RENPAR'15 / CFSE'3 / SympAAA'2003
La Colle sur Loup, France, 15 au 17 octobre 2003

Spécification et conception de systèmes sur puce avec SystemC
Etude de cas : le turbo-codage

Résumé
Actuellement, les systèmes deviennent de plus en plus complexes et la tendance est à l'intégration des parties logicielle et matérielle sur une même puce (System On Chip : SoC). Afin de faciliter la conception d'un système, l'utilisation d'un langage dit «système» pour la conception de haut niveau est nécessaire. Dans cet article, nous traitons d'une approche de spécification permettant le raffinement progressif avec un langage unifié : SystemC. L'objectif est d'évaluer la capacité de ce langage à modéliser différents niveaux d'abstraction. Cette approche est expertisée à travers une étude de cas du domaine des télécommunications : la conception d'un Turbo-Codeur. La spécification est progressivement raffinée depuis le niveau fonctionnel vers une spécification au cycle près. Nous présentons la démarche de spécification, de raffinement et de validation avec SystemC ainsi que les performances de simulation. Ce travail a permis d'analyser le potentiel de SystemC dans le cadre de la conception matérielle des systèmes intégrés.

Mots-clés : Conception de systèmes, SoC, langage de haut niveau, SystemC, turbo-codes.

1) 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 (*time_to_Market*). La conception des SoCs est un nouveau concept qui exige typiquement la modélisation et l'intégration simultanée du logiciel et du matériel.
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 [1]. 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 [2,3,4,5]. 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 et/ou à un niveau d'abstraction qui ne fait pas d'hypothèse sur les cibles d'implantation. Dans un contexte de concurrence entre langages système, les approches basées sur des langages de type C/C++ tel SystemC sont particulièrement intéressantes. Dans cet article, la capacité de SystemC [6] à modéliser différents niveaux d'abstraction est évaluée à travers la conception et l'implémentation d'une fonction de codage canal utilisée en télécommunication, le turbo-codage.
L'article est construit comme suit : la section 2 analyse le besoin de la spécification système dans le flot de conception et résume différentes approches pour la mise en œuvre d'un langage de modélisation système. La troisième section s'intéresse à l'approche associée au langage SystemC (principales caractéristiques, raffinements élémentaires) dans le flot de conception matériel. La section 4 décrit les différents étapes de modélisation en SystemC d'un turbo-codeur. Les résultats et les performances obtenues sont présentés en section 5.

2) Langage Systèmes : initiatives et approches associées

Les systèmes devenant de plus en plus complexes, de nouvelles méthodes de conception sont nécessaires, dont entre autre la réutilisation (utilisation de composants virtuels) ou l'abstraction des spécifications. Dans ce contexte, disposer d'un langage de haut niveau permettant de traiter de manière unifiée plusieurs étapes de conception devient une nécessité. Comme le logiciel tend à devenir la partie la plus importante des systèmes électroniques, les efforts sont focalisés vers un langage autorisant des descriptions conjointes logicielle et matérielle. De plus, travailler au niveau système permet d'obtenir des modèles qui autorisent une meilleure gestion de la complexité des systèmes [7].

a) Nouveaux langages et méthodologie basées sur HDLs

L'Initiative SLDL (VHDL International's System-Level Design Language) [8], financée 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. 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.

Co-Design Automation, une des compagnies d'automatisation de conception électronique, propose le langage Superlog. Les auteurs de Superlog [9], estiment que ce langage est d'une part simple à lire et suffisamment performant pour résoudre les problèmes liés à un SoC. Toutefois, Superlog est centré sur la conception matérielle et présente quelques déficits au niveau de la conception de haut niveau.

OVI (Open Verilog Initiative) et IEEE ont travaillé pour améliorer Verilog depuis 1994 [10]. 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 ce qui a favorisé l'aspect Orienté Objet. 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. Néanmoins, pour les systèmes tendant à être dominés par le logiciel, l'adoption d'une méthodologie basée sur un langage HDL n'est pas vraiment bien appropriée.

b) 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 [5]. 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 gérer 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.

CynApps propose Cynlib [2]. C'est une source ouverte basée sur C++ pour modéliser le matériel. Il s'agit d'un ensemble de classes C++ qui implémentent des concepts existants dans les langages matériels tels que Verilog et VHDL. La limitation majeure de Cynlib est que la suite de CynApps vise plus spécialement la conception du matériel.

SpecC est une extension de ANSI-C [11]. L'approche SpecC vise à favoriser une méthodologie de Co-Design focalisée sur les IPs (Propriété Intellectuelle ou composants virtuels) pour la spécification, la modélisation et les systèmes embarqués au niveau système.

Actuellement, l'initiative SystemC semble être plus avantagée par rapport aux autres approches [12]:

- ✓ SystemC est supporté par les principales sociétés de la CAO et de l'industrie électronique et établi 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 partage.

✓ Tout comme SpecC, SystemC répond aux besoins techniques pour le matériel, le software et répond en ce sens aux attentes des concepteurs systèmes.

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

3) Approche 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 [13]. Il supporte la spécification de niveau RTL, de niveau comportemental et de niveau système. SystemC introduit de nouveaux concepts afin de supporter la modélisation et la description du matériel et ses caractéristiques inhérentes comme la concurrence et l'aspect temporel. Pour passer de la description au niveau système à l'implémentation, les différents niveaux d'abstraction doivent être bien définis et les méthodes de raffinement bien précises et claires.

a) Niveaux d'abstraction

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. Le flot complet, présenté dans la figure 1, fait appel au même langage du début à la fin. On peut donc, par raffinements successifs, passer d'un niveau donné au niveau suivant sans avoir à réaliser un changement de langage de modélisation qui pose quelques problèmes (sémantique différente, syntaxe différentes etc.), sans même parler du temps inutilement dépensé pour cela. On peut ainsi passer d'un niveau à un autre en manipulant la même sémantique, la même syntaxe, le même environnement de simulation, les mêmes benchmarks etc. Il y a donc de réels gains en ce qui concerne la vérification des raffinements réalisés.

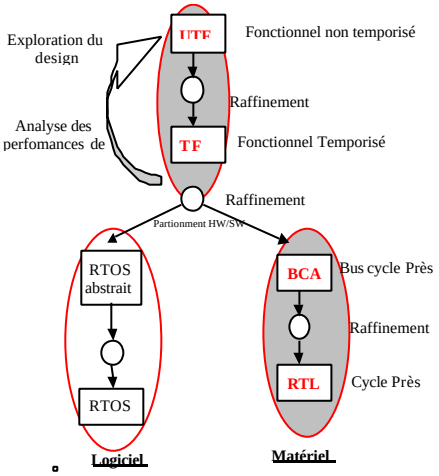


Fig. 1 : Flot de raffinement SystemC (source : OSCI)

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 les suivants:

Niveau fonctionnel non temporisé (FU) : L'objectif est de réaliser une validation des concepts de base du système, et d'exprimer le système sous forme de modules fonctionnels. On produit donc une spécification exécutable où l'architecture du système n'est pas encore définie.

Niveau Fonctionnel Temporisé (FT) : A ce niveau, on décompose le système en sous-systèmes et on impose des contraintes temporelles sur les modules identifiés (comportement temporel du système) mais la description reste non « clockée » (non cadencée par une référence temporelle). Une analyse grossière de performance du système peut ainsi être faite en vue du partitionnement.

Niveau Bus Cycle Près (BCA) : A ce niveau, la modélisation des communications entre les différents blocs du 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 FU.
Niveau Cycle Près (CA) : A ce niveau, tout le fonctionnement du système doit être maintenant spécifié au cycle près.

b) Méthodologie de raffinement avec SystemC

Pour pouvoir passer d'un niveau à un autre, il est nécessaire de réaliser un certain nombre de raffinements élémentaires. Typiquement pour le flot matériel, il s'agit du raffinement d'atomicité, du raffinement algorithmique, du raffinement des données et du raffinement des communications. L'ordre et le nombre répétition de chacune de ces étapes ne sont pas figés, cela dépend de la complexité et de la nature du système à traiter.
Raffinement d'atomicité (Rat) : Cette étape consiste à transformer un programme qui se déroule de manière séquentielle en un nouveau programme contenant des processus pouvant s'exécuter en parallèle.
Raffinement algorithmique (Ral): On procède ici au découpage des taches complexes en une séquence de taches de complexité inférieure (plus petites et plus simples). On procède à une division du bloc en sous blocs qui mènent chacun un fonctionnement interne ne dépendant que d'une liste de sensibilité.
Raffinement de communication (RC) : On transforme à ce niveau les moyens de communication abstraits ou les bus « primaires » en bus plus raffinés pour obtenir des modules plus réalistes.
Raffinement de données (RD) : Ce raffinement consiste à remplacer tous les types de données abstraits (C/C++) par des types de données directement implémentables sur une cible logicielle ou matérielle.

4) Expérimentation

Afin d'évaluer les capacités et les performances de SystemC, nous avons modélisé une fonction de codage de canal, les Turbo-codes. L'idée associée au concept de codage canal est de protéger l'information émise contre les imperfections du canal de transmission (bruit, interférences, multi trajets) en ajoutant au message une certaine redondance. Les Turbo-Codes introduits en 1993 constituent une des techniques de codage correcteur d'erreurs la plus performante actuellement [14].

a) Turbo codeur

Le turbo codeur consiste en la concaténation parallèle de deux codes convolutifs systématiques récursifs (CRSs). Un entrelaceur introduit une permutation pseudo aléatoire sur l'ordre de la séquence de données à coder (figure 2).

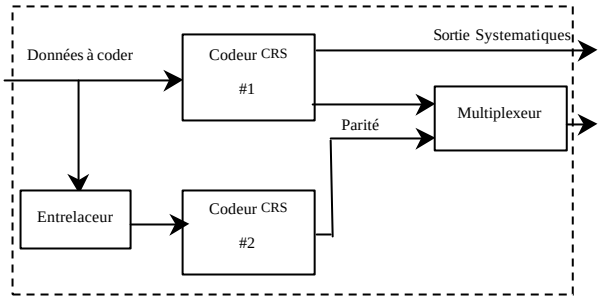
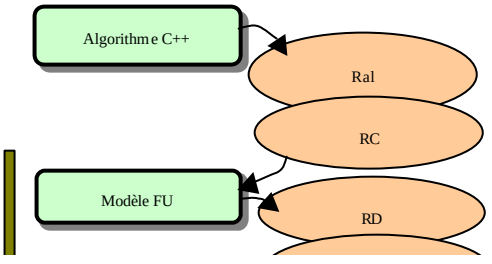


Fig. 2: Turbo-Codeur convolutif

Les bits d'entrées (données à coder) sont délivrés au premier sous codeur (CRS #1) qui génère un ensemble de bits systématiques et redondants. La séquence à coder est délivrée au deuxième sous codeur (CRS #2) dans un nouvel ordre produit par l'entrelaceur. Le CRS #2 génère à son tour un ensemble de bits systématiques et redondants. Pour ne pas transmettre inutilement deux redondances de bits systématiques, un masquage est opéré de façon à produire uniquement l'information systématique à partir du CRS #1 suivi des bits de redondances introduits respectivement par le premier et le second CRS. On obtient ainsi un codeur de rendement 1/3.



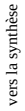


Fig. 3: Flot de raffinement matériel du turbo codeur

Lors de cette expérimentation, nous avons réalisé dans un premier temps la spécification en C/C++ de l'application indépendamment des propriétés de l'application en terme de temps, synchronisation etc. Cela nous a permis d'opérer plus facilement la vérification fonctionnelle. Nous avons ensuite raffiné la spécification du C/C++ en SystemC. Cela nous a permis d'obtenir une référence fonctionnelle pour les différentes étapes de raffinement à suivre en gardant les mêmes vecteurs de test. La figure 3 montre les primitives de raffinement dans le flot de raffinement du turbo codeur. Les détails qui concernent chaque étape de raffinement sont analysés dans les sous sections suivantes.

b) Modèle fonctionnel (FU)

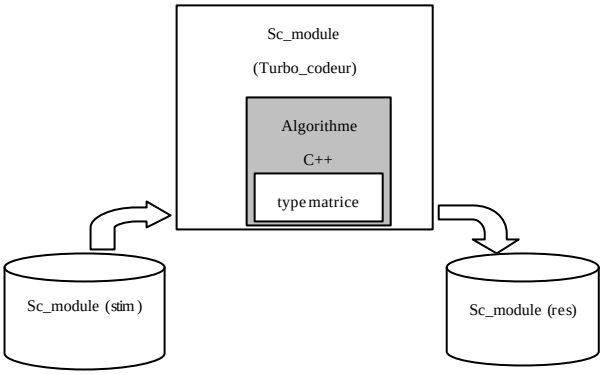


Fig. 4 : Structure du modèle FU du Turbo-Codeur

À ce stade du processus de raffinement, la version algorithmique C++ du turbo codeur est simplement placée dans un module conteneur (récipient) de SystemC. A l'intérieur du module, la fonctionnalité est placée dans le processus Turbo codeur. Les bits à coder sont fournis par le module stimulus (*sc_module stim*) et les résultats sont passés à un module résultat (*sc_module res*) (figure 4).
Le mécanisme de communication est réalisé entre ces modules à l'aide du canal primitif *sc_fifo* disponible dans la version 2.0 de SystemC. Le *sc_module stim* contient un processus *sc_thread* qui peut être suspendu et réactivé par les évènements ou par les signaux du module. Le module *res* quant à lui contient un processus de type *sc_method* (processus sensible aux entrées).

c) Modèle fonctionnel temporisé (FT)

i) V.FT1

Dans la version algorithmique, un type de données «*matrice* » est utilisé pour simplifier le traitement des données. On opère un raffinement de données pour omettre ce type de données et avoir ainsi un code sans variable globale ni pointeurs pour les allocations dynamiques de mémoire. Le système raffiné est validé avec les mêmes séquences test de bits que pour la version algorithmique.

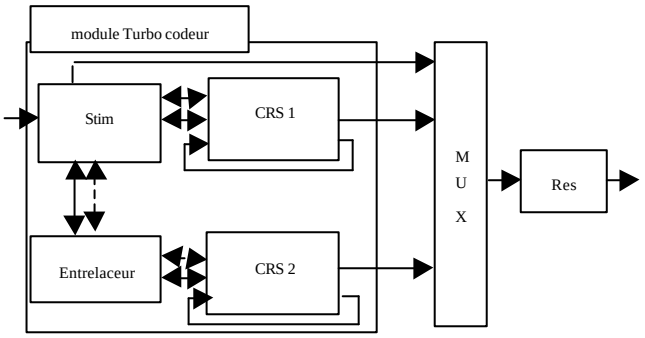


Fig. 5: Structure du modèle FT1 du Turbo-Codeur

On ajoute un niveau de hiérarchie au module Turbo_codeur qui instancie deux modules CRSs, un module entrelaceur et un module multiplexeur (figure 5). Chaque module englobe un processus de type *sc_thread* réalisant la fonctionnalité appropriée. On affecte chaque module d'un comportement synchrone en ce qui concerne sa communication en entrée ou en sortie.

ii) V.FT2

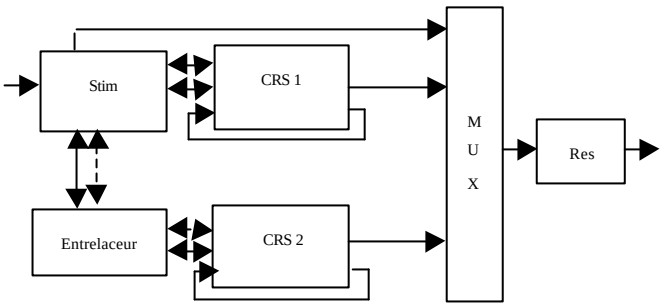


Fig. 6: Structure du modèle FT2 du Turbo-Codeur

Il n'y a aucun intérêt à utiliser une organisation hiérarchique pour cet exemple. La hiérarchie de module est utilisée quand il y a un mécanisme d'héritage reliant le module récipient aux sous modules. C'est pourquoi, dans cette version du niveau FT, on assure la séparation des modules ayant des fonctionnalités indépendant comme l'exige la primitive « raffinement d'atomicité ». Cette description est schématisée par la figure 6. Un mécanisme d'événements ajuste le lien entre les modules et donc des lignes de code sont ajoutées pour interfacer les sous modules. Les communications entre les modules de simulation (*stim, res*) sont modélisées en utilisant la canal primitif *sc_fifo*.

iii) V.FT3

On considère à ce stade le module CRS pour le raffiner. Une description statique du module CRS est opérée de manière à recevoir un seul bit comme donnée d'entrée. Une autre couche de hiérarchie est ajoutée suite à un raffinement d'atomicité : un module CRS instancie 3 modules modélisant le fonction logique XOR et deux autres modules modélisant la fonction mémorisation. Ces modules implémentent des processus de type *sc_method*.

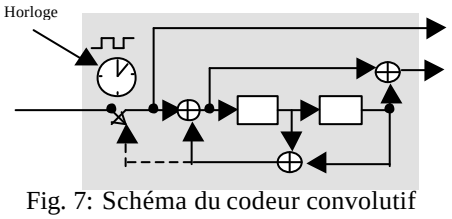


Fig. 7: Schéma du codeur convolutif

CRS est maintenant un module dans un niveau CA (cycle près). Les transformations de données sont cadencés par un signal d'horloge. La mise à jour des valeurs de données pendant l'exécution se fait suivant le front montant de l'horloge. Pour simuler tous les modules du Turbo_codeur, les processus (*do_entrelaceur*, *do_stim*, *do_res*) sont synchronisés via des opérations de conversion des types de données selon la nature des ports de communication et des signaux de type poignée de main.

5) Résultats

Les résultats présentés correspondent à un Turbo-codeur dont les CRSs ont comme matrice génératrice G = [1 1 1, 1 0 1] = [g0, g1]=$[7,5]_{octal}$. Les états des codeurs convolutifs sont initialement tous à zéros.

Pour évaluer les performances de simulation de SystemC, nous avons relevé les temps simulations associés à chaque étape de raffinement précédemment décrite avec les mêmes vecteurs de test. On observe en pratique de grandes différences en ce qui concerne les temps de simulation selon le niveau d'abstraction (figure 8) : les versions à haut niveau d'abstraction prennent très peu de temps par rapport à celles de niveau plus bas (FT 1,2,3).

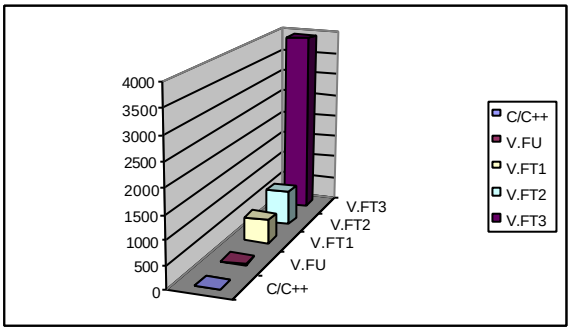


Fig. 8 Temps de simulation selon le niveau de raffinement (Pentium 3, 800MHz 128Mo, Linux Mandrake 8.2)

En fait, la majorité du temps est passée dans le simulateur de SystemC à gérer l'activation des processus ainsi qu'à identifier les processus éligibles. L'ordonnanceur est modélisé pour la simulation comme étant un « *thread* » actif; il communique avec un autre processus contenant le code des autres processus de la simulation [15]. Naturellement, plus on ajoute des détails, plus la simulation prend du temps. En particulier, le raffinement d'atomicité ralentit les simulations à cause de l'accroissement du nombre de processus ce qui explique la grande différence de temps entre les niveaux V.FT2 et V.FT3.

Le noyau de simulation associé à SystemC doit manipuler cette charge de travail complémentaire en plus d'un nombre accru d'événements. En outre, le processus du noyau communique avec les autres processus fils selon des fonctions internes dans le système d'exploitation employé: la réalisation de la simulation sous un environnement UNIX, Linux ou Windows NT conduit à un écart de temps différent selon les caractéristiques de l'environnement.

Notons enfin qu'avec les outils disponibles, il est difficile de quantifier de manière exacte le temps employé par l'ordonnanceur (scheduler) SystemC.

6) Conclusion

Le marché des systèmes matériels enfouis connaît une très forte croissance et le nombre de SoC produits ne cesse de croître. La complexité croissante des applications à intégrer requiert cependant de nouvelles techniques de conception. Actuellement, une des solutions les plus pertinentes réside en l'élévation du niveau d'abstraction et en l'utilisation d'un langage unifié. Le langage SystemC a été développé dans ce contexte. Au travers de la spécification d'une fonction de télécommunication, le turbo-codage, nous avons analysé une approche de spécification et raffinements progressifs associés au flot de conception matériel avec le langage SystemC. Le surcoût en temps de simulation selon le niveau de raffinement est également présenté.

La spécification de plus bas niveau du turbo-codeur, dite au cycle près, peut être synthétisée par un outil de synthèse du type *CoCentric SystemC Compiler* de *Synopsys* afin de générer une structure de type RTL que les outils de conception "conventionnels" sauront traiter. Cette étape est actuellement en cours de réalisation.

Bibliographie

1. Luc Séméria, "Applying pointer analysis to the synthesis of hardware from C", mémoire de thèse en génie électrique Université de Stanford, June 2001.
2. 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
3. Pete Hardee " Getting Matériel and Logiciel to Speak the Same Language". Dedicated Systems Magazine pp 6-9, July 2001.
4. Alan Fitch, "Application of SystemC to hw/sw Co-Design". IEE Seminar - Hardware-software Co-Design. December 2000.
5. 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.
6. Open SystemC Initiative (OSCI), www.systemc.org.
7. Santarini, Michael. “Million-gate ASICs will require hierarchical flow”, EE Times,
8. http://www.sldl.org/
9. http://www.superlog.org
10. http://www.eda.org/ovl/
11. http://www.SpecC.org
12. Pete Hardee " Getting Hardware and Software to Speak the Same Language". Dedicated Systems Magazine pp 6-9, July 2001.
13. Synopsys, SystemC version 2.0 User’s guide.2002
14. C.Berrou, A. Glavieux et P. Thitimajshima "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes", IEEE International Conference on Communications, ICC-93, May 1993, vol. 2, pp. 1064-1070.
15. Wolfgang Mueller, "The Simulation Semantics of SystemC", DATE 01, Munich, Germany, March 2001.