A. Spécification et conception de systèmes sur puce : B. SystemC et approches actuelles

Fatma abbes[1], Mohamed abid[1], Emmanuel Casseau[2]
[1]GMS, ECOLE NATIONALE D'INGENIEURS DE SFAX, 3038 SFAX, TUNISIE
[2]L.E.S.T.E.R., UNIVERSITE DE BRETAGNE SUD, RUE SAINT MAUDE - 56100 LORIENT, FRANCE
fatma_abbes@yahoo.fr, mohamed.abid@enis.rnu.tn, emmanuel.casseau@univ-ubs.fr

Résumé

Actuellement, les systèmes deviennent de plus en plus complexes et la tendance est à l'intégration du logiciel et du matériel sur un même circuit (SoC pour System On Chip). L'intégration d'un système nécessite de relever de nouveaux défis, en particulier en ce qui concerne l'utilisation d'un langage unifié pour la conception de haut niveau dite conception système. Dans cet article, nous décrivons les différentes approches de spécification avec des langages de modélisation à un niveau d'abstraction élevé et plus particulièrement le langage SystemC, en développant l'approche de modélisation associée, les outils qui supportent ce langage ainsi que les travaux actuels correspondants. Ce travail de synthèse est réalisée afin de mettre en avant l'importance de cette évolution en conception de Systèmes sur Puce (SoC).

Mots-clés : **Conception de niveau système, intégration système, SoC, SystemC.**

I. INTRODUCTION

La complexité croissante des applications entraîne une augmentation considérable de la durée de développement avec les méthodes de conception conventionnelles. Une des contraintes majeures est le délais de mise sur le marché ou « Time_to_Market ». La conception des SoCs (figure 1) est un nouveau concept. Il exige typiquement la modélisation et l'intégration simultanée du « software » (logiciel) et du « hardware » (matériel) : c'est l'approche dite co-design.

La conception de haut niveau représente l'un des points clés de la conception des SoCs. Cependant, l'utilisation de langages de conception différents (C++, HDLs, etc.), d'outils de CAO plus ou moins compatibles et l'usage de flots de conception séparés pour le matériel et le logiciel handicapent fortement la conception des SoCs [19].

Ainsi, plusieurs travaux de recherches concernant l'exploration de nouveaux langages ou de nouvelles méthodologies de conception pour arriver à modéliser les parties logicielles et matérielles et supporter les interfaces entre les deux ont démarré ces dernières années [18].

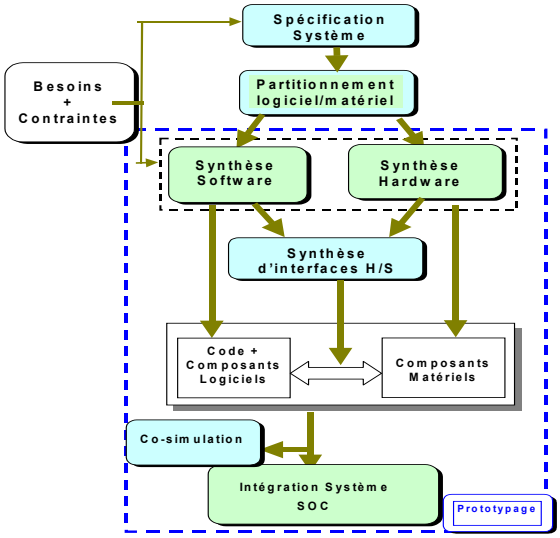

Fig. 1 : Flot typique de conception des SoCs

Une des questions soulevées est de savoir comment modéliser et décrire la fonctionnalité d'un système à un niveau de détail suffisant permettant de prévoir son comportement intégral, sans ambiguïté ou à un niveau d'abstraction qui ne fait pas d'hypothèse sur les cibles d'implantation. Le langage SystemC a été développé comme un langage de modélisation dans cette optique [14].

L'objet de ce papier est d'étudier, dans le cadre de la conception des SoCs, le besoin en terme de spécification au niveau système dans un flot de co-design d'une part et, d'autre part, les différentes approches existantes pour la mise en oeuvre d'un langage de modélisation système. Le but de cette analyse est de prouver, dans le contexte actuel de concurrence entre langages système, l'importance des approches basées sur des langages de type C/C++, notamment SystemC qui semble récolter beaucoup d'intérêt.

L'article est construit comme suit : la première partie décrit les différentes approches existantes pour la mise en oeuvre d'un langage de modélisation à un niveau d'abstraction élevé. La section suivante s'intéresse à l'approche associée au langage SystemC. ses principales caractéristiques et son intérêt lors de la spécification et les raffinements successifs à

plusieurs niveaux d'abstraction et plus particulièrement dans le flot de conception matériel sont présentés. Les troisième et quatrième sections présentent les outils adoptant le langage SystemC et les travaux de recherche associés. La dernière partie de l'article est consacrée aux conclusions et perspectives ainsi qu'à une présentation de nos travaux futurs concernant notre approche de spécification et de raffinement.

II. LANGAGES SYSTEME : INITIATIVES ET APPROCHES ASSOCIEES

Actuellement, les efforts se multiplient afin d'étudier et de mettre en place un langage permettant de décrire le matériel à un niveau d'abstraction élevé. De plus, comme le logiciel tend à devenir la partie la plus importante des systèmes électroniques (le software peut représenter jusqu'à 90% du modèle) (figure 2), les efforts sont focalisés vers un langage autorisant des descriptions conjointes logicielle et matérielle.

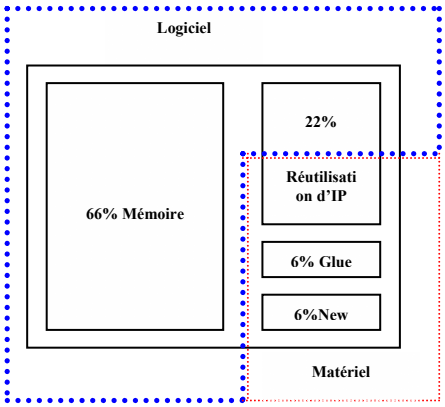

Fig. 2 : Exemple de répartition logiciel/matériel

Travailler à niveau système permet d'obtenir des modèles afin de gérer la complexité croissante des systèmes et d'être indépendant de la technologie [21].

A. *Introduction de nouveaux langages*

Il n'existe pas actuellement de langage permettant de modéliser parfaitement le niveau système [1]. Cela a poussé la communauté à définir de nouveaux langages avec lesquels on peut décrire les exigences d'une modélisation de niveau système.

L'Initiative SLDL (VHDL International's System-Level Design Language) [11], financé principalement par le comité VHDL International, estime qu'il est trop difficile de considérer tous les critères d'une spécification de niveau système, pour tous les niveaux d'abstraction et dans tous les domaines d'applications dans un seul langage. En 1999, ce groupe proposa Rosetta [5]. Il s'agit d'un support pour la conception dans différents domaines employant une sémantique commune et une syntaxe appropriée pour chacun. Il permet de décrire les exigences selon une variété de points ou facettes. Chaque facette laisse la possibilité d'approcher le problème selon différents angles (surface, consommation, interface utilisateur, etc.) et permet le développement et l'utilisation d'outils qui sont spécifiques à la nature de chaque facette. Toutefois Rosetta reste actuellement incomplet sémantiquement et sa finalisation peut prendre encore quelques années en considérant la complexité de ce projet [17].

Co-Design Automation, une des compagnies d'automatisation de conception électronique (EDA : Electronic Design Automation), propose le langage Superlog comme solution de conception. Les auteurs de Superlog [13], avec Phil Marby, le développeur de Verilog, Peter Flake, le développeur de HILO (High Logic), estiment que ce langage est d'une part simple à lire et suffisamment performant pour résoudre les problèmes liés à un SoC [5]. Toutefois, Superlog est centré sur la conception matérielle et présente quelques déficits au niveau de la conception de haut niveau.

B. *Méthodologie basée sur les langages HDLs*

OVI (Open Verilog Initiative) et IEEE ont travaillé pour améliorer Verilog depuis 1994 [7]. Le but essentiel est d'ajouter des constructions à ce langage afin de permettre son utilisation exclusive pour la conception des SoCs. La base de ces modifications est la capacité de ce langage à être lié à un code écrit en C/C++ à travers son interface de programmation de langage (PLI : Programming Language Interface) ce qui a favorisé l'aspect Orienté Objet avec cette possibilité. Toutefois, ces constructions conduisent à une diminution des performances en temps de simulation.

Des études similaires sur l'extension du langage VHDL ont également eu lieux.

En fait, les systèmes tendant à être dominés par le logiciel, l'adoption d'une méthodologie basée sur un langage HDL n'est pas vraiment appropriée.

On peut par ailleurs noter que VHDL International, qui a le plus d'intérêt dans la persistance du langage VHDL, investie dans le langage (Rosetta) et semble abandonner les efforts d'extension de VHDL.

C. *Méthodologies basées sur C++*

La conception de systèmes sur puces (SoC) basé sur le langage C a déjà donné des résultats intéressants en terme de gain en temps de développement et de vérification [16]. Ce langage est typiquement choisi pour sa popularité dans le développement logiciel.

Les méthodologies utilisant des langages basés sur le C++ tels SystemC, Ocapi, SpecC ont pour objectif de gèrer la complexité et l'hétérogénéité des SoCs. Le principe de l'Orienté Objet permet de séparer l'interface de l'implémentation, et favorise la réutilisation à un niveau d'abstraction élevé.

Une des questions qui se posent est la suivante : pourquoi C++ et non le langage Java ? Principalement, pour simuler des systèmes très complexes, la rapidité est exigée.

- ✓ La technologie de compilation à base de C++ est plus évoluée, mure et fournit des exécutables au temps de simulation acceptable.
- ✓ Java empêche l'accès direct au matériel ce qui n'est pas favorable pour écrire un code d'initialisation et les pilotes des composants matériels
- ✓ C/C++ permet la réutilisation plus facile des cœurs de fonction développés pour implémenter et simuler les systèmes embarqués.

CynApps propose Cynlib [17]. C'est une source ouverte basée sur C++ pour modéliser le matériel. Il s'agit d'un ensemble de classes C++ qui implémente des concepts existants dans les langages matériels tels que Verilog et VHDL. La limitation majeure de Cynlib est que la suite de CynApps vise spécialement la conception du matériel. Une extension de ce langage est faite au sein du groupe CIRCUS (Codesign Interest for Research Captivated US) : Syslib. C'est un langage qui repose sur deux langages existants, Cynlib et C++. Il élargit les fonctionnalités de Cynlib dédiées pour le niveau système. Un outil graphique appelé Picasso supporte ce langage et permet de définir une méthodologie appelée Syslib_Picasso qui fait le lien entre le langage Cynlib et l'outil Picasso pour modéliser des SoCs selon la méthodologie CIRCUS [18].

SpecC est une extension de ANSI-C [12]. L'initiative de SpecC (STOC : SpecC Technologie Open Consortium) vise à favoriser une méthodologie de Co-Design focalisée sur les IPs (Propriété Intellectuelle) pour la spécification, la modélisation et les systèmes embarqués au niveau système. Le développement de ce langage, depuis 1999, est mené par Daniel Gajski à l'université de California–Irvine et financé par Toshiba [4].

Actuellement, l'initiative SystemC semble être plus avantagée par rapport aux autres approches [20]:

- ✓ SystemC est supporté par les principales sociétés de la CAO et de l'industrie électronique et établit une plate-forme de modélisation commune qui sert aussi comme base pour l'échange des IPs et des spécifications exécutables.
- ✓ OSCI (Open SystemC Initiative) établit un mécanisme ouvert pour certaines compagnies pour faire des contributions techniques et les partager avec des communautés larges.
- ✓ Tout comme SpecC, SystemC répond aux besoins techniques pour le matériel, le software et répond en ce sens aux attentes des concepteurs systèmes.

STOC et OSCI investissent énormément au profit de l'interopérabilité afin de garantir le maximum de bénéfices pour la conception système [18].

Enfin, notons que, d'après CynApps, Cynlib est compatible avec SpecC et Open SystemC.

Nous proposons d'aborder dans la suite de cet article l'approche associée à SystemC.

III. APPROCHE DE CONCEPTION ET DE SPECIFICATION A PARTIR DE SYSTEMC

Le but de SystemC est de définir une plate-forme de modélisation à base de C++ (librairies de classes C++) et d'un noyau de simulation [17]. Il supporte la spécification de niveau RTL, de niveau comportemental et de niveau système.

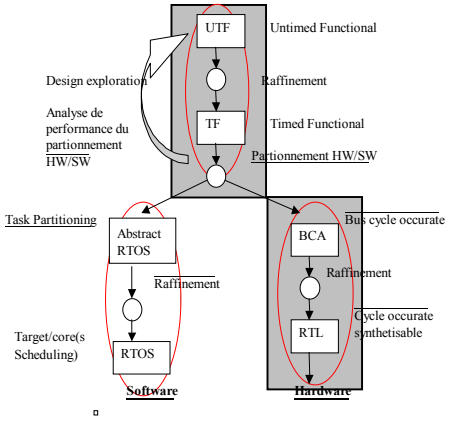

Fig. 3 : Flot de raffinement SystemC (source : OSCI)

L'OSCI propose une méthodologie de raffinement progressive d'une spécification, en débutant avec la description du système à un très haut niveau, puis, en s'approchant progressivement, étape pas étape, du comportement final du circuit. Il est possible d'entrer la spécification au niveau fonctionnel et de la simuler pour en vérifier la validité. Puis, le code est raffiné au niveau comportemental; pour cela, on ajoute entre autre des notions de temps, ce qui permet de faciliter l'étape de partitionnement. Enfin, on raffine les spécifications vers des phases d'implantation. Le flot complet, présenté par le figure 3, n'utilise qu'un seul langage du début à la fin, c'est là son principal intérêt.

On peut donc, par raffinement successif, passer d'un niveau donné au niveau suivant sans avoir à réaliser un changement de langage de modélisation qui posait obligatoirement des problèmes

auparavant (sémantique différente, syntaxe différentes etc.), sans même parler du temps inutilement dépensé pour cela.

C'est principalement le niveau de détail du temps qui différencie les niveaux d'abstraction en SystemC. La conception est progressivement raffinée par l'ajout du code du matériel ainsi que le chronométrage nécessaire.

Les principaux niveaux d'abstraction liés au flot matériel sont :

Untimed Functional Level (UFL)

L'objectif est de réaliser une validation des concepts de base du système, et d'exprimer le système sous forme de modules fonctionnels sans chronométrage de l'information et de vérifier son fonctionnement avant de poursuivre les étapes suivantes du flot. On produit donc une spécification exécutable où l'architecture du système n'est pas encore définie.

Timed Functional Level (TFL)

A ce niveau, on modélise des « pseudo-temps » d'exécution pour chaque module, tout en respectant toujours les contraintes du système (cahier des charges).

Bus Cycle Accurate Level

A ce niveau, la modélisation des communications entre les différents blocs de notre système est nécessaire. Normalement la modélisation des canaux de communication doit être faite cycle par cycle. Par contre le comportement des modules peut rester au niveau UFL.

Cycle Accurate Level

A ce niveau, tout le fonctionnement du système doit être maintenant spécifié au cycle près.

IV. OUTILS ET ENVIRONNEMENTS AUTOUR DE SYSTEMC

Comme nous l'avons vu précédemment, la conception de SoCs basée sur le C a déjà donné des résultats intéressants en terme de gain en temps de développement et de vérification [15]. Cependant, une approche système ne repose pas simplement sur le langage. Il est essentiel d'avoir un support en terme de méthodologie et une disponibilité d'outils EDA (Electronic Design Automation). Le choix d'un langage est aussi conditionné par la disponibilité d'outils de spécification, de synthèse et de simulation utilisant ce langage.

Au niveau système, spécialement au début du cycle de vie du produit, les complexités de l'implémentation et la technologie du produit exigent une méthodologie de conception robuste et une suite d'outils sophistiqués pour automatiser le processus. De nombreux outils de synthèse d'architectures, permettant de passer de manière automatique d'une spécification de niveau comportemental à une spécification de niveau structurel (RTL), sont par exemple déjà en cours de développement comme le suggère la méthodologie proposée par l'OSCI [20].

Dans la suite de ce paragraphe, nous présentons les principaux outils qui ont adopté SystemC comme langage support.

A. CoCentric SystemC Compiler (CCSC)

CCSC de Synopsys permet de synthétiser une description matérielle à partir d'un code source rédigé avec SystemC (figure4). C'est un outil utile pour les équipes de conception qui partent d'une description C/C++ de niveau système pour la partie matérielle pour ensuite migrer vers :

- ✓ Une description niveau « Porte Logique »,
- ✓ Une implémentation Verilog synthétisable

ou

- ✓ Une description VHDL-RTL.

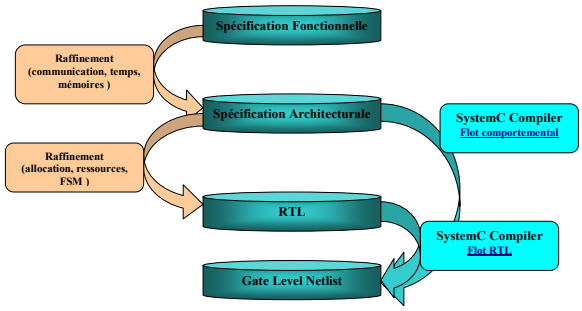

Fig. 4 : Flot d'implémentation matériel avec CoCentric SystemC Compiler

SystemC Compiler est un outil qui peut accepter des descriptions SystemC comportementales et SystemC RTL et exécute la synthèse comportementale ou la synthèse RTL.

B. CoCentric System Studio

Cet outil de Synopsys permet la vérification aux différents niveaux d'abstraction avec SystemC. Avec cet outil, il est possible de vérifier la spécification logicielle d'un SoC en combinant différents niveaux d'abstraction (co-simulation multi-niveaux), ce qui en fait son principal intérêt [3].

C. TestBuilder

TestBuilder a été développé par Cadence à l'origine indépendamment de SystemC. Cet outil se présente comme une bibliothèque de classe C++ permettant le développement de TestBenchs avancés. Il utilise

la méthodologie de vérification basée sur les transactions (The Transaction-Based Verification). Bien que SystemC possède les bases nécessaires pour générer des TestBenchs (concurrence, etc.), TestBuilder est particulièrement bien adapté à cet objectif [2]. TestBuilderSC est la version de TestBuilder dédiée à SystemC.

V. TRAVAUX DE RECHERCHE AUTOUR DE SYSTEMC

Il semble établi que SystemC est un candidat privilégié au futur standard pour la spécification de SoCs. Actuellement, SystemC est principalement utilisé (ou expérimenté) en tant qu'environnement de modélisation et de simulation. Des travaux sont en cours pour contribuer à l'approche SystemC en tant que méthodologie de raffinement couvrant tous les niveaux d'abstraction (enrichissement de la librairie de classes pour garantir la standardisation). Les sociétés telles Coware et Synosys travaillent sur l'automatisation du flot de conception associé à SystemC à travers la mise en œuvre d'outils de synthèse comme N2C et CoCentric Compiler et d'outils de vérification comme CoCentric Studio.

On peut également mentionner les travaux suivants :

CIRCUS (École Polytechnique 2002) [10]

- ✓ Analyse et modification de l'ordonnancement des tâches du simulateur de SystemC 2.0
- ✓ Analyse et conception d'une plate-forme de Co-Design
- ✓ Comparaison de SystemC de niveau UTF et Syslib FL.

LASSO (Laboratoire d'Analyse et de Synthèse des Systèmes Ordinés) [9]

- ✓ Modification de SystemC, méthodologies de modélisation et modélisation UML pour Co-Design.
- ✓ Visualisation des résultats de simulation de matériel modélisé avec SystemC.

ENSTA (Ecole Nationale Supérieur des Technologies Avancées, Paris) [6]

- ✓ Vérification formelle appliquée à SystemC.

IRIT (Institut de Recherche en Informatique de Toulouse) [8]

- ✓ Elaboration d'une bibliothèque de composants de simulation de processeurs réutilisables fondée sur SystemC.

VI. CONCLUSION ET PERSPECTIVES

Les systèmes deviennent de plus en plus complexes. La densité de transistors et de composants par circuit augmente et rendra, dans un avenir proche, le travail de conception ingérable. Plus que jamais, des principes et règles à suivre s'imposent, dont entre autre la réutilisation ou l'abstraction des spécifications du système. Plusieurs initiatives ont été lancées dans cette direction : SystemC, SpecC, Cynlib pour ne nommer que les plus populaires. Elles reposent sur de nouvelles méthodes de modélisation matérielle (et logicielles dans certains cas) basé sur le langage C++ et les fondements de l'orienté objet. SystemC semble être l'approche la plus populaire actuellement.

Toutefois, il ne faut pas perdre de vue que SystemC est un langage qui est en train de mûrir. Aujourd'hui, beaucoup de compagnies et d'universités s'y intéressent et désirent l'intégrer dans leurs outils. Cependant, comme pour tout langage/produit en cours de maturation, tout n'est pas encore opérationnel avec SystemC et de nombreux travaux demeurent encore inachevés. Les fonctionnalités de SystemC sont en train de s'enrichir avec le développement de librairies de classes génériques ou spécifiques. De leur coté, les outils associés qui sont en train d'apparaître vont permettre d'enrichir la méthodologie de « design » associée à ce langage.

Parmis les évolutions futures prévues au roadmap SystemC, on peut citer :

- ✓ Dans les futures version 2x, des méthodes de synchronisation de processus (simulation software), d'interruption, ainsi que la modélisation de performances du système devraient être intégrées.
- ✓ Les versions 3.x devraient permettre la modélisation de systèmes contenant des ordonnanceurs et des RTOS,
- ✓ Les versions 4.x doivent de leur coté permettre la modélisation des systèmes analogiques avec les systèmes numérique.

Rappelons que SystemC est né d'un besoin qui se fait pressant par rapport aux contraintes et à la complexité des systèmes actuels. L'objectif final de SystemC est d'offrir un langage permettant de modéliser tous les niveaux et donc de remplacer les laborieuses descriptions multi-langages malheureusement habituelles en conception de SoCs.

Pour passer d'une description de niveau système à l'implémentation, les différents niveaux d'abstraction doivent être bien définis et les méthodes de raffinement bien précises et claires. Nos travaux en cours en ce sens consistent à définir une méthodologie de raffinement aux niveaux systèmes avec ce langage. Nous proposons de traiter quatre types de raffinements de l'Orienté Objet définissant, plus généralement, le raffinement dans le flot matériel avec SystemC [22] : le

raffinement d'atomicité, le raffinement algorithmique, le raffinement des données et le raffinement des communications.

Notre démarche de raffinement consiste en l'enchaînement de ces étapes élémentaires qui va nous permettre de conduire un système exprimé du niveau fonctionnel jusqu'au niveau cycle près. L'ordre et le nombre de répétitions de chacune de ces étapes ne sont pas figé; cela dépendra de la complexité et de la nature du système à traiter.

Références

[1] Alan Fitch, "Application of SystemC to hw/sw Co-Design". IEE Seminar - Hardware-software Co-Design. December 2000.

[2] C. Norris Ip, Stuart Swan "Using Transaction-Based Verification in SystemC ", rapport technique CADENCE Inc. 11/6/2002

[3] "Cocentric System Studio enables verification at multiple levels of abstraction with systemC", rapport technique janvier 2002 disponible à l'adresse www.synopsys.com/products.

[4] Daniel D.Gajski, Rainer Domer , Jianwen Zhu, "IP-Centric methodology and design with the SpecC Language" NATO-ASI Workshop on System Level Synthesis, Aout 1998.

[5] EDN issue 5 Juin 2000

[6] http://lester.univ-ubs.fr/

[7] http://www.eda.org/ovl/

[8] http://www.irit.fr/

[9] http://www.iro.umontreal.ca/labs/lasso/

[10] http://www.polymtl.ca/

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

[12] http://www.SpecC.org

[13] http://www.superlog.org

[14] http://www.systemc.org/

[15] K.WAKABAYASHI,T.OKAMOTO, "C-Based SoC Design Flow and EDA Tools: An ASIC and System Vendor Perspective," IEEE transactions on computer aided design of integrated circuits and systems, VOL.19,NO.12,Décembre 2000,pp.1507-1522.

[16] K.WAKABAYASHI,T.OKAMOTO, "C-Based SoC Design Flow and EDA Tools: An ASIC and System Vendor Perspective," IEEE transactions on computer aided design of integrated circuits and systems, VOL.19,NO.12,Décembre 2000,pp.1507-1522.

[17] L. Filion, G. Bois, and E. M. Aboulhamid, "Syslib: A System-Level Language Extended from Cynlib for SoC," Proceedings of The 11th Annual International HDL Conference, San Jose, CA, March 11-12, 2002, pp. 191-197

[18] L. Filion, G. Bois, and E. M. Aboulhamid, "Picasso-Syslib : méthodologie complète de conception des systèmes embarqués", cours ELE6904 Seminaires de l'Ecole Polytechnique Montréal

[19] Luc Séméria, "Applying pointer analysis to the synthesis of hardware from C", mémoire de thèse en génie électrique Université de Stanford, June 2001.

[20] Pete Hardee " Getting Hardware and Software to Speak the Same Language". Dedicated Systems Magazine pp 6-9, July 2001.

[21] S. Benedetto and G. Montorsi, "Design of parallel concatenated convolutional codes" IEEE Trans. Commun., vol. 44, pp. 591-600, May 1996.

[22] Steve Holloway, David Long, Alan Fitch, "From Algorithmic to SOC with SystemC and Cocentric System Studio", Synopsys Users Group (SNUG) Europe 7.,8 Mars 2002.