Conception d'interface pour processeur embarqué dans les systèmes sur puce

Issam MAALEJ, Guy GOGNIAT**, Mohamed ABID*, Jean Luc PHILIPPE***

* ECOLE NATIONALE D'INGENIEURS DE SFAX
DEPARTEMENT DE GENIE ELECTRIQUE, *B.P : W 3038* SFAX TUNISIE
Fax : 21674275595 , Tel : 0021674274088 / 0021674274409
** L.E.S.T.E.R., UNIVERSITE DE BRETAGNE SUD, FRANCE ; RUE SAINT MAUDE - 56100 LORIENT.
TEL : 02 97 87 45 60 – FAX : 02 97 87 45 00
Email *: issam.maalej@webmails.com*

Résumé : *Ce papier s'inscrit dans le cadre de l'intégration des systèmes sur puce. Il s'intéresse à la conception d'interface logicielle/matérielle afin de proposer au concepteur un modèle d'interface générique qui permet de faciliter son intégration. En premier lieu, nous introduisons la conception des systèmes embarqués monopuce. Une architecture cible autour du cœur de processeur "léon" associé à des accélérateurs matériels est ensuite présentée. En second lieu, nous proposons une approche de conception d'interface logicielle/matérielle permettant de faciliter l'intégration de la communication dans ces monopuces.*

1 Introduction

Les applications modernes sont de plus en plus complexes. Le temps de mise sur le marché (" time to market") doit être de plus en plus court. Chercheurs et industriels se sont focalisés sur les nouvelles méthodes et environnements de conception afin de faciliter la conception de tels systèmes et maîtriser leur complexité. En effet, dés les années 90, les chercheurs se sont concentrés sur l'innovation et la mise en œuvre d'une nouvelle méthodologie de conception : "co-design" [Abi98]. Toutefois, Jusqu'à la fin des années 90 la conception des systèmes dédiés n'est implémentée que sur cartes (circuits imprimés).

Aujourd'hui, le développement technologique des composants submicroniques avec un taux d'intégration supérieur à un million de transistors a ouvert la porte à l'implémentation de ces systèmes sur un seul Chip. Cette nouvelle génération de systèmes embarqués (SoC : "System on Chip") permet de maîtriser des problèmes encore plus complexes du fait qu'avec un seul chip on peut atteindre des performances (MIPS ou FLOPS) plus élevées que celles pouvant être réalisées avec une carte.

Une phase clé lors de la conception des SoC est l'intégration des blocs IPs ("Intellectual Properties") [Air00] ce qui reste une opération difficile et coûteuse en temps de conception. C'est pourquoi nous proposons la définition d'interfaces de communication génériques pour assurer la connexion entre les différents blocs IP et plus particulièrement les cœurs de processeurs avec les accélérateurs.

Peu de travaux se sont intéressés à ce problème de synthèse d'interface de communication. La plupart peuvent être classés en deux catégories. La première traite le problème pendant la synthèse haut niveau grâce à des outils ou des langages de spécification système comme COSMOS [Ben95], POLIS [Bal97], SystemC [Kje01] et CoWare [Ver98], qui permettent la génération automatique d'interfaces de communications. Cependant, les résultats restent souvent non optimums et ne répondent pas systématiquement aux contraintes exigées. Par ailleurs, ces outils ne permettent pas généralement la synthèse de bas niveau. La deuxième s'intéresse aux composants standards qui ont des protocoles incompatibles. L'IMEC propose aussi une solution de conception d'interface de communication matérielle [Lin97]. Cette solution est intéressante lorsque tous les IPs utilisés ont les mêmes protocoles de communications. [Lys02] utilise des "wrappers" afin d'assurer la communication entre les accélérateurs matériels et le bus. [Cha02] intègre une architecture micro-réseau dans son SoC afin d'assurer la communication entre les différent composants logiciels et matériels. Un effort de standardisation pour la communication VCI a été lancé par l'association VSIA [Vsi02] ("Virtual Socket Interface Alliance"). Cette norme annonce 3 standards pour la communication. Le "System Level InterFace documentation standard (SLIF)" et le "System-Level Data-Type standard" regroupent plusieurs transaction et message pour la description de la couche communication au niveau système. Le "On Chip Bus Virtual Component Interface (OCB VCI)" définie un ensemble de signaux assurant une interface générique pour la communication entre IPs et bus.

Pour la plupart de ces travaux les connexions sont soit points à points soit par un bus externe. Dans le cadre de ce travail, nous considérons la communication à travers le bus interne du processeur. Pour cela nous proposons une étude sur la synthèse des communications lors de l'assemblage de ce type de SoC. Cette étude concerne, actuellement, la communication entre un processeur et des accélérateurs matériels (plus généralement des IPs) pour un type d'application en traitement de signal. L'objectif étant de généraliser plus tard cette approche pour différents types d'IPs et de processeurs.

Ce papier est organisé comme suit : Dans la section 2, nous développons une formulation du problème. Dans la section 3, nous détaillons la méthode de conception des interfaces de communication. Dans la section 4, afin d'illustrer la mise en œuvre de l'interface, nous appliquons notre approche à une DCT 1D et une DCT 2D avec les bus AMBA AHB et PCI. Enfin, nous finissons par une conclusion ainsi que les travaux futurs.

2 Formulation du problème

A – Architecture et fonctionnement des SoCs

L'architecture du SoC que nous considérons est proposée dans la figure 1. Elle est décomposée en couches en vue de maîtriser la

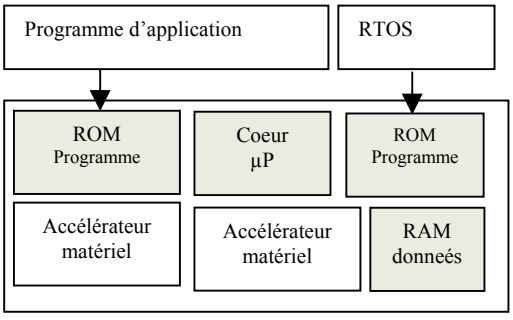

Figure 1 : Système intégré sur Puce (FPGA ou ASIC)

complexité pour la partie matérielle et pour la partie logicielle. On peut décomposer ces couches en deux principales. La première est formée par les RAMs, ROMs et le microprocesseur. Cette partie constitue la couche logicielle. La deuxième est la couche matérielle.

Le fonctionnement de ce type de système se décrit comme un ensemble de composants logiciels et matériels travaillant conjointement. La partie logicielle, s'exécutant sur un processeur embarqué, communique avec d'autres applications s'exécutant simultanément en matériel. Ces derniers sont intégrés sur la même puce que le processeur.

B – Interconnexion pour les SoCs

Comme l'architecture du SoC intègre plusieurs composants. Il faut assurer un ensemble d'interconnexions afin d'assembler le système complet. Actuellement, on distingue 3 approches : (1) Interconnexion par bus externe standard. (2) Connexion point à point. (3) Connexion à travers des FIFOs ou mémoires partagées.

En ce qui concerne l'architecture du Soc que nous avons adoptée, la connexion se fait par l'intermédiaire du bus interne du processeur. De ce fait, il s'agit plus d'intégrer des composants matériels au cœur de processeur que d'assembler un SoC habituel. Le principe de fonctionnement global ressemble plus à celui d'un ASIP.

C – Présentation du processeur

L'architecture se base sur la conception d'une application embarquée sur une seule puce autour du processeur LEON [Gai00]. Ce processeur est un processeur SPARC compatible dédié pour les applications embarquées. Il comporte un cache séparé en deux couches instructions et données, le bus AMBA AHB et APB [Amb02], UARTs, "timers", contrôleur d'interruption, un port d'entrés/sorite 16-bits, et une interface pour le coprocesseur "Meiko floating-point unit".

Pour la communication des données, le processeur LEON utilise le bus interne AMBA (Advanced Microcontroller Bus Architecture) qui est structuré en deux BUS. Le premier est le bus AHB (Advanced High-performance Bus). C'est un bus interne, conçue pour les transferts hauts débit du processeur. Il est équipé de plusieurs protocoles permettant un maximum d'optimisation au niveau temps de transfert. Mais ces protocoles, rendent plus compliqués la connexion des périphériques avec ce bus. Le bus APB (Advanced Périphéral Bus) est spécialisé pour la communication avec des périphériques. Sans protocole compliqué, il présente une interface très simple à utiliser. Toutefois, il présente des débits de transferts modestes. Le processeur est aussi équipé d'un bus d'entées/sorties 32 bits qui est le bus PCI.

3 Conception d'interface de communication

A – Motivation

Afin d'aider le concepteur à concevoir la communication entre le processeur « léon » et le(s) accélérateur(s), et lui éviter de définir et raffiner lui-même les connexions des bus, des pines etc., nous proposons un modèle permettant à chaque accélérateur de se connecter au processeur par configuration. Ce modèle intègre une interface qui est connectée en premier lieu avec l'IP (figure 2-a). Ensuite, tout le bloc accélérateur matériel et interface sont associés au reste du system on chip (figure 2-b) à travers le bus AMBA. Dans ce cas, lors de l'intégration sur le bus AMBA, la connexion avec le processeur devient ainsi plus simple.

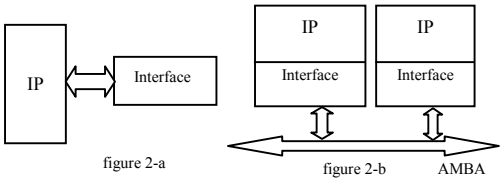

Figure 2 : Mécanisme d'interface pour la communication

Cette approche permet donc au concepteur de simplifier la phase d'intégration. L'interface est considérée d'un coté comme un esclave du processeur sur le bus AMBA ; et de l'autre côté, elle est considérée comme maître de l'accélérateur matériel. Ainsi, l'interface permet la séparation entre l'IP et le processeur. L'interface assure la gestion entre le processeur et chaque IP individuellement. Elle assure donc l'adaptation du protocole de communication du processeur en fonction des caractéristiques des IPs. Dans la suite de cette section, nous proposons des solutions afin de résoudre les points clés du modèle d'interface à savoir : l'adaptation du protocole de communication, la sélection et la configuration.

B – Principe de communication

En général, la communication entre un processeur et les périphériques nécessite la mise en œuvre de plusieurs protocoles et gestionnaires matériels et logiciels selon le besoin.

La plupart du temps, le processeur ne commande pas directement les périphériques. Il utilise pour cela un circuit spécialement adapté, un « contrôleur », de caractéristiques propres à chaque périphérique. Les commandes du processeur vers le périphérique s'opère alors par l'intermédiaire de ce contrôleur. Ces commandes correspondent directement aux mécanismes physiques du périphérique tels, par exemple, que le déplacement du bras d'un lecteur de disque. Par ailleurs, les contrôleurs sont souvent d'une grande complexité électronique.

On relie les contrôleurs au bus du processeur et on leur alloue, à chacun, un certain nombre d'adresses. À ces différentes adresses, on émet des commandes de pilotage ou on effectue des entrées-sorties de données. Le contrôleur reconnaît ses adresses grâce à une logique de décodage. Dans certains systèmes, les contrôleurs ne sont pas reliés au bus mais à des voies d'entrées-sorties spécifiques.

Les contrôleurs et le processeur opèrent en parallèle. Le contrôleur dispose d'une mémoire tampon, pour lui permettre, par exemple, de lire des données en provenance du périphérique d'entrée pendant que le processeur traite une autre tâche.

C – Processus de sélection

Comme pour les contrôleurs, on a alloué, à chaque accélérateur, un certain nombre d'adresses. La sélection est prise en charge par l'interface de communication. Cette dernière est chargée de comparer à chaque fois l'adresse délivrée par le processeur LEON avec l'adresse réservée à l'accélérateur. Ce qui est équivalent à réserver un comparateur 32 bits. Comme on peut utiliser autant d'interfaces que d'accélérateurs, on est amené à associer à chaque interface un comparateur 32 bits.

Afin d'optimiser au niveau temps et surface, on a ajouté un

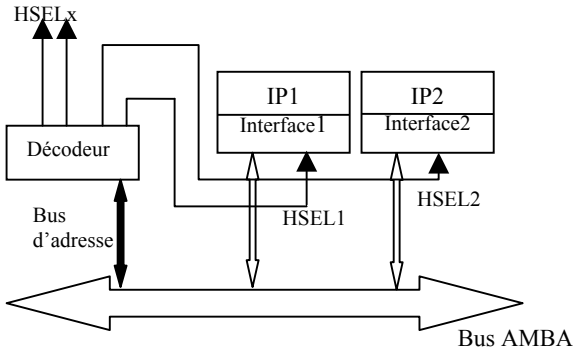

Figure 3 : le décodeur génère les signaux de sélections pour l'interface.

décodeur (voir figure 3). Ce décodeur se charge de la génération des signaux de sélection HSELX (X symbolise le numéro du signal). Ainsi, l'interface se chargera seulement de détecter la mise à '1' du signal de sélection. Le décodeur réserve des plages d'adresses à partir d'une adresse initiale et en fonction de la taille de plage choisie.

D – Principe de l'interface de communication

Le principal objectif de l'interface est donc d'adapter le protocole de communication de l'IP à celui du processeur LEON. Le bus AMBA AHB du LEON admet plusieurs protocoles. Ces protocoles sont contrôlés par le maître. En ce qui nous concerne le bus AMBA est configuré en maître. Donc ces protocoles sont imposés par le bus aux esclaves. Ce qui induit, que l'interface doit décoder le type de configuration imposé par ce protocole afin d'assurer un transfert de données correct. Pour cela, nous nous somme basés sur le chronogramme (figure 4) de transfert simple des données. Ensuite, nous nous somme basés sur les protocoles clés qui font la différence de transfert au moment de l'exécution. Ce qui est le cas pour le Hsize qui nous indique la taille du

transfert qui peut varier de 8 bits jusqu'à la taille maximale du bus qui est de 32 bits pour notre cas et qui peut arriver jusqu'à 1024 bits pour d'autres processeurs utilisant le même bus AMBA. Pour l'exemple de la DCT on utilise un transfert de 64 bits (2*32). Dans la suite on impose une taille de 32 bits ce qui est équivalent à Hsize = "010 ".

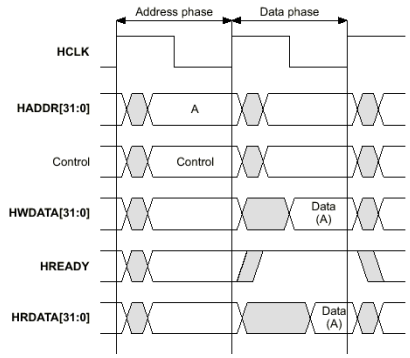

Figure 4 : Chronogramme de transferts simples

E - Modélisation de l'interface

L'interface est considérée comme un esclave pour le bus AHB, ce qui conduit à utiliser les signaux d'entrées/sorties de l'unité AHB SLAVE. Le signal HMASTER[3:0] n'est pas pris en compte puisqu'il n'y a pas plusieurs maîtres sur le bus. En effet, le processeur LEON est le seul maître du système.

L'entité correspondante à cette interface est nommée AHB. Elle est modélisée par la figure 5 :

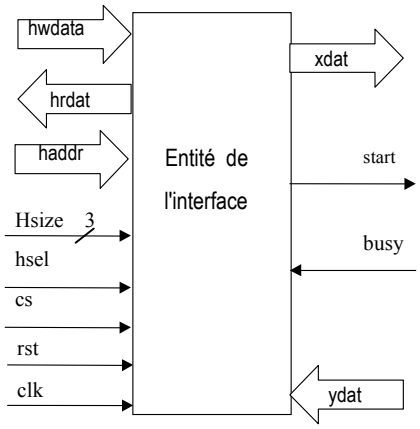

Figure 5 : Interface entre la DCT et le bus AHB

L'entité AHB reçoit les signaux CS et HSEL qui indiquent qu'elle est sélectionnée. Une fois sélectionnée, elle exécute le transfert des données. Une fois le transfert terminé, l'interface fournit à l'accélérateur un signal qui déclenche le calcul de l'accélérateur (start). Lorsque le calcul de la DCT est terminé (indiqué par busy à l'état bas), l'interface récupère le résultat de la DCT sur le bus de donnée du bus AHB.

F – Description de l'interface

L'interface initialement décrite pour la communication de la DCT avec le bus AMBA AHB est ensuite étendue pour assurer la communication de façon plus générique. Nous avons donc défini une description plus générale de l'interface.

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 "*clk"*. Chaque processus est contrôlé par des conditions propres à son fonctionnement. Le premier processus constitue la procédure de lancement du transfert des données. Il détecte le début de l'opération de transfert dés que toutes les conditions de fonctionnement sont validées. Ce processus a pour fonction de valider les opérations de transferts en écriture et en lecture. Ces deux opérations constituent successivement le deuxième et le troisième processus. Les conditions de fonctionnement de ces processus sont commandés uniquement par le premier processus, le quatrième et le cinquième processus qui correspondent successivement aux processus de réinitialisation d'écriture et de lecture.

En résumé, on a un fichier pour la déclaration des types et des signaux organisé ainsi :

```
Library ……..
package nom_bus is
type slv_in_type is record   ……….    end record ;/* type d'entrée
type interface_type is array ( 0  to k ) of slv_in_type ;/* type de sortie
type SIGNAL_SLV_IN_TYPE is record
…  /* Signaux d'entrées venant du bus du processeur
end record ;
type Signal_SLV_OUT_TYPE is record
…  /* Signaux de sorties vers le bus du processeur
end record ;
end ;
```

Le fonctionnement de l'interface est décrit comme ci-dessous :

```
Library ……..
use work.nom_bus.all ; /* Appel du package
entity Bus is
port(
cs : in std_logic;
clk : in std_logic;
rst : in std_logic;
BUSI : in SIGNAL_SLV_IN_TYPE ;
ydata : in interface_type;
BUSO : out SIGNAL_SLV_OUT_TYPE ;
start : out std_logic ;
xdata : out interface_type);
end ahb ;
--------
architecture …... is
…
Process (clk)
begin
-- Processus lancement --
if Condition de transert= vraie   then
                    (Read adress)
                    if ( c'est une écriture ) then write <= '1';
                                        else read <= '1'; end if;
                    else write <= '0'; read <= '0'; end if;
--Processus Ecriture --
if ( write ='1') and (address = int_waddr) and (rop ='0')
        then (écriture ...) ;end if;
--Processus lecture --
…
--Réinitialisation écriture --
if (int_waddr  = "Adresse de fin de transfert") and (write ='1')
then        wop <='0';
            int_waddr <= " Adresse de debut de transfert ";
            start<='1'; end if;
--Réinitialisation lecture --
…
End process;
End architecture;
```

Comme on peut le remarquer seul le premier processus dépend des signaux du bus du processeur (contrôles, adresse).

4 Applications

L'interface décrite dans le chapitre précédent est assez générique pour pouvoir assurer la communication entre plusieurs types de bus et d'accélérateurs à grain faible. Pour valider cette approche nous avons réalisé les expériences décrites ci-après.

A – Interface AHB – DCT 1D

Cette application consiste en la réalisation de la communication du bus AMBA AHB du processeur LEON et une DCT. Le bus présente une largeur de donnée de 32 bits. La DCT fait le traitement de vecteurs de 64 bits. Ce qui se traduit par une configuration des deux premiers types du package comme suit :

```
type slv_in_type is record
 vec : std_logic_vector (31 downto 0 );      end record ;
 type interface_type is array ( 0  to 1 ) of slv_in_type ;/* 2x32=64bits
```

Ensuite l'identification des signaux du bus AHB sont enregistrés dans les deux autres types du packages.

```
type SIGNAL_SLV_IN_TYPE is record
hsel :  std_logic;
hwrite :  std_logic ;
hsize :  std_logic_vector(2 downto 0);
hburst :  std_logic_vector(2 downto 0);
htrans :  std_logic_vector(1 downto 0) ;
hwdata : slv_in_type;
haddr :  std_logic_vector(HAMAX - 1 downto 0) ;
end record ;
type AHB_SLV_OUT_TYPE is record
hready : std_logic ;
hrdata : slv_in_type;
end record ;
```

Une fois l'identité de l'interface mise à jour il suffit d'insérer dans le premier processus les conditions de mise en marche du transfert.

```
if ( cs='1') then    address <= AHBI.haddr;
                     if ( AHBI.hwrite = '1' ) then write <= '1';
                                              else read <= '1'; end if;
                    start<='0';
                    else write <= '0'; read <= '0'; end if;
```

B – Interface PCI – DCT 1D

En deuxième lieu nous avons réalisé la communication du bus PCI externe du processeur LEON et une DCT. Le bus présente lui aussi une largeur de donnée de 32 bits. Donc, par rapport à la configuration il nous faut définir les signaux du bus PCI dans les deux derniers types. Ensuite, il faut redéfinir le premier processus.

```
if ( cs='1') and (pci_frame_n = '0') and (pci_irdy_n ='0')
          then    address <= pci_ad;
                  if (pci_cbe_n = "0011" ) then write <= '1';
                                           else write <= '0'; end if;
           if (pci_cbe_n = "0010" ) then read <= '1';
                                    else read <= '0'; end if;
          start<='0';
```

Cette application a permis de valider la possibilité de généraliser l'interface de communication conçue pour pouvoir l'utiliser pour différents bus.

C – Interface AHB – DCT 2D

Le modèle d'interface que nous avons modélisé permet la communication avec d'autres accélérateurs matériels. Pour valider cette approche nous avons utilisé cette même interface pour assurer la communication du processeur LEON avec une DCT 2D. En fait, la DCT 1D fait le calcul d'un vecteur de 8 pixels, donc de 64 bits. La DCT 2D fait le calcul par bloc. Ce dernier, est une matrice de 8 x 8 pixels ce qui est équivlant à 512 bits. Donc la DCT 2D fait le calcul de 512 bits et transmet comme sortie également 512 bits.

Ainsi, le fonctionnement du hardware est décrit comme suit. Le processeur envoi les premiers 512 bits. Ce qui est équivalent à 16 mots de 32 bits. Une fois les 16 mots envoyés, la DCT 2D reçoit le signal START pour commencer le calcul. La fin de calcul est toujours assurée par une mise à zéro du signal BUSY.

Par rapport à la DCT 1D, la DCT 2D a une taille de bus de donnée de 512 bits. Ce qui ce traduit dans le package seulement par le changement du deuxième type.

```
 type interface_type is array ( 0  to 15) of slv_in_type ;/* 16x32=512bits
```

Ce changement par rapport à l'interface avec la DCT 1D est suffisant pour assurer la configuration de l'interface afin de garantir la communication sur le bus AMBA AHB entre le processeur LEON et la DCT 2D.

5 Conclusions et perspectives

Dans cet article nous proposons un modèle d'interface permettant le raffinement de la communication entre le processeur LEON et différents accélérateurs matériels. Cette approche permet au concepteur de simplifier la phase d'intégration. Ce modèle est suffisamment générique afin de pouvoir s'adapter aux différents protocoles de communication.

L'approche proposé est efficace jusqu'à maintenant pour les accélérateurs matériels à grain faible. Elle permet la communication avec tout les IPs qui ont des signaux de contrôle de type BUSY et START. L'élaboration de cette interface entre dans le cadre de la conception de communication génériques avec les IPs. Aussi, ce travail va être étendu pour assurer la communication avec d'autres types d'IPs.

En effet, ce travail s'intègre dans un projet de conception de système sur puce à base de processeur LEON. Ce dernier intègre deux types de bus. Le bus AHB qui est un bus interne et le bus PCI qui est un bus pour la communication externe. Toutefois, ce travail est extensible pour d'autres processeurs comme le processeur ARM qui utilise le même bus AMBA. Par ailleurs, il est valable pour d'autres types comme le bus PCI. Cette approche peut être extensible pour d'autres types de bus vue la description générique qu'elle présente.

D'autres applications utilisant cette approche sont actuellement en cours d'études. L'objectif est de dégager les points clés de la conception d'interface de communication afin de proposer une méthode générale et automatique de synthèse des communications de système sur puce.

Références

[Abi98] : Abid M. , "Hardware/Software Co-design méthodology for design of embedded system" Integrated Computer Aided Engineering Journal, Vol. 5, pp69-83,1998.

[Air00] : Airiau R., Carer A., Casseau E., Martin E., O.Sentieys, "Méthodologies de conception des composants virtuels pour les applications de TDSI". Janvier 2000.

[Amb02] : AMBA Specification (Rev2.0), IHI0011A.pdf du site http://www.arm.com/

[Bal97] : Balarin F., Chiodo M., Lavagno L., Passerone C., Sangiovanni-Vincentelli A., Sentovich E., Suziki K., Tabbra B., Hardware-Software Co-Design of embedded system : The Polis Approach, Kluwer 1997.

[Gai00] : Gaisler Jiri " LEON-1 Processor – First Evaluation Results" http://www.gaisler.com

[Kje01] : Kjetil Svarstad, Gabriela Nicolescu, Ahmed A. Jerraya "A model for describing communication between aggregate objects in the specification and design of embedded systems", DATE 2001.

[Lin97] : Bill Lin, Vercauteren Steven, Hardware/Software Communication and system Integration For Embedded Archtectures.

[Lys02] : Lysecky Roman, Vahid Frank "Prefetching for improve bus wrapper performance in cores" ACM-Transaction 2002

[Ver98] : Vercauteren S., 'Hardware/Software co-design of application-spécific Heteregeneous architectures', IMEC, rapport de these, Déc. 1998.

[Vsi02] : Virtual Socket Inetrface Alliance, http://www.vsi.org.