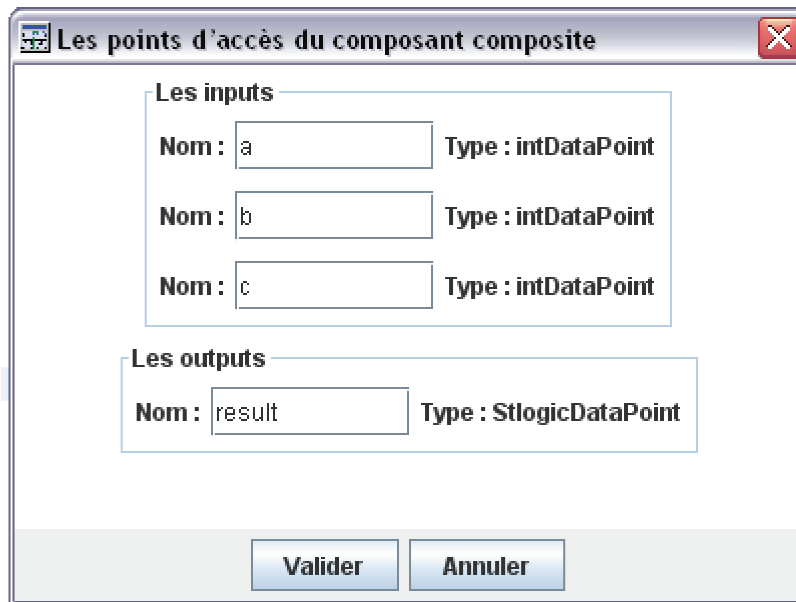


Figure 5.17 : Les points d'accès du composant Codesign

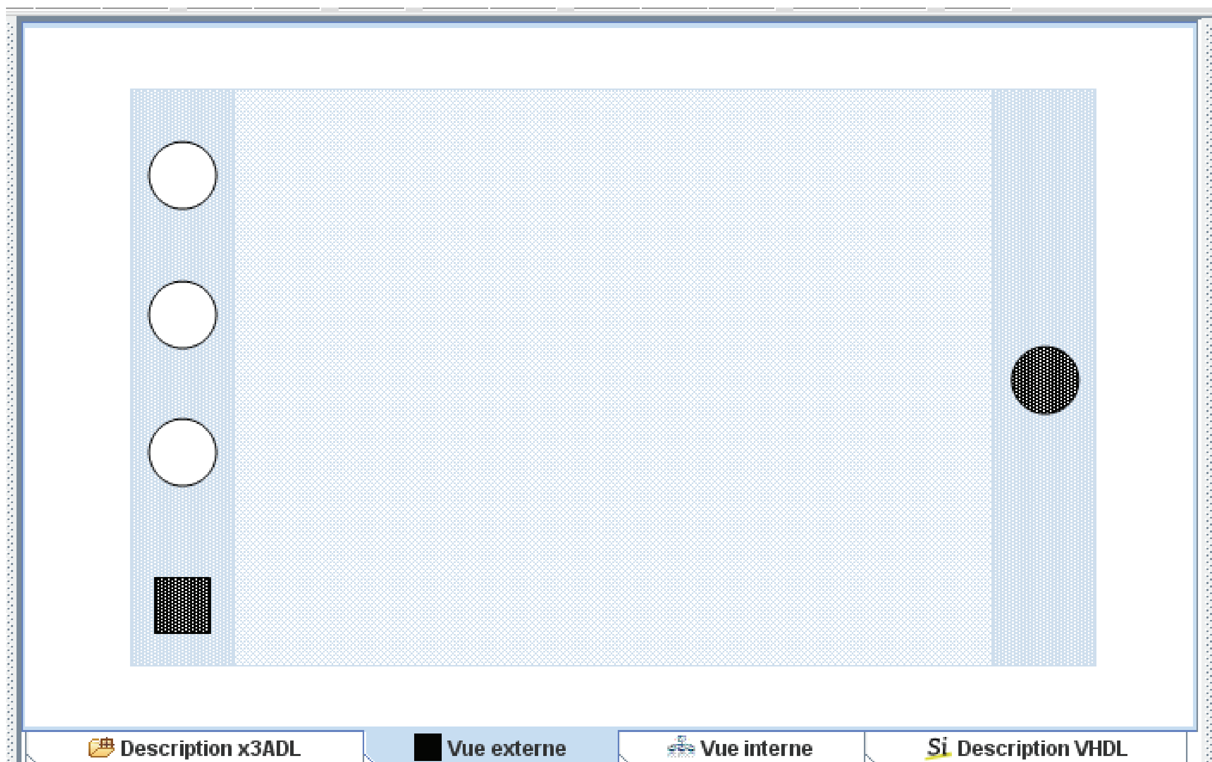


Figure 5.18 : La vue externe du composant Codesign

5.3.4 Réalisation de la connexion interne

Cette opération consiste à indiquer par quel point d'accès du récepteur, l'émetteur va envoyer les données. Dans notre cas, l'émetteur est le composant software cpro1 et le récepteur est le composant hardware alu-8. A la demande de la réalisation de la connexion interne, IASASTUDIO affiche une fenêtre contenant les points d'accès du composant récepteur (Figure 5.19). Ainsi, la vue finale du composant Codesign est illustrée dans la figure 5.20, et ses propriétés sont décrites dans la figure 5.21.

Figure 5.19 : Réalisation de la connexion interne du composant Codesign

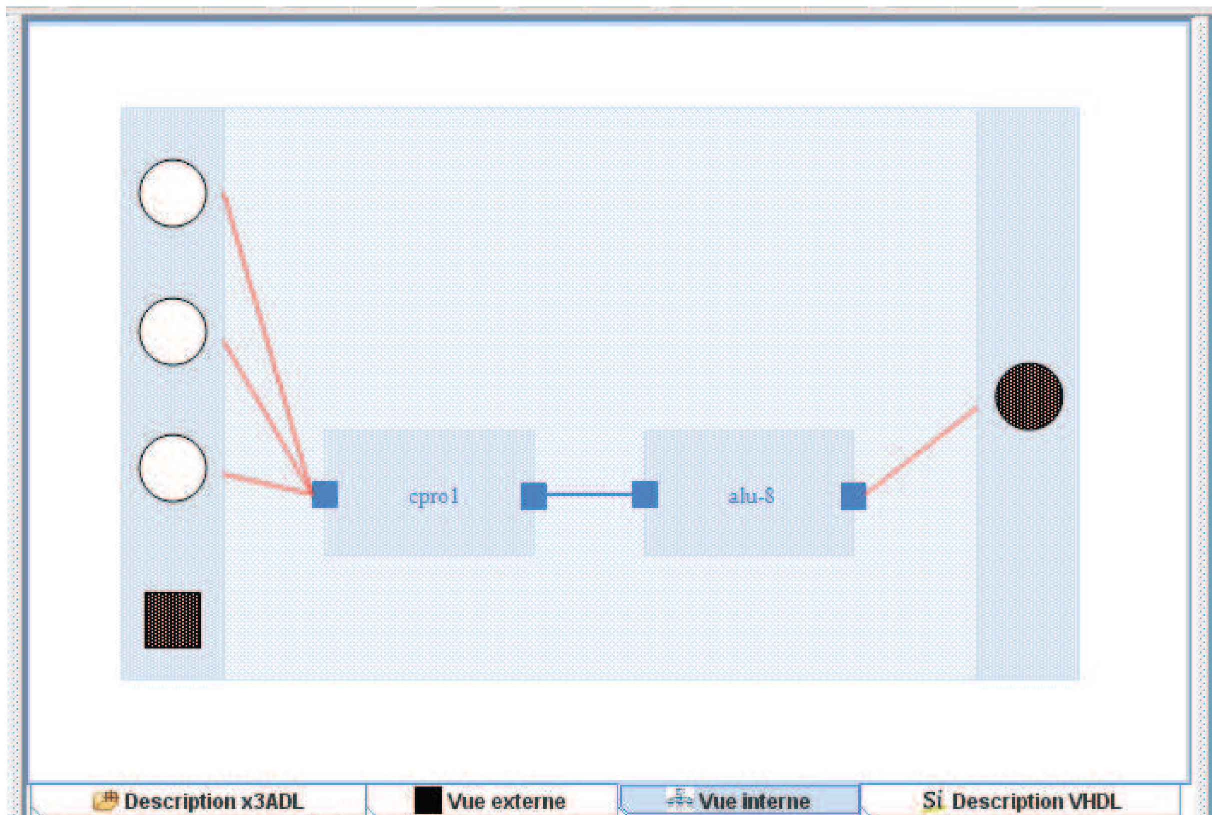


Figure 5.20 : La vue finale du composant Codesign dans IASASTUDIO

Composant	Déployé en tant que	Type	Référence
Codesign	HS	Composite	Communication
cpro1	SOFT	Primitif	Processeur
alu-8	HARD	Primitif	Unité arithmétique et logique

Propriétés | Connexion | Problèmes | Console

Figure 5.21 : Les propriétés du composant Codesign

5.3.5 Réalisation de la communication

La communication entre un composant matériel et un composant logiciel dans IASASTUDIO peut se faire dans les deux sens : du hardware vers le software ou du software vers le hardware.

Etant donné que les composants sont implémentés dans des technologies différentes (JAVA pour le logiciel et VHDL pour le matériel), nous avons choisi d'élaborer la communication à travers des fichiers XML. Dans les deux sens de la communication,

l'émetteur envoie un fichier XML, dans lequel sont spécifiées les données attendues par le récepteur.

Après la réception du fichier XML de la communication, le composant récepteur va lire ce fichier pour extraire les données attendues.

Dans le cas de la communication du hardware vers le software, la donnée extraite sera enregistrée dans la valeur du point d'accès correspondant.

Dans le cas de la communication du software vers le hardware, les données extraites du fichier XML seront enregistrées dans le fichier qui sera soumis à la simulation. De plus, le programme VHDL qui permet de lire le fichier XML sera généré automatiquement et enregistré dans un fichier VHDL appelé READ.

Le fichier XML de la communication contiendra, dans ce cas, en plus des données envoyées par le composant software, des paramètres spécifiques pour la simulation[5]. Ces paramètres peuvent être spécifiés par le concepteur, à la demande de la détermination des paramètres, la fenêtre de la figure 5.22 apparait. Dans le cas où le concepteur ne souhaite pas établir ces paramètres, le système prendra des paramètres définis par défaut dans IASASTUDIO.

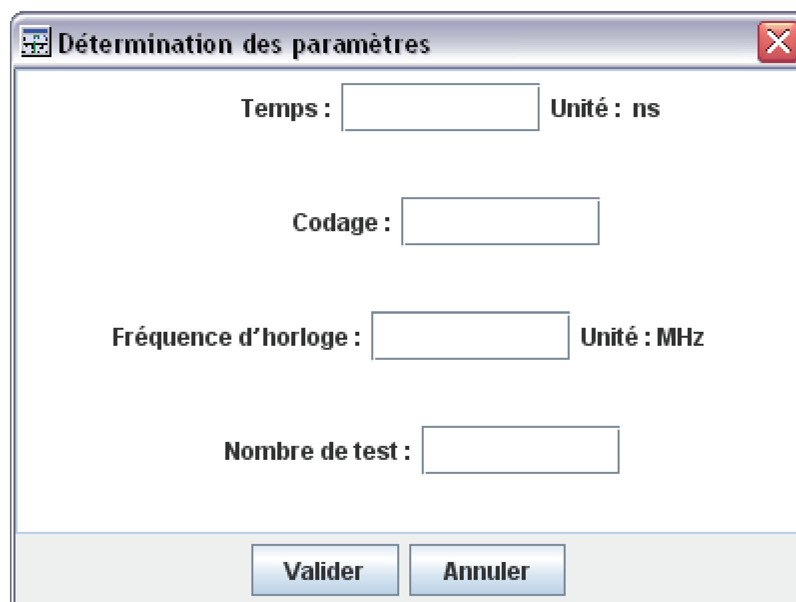


Figure 5.22 : Détermination des paramètres de la simulation

[5] Appliquer des stimuli (également décrits en VHDL) et observer l'évolution des signaux du modèle dans le temps par simulation.

Ainsi, après avoir déterminé les paramètres de la simulation, la communication entre le composant cpro1 et le composant alu-8 peut commencer. Elle s'effectue par un clic sur le bouton exécution. L'exécution pas à pas du composant Codesign est décrite dans l'onglet Console (Figure 5.23). Le fichier de communication XML et le fichier VHDL READ, ainsi que la description x3ADL du composant Codesign peuvent alors être visualisés (Figure 5.24, Figure 5.25, Figure 5.26).

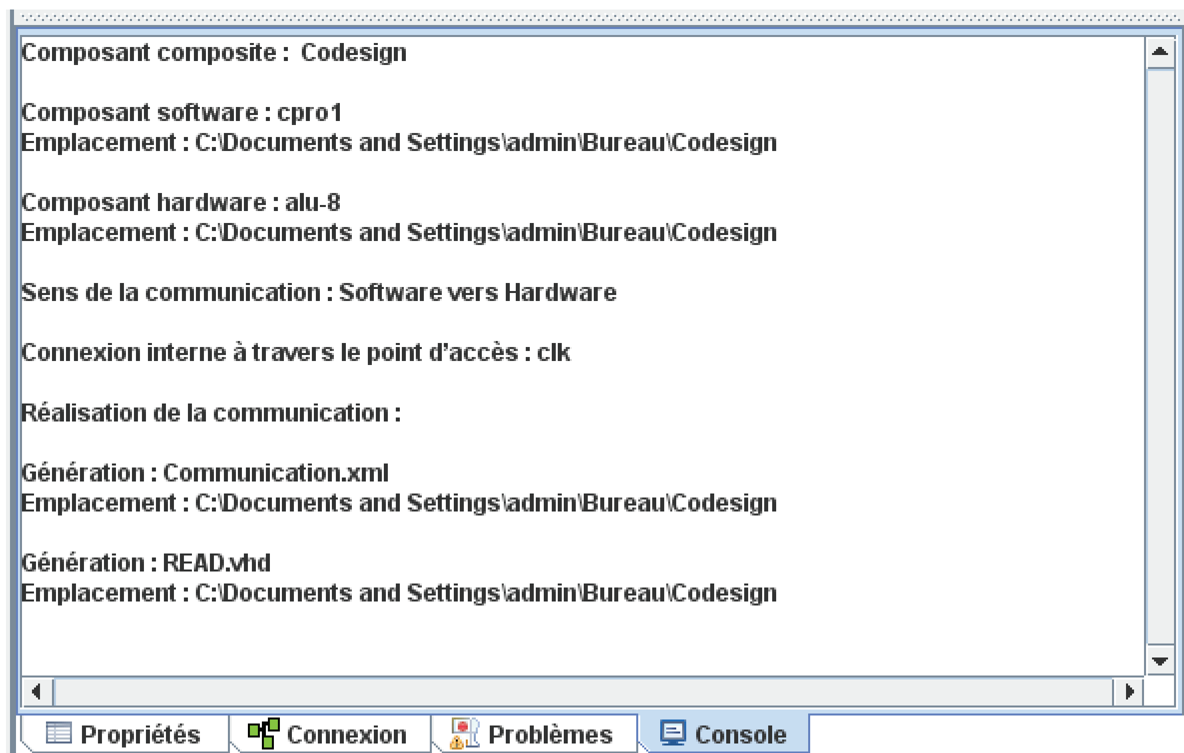


Figure 5.23 : Exécution pas à pas dans IASASTUDIO

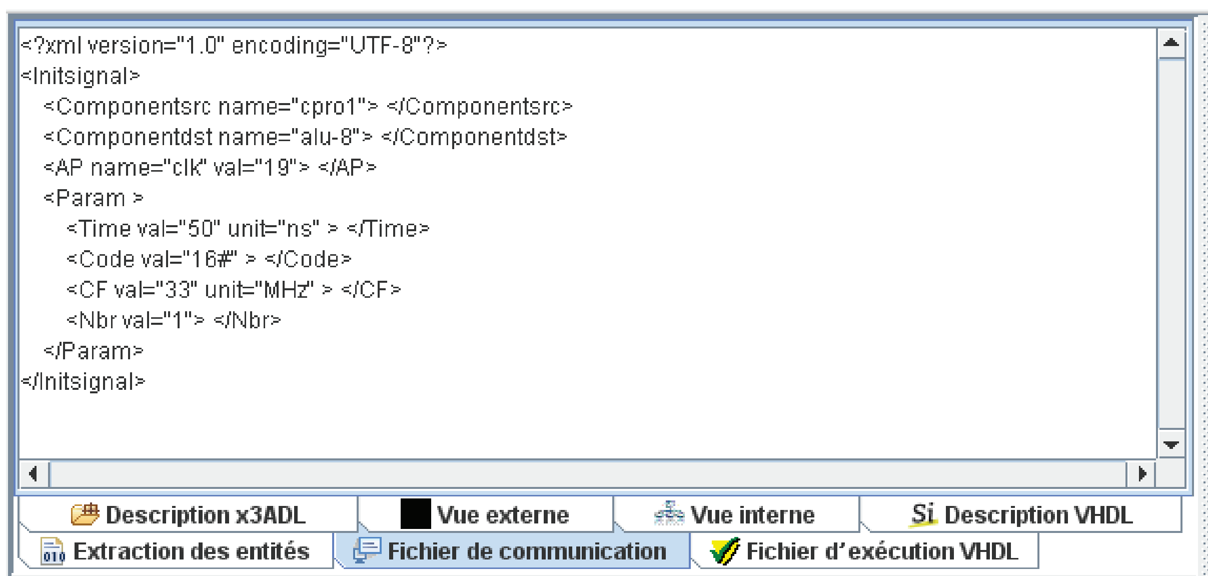


Figure 5.24 : Le fichier de communication

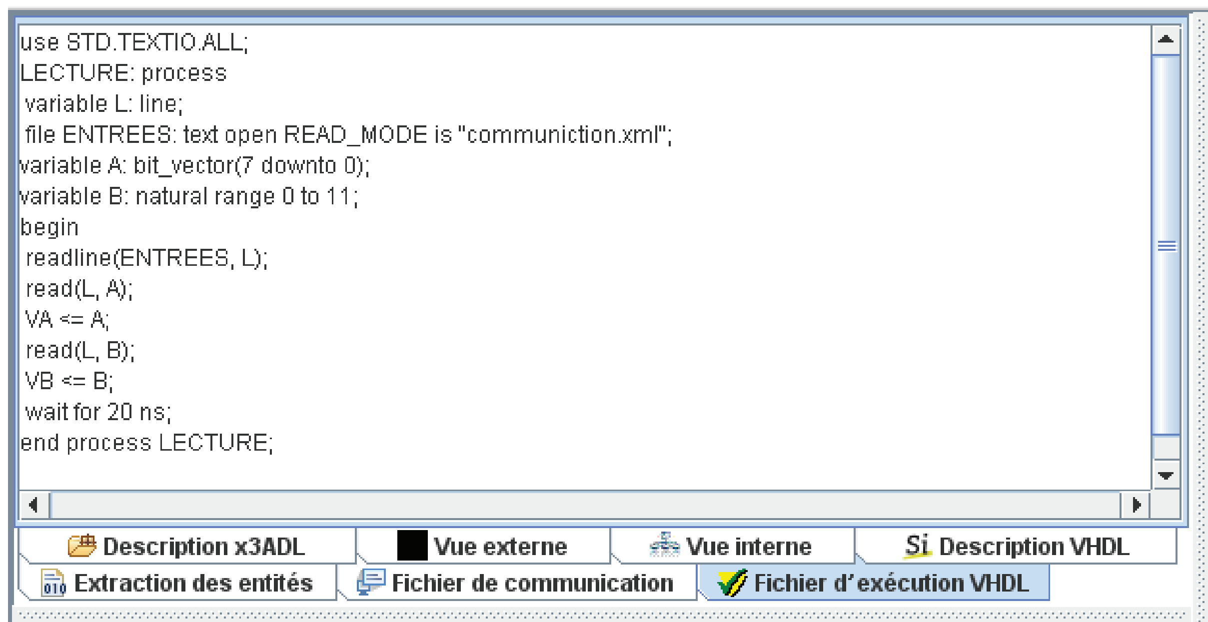


Figure 5.25 : Le fichier VHDL READ

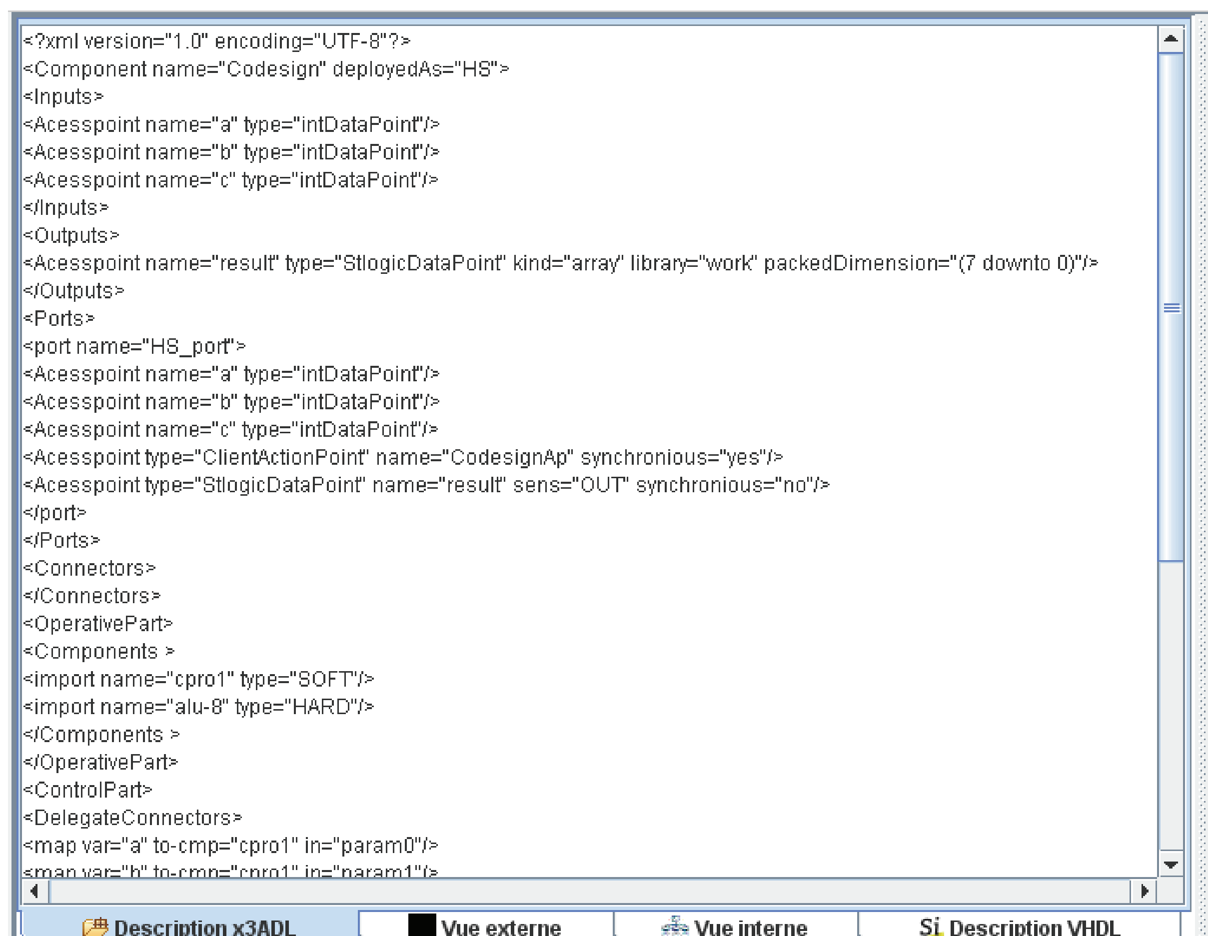


Figure 5.26 : Description x3ADL du composant Codesign

Au fur et à mesure de l'avancement du projet, les fichiers suivants sont générés automatiquement et enregistrés dans l'emplacement choisi pour le composant composite (Figure 5.27) :

- ✓ Codesign.xml : description x3ADL du composant composite Codesign.
- ✓ scpro1.xml : description x3ADL du composant software cpro1.
- ✓ halu-8.xml : description x3ADL du composant hardware alu-8.
- ✓ alu-8.xml : l'entité du composant hardware alu-8.
- ✓ alu-8.vhd : description VHDL du composant hardware alu-8.
- ✓ Communiction.xml : le fichier de communication.
- ✓ READ.vhd : le fichier VHDL READ.

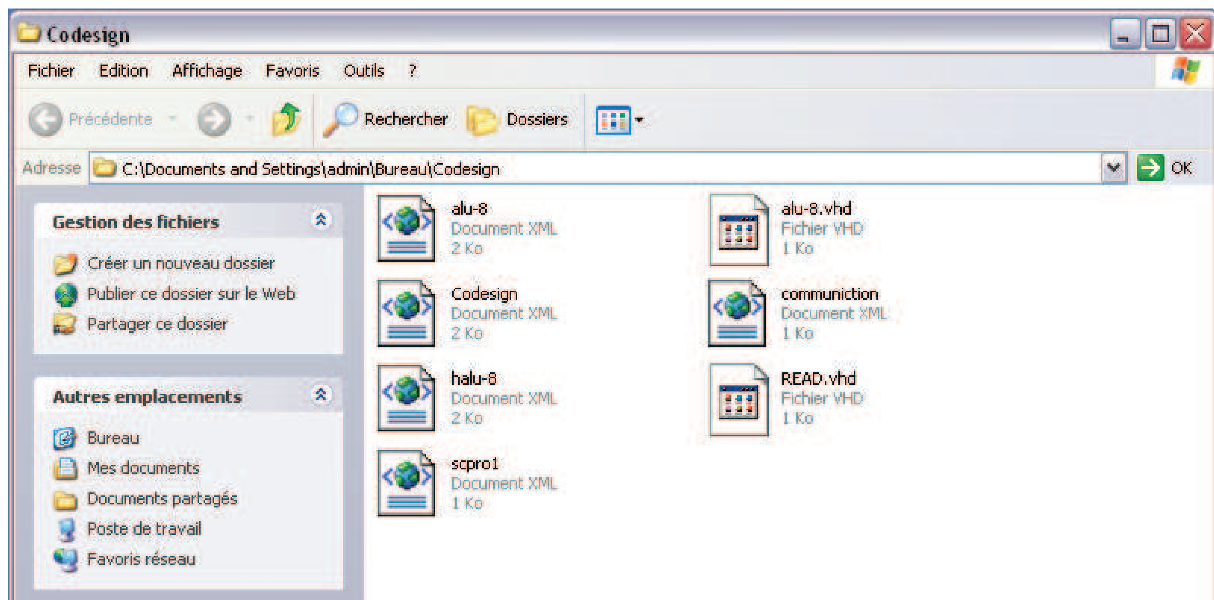


Figure 5.27 : Les fichiers du projet Codesign

5.4 Conclusion

Dans ce chapitre, nous avons présenté l'environnement IASASTUDIO que nous avons développé pour la communication entre un composant matériel et un composant logiciel d'un système de codesign matériel/logiciel selon l'approche intégrée IASA.

IASASTUDIO est doté d'une bibliothèque de composants softwares implémentés en JAVA et d'une bibliothèque de composants hardwares implémentés en VHDL. Notre approche part d'une cospécification de l'application faite en x3ADL, et aboutit à un fichier de

communication en XML, dans lequel sont spécifiées les données de la communication. Ce fichier sera utilisé soit pour l'exécution du composant software, soit pour la simulation du composant hardware, selon le sens de la communication. De plus, tous les fichiers nécessaires à l'implémentation du composant composite sont générés automatiquement et enregistrés dans l'emplacement du projet.

L'avantage principal de notre approche est que le concepteur n'a pas besoin de se soucier des détails de bas niveau qui sont souvent la source d'erreurs principale dans les systèmes de codesign.

CONCLUSION

Dans ce travail, nous avons présenté une approche de l'architecture logicielle pour la résolution du problème de la synthèse des interfaces de communication dans un système de codesign matériel/logiciel. L'approche proposée est basée sur le modèle de composant de l'approche intégrée IASA.

Aujourd'hui, la croissance dans le marché des SoC nécessite des solutions qui garantissent les performances demandées tout en minimisant les coûts de développement. Le hardware/software codesign est la solution pour satisfaire ces exigences.

Le codesign matériel/logiciel a été introduit pour faire face aux problèmes de la conception traditionnelle. Le processus de la conception concurrente repose sur le développement en parallèle du matériel et du logiciel, en retardant leur séparation. Le partitionnement matériel/logiciel du système produit une architecture composée d'une partie matérielle et d'une partie logicielle qui ont besoin de communiquer.

Après l'étude des différentes approches de cosynthèse, nous avons constaté que la conception des interfaces de communication est une tâche difficile à mettre en œuvre, et surtout source d'erreurs car la nature hétérogène des interfaces nécessite des compétences issues des domaines du logiciel et du matériel. Mais le problème majeur de la cosynthèse des interfaces est le niveau d'abstraction de l'application qui n'est pas assez élevé pour prendre en charge tous les aspects de la communication entre le matériel et le logiciel d'une manière efficace.

Afin de surmonter les problèmes de la cosynthèse des interfaces de communication dans le codesign, nous avons proposé dans ce mémoire de ramener le problème de la communication dans un système de codesign à un niveau d'abstraction très élevé qui est le niveau architecture.

Aujourd'hui, la phase architecture dans un processus de conception s'est dotée de concepts, outils et méthodologies qui ont donné naissance à une nouvelle discipline du génie logiciel connue sous le nom d'Architecture Logicielle.

L'approche intégrée IASA (Integrated Approach to Software Architecture) est une approche d'architecture logicielle qui se distingue par des modèles homogènes, un langage de description d'architecture et une méthodologie de conception. Les modèles sont adaptés à la spécification sans contraintes d'architectures logicielles, à la mise en évidence et la prise en

charge d'aspects spécifiques dans une architecture logicielle et à la spécification simultanée des aspects structurels et comportementaux.

Le langage permet de spécifier les aspects structurels, comportementaux et aussi les propriétés non fonctionnelles. Parmi ces dernières, la capacité du langage à permettre la fixation explicite du déploiement d'un composant dans une nature bien précise. Cette caractéristique a été très importante pour nos travaux. C'est à travers cette caractéristique que nous pouvons spécifier aisément une architecture de codesign en indiquant que certaines instances de composant devraient être déployées dans une nature logicielle (PROCESSUS, THREAD, SERVLET, etc..) et d'autres dans une nature matérielle.

L'approche IASA utilise un modèle de composant composite dans lequel le contrôle global de l'architecture est supporté par un composant explicite, appelé contrôleur. Ce qui n'est pas le cas des autres approches d'architecture logicielle qui font abstraction de ce composant.

Le travail présenté dans ce mémoire peut se résumer dans les points suivants :

- ✓ La définition d'un modèle de composant spécifique pour les composants de codesign matériel/logiciel.
- ✓ L'introduction dans IASA des cas de déploiement suivants: le cas HARD pour les composants matériels, le cas HS pour les composants mixtes matériel/logiciel et le cas HS_COMMUNICATION qui permet d'agir au niveau de la gestion du flot de contrôle.
- ✓ L'enrichissement du langage x3ADL pour lui permettre de prendre en charge à la fois les composants matériels et les composants logiciels.
- ✓ Un processus de transformation d'une cospécification abstraite décrite en x3ADL vers une vue d'implémentation représentée par la technologie JAVA pour les composants softwares et le langage VHDL pour les composants hardwares.

Cette contribution a été validée par :

- ✓ La mise en place d'un outil de spécification graphique d'architecture logicielle, appelé IASASTUDIO. Cet outil permet de générer en temps réel la description x3ADL.
- ✓ IASASTUDIO permet de réaliser la communication dans les deux sens : du hardware vers le software et du software vers le hardware. Il est doté d'une bibliothèque de composants softwares implémentés en JAVA et d'une bibliothèque de composants

hardwares implémentés en VHDL. Ainsi, il est possible de réaliser facilement la communication entre un composant matériel et un composant logiciel sans se soucier des détails de bas niveau.

Dans ce travail, nous n'avons pas exploité toutes les capacités de l'approche IASA, notamment l'orienté aspects et la spécification libre de topologie dans laquelle il est possible d'établir des connecteurs entre point d'accès de ports distincts. De plus, l'approche proposée ne résout pas tous les problèmes de la cosynthèse des interfaces de communication d'un système de codesign. Ainsi, les perspectives suggérées de ce travail peuvent se résumer dans les points suivants :

- L'exploitation de l'approche de conception par aspects dans la cosynthèse des interfaces de communication d'un système de codesign.
- La poursuite du travail dans l'élaboration du modèle de composant spécifique pour la gestion de la communication dans un système de codesign par la prise en charge des caractéristiques de la communication suivantes : le protocole de la communication, les types de transferts (synchrones ou asynchrones) et les caractéristiques de performance.

Enfin, nous tenons à préciser que pour l'instant notre travail s'est concrétisé par trois communications acceptées pour présentation à des conférences internationales :

1. "A software architecture approach for the synthesis of communication interfaces in the hardware / software co-design", The Sixth International Multi-Conference on Computing in the Global Information Technology, ICCGI 2011, Luxembourg.
2. "A Component Model for the Synthesis of Communication Interfaces in the Hardware / Software Codesign", The International Conference on Computer Science, Engineering & Applications, (ICCSEA – 2011), Dubai, UAE.
3. "A component model for the synthesis of communication interfaces", International Arab Conference on Information Technology (ACIT'2010), Libya.

APPENDICE A : LISTE DES ABREVIATIONS

ACTOAP	: Action Oriented Access Point
ADL	: Architecture Description Language
ASIC	: Application Specific Integrated Circuit
ASIP	: Application Specific Instruction-set Processor
CFSM	: Codesign Finite State Machine
COTS	: Component On The Shelf
DE	: Discrete Event
DFG	: Data Flow Graph
DMA	: Direct Memory Access
DOAP	: Data Oriented Access Point
DSP	: Digital Signal Processor
EJB	: Entreprise Java Beans
FDT	: Formal Description Technique
FPGA	: Field Programmable Gate Array
FSM	: Finite State Machine
FSMD	: Finite State Machines with Datapath
GA	: Genetic Algorithm
HCFSM	: Hierarchical Concurrent Finite State Machines
HDL	: Hardware Description Language
HPN	: Hierarchical Petri Nets
IASA	: Integrated Approach to Software Architecture
IEEE	: Institute of Electrical and Electronics Engineers
ILP	: Integer Linear Programming
MDA	: Model Driven Architecture
MoC	: Model of Computation
POJO	: Plain Old Java Object
PSO	: Particle Swarm Optimization
RISC	: Reduced Instruction Set Computer
RTL	: Register Transfer level
SA	: Simulated Annealing
SDF	: Synchronous DataFlow
SDL	: Specification and Description Language
SLDL	: System Level Design Languages
SoC	: System on Chip
SR	: Synchronous/Reactive
TA	: Transaction Accurate
TS	: Tabu Search
VHDL	: Very High Speed Integrated Circuit, Hardware Description language
VLIW	: Very Long Instruction Word
VLSI	: Very-Large-Scale Integration

REFERENCES

1. G. De Micheli and R. GUPTA, "Hardware/Software Co-Design", Proceedings of the IEEE, Vol. 85, no. 3, pp. 349-365, 1997.
2. E. Stoy, "A petri net based unified representation for Hw/Sw codesign", PhD thesis, Departement of Computer and Information Science, Linköping University, Sweden, 1995.
3. A. Shaout, A. H. El-Mousa and K. Mattar, "Models of Computation for Heterogeneous Embedded Systems", Electronic Engineering and Computing Technology, Lecture Notes in Electrical Engineering, vol. 60, pp. 201-213, 2010.
4. F. Vahid, "What is Hardware/Software Partitioning?", ACM/SIGDA E-NEWSLETTER, Vol. 39, No. 6, pp. 4-7, 2009.
5. D. Bennouar, "Une approche intégrée pour l'architecture logicielle ", Thèse de doctorat, Ecole Nationale Supérieure d'Informatique, Alger, 2009.
6. Unified Modeling Language: Infrastructure, version 2.0, 3rd revised submission to OMG RFP ad/00-09-01, January 2003.
7. D. Garlan, S.-W. Cheng, and A. J. Kompanek, "Reconciling the Needs of Architectural Description with Object-Modeling Notations", Science of Computer Programming Journal, Special UML Edition, Elsevier Science, vol.44, pp. 23-49, 2001.
8. C. Hofmeister, R. L. Nord and D. Soni, " Describing software architecture with UML", In Proceeding of the First Working IFIP Conference on Software Architecture, pp. 145-160, 1999.
9. J. D. Poole, "Model-Driven Architecture: Vision, Standards and Emerging Technologies", ECOOP 2001, Workshop on Metamodeling and Adaptive Object Models, 2001.
10. N. Medvidovic and R. N. Taylor, "A classification and comparison framework for software architecture description languages", In IEEE Transactions on Software Engineering, Vol. 26, no. 1, pp. 70–93, 2000.
11. A. Saadi, " Un langage d'action pour la spécification et la validation du comportement d'une architecture logicielle ", mémoire de magistère, Département d'Informatique, USDB, Algérie, 2008.
12. K. Bentlemsan, " Une approche architecture logicielle pour la composition de services web ", mémoire de magistère, Département d'Informatique, USDB, Algérie, 2010.
13. A. Chow, D. Hopkins, R. Drost and R. Ho, "Exploiting Capacitance in High-Performance Computer Systems ", Sun Microsystems Laboratories, Menlo Park, CA 94025, USA, 2008.
14. D. Heller, "Modélisation et évaluation de performances pour le codesign", Thèse de doctorat, Université de Nantes, France, 1998.
15. D. W. Franke and M.K. Purvis, "Hardware/Software Codesign: A Perspective", Proc. of 13th Intl. Conference on Software Engineering, Austin, Texas, USA, pp. 344-352,1991.
16. W. H. Wolf, "Hardware-software Co-design of Embedded Systems", Proceedings of the IEEE, vol.82, no.7, pp. 967-989, 1994.
17. S. Kumar, "The codesign of embedded systems: a unified hardware/software representation", Édition illustrée, 1996.
18. A. Kalavade, "System-Level Codesign of Mixed Hardware-Software Systems", Ph.D. thesis, University of California, Berkeley, USA, 1995.

19. G. Gogniat, "Architecture générique et synthèse des communications pour la conception conjointe des systèmes embarqués logiciel/matériel", Thèse de doctorat, Université de Nice - Sophia Antipolis, France, 1997.
20. P.R. Schaumont, "A Practical Introduction to Hardware/Software Codesign", Springer, USA, 2010.
21. D. C. de Souza," Algorithme de partitionnement appliqué aux systèmes dynamiquement reconfigurables en télécommunications", Thèse de doctorat, École Nationale Supérieure des Télécommunications, Paris, 2006.
22. W. H. Wolf, "A decade of hardware/software codesign", Proceedings of the IEEE 5th Int. Symposium on Multimedia Software Engineering, pp. 38-42, 2003.
23. S. Prakash and A.C. Parker, "SOS: Synthesis of Application-Specific Heterogeneous Multiprocessor Systems", Journal of Parallel and Distributed Computing, vol.16, pp. 338-351, 1992.
24. R. K. Gupta and G. De Micheli, "Hardware/Software Cosynthesis for Digital Systems", IEEE Design & Test of Computers, vol.10, no.3, pp. 29-41, 1993.
25. R. Ernst, J. Henkel, and T. Benner, "Hardware/Software Cosynthesis for Microcontrollers", IEEE Design & Test of Computers, Volume 10, Issue 4, pp. 64-75, 1993.
26. P. Grange, "Le livre blanc des systèmes embarqués", Publication : Syntec informatique, Paris, 2009.
27. I. Bolsens, H. J. De Man, B. Lin, K. Van Rompaey, S. Vercauteren, and D. Verkest," Hardware/software co-design of digital telecommunication systems", Proceedings of the IEEE, Vol. 85, no. 3, pp. 391-418, 1997.
28. M.D. Edwards and J. Forrest," Software acceleration using programmable hardware devices ", IEE Proc Computers and digital techniques, Volume 143, Issue 1, pp. 55-63, 1996.
29. G. F. Marchioro, "Découpage transformationnel pour la conception de systèmes mixtes logiciel/matériel", Thèse de doctorat, Institut national polytechnique de Grenoble, France, 1998.
30. Ptolemy Environment, http://ptolemy.eecs.berkeley.edu/
31. Metropolis Environment, http://embedded.eecs.berkeley.edu/metropolis/index.html
32. CoWare Platform, http://www.coware.com/products/platformarchitect.php
33. D. Hommais, "Une méthode d'évaluation et de synthèse des communications dans les systèmes intégrés matériel-logiciel ", Thèse de doctorat, université PARIS VI, 2001.
34. V. Mooney, T. Sakamoto and G. De Micheli, "Run-time scheduler synthesis for hardware-software systems and application to robot control design", In IEEE editor, Proceedings of the CHDL'97, pp. 95-99, 1997.
35. A. A. Jerraya, J. M. Daveau, G. F. Marchioro, C. Valderrama, M. Romdhani, T. Ben Ismail, N. E. Zergainoh, F. Hessel, P. Coste, Ph. Le Marrec, A. Baghdadi, and L. Gauthier," Hardware/Software Co-design ", IN: Design of Systems on a Chip: Design and Test, pp. 133-158, 2007.
36. C. A. Valderrama, "Prototype virtuel pour la génération des architectures mixtes logicielles/matérielles", Thèse de doctorat, Institut national polytechnique de Grenoble, France, 1998.
37. J. Henkel, T. Benner and R. Ernst, " Hardware generation and partitioning effects in the COSYMA system ", In Proceedings of the Second International Workshop on Hardware/Software Codesign, CODES/CASHE ’93, 1993.
38. D. Herrmann, J. Henkel and R. Ernst, "An approach to the adaptation of estimated cost parameters in the COSYMA system ", International Conference on Hardware Software

Codesign, Proceedings of the 3rd international workshop on Hardware/software co-design, Grenoble, France, pp. 100 - 107, 1994.

39. R. K. Gupta, C. Coelho, and G. De Micheli, "Synthesis and Simulation of Digital Systems Containing Interacting Hardware and Software Components", Proceedings of the 29th ACM/IEEE Design Automation Conference, pp. 225-230, 1992.
40. R. K. Gupta, "Co-synthesis of hardware and software for digital embedded systems", Ph. D. Dissertation, the department of electrical engineering and the committee on graduate studies of Stanford university, USA, 1993.
41. M. B. Abdelhalim and S.E.D. Habib, "An integrated high-level hardware/software partitioning methodology", Design Automation for Embedded Systems, vol.15, no.1, pp. 19-50, 2011.
42. G. J. Holzmann," Practical methods for the formal validation of SDL specifications ", Special issue on practical use of FDTs in communications & distributed systems, Volume 15, pp.129 – 134, 1992.
43. K. J. Turner, "The Formal Specification Language LOTOS", Department of Computing Science, University of Stirling, Scotland, 1996.
44. M. A. Fecko, M.U. Uyar, P.D. Amer, A.S. Sethi, T. Dzik, R. Menell and M. McMahon, " Success story of formal description techniques: Estelle specification and test generation for MIL-STD 188-220 ", Computer Communications, Vol. 23, no. 12, pp. 1196-1213, 2000.
45. G. Berry, "A hardware implementation of pure ESTEREL", Special Issue on Parallel and Distributed Computing, Vol. 17, no. 1, pp. 95-130, 1992.
46. StateCharts, http://statecharts.net/engl_index.htm
47. D. D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, and S. Zhao, "SpecC, Specification Language and design Methodology", Kluwer Academic, 2000.
48. Open SystemC Initiative, http://www.systemc.org/
49. J. M. Daveau, "Spécifications systèmes et synthèse de la communication pour le co-design logiciel/matériel ", Thèse de doctorat, Institut national polytechnique de Grenoble, France, 1997.
50. R. Niemann, "Hardware/software co-design for data flow dominated embedded systems", Boston: Kluwer Academic Publishers, 1998.
51. L. A. Cortes, P. Eles, and Z. Peng, " A survey on hardware/software codesign representation models", Dept. of Computer and Information Science, Linköping University, Linköping, Sweden, 1999.
52. D. D. Gajski, S. Abdi, A. Gerstlauer and G. Schirner, "Embedded System Design: Modeling, Synthesis and Verification", Springer Science + Business Media, 2009.
53. S. Edwards, L. Lavagno, E. A. Lee, A. Sangiovanni-Vincentelli, "Design of Embedded Systems: Formal Models, Validation, and Synthesis", Proceedings of the IEEE, Vol. 85, pp. 366-390, 1999.
54. D. D. Gajski, J. Zhu, and R. Dömer, "Essential Issues in Codesign", Technical report ICS- 97-26, University of California, Irvine, USA, 1997.
55. G. Bosman, "A Survey of Co-Design Ideas and Methodologies," Master's Thesis, Vrije Universiteit, Amsterdam, 2003.
56. A. Jantsch and I. Sander, "Models of Computation in the Design Process", Royal Institute of Technology, Stockholm, Sweden, 2005.
57. K. Ben Chehida, " Méthodologie de Partitionnement Logiciel/Matériel pour Plateformes Reconfigurables Dynamiquement", Thèse de doctorat, Université de Nice - Sophia Antipolis, France, 2004.

58. A. Agrawal, "Hardware Modeling and Simulation of Embedded Applications", M.S. Thesis, Department of Electrical Engineering, Vanderbilt University, Nashville, Tennessee, 2002.
59. S. J. Chen, G. H. Lin, P. A. Hsiung and Y. H. Hu, "Hardware Software Co-Design of a Multimedia SOC Platform", Springer Science + Business Media, 2009.
60. R. Ruelland, " Apport de la co-simulation dans la conception de l'architecture des dispositifs de commande numérique pour les systèmes électriques", Thèse de doctorat, Institut national polytechnique de Toulouse, France, 2002.
61. E. A. Lee, "Disciplined Heterogeneous Modeling", Springer-Verlag Berlin Heidelberg, MODELS 2010, Part II, LNCS 6395, pp. 273–287, 2010.
62. X. Liu, J. Liu, J. Eker and E. A. Lee, "Heterogeneous Modeling and Design of Control Systems", in Software-Enabled Control: Information Technology for Dynamical Systems, 2003.
63. G. De Jong, "A UML-Based Design Methodology for Real-Time and Embedded Systems", In Proceedings of the Design Automation and Test in Europe Conference (DATE'02), Paris, 2002.
64. J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming heterogeneity the Ptolemy approach", Proceedings of the IEEE, vol.91, no.1, pp. 127-144, 2003.
65. Z. A. Mann, "Partitioning algorithms for hardware/software co-design", PhD thesis, Budapest University of Technology and Economics, Hungary, 2004.
66. Y. Atat, " Conception de haut niveau des MPSoCs à partir d’une spécification Simulink : Passerelle entre la conception au niveau Système et la génération d’architecture", Thèse de doctorat, Institut national polytechnique de Grenoble, France, 2007.
67. P. Arato, Z. A. Mann and A. Orban, "Algorithmic Aspects of Hardware/Software Partitioning", ACM Transactions on Design Automation of Electronic Systems, Vol. 10, No. 1, pp. 136–156, 2005.
68. W. Jigang, T. Srikanthan and T. Jiao, "Algorithmic aspects for functional partitioning and scheduling in hardware/software co-design", Design Automation for Embedded Systems, vol.12, no.4, pp. 345-375, 2008.
69. Z. A. Mann, A. Orban and P. Arato, " Finding optimal hardware/software partitions ", Formal Methods in System Design, Vol. 31, No. 3, pp. 241–263, 2007.
70. M. Israel, D. Dupont, " De la spécification formelle au partitionnement matériel logiciel", Traitement du Signal, vol.14, no.6, pp. 559-568, 1997.
71. T. Ma, J. Yang, and Xi. Wang, "Low Power Hardware-Software Partitioning Algorithm for Heterogeneous Distributed Embedded Systems", Lecture Notes in Computer Science, vol. 4096/2006, pp. 702-711, 2006.
72. G. Stitt, "Hardware/Software Partitioning with Multi-Version Implementation Exploration", Proceedings of the 18th ACM Great Lakes symposium on VLSI, pp. 143-146, 2008.
73. P. Faes, P. Bertels, J. V. Campenhout and D. Stroobandt, "Using method interception for hardware/software co-development", Design Automation for Embedded Systems, vol.13, no.4, pp. 223-243, 2009.
74. W. Jigang, T. Srikanthan and G. Chen, " Algorithmic Aspects of Hardware/Software Partitioning: 1D Search Algorithms", IEEE TRANSACTIONS ON COMPUTERS, VOL. 59, NO. 4, pp. 532-544, 2010.
75. M. Loghi, T. Margaria, G. Pravadelli and B. Steffen, "Dynamic and Formal Verification of Embedded Systems: A Comparative Survey", International Journal of Parallel Programming, vol. 33, no. 6, pp. 585-611, 2005.

76. A. Aljer, " Co-design et raffinement en B : BHDL Tool, plateforme pour la conception de composants numériques", Thèse de doctorat, Université des Sciences et Technologies de Lille, France, 2004.
77. S. Yoo and A.A. Jerraya, "Hardware/software cosimulation from interface perspective", IEE proceedings Computers and digital techniques, vol. 152, no. 3, pp. 369-379, 2005.
78. M. Azizi, " Covérification des Systèmes Intégrés", Thèse de doctorat, Université de Montréal, Canada, 2000.
79. F. Fummi, M. Loghi, M. Poncino and G. Pravadelli, "A cosimulation methodology for HW/SW validation and performance estimation", ACM Transactions on Design Automation of Electronic Systems (TODAES), vol. 14, no. 2, Article 23, 2009.
80. P. Coste, "Conception des systèmes hétérogènes multilangages", Thèse de doctorat, Université Joseph Fourier - Grenoble I, France, 2001.
81. P. Herber, F. Friedemann and S. Glesner, "Combining Model Checking and Testing in a Continuous HW/SW Co-verification Process", Lecture Notes in Computer Science, pp. 121-136, 2009.
82. M. He, M. C. Tsai, X. Wu, F. Wang and R. Nasr, "Hardware/Software Codesign – Pedagogy for the Industry", Proceedings of the Fifth International Conference on Information Technology: New Generations, pp. 279-284, 2008.
83. F. Gharsalli, " Conception des interfaces logiciel-matériel pour l'intégration des mémoires globales dans les systèmes monopuces", Thèse de doctorat, Institut national polytechnique de Grenoble, France, 2003.
84. E. G. N. Nicolescu, " Spécification et validation des systèmes hétérogènes embarqués", Thèse de doctorat, Institut national polytechnique de Grenoble, France, 2002.
85. A. Chureau, " Définition d'une représentation intermédiaire basée sur une approche service pour le prototypage virtuel de systèmes sur puce", Thèse de doctorat, Institut national polytechnique de Grenoble, France, 2008.
86. P. Coussy, " Synthèse d’interface de communication pour les composants virtuels", Thèse de doctorat, Université de Bretagne Sud, France, 2003.
87. M. Koudil, "Une approche orientée objet pour le codesign", Thèse de doctorat d’état, Institut National d’Informatique, Alger, 2002.
88. M. O’nils, "Specification, Synthesis and Validation of Hardware/Software Interfaces ", Doctoral thesis, Department of Electronics, Royal Institute of technology, Stockholm, 1999.
89. K. Popovici and A. A. Jerraya, "Simulink based hardware-software codesign flow for heterogeneous MPSoC ", Proceedings of the 2007 summer computer simulation conference, pp. 497-504, 2007.
90. D. Lyonnard, " Approche d'assemblage systématique d'éléments d'interface pour la génération d'architecture multiprocesseur", Thèse de doctorat, Institut national polytechnique de Grenoble, France, 2003.
91. S. Chaitanya, B. Urgaonkar and A. Sivasubramaniam, " QDSL: a queuing model for systems with differential service levels", Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, Vol. 36, no. 1, pp. 289-300, 2008.
92. W. Klingauf, "Systematic Transaction Level Modeling of Embedded Systems with SystemC", Proceedings of the conference on Design, Automation and Test in Europe, Vol. 1, pp. 566 - 567, 2005.
93. D. Shin, A. Gerstlauer, J. Peng, R. Dömer and D. D. Gajski, " Automatic generation of transaction level models for rapid design space exploration", Proceedings of the 4th international conference on Hardware/software codesign and system synthesis, pp. 64 - 69, 2006.

94. G. Schirner and R. Dömer, " Quantitative analysis of the speed/accuracy trade-off in transaction level modeling ", ACM Transactions on Embedded Computing Systems (TECS), Vol. 8, no. 1, 2008.
95. A. Mathur and V. Krishnaswamy, "Design for verification in system-level models and RTL", Proceedings of the 44th annual Design Automation Conference, pp. 193 - 198, 2007.
96. D. S. Narbonne, C. Chan, Y. Mahajan and S. Malik, "Supporting RTL flow compatibility in a microarchitecture-level design framework", Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis, pp. 343-352, 2009.
97. Y. Paviot, "Partitionnement des services de communication en vue de la génération automatique des interfaces logicielles/matérielles", Thèse de doctorat, Institut national polytechnique de Grenoble, France, 2004.
98. M. W. Youssef, " Étude des interfaces logicielles/matérielles dans le cadre des systèmes multiprocesseurs monopuces et des modèles de programmation parallèle de haut niveau ", Thèse de doctorat, Université Joseph Fourier - Grenoble I, France, 2006.
99. J. Quartana, " Conception de réseaux de communication sur puce asynchrones : application aux architectures GALS", Thèse de doctorat, Institut national polytechnique de Grenoble, France, 2004.
100. A. Grasset, "Synthèse des interfaces de communication dans la conception des systèmes monopuces : de la spécification à la génération automatique", Thèse de doctorat, Institut national polytechnique de Grenoble, France, 2005.
101. I. Maalej, G. Gogniat, M. Abid and J. L. Philippe, "Conception d’interface pour processeur embarqué dans les systèmes sur puce", L.E.S.T.E.R., UNIVERSITE DE BRETAGNE SUD, FRANCE, 2002.
102. C. Chavet, "Synthèse automatique d'interfaces de communication matérielles pour la conception d'applications du domaine du traitement du signal", Thèse de doctorat, Université de Bretagne Sud, France, 2007.
103. G. N. Khan and U. Ahmed, "CAD tool for hardware software co-synthesis of heterogeneous multiple processor embedded architectures", Design Automation for Embedded Systems, vol.12, no. 4, pp. 313–343, 2008.
104. K. Hao and F. Xie, "Componentizing hardware/software interface design", Proceedings of the Conference on Design, Automation and Test in Europe, pp. 232-237, 2009.
105. S. Narayan and D. Gajski, "Synthesis of System-Level Bus Interfaces", Proceedings of the European Design Automation Conference with Euro-VHDL, pp. 395-399, 1994.
106. W. Ecker, M. Glesner and A. Vombach, "Protocol merging: A VHDL Based Method for Clock Cycle Minimising and Protocol Preserving Scheduling of IO Operations", Proceedings of the European Design Automation Conference with Euro-VHDL, pp. 624-629, 1994.
107. J. Henkel, R. Ernst, U. Holtman and T. Benner, "Adaptation of Partitioning and High Level synthesis in Hardware/Software Co-Synthesis", Proceedings of IEEE International Conference on Computer Aided Design, pp. 96-100, 1994.
108. J. Madsen and B. Hald, "An Approach to Interface Synthesis", Proceedings of the 8th International Symposium on System Synthesis, pp. 16-21, 1995.
109. M. B. Srivastava and R. W. Brodersen, "SIERA : A Unified Framework for Rapid Prototyping of System Level Hardware and Software", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 14, No. 6, pp. 676-693, 1995.
110. R. K. Gupta and G. De Micheli, "A co-synthesis approach to embedded system design automation", Design Automation for Embedded Systems, volume 1, pp. 69-120, 1996.

111. J. M. Daveau, G. F. Marchioro, T. Ben-Ismail and A. A. Jerraya, "Protocol selection and interface generation for hw-sw codesign", IEEE Transaction on VLSI Systems, Special Issue On Design Automation of Complex Integrated Systems, vol.5, no. 1, pp. 136–144, 1997.
112. R. Passerone, J. A. Rowson, and A. Sangiovanni-Vincentelli, "Automatic synthesis of interfaces between incompatible protocols", Proceedings of the 35th annual Design Automation Conference, pp. 8-13, 1998.
113. S. Vercauteren, J. V. D. Steen, and D. Berkest, "Combining software synthesis and hardware/software interface generation to meet hard realtime constraints", Proceedings of the conference on Design Automation and Test in Europe DATE, 1999.
114. P. Chou, R. Ortega, K. Hines, K. Partridge and G. Borriello, " IPCHINOOK: An integrated IP-based design framework for distributed embedded systems", In Proceedings of the 36th Design Automation Conference, pp. 44–49, 1999.
115. M. O'Nils and A. Jantsch, "Device Driver and DMA Controller Synthesis from HW /SW Communication Protocol Specifications ", Design Automation for Embedded Systems, Volume 6, Number 2, pp. 177-205, 2001.
116. A. Sarmento, W. Cesario and A.A. Jerraya, "Automatic building of executable models from abstract SoC architectures made of heterogeneous subsystems", In 15th IEEE International Workshop on Rapid System Prototyping (RSP), pp. 88–95, 2004.
117. J. Smith and G. D. Micheli, "Automated composition of hardware components", In Proceeding of 35th Design Automation Conference, pp. 14-19, 1998.
118. C. Schurgers, F. Catthoor and M. Engels, "Memory optimization of MAP turbo decoder algorithms", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.9, no. 2, pp. 305-312, 2001.
119. V. Kathail, S. Aditya, R. Schreiber, B.R. Rau, D.C. Cronquist, and M. Sivaraman, "Pico : Automatically designing custom computers", Computer, vol. 35, no. 9, pp. 39-47, 2002.
120. S. Gupta, "Tutorial for the SPARK Parallelizing High-Level Synthesis Framework", Center for Embedded Computer Systems, University of California at Irvine, 2003.
121. M. Kudlur, K. Fan and S. Mahlke, "Streamroller: Automatic Synthesis of Prescribed Throughput Accelerator Pipelines", CODES+ISSS'06, pp. 270-275, Seoul, Korea, 2006.
122. J. Cong and W. Jiang, "Pattern-based Behavior Synthesis for FPGA Resource Reduction", Proc. 16th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA 2008), Monterey, USA, 2008.
123. Catapult C Synthesis, http://www.mentor.com/esl/catapult/overview
124. SynDEx: system level CAD software, http://www.syndex.org/
125. GAUT - High-Level Synthesis tool, http://www-labsticc.univ-ubs.fr/www-gaut/
126. Y. Mancer and D. Bennouar,"A software architecture approach for the synthesis of communication interfaces in the hardware / software co-design", The Sixth International Multi-Conference on Computing in the Global Information Technology, ICCGI 2011, Luxembourg, 2011.
127. A. E. Ozcan, " Conception et Implantation d'un Environnement de Développement de Logiciels à Base de Composants", Thèse de doctorat, Institut national polytechnique de Grenoble, France, 2007.
128. S. Chardigny, "Extraction d'une architecture logicielle à base de composants depuis un système orienté objet", Thèse de doctorat, Université de Nantes, ÉCOLE DOCTORALE STIM, France, 2009.
129. R. Allen, R. Douence, and D. Garlan," Specifying and analyzing dynamic software architectures", Lecture Notes in Computer Science, Vol. 1382, pp. 21-37, 1998.

130. N. Medvidovic, "Architecture-Based Specification-Time Software Evolution", Phd Thesis, UNIVERSITY OF CALIFORNIA, IRVINE, 1999.
131. G. Zelesnik, "The UniCon Language Reference Manual", School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213-3890, 1996.
132. D. Garlan and D.Perry, " Introduction to the special issue on software architecture ", IEEE Transactions on Software Engineering, vol. 21, no. 4, pp. 269-274, 1995.
133. R. Koschke, "Atomic Architectural Component Recovery for Program Understanding and Evolution ", Thèse de doctorat, Université de Stuttgart, 2000.
134. D. C. Luckham, J. L. Kenney, L. M. Augustin, J. Vera, D. Bryan and W. Mann," Specification and analysis of system architecture using rapide ", IEEE Transactions on Software Engineering, Vol. 21, no. 4, pp. 336–354, 1995.
135. J. Magee, N. Dulay, S. Eisenbach and J. Kramer, "Specifying Distributed Software Architectures", Proc. 5th European Software Engineering Conf. (ESEC 95), pp. 137–153, 1995.
136. D. Bennouar, T. Khammaci and A. Henni, "Modeling The Component's Interaction Point In The IASA Approach", The Mediterranean Journal of Computer and Networks, Vol. 4, N° 4, UK, 2008.
137. D. Garlan, "Software Architecture ", Published In Encyclopedia of Software Engineering, John Wiley & Sons, 2001.
138. D. Bennouar and A. Henni, " SEAL: An aspect oriented ADL ", Conference ACIT, Sanaa, Yemen, 2009.
139. E. M. Dashofy, A. V. D. Hoek and R. N. Taylor, "An infrastructure for the rapid development of XML-based architecture description languages", Proceedings of the 24th International Conference on Software Engineering, pp. 266- 276, 2002.
140. xADL 2.0, http://www.isr.uci.edu/projects/xarchuci/index.html
141. B. Schmerl, "xAcme: CMU Acme Extensions to xArch", 2001. http://www.cs.cmu.edu/~acme/pub/xAcme/guide.pdf
142. D. Garlan, " Software architecture: a roadmap ", In ICSE '00: Proceedings of the Conference on The Future of Software Engineering, ACM , New York, NY, USA, pp. 91-101, 2000.