Conception en vue de Test pour l'Architecture d'un Réseau sur Puce Asynchrone

Xuan-Tu TRAN, Jean DURUPT,
et François BERTRAND
CEA/LETI
17, rue des Martyrs, 38054 GRENOBLE

Vincent BEROULLE,
et Chantal ROBACH,
INPG/LCIS
50, rue Barthlmy de Laffemas, 26902 VALENCE

E-mail: Xuan-Tu.Tran@cea.fr

Résumé

A cause de la complexité de beaucoup d'applications et de l'intégration, les concepteurs embarquent de plus en plus de ressources de calcul (i.e., IPs) dans un système sur puce. Cependant, ceci rend le test de fabrication de ces systèmes plus difficile, notamment pour les systèmes sur puce à base de réseaux sur puce asynchrone. L'objectif de cet article est de proposer une architecture DFT ("Design for Test") innovante pour ces systèmes sur puce. Cette architecture est modulaire, générique, dimensionnable, configurable. Elle est mise en œuvre en logique asynchrone pour bien s'adapter à la plateforme GALS (Globalement Asynchrone, Localement Synchrone). Plusieurs résultats et conclusions seront présentés.

1. Introduction

A cause de la difficulté croissante de la distribution d'horloge sur une puce de grande dimension, les concepteurs de SoCs se sont intéressés à la logique asynchrone. La feuille de route de l'ITRS prédit que l'approche GALS (Globalement Asynchrone - Localement Synchrone) deviendra la tendance principale dans un proche avenir. Dans un système GALS, un certain nombre d'îles synchrones communiquent de façon asynchone entre elles en employant un réseau de communication asynchrone. En conséquence, le paradigme de réseaux sur puce (NoCs) asynchrones est devenu une solution alternative pour gérer la communication sur les systèmes sur puce grâce à ses avantages : bonne efficacité, dimensionnalité, polyvalence, et fort débit. Quelques NoCs asynchrones sont proposés comme les architectures CHAIN [1], NEXUS [2]. Néanmoins, la testabilité des blocs asynchrones est un gros problème dans la conception d'un système GALS. En d'autres termes, un des défis d'une conception des systèmes GALS est d'assurer le fonctionnement correct des réseaux de la communication asynchrones. Résoudre ce problème permettra aux systèmes GALS de devenir réalisables.

Malheureusement, il est très difficile de tester des circuits asynchrones en raison de nombreuses boucles de rétroaction (feedback loops). Un seul défaut peut causer le blocage du circuit entier. Quelques travaux sur le test des circuits asynchrones sont proposés dans [3]. Dans ces propositions, le test des circuits asynchrones est fait en insérant des chaînes de scan (scan-latches) pour casser les boucles de rétroaction. Ces travaux sont peu intéressant au niveau du coût du test si le circuit a de nombreuses portes de Muller[1].

Le CEA-LETI a proposé une architecture de réseaux sur puce asynchrone (ANoC) [4], adaptée aux systèmes GALS, fournissant une latence faible pour la qualité du service (QoS). Elle est mise en œuvre en utilisant le style de conception de type QDI (quasi-insensible aux délais). Pour pouvoir utiliser cette architecture dans les applications industrielles, une méthodologie de test s'avère nécessaire.

Dans cet article, nous présentons une méthodologie innovante de DFT pour l'architecture ANoC. Une architecture générique, modulaire, dimensionnable et configurable est modélisée en logique asynchrone. Cette architecture convient à la conception d'ANoC et est réutilisable pour les prochaines conceptions.

L'organisation de l'article est la suivante : La section 2 présente brièvement le contexte et les objectifs de notre conception ; La section 3 rappelle l'architecture ANoC, pour laquelle nous développons une architecture de DFT ; Cette architecture de DFT est présentée dans la section 4 ; La section 5 présente la conception et la validation de cette architecture ; Et finalement, quelques résultats et conclusions sont donnés dans la section 6.

2. Tester un système à base de réseaux sur puce

Beaucoup d'architectures de NOC sont proposées [5, 6, 7, 8] avec de nombreuses topologies et méthodologies de conception, mais il y a peu de travaux sur le test.

La stratégie de test pour les systèmes à base de réseaux sur puce posent trois problèmes : tester les IPs (i.e., les ressources de calcul) et leurs interfaces de réseau correspondantes ; tester l'infrastructure d'interconnexions elle-même (les nœuds et les interconnexions entre les nœuds) ; et tester le système intégré entier.

Pour tester les ressources embarquées et leurs interfaces de réseau correspondantes, il faut des mécanismes d'accès pour le test (TAM : Test Access Mechanism) afin de transporter les vecteurs de test de la source aux ressources sous test et des résultats de test des ressources sous test à l'extérieur. Quelques recherches proposent de réutiliser le

[1]La porte de Muller est une fonction séquentielle qui recopie la valeur de l'entrée sur sa sortie losque ses entrées sont égales.

réseau sur puce comme un TAM de grande bande-passante pour les ressources embarquées [9, 10]. Cette proposition ne nécessite pas de coût supplémentaire de matériel pour le TAM et rend possible le test des ressources en parallèle.

Pour tester l'infrastructure d'interconnexions, nous devons considérer deux aspects : le test des nœuds et le test des interconnexions entre les nœuds. Puisque les nœuds se composent de FIFOs et de logique de routage, des travaux proposent de tester ces pièces séparément, en utilisant un BIST pour les FIFOs et une méthode traditionnelle pour les logiques de routage. Cependant, les FIFOs sont distribuées partout dans la puce et donc posent un grand défi pour cette approche. Nous préférons tester le nœud entier avec une architecture unique de DFT. Cette approche devient plus favorable quand les nœuds du réseau sont conçus en logique asynchrone.

3. Rappel de l'Architecture ANoC

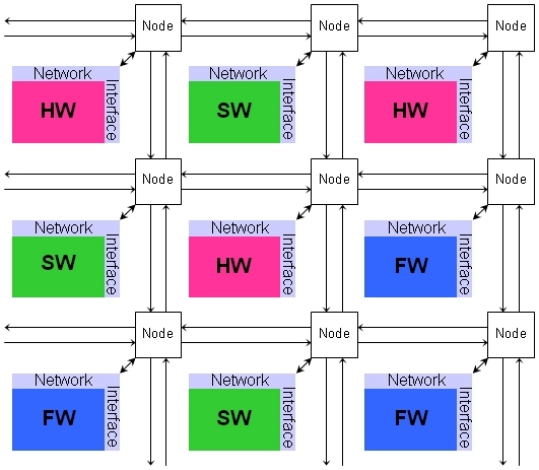

FIG. **1. L'architecture de reseaux sur puce.**

L'architecture ANoC développée par le CEA-LETI [4] se compose de nœuds asynchrones, de liens entre les nœuds, de ressources de calcul synchrones, et d'interfaces entre le réseau et les ressources, (cf. Figure 1). Les nœuds sont les éléments de base du réseau et ils ont habituellement 5 ports bi-directionnels qui sont reliés à quatre noeuds voisins et à la ressource de calcul synchrone la plus proche par une interface asynchrone/synchrone. Ils calculent où transmettre des données entrantes, arbitrent entre les données concurrentes potentielles, et transmettent finalement les données choisies aux liens choisis. Les liens entre les nœuds sont des chemins de données bi-directionnels de 32 bits, mis en œuvre avec un protocole de signalisation de type poignée de main au niveau flit[2]. L'interface de réseau est un bloc d'interface asynchrone/synchrone qui contient une interface GALS afin d'exécuter la synchronisation entre les domaines synchrones et asynchrones. Les ressources de calcul synchrones peuvent être des blocs durs (CPU, DSP, mémoires,...), des blocs génériques (FFT, MPEG,...), ou des blocs configurables (Décodeur, CDMA, OFDM,...).

[2]Un message est composé de plusieurs paquets, chaque paquet étant composé de plusieurs flits.

4. Une Architecture de DFT pour les NoCs Asynchrones

4.1. IEEE 1500 et le test des coeurs embarques

Pour tester les coeurs embarqués dans les SoCs, une architecture générale est proposée dans [11]. Dans cette architecture, le coeur embarqué est associé à une enveloppe de test pour en améliorer la contrôlabilité et l'observabilité. Les vecteurs de test sont transportés de la source au coeur sous test et les résultats de test sont transportés du coeur à l'extérieur par les TAMs.

Afin de rendre cette architecture réutilisable, une norme pour le test des coeurs embarqués a été développée, appellée IEEE Std. 1500 [12]. L'objectif de la norme IEEE 1500 est de développer une enveloppe de test pour le coeur embarqué et un langage de test. Les TAMs sont définis par les intégrateurs de système donc ils peuvent être en série, en parallèle (bus), ou en architecture de NoC. Les vecteurs de test et les résultats de test sont transportés autour du coeur par les cellules de l'enveloppe (wrapper) avec une horloge de test. Dans notre cas, il n'y a aucune horloge globale entre les différents éléments du réseau donc l'enveloppe IEEE 1500 ne convient pas. Par conséquent, nous présentons dans cet article une architecture innovante de DFT pour les NoCs asynchrones, appelée ANoC-TEST, décrite dans les sections suivantes.

4.2. Architecture ANoC-TEST

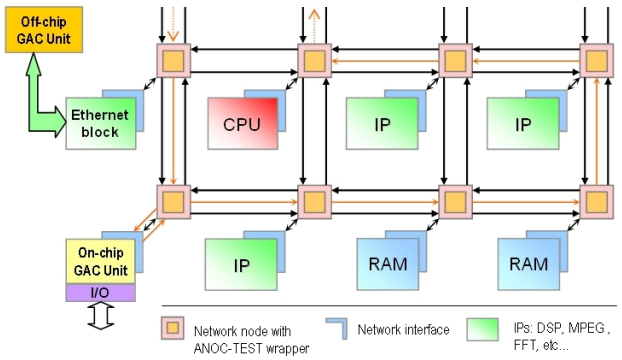

FIG. **2. Architecture generale ANoC-TEST.**

Dans notre architecture de test, les nœuds du réseau sont entourés par les enveloppes de test, qui sont mises en œuvre par les circuits asynchrones pour mieux s'adapter au paradigme GALS. Pour réduire le coût du TAM nous avons réutilisé les liens entre les nœuds pour construire un TAM avec une bande-passante élevée.

Pour générer les vecteurs de test, analyser les résultats de test et contrôler le flot de test, nous avons modélisé une unité multifonctions, appelé unité GAC (Générateur, Analyseur, et Contrôleur). Cette unité peut être mise en œuvre sur la puce comme une IP reliée à un nœud du réseau, ou hors de la puce comme un programme d'un ordinateur qui communique avec l'ANoC par les ports d'entrée/sortie ou les ports Ethernet de l'ANoC. L'architecture générale ANoC-TEST est présentée dans la Figure 2.

Dans cette architecture, les vecteurs de test générés par l'unité GAC sont transmis au noeud sous test par les interconnexions entre les nœuds, et sont appliqués au noeud par l'enveloppe de test. Les résultats sont ensuite extraits du noeud par l'intermédiaire de l'enveloppe, sont transmis à l'unité GAC par les interconnexions, et sont analysés par l'unité GAC.

4.3. L'enveloppe de test de l'ANoC-TEST

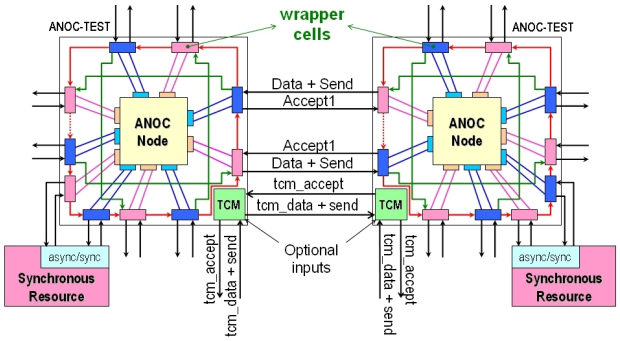

FIG. **3. Les connexions entre deux enveloppes de test.**

Puisque le nœud du réseau a 5 ports d'entrée/sortie, l'enveloppe de test se compose de 5 cellules d'entrée et de 5 cellules de sortie. Pour contrôler ces cellules d'entrée/sortie de l'enveloppe, un module de contrôle de test (TCM : Test Control Module) local est nécessaire. Le rôle de l'enveloppe est de transporter les vecteurs au nœud et de récupérer les résultats. Pour cela, le TCM doit contrôler les cellules d'entrée/sortie de l'enveloppe pour réaliser des opérations telles que la mise à jour des vecteurs, le décalage des vecteurs, le chargement des vecteurs, l'acquisition des résultats, le décalage des résultats, la transmission des résultats au TAM.

Pour améliorer la qualité du service (QoS), le CEA-LETI présente dans son architecture ANoC k canaux virtuels avec k niveaux de priorité. Tous ces niveaux de priorité sont arbitrés de telle sorte qu'un seul canal virtuel soit établi à la fois. Par conséquent, l'enveloppe de test ne doit pas apporter de changements aux valeurs de données, et à leurs niveaux de priorité. La conception des cellules d'entrée/sortie de l'enveloppe de test de l'ANoC-TEST et du module TCM est décrite dans [13]. Les connexions entre deux enveloppes de test sont décrites dans la Figure 3.

4.4. L'unite GAC et l'algorithme de test

L'unité GAC (Générateur, Analyseur, et Contrôleur) peut être mise en œuvre sur puce comme une IP, ou peut être mise en œuvre hors puce comme un programme s'exécutant sur un ordinateur. Si l'unité GAC est mise en œuvre sur puce, elle peut être modélisée comme un bloc synchrone relié au réseau de communication asynchrone par l'interface de réseau asynchrone/synchrone. Si l'unité GAC est mise en œuvre à l'extérieur, elle doit être reliée au réseau de la communication asynchrone par les ports d'entrée/sortie ou les ports Ethernet.

Le rôle de l'unité GAC est de générer les vecteurs de test pour les nœuds asynchrones et les ressources de calcul, de configurer les enveloppes de test par un canal série asynchrone de configuration, et d'analyser les résultats de test. Le flot de contrôle de l'unité GAC est établi par une machine d'état fini (FSM) et est décrit dans la Figure 4.

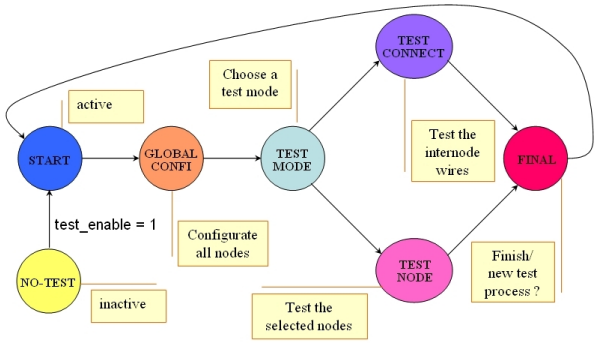

FIG. **4. Le flot de contrôle de l'unite GAC.**

5. Conception et Vérification

5.1. Synchronisation au niveau flit

La communication entre les enveloppes ou entre les enveloppes et les nœuds, aussi bien que la communication entre les cellules des enveloppes, est établie par un protocole, appellé protocole de poignée de main au niveau flit. Ce protocole utilise deux signaux "Send" et "Accept" qui synchronisent la communication entre deux blocs.

Pour mettre en œuvre le protocole de poignée de main au niveau flit pour k canaux virtuels dans le réseau, nous avons besoin de k signaux "Send" et k signaux "Accept" : $send < i >$ et $accept < i >$, où i varie de 0 à $k-1$. Et l'expéditeur peut envoyer un nouveau flit sur le canal virtuel i avec $send < i >= 1$, si et seulement si, le récepteur a indiqué $accept < i >= 1$ au cycle précédent. Avec ce protocole de "Send/Accept", les transactions de flit sont réalisées dans plusieurs canaux virtuels avec l'assurance qu'un canal physique est libre.

5.2. Conception des logiques asynchrones en QDI

Pour concevoir les enveloppes de l'ANoC-TEST, le style de conception QDI (quasi-insensible aux délais) [14] est utilisé. Nous avons utilisé un protocole de signalisation à 4-phase RTZ pour les canaux asynchrones. Afin de réduire la consommation d'énergie, la signalisation du code "1-of-4" est utilisée [15].

5.3. Modelisation des circuits asynchrones en SystemC

Tous les nœuds asynchrones et les enveloppes de test sont mis en œuvre dans le style des circuits asynchrones QDI, qui peuvent être modélisés en langage CHP [14], Tangram [16], ou Balsa [17]. Afin de modéliser la logique asynchrone dans l'environement SystemC TLM, nous avons utilisé les éléments de base "sc_fifo" de SystemC pour décrire le comportement de la logique asynchrone. Les éléments "sc_fifo" établissent une communication de blocage avec les opérateurs associés : $.read()$

et $.write()$. Grâce aux caractéristiques de classe du C++, nous pouvons modéliser les différents types de matériel (tels que le multi-rail) facilement. Pour modéliser les processus de type non-déterminisme comme dans le langage CHP, une fonction $.probe()$ est ajoutée aux éléments sc_fifo (cf. à [4] pour plus de détail sur la conception des circuits asynchrones en SystemC).

5.4. La plateforme de verification

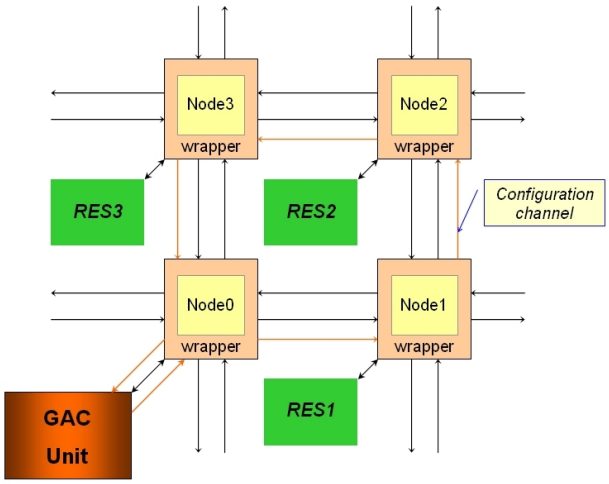

FIG. **5. Un testbench avec 4 noeuds.**

L'architecture DFT proposée, ANoC-TEST, a été validée par un modèle de test-bench avec 4 nœuds et ressources, illustrée dans la Figure 5. Tous les processus de test de ce modèle sont contrôlés par l'unité GAC qui est modélisée dans SystemC/C++. L'unité GAC génère également les vecteurs de test des nœuds asynchrones, des ressources de calcul, et des interconnexions entre les nœuds. Les résultats de test sont reçus et stockés dans un fichier de résultat de test et puis sont comparés aux vecteurs de test pour analyser les défauts des circuits. De plus, l'architecture de test peut se tester elle-même en se servant de configurations générées par l'unité de GAC.

6. Résultats et Conclusion

L'architecture de DFT proposée est une architecture générique, flexible, et configurable. Elle peut être configurée pour s'adapter aux architectures ANoC plus grandes. Elle peut ainsi être modifiée pour tester les noeuds et les ressources en parallèle. Comme tous les nœuds sont identiques, les vecteurs de test peuvent être réutilisés et les résultats de test peuvent être comparés entre eux ; c'est très utile en particulier pour le test des nœuds en parallèle.

Cette architecture de DFT est utiliée pour tester non seulement les nœuds du réseau et les interconnexions entre les nœuds, mais également les ressources de calcul (IPs) et les interfaces du réseau asynchrone/synchrone.

Dans cette conception, nous réutilisons les interconnexions entre les nœuds pour établir les mécanismes d'accès de test à bande-passante élevée. Ceci évite la congestion lors de la création du layout. Avec un temps de cycle de 4ns et une largeur de 32 bits, un chemin de test a un débit de 1Gbytes/s.

L'enveloppe de l'ANoC-TEST a été modélisée et validée en langage SystemC, comme décrit dans la section V. Ce langage permet de simuler les comportements des circuits asynchrones. Le coût en surface de cette enveloppe est évalué à environ 8 Kgates, rapporté à 4,5 Mgates de notre ANoC cible avec 20 noeuds, 23 IPs (sans mémoires).

Références

[1] J. Bainbridge and S. Furber. CHAIN : a Delay-Insensitive Chip Area Interconnect. *IEEE Micro*, Vol. 22(5) :16–23, Sept.-Oct. 2002.

[2] A. Lines. Asynchronous Interconnect for Synchronous SoC Design. *IEEE Micro*, Vol. 24(1) :32–41, Jan.-Feb. 2004.

[3] A. Efthymiou, J. Bainbridge, and D.A. Edwards. Adding Testability to an Asynchronous Interconnect for GALS SoC. In *Proc. of the 13th Asian Test Symposium*, Taiwan, Nov. 2004.

[4] E. Beigné, et al. An Asynchronous NoC Architecture Providing Low Latency Service and Its Multi-Level Design Framework. In *Proc. of ASYNC'05*, pp. 54–63, New York, USA, Mars 2005.

[5] W. J. Dally and B. Towless. Route Packets, not Wires : On-Chip Interconnection Networks. In *Proc. of DAC'01*, pp. 684–689, Las Vegas, NV, USA, June 2001.

[6] L. Benini and G. De Micheli. Networks on Chip : a New SoC Paradigm. *IEEE Computer*, Vol. 1, pp. 70–78, Jan. 2002.

[7] A. Jantsch and H. Tenhunen, *Networks on Chip*, Kluwer Academic Publisher, Feb. 2003.

[8] B. Vermeulent, et al. Bringing Communication Networks on a Chip : Test and Verification Implications. *IEEE Communication Magazine*, pp. 74–81, Sep. 2003.

[9] É. Cota, et al. Reusing an On-Chip Network for the Test of Core-Based Systems. *ACM Trans. on Design Automation of Electronic Systems*, Vol. 9(4) :471–499, Otc. 2004.

[10] M. Nahvi and A. Ivanov. Indirect Test Architecture for SoC Testing. *IEEE Trans. on Computer-Aided Design of Integrated Circuits & Systems*, Vol. 23(7) :1128–1142, July 2004.

[11] Y. Zorian. Testing Embedded-Core based System Chips. In *Proc. of the Int'l Test Conference (ITC'98)*, pp. 130–140, Washington, DC, USA, Oct. 1998.

[12] IEEE Std. 1500. *IEEE 1500 Standard for Embedded Core Test*. http ://grouper.ieee.org/groups/1500/.

[13] X.-T. Tran, J. Durupt, F. Bertrand, V. Beroulle, and C. Robach. A DFT Architecture for Asynchronous Networks-on-Chip, *The 11th European Test Symposium*, 2006.

[14] M. Renaudin, et al. ASPRO–216 : a Standard Cell QDI 16-bit RISC Asynchronous Microprocessor. In *Proc. of the ASYNC'98*, pp. 22–31, San Diego, USA, April 1998.

[15] J. Bainbridge, et al. Delay-Insensitive, Point-to-Point Interconnect using m-of-n Codes. In *Proc. of the Int'l Symposium on Asynchronous Circuits and Systems*, pp. 132–140, Vancouver, BC, Canada, May 2003.

[16] K. van Berkel, et al. The VLSI Programming Language Tangram and its Translation into Handshakes Circuits. *European Design Automation Conf.*, pp. 384–389, Amsterdam, Feb 1991.

[17] A. Bardsley, and D. Edwards. Compiling the Language Balsa to Delay–Insensitive Hardware. *Hardware Description Languages and their Applications*, pp. 89–91, April 1997.