THÈSE DE DOCTORAT DE L'UNIVERSITÉ PARIS VI

SPÉCIALITÉ INFORMATIQUE

Présentée par Mounir BENABDENBI

Pour obtenir le titre de

DOCTEUR DE L'UNIVERSITÉ PARIS VI

CONCEPTION EN VUE DU TEST DE SYSTÈMES INTÉGRÉS SUR SILICIUM (SOC)

Soutenue le 27 septembre 2002, devant le jury composé de

Mme Meryem MARZOUKI	Directrice de thèse
M. Christian LANDRAULT	Rapporteur
M. Paolo PRINETTO	Rapporteur
M. Alain GREINER	Examinateur
M. Laroussi BOUZAIDA	Examinateur

Résumé

Avec les progrès liés à la densité d'intégration et à l'utilisation de blocs préconçus, le test des systèmes intégrés sur silicium (SoC) doit faire face à de nouveaux problèmes et se trouve être un des facteurs limitant le progrès de l'industrie des semiconducteurs.

Les systèmes sur une puce ne pouvant être testés comme les systèmes sur carte, de nouvelles architectures de test doivent être développées. Dans cette thèse nous décrivons un mécanisme d'accès au test (TAM) des coeurs contenus dans un SoC. Ce mécanisme appelé CAS-BUS résout une partie des problèmes qui peuvent être rencontré lors du test des SoC. Ce TAM est paramétrable, flexible et dynamiquement reconfigurable. L'architecture CAS-BUS est contrôlée par les éléments du Boundary Scan et est compatible avec le standard IEEE P1500 tel qu'il est défini actuellement.

Cette architecture a été étendue sous deux formes différentes. La première extension permet le test de SoC contenant des coeurs munis de wrapper et des coeurs boundary scan. La seconde extension a été développée pour tenir compte du cas où le nombre de broches de test au niveau du SoC est limité. La solution proposée est basée sur une méthode de compression/décompression et expansion de données de test faiblement corrélés. Cette méthode permet une réduction du temps de test par rapport à une expansion simple.

L'architecture CAS-BUS et ses deux déclinaisons permettent à l'intégrateur système de faire des choix, des compromis pour optimiser le temps de test et la surface additionnelle nécessaire à son implémentation. Un ensemble de logiciels a été développé pour générer automatiquement les éléments des architectures. Quelques résultats expérimentaux sont présentés ainsi qu'une première évaluation de l'architecture appliquée à un circuit de référence.

Mots clés : SoC, testabilité, TAM, wrapper, P1500, IP.

Abstract

While geometry shrinking and design reuse allow impressive gains, System on a Chip (SoC) testing faces new set of problems and has become one of the bottlenecks of the IC industry progress.

As System on a Chip cannot be tested as System on a Board (SoB), some new test architectures must be developed. This thesis describes a Test Access Mechanism (TAM) named CAS-BUS that solves some of the new problems the test industry has to deal with. This TAM is scalable, flexible and dynamically reconfigurable. The CAS-BUS architecture is compatible with the IEEE P1500 standard proposal in its current state of development, and is controlled by Boundary Scan features.

This basic CAS-BUS architecture has been extended with two independent variants. The first extension has been designed in order to manage SoC made up with both wrapped cores and non wrapped cores with Boundary Scan features.

The second deals with a test pin expansion method in order to solve the I/O bandwidth problem. The proposed solution is based on a new compression/decompression mechanism which provides significant results in case of non correlated test patterns processing. This solution avoids TAM performance degradation.

These test architectures are based on the CAS-BUS TAM and allow trade-offs to optimize both test time and area overhead. A tool-box environment is provided, in order to automatically generate the needed components to build the chosen SoC test architecture. Some experimental results are presented for each architecture. A first evaluation of the CAS-BUS TAM applied to a SoC benchmark is also described in this document.

Keywords : SoC, DFT, TAM, wrapper, P1500, IP cores.

A ma famille et à Sarah...

Remerciements

Je tiens tout d'abord à exprimer ma reconnaissance à ma directrice de thèse Madame Meryem Marzouki chercheur au CNRS pour tous les précieux conseils qu'elle m'a donnés, pour la confiance qu'elle m'a témoigné et pour le temps qu'elle a consacré pour diriger cette thèse.

Je remercie le professeur Alain Greiner, directeur du département ASIM du LIP6, pour m'avoir accueilli dans son équipe et pour l'intérêt qu'il a porté à mes travaux.

Je remercie sincèrement M. Christian Landrault, directeur de recherche CNRS au LIRMM, et M. Paolo Prinetto, Professeur à l'université Polytechnique de Turin, d'avoir accepté d'être les rapporteurs de ma thèse de doctorat.

J'associe à ces remerciement M. Laroussi Bouzaida, responsable du département "Test Solutions" Central R&D à STmicroelectronics Crolles, et le professeur Alain Greiner, pour avoir accepté de faire partie de mon jury de thèse.

Je tiens à remercier Walid Maroufi, actuellement ingénieur chez Nortel à Ottawa, pour sa précieuse collaboration ainsi que les étudiants ayant participé au développement de ce travail, Keu Le Yin, Gilles Richard, Zakaryae Bekkouri et Pedro Correia.

Je remercie Laurent Ducerf-Baudot, Arnaud Caron et Fabrice Ilponse pour leurs conseils avisés et le reste de l'équipe ASIM avec laquelle ce fut un plaisir de travailler.

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Contexte de recherche

Tout le monde aura remarqué ces dernières années l'inéluctable course à la miniaturisation de tout appareil électronique. Plus petit, plus de fonctionnalités, plus rapide, plus d'autonomie... C'est ce qui est attendu par le grand public de tout nouveau produit issu de l'industrie des semiconducteurs. Ces progrès sont en grande partie dus à une capacité d'intégration de plus en plus élevée. Un circuit intégré peut contenir aujourd'hui quelques millions de transitors, voire quelques dizaines de millions, ceci étant sans commune mesure avec les dizaines de milliers d'il y a quelques années. Et cette tendance n'est pas prête de s'arrêter, au vu de l'étude délivrée par la SIA [SIA99].

Désormais il est possible d'intégrer un système complet sur une seule puce, système qui jusque là tenait sur une carte ou sur un MCM (Muti Chip Module). Ces systèmes intégrés sur une puce, appelés SoC (System on a Chip), se développent inévitablement et se retrouvent dans des domaines comme les télécommunications, l'avionique, la construction automobile...

Cependant, la façon de concevoir un SoC diffère désormais quelque peu de celle qui prévalait pour un circuit intégré classique. Cela est principalement dû à la pression du "Time To Market" qui mesure le temps nécessaire à la mise sur le marché du circuit, de sa conception à sa fabrication. Face à une concurrence exacerbée, une entreprise doit mettre son produit sur le marché la première si elle veut être rentable. Le "Time To Market" doit donc être réduit au maximum et cela a une conséquence directe sur le flot de conception du circuit. Un SoC est un circuit constitué de plusieurs millions de transistors, et même si les outils de CAO sont performants, les concepteurs ne peuvent plus se permettre de concevoir un système complet sans utiliser de briques de base. On se dirige alors vers une méthodologie de conception basée sur la réutilisation de matériel. Un SoC est constitué généralement d'un bus système et de différents éléments tels que processeur, DSP, RAM,

ROM.... Ces éléments se trouvent sous la forme de coeurs, aussi appelés "IP cores", et sont intégrés dans un SoC comme le sont les blocs logiques dans un ASIC.

Cette évolution du flot de conception implique une nouvelle répartition des tâches dans l'industrie VLSI. On distingue maintenant les entreprises qui fournissent les coeurs, d'où le terme IP (Intellectual Property) core, et celles qui les intègrent pour réaliser les circuits plus complexes que sont les SoC.

Cette nouvelle façon de concevoir, basée sur le "design reuse", a aussi un impact sur les méthodes de conception en vue du test des systèmes intégrés. On différencie désormais le test au niveau coeur du test au niveau système. Le nombre de transistors et la complexité des éléments qui le constituent impliquent que là aussi les méthodes de test doivent être révisées. Le test au niveau coeur est de la responsabilité du fournisseur d'IP et le test au niveau SoC de celle de l'intégrateur système.

Cependant, comme ce fut le cas à la fin des années 80 avec le test sur cartes et les standards IEEE 1149.* ([MO96]), l'industrie a besoin d'un certain nombre d'harmonisations, de normes et de standards. Actuellement deux groupes constitués d'entreprises et de chercheurs travaillent sur l'harmonisation des méthodes de conception de circuits à base de coeurs et à la définition de nouveaux standards. Il s'agit d'une part du consortium VSI Alliance [vsi] et d'autre part du groupe IEEE P1500 [p15] qui s'occupe plus particulièrement des problèmes liés au test. Le test des SoC n'en est encore qu'à ses balbutiements et déja de nombreux problèmes sont soulevés. Certains peuvent être résolus par la mise en place de nouveaux standards. D'autres nécessitent le développement de nouvelles architectures et de nouvelles méthodologies de test.

Vers une architecture de test des SoC

Le travail effectué dans cette thèse s'inscrit dans le cadre de la recherche de nouvelles architectures de test au niveau système. Nous nous sommes intéressés plus particulièrement aux mécanismes d'accès au test des coeurs, c'est à dire aux moyens nécessaires au transport et à l'application des données de test, des entrées/sorties du SoC vers les entrées/sorties des coeurs. Une architecture que l'on a appelé "CAS BUS" a été définie dans ce cadre de recherche. Cette architecture a évolué vers deux autres architectures qui répondent chacune à des besoins spécifiques.

Le premier chapitre définit les problèmes que nous avons choisi de résoudre tout au long de cette thèse.

Le deuxième chapitre fait un état de l'art du test au niveau système. Il décrit les différentes architectures liées au transport des données de test dans un système intégré.

Dans le troisième chapitre nous présentons l'architecture CAS BUS développée et les moyens mis en oeuvre pour gérer son contrôle. Cette architecture s'adresse aux systèmes intégrant des coeurs équipés d'une interface dédiée au test nommée "wrapper". Dans cette partie sera décrite une extension de cette architecture qui répond au problème du test des SoC qui contiennent à la fois des coeurs équipés de wrappers et des coeurs équipés d'éléments Boundary Scan [Boa90]. La présence de ces derniers dans un SoC nécessite un traitement particulier qui est étudié dans cette partie.

Il arrive parfois que le nombre de broches dédiées au test du circuit soit inférieur à la largeur du bus de test du système. Cela implique la mise en place de mécanismes d'expansion de données. Cette contrainte a pour effet d'augmenter le temps de test du circuit. Le quatrième chapitre de ce document présente une nouvelle méthode de compression/expansion/décompression de données qui permet de réduire ce temps additionnel. L'implémentation de cette méthode dans le cadre de l'architecture CAS BUS y est présentée.

Dans le cinquième chapitre, nous présentons les différents outils développés autour de ces architectures et les résultats obtenus.

Enfin en conclusion nous commenterons l'intérêt de ces différentes architectures et présenterons les perspectives de ce travail.

Chapitre 1

Problématique

Avant de s'intéresser aux mécanismes d'accès au test des coeurs dans les SoC, il serait utile de définir auparavant ce que sont les SoC et les IP cores. Les spécificités du test des SoC ([CP96], [ZMD98]) par rapport au test des systèmes sur cartes seront abordées pour définir les nouveaux problèmes à résoudre. Cela nous mènera ensuite à définir les points particuliers qui seront traités dans cette thèse.

1.1 SoC et "IP Core"

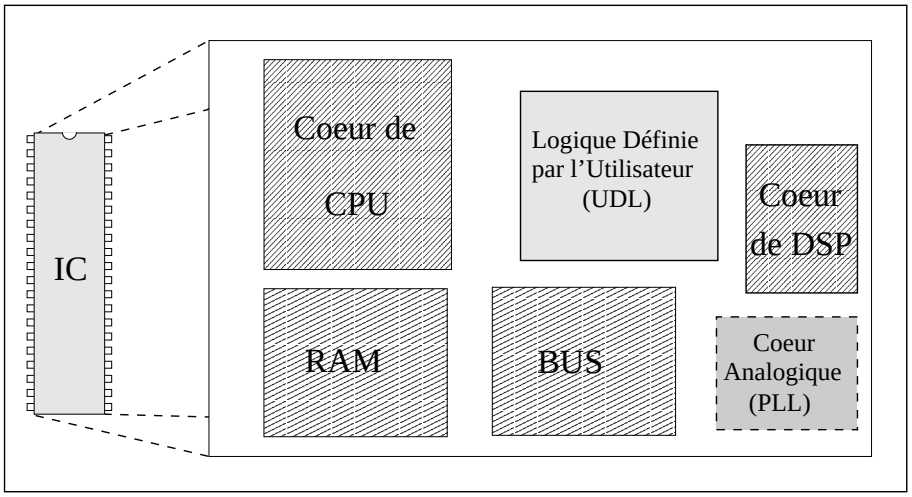


Figure 1.1: Exemple de système intégré à base de coeurs

Les capacités d'intégration permettent désormais de concevoir des systèmes intégrés constitués de blocs de nature et de complexité différentes (figure 1.1) ([HR96], [vsi]). Leur

conception requiert de multiples compétences en raison de la complexité des applications actuelles. L'équipe de conception doit intégrer sur un même circuit des coeurs de microprocesseurs, des DSP, des fonctions spécifiques particulièrement complexes, sans oublier les parties analogiques et/ou RF. Cela nécessite la mise en commun d'un ensemble de savoir faire et d'expertises que ne peut posséder une équipe seule. La conception de circuits basée sur un assemblage de parties ou de blocs préconçus est une solution à ce problème. Il s'agit donc d'utiliser ou de réutiliser des blocs qui ont déjà été conçus et vérifiés. Cela implique que le concepteur du SoC, appelé aussi intégrateur système, ait accès à une large bibliothèque de blocs afin d'implémenter rapidement son application. Ce concept de réutilisation n'est pas nouveau mais son application est désormais inévitable, les systèmes étant de plus en plus complexes.

Une seule équipe ne peut pas à la fois concevoir un SoC et les différents coeurs qu'il contient. Un partage des tâches est devenu nécessaire. On trouve aujourd'hui des sociétés spécialisées dans la conception de blocs réutilisables et d'autres dans l'intégration au niveau système.

On retrouve dans la littérature ces blocs sous le terme de macros, composants virtuels ou "IP cores" (Intellectual Property). Plusieurs classes caractérisent les IPs, mais on peut tout de même dégager trois grandes catégories.

1.1.1 Hard Cores

La première est la catégorie HARD, ou IP matériel. Un IP de ce type est optimisé en performances (puissance, taille...) pour une technologie spécifique. Il est délivré sous forme de netlist placée, routée, optimisée pour une bibliothèque technologique spécifique. L'inconvénient d'un IP HARD est qu'il est beaucoup moins portable que les deux autres catégories d'IP à cause de sa dépendance vis-à-vis d'une technologie cible. Par contre il a l'avantage d'être prédictif au niveau des performances finales et la propriété intellectuelle est fortement préservée. Un "hard core" est perçu par l'intégrateur système comme une boîte noire.

1.1.2 Soft Cores

La seconde famille est la famille des SOFT IPs (IPs logiciels). Le coeur est livré sous sa forme HDL (Hardware Design Language) synthétisable. Le principal avantage de cette classe d'IP est que grâce à cette description de haut niveau, la flexibilité et la portablité du coeur est assurée. Cependant l'intégrateur système, avec ce type de coeur, ne peut prévoir

ni sa taille ni ses performances (puissance, délais...). De plus la protection de la propriété intellectuelle n'est pas assurée puisque le fournisseur d'IP livre son composant virtuel avec le code RTL (Register Transfer Level) source.

1.1.3 Firm Cores

La troisième catégorie d'IP est une catégorie intermédiaire entre les catégories Hard et Soft, c'est la famille des FIRM "cores". Un IP de ce type est conçu de façon à optimiser sa surface et ses performances à travers un placement des modules qui le composent. Le coeur de type FIRM inclut une combinaison du RTL synthétisable, les références de la bibliothèque technologique cible, le détail du "floorplan" et une netlist de porte complète ou partielle. Le FIRM core n'est pas routé. Ce type de macrobloc est donc un compromis entre les IPs qui sont complètement logiciel et ceux complètement matériel. Ses performances et sa taille sont prévisibles mais la protection de la propriété intellectuelle peut ne pas être assurée.

1.1.4 Harmonisation et standards

Ayant d'un coté les fournisseurs d'IPs et de l'autre les intégrateurs systèmes, les échanges entre les uns et les autres supposent que l'ensemble des informations est échangé sans ambiguïté. Pour le développement des SoC, les méthodologies de conception doivent être réadaptées et un effort important de standardisation doit être effectué. Initiés par les grands industriels de la CAO, ces efforts de standardisation se sont concrétisés par la création du consortium VSI Alliance [vsi]. Son rôle est entre autres de définir des règles de conception et les formats de description des IPs.

En ce qui concerne la conception en vue du test, un groupe IEEE composé de nombreux acteurs majeurs de l'industrie du semiconducteur et d'universitaires s'est formé en 1997 et développe actuellement un ensemble de standards : le groupe IEEE P1500 [p15].

1.2 Spécificités du test des SoC

Etant donné la diversité des types de coeurs à intégrer dans un SoC et les nouveaux problèmes qui en découlent, il apparaît comme nécessaire de structurer la conception en vue du test des SoC. Les circuits devenant de plus en plus complexes, la part du test dans le "Time to Market" n'a cessé de croître. Nous sommes passés progressivement des systèmes

sur cartes aux MCM et des MCM aux systèmes sur une puce. Le flot de fabrication des

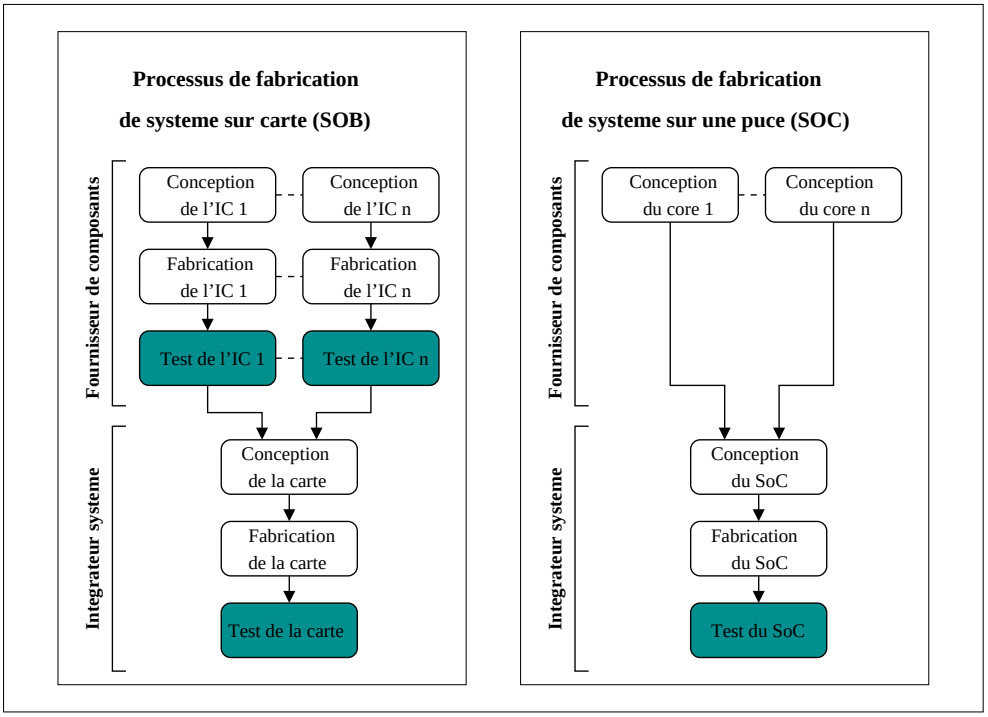


Figure 1.2: Flot de fabrication des systèmes (d'après [ZMD98])

systèmes a évolué et de ce fait la façon de les tester aussi (figure 1.2).

Les stratégies de test des systèmes sur cartes ou des MCM ne peuvent plus être appliquées au test des SoC [DMZ99]. Les cartes et les MCMs intègrent des composants physiques testés individuellement (donc considérés comme bons) et qui sont testés ensuite au niveau système. Pour les SoC, les composants étants virtuels, les différents coeurs ne pourront être testés que lorsque le circuit aura été fondu sur silicium, en même temps que le système. Cela implique que la testabilité du SoC doit être prise en compte très tôt, dès la phase de conception [ZBC97]. Cette prise en compte doit se faire à deux niveaux : au niveau des IP cores et au niveau système.

1.2.1 Au niveau IP

Du coté du fournisseur d'IP les techniques de test sont celles classiquement utilisées : insertion de points de test, de scan ou de BIST (Built In Self Test). Avec le coeur testable le concepteur doit aussi fournir les vecteurs de test, les réponses associées et les plans de test complets. Cependant il n'a pas forcément une bonne connaissance de l'application finale de son IP dans le SoC. Il ne sait pas quel est le niveau de test requis pour son IP. Doit-il

fournir une version scannable, bistée... ? Quel type de fautes doit il détecter ? Quel est le taux de couverture souhaité par l'intégrateur système ? Un taux de couverture élevé peut impliquer un coût important au niveau du test du SoC, alors qu'un taux plus faible met en danger la qualité du test du SoC.

1.2.2 Au niveau SoC

Du coté de l'intégrateur système de nombreux éléments doivent être pris en compte pour assurer la testabilité du SoC. L'une des difficultés majeures dans le processus de réalisation d'un SoC est l'intégration et la coordination des éléments de test dans le circuit. Le test d'un système sur puce est très complexe à définir, comparé au test de systèmes sur cartes qui parfois correspond principalement au test d'interconnexion entre les circuits.

Pour les SoC le test est en fait un test composé. Il comprend les tests individuels des différents IP cores, le test de toute la logique définie par l'utilisateur (UDL) et le test de l'ensemble du système.

Pour tester les coeurs individuellement puis collectivement se pose le problème du transfert des vecteurs de test. En effet pour définir les vecteurs de test à appliquer au niveau système et établir des plans de test, il faut traduire les vecteurs fournis avec les coeurs pour pouvoir les intégrer dans des sessions de test au niveau SoC. A l'heure actuelle le manque d'outils de CAO rend ces transferts d'un niveau à l'autre fastidieux. Seul Philips propose pour ses propres produits, une méthodologie et des outils pour palier ce problème avec son approche Macrotest ([ML97], [MA98]).

Cependant au delà des problèmes liés à la communication entre fournisseurs et intégrateurs d'IP, se posent les problèmes concernant les architectures permettant de tester les coeurs et leur environnement.

1.3 Vers une organisation des structures de test

1.3.1 Architecture de test des SoC

D'une façon générale l'architecture de test d'un système intégré sur une puce peut se décomposer en trois principaux éléments comme le montre la figure 1.3 : les "source"/"sink", le "wrapper" et le TAM.

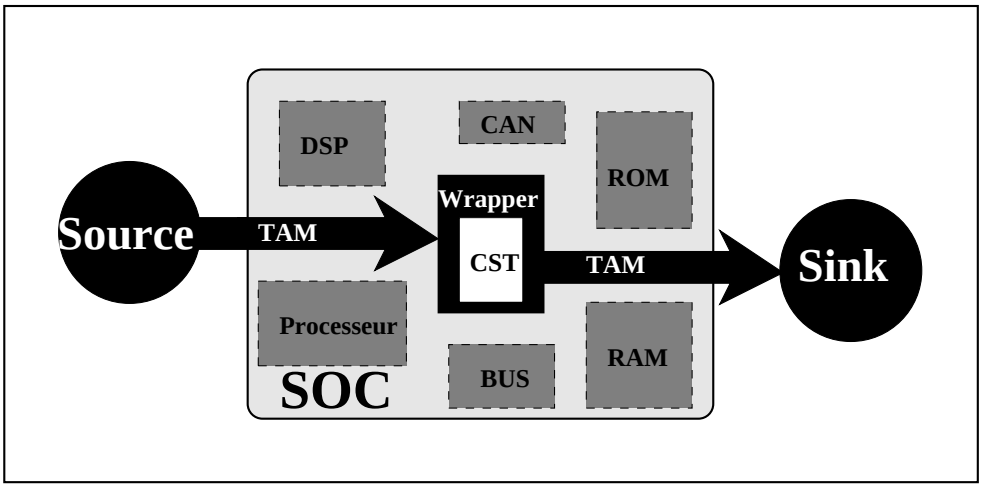


Figure 1.3: Architecture conceptuelle pour le test des SoC

Les "source" et "sink"

Le "source" est le terme désignant le matériel fournissant les vecteurs de test. Il peut s'agir d'un testeur industriel quand le "source" est externe au circuit ou d'un générateur de type BIST quand le "source" est interne. Parallèlement au "source" le "sink" correspond à l'analyseur de réponses aux stimuli appliqués. De même il peut être interne ou externe au SoC, sous la forme d'un analyseur de signature ou d'un testeur. On peut combiner des "source"/"sink" internes et externes.

Le wrapper

Pour que le coeur soit intégrable facilement dans le système embarqué, le fournisseur doit délivrer son composant virtuel avec une interface de test, interface appelé communément "wrapper" mais que l'on retrouve aussi sous les termes "socket" ([KW97]), "collar" ([CP96]) ou "test shell" ([M+98]). Cette interface doit permettre l'accès du test aux entrées/sorties du coeur. Elle permet d'isoler ([MNBB98]) électriquement l'IP quand le SoC est en mode test. Il est ainsi protégé des perturbations qui peuvent être provoquées par les IPs voisins. Le concepteur d'IP peut aussi grâce à ce wrapper implémenter les modes de contrôle du coeur. Les principaux modes de fonctionnement du bloc virtuel sont le mode normal, le mode test interne et le mode test externe. Ce dernier correspond au test des interconnexions entre IPs au niveau du SoC. Le wrapper permet donc les échanges de données de test entre le TAM et le circuit sous test.

Le TAM

Le TAM (Test Access Mechanism) est un terme qui englobe les moyens mis en oeuvre pour transporter les stimuli et réponses de test des entrées/sorties du SoC aux entrées/sorties de chaque élément du circuit (coeur et UDL). Le choix du TAM dépend des "source"/"sink", des types de coeurs intégrés et des contraintes physiques imposés à l'intégrateur système.

1.3.2 IEEE P1500 et notre approche

Le groupe IEEE P1500 chargé d'harmoniser le test des SoC définit actuellement deux standards.

Un langage décrivant les informations de test échangées entre fournisseurs et utilisateurs d'IP est en cours de finalisation. Le fichier CTL (Core Test Language) échangé contient la description de tous les éléments liés au test du coeur : vecteurs de test, description des E/S...

Le deuxième effort de standardisation du groupe est porté sur la définition du wrapper ([A+99]). Ce wrapper occupe une position cruciale puisqu'en plus d'être une interface de test, c'est aussi l'interface entre les fournisseurs et les utilisateurs d'IPs. L'architecture du wrapper P1500 est aussi en cours de finalisation.

Le groupe laisse à l'intégrateur système le choix des "source"/"sink" et du TAM à intégrer dans le SoC. Ces éléments n'étant pas standardisés, nous avons décidé de nous intéresser au TAM.

Les TAM peuvent se classer en deux catégories : les TAM correspondant à la réutilisation des ressources fonctionnelles pour transporter les vecteurs de test et les TAM définis comme matériels supplémentaires dédiés au test.

Les ressources fonctionnelles réutilisées pour le test offrent peu de flexibilité. Le plus souvent ces ressources correspondent au bus système. La largeur du bus est fixée, l'intégrateur système ne peut pas faire de compromis surface rajoutée/temps de test. Avec ce type de TAM la méthodologie de test est dominée par le test fonctionnel et l'ajout de structures scan et BIST est difficile à intégrer [ZMD98]. Actuellement le bus système utilisé le plus souvent comme TAM dans les SoC est le bus AMBA (Advanced Microcontroller Bus Architecture) de la société ARM, qui est devenu un standard de fait ([ARM]).

Les systèmes sur puce pouvant être très différents, la problématique est de définir un TAM qui soit flexible, paramétrable, adaptable au SoC et permettant un maximum de liberté au niveau du choix des techniques de test. Nous nous sommes donc focalisés sur la

définition d'un mécanisme d'accès au test des coeurs, dédié uniquement au test et indépendant des types de wrappers utilisés. Cette indépendance devrait permettre d'augmenter le champ d'utilisation du TAM et préserverait la compatibilité avec le futur standard IEEE P1500.

Cela nous a conduit à définir et développer l'architecture CAS-BUS, une architecture de test paramétrable, reconfigurable et flexible. Cette architecture a évolué vers deux autres versions, chacune répondant à des problèmes spécifiques.

Le standard P1500 est en cours de définition, son adoption et son application prendront certainement un peu de temps. Nous sommes donc dans une période de transition où les intégrateurs système peuvent intégrer des coeurs boundary scan dans leurs SoC. Ces coeurs vont se retrouver intégrés avec des coeurs équippés de wrappers P1500. Nous verrons dans le chapitre suivant que le contrôle de ces deux types de coeur est différent. Se pose alors le problème du test et du contrôle simultanés de ces deux types de coeurs. Nous proposons une extension de l'architecture CAS-BUS qui répond à ce premier problème.

Le second problème auquel nous nous sommes intéressé est celui lié à la bande passante du TAM. Lorsque le nombre de broches de test du SoC est limité et inférieur à la largeur du bus de test, un mécanisme d'expansion de données doit être mis en oeuvre. Ce mécanisme d'expansion peut apporter une première réponse mais a pour conséquence une augmentation du temps de test. Nous avons développé une méthode qui permet de coupler expansion et décompression de données. L'utilisation de vecteurs de test compressés permet de réduire le temps de test dû à l'expansion. Nous avons une nouvelle fois étendu l'architecture CAS-BUS pour intégrer ce nouveau mécanisme.

Chapitre 2

Etat de l'art et concepts de base

2.1 Introduction

La conception et le développement de la testabilité d'un SoC implique une gestion difficile des compromis coûts/qualité du test. Les principales contraintes influençant le choix de l'architecture de test sont :

- la surface supplémentaire due au test

- l'impact sur les performances du circuit

- la puissance supplémentaire dissipée

- le temps d'application des vecteurs de test

- le temps nécessaire au développement et à l'intégration du test dans le SoC

- la qualité du test

Optimiser une architecture pour répondre à tous ces critères est un problème NP complet ([Zor97]). Aussi, chaque SoC possédant ses propres contraintes de conception, le besoin d'avoir plusieurs architectures différentes se fait sentir.

Le TAM étant défini comme l'élément reliant les source/sink aux wrappers des IP cores, sa conception est régie par le choix d'un compromis bande passante/coûts induits. La bande passante du TAM est limitée par celle des source/sink et par la surface maximale autorisée pour son implémentation. Le temps d'application des vecteurs de test dépend de la largeur de cette bande passante et du volume de données qui doivent y transiter.

La définition d'un TAM pour un SoC dépend fortement des générateurs et analyseur de vecteurs de test et surtout de leur emplacement par rapport au SoC : externe et/ou interne.

Lorsque les source/sink sont externes au SoC (figure 2.1), la largeur du TAM est déterminée par le nombre d'entrées/sorties du circuit. La longueur du TAM peut être importante et cela peut entrainer des problèmes de délais. Avoir un source/sink externe pour envoyer et analyser les données équivaut à utiliser un testeur industriel. Cependant pour tester des circuits aussi complexes que les SoC il faut disposer d'un testeur très évolué qui permet l'accès à de nombreuses broches et qui intègre de nombreuses fonctionnalités. Ce type de testeur industriel est par conséquent très cher.

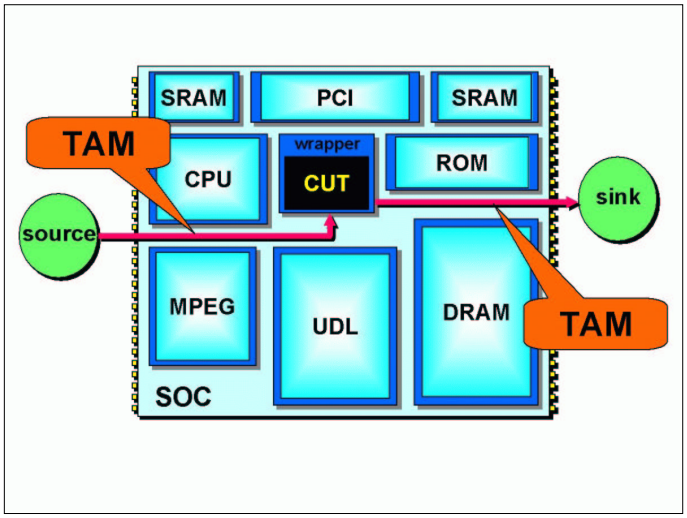


Figure 2.1: SoC avec des source et sink externes d'après [ZM99]

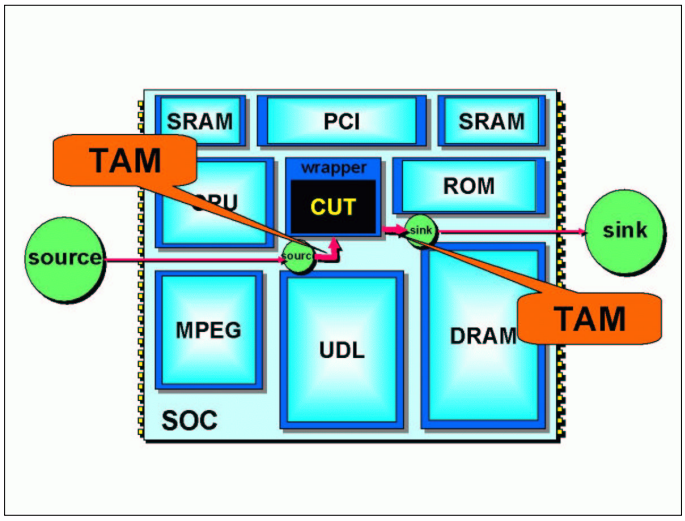


Figure 2.2: SoC avec des source et sink internes d'après [ZM99]

Un moyen de diminuer la complexité du testeur est de transférer une partie de ses fonctionnalités à l'intérieur du circuit (figure 2.2). Les source/sink internes peuvent être de

simples LFSR (Linear Feedback Shift Register) et MISR (Multiple Input Shift Register) ou des systèmes plus complexes. Avec ce type d'architecture la largeur du TAM est beaucoup moins dépendante du nombre de broches du SoC. Les source/sink pouvant être proche du CUT (core under test), la longueur du TAM peut être ajustée. Cependant le gain dû à l'utilisation d'un testeur plus simple se traduit par une augmentation de la surface allouée au test pour le SoC.

En fonction des source/sink et des contraintes imposées au concepteur du SoC, les TAM peuvent se classer en deux catégories : les TAM utilisant les ressources fonctionnelles du circuit et les TAM nécessitant la mise en oeuvre de matériel supplémentaire. Avant de détailler ces deux types de TAM, il est nécessaire de faire un rappel sur la norme IEEE 1149.1, de faire un état de l'art des normes développées, ou en cours de développement, par le groupe IEEE P1500 et le consortium VSIA. Ces normes sont et seront utilisées dans les architectures de test, et influeront sur la conception des mécanismes d'accès au test, ce qui est notamment notre cas avec l'architecture CAS-BUS que nous présentons dans le chapitre suivant.

2.2 La norme IEEE 1149.1 Boundary Scan

La technique Boundary Scan (BS) est apparue à la fin des années 80. C'est une extension des techniques de scan du niveau circuit vers le niveau carte. Face à la miniaturisation des cartes et des composants et à l'apparition des techniques de montage en surface SMT (Surface Mounted Technology), cette technique s'est imposée peu à peu comme standard. Elle permet l'accès sériel des vecteurs de test aux entrées/sorties de tous les circuits composants la carte. Outre le test de production, elle permet aussi d'effectuer les test de maintenance. Nous proposons dans ce qui suit une très brève description de ce standard, cette norme étant désormais bien connue dans le monde de la microélectronique. De plus amples informations sont disponibles dans [Boa90], [Mar99].

2.2.1 Architecture générale

Un composant à la norme Boundary Scan intègre plusieurs éléments dédiés au test, il est principalement constitué (figure 2.3) :

- d'un port d'accès au test, le TAP (Test Access Port), constitué de quatres signaux obligatoires (TDI, TDO, TMS et TCK) et d'un signal optionnel TRST.

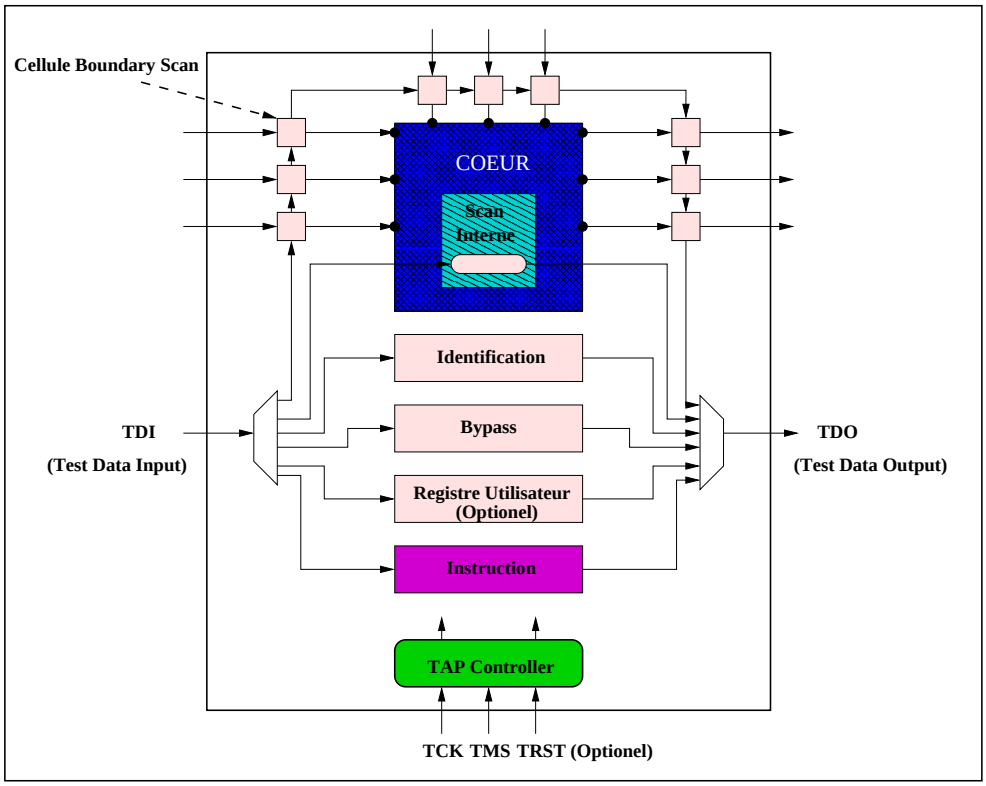


Figure 2.3: Architecture d'un circuit Boundary Scan

- de registres de données. Il s'agit des registres d'identification, de bypass, du registre Boundary Scan, du registre de scan interne du circuit et du (ou des) registres utilisateurs.

- d'un registre d'instructions.

- d'une machine d'état contrôlant l'architecture BS, le TAP Controller.

Les registres de données

Selon l'instruction chargée dans le registre d'instruction le registre sélectionné entre TDI et TDO peut être :

– Le registre de Bypass (obligatoire). C'est un registre sur un seul bit. Sur une carte composée de plusieurs circuits BS, les circuits qui ne sont pas impliqués dans un test donné sont mis dans le mode bypass (contournement) pour réduire la longueur du chemin de test global.

– Le registre boundary scan (obligatoire). Il correspond à la concaténation des cellules boundary scan (figure 2.4). Son rôle est de contrôler et d'observer les entrées/sorties du circuit. La cellule BS possède quatre modes de fonctionnement :

1 Le mode normal où les données sur l'entrée IN sont directement dirigées vers la sortie OUT. Cela correspond au mode de fonctionnement normal du circuit.
2 Le mode de décalage appelé mode SCAN qui permet le décalage de données dans le registre BS, d'une cellule BS à la suivante, le chaînage s'effectuant grâce aux entrées/sorties de scan Sin et Sout.
3 Le mode de capture, appelé aussi mode d'échantillonnage, qui permet de stocker la donnée présente sur IN dans la bascule QA.
4 Le mode de mise à jour (update) qui permet l'application des données présentes sur QA vers la sortie OUT.

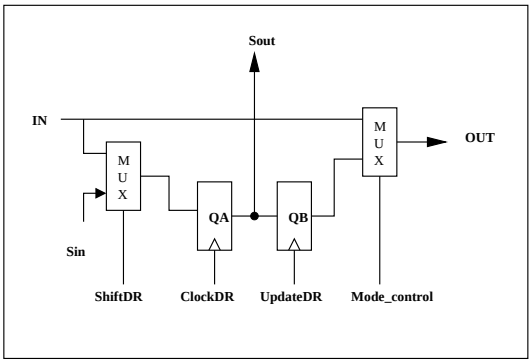


Figure 2.4: Exemple d'implémentation d'une cellule Boundary Scan

- Le registre d'identification (optionnel). C'est un registre de 32 bits correspondant au code d'identification constructeur du circuit.
- Le registre de scan (optionnel). Il correspond au registre de scan interne du circuit.
- Le ou les registres utilisateurs (optionnels). La norme prévoit la possibilité de rajouter des registres de données permettant l'ajout de fonctionnalités de test supplémentaires.

Le registre d'instruction

Ce registre permet de charger en série une instruction tout en maintenant l'ancienne active. C'est un registre à deux étages, le premier pour le chargement en série de l'instruction, le second permettant sa mise à jour. Il est connecté à de la logique de décodage qui va entre autres sélectionner le registre de données correspondant.

Le TAP Controller

C'est une machine à états finis comportant seize états (figure 2.5), pilotée par le signal TMS (Test Mode Select) et l'horloge TCK. La transition d'un état à l'autre dépend de la valeur de TMS. Cet automate se compose principalement de deux branches, l'une pilotant l'activité des registres de données, l'autre celle du registre d'instruction.

L'état Test Logic Reset correspond au mode normal de fonctionnement du circuit. Le lancement d'un test intégré de type BIST peut être effectué grâce à l'état RUN Test/Idle. Les états de chacune des deux branches de l'automate permettent, entre autres, de charger par décalages successifs les registres concernés et de mettre à jour ces données.

Cet automate génère les signaux de contrôle suivants : Reset, Select, Enable, Shift_IR, Clock_IR, Update_IR, Shift_DR, Clock_DR et Update_DR.

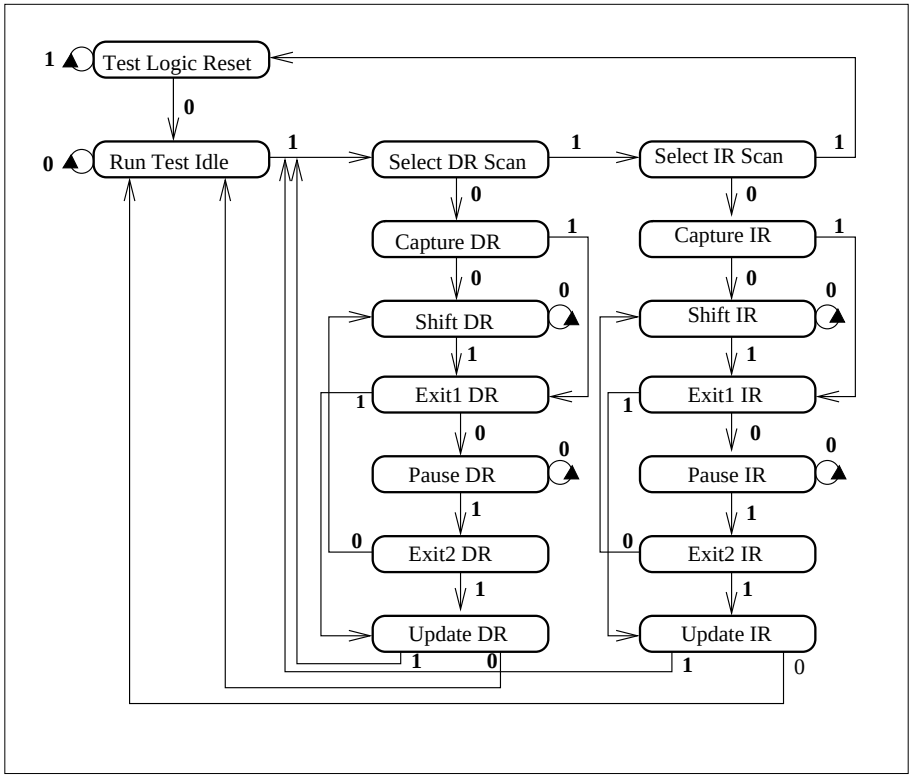


Figure 2.5: La machine d'état TAP

2.2.2 Le test Boundary Scan

La procédure de test d'un circuit BS est basée sur la succession d'actions du type suivant :

- Initialisation de la logique de test

- Chargement d'une instruction

- Application de vecteurs de test

- Lecture de réponses aux stimulis

Le chargement d'une instruction, la lecture et l'application des vecteurs de test se fait par décalages sur TDI et TDO.

Trois modes de test au niveau carte sont disponibles : le test externe, le test interne et le test d'échantillonnage.

Le test externe permet grâce à l'instruction EXTEST de tester les interconnexions entre circuits.

Le test interne (instruction INTEST) consiste comme son nom l'indique à tester le circuit lui même. La contrôlabilité et l'observabilité des entrées/sorties fournies par les éléments du standard rendent ce test possible.

Le test d'échantillonage (instruction SAMPLE/PRELOAD) correspond à la capture à la volée des données présentes aux entrées et aux sorties du circuit.

Il existe d'autres test liés à l'utilisation de la norme 1149.1 mais nous ne les traiterons pas dans ce document.

Le succès de ce standard est incontestable, il correspondait à un réel besoin de l'industrie et son application en est la preuve. Ce succès est tel que, face aux nouveaux problèmes soulevés par le test des SoC, des architectes ont utilisé et adapté ces techniques boundary scan pour améliorer la testabilité de leurs systèmes sur puce. On assiste ainsi à une migration des techniques initialement prévues pour les systèmes sur carte vers une utilisation sur des circuits à base de coeurs. Certaines adaptations de cette norme seront présentées dans la section quatre de ce même chapitre.

2.3 Les standards en développement par le groupe IEEE P1500

2.3.1 Histoire et objectifs

Comme le JTAG (Joint Test Action Group) dans les années 80, le groupe IEEE P1500 s'est formé pour essayer de proposer des solutions aux problèmes de test rencontrés dans l'industrie. En septembre 1995, un comité de recherche sur les techniques de test des systèmes à base de coeurs a vu le jour. Ce comité, issu du IEEE TTTC (Test Technology Tech-

nical Council), s'est transformé en groupe de travail en juin 1997 sous le nom de P1500. Ce groupe avait pour but de créer des standards pour faciliter l'utilisation et la réutilisation des techniques et méthodes de test des systèmes à base de coeurs. Les trois projets en cours de standardisation sont :

- le test des "mergeable cores" (coeurs "fusionnables").

- le développement d'un langage de description des informations de test d'un core.

- la définition d'une interface de test autour des "non-mergeable cores".

Les mergeable cores

Les "mergeable cores" correspondent dans l'esprit du groupe P1500 aux soft cores c'est à dire aux IPs disponibles sous une forme RTL synthétisable. Ils peuvent être fusionnés avec d'autres IPs ou avec de la logique définie par l'intégrateur système. Ces coeurs n'incluent pas d'éléments de DFT et ne disposent pas d'interface de test spécifiée. L'idée du standard est de faciliter l'interopérabilité du point de vue du test de ces coeurs avec les autres IPs lors de l'intégration au niveau du système. Ce standard voudrait définir un processus d'insertion de DFT dans ce type de coeur avec au final un document standard précisant les spécificités du test lié au coeur. Cette standardisation débute et très peu d'informations sont disponibles à son sujet.

Le CTL

Le deuxième projet de standardisation concerne le CTDL (Core Test Description Language), appelé aussi CTL. C'est un langage dont l'objectif serait de décrire toutes les informations liées au test du coeur et ainsi de faciliter les échanges entre fournisseurs et intégrateur d'IPs. Parmi ces informations on peut trouver :

- Les méthodes de test utilisées.
- Les modes de test et les protocoles correspondant.
- Les données de test à appliquer qui peuvent se trouver sous la forme de listes de vecteurs, d'algorithmes spécifiques pour le test des mémoires, les polynômes des éléments pour le BIST...
- Les informations sur les modèles de fautes et les taux de couverture.
- Le nombre, l'ordre et la longueur des chaines de scan internes, des LFSR ...
- Les informations sur le moyen de procéder au debug et au diagnostic de l'IP.
- ...

Le langage du standard IEEE 1450 STIL (Standard Test Interface Language) [Soc99] va être étendu pour inclure le CTDL. En effet STIL est à l'heure actuelle un langage qui décrit seulement les vecteurs de test et les formes d'ondes à appliquer à un circuit.

Le Wrapper P1500

Le troisième projet du groupe P1500, bien plus avancé que les deux précédents, consiste à définir une interface de test standard, appelée wrapper, dont le but est à la fois de permettre l'accès des données de test aux entrées/sorties du coeur et de positionner celui-ci dans le mode de fonctionnement choisi. Ce wrapper s'adresse aux coeurs dits "non mergeable", ce qui correpond aux IPs hard tels qu'on les a defini précédemment ou du moins aux IPs qui ne peuvent pas être fusionnés à d'autres éléments du système. Ce projet nous intéresse plus particulièrement car le wrapper fait le lien entre le TAM et le coeur lui même. La description du wrapper qui suit est issue des informations disponibles sur le site web du P1500 [p15]. Le standard est en cours de développement et certaines zones d'ombre subsistent, certains points restent encore à définir.

2.3.2 Architecture du wrapper P1500

Le wrapper P1500 permet d'acheminer les données de test aux entrées/sorties du coeur, soit en série par l'interface série SIL (Serial Interface Layer) (figure 2.6 (a)) soit en parallèle par l'interface PIL (Parallel Interface Layer) (figure 2.6 (b)). L'interface SIL est actuellement assez bien définie, ce qui n'est pas encore le cas de l'interface PIL. Nous décrivons dans ce qui suit ce qui a trait à l'interface SIL.

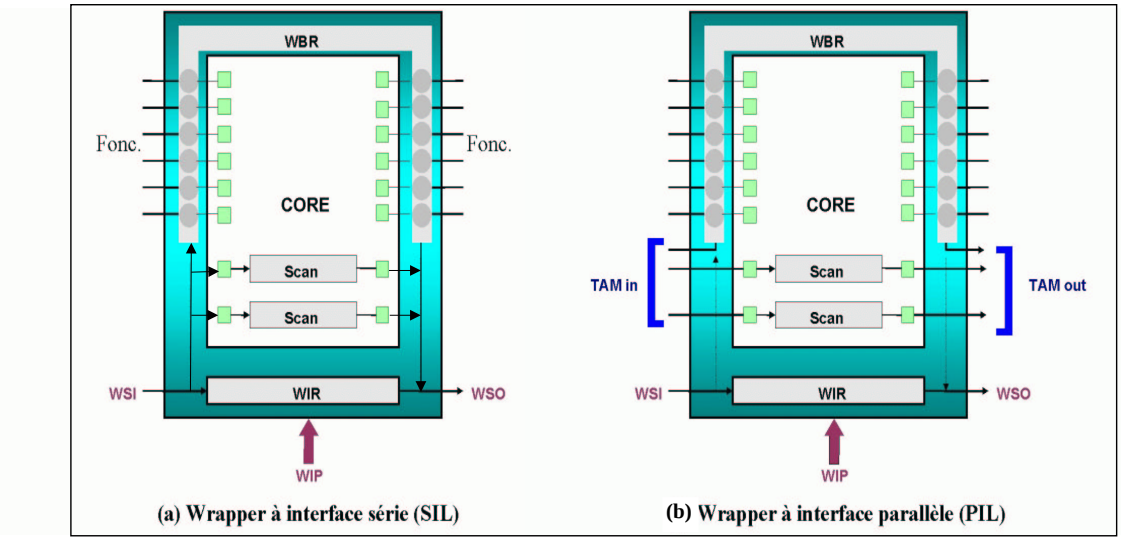


Figure 2.6: Architecture du wrapper P1500 (d'après [p15])

Le wrapper est constitué principalement :

- d'un registre d'instruction WIR (Wrapper Instruction Register).

- d'un registre de données WBR (Wrapper Boundary Register).

- d'un registre de bypass WBY (Wrapper Bypass Register).

- d'un port d'accès constitué de WSI (Wrapper Scan IN), WSO (Wrapper Scan Out) et du WIP (Wrapper Interface Port) qui inclut plusieurs signaux de contrôle.

En plus des registres de données WBR et WBY, la norme permet l'ajout de registres spécifiques à l'utilisateur. Les registres internes du coeur se retrouvent sous le nom de CDR (Core Data Register). Les registres de données sont regroupés sous l'appellation WDR (Wrapper Data Register). Les fonctionnalités des registres du wrapper sont semblables à celles de leurs homologues du standard IEEE 1149.1 (figure 2.7). On retrouve les fonctions de capture, de décalage et de mise à jour, la fonction de capture pour le WIR étant optionnelle. Ces registres doivent permettre de garder les signaux et les modes de fonctionnement stables pendant les opérations de décalage.

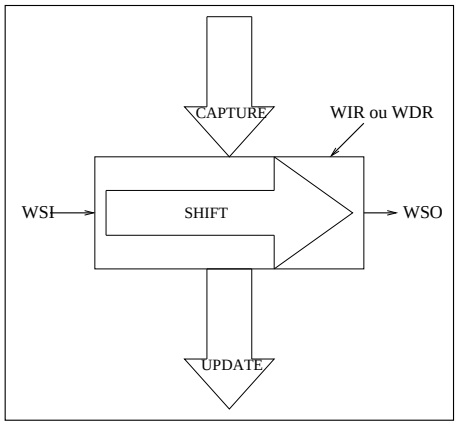


Figure 2.7: Comportement des registres (d'après [p15])

Le WIP

Les signaux WSI et WSO permettent d'accéder en série aux différents registres. Les différents signaux de contrôle sont fournis soit par le registre d'instruction et sa logique de décodage (signaux statiques), soit par le WIP (signaux dynamiques). Le WIP est constitué des signaux suivants :

– WRCK : signal d'horloge utilisé pour commander les registres.
– WRSTN : signal de reset asynchrone du wrapper.
– UpdateWR : signal de mise à jour des données du registre sélectionné.
– ShiftWR : signal de décalage des données contenues dans le registre.
– CaptureWR : signal de capture de données dans le registre.
– SelectWIR : signal permettant la sélection du registre d'instruction comme registre

actif entre WSI et WSO.

– TransferDR : signal qui permettrait d'activer la fonction "transfert de données" dans le registre WBR.

Le WBR

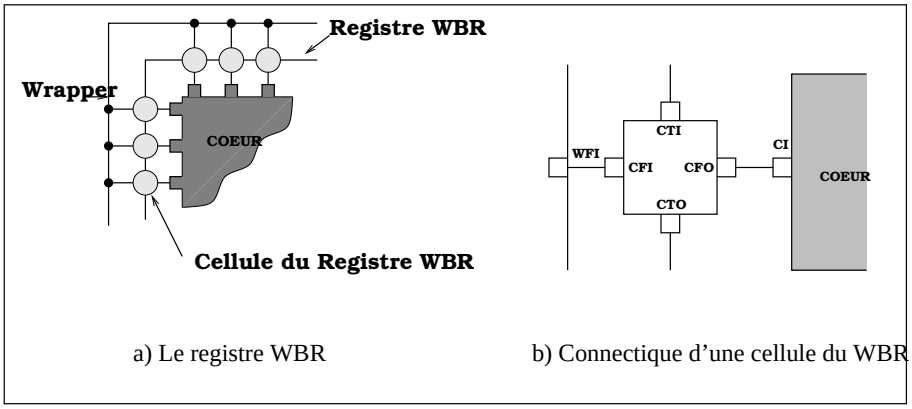


Figure 2.8: Le registre de données WBR et sa connectique (d'après [p15])

L'ensemble des cellules du wrapper présentes aux entrées/sortie de l'IP constituent le registre WBR (figure 2.8). Une cellule du WBR s'intercale entre l'entrée du wrapper WFI (Wrapper Functional Input) et l'entrée du coeur CI (Core Input). Le chemin CFI/CFO (Cell Functional Input/Cell Functional Output) constitue le chemin que prennent les données en mode fonctionnel. Le chemin CTI/CTO (Cell Test Input/Cell Test Output) correspond quant à lui au chemin de test en mode décalage.

Chacune de ces cellules doit se trouver dans un des modes suivant : normal, inward facing, outward facing ou safe (figure 2.9).

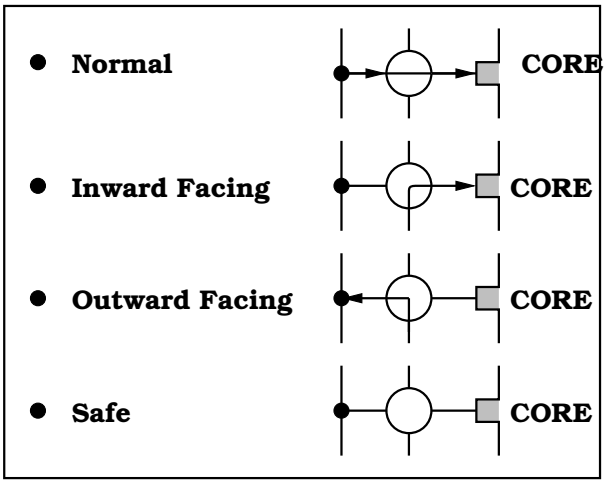


Figure 2.9: Mode de fonctionnement des cellules (d'après [p15])

Le mode normal correspond au fonctionnement normal du circuit, les cellules sont alors transparentes et sans effet.

Le mode inward facing correspond au mode de test, les stimuli sont appliqués sur les connecteurs de l'IP. Ce mode affecte le core. Le WBR permet le contrôle des entrées du core et l'observation des sorties.

Le mode outward facing est le symétrique du mode inward facing. Ce mode affecte le système. Le WBR permet le contrôle des valeurs de sortie du wrapper et l'observation de ses entrées.

Le mode safe fait en sorte que les données du wrapper n'altèrent pas le système ou le core. C'est un mode d'isolation qui permet de forcer les connecteurs de l'IP à des valeurs prédéterminées. C'est un mode qui n'est pas obligatoire mais tout de même recommandé.

Différents types de cellules sont prévus par la norme. Ces cellules sont représentées par leurs schémas conceptuels (figure 2.10). Leur implémentation n'est pas encore définie.

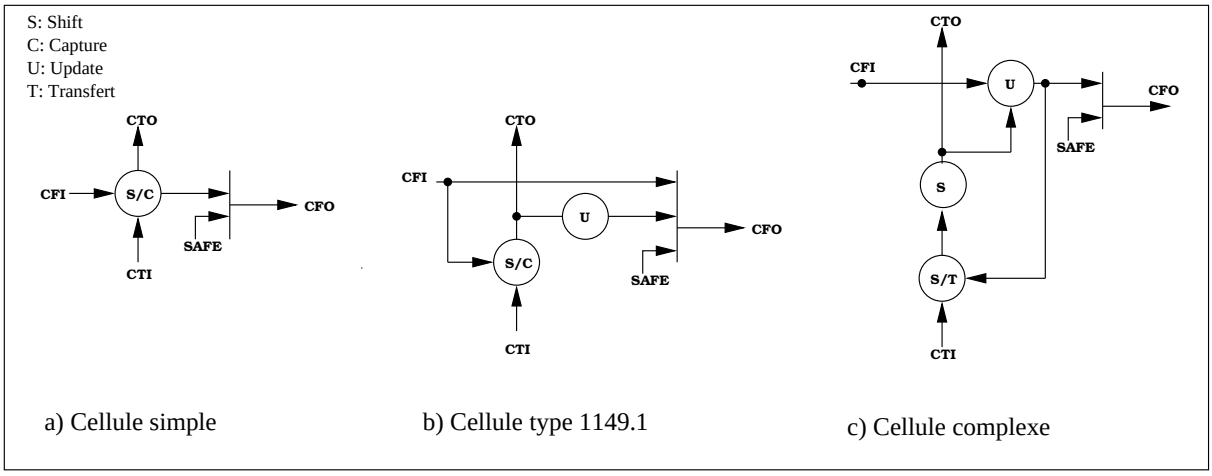


Figure 2.10: Schémas conceptuels des différents types de cellules (d'après [p15])

Elles doivent permettre les événements :

- de capture de données (capture)

- de mise à jour de ces données (update)

- d'application des données aux entrées/sorties du coeur (apply)

- de transfert. Cette fonctionnalité, pour laquelle on dispose de peu d'informations, devrait permettre de transférer les données présentes à la sortie de l'étage update vers le chemin de décalage (figure 2.10 (c)).

La présence des cellules du WBR aux entrées/sorties fonctionnelles de l'IP sont obligatoires. Pour les entrées/sorties dédiées au test cette présence est optionnelle.

Le WIR

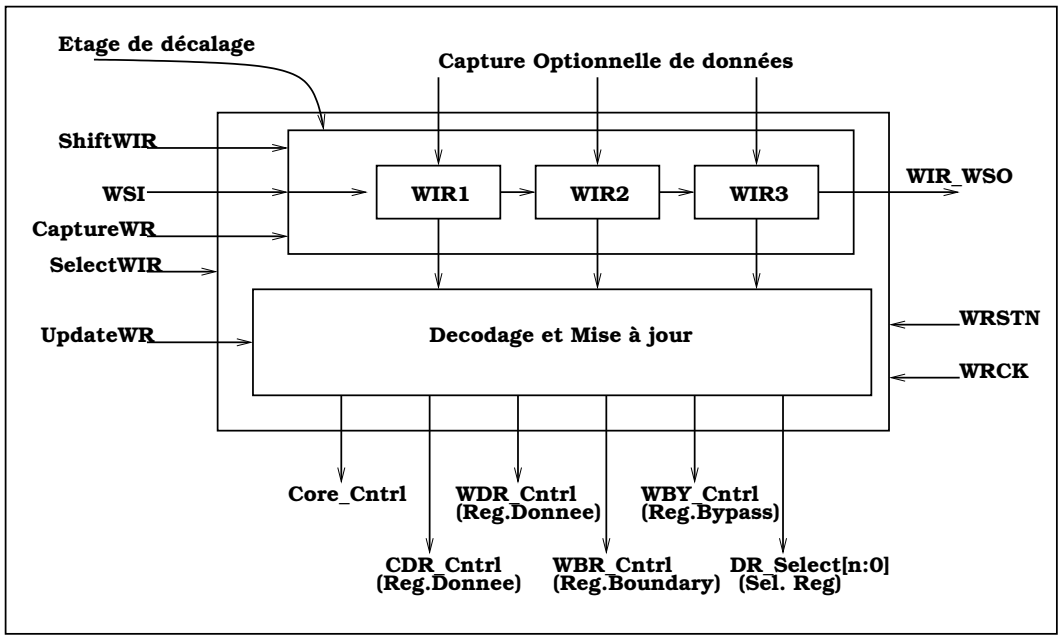


Figure 2.11: Architecture du registre d'instruction

C'est un registre à deux étages (figure 2.11), un étage de décalage et un étage de mise à jour. Il doit permettre le chargement en série d'une instruction tout en maintenant l'instruction précédente active. De manière optionnelle la norme permet le chargement en parallèle d'une nouvelle instruction. C'est un registre dont la longueur minimale est de trois bits. Ce registre est connecté à un étage de décodage qui génère les signaux de contrôle du WBR et du registre de Bypass ainsi que les commandes des différents multiplexeurs. En fonction de l'instruction active, les registres de données concernés sont positionnés entre WSI et WSO. Le WIR est synchrone sur l'horloge WRCK. Le signal de reset asynchrone WRSTN, actif sur niveau bas, lorsqu'il est appliqué, configure le wrapper en mode bypass. Le WIR charge alors l'instruction correspondante (WBYPASS).

2.3.3 Les instructions permises par le standard P1500

Comme son homologue du boundary scan, le WIR permet le chargement d'un ensemble d'instructions, certaines obligatoires, d'autres optionnelles. Elles sont définies à l'heure actuelle comme suit :

- WBYPASS : obligatoire
  Cette instruction positionne le registre de bypass (un ou deux bits) entre WSI et WSO. Elle force le core dans son mode de fonctionnement normal.
- WPRELOAD : obligatoire
  L'instruction WPRELOAD permet de charger en série le registre WBR puis met à jour l'étage update. Cette instruction est souvent utilisée avant de faire un test interne ou un test externe.
- WCLAMP : obligatoire
  Elle force les états de sortie des cellules du registre WBR et sélectionne le registre de bypass comme registre actif entre WSI et WSO. Cette instruction doit être précédée de l'instruction WPRELOAD. Le core est mis dans un mode "safe".
- WEXTEST : obligatoire
  Equivalente à celle du boundary scan, cette instruction permet d'effectuer le test d'interconnexion du core encapsulé avec son environnement. Le WBR est positionné entre WSI et WSO, tandis que le core est en mode "safe". Les valeurs de sortie du wrapper sont déterminées par les valeurs du registre WBR.
- WSAFESTATE : optionnelle
  Cette instruction agit de la même façon que l'instruction WCLAMP mais ne nécessite pas l'utilisation de l'instruction WPRELOAD.
- WSWCORETEST :
  Elle permettrait le test interne du coeur en utilisant le registre WBR comme registre actif entre WSI et WSO.
- WSCORETEST :
  Cette instruction permettrait d'effectuer le test interne du coeur, mais contrairement à l'instruction WSWCORETEST, le registre actif entre WSI et WSO correspondrait à la concaténation du registre WBR et du (ou des) chaines de scan interne du coeur.
- WCORETEST :
  Cette instruction concerne le test interne du coeur mais sa fonctionnalité n'est pas encore bien définie.

La norme prévoit qu'au moins une instruction CORETEST (test interne) soit obligatoire. Des instructions permettant l'utilisation de plusieurs chaînes de scan en parallèle commencent à être définies. Ces instructions s'adresseront vraisemblablement aux wrappers possédant l'interface parallèle (PIL). Nous avons pour l'instant les instructions WEXTESTP et WPRELOADP qui permettraient d'effectuer les mêmes opérations que leurs équivalents séries, mais l'accès se ferait sur plusieurs chaînes en parallèle, probablement avec plu-

sieurs WSI/WSO et plusieurs WBR. La définition des codes binaires correspondant aux différentes instructions est laissée à l'appréciation de celui qui conçoit le wrapper. Nous ne disposons pas pour l'instant d'informations sur des instructions qui permettraient par exemple de lancer le BIST d'un core, du type RUNBIST. Cela devrait être à notre avis une des instructions CORETEST à définir.

2.3.4 Le wrapper développé par VSI Alliance

Le consortium VSI Alliance a développé un wrapper dont l'utilisation est recommandée en attendant que le standard IEEE P1500 soit adopté. Le wrapper proposé par VSI Alliance possède un degré de flexibilité plus important que celui du P1500. Par exemple, la présence du registre d'instruction dans ce wrapper est optionnelle. En cas d'absence de ce registre, certains signaux de contrôle doivent être alors ajoutés au niveau de l'interface du wrapper et générés au niveau SoC. Le wrapper P1500, à travers WSI et WSO, permet un seul chaînage du registre WBR. Pour VSI Alliance, le WBR peut posséder une ou plusieurs entrées/sorties qui lui sont propres. Ceci permet ainsi de diviser ce registre en plusieurs blocs de longueurs différentes.

La spécification du wrapper développé par VSIA est beaucoup plus complète que celle du wrapper P1500 mais pour des raisons de confidentialité nous ne pouvons présenter dans le détail ce wrapper.

2.3.5 Conclusion

L'architecture des wrappers P1500 et VSIA et leurs fonctionnements sont proches de ce qui a été développé pour le standard Boundary Scan. La principale différence réside dans l'absence dans ces wrappers d'automate gérant les signaux de contrôle.

Le groupe P1500 considère que le contrôle des éléments constituant le wrapper s'effectue à l'extérieur de celui-ci et peut être réalisé librement par l'intégrateur système. Un générateur de wrapper P1500, implémentant certaines des fonctionnalités décrites précédemment a été développé au laboratoire. Une description en sera faite dans le chapitre cinq de ce document.

En ce qui concerne la définition de la compatibilité P1500 d'un core par rapport à un autre, le groupe définit deux types de labels :

- IEEE 1500 Unwrapped

Un coeur est dit IEEE 1500 Unwrapped si le coeur délivré ne dispose pas de wrapper P1500 mais dispose du fichier CTL permettant de le concevoir, automatiquement ou manuellement.

- IEEE 1500 Wrapped

Le coeur envoyé à l'intégrateur système est équipé du wrapper P1500 et accompagné de son fichier CTL.

Avec ce dernier type de label, l'implication sous-jacente est que le fournisseur d'IP devra soit définir un ensemble de wrappers pour un core donné, puis l'intégrateur système choisira celui qui lui convient, soit il devra le concevoir à la commande, en fonction des paramètres exigés (type d'interface, largeur du ou des TAM, instructions optionnelles choisies...). Une bonne commmunication entre fournisseur et utilisateur d'IP sera alors nécessaire.

2.4 Les différents types de TAM existants

En fonction des différentes contraintes qui lui sont imposées pour la conception de son SoC, l'intégrateur système doit choisir le type de TAM qu'il peut utiliser. Ce choix crucial dépend non seulement des contraintes citées au début de ce chapitre, mais aussi des orientations et du savoir faire de l'entreprise. Principalement il aura le choix entre utiliser les ressources fonctionnelles du système pour transporter ses données de test ou alors ajouter du matériel non fonctionnel, dédié au test. Bien évidemment toutes les solutions disponibles possèdent leurs avantages et leurs inconvénients. La seconde solution qui consiste à ajouter des éléments de test offre par nature la possibilité de développer de nombreuses techniques. Nous allons décrire dans cette section quelques TAMs précisant les principales approches actuellement utilisées.

Nous décrirons d'abord les techniques réutilisant les ressources fonctionnelles du circuit. Nous présenterons ensuite les architectures basées sur la réutilisation des techniques du Boundary Scan et du BIST. Nous présenterons pour finir les spécificités des architectures s'appuyant sur les bus de test.

2.4.1 Les ressources système utilisées comme TAM

La première approche est celle consistant à utiliser le matériel présent dans le système pour acheminer les données de test des entrées/sorties du SoC vers le ou les cores cibles.

On trouve principalement deux techniques permettant de valider cette approche : une technique basée sur l'utilisation de la transparence des coeurs, l'autre sur la réutilisation du bus système.

Transparence des coeurs

Le principe de cette technique est de propager des données à travers différents IP cores du circuit vers le coeur à tester en utilisant les propriétés de transparence des IPs traversés. La propagation de ces données sans perte d'information, peut être réalisée grâce à la présence de chaînes de scan internes ou en utilisant les fonctions arithmétiques de l'IP (données + 0, données x 1,...). De même les propriétés des mémoires ou des portes logiques (and, inv, or, ...) peuvent aussi être utilisées.

Une méthode de test nommée SOCET a été développée par I. Ghosh, N. K. Jha et S. Dey, utilisant cette technique ([GJD97], [GDJ98]). Cette méthode agit à deux niveaux.

Au niveau core elle analyse et utilise les fonctionnalités de l'IP pour définir un mode de transparence. Plusieurs versions de l'IP équipé du mode transparent sont générées, en fonction de paramètres comme la surface additionnelle et le temps de latence. Le temps de latence correspond au temps nécessaire à la traversée du coeur et a un impact direct sur le temps de test global.

Au niveau système, cette méthode analyse la topologie du SoC, puis sélectionne la version du coeur permettant d'atteindre les objectifs fixés en termes de surface additionnelle et de temps de test.

AVANTAGES DE LA TECHNIQUE DE TRANSPARENCE

La surface additionnelle induite est faible.

INCONVÉNIENTS

- L'accès au core à tester dépend des autres cores et de la logique du système.

- Lors de la conception de l'IP, l'environnement dans lequel il va être placé est inconnu. Cela implique pour le fournisseur d'IP de décliner son IP suivant plusieurs niveaux de transparence. Un niveau élevé se traduira par une surface additionnelle importante, tandis qu'un niveau faible entraînera pour l'utilisateur du coeur la nécessité de rajouter un TAM supplémentaire.

- La transformation des vecteurs de test livrés avec le coeur en vecteurs de test au niveau système peut devenir complexe et difficile à automatiser. Le traitement des vecteurs dépend des différents blocs du systèmes et de leurs latences.

Bus système utilisé comme TAM

La plupart des systèmes sur puce intègrent un ou plusieurs bus fonctionnels. Une démarche naturelle consiste à utiliser ce bus pour acheminer les données de test vers les IPs qui y sont connectés. Le bus AMBA (Advanced Microcontroller Bus Architecture) de la société ARM est devenu grâce à son succès dans l'industrie, un standard de fait. Dans le cadre de cette approche, à notre connaissance seul le bus AMBA apporte une stratégie de test basée sur la réutilisation du bus système. Pour bien comprendre l'intérêt de cette approche, une présentation du fonctionnement global du test avec AMBA est nécessaire. Une description plus détaillée d'AMBA peut être trouvée dans [FW98] et [ARM].

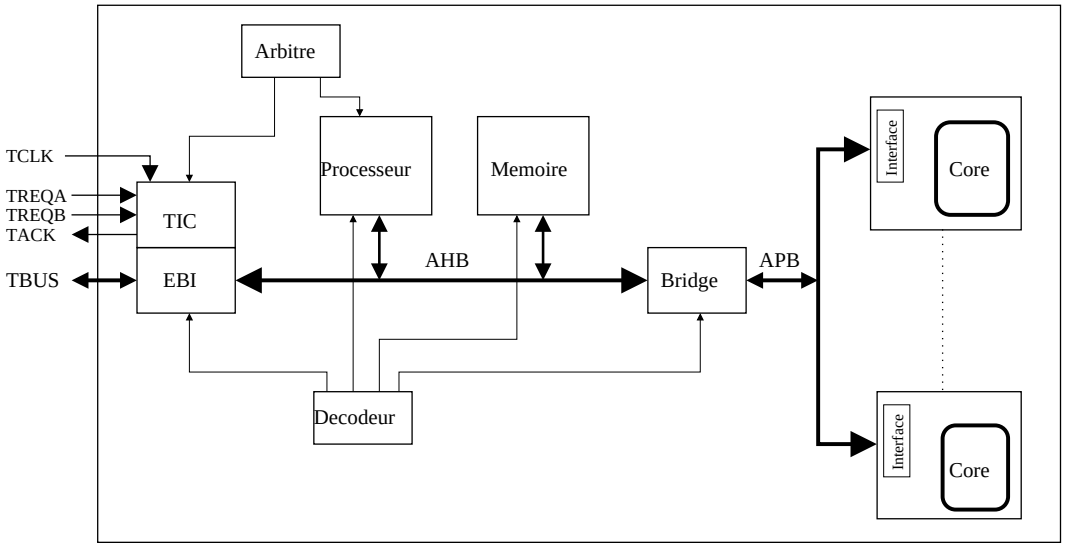


Figure 2.12: SoC intégrant le bus AMBA (d'après [ARM])

La figure 2.12 représente un système bâti autour du bus AMBA, qui peut être de largeur 32, 64 ou 128 bits. Ce bus est composé de deux parties : le bus système AHB (Advanced High-performance Bus) et le bus périphérique APB (Advanced Peripheral Bus). L'AHB est un bus "rapide", permettant d'avoir plusieurs maîtres, d'envoyer des données en rafale... L'APB est par contre un bus "lent" sur lequel sont connectés des modules périphériques comme les "timer", les contrôleurs d'interruptions...L'APB et ses modules sont orientés basse consommation, contrairement à l'AHB et ses IP dédié aux hautes performances. L'AHB et l'APB sont reliés par un "bridge".

Classiquement, sur l'AHB sont branchés un processeur de type ARM, une mémoire, un contrôleur de test (TIC) et une interface de bus externe (EBI). La sélection du module maître sur le bus s'effectue grâce au module d'arbitrage tandis que le choix de l'IP concerné

par les transferts sur le bus est réalisé grâce au décodeur d'adresse.

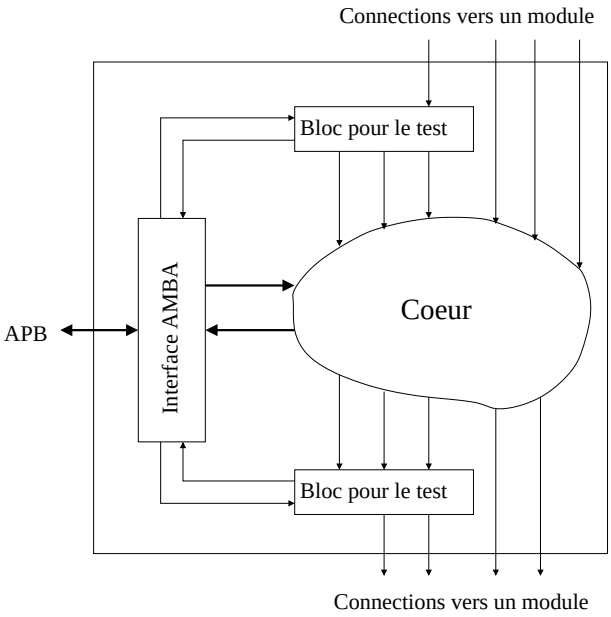


Figure 2.13: Accès au test d'un coeur branché sur le bus APB

La stratégie d'AMBA consiste à tester individuellement et indépendamment les modules esclaves branché sur l'AHB et les modules connectés à l'APB. L'accès à un IP sur le bus lent se fait à travers une interface AMBA et des blocs de test (figure 2.13). Cette interface est constituée de registres et de logique de décodage d'adresses. Les blocs de test ajoutés permettent d'apporter une contrôlabilité et une observabilité aux entrées/sorties du core qui ne sont pas connectées au bus. Ces entrées/sorties peuvent être reliées à d'autres cores, à de la logique définie par l'intégrateur système (UDL) ou directement aux broches du SoC. Ces blocs sont mis en place pour améliorer la couverture de faute de l'IP et donc du système.

Le TIC (Test Interface Controller) gère les transferts de données de test sur le bus. Pendant le mode de test il prend la main et devient maître sur le bus. Il est contrôlé par quatre signaux : TCLK, TREQA, TREQB et TACK. Les vecteurs de test sont transférés en parallèle grâce au bus de test TBUS (32 bits), de l'extérieur du circuit vers le bus AMBA, à travers l'EBI (External Bus Interface). L'horloge de test est TCLK mais peut aussi être une horloge interne du SoC. En fonction des valeurs de TREQA (Test Request A), TREQB (Test Request B) et de TACK, différents types de données peuvent être appliqués au système :

1. Des données d'adresse. L'envoi d'une adresse revient à choisir l'IP qui va être testé.

2. Des données d'écriture. Ce sont les vecteurs à appliquer au coeur cible.

3. Des données de lecture. Ce sont les vecteurs réponses qui vont être acheminés vers l'extérieur du SoC.

TREQA (entrée)	TREQB (entrée)	TACK (sortie)	Description
0	0	0	mode normal
1	0	0	Requête pour entrer en mode test
0	1	0	Réservé
X	X	1	Entrée dans le mode test
1	1	1	Envoi d'un vecteur d'adresse
1	0	1	Envoi d'un vecteur à écrire
0	1	1	Envoi d'un vecteur à lire
0	0	1	Requête pour sortir du mode test

TAB. 2.1 – Signaux de contrôle du TIC

Les valeurs des signaux contrôlant le TIC sont regroupées dans le tableau 2.1.

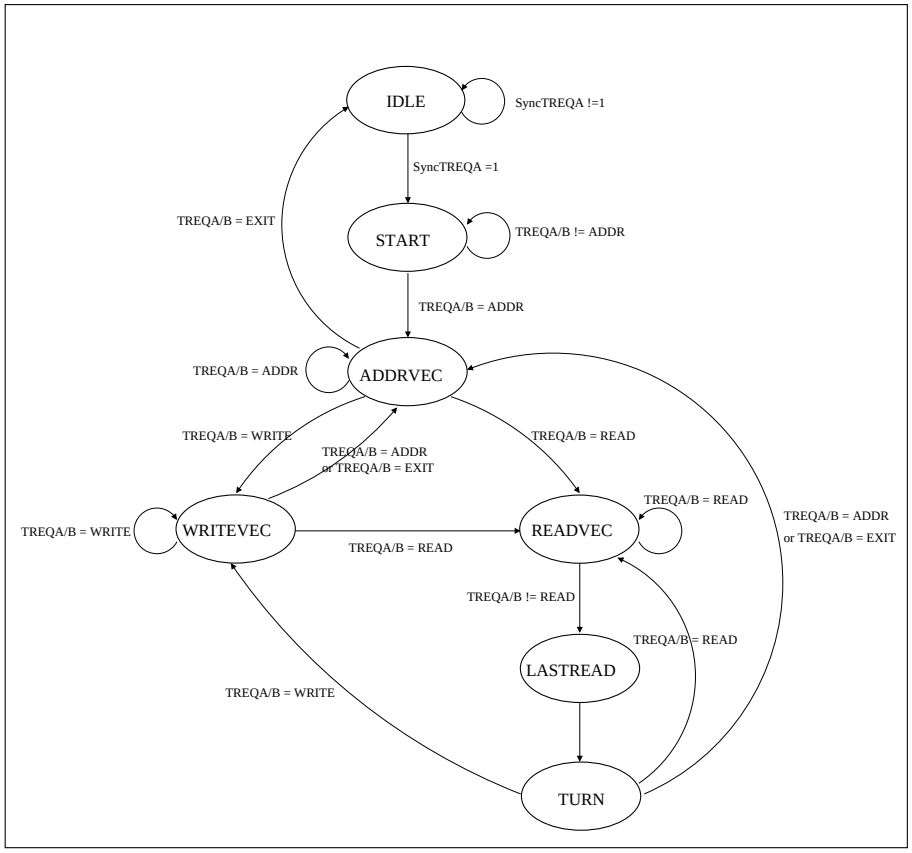


Figure 2.14: Diagramme d'états du TIC (d'après [ARM])

Sur la figure 2.14 correspondant au diagramme d'états du contrôleur de test, on voit que pour passer de l'état IDLE à l'état START, autrement dit pour entrer dans le mode de test, le signal syncTREQA doit être activé. Ce signal syncTREQA correspond en fait au signal

TREQA synchronisé sur l'horloge de test choisie, le plus souvent TCLK. Avant d'activer le signal TACK, initiant le mode de test, le contrôleur de test attend que les transferts en cours soient terminés et surtout que le basculement de l'horloge système vers l'horloge de test soit bien effectué. Sur ce schéma on note aussi la présence d'un état LASTREAD et d'un état TURN (TURNAROUND). Ce sont principalement deux états d'attente permettant au bus de se configurer pour effectuer un changement de sens de transfert. Ces états sont nécessaires lorsqu'on passe d'une lecture de données à une écriture.

AVANTAGES LIÉS À L'UTILISATION DU BUS AMBA

- La surface additionnelle dûe au test est faible. L'architecture n'introduit pas de bus supplémentaire et donc évite d'éventuels problèmes de routage du circuit.

- Cette architecture favorise la réutilisation des données de test accompagnant un coeur. Lorsque l'IP est utilisé dans un autre SoC intégrant ce bus, les vecteurs de test qui ont été définis sont réutilisables, seule la gestion des adresses correspondant à l'IP est à redéfinir.

- L'accès en parallèle aux entrées/sorties du coeur réduit le temps de test.

- Le test d'un coeur est indépendant de son environnement.

INCONVÉNIENTS DE CETTE APPROCHE

- Cette architecture favorise principalement le test fonctionnel. L'ajout de matériel permettant d'utiliser les techniques de scan ou de BIST est difficile à mettre en oeuvre. Nous avons vu que pour accéder aux entrées/sorties de coeur non connectées au bus, il fallait rajouter de façon ad-hoc des blocs de test supplémentaires. Le test des interconnexions entre coeurs nécessite aussi l'ajout de matériel dédié.

- L'architecture est figée, offrant peu de degrés de liberté dans l'établissement d'un compromis surface ajoutée/temps de test. Le fait de réutiliser le bus système fixe la largeur du TAM.

- On ne peut tester qu'un IP à la fois.

Conclusion

Globalement on peut dire que les TAM basés sur la réutilisation du matériel fonctionnel sont intéressants du point de vue de la surface rajoutée, cependant ils s'avèrent contraignant pour l'ingénieur chargé de définir la stratégie de test. Pour un test de qualité, avec une couverture de fautes élevée, ce type de TAM en général ne s'utilise pas seul. C'est le cas du bus AMBA dont l'architecture est complétée chez Philips par une approche structurelle permettant d'obtenir le taux de couverture désiré ([FW98]).

2.4.2 Ressources supplémentaires dédiées au test

Nous avons vu que l'approche décrite précédemment pouvait être insuffisante pour atteindre certains niveaux de qualité du test. Pour pallier ce problème une autre approche consiste à intégrer dans le SoC du matériel supplémentaire, non fonctionnel, spécifique au test. Les techniques utilisées au niveau SoC sont issues des techniques de DFT utilisées au niveau circuit. Certaines architectures de test au niveau SoC sont basées sur la réutilisation des techniques du Boundary Scan et du BIST.

Adaptation des techniques du Boundary Scan

Pour des systèmes sur puce intégrant uniquement des coeurs Boundary Scan, des architectures ont été proposées. Pour ce type de SoC, une des difficultés consiste à définir un mécanisme de test et de contrôle hierarchique des différents coeurs, tout en ne violant pas le standard IEEE 1149.1.

Architecture TLA

Une première architecture a été proposée par Lee Whetsel ([Whe97]) : l'architecture TLA (TAP Linking Architecture)(figure 2.15).

Les blocs logiques ou les coeurs qui ne sont pas équipés de wrappers ou qui n'intègrent pas d'éléments Boundary Scan, les NTCs (Non Taped Cores), peuvent être regroupés et équipés des éléments de ce standard, notamment du contrôleur (ici TAP1).

Cette architecture est basée sur l'utilisation d'un contrôleur central au niveau du système, le TLM, chargé d'attribuer le contrôle du test à l'un des trois TAP. Le TLM (TAP Linking Module) est constitué principalement d'un contrôleur TAP et de registres. Les signaux TMS, TCK, TDI et TDO sont distribués à chacun des trois modules boundary scan du SoC, en plus du TLM. En fonction des instructions chargées dans le TLM, les données présentes sur TDI seront appliquées soit au niveau du système, soit à l'un des coeurs. Le transfert du contrôle sur le bus TDI d'un TAP à l'autre est assuré par le TLM.

Cette architecture répond au problème du test hiérarchique et ne viole pas le standard IEEE 1149.1. Cependant le standard définit comme optionnelle l'utilisation de certains signaux comme le signal ENABLE. Or dans cette architecture sa présence est obligatoire et beaucoup de coeurs n'en sont pas équipés. Les TAPed cores peuvent être d'anciens circuits réutilisés comme coeurs et ne disposent que des quatres signaux obligatoires (TMS, TCK, TDI, TDO) et éventuellement TRST. Cela pose le problème de l'intégration de ces coeurs dans ce type d'architecture. Pour réutiliser ces coeurs et les intégrer dans une architecture

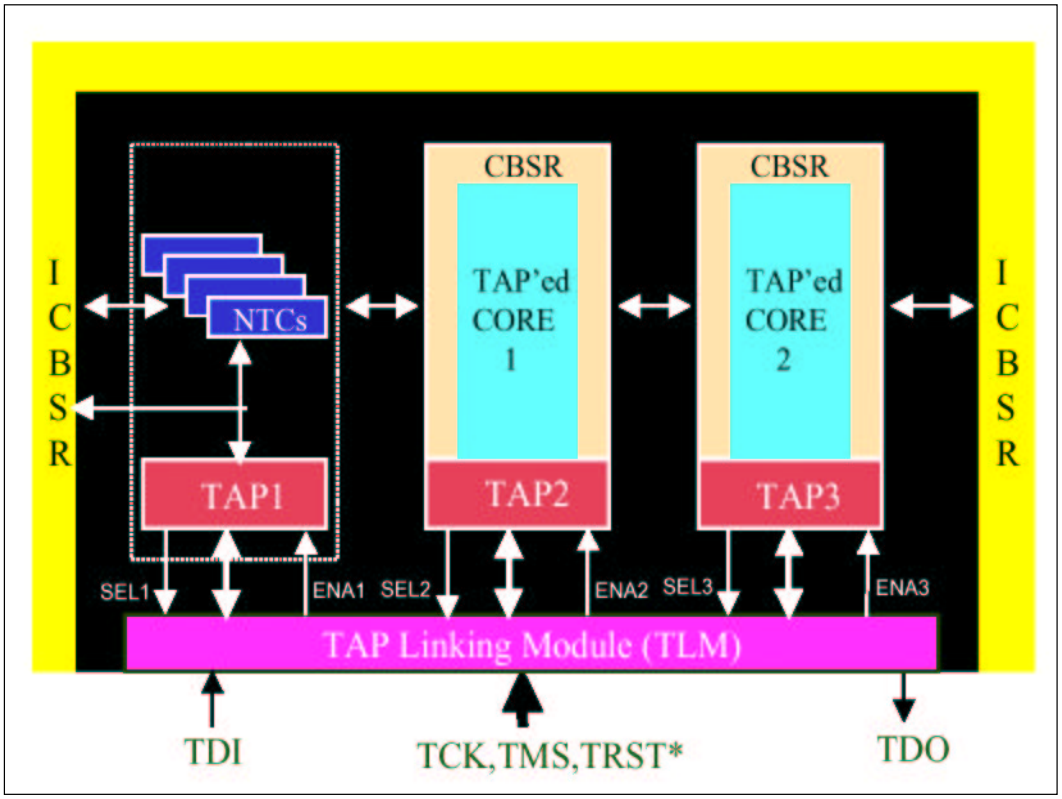


Figure 2.15: Architecture TLA

TLA, il devient nécessaire de passer par une étape de conception pour les adapter à l'architecture. De plus, le fait d'intégrer cette nouvelle broche ENABLE implique que les vecteurs de test délivrés avec le coeur doivent être redéfinis. Cette architecture ne va pas dans le sens du "design-reuse" et du "test-reuse" de plus en plus nécessaires.

Architecture HTAP

Une autre architecture visant le même objectif a été présentée par Debashis Bhattacharya dans [Bha98]. L'architecture HTAP (Hierarchical Test Access Port) est principalement constituée d'un "Snoopy TAP" et d'un "switch" programmable (figure 2.16).

Le "switch" est un module chargé d'activer et d'aiguiller les signaux TDI, TMS et TDO vers le TAPed core prévu. Il contient un registre de données, le registre SDR (Switch Data Register). Le chargement d'une valeur dans ce registre permet de positionner le switch dans la configuration voulue.

Le snoopy TAP (SN) est un controleur TAP modifié, composé de deux parties (figure 2.17) : une première partie correspondant au controleur TAP du boundary scan et une partie "snoopy" constituée également des 16 états du controleur TAP. Le rôle de la première

partie est de contrôler le test des NTCs (Non Taped Cores) et le test au niveau système. La partie snoopy attribue la gestion du contrôle à l'un des deux TAPed core.

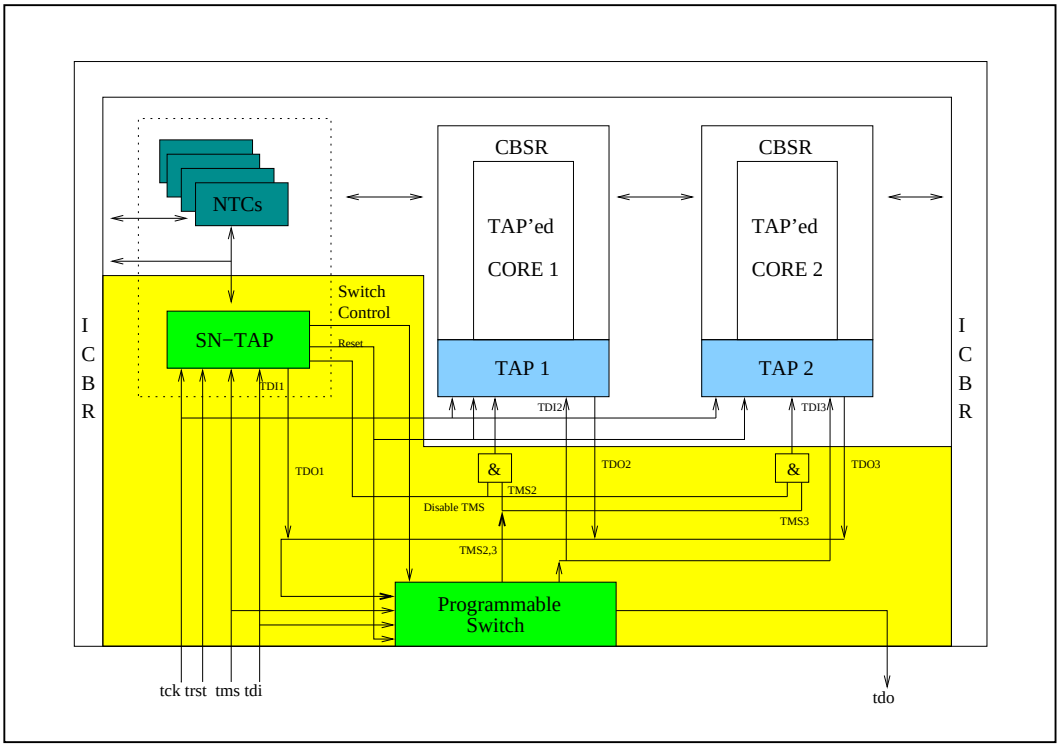


Figure 2.16: Architecture HTAP

Au niveau des NTCs, en plus du snoopy TAP et des registres boundary scan usuels, un registre de contrôle est ajouté. Ce registre, appelé registre CR (Control Register), est contitué de trois champs. Un champ pour le mode (mode normal ou mode snoopy), un champ ref-C0 et un champ ref-C1. Ces deux derniers champs correspondent aux valeurs de deux compteurs. Ces compteurs permettent de sortir de la partie snoopy vers la partie du snoopy TAP correspondant au mode de fonctionnement normal du TAP (figure 2.17).

Une séquence de test avec cette architecture HTAP peut se dérouler comme suit :

1 Fonctionnement normal du snoopy TAP (chargement et mise à jour du registre Boundary Scan du SoC par exemple).

2 Chargement de l'instruction selectionnant le registre CR comme registre de données.

3 Chargement du registre CR (Mode=1, ref-C0=5, ref-C1=6)

4 Chargement de l'instruction selectionnant le registre SDR comme registre de données.

5 Chargement du registre SDR (le contrôle sera transféré au TAP2)

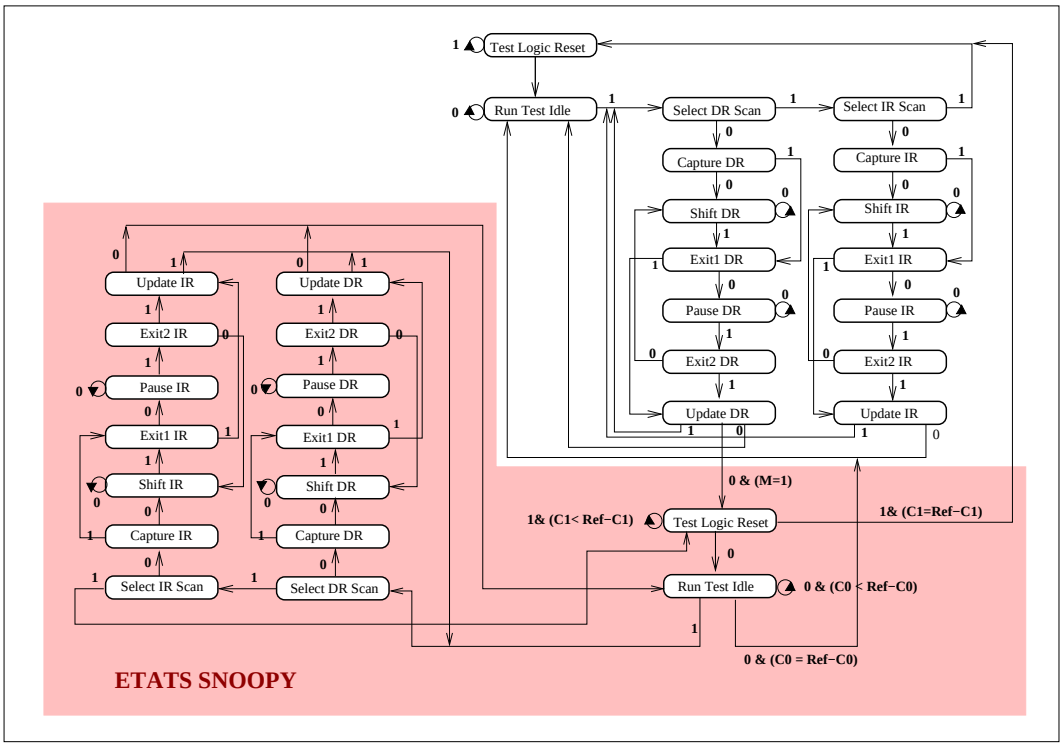


Figure 2.17: Diagramme d'états du Snoopy TAP

6 Activation de TMS2. On entre dans le mode snoopy. L'état courant de la partie snoopy du controleur est le même que celui du TAP2. On effectue le test du core 2.

7 On passe 5 cycles (ref-C0) dans l'état SN-RT-Idle puis on sort de la partie snoopy vers l'état Run Test Idle de la partie principale. Le TAP2 n'est plus actif.

8 Chargement de l'instruction selectionnant le registre SDR comme registre de données.

9 Chargement du registre SDR (le controle sera transféré au TAP3).

10 Activation de TMS3...

L'architecture HTAP permet, comme l'architecture TLA, d'effectuer un test hiérarchique des éléments d'un SoC. Elle possède l'avantage de favoriser l'intégration de TAPed cores sans avoir à les modifier.

Cependant ces deux architectures n'offrent pas la possibilité de tester plusieurs coeurs en même temps, que ce soit en parallèle ou en série.

Autres approches

Pour pouvoir tester plusieurs coeurs en même temps une solution a été proposée par A. Benso et al dans [BBG+98]. Cette approche vise les SoC intégrant des "soft cores". Un

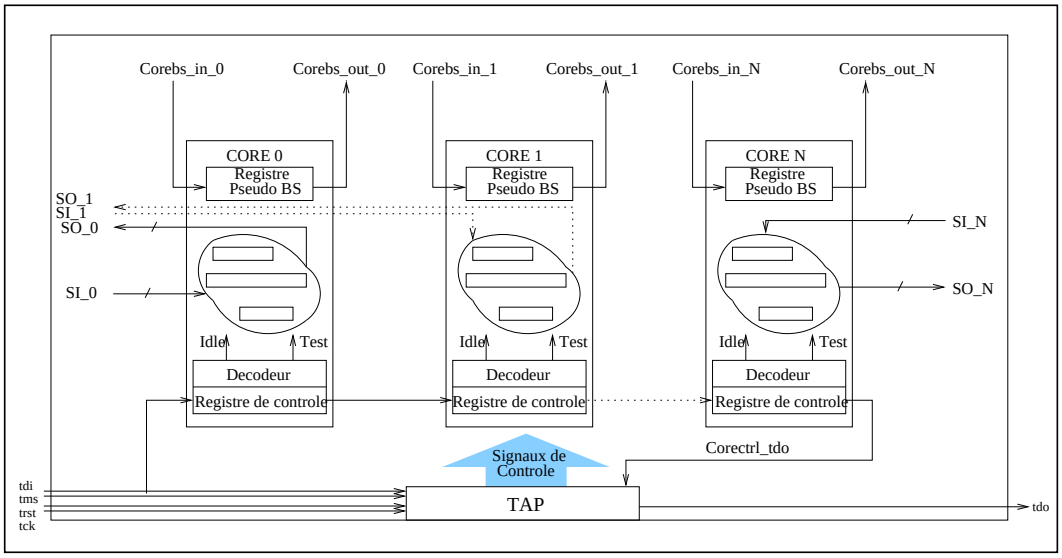


Figure 2.18: Approche BS permettant le test de plusieurs coeurs en parallèle

wrapper est ajouté autour de chaque coeur (figure 2.18). Il contient un registre pseudo BS (Boundary Scan) et une partie contrôle. Le terme pseudo est utilisé car les cellules du registre ne sont pas tout à fait les mêmes que celles du standard, elles sont contrôlées par les signaux *idle* et *test* issus du décodeur. Le contrôleur TAP central est légèrement modifié de façon à générer les signaux contrôlant à la fois les éléments BS au niveau SoC et les modules de contrôle au niveau core. On remarque que dans cette architecture TDI et TDO ne sont pas connectés aux entrées/sorties des pseudo registres BS. Un bus de test SI/SO transporte les données de test vers chaque coeur. La largeur de chaque bus SI/SO est égale au nombre de chaînes de scan interne de chaque IP.

Si un mécanisme de multiplexage des différents SI/SO n'est pas prévu pour compléter cette architecture, la surface additionnelle due au test risque de devenir vite élevée : le nombre de broches de test au niveau du SoC serait dans ce cas au moins égal à deux fois le nombre de chaînes de scan internes de l'ensemble des coeurs, ce qui pourrait poser un problème pour le "packaging" du circuit. Cependant, cette approche offre l'avantage de tester en parallèle plusieurs IPs et donc d'avoir un gain important en ce qui concerne le temps de test.

Une solution permettant d'optimiser le temps de test en utilisant les éléments du Boundary Scan a été présentée dans [KNS98]. L'architecture est basée sur l'utilisation de switchs (figure 2.19) et de cellules BS partagées entre deux coeurs. Cette architecture, bien qu'elle soit basée sur un transport en série des données de test, permet d'atteindre le coeur choisi en un temps réduit. Cependant, comme on le voit sur la figure, certaines

cellules BS doivent être bidirectionnelles.

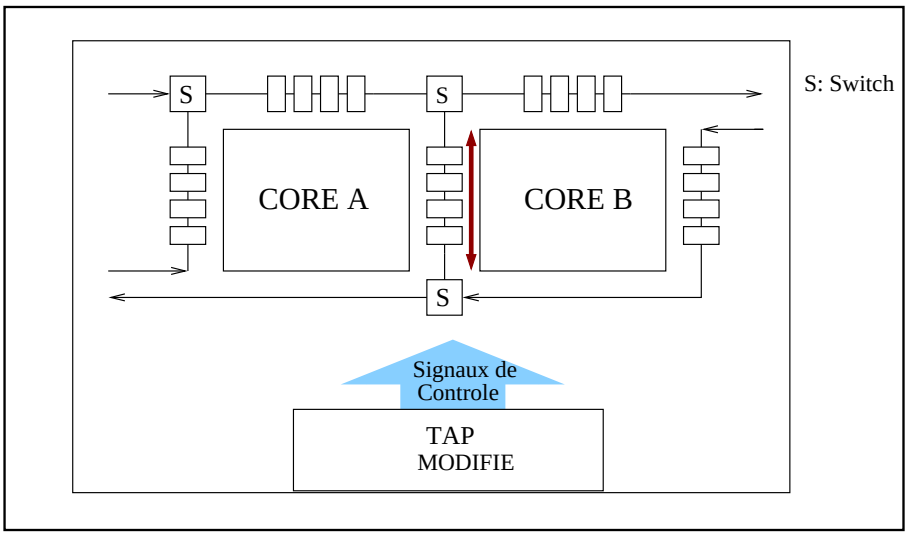


Figure 2.19: Approche Boundary Scan optimisé

Cette solution offre la possibilité de réduire le temps de test et la surface additionelle globale. Néammoins avec cette architecture deux coeurs adjacents ne peuvent être testés en même temps. De même se pose le problème du test des interconnexions entre ces coeurs.

Adaptation des techniques du BIST

Les circuits intégrés étant constitués d'un nombre de plus en plus élevé de transistors, le nombre de vecteurs de test nécessaires croît lui aussi de façon importante. Le problème du temps de test et de la taille des mémoires contenant ces vecteurs devient prépondérant. Nous avons vu dans la première partie de ce chapitre qu'une solution à ces problèmes serait d'intégrer dans le circuit les "sources" et les "sinks". Les architectures basées sur l'utilisation des techniques du BIST s'inscrivent dans ce type d'approche. Deux d'entre elles sont décrites ci-dessous.

Architecture HD-BIST

La première, comme pour le Boundary Scan, s'intéresse au problème du test hiérarchique. L'architecture HD BIST (Hierarchical Distributed Bist), détaillée dans [BCC+99], est représentée sur la figure 2.20. Cette architecture est constituée de blocs contenant des coeurs "bistés", reliés à un bus de test appelé Test Chain. Les blocs ainsi chaînés sont reliés à un processeur de test qui gère le séquencement des tests à effectuer. Un bloc contient un coeur, un contrôleur de BIST et une interface qui fait le lien avec le bus de test. Les

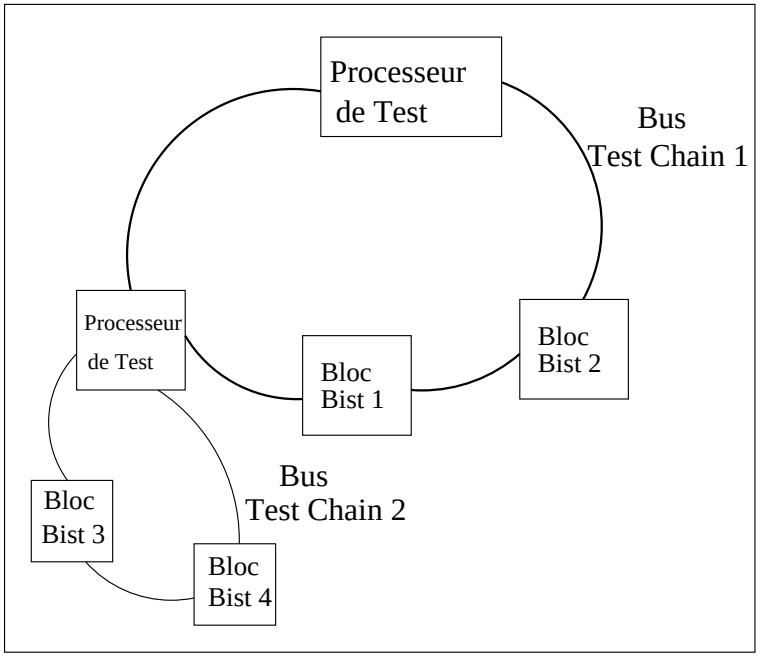


Figure 2.20: Architecture HD-BIST

échanges entre les différents blocs et le processeur sont basés sur l'utilisation d'un protocole de communication de type LAN (Local Area Network). Pour ce qui est de la hiérarchie, chacun des blocs peut correspondre lui même à un sous réseau constitué d'un processeur et de blocs bistés.

Cette architecture permet d'effectuer un test séquentiel ou un test en parallèle. Cela dépend des contraintes physiques du circuit : le test en parallèle de plusieurs blocs entraîne une augmentation de la consommation globale ainsi qu'une augmentation du bruit et des problèmes qu'il peut entraîner. L'inconvénient principal de ce type d'architecture est la surface de test rajoutée pour le BIST, aussi bien au niveau coeur qu'au niveau système. De plus le test des interconnexions entre blocs ne semble pas être pris en compte.

Architecture Modular Logic BIST

Le deuxième exemple d'architecture que nous avons choisi pour illustrer l'utilisation des techniques du BIST est celui présenté dans [RT98], l'architecture étant nommée "Modular Logic BIST" (figure 2.21).

Au niveau système, un contrôleur TAP pilote à la fois un contrôleur de BIST pour la partie logique des cores et un contrôleur de BIST pour la partie mémoire. Le contrôleur de la partie logique est constitué d'un LFSR, d'un analyseur de signature, d'un compteur de vecteurs et d'un compteur de décalages. Une version programmable de ce contrôleur existe

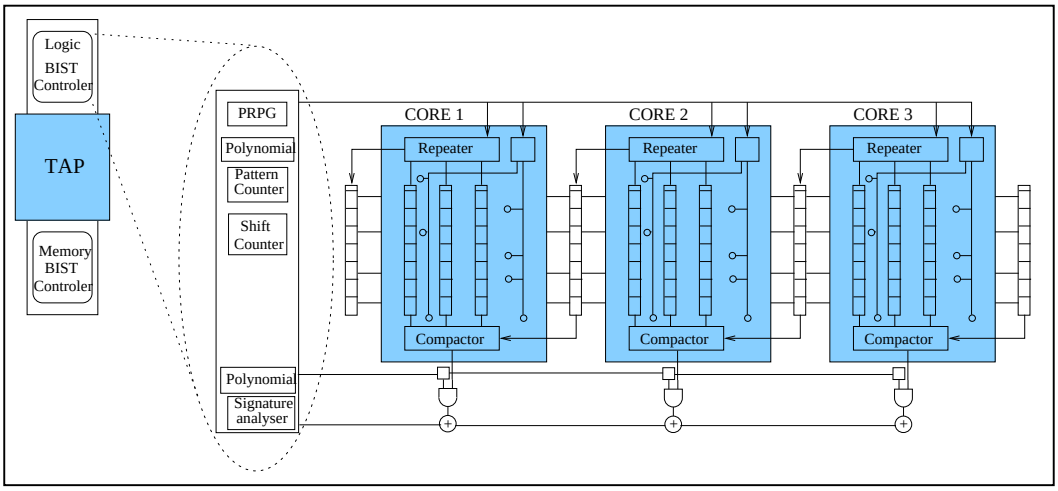


Figure 2.21: Architecture Modular Logic BIST

et permet d'initialiser les polynomes du LFSR et de l'analyseur de signature. Les cores à tester avec cette architecture doivent être full scan et contenir un répéteur ("repeater") et un compacteur de réponses. Les données de test sont envoyées en série par le LFSR vers le coeur cible et le répéteur de celui-ci les convertit en données parallèles à appliquer aux différentes chaînes de scan. Les signatures des différents coeurs sont mélangées pour générer une signature finale qui sera traitée dans le contrôleur central. La participation de la signature d'un coeur au mélange final dépend de l'initialisation du polynôme du contrôleur. Cela permet de faire du test de diagnostic.

Le principal avantage de cette architecture est qu'elle permet de réduire la surface de test additionnelle par rapport à une architecture où chaque coeur inclus un LFSR, un MISR et un controleur de BIST. Elle permet de tester en parallèle plusieurs coeurs et de faire du test de diagnostic. Cependant, elle impose aux fournisseurs d'IP une certaine façon de concevoir leurs composants.

2.4.3 Approche bus supplémentaire dédié au test de SoC

Les approches Boundary scan et BIST sont intéressantes pour le test des SoC mais comme nous l'avons vu elles possèdent leurs limites. Lorsque le temps de test devient un facteur critique, le chargement en série des chaînes de scan des différents coeurs n'est plus possible. L'utilisation du seul couple TDI/TDO pour acheminer les données de test devient insuffisant. De même certains coeurs ne sont pas bistés et sont prévus pour être testés par les techniques de scan classique, et peuvent contenir de nombreuses chaînes de scan. Dans ce cas l'approche bus de test supplémentaire s'impose.

Il existe principalement trois types de bus de test utilisés comme TAM :

- Le bus de type multiplexé (figure 2.22) : l'accès à un coeur se fait en parallèle à travers un multiplexage des N fils.

- Le bus de type "daisy chain" : le bus de largeur N connecte les différents coeurs en série.

- Le bus de type "scalable" : c'est un bus dont l'architecture n'est pas figée. La largeur du bus peut varier en fonction de différentes contraintes (temps de test, surface, ...).

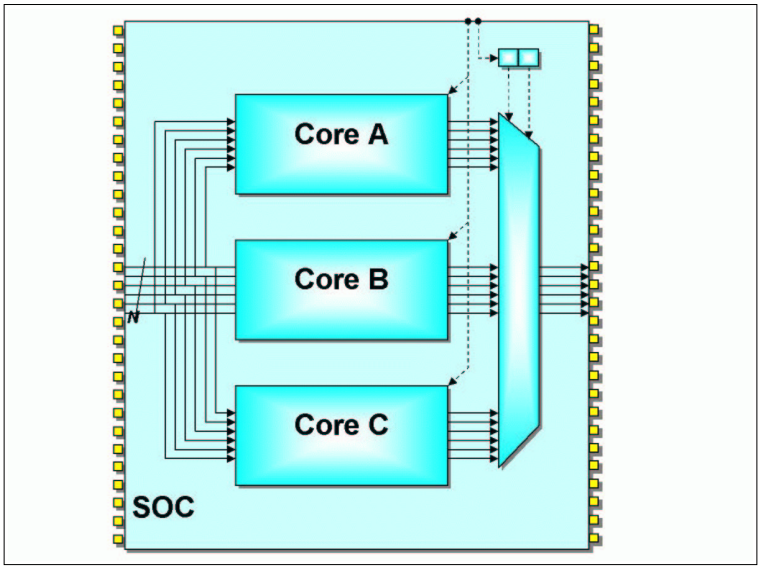


Figure 2.22: Bus de type Multiplexé

Les bus de type multiplexé

Les bus de type multiplexé permettent de tester un coeur directement, indépendamment de son environnement. La conversion des vecteurs de test du niveau coeur au niveau système est facilitée avec ce type de bus. Il permet aussi de simplifier les phases de mise au point et de diagnostique du coeur. Cependant le nombre d'entrées/sortie de l'IP core sont le plus souvent supérieure aux nombres de broches du SoC. Ceci implique que des mécanismes de conversion des données (parallèle/série et série/parallèle) doivent être mis en place. Les paramètres du bus dépendent des coeurs utilisés et de leur nombre. Cela fait de l'architecture à bus multiplexé une architecture non scalable et qui doit être redéfinie pour chaque SoC. De plus le routage et le multiplexage des différents coeurs peut s'avérer coûteux en termes de surface additionnelle.

Les bus de type daisy-chain

Les bus de type daisy-chain peuvent être considérés comme les duals des bus de type multiplexés. Ils possèdent l'avantage d'être moins coûteux en surface supplémentaire mais cela se fait au détriment du temps de test.

Les bus de type scalables

Pour ce qui est des bus scalables, leurs architectures étant non figées, leur implémentation peut s'adapter en fonction des différentes contraintes imposées. Pour illustrer ce type de bus nous allons décrire les trois principales architectures présentes dans la littérature.

Architecture Test Bus

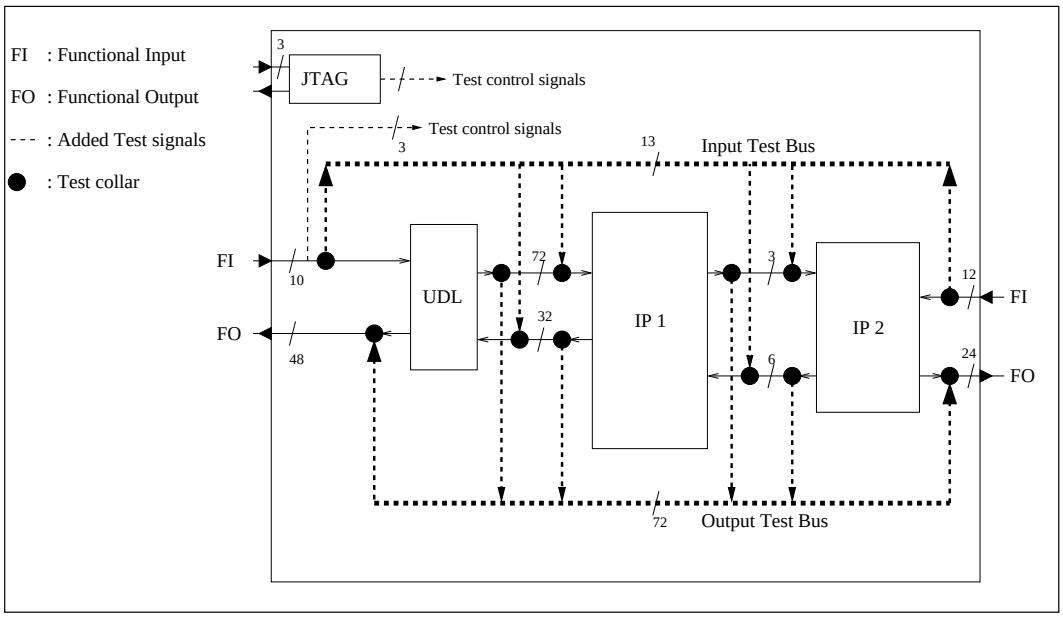


Figure 2.23: Architecture Test Bus

L'architecture "Test Bus" développée par Duet Technologies ([VB98]) est basée sur la réutilisation des broches d'entrée/sortie fonctionnelles du SoC (figure 2.23). Un ou plusieurs bus sont connectés sur ces broches pour transporter les données jusqu'au coeur concerné. Généralement le SoC contiendra un bus de test transportant les données à appliquer et un bus de test pour acheminer les réponses vers les broches de sorties. Si certaines broches sont bidirectionnelles, le SoC peut inclure aussi un bus de test bidirectionnel. Ces bus sont connectés en parallèle aux différents coeurs et aux blocs définis par l'intégrateur (UDL).

Chaque coeur est pourvu d'un wrapper appelé "test collar" dont les cellules sont de complexité variable. Cette complexité dépend du type de coeur à encapsuler et des contraintes liées aux performances et à la surface. Les bus étant branchés sur les broches du SoC, des techniques de multiplexage sont mises en place. Le contrôle de cette architecture peut se faire à travers la réutilisation de certaines broches du SoC ou grâce à un contrôleur boundary scan intégré dans le SoC.

Cette architecture a pour avantage de ne pas augmenter le nombre de broches du SoC tout en utilisant des bus de largeur importante. Elle permet d'accèder aux blocs UDL et de tester les interconnexions entre blocs. L'intégrateur système en fonction des contraintes peut définir la largeur des bus qu'il va utiliser et le nombre de coeurs qu'ils desservent.

L'inconvénient avec cette architecture est que pour chaque bus de test un seul coeur est actif à un moment donné. Le test de plusieurs coeurs en même temps n'est pas possible sur le même bus.

Architecture Ports de Test Adressables

Une autre architecture basée sur la réutilisation des entrées/sorties fonctionnelles du système intégré a été développée par Lee Whetsel chez Texas Instruments ([Whe99]). Dans cette approche (figure 2.24), chaque coeur est connecté aux bus de test à travers un bloc appelé port de test adressable.

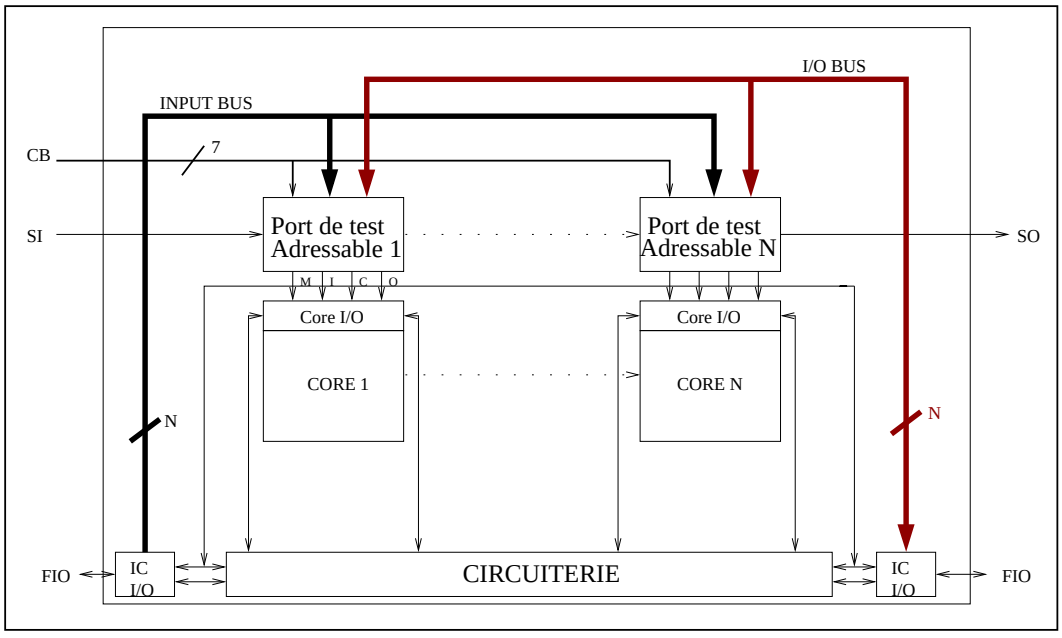


Figure 2.24: Ports de Test Adressables

Chaque port de test contient un registre d'adresse, un contrôleur de test sous la forme d'un automate à états finis, un port d'entrée et un port d'entrée/sortie. Il fournit au coeur des signaux de mode (M :mode test, mode normal), de controle (C), et des signaux de données (I et O). Lors de l'initialisation, les registres d'adresse sont chargés en série grâce à SI et SO. Ensuite une valeur est appliquée au bus d'entrée. L'activation d'un port de test adressable (PTA) se fait si la valeur de cette donnée est identique à celle présente dans le registre d'adresse. Le PTA est géré par un bus de contrôle (CB) issu des broches du SoC. Parmi les signaux constituant CB on peut noter la présence de deux signaux prévus pour le test analogique. Dans le PTA, le port d'entrée peut contenir un générateur de vecteurs de type LFSR. Le port de sortie peut contenir un comparateur ou un compacteur de données.

Cette approche autorise l'utilisation d'une large palette de tests différents. Elle permet d'accéder en parallèle aux entrées/sorties d'un coeur, qu'il soit encapsulé ou pas. Si celui-ci est équipé d'un wrapper le test d'interconnexion est alors possible. Les PTA permettent d'effectuer l'autotest des coeurs, si ceux-ci n'intègrent pas les éléments nécessaires. Les coeurs analogiques peuvent aussi être testés dans cet environnement.

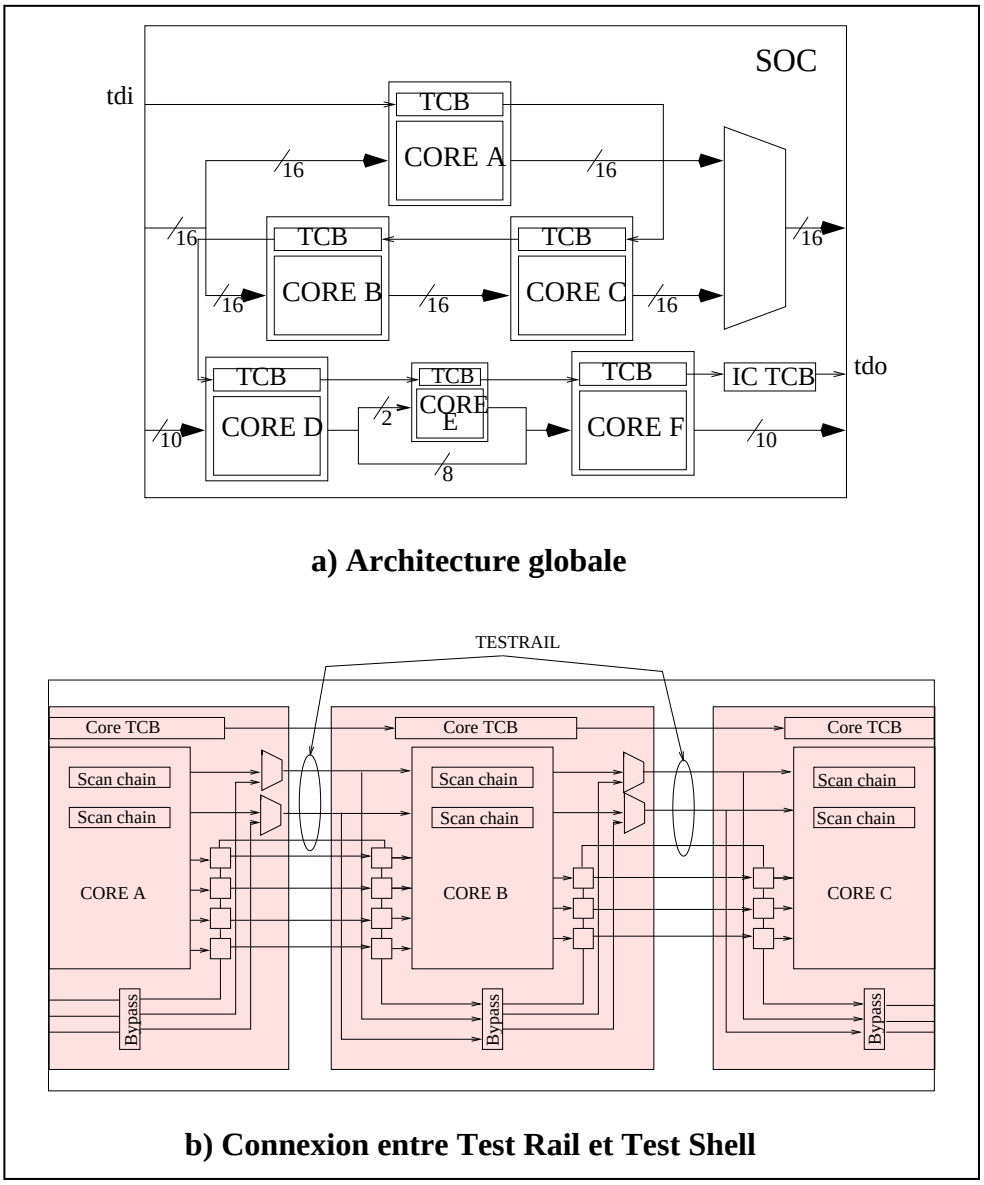
Bien que possédant de nombreuses fonctionnalités, les ports de test adressables ne permettent pas d'effectuer le test simultané de coeurs différents, branchés sur le même bus. De plus la complexité d'un PTA peut s'avérer coûteuse en surface.

Architecture Test Rail

Une architecture combinant l'accès série et l'accès en parallèle d'un bus de test a été développée par Philips et présentée dans [M+98].

Ce TAM scalable est composé principalement d'un ou plusieurs bus de test, appelés "Test Rail", connectés aux différents coeurs à travers un wrapper appelé "Test Shell" (figure 2.25), wrapper semblable à celui développé par le groupe VSIA.

Le Test Shell est constitué d'un TCB, d'un registre boundary et d'un registre de bypass. Le TCB (Test Control Block) est un module contenant un registre d'instruction et de la logique de décodage. Il génère une partie des signaux qui contrôlent le registre boundary et le registre de bypass. Grâce au TCB quatre modes de fonctionnement sont possibles : le mode de fonctionnement normal de l'IP, le mode test interne du coeur, le mode test des interconnexions et le mode bypass. Le contrôle du test se fait au niveau coeur mais aussi au niveau système grâce à la présence d'un TCB supplémentaire (figure 2.25 (a)). Les différents TCB sont chargés en série à l'aide de TDI/TDO. Les autres signaux de contrôle peuvent être fournis directement à partir des entrées du SoC ou par l'intermédiaire d'un controleur TAP si le SoC en est équipé. Le registre boundary est composé d'un ensemble



a) Architecture globale

b) Connexion entre Test Rail et Test Shell

Figure 2.25: Les "Test Rail" et "Test Shell" de Philips

de cellules permettant l'isolation du coeur, l'application de stimuli de test et le mode de fonctionnement normal. La largeur du registre de bypass est égale à la largeur du Test Rail (figure 2.25 (b)).

Cette architecture est flexible et scalable. Selon les besoins, un coeur peut être testé directement, en parallèle, ou en série avec un autre coeur. En fonction des contraintes de temps de test et de surface, l'intégrateur système définit le nombre et la largeur des Test Rail. La flexibilité de cette architecture s'applique aussi à l'ordonnancement des tests à effectuer. Sur la figure 2.25 (b) on peut voir qu'avec le même Test Rail les coeurs A et C

peuvent être testés en série pour une session donnée. Lors de la session suivante le coeur B peut être testé individuellement ou en série avec un des deux autres coeurs.

Cette architecture de taille variable offre la possibilité de faire des compromis en fonction du nombre de broches de test du SoC, de la surface additionnelle autorisée et du temps de test.

2.5 Optimisation des TAM

La définition d'un TAM, comme nous l'avons dit dépend de plusieurs critères. Plusieurs études ont été faites pour déterminer les caractéristiques du TAM. Dans ces études le TAM visé est la plupart du temps un bus de test. Nous présentons ici brièvement quelques études ayant trait aux contraintes pesant sur le choix d'un TAM optimal.

Dans [GM01] et [Cha01] des algorithmes sont présentés pour optimiser les largeurs des TAM en fonction du temps de test. Une extension de ces techniques d'optimisation a été proposée dans [Cha00] pour tenir compte dans la définition du TAM des contraintes de placement-routage et de puissance dissipée.

Une approche présentée dans [ICM01] fait entrer la conception du wrapper dans le processus d'optimisation globale du TAM. Pour le wrapper, l'optimisation consiste à connecter entre elles et à équilibrer les chaînes de scan du coeur pour optimiser le temps de test. Le registre WBR du wrapper est considéré comme une chaîne de scan. En ce qui concerne le TAM l'optimisation consiste à déterminer, pour un SoC donné, le nombre de bus de test nécessaires, la largeur de ces bus et le partitionnement des coeurs branchés à ces bus.

La taille du TAM dépend aussi de l'ordonnancement des différents tests à appliquer dans le SoC. Un algorithme tenant compte de ce paramètre et de la puissance dissipée est détaillé dans [LP01].

Pour définir son TAM, l'intégrateur système peut se retrouver devant le problème décrit par la figure 2.26. M représente le nombre de broches d'entrée/sortie de test du SoC et N la largeur du TAM. Lorsque M<N des mécanismes d'expansion de données peuvent être mis en place. Une paire de broches externes peut être par exemple distribuée à plusieurs entrées/sorties du TAM (figure 2.27). Cependant cette expansion se fait au détriment du temps de test. L'expansion puis la décompression de données compressées permet de réduire le temps de test additionnel.

Les techniques de compression de données possèdent plusieurs avantages. Elles permettent de réduire le temps nécessaire à l'application des vecteurs. Elles peuvent réduire

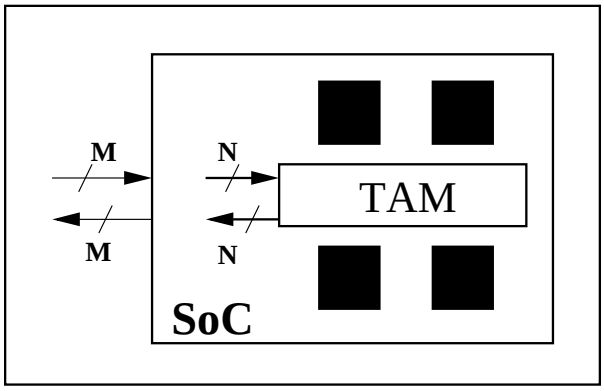


Figure 2.26: Problème : M<N

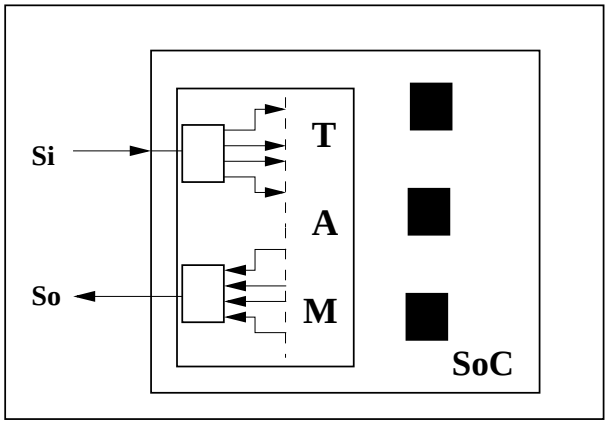


Figure 2.27: Expansion des données

le nombre de broches de test au niveau du SoC mais elles peuvent aussi réduire considérablement la taille des données en mémoire dans le testeur. Ce dernier point n'est pas à négliger car, les SoC étant de plus en plus complexes, les données de test ont tendance à exploser en volume. Cette augmentation du volume a pour conséquence l'utilisation de testeurs de plus en plus coûteux.

Pour avoir moins de données à appliquer aux broches du SoC, on peut soit utiliser des techniques liées au BIST, en transférant une partie des fonctionnalités du testeur dans le circuit, soit compresser les données de test.

Les techniques de compression/décompression actuellement utilisées sont basées sur l'utilisation de codages statistiques et de méthodes probabilistes ([LH87], [WNC87]). Dans ces techniques, un échantillon de donnée est codé selon son nombre d'occurences par rapport à l'ensemble des échantillons. Pour un codage de Huffman, par exemple, la taille de l'échantillon codé sera faible si l'échantillon apparaît souvent. Une technique utilisant une variante du codage de Huffman pour coder les vecteurs de test a été présentée dans

[JGDT99]. Dans cette approche, une variante du codage d'Huffman est utilisée pour éviter que la taille du décompresseur interne au SoC ne devienne trop importante.

On trouve aussi des techniques de compressions utilisant non pas les fréquences d'apparition des échantillons, mais les corrélations entre deux vecteurs successifs. Deux vecteurs successifs peuvent être peu différents et au lieu d'envoyer sur la broche de test le vecteur lui même on peut envoyer un vecteur "différence".

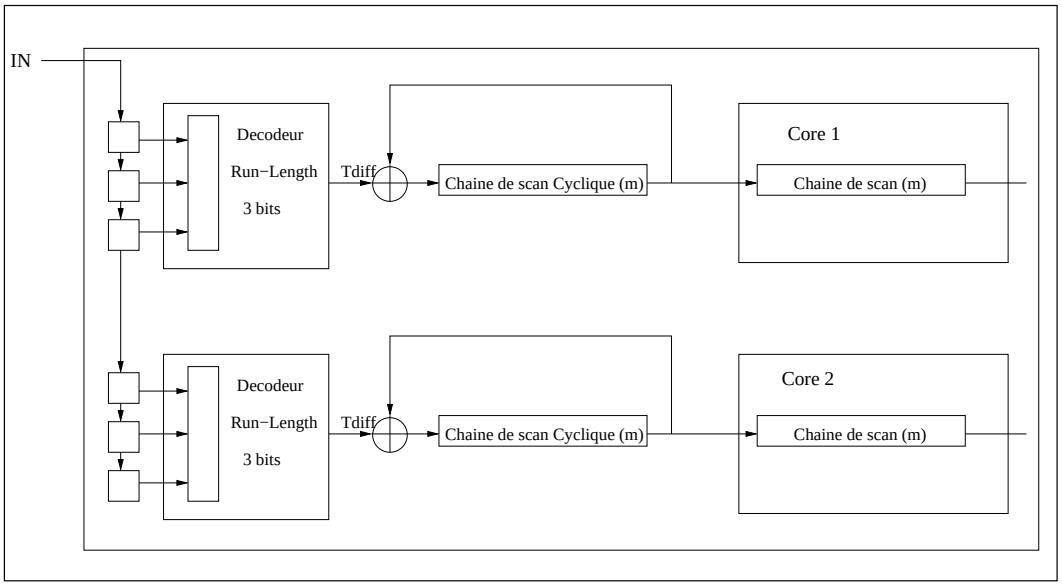


Figure 2.28: Expansion des données

Pour illustrer ces techniques, prenons le cas de celle présentée dans [JT98]. La figure 2.28 décrit le principe utilisé par cette approche. Pour charger la chaîne de scan interne du coeur, une chaîne de scan cyclique est utilisée. Cette chaîne de scan cyclique utilise un ou exclusif qui fait la somme entre la dernier bit de la chaîne et l'entrée Tdiff. La chaîne de scan cyclique contenant le vecteur t1, pour charger t2 il suffit de présenter sur Tdiff non pas t2 mais la différence (t2-t1). En tirant partie du fait que deux vecteurs successifs sont voisins, cette différence sera composée de beaucoup de 0. Coder cette différence devient alors très intéressant. Dans cette approche cette différence est codée en utilisant un codage répétitif ("Run-Length") sur trois bits qui compte le nombre de 0 avant l'apparition d'un 1. La longueur des codes à appliquer en entrée du SoC étant inférieure à celle des échantillons décompressés, on peut alimenter avec une seule entrée plusieurs chaînes de scan.

Une autre approche basée sur l'utilisation des chaînes de scan cycliques est décrite dans [CC01]. Cependant, au lieu d'utiliser un codage répétitif, elle utilise le codage de Golomb.

2.6 Conclusion

Nous avons vu que, globalement, il existe deux approches permettant d'effectuer le test des systèmes intégrés. La première consiste à réutiliser les ressources fonctionnelles pour le test tandis que la seconde est basée sur l'utilisation de matériel supplémentaire dédié. Certaines architectures combinent ces deux façons d'aborder le test des SoC. Cependant, quelle que soit l'approche choisie, les techniques utilisées sont celles issues des techniques de DFT classiques.

La variété des coeurs fournis et le besoin croissant de réutilisation incitent la communauté industrielle à définir certaines normes pour le test. Le groupe IEEE P1500 travaille à cela mais se propose de ne pas s'occuper de la définition du TAM, celle-ci étant laissée à l'intégrateur système. Même si les SoC sont des systèmes complexes, ils sont malgré tout destinés à être intégrés sur des cartes, ils sont donc souvent équipés des éléments du standard IEEE 1149.1. Cependant, avec l'arrivée du standard P1500, se posent les problèmes liés à l'interfaçage des éléments des deux normes. Ces problèmes commencent à être étudiés et une première approche a été récemment présentée dans [WR01]. Ces premiers travaux décrivent comment gérer au niveau système le contrôle des wrapper P1500 par un contrôleur TAP.

Les SoC et les coeurs qu'ils contiennent intégrant de plus en plus de transistors, l'utilisation du boundary scan seul n'est donc plus suffisante pour effectuer le test du circuit. Le plus souvent les éléments boundary scan sont combinés à d'autres techniques comme nous l'avons vu précédemment. C'est le cas chez certains industriels comme Fujistu et Logic Vision qui utilisent le standard IEEE 1149.1 pour piloter les coeurs bistés ([p15]), ou comme Philips qui contrôle le fonctionnement des "Test Shell" ([vBvH99]).

Avec une densité d'intégration sans cesse croissante, l'ordre d'importance des contraintes de conception est bouleversé. Le temps de test par exemple devient une contrainte prépondérante par rapport à la surface ajoutée. Le nombre de vecteurs à appliquer à un SoC croît rapidement. Cela favorise l'utilisation du BIST qui permet un test "at speed" (à la vitesse de fonctionnement du circuit). Le BIST permet de s'affranchir des problèmes de mémoire du testeur. La prépondérance du temps de test sur les autres contraintes joue aussi en faveur d'une utilisation des bus de test, pour une application des vecteurs de test en parallèle.

Les TAM constitués de bus de test possèdent de nombreux avantages. Cependant les architectures actuelles sont basées sur l'utilisation d'un wrapper propriétaire. C'est le cas du "Test Rail" qui est lié au "Test Shell" et du "Test Collar" lié au "Test Bus". Le besoin se fait sentir d'avoir des architectures modulaires qui séparent bien le bus de test du wrapper.

Ce besoin est conforté par l'arrivée prochaine du wrapper P1500.

La pression du "Time to Market" se faisant de plus en plus sentir, et la phase de test devenant la phase critique, le "test-reuse" devient aussi inévitable que le "design-reuse". Les TAM doivent être alors scalables et hiérarchiques.

Chapitre 3

Architectures CAS-BUS

3.1 Introduction

Nous avons vu dans le chapitre précédent que le groupe IEEE P1500 ne se fixait pas comme but de définir un TAM standard. Le choix du TAM est donc laissé à l'intégrateur système. Nous avons vu aussi que le "design-reuse" devait désormais s'accompagner du "test-reuse". Ces constatations nous ont conduit à définir un TAM scalable reconfigurable et compatible avec n'importe quel wrapper. Avec le développement en cours du wrapper IEEE P1500, nous avons développé notre architecture pour qu'elle soit plus particulièrement compatible avec ce wrapper. Dans la première partie de ce chapitre nous décrirons l'architecture CAS-BUS destinée aux SoC contenant des IPs au standard P1500. Les éléments composant cette architecture seront présentés ainsi que la façon de les contrôler.

La seconde partie s'adresse aux SoCs contenant à la fois des IPs au standard IEEE P1500 et des IPs au standard IEEE 1149.1. L'architecture CAS-BUS a dû évoluer pour tenir compte des spécificités de ce type de SoC. Nous présenterons les modifications qui ont été apportées aussi bien au niveau de l'architecture qu'au niveau du contrôle.

3.2 Architecture pour les coeurs équipés de wrappers

3.2.1 Description de l'architecture

CAS-BUS est un TAM, sa fonction est donc de transporter les données de test des entrées du SoC vers les entrées du (ou des) coeur(s) et d'acheminer les réponses des sorties

du (ou des) coeur(s) vers les sorties du SoC (figure 3.1). Il est constitué principalement de deux éléments :

- **Un ensemble de CAS ("Core Access Switch")**. Chaque IP possède son propre CAS. Le CAS est un routeur programmable qui permet d'aiguiller les données présentes sur le bus de test vers le coeur. Ces données sont aussi aiguillées du coeur vers le bus de test. Le CAS est connecté au coeur à travers deux bus de largeur P (un pour les entrées et un pour les sorties).
- **Un bus de test**, qui correspond à un ensemble de fils qui connectent les CAS entre eux et qui sont reliés aux entrées/sorties du SoC. Le bus est de largeur N

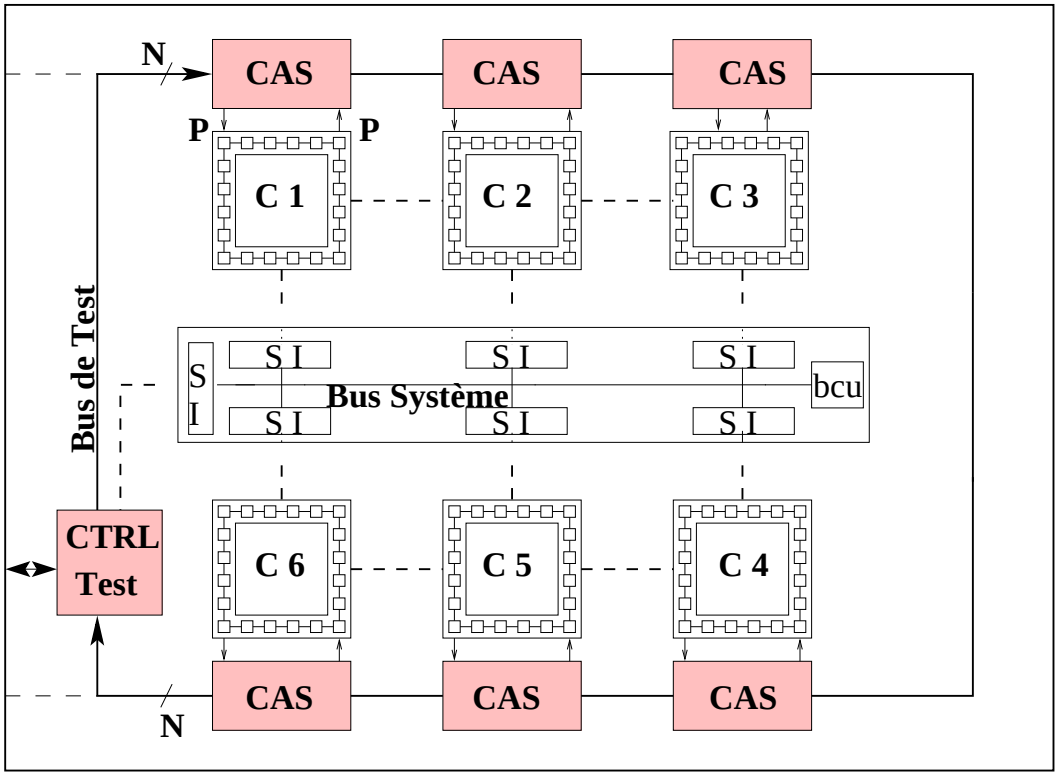


Figure 3.1: Architecture CAS-BUS de base

Chaque CAS choisit parmi les N fils du bus de test les P fils qui vont être connectés au coeur. P correspond au nombre d'entrées de test du coeur (hors entrées de contrôle). Nous considérons qu'il y a autant d'entrées que de sorties de test pour un coeur donné, ce qui implique qu'il y a 2 P connexions entre le CAS et l'IP. Pour un coeur donné, la valeur de P varie en fonction du type de test utilisé :

- Pour les coeurs utilisant les techniques de can, P est égal au nombre de chaînes de scan (figure 3.2-a).
- Pour les coeurs incluant un BIST interne, P est généralement égal à 1 (figure 3.2-b).

– Pour les coeurs utilisant des "sources" et "sinks" externes, P dépend de la nature de ces modules externes. Dans le cas où on utilise un LFSR et un MISR, P peut être égal à 1 (figure 3.2-c).
– Pour le test hiérarchique, nous considérons qu'un coeur peut être lui même un SoC équipé de l'architecture CAS-BUS. Dans ce cas P est égal à la largeur du bus de test interne du coeur (figure 3.2-d).

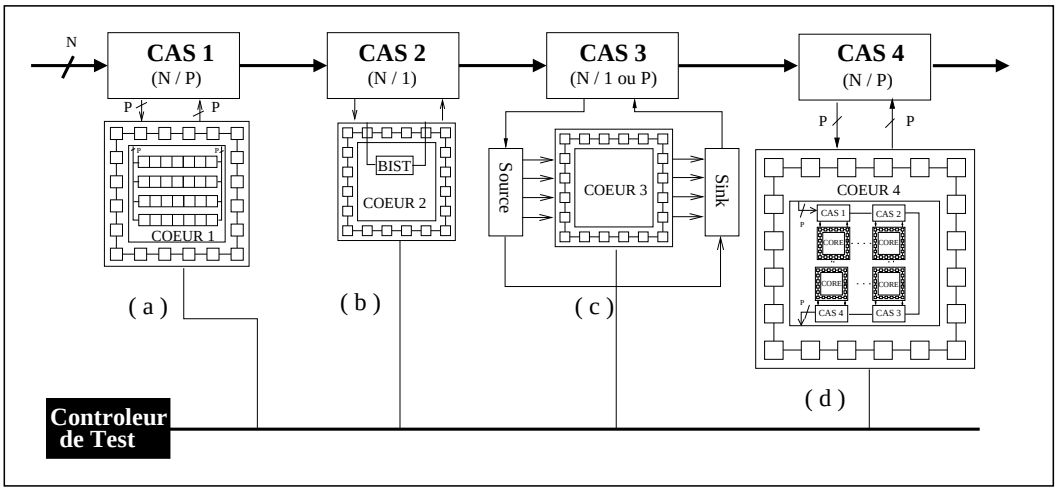


Figure 3.2: Les différents type de coeurs compatibles avec l'architecture CAS-BUS

Tous les signaux qui contrôlent les différents CAS et les coeurs wrappés sont générés par un contrôleur central. Ce contrôleur doit assurer la synchronisation des opérations des différents éléments.

3.2.2 Le Core Access Switch (CAS)

Le Core Access Switch (CAS) est un routeur ou aiguilleur de données (figure 3.3). Il est constitué principalement de deux modules :

– Un registre d'instruction que l'on appellera CIR (CAS Instruction Register). C'est un module à deux étages, l'un pour charger une instruction, l'autre pour la mise à jour de cette instruction. Il est de largeur k.
– Un "Switch". Contrôlé par le CIR, il est chargé d'effectuer le routage proprement dit des données.

Le CAS sélectionne P fils parmi les N entrées e_i. Ces P fils constituent les sorties o_i du CAS qui sont connectées aux entrées du wrapper. Les sorties du wrapper sont connectées aux entrées i_i du CAS. Ces entrées sont connectées aux sorties s_i correspondantes. Les entrées e_i qui ne sont pas sélectionnées par le Switch sont connectées aux sorties s_i

correspondantes.

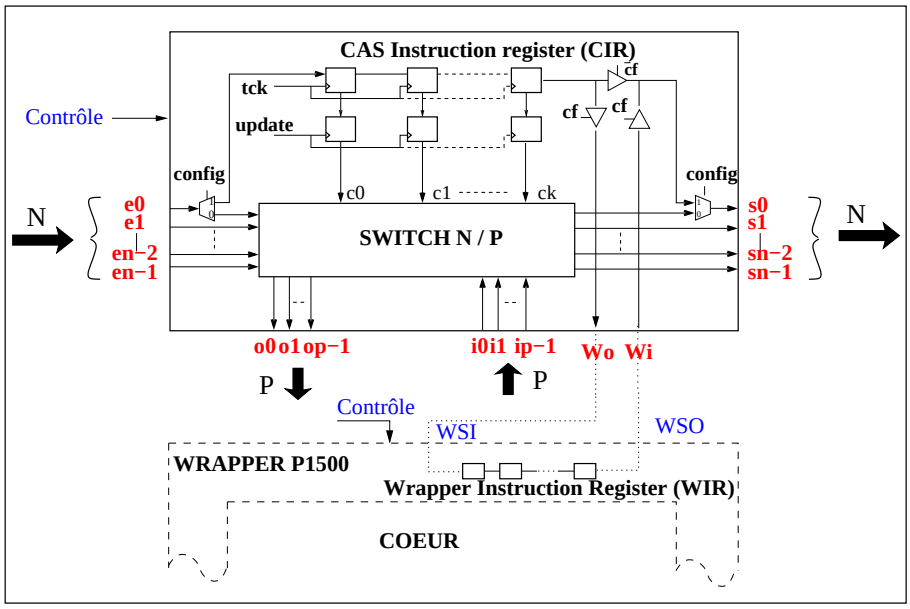


Figure 3.3: Architecture du routeur CAS

Le CAS possède deux modes de fonctionnement :

– Le mode de configuration. Ce mode permet le chargement du registre d'instruction (CIR).
– Le mode de test. L'aiguillage des données aux entrées/sorties du CAS se fait conformément à l'instruction appliquée au Switch. Chaque valeur du mot de controle c_i correspond à un routage précis.

Le passage d'un mode à l'autre s'effectue grâce au signal *config*. Les entrées/sorties e_0 et s_0 servent, selon le mode en cours, soit à charger le registre CIR soit à transporter les données de test.

Le CAS offre aussi la possibilité de charger le registre d'instruction du wrapper (WIR) en série avec le CIR. Ainsi, durant le mode de configuration, on peut configurer à la fois le CAS et le wrapper du coeur. Grâce au signal de controle cf, l'utilisateur peut décider s'il veut charger son CIR indépendamment du WIR. Ce choix se fait en fonction de deux critères :

– Le besoin de charger une nouvelle instruction. Si le mode de fonctionnement du wrapper doit rester le même alors on n'a pas besoin de charger à nouveau le WIR.
– L'intégrateur système a prévu de configurer ses wrappers indépendamment des CAS, à un autre moment ou par d'autres moyens.

Cette fonctionnalité du CAS offre l'avantage d'éviter une session spéciale dédiée au chargement des différents WIR et de ne modifier que les WIR qui ont besoin de l'être. Cela se traduit par une simplification au niveau du programme de test et un petit gain en temps de test.

Implémentations du Switch

Le Switch est le module principal du CAS. C'est lui qui doit implémenter les différents schémas de routage. Le nombre de schémas est égal au nombre de facons de selectionner P fils parmi N. Nous appellerons ce nombre m. Si l'on considère que le Switch est en deux parties, une qui fournit les données au wrapper et une qui les reçoit, il y a alors 2m combinaisons à gérer. En fonction de N et P ce nombre de combinaisons peut exploser très rapidement. Comme la surface du Switch dépend du nombre de combinaisons à implémenter, nous avons décidé de limiter le nombre de combinaisons d'une part et de définir deux architectures différentes du Switch d'autre part. Ces deux architectures possèdent des avantages et des inconvénients qui seront discutés.

Limitation du nombre de combinaisons.

Pour réduire le nombre de combinaisons à implémenter nous avons construit le switch à partir de cette définition : quand une entrée e_i est connectée à une sortie o_j, l'entrée i_j correspondante est connectée à la sortie s_i. Cette définition permet de s'affranchir d'un contrôle différent pour chacune des deux parties du Switch. Au niveau du contrôle, les deux parties sont symétriques et de ce fait le nombre de combinaisons pour l'ensemble du Switch passe de 2m à m. Cette réduction du nombre de combinaisons permet de simplifier le contrôle du routage des données et de réduire la surface du Switch.

Les deux types de Switch

Les deux architectures que nous avons définies se différencient principalement par le nombre de décodeurs qu'elles incluent (figure 3.4). Nous avons défini un Switch avec un seul décodeur et un Switch avec autant de décodeurs que d'entrées, c'est-à-dire N. Le controle du routage des données est soit centralisé, soit distribué.

Switch avec un décodeur

Dans cette architecture (figure 3.4), le décodeur doit permettre d'effectuer l'implémentation de toutes les combinaisons. P fils doivent être sélectionnés parmi N. On peut prendre P fils parmi N, mais une fois ces P fils sélectionnés, on peut les arranger de P ! façons différentes. m correspond donc au nombre de combinaisons C(N,P) multiplié par P !. A ces

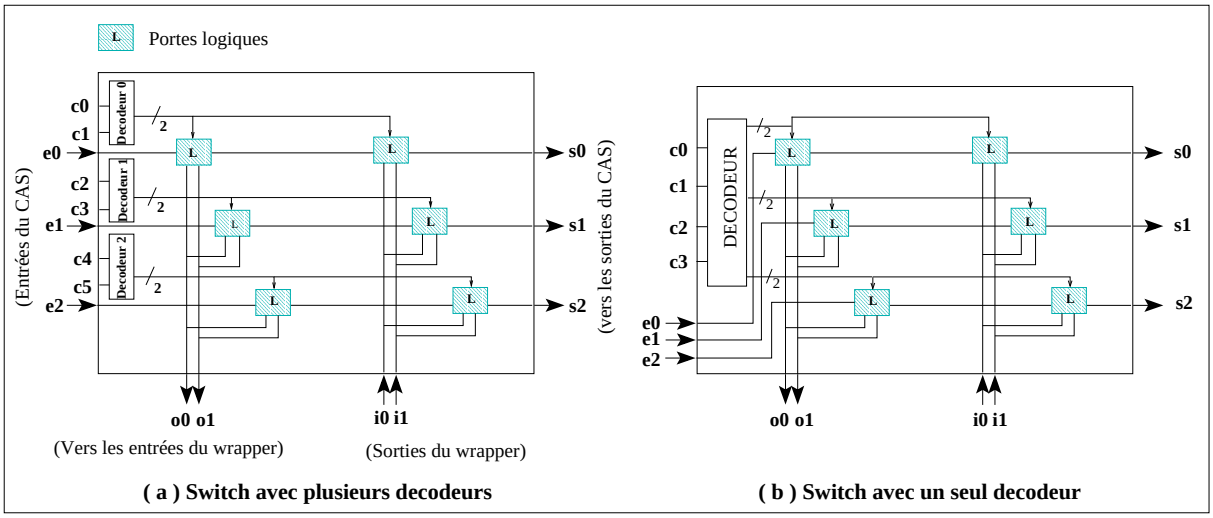


Figure 3.4: Les deux types de Switch

combinaisons on se propose d'ajouter deux combinaisons supplémentaires. On veut pouvoir connecter les entrées e_i directement aux sorties s_i, pour fournir un mode bypass. On veut aussi pouvoir mettre en haute impédance les entrées/sorties du Switch lorsqu'on charge le registre CIR. Le nombre de combinaisons totales que doit permettre le décodeur est donc :

m=N !/((N-P) !) + 2

Si k correspond au nombre de bits de contrôle du décodeur (c0...ck), alors :

$k = \lceil log_2(m) \rceil = \lceil log_2(m) \rceil (N!/((N-P)!) + 2)$

Switch avec plusieurs décodeurs

Le Switch contient un décodeur par entrée e_i, soit N décodeurs au total. Chacune de ces entrées peut être connectée à une des sorties o_i ou à la sortie s_i. Pour chaque décodeur le nombre de combinaisons total est donc :

m=P+1.

Chaque décodeur devra être contrôlé par $\lceil log_2(P+1) \rceil$ bits. Pour un Switch avec N décodeurs, on aura alors :

$k = N.\lceil log_2(P+1) \rceil$

Pour les deux types de Switch, une bascule est insérée entre e_i et s_i (non représentée sur la figure). Le chemin e_i/s_i passe par cette bascule lorsque l'entrée e_i est connectée directement à la sortie s_i. La présence de ces bascules est nécessaire si on veut éviter les problèmes de délais qui peuvent apparaître lorsque certaines lignes du bus de test sont

mises en mode bypass dans plusieurs modules CAS consécutifs. On ne met pas de bascules sur les chemins e_i/o_j et i_j/s_i puisque les entrées/sorties i_j/o_j sont connectées au wrapper à l'entrée duquel nous supposons qu'il y a des bascules.

Nous verrons dans le chapitre 5 que le choix d'une architecture de Switch par rapport à une autre se fait en fonction des valeurs de N et P. Ce choix, avec le choix de la valeur de N, participe à la définition du compromis surface ajoutée/temps de test. Un générateur de CAS, prenant comme paramètre d'entrée les valeurs N, P et le type de Switch choisi, a été développé et sera présenté dans ce chapitre.

3.2.3 Le contrôle de CAS-BUS

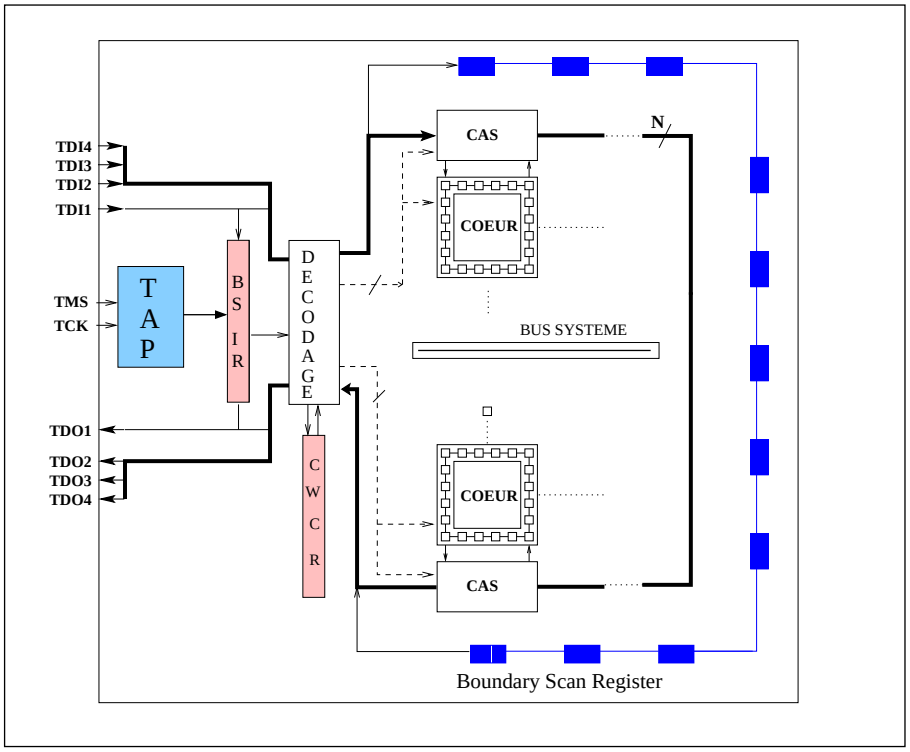


Figure 3.5: Contrôle de l'architecture CAS-BUS

Pour contrôler l'architecture CAS-BUS, nous avons eu le choix entre deux solutions. Soit nous définissions un mécanisme dédié à l'architecture CAS-BUS, soit nous utilisions un mécanisme de contrôle standard déja existant. Nous avons décidé d'utiliser les mécanismes de contrôle du standard 1149.1 et cela pour plusieurs raisons :

– Le SoC ayant de grandes chances d'être mis sur carte, l'intégration des éléments boundary scan est de toute façon prévue. Ces éléments possèdent alors une double

fonction : permettre le test boundary scan du SoC et permettre le controle de CAS-BUS. L'intégrateur système évite ainsi d'ajouter du matériel spécifique.
- Ce standard est bien connu dans le monde du VLSI et de nombreux outils de test utilisent les protocoles de ce standard. En pilotant CAS-BUS de cette façon on favorise le "test reuse".
- Les mécanismes de contrôle du standard IEEE 1149.1 sont simples et nous permettent de contrôler à la fois le boundary scan du SoC, les wrappers et les différents CAS.

En ce qui concerne CAS-BUS, le contrôleur de test doit permettre l'exécution de trois étapes : la configuration des différents wrappers, la configuration de tous les CAS et l'application et l'analyse des vecteurs de test. Pour implémenter ces fonctions nous avons dû étendre l'architecture du boundary scan (figure 3.5) :

- Nous utilisons N couples TDI/TDO au lieu d'un seul.
- Nous ajoutons un registre de données, défini par la norme comme registre optionnel. Nous appelons ce registre le registre CWCR (CAS Wrapper Coupling Register).
- Trois instructions sont ajoutées, elles aussi définies par la norme comme instructions utilisateur.

Le registre CWCR

C'est un registre à deux étages qui permet le décalage et l'application des données. Chaque bit du registre correspond à la valeur du signal c_f de chaque CAS. La longueur du registre est donc égale au nombre de CAS présents dans le SoC. La valeur chargée dans ce registre CWCR permet de déterminer quels sont les couples CIR/WIR qui vont être reliés lors de l'étape de configuration.

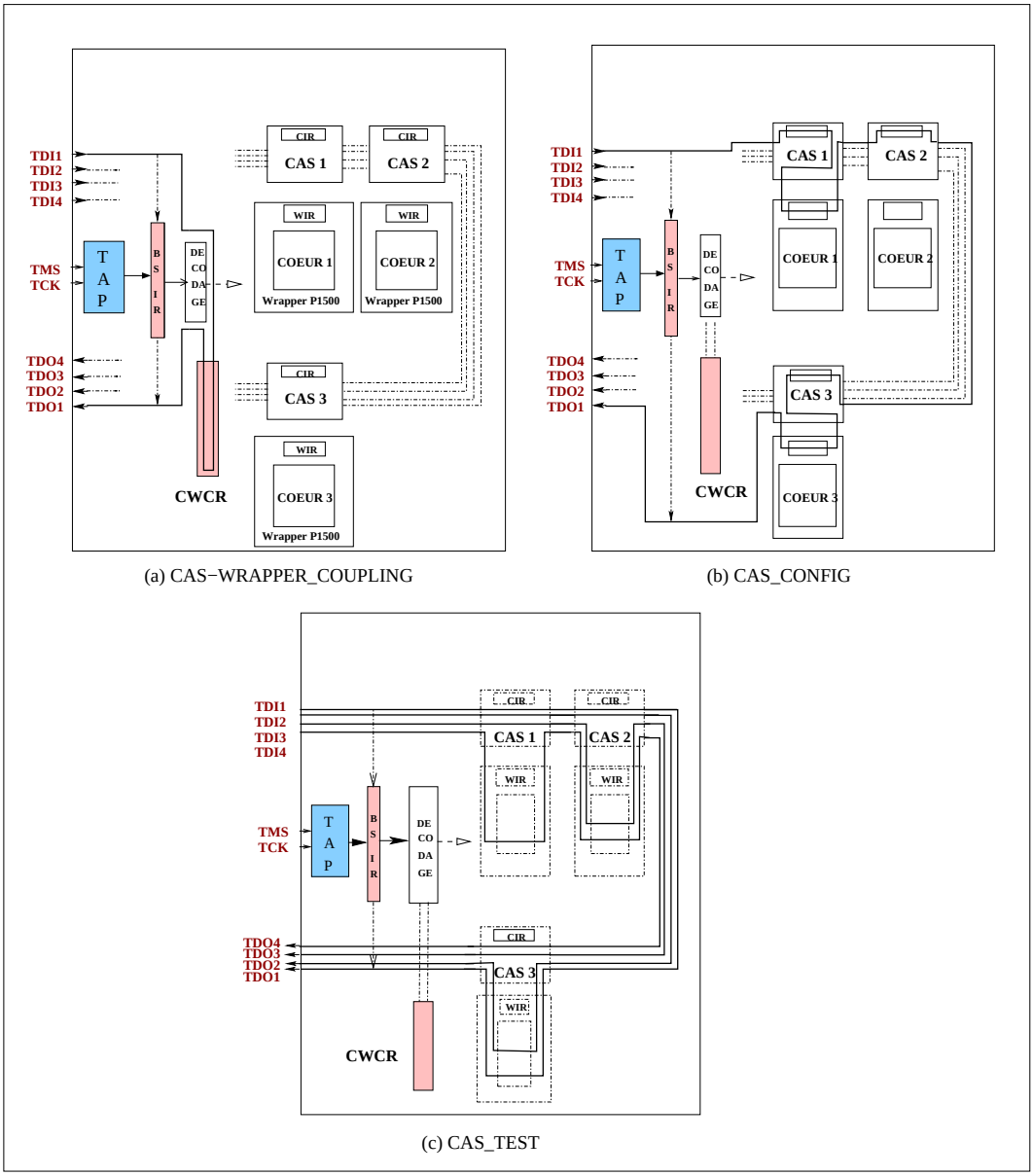
Les trois nouvelles instructions (figure 3.6)

L'instruction CAS_Wrapper_Coupling (figure 3.6 a)

Elle permet de charger le registre CWCR. TDI1 et TDO1 sont connectés à ce registre pour le charger avec les valeurs des différents signaux c_f. Cette instruction permet de fixer le schéma de couplage des différentes paires CIR/WIR.

L'instruction CAS_Config (figure 3.6 b)

Cette instruction permet de configurer les CAS et les wrappers sélectionnés par l'instruction CAS_Wrapper_Coupling. Le couple TDI1/TDO1 est connecté respectivement à l'entrée e_0 du premier CAS et à la sortie s_0 du dernier. La logique de décodage du registre d'instruction du boundary scan génère un signal *config* qui permet de sélectionner



(a) CAS−WRAPPER_COUPLING

(b) CAS_CONFIG

(c) CAS_TEST

Figure 3.6: Les trois nouvelles instructions

le CIR comme registre actif dans chacun des CAS. Les CIR et les WIR sont chaînés et leur chargement se fait en série par décalages successifs. Le signal *update* met à jour le schéma de routage de chacun des CAS et positionne les wrappers sélectionnés dans le mode de test choisi.

L'instruction CAS_Test (figure 3.6 c)

Lorsque cette instruction est chargée, les entrées/sorties TDI/TDO sont connectées aux premier et dernier CAS. Le schéma de routage vers les différents IPs étant établi, on peut

alors charger les vecteurs de test dans les différents registres des coeurs, appliquer ces vecteurs et récupérer les vecteurs réponses par décalages successifs.

Les CAS et les wrappers du SoC sont pilotés par les signaux de controle issus du controleur TAP central, les signaux générés par la logique de décodage associée au registre d'instruction et les signaux c_f issus du registre CWCR.

3.2.4 Commentaires sur l'architecture

L'architecture CAS-BUS est une architecture utilisant un bus dédié au test. C'est une architecture scalable, qui par le choix de N et du type de Switch, permet de faire des compromis surface ajoutée/temps de test. Par sa flexibilité, elle permet de s'adapter à différents types de SoC. Elle est de plus dynamiquement reconfigurable. Ces différents aspects offrent à l'intégrateur système et à l'ingénieur de test une grande liberté d'action dans le choix de la conception et du test du SoC :

- Cette architecture permet l'intégration de coeurs munis de différents types de DFT.
- Elle permet de fournir un accès pour le test en parallèle de plusieurs coeurs. Le degré de parallélisme dépend bien évidemment des valeurs de N et P. Si N est limité, les coeurs peuvent être chaînés, complètement ou partiellement. Ainsi plusieurs coeurs peuvent être activés simultanément pour le test.
- Grace à sa reconfigurabilité, elle permet d'optimiser les performances du test. Dans le déroulement d'un programme de test, le schéma de routage des données à appliquer peut varier d'une session à l'autre. Une bonne collaboration entre l'intégrateur système et l'ingénieur chargé du développement du programme de test est nécessaire pour la définition du compromis surface ajoutée/temps de test. Pour des coeurs équipés de chaînes de scan,pour une session de test donnée, le programmeur de test peut prévoir par exemple de définir un routage qui réduise le temps de test. Cela peut se faire en connectant correctement les différentes chaînes de scan du SoC, pour avoir sur chacune des lignes du bus de test une longueur totale à peu près identique. Le test des interconnexions peut aussi être optimisé et réduire ainsi le temps de test global.
- Cette architecture permet, a priori, de faire du test de maintenance. A travers une configuration correcte, certains coeurs peuvent être testés pendant que d'autres sont en mode de fonctionnement normal.

Un autre avantage de l'architecture que nous proposons est qu'elle est indépendante du wrapper utilisé. Nous l'avons développée pour qu'elle soit compatible avec le futur stan-

dard IEEE P1500, mais elle peut très facilement s'adapter à n'importe quel autre wrapper. Contrairement à CAS-BUS, les approches "Test Bus" [VB98] et "Test Rail" [M$^+$98] sont étroitement liées au wrapper défini. De plus, l'approche Test Rail de Philips offre peu de choix en ce qui concerne le schéma de routage des données.

L'architecture "Ports de Test Adressables" [Whe99] définie par Lee Whetsel (Texas Instruments) possède quelques similitudes avec notre approche CAS-BUS. Elle est modulaire, n'est pas liée aux wrappers et utilise des bus de test. Cependant elle ne permet pas d'effectuer un test simultané de coeurs différents. C'est une architecture qui utilise un bus de test pour les données en entrées et un bus pour les données en sorties. Dans notre approche le même bus joue les deux rôles. Notre module CAS est chargé de faire le routage des données contrairement aux modules de l'architecture de Whetsel qui n'effectuent pas ce routage. Ils intègrent, en revanche, d'autres fonctionnalités que les modules CAS ne possèdent pas. Les modules de Whetsel contiennent entre autre des générateurs de stimuli et des comparateurs de réponses, des automates...Cependant ces fonctionnalités supplémentaires possèdent un coût en terme de surface additionnelle et ne sont pas toujours nécessaires.

Grâce à cette modularité, CAS-BUS va dans le sens du "design-reuse". Cette architecture permet une intégration de type "Plug-and-Play" des différents modules.

Pour ce qui est du contrôle, le choix que nous avons fait va aussi dans le sens du "design-reuse", mais ce qui est encore plus intéressant c'est qu'en utilisant le mécanisme simple d'un standard bien connu on favorise aussi le "test-reuse". Ce mécanisme appliqué à un bus de test de largeur supérieure à 1 permet de tester plusieurs coeurs en même temps. Les entrées/sorties TDI_i/TDO_i (en particulier TDI_1/TDO_1) servent non seulement à transporter les données de test vers les coeurs mais aussi à positionner les CAS et les wrappers dans le bon mode. Une architecture utilisant l'approche "Test Rail" de Philips intègre aussi le boundary scan au niveau du SoC [vBvH99]. Cependant, dans cette architecture, le couple TDI/TDO a pour seule fonction de charger les différents TCBs du système pour mettre les "Test Shel"l et le système dans les bons modes de fonctionnement. Quant aux données, elle sont appliquées aux différents coeurs en utilisant directement le (ou les) "TestRail(s)" .

Jusqu'ici nous avons pour notre architecture considéré qu'il n'y avait qu'un seul bus de test qui reliait les différents CAS. Rien n'interdit qu'il y en ait plusieurs à condition que les CAS de chaque bus soient pilotés par le même contrôleur central. Le choix du nombre de bus de test peut entrer lui aussi dans la définition du compromis surface ajoutée/temps de test.

Nous disions au début de ce chapitre que l'architecture CAS-BUS permettait de faire du test hiérarchique, c'est-à-dire le test d'un IP contenant lui même une architecture CAS-BUS. En fait cela dépend du mécanisme de contrôle choisi. Dans le cas où les signaux de contrôle sont présents sur les entrées primaires de l'IP à tester, ce test est possible. Par contre, si on utilise, au sein de l'IP, un contrôleur boundary scan pour générer ces signaux, le test n'est pas possible. Dans ce dernier cas le coeur à tester est considéré comme un TAPed core. L'utilisation du standard boundary scan pour contrôler notre architecture possède de nombreux avantages mais par contre ne nous permet pas de faire du test hiérarchique.

3.3 Extension de CAS-BUS pour le test des TAPed Cores

Les coeurs au standard boundary scan sont communément appelés "TAPed Core". Ce sont des coeurs correspondants à des circuits encapsulé avec les éléments du Boundary Scan et qui ont été définis pour être intégrés sur des cartes. Dans le cadre du "design reuse" ces circuits sont réutilisés en tant que coeurs pour être intégrés dans un SoC. Ainsi ces coeurs sont déja équipé d'un wrapper (Boundary Scan) et ajouter un wrapper P1500 n'est pas économiquement intéressant. Cet ajout serait coûteux en surface additionnelle. Retirer les éléments du Boundary Scan pour les remplacer par les éléments du standard P1500 nécessiterait une nouvelle phase de conception et serait préjudiciable en terme de "Time to Market".

Pour permettre d'effectuer le test hiérarchique avec l'architecture CAS-BUS et surtout pour pouvoir tester simultanément des coeurs au standard boundary scan et au standard P1500, nous avons du définir une extension de l'architecture CAS-BUS.

3.3.1 Pourquoi cette extension ?

Nous aurions pu considérer que les TAPed cores sont testés à part, avec une autre architecture de test. Mais nous avons voulu les intégrer à l'architecture CAS-BUS pour pouvoir les tester en même temps que les autres coeurs. Ce choix permet non seulement d'éviter de rajouter du matériel supplémentaire mais permet aussi d'optimiser le temps de test en profitant des mécanismes de reconfiguration.

Avec l'architecture que nous avons présentée jusqu'ici, il n'est pas possible d'effectuer le test simultané des coeurs au standard IEEE P1500 et des coeurs au standard IEEE 1149.1. Cette impossibilité provient essentiellement du fait que les deux types de coeurs sont contrôlés par un mécanisme différent. Les wrappers sont contrôlés par des signaux

externes, générés au niveau système, alors que les coeurs boundary scan sont contrôlés à la fois par des signaux externes (TMS, TRST) et par des signaux internes, signaux issus du contrôleur TAP. La présence de cet automate TAP interne entraîne la modification d'un certain nombre d'éléments de notre architecture.

Le contrôle des TAPed core devient nécessairement un contrôle hiérarchique, ce qui n'était pas le cas pour les coeurs wrappés. Les entrées TMS des coeurs boundary scan doivent être alors pilotées par le contrôleur central du SoC. Au niveau système, le controleur TAP tel que nous l'avons défini ne permet pas de faire ce controle hiérarchique.

Pour un wrapper P1500, le choix registre d'instruction/registre de données se fait grâce au WIP (signal *SelectWIR*). Pour un coeur boundary scan, ce choix dépend de l'état actif du controleur TAP. Ainsi la sélection du registre d'instruction du TAPed core dépend non seulement de la valeur de TMS mais aussi de l'état précédent. Cette différence de mode de sélection a pour conséquence que les TAPed cores ne peut être configurés en même temps que les autres coeurs.

Pour résoudre ces problèmes nous avons défini un nouveau module qui permet le contrôle hiérarchique et un nouveau type de CAS à utiliser avec les TAPed cores. Nous appelerons ces nouveaux CAS les TAPCAS.

3.3.2 Un CAS dédié : le TAPCAS

Puisque les coeurs boundary scan ne peuvent être configurés en même temps que les coeurs P1500, nous avons défini au niveau du contrôleur central, une nouvelle instruction dédiée à la configuration des TAPed cores : l'instruction TAP_CONFIG. La configuration des wrappers se fait toujours à l'aide de l'instruction CAS_CONFIG. La figure 3.7 montre un exemple de SoC contenant les deux types de coeurs.

L'instruction TAP_CONFIG permet de connecter entre eux les différents TAPCAS. Ces TAPCAS sont aussi connectés aux entrées/sorties TDI/TDO des coeurs boundary scan. Cette instruction permet aussi de connecter TDI_1 et TDO_1 respectivement au premier et au dernier TAPCAS.

Le TAPCAS est légèrement différent du CAS. Le Switch et le registre d'instruction (CIR) restent les mêmes mais les "tristates" commandés par le signal c_f disparaissent. En revanche des multiplexeurs sont introduits en entrée et en sortie du TAPCAS, sur e_0 et s_0. Le TAPCAS étant connecté aux broches TDI/TDO du coeur, P n'est plus variable et vaut 1.

Deux types d'architectures du TAPCAS sont nécessaires. L'utilisation de l'une ou l'autre

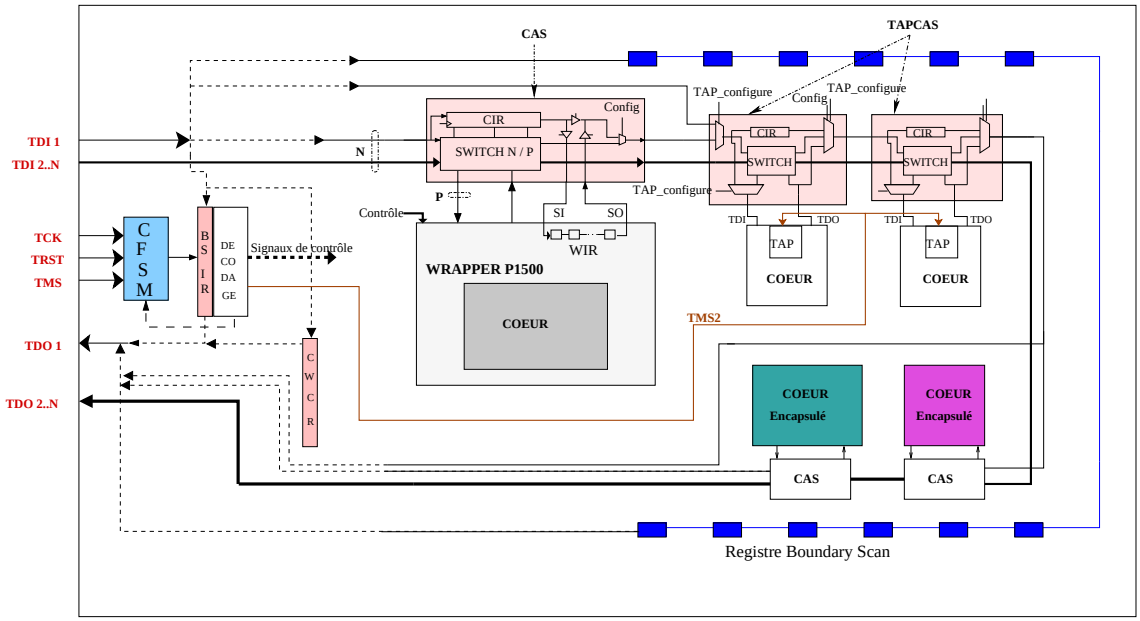


Figure 3.7: Architecture CAS-BUS étendue

des architectures dépend du placement relatif du TAPCAS dans le chaînage global. Si un TAPCAS suit un CAS alors il doit posséder un multiplexeur en entrée. S'il suit un autre TAPCAS ce multiplexeur n'est pas nécessaire.

En fonction de l'instruction boundary scan chargée au niveau SoC, les données présentes à l'entrée du TAPCAS seront aiguillées vers des registres différents. Le multiplexeur aiguillera ces données vers :

– **Le registre d'instruction du TAPCAS (CIR)**.
  Le chargement de ce registre s'effectue lorsque l'instruction CAS_CONFIG est active. Comme pour les CAS cette étape permet de sélectionner le schéma de routage du Switch.

– **Le registre d'instruction interne du coeur 1149.1**.
  Pour accéder à ce registre, l'instruction TAP_CONFIG (figure 3.8) doit être chargée. Pendant cette phase, les registres d'instruction des TAPed cores sont chaînés, le chargement des instructions se fait en série. Les données présentes en entrée du TAPCAS, sur l'entrée e_0, ne passent pas par le Switch et sont transmises directement au registre d'instruction via TDI. Les opérations de décalage, mises à jour des registres internes des TAPed cores, se font grâce au signal TMS2. Ce signal TMS2 est distribué à chacun des coeurs boundary scan. Il est généré par le controleur TAP du SoC. Sa génération est détaillée dans la section suivante. L'instruction TAP_CONFIG permet ainsi de préparer les coeurs boundary scan pour le test en les positionnant dans

le bon mode.

– **Le registre de données interne du coeur 1149.1**.
Lorsque l'instruction CAS_TEST est chargée, les TAPCAS fonctionnent comme les CAS. Une des entrées du CAS est connectée sur TDI et réciproquement, TDO est connectée à une des sorties. Le Switch joue son rôle d'aiguilleur. Les données de test sont alors décalées dans l'un des registres de données internes du coeur. On peut noter ici que l'utilisation du registre de bypass interne du coeur n'est plus nécessaire puisque cette fonction peut être réalisée par le Switch du TAPCAS. Après application, les vecteurs réponses peuvent être décalés vers le coeur suivant ou vers les sorties du SoC. Pendant cette phase de test, le signal TMS2 est là aussi actif et pilote les coeurs boundary scan.

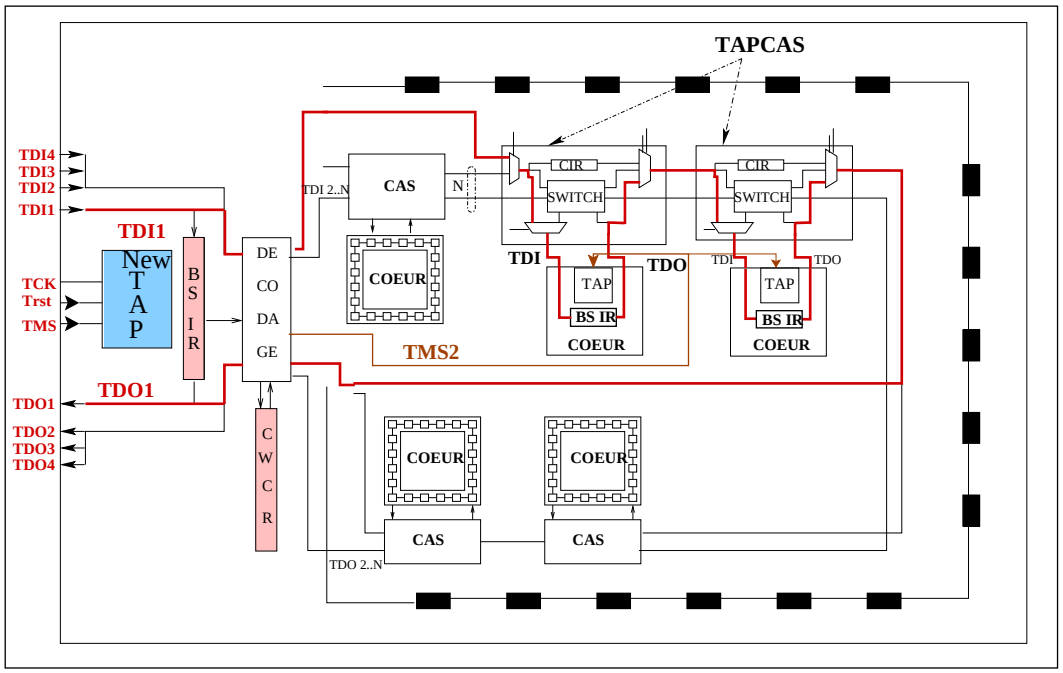


Figure 3.8: Effet de l'instruction TAP_CONFIG

L'aiguillage des données vers l'un ou l'autre de ces registres se fait grâce au signal **TAP_configure** (figure 3.7 et figure 3.8). Ce signal est activé lorsque l'instruction TAP_CONFIG est chargée dans le registre d'instruction du SoC. C'est un signal issu de la logique de décodage associée à ce registre.

3.3.3 Extension du contrôle

Pour permettre un contrôle et un test hiérarchiques, nous avons du apporter quelques modifications au contrôleur central. La machine à états finis du controleur TAP central doit pouvoir contrôler les éléments boundary scan du SoC mais aussi les éléments boundary scan des TAPed cores. Le contrôle de ces derniers passe par un contrôle des machines à états finis internes de ces coeurs. Nous apellerons CFSM (Central FSM) le controleur de test au niveau SoC et IFSM (Internal FSM) le controleur de test interne d'un coeur boundary scan.

Pour le CFSM, aux 16 états qui constituent un contrôleur TAP classique, nous avons ajouté un ensemble de 16 nouveaux états connectés à l'état Run Test/Idle (figure 3.9). Le CFSM est constitué en fait de deux contrôleurs TAP qui communiquent. Excepté les états *Test Logic Reset2* et *Run Test/Idle2*, les nouveaux états (deuxième partie du CFSM) ne génèrent pas de signaux. Ce ne sont que des états de transition. La sortie de la deuxième partie du CFSM vers la première se fait grâce à un petit compteur (figure 3.10).

En fonction de l'instruction chargée, le signal TMS permettra de piloter soit le CFSM et les IFSM en même temps soit le CFSM seul. Pour l'architecture CAS-BUS que nous proposons, les instructions à charger dans le registre d'instruction du SoC doivent respecter la chronologie suivante :

- 1. **CAS_WRAPPER_COUPLING** : on sélectionne les wrappers à configurer via l'architecture CAS-BUS.
- 2. **CAS_CONFIG** : on charge en mémoire le schéma de routage des CAS et des TAP-CAS et on configure les wrappers.
- 3. **TAP_CONFIG** : on positionne les coeurs boundary scan dans le bon mode de test.
- 4. **CAS_TEST**. On applique les données de test et on récupère les réponses. Les coeurs boundary scan et les coeurs wrappés sont testés simultanément à travers le même bus.

Il faut noter que lorsque les instructions **TAP_CONFIG** et **CAS_TEST** sont actives le signal TMS2 recopie le signal TMS. Pour mieux comprendre le lien existant entre le CFSM et la génération de TMS2 nous allons détailler ce qui se passe lorsque ces deux instructions sont chargées.

Effet de l'instruction TAP_CONFIG

Cette instruction est nécessaire pour charger les registres d'instruction internes des TAPed cores. Cette phase de test justifie l'adjonction des 16 états supplémentaires à la première partie du CFSM (figure 3.9). Cette deuxième partie du CFSM sert à contrôler les

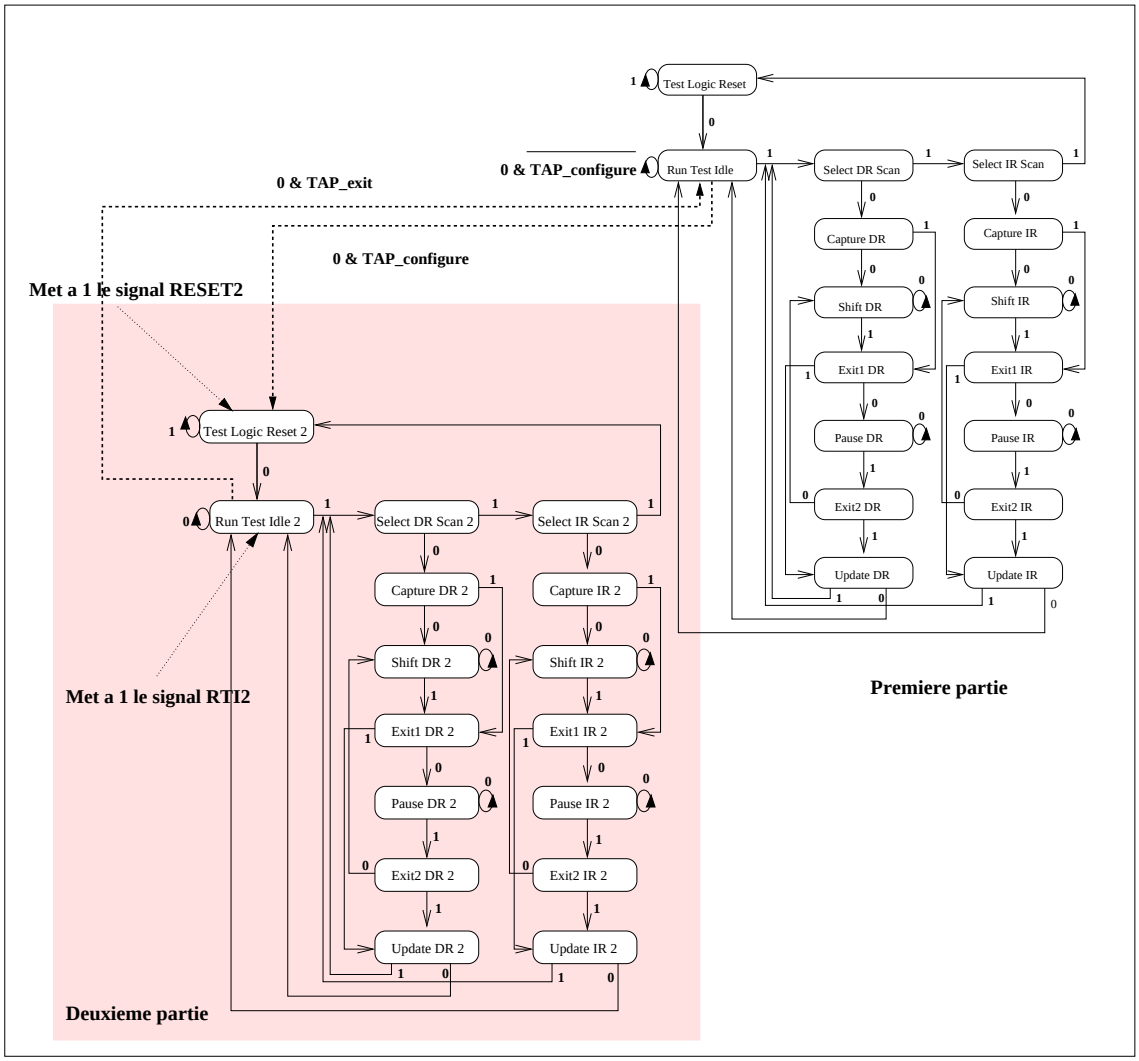


Figure 3.9: Le controleur TAP Central (CFSM)

registres des IFSM sans pour autant affecter les autres registres.

Lorsque l'instruction TAP_CONFIG est chargée puis mise à jour, le signal de contrôle TAP_configure est mis à 1. L'activation de ce signal permet de passer de la première à la deuxième partie du CFSM, de l'état *Run Test/Idle* à l'état *Test Logic Reset2*.

Cet état permet d'initialiser le compteur en mettant à 1 le signal *reset2*. Le compteur se trouve alors dans l'état INIT (figure 3.10 (a)). Dans cet état INIT, le signal TAP_exit est mis à 0. TMS2 recopie alors la valeur de TMS. Le CFSM et les IFSM peuvent évoluer alors de manière synchrone, les états étant équivalents. Plusieurs cycles doivent être passés dans l' état *Test Logic Reset2* pour synchroniser les états des IFSM et du CFSM. Ce nombre de cycles doit être supérieur ou égal à 5 pour permettre un reset des coeurs boundary scan.

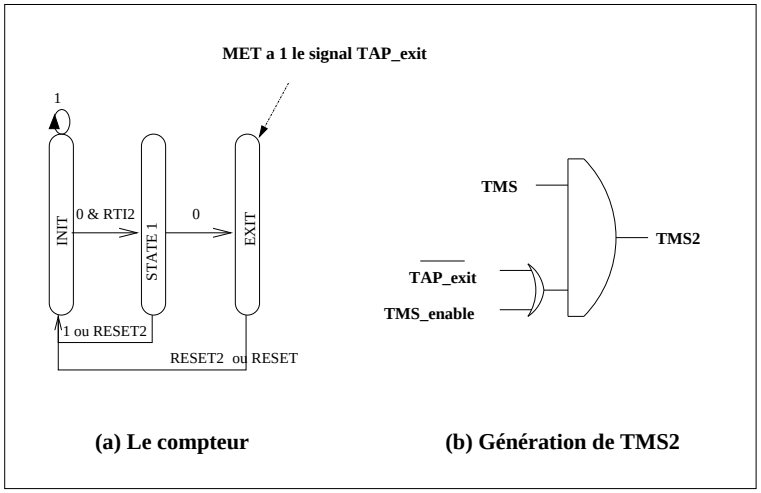


Figure 3.10: Module de comptage du CFSM et génération de TMS2

Cela est nécessaire surtout si, comme la norme boundary scan le permet, les TAPed cores ne sont pas équipés de la broche TRST. Après ces cinq cycles, les différents IFSM sont dans l'état *Test Logic Reset* et le CFSM est dans l'état *Test Logic Reset2*.

Quand on passe de l'état *Test Logic Reset2* à l'état *Run Test Idle2*, le signal RTI2 est mis à 1 et reset2 à 0. Si TMS est maintenu pendant deux cycles, le compteur passe dans l'état *EXIT*. Cet état du compteur met le signal TAP_exit à 1. On sort alors de la deuxième partie du CFSM vers la première partie. TMS2 ne recopie plus TMS. TMS2 vaut alors 0 et l'état courant des IFSM est l'état *Run Test Idle*. Les IFSM restent alors dans cet état jusqu'à la prochaine activation de TMS2. Nous voyons bien ici que le compteur par deux permet de passer d'une partie de l'automate à l'autre mais aussi de gérer l'activation de TMS2. Ce mécanisme permet de gérer le contrôle des IFSM et de sortir de la deuxième partie du CFSM.

Par contre si TMS n'est pas maintenu à 0 pendant deux cycles, l'état courant du compteur oscille entre *INIT* et *STATE1*, et TMS2 continue à recopier TMS. L'état courant du CFSM est un des 16 états de la deuxième partie. L'état courant des IFSM est celui correspondant au CFSM. Les états des différents contrôleurs étant parcourus de manière identique, si le CFSM se trouve dant l'état *Update IR2*, les IFSM se trouverons dans l'état Update IR. Grâce à cette correspondance des états nous pouvons effectuer le chargement des registres d'instruction des coeurs boundary scan.

Effet de l'instruction CAS_TEST

Cette instruction est utilisée pour charger les vecteurs de test, les appliquer et récupérer les réponses aux stimuli. A travers une configuration correcte des CAS, des TAP-

CAS, des wrappers et des TAPed cores, tous les registres de données du SoC sont reliés et connectés comme des chaînes de scan multiples. Les registres de données internes des coeurs boundary scan (registres boundary, registres bypass ou chaines de scan) peuvent être alors traités comme les registres des coeurs wrappés (registres WBR, registres bypass ou chaines de scan).

Lorsque l'instruction CAS_TEST est mise à jour, le signal TMS_enable est mis à 1 et le signal TAP_configure est mis à 0. TMS2 recopie alors TMS. La mise à 0 du signal TAP_configure permet d'une part d'aiguiller, dans les TAPCAS, les données à travers les Switch, et d'autre part de désactiver la deuxième partie du CFSM. A partir de cet instant TMS pilote simultanément les coeurs wrappés et les coeurs boundary scan.

Au moment de la mise à jour de l'instruction CAS_TEST, quand on quitte l'état *Update IR* du CFSM, le prochain état doit absolument être l'état *Run Test Idle*. Cet état permet la synchronisation entre le CFSM et les IFSM. L'instruction précédente (TAP_CONFIG) a laissé les IFSM dans l'état *Run Test Idle*. Le CFSM peut alors évoluer en parallèle avec les IFSM, les états courants étant en correspondance.

Les IFSM restent pilotés par TMS jusqu'à ce qu'une nouvelle instruction au niveau SoC soit chargée. La mise à jour de cette instruction désactive le signal TMS_enable, ce qui revient à forcer à 0 le signal TMS2. Les IFSM ne sont plus contrôlés par le CFSM et se placent dans l'état *Run Test Idle*.

Cependant, il faut noter que pendant le chargement de cette nouvelle instruction, les IFSM chargent aussi une nouvelle instruction en parallèle. Ainsi des valeurs inconnues sont chargées dans les registres d'instruction des coeurs boundary scan. Bien évidemment, cela n'est pas souhaitable, mais n'est pas préjudiciable au fonctionnement global. En effet au moment où l'on charge la nouvelle instruction les vecteurs réponses contenus dans les différents registres de données ont déjà tous été récupérés. De plus, si une nouvelle session de test est prévue, les registres d'instructions des coeurs boundary scan seront alors chargés avec des valeurs connues et correctes.

3.3.4 Commentaires sur l'architecture

Le principal avantage de cette extension de l'architecture CAS-BUS est que les coeurs boundary scan n'ont pas besoin d'être testés à part. Cela permet d'éviter une augmentation de la surface due au test. Ces coeurs peuvent donc être testés en même temps que les autres IPs, et ainsi intégrer le processus d'optimisation du temps de test. Cependant cette

architecture possède des limitations. De plus certaines précautions doivent être prises lors du test des coeurs boundary scan.

Dans le cadre d'une session de test, il se peut qu'un (ou plusieurs) TAPed core ne doive pas être testé. Il ne doit donc pas faire partie du chaînage global, son TAPCAS est mis alors en bypass. Cependant, même s'il ne participe pas au test global, il faut mettre ce TAPed core dans le mode bypass. En effet, lorsque l'instruction CAS_TEST est chargée, TMS2 est actif. L'état courant de son IFSM va changer, et des valeurs peuvent être chargées dans le registre de données sélectionné. Il est donc préférable que ce registre soit le registre de bypass.

Avec cette nouvelle architecture on peut aussi tester les coeurs boundary scan comme s'ils étaient des circuits boundary scan placés sur une carte. Nous avons vu que l'instruction TAP_CONFIG nous permettait de positionner les coeurs dans le bon mode de test, mais rien n'interdit de l'utiliser aussi pour appliquer les vecteurs de test. En effet, pendant cette phase de test, les broches TDI1 et TDO1 sont connectées directement au premier et au dernier coeur. Les registres des coeurs sont à cet instant reliés en série comme ils l'auraient été sur une carte. De plus le signal TMS externe contrôle directement ces registres.

Cette option devient une nécessité lorsqu'il s'agit de lancer l'autotest d'un coeur 1149.1. En effet l'exécution d'un test BIST des coeurs boundary scan ne peut se faire qu'en utilisant l'instruction TAP_CONFIG. En utilisant l'instruction CAS_TEST, charger les différentes chaînes du SoC et exécuter une instruction RUN_BIST dans les coeurs boundary scan n'est pas possible. Cela vient du fait que la norme boundary scan prévoit que l'exécution du BIST doive se faire dans l'état Run Test Idle pendant n cycles. Or pour charger une chaine de scan le CFSM ne peut rester dans cet état.

Pour effectuer le test BIST d'un coeur boundary scan les opérations à effectuer sont les suivantes :

- Charger l'instruction CAS_CONFIG dans le CFSM pour permettre la configuration des CAS et TAPCAS.

- Charger l'instruction TAP_CONFIG dans le CFSM.

- Charger l'instruction RUN_BIST dans le ou les coeurs boundary scan concernés.

- En sortant de la deuxième partie du CFSM, rester dans l'état Run Test Idle du CFSM le temps nécessaire à l'exécution du BIST.

- Charger l'instruction TAP_CONFIG.

- Charger dans le coeur boundary scan, l'instruction positionnant son TDI et son TDO

sur l'analyseur de signature.

- Sortir la signature.

Si on compare notre architecture avec les architectures HTAP et TLA décrites dans le chapitre précédent, on s'aperçoit qu'elle permettent toutes les trois de contrôler les TAPed cores de manière hiérarchique. Cependant les architectures HTAP et TLA permettent de contrôler et de tester les coeurs un par un, un seul d'entre eux étant actif à la fois. Dans l'approche que nous proposons les coeurs boundary scan peuvent être testés en même temps, en série ou en parallèle, selon la configuration choisie. L'architecture CAS-BUS pour les TAPed core apporte donc plus de flexibilité et permet d'optimiser le temps de test pour une surface additionnelle réduite (voir chapitre 5). De plus ces coeurs peuvent être testés avec des coeurs munis de wrappers ce qui n'est pas le cas des architectures précitées.

3.4 Conclusion

Nous avons présenté dans ce chapitre l'architecture CAS-BUS. C'est un TAM scalable, flexible, reconfigurable dynamiquement et complètement modulaire. Cette architecture permet à l'intégrateur système de faire des compromis surface additionnelle/temps de test en choisissant : le nombre de bus, la largeur de chaque bus (N), le type de Switch des modules CAS et le nombre de coeurs à tester en même temps. Par sa modularité elle permet une intégration de type "Plug-and-Play" favorisant le "design-reuse".

Le contrôle de cette architecture est simple, basé sur l'utilisation des éléments du standard IEEE 1149.1. Telle qu'elle est définie, l'architecture CAS-BUS est compatible avec le standard boundary scan au niveau SoC et compatible avec l'état actuel du standard P1500 au niveau coeur. Cette compatibilité à deux niveaux favorise le "test-reuse".

Nous avons présenté aussi dans ce chapitre une extension de cette architecture qui résout dans une large mesure le problème du test des systèmes intégrant à la fois des coeurs munis de wrappers et des coeurs boundary scan. Cette nouvelle architecture permet de tester ces deux types de coeurs simultanément. De plus cette extension n'affecte pas les fonctionnalités que procure l'architecture CAS-BUS de base.

Chapitre 4

Inadéquation largeur de TAM/nombre de broches de test

4.1 Introduction

Parmi les contraintes qui influent sur le choix du TAM et de sa largeur nous avons cité le temps de test et la surface additionnelle. Cependant il y a une autre contrainte qui peut avoir son importance : le nombre de broches dédiées au test disponibles au niveau du SoC.

Nous avons vu dans le chapitre 2 que des techniques permettaient d'optimiser la largeur du TAM en fonction du temps de test. Cependant cette largeur peut être limitée par le nombre de broches du circuit.

L'efficacité des techniques présentées dans ce chapitre est liée à la corrélation des données de test à compresser. Ces techniques sont basées sur le codage statistique ou différentiel et tirent parti de la distribution probabiliste des données. De ce fait elles s'avèrent peu efficaces lorsque les données sont très faiblement corrélées et lorsque les échantillons de données ont une probabilité d'occurence voisine.

Pour surmonter ces inconvénients, nous avons développé une technique de compression spécifique à ces données faiblement corrélées. Cette technique peut s'adapter à différents TAM mais est d'abord dédiée à l'architecture CAS-BUS qui est basée sur l'utilisation d'un bus de test. Dans ce chapitre nous présenterons d'abord cette technique puis nous verrons comment nous l'appliquons à l'architecture CAS-BUS. Avant de conclure, les limitations de la nouvelle architecture CAS-BUS seront discutées.

4.2 Théorie

La méthode que nous proposons est une technique de compression sans pertes. Nous considérons que les données à compresser sont composées d'échantillons de longueur n bits. Le choix de la valeur de n sera décrit plus tard.

4.2.1 Expansion sans compression

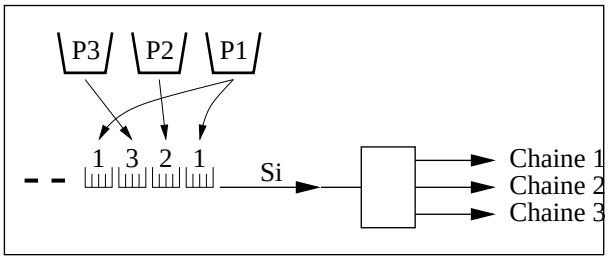


Figure 4.1: Exemple d'expansion

Dans un premier temps, pour faciliter la description qui suit, nous considèrerons que n vaut 4. La figure 4.1 décrit l'expansion d'une entrée Si vers trois chaînes de scan. Les échantillons arrivent en série sur *Si*. P1, P2 et P3 représentent respectivement la longueur des chaînes de scan 1, 2 et 3. Nous considérons que les échantillons présents sur *Si* sont transférés de manière cyclique sur les chaînes correspondantes. Sans compression, le temps nécessaire au transfert de tous les vecteurs de test est égal à (P1+P2+P3) cycles. S'il n'y avait pas d'expansion à effectuer, avec trois entrées, le temps de test total serait réduit à max(P1,P2,P3) cycles.

4.2.2 Compression

Pour compenser l'augmentation du temps de test dû à l'expansion, une méthode de compression/décompression doit être appliquée.

Pour un échantillon de quatre bits, le nombre de combinaisons possibles des valeurs de l'échantillonest égal à 16. Nous classons arbitrairement ces 16 combinaisons dans quatre groupes contenant chacun quatre combinaisons. La figure 4.2 (a) montre un partitionnement possible des 16 combinaisons.

Dans un groupe, chaque combinaison est codée sur deux bits (les deux bits d'offset de la combinaison). Les numéros de groupe sont aussi codés sur deux bits. Nous considérons une autre façon de sélectionner les groupes. Nous chaînons les quatres groupes entre eux,

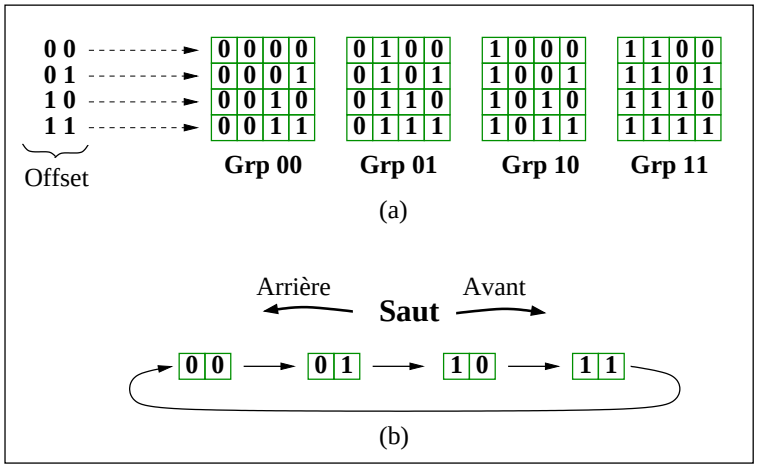


Figure 4.2: Traitement des échantillons

comme le montre la figure 4.2 (b). A partir d'un groupe donné, pour aller vers un autre groupe, trois types de déplacements sont possibles :

- Un saut en avant
- Un saut en arrière
- Deux saut en avant (ou, de façon équivalente, deux sauts en arrière

L'idée est de commencer par un groupe de départ (par exemple le groupe 00) et de coder les échantillons sur deux bits (les deux bits d'offset). Si l'échantillon suivant est dans un groupe différent, il faut signaler le sens et la longueur du saut à effectuer. Nous avons décidé d'utiliser un signal supplémentaire, que nous avons appelé **Grp_chg**, qui précise le nombre de sauts nécessaires pour passer au groupe suivant. Lorsque Grp_chg est actif, le sens du saut est donné par la valeur présente sur **Si**. Par exemple pour passer du groupe 00 au groupe 10, Grp_chg doit être à 1 pendant deux cycles et pendant ces deux cycles le signal *Si* doit être à 1 (deux sauts en avant). Ce signal *Si* peut aussi être à 0 (sauts en arrière) pendant ces deux cycles, on retombe alors sur le même groupe puisque les groupes sont chaînés. Sur *Si* on retrouvera un échantillon codé sur deux, trois ou quatre bits.

Pour chaque échantillon de quatre bits, le gain en nombre de cycles par rapport à une simple expansion, peut être calculé de la façon suivante :

- Pas de saut (pas de changement de groupe) : seul les bits d'offset sont utilisés. L'échantillon est codé sur deux bits (deux cycles). On alors 50% de gain.

- Un saut : l'échantillon est codé sur 3 bits (deux bits d'offset + un bit précisant la direction du saut). Le gain est alors de 25%.

- Deux sauts : l'échantillon est codé sur 4 bits, deux bits d'offset et deux bits qui in-

diquent le sens du saut. Pour ce cas le gain est alors nul.

Quand les données à compresser ne sont pas corrélées, pour l'ensemble des échantillons, statistiquement, on a 25% de chance de faire un saut en avant, 25% de chance de faire un saut en arrière, 25% de chance de faire deux sauts et 25% de chance de ne faire aucun saut. Cette équiprobabilité vient du fait qu'un échantillon a une chance sur quatre de se trouver dans un des quatres groupes. Ainsi le gain moyen avec ce type de codage est égal à :

gain_moyen= 25% x 50% (0 saut)+ 25% x 25% (1 saut avant) + 25% x 25% (1 saut arrière)+ 25% x 0% (2 sauts)

gain_moyen= 25%

Le gain maximal est atteint lorsqu'il n'y a pas de sauts à effectuer. Il est alors égal à 50%

4.3 Application à l'architecture CAS-BUS

Comme nous l'avons dit auparavant cette méthode de compression/décompression/expansion est dédiée à l'architecture CAS-BUS. Elle s'adresse au test des coeurs wrappés et contrôlés par les éléments du boundary scan. Cependant elle peut être adaptée et s'intégrer dans d'autres architectures.

4.3.1 Présentation de l'architecture

Nous avons implémenté cette méthode en considérant que les échantillons à compresser sont de longueur 4. Cette valeur ne correspond pas à la taille de l'échantillon qui fournit un gain en compression optimal. Ce gain est optimal pour un échantillon de longueur 3. Les techniques d'optimisation de la méthode sont présentées dans la section suivante.

Nous avons aussi implémentée la méthode pour des échantillons de longueur 3 (optimum) mais pour rester dans la continuité de ce qui est présenté auparavant, nous décrivons ce qui suit pour un échantillon de longueur 4.

La figure 4.3 présente l'architecture CAS-BUS intégrant les éléments nécessaires à l'application de cette méthode. L'entrée série **Si** décrite auparavant correspond à l'entrée **TDI**. Pour éviter d'ajouter un signal supplémentaire, nous avons décidé d'affecter le signal **Grp_chg** à **TMS** pour indiquer le saut de groupe. Un module de décompression/expansion a été ajouté à l'entrée du TAM, de même qu'un MISR à sa sortie. Ce MISR sert à compresser les sorties du TAM en une signature qui sera décalée vers TDO.

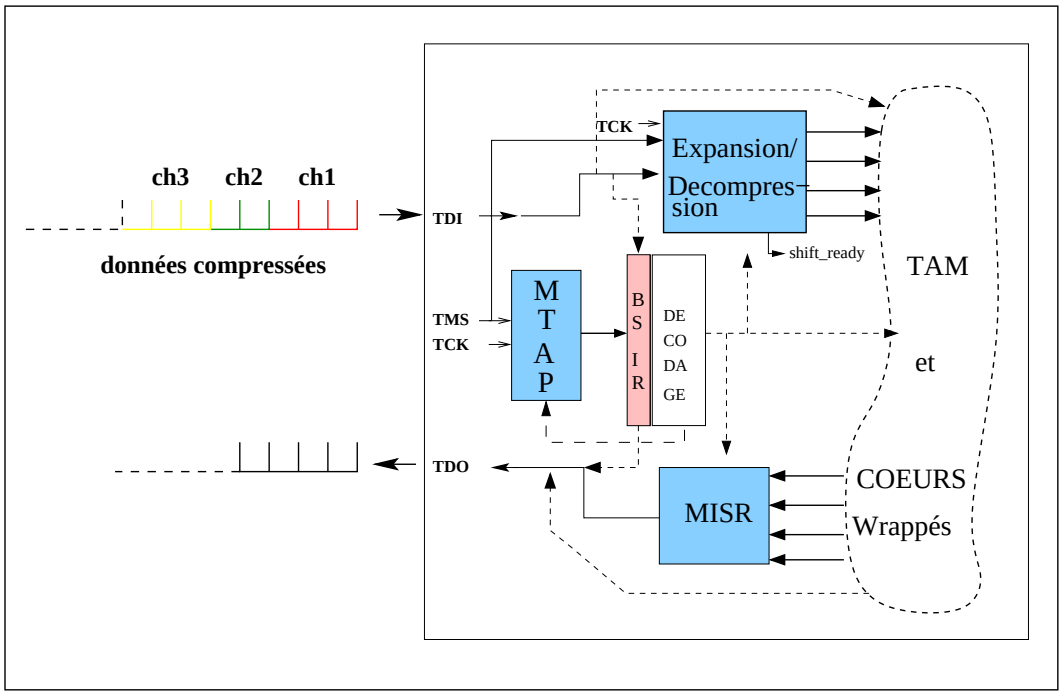


Figure 4.3: Architecture CAS-BUS avec décompression/expansion

4.3.2 Contrôle de l'architecture

En plus des trois instructions décrites dans le chapitre précédent (CAS_WRAPPER_COUPLING, CAS_CONFIG et CAS_TEST), nous avons ajouté une quatrième instruction que l'on a nommée **DECOMPRESS_test**. On utilise cette instruction à la place de CAS_TEST lorsqu'on veut transférer puis appliquer les vecteurs de test. Cette nouvelle instruction permet de connecter TDI au module d'expansion et TDO à la sortie du MISR.

Lorsque l'instruction DECOMPRESS_test est chargée, les décalages des données de test se font dans l'état shift_DR. Pour décompresser un échantillon (4 bits) il faut effectuer le nombre de sauts adéquats (1 ou 2), décaler, faire le nombre de sauts pour l'échantillon suivant, décaler etc. Il faut un cycle pour faire un saut et deux pour deux sauts. En partant de l'état shift_DR, il faut donc pouvoir y revenir en 1 ou 2 cycles. Cependant le controleur TAP tel qu'il est défini par la norme IEEE 1149.1 ne permet pas de revenir dans cet état avant 3 cycles.

Pour remédier à cela, et donc pour ne pas perdre de cycles lors de la décompression et du décalage des données, nous avons dû utiliser un contrôleur TAP modifié (MTAP). La figure 4.4 décrit ce nouvel automate adapté aux besoins de la méthode.

Cet automate se comporte exactement comme l'automate défini par le standard sauf

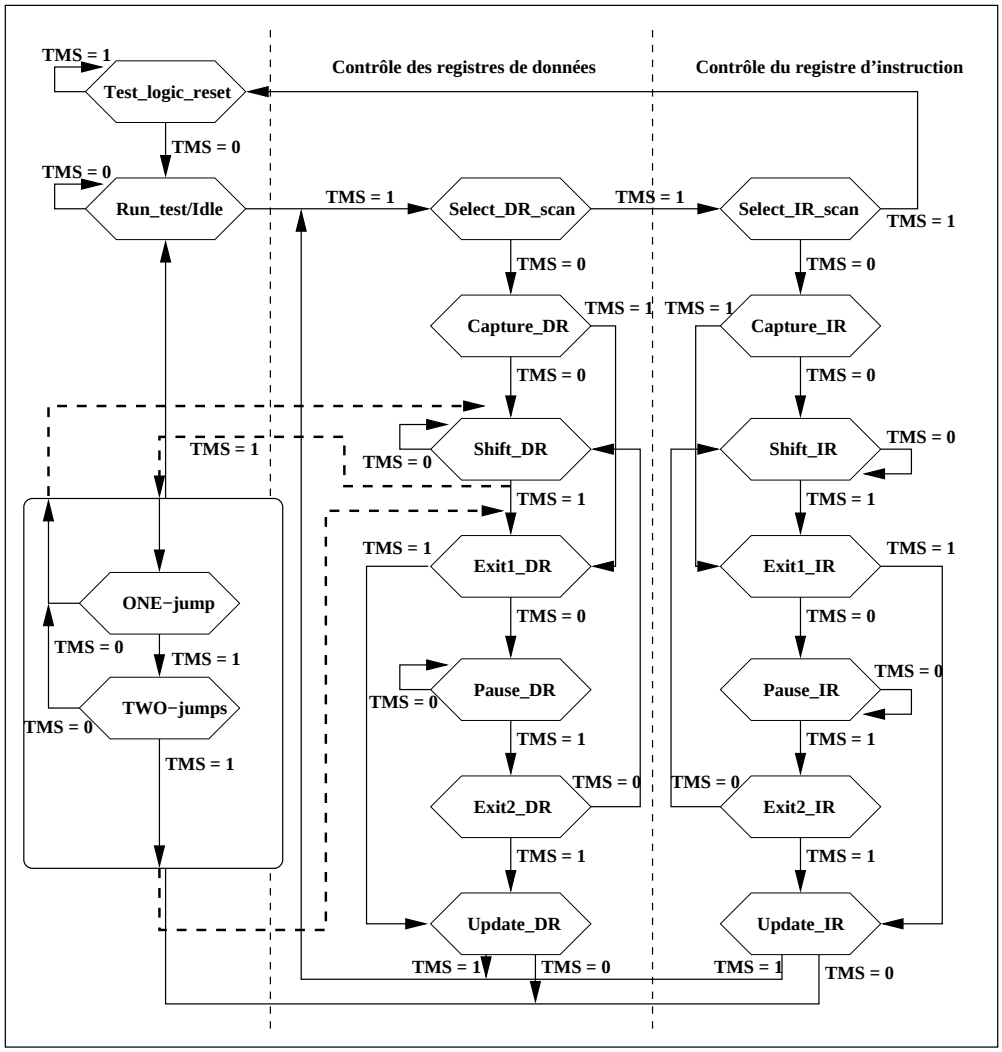


Figure 4.4: Le controleur MTAP

quand l'instruction DECOMPRESS_test est chargée. Dans ce cas deux états sont ajoutés dans la partie contrôle du registre de données. Ces deux états, **ONE_JUMP et TWO_JUMPS**, sont automatiquement "greffés" quand l'instruction DECOMPRESS_test est décodée. Avec ce contrôleur les contraintes de retour dans l'état shift_DR sont respectées. En fonction de la valeur de TMS on peut faire un ou deux sauts et revenir dans l'état shift_DR. Lorsque TMS indique un saut, le sens du saut est donné par la valeur présente sur TDI (1 pour un saut avant, 0 pour un saut arrière).

En plus des signaux traditionnels que génère un controleur TAP du Boundary Scan, le MTAP génère un signal d'activation (ACT) qui vaut 1 lorsqu'on est dans un des états suivants : ONE_JUMP, TWO_JUMPS et shift_DR. Ce signal permet d'activer le module de

décompression/expansion.

4.3.3 Le module de décompression/expansion

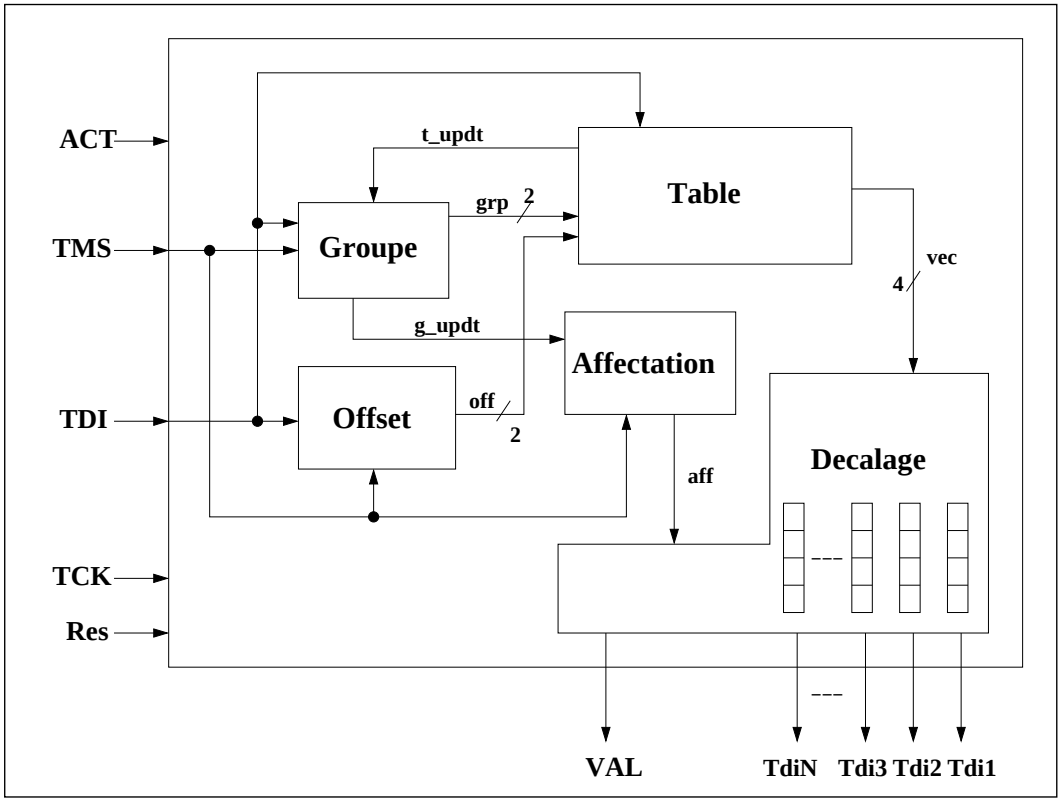


Figure 4.5: Le module de décompression/expansion

Le rôle du module, dans un premier temps, est de fournir un échantillon de 4 bits à partir des informations disponibles sur TDI et TMS. Cet échantillon est ensuite stocké dans un registre. Lorsque tous les registres de sortie sont pleins, les échantillons sont appliqués au TAM, par décalage, sur 4 cycles. Le signal VAL en sortie du module devient actif lorsque ces registres sont pleins. Les données présentes à l'entrée du TAM sont décalées tant que le signal VAL est à l'état haut.

La figure 4.5 décrit l'architecture du module de décompression/expansion. Il est composé de quatre sous-modules :

- **Groupe**. En fonction du groupe de départ, de TDI et de TMS, il détermine le groupe actif et prévoit le groupe qui va suivre.
- **Offset**. Ce module contrôlé par TMS et TDI enregistre les deux bits qui, pour un groupe donné, vont pointer sur l'échantillon désiré.

- **Table**. C'est une mémoire qui contient les 16 combinaisons d'échantillons, répartis entre quatre groupes. Cette table présente en sortie l'échantillon décodé sur 4 bits, le vecteur de coordonnées (grp,off). Avant toute session de test cette table est initialisée. Les 16 combinaisons sont chargées en série en utilisant TDI. Les valeurs d'initialisation correspondent aux 16 échantillons répartis dans quatre groupes. Comme nous l'avons vu, la répartition de ces échantillons et l'ordonnancement des groupes dépend des vecteurs de test à appliquer. Le signal t_updt précise que les 64 bits d'initialisation sont chargés et que la décompression peut commencer.
- **Affectation**. Ce module permet de générer le signal aff. Ce signal est activé quand un échantillon présent à la sortie du module Table doit être affecté à un des registres du module décalage. L'instant d'affectation dépend de TMS et de la mise à jour du groupe actif.
- **Décalage**. Il contient N registres de 4 bits. Lors de l'affectation, les 4 bits de l'échantillons sont mémorisés dans le registre actif, en un cycle. Les N registres sont chargés un par un. Lors de la dernière affectation le module active le signal VAL.

Du point de vue du fonctionnement général, les échantillons compressés sont chargés en série dans le module de décompression/expansion. Les informations présentes sur TDI sont triées (sens du saut et offset), l'échantillon est reconstruit grâce à la table puis cet échantillon est affecté à un des N registres. Ce processus se fait de manière continue, sans perte de temps, jusqu'à ce que les N registres soient pleins. Cela est rendu possible par le fait que TMS est directement connecté au module. Pendant les 4 cycles où l'on applique les contenus des registres au TAM (évacuation), le processus de décompression continue à s'exécuter. Ce processus s'arrête quand on n'est pas dans un des trois états ONE_jump, TWO_jumps et Shift_DR.

La figure 4.6 illustre ce processus pour un TAM de largeur 3 (3 chaînes). CHi-VALj représente un échantillon de 4 bits appartenant à la chaine i. L'arrivée des échantillons sur la chaine i se fait dans l'ordre j. CHi-CDj correspond au code compressé de CHi-VALj. Les quatres groupes sont représentés par les lettres A, B, C et D. Le groupe de départ est le groupe B. Lorsque 3 codes sont chargés (CH1-CD1, CH2-CD1, CH3-CD1), les valeurs décompressées correspondantes (CH1-VAL1, CH2-VAL1, CH3-VAL1) sont appliquées sur TDI1, TDI2 et TDI3. Sans aucune compression il faudrait 12 cycles pour faire l'expansion des échantillons sur 3 chaînes. Dans cet exemple la décompression et l'expansion de CHi-VAL1 prennent 9 cycles. On gagne 3 cycles, soit un gain de 25%

Pour notre architecture CAS-BUS, le signal VAL est connecté aux différents CAS et aux wrappers. Il remplace le signal shift_dr généré par le controleur TAP, activant le décalage

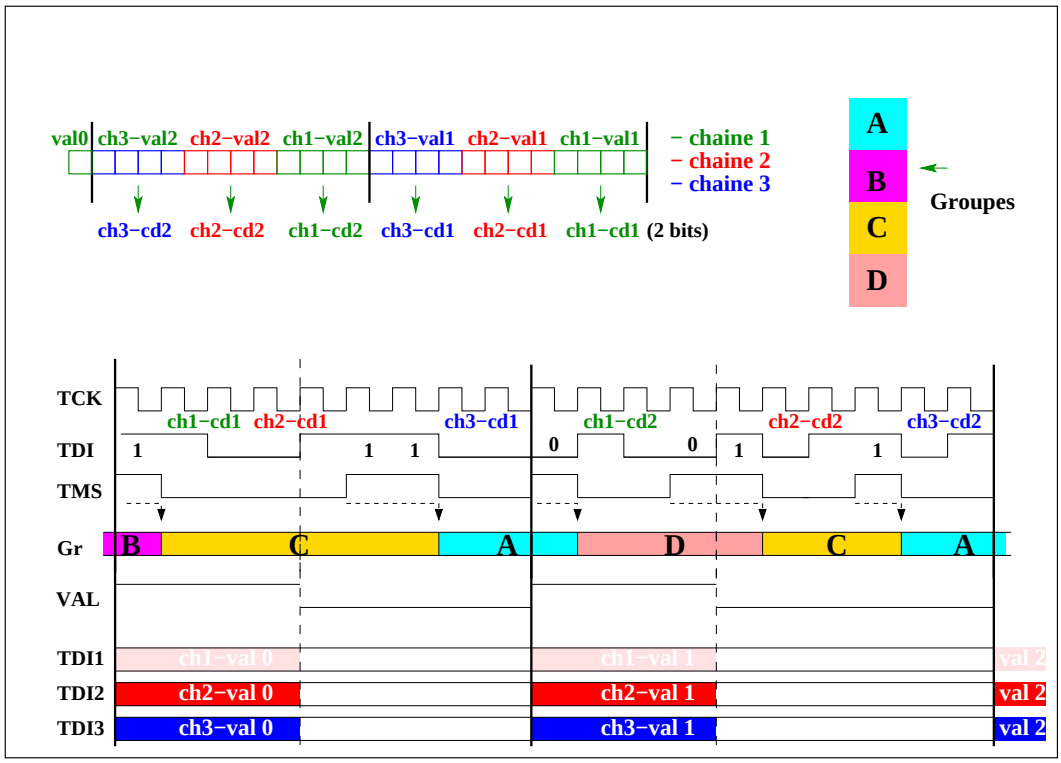


Figure 4.6: Exemple de décompression/expansion

des différents registres de test du SoC.

Le module de décompression/expansion a été décrit en VHDL comportemental générique pour N chaînes de sorties. Quelques résultats de synthèse sont présentés dans le chapitre suivant.

4.4 Optimisations de la méthode

Nous avons vu qu'avec des échantillons de quatre bits et un partitionnement de ces échantillons en quatre groupes nous avions un gain maximal égal à 50%. Ce gain dépend à la fois du nombre de groupes et de la longueur de l'échantillon. En effet si on prend un échantillon de 6 bits et un partitionnement en 4 groupes des 64 possibilités, on obtient un gain maximal de 33%. Si ces combinaisons sont réparties entre 8 groupes, le gain maximum est de 50% mais le gain moyen est égal à 16%. Une première optimisation de la méthode consiste donc à déterminer la longueur de l'échantillon (n) et le nombre de groupes à prévoir pour avoir un gain moyen maximal.

Si on considère que n=a+b, on va avoir un partitionnement composé de 2^a groupes,

chacun des groupes contenant 2^b échantillons.

Soit l la longueur des données à compresser. Le nombre d'échantillons à compresser est égal à l/(a+b). Le nombre de cycles correspondant aux données compressées est donné par :

nb_cycles= (nombre de cycles moyen par échantillon) x nombre d'échantillons

En partant d'un groupe donné on a une probabilité P de se trouver dans un autre groupe, avec P=1/(2^a). Puisque nous supposons que les données ne sont pas corrélées nous considérons que P est constante quel que soit le groupe cible.

Si le groupe cible est le même que celui du départ, le nombre de cycles de l'échantillon compressé est égal à b. Si le groupe cible est un groupe voisin direct, l'échantillon compressé sera sur (b+1) bits. Si le groupe cible nécessite deux sauts, l'échantillon compressé sera sur (b+2) bits...Enfin si le groupe cible est diamétralement opposé au groupe de départ, il faudra faire 2^a/2 sauts et l'échantillon compressé tiendra sur (b+$2^{(a-1)}$) bits. Le nombre de cycles moyen par échantillon est donné alors par :

$$nb\_cyclesmoyenparchantillon = b.P+(b+1).2P+(b+2).2P+(b+3).2P+...+(b+2^{(a-1)}).P$$

A l'exception du premier et dernier élément du second membre de cette équation, tous les termes sont multiplié par 2. Ce facteur provient du fait que deux groupes cibles peuvent être atteint en fonction du sens du saut : saut avant ou saut arrière. La probabilité de faire un ou plusieurs sauts est donc égale à 2P. La probabilité de faire 0 sauts ou $2^{(a-1)}$ sauts est égale à P puisqu'il n'y a qu'une façon d'atteindre les groupes correspondants.

Le nombre de cycles correspondant à l'ensemble des échantillons compressés est donc égal à :

$$nb\_cycles = [P.b + 2P.\sum_{i=1}^{2^{(a-1)}-1}(b+i) + P.(b+2^{(a-1)})].l/(a+b)$$

Après développement puis réduction on obtient :

$$nb\_cycles = (b+2^{(a-2)}).(l/(a+b))$$

Le gain en pourcentage est donné par gain= (l- nb_cycles)/l

En fonction de a et b le gain moyen de l'ensemble des données compressées est donné par la formule :

$$gain(\%) = 100.(1-(b+2^{(a-2)})/(a+b))$$

Nous avons tracé la fonction gain = f(a) pour plusieurs valeurs de b (figure 4.7). Nous obtenons des solutions (a,b) qui maximisent ce gain. Ces solutions sont (1.5,0) et (1.9,1). Nous écartons la première solution qui correspond à b=0. Cette solution ne nous intéresse pas puisqu'elle signifie qu'on n'envoie aucune donnée sur l'entrée Si.

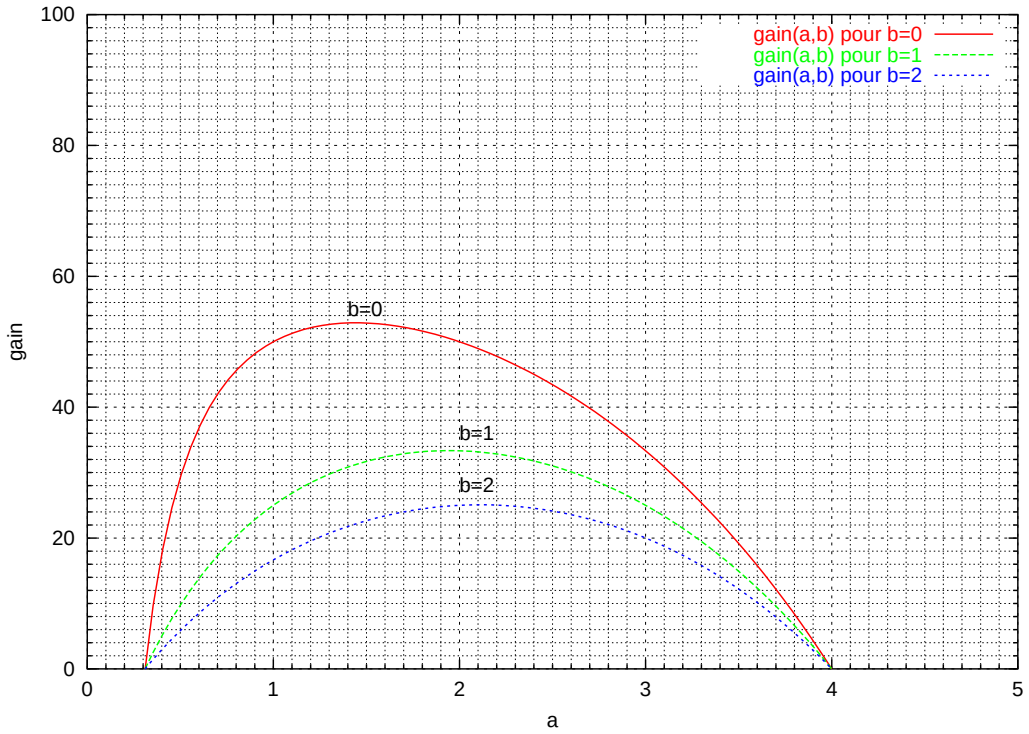


Figure 4.7: Gain moyen en fonction de a et b

Avec la deuxième solution on a a=1.9 et b=1. On obtient donc un gain moyen maximal pour a=2 et b=1, soit n=3. Le gain moyen dans ce cas est égal à 33%. Le gain maximal correspondant à aucun saut est égal à 66%.

La deuxième optimisation que nous avons apporté à la méthode consiste à définir, à partir des données à compresser, le partitionnement idéal des échantillons dans les différents groupes. A ce partitionnement, il faut ajouter un ordonnancement optimisé des groupes entre eux. Ces optimisations permettent de réduire le nombre de sauts et donc d'augmenter le gain en compression final.

Pour réaliser ces partitionnement nous avons développé la méthode de façon à ce que les bits de donnée (a) ne correspondent pas à l'offset de l'échantillon mais à la position de l'échantillon dans le groupe. Cette flexibilité nous permet de mettre dans un même groupe n'importe quel échantillon.

Cette méthode a été développée au départ pour compresser des données de test peu ou pas corrélées. Cependant si une corrélation existe, même faible, l'utilisation de ces optimisations permettre d'améliorer le taux de compression. Un outil a été développé pour compresser les données et effectuer ces optimisations. Cet outil ainsi que les partitionnements proposés seront présentés dans le chapitre suivant.

4.5 Commentaires

4.5.1 Cas particuliers

Nous avons traité jusqu'ici le cas d'une décompression/expansion d'une entrée TDI vers N sorties. Deux autres cas peuvent apparaitre.

Premier cas : N=1

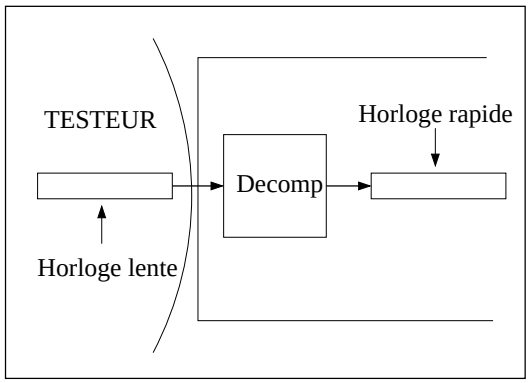


Figure 4.8: Décompression sans expansion

Dans cette configuration, le TAM est de largeur 1, et une seule entrée TDI est disponible. Il n'y a donc pas d'expansion à réaliser. La méthode peut être appliquée à condition de disposer de deux horloges de test (figure 4.8). Lorsque l'échantillon est sur 4 bits, il faut entre 2 et 4 cycles pour charger un échantillon codé et 4 cycles pour l'évacuer hors du module de décompression. Il faut donc une horloge lente pour charger les données compressées et une horloge rapide pour consommer les données décompressées.

On peut noter que lorsqu'il y a expansion (N supérieur ou égal à 2), on n'a pas besoin d'avoir deux horloges différentes. Le nombre de cycles nécessaires au chargement des N registres du module décalage (entre 2N et 4N cycles) est supérieur ou égal au nombre de cycles nécessaires à leur déchargement (4 cycles).

Deuxième cas : M entrées TDI

Du point de vue du temps de test, une décompression/expansion sur une ligne du TAM est moins efficace qu'une connexion directe de cette ligne sur TDI. Une règle simple consiste donc à minimiser le nombre de chaînes à expanser. Ainsi, lorsqu'on a M entrées TDI et N chaînes (TAM de largeur N), on connecte M-1 entrées sur M-1 chaînes, l'entrée TDI restante est alors réservée à la décompression et l'expansion vers les N-(M-1) autres chaînes du TAM. L'affectation des chaînes soit aux entrées TDI soit aux sorties du module d'expan-

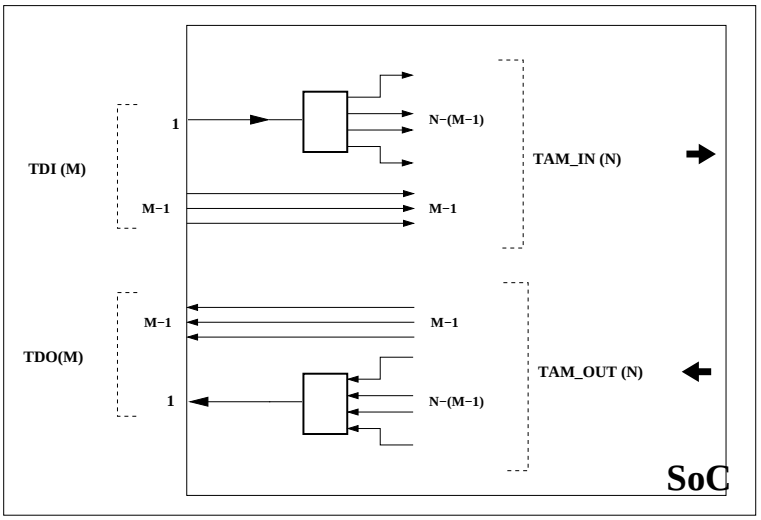


Figure 4.9: Décompression/expansion M ->N

sion se fait après une évaluation globale du temps de test. Cette affectation dépend de la longueur des chaînes et du nombre de vecteurs à appliquer.

Pour l'architecture CAS-BUS, une façon de faire serait de brancher les chaînes les plus courtes sur le module d'expansion, le décalage sur ces chaînes serait contrôlé par le signal VAL. Les autres chaînes seraient directement branchées sur les autres entrées TDI, le décalage sur ces chaînes serait piloté par le signal shift_dr issu du contrôleur TAP.

Lorsqu'il y M entrées TDI, on pourrait être tenté par la mise en place de plusieurs décompresseurs. Cela est possible à condition de prévoir en entrée, pour chaque décompresseur, un signal de donnée et un signal indiquant le saut. On pourrait ainsi intégrer M/2 décompresseurs. Cependant le gain en compression global obtenu avec ces M/2 modules ne sera pas forcément intéressant suivant l'architecture du SoC (nombre et longueur des chaines de scan...)

- Pas possible d'avoir un décompresseur avec plusieurs TDI en entrée car sur chaque TDI les sauts ne sont pas fait au même moment

4.5.2 Limites de cette architecture

Le décompresseur tel que nous l'avons présenté ne possède qu'une entrée TDI. Il ne peut pas en posséder plus car l'indication de saut ne se fait qu'avec un seul signal. Hors, pour un décompresseur à plusieurs TDI, sur chaque entrée, les sauts ne se font pas au même moment.

Nous avons vu au chapitre précédent une version de l'architecture CAS-BUS permettant de tester simultanément des coeurs munis de wrappers et des coeurs boundary scan. L'architecture que nous venons de présenter offre une solution aux problèmes liés au manque de broches de test du système intégré. On peut se demander si cette dernière architecture permet aussi de tester les coeurs boundary scan.

Nous avons vu que l'automate contrôlant l'architecture pour les coeurs boundary scan (CFSM) était constitué de 32 états : 16 états pour l'exécution de l'instruction TAP_CONFIG et 16 états pour l'instruction CAS_TEST. L'automate MTAP que nous présentons dans ce chapitre contient 18 états, nécessaires à l'exécution de l'instruction DECOMPRESS_TEST. Dans l'hypothèse où le SoC à tester contiendrait des coeurs wrappés, des coeurs boundary scan et disposerait d'une seule broche TDI pour un bus de largeur N, il faudrait implémenter la méthode de décompression/expansion que nous avons vu.

Pour cela il faudrait ajouter au controleur MTAP, les 16 états permettant l'exécution de l'instruction TAP_CONFIG. La décompression n'intervenant pas les registres internes des coeurs boundary scan peuvent être chargés.

Par contre pour effectuer le test proprement dit (chargement des stimulis, application et déchargement des réponses), la fusion des instructions CAS_TEST et DECOMPRESS_TEST n'est pas possible. Les contraintes de temps imposées par la méthode de décompression nous ont poussé à ajouter deux états supplémentaires dans le controleur MTAP, utilisés lorsque l'instruction DECOMPRESS_TEST est chargée. Dans le chapitre précédent nous avons vu que pendant le test de l'ensemble des coeurs, les contrôleurs des coeurs boundary scan suivaient fidèlement le contrôleur central. Si on veut faire de la décompression/expansion et tester à la fois les coeurs wrappés et les coeurs boundary scan, il faut que là aussi les automates internes des coeurs boundary scan suivent en parallèle l'automate central. Or l'automate central, pour la décompression est constitué de 18 états, ce qui n'est pas le cas des automates internes qui n'en contiennent que 16. Les différents automates des coeurs boundary scan ne peuvent donc pas se synchroniser sur le contrôleur central.

Avec l'implémentation de la méthode de décompression que nous avons proposée il n'est pas possible de décompresser les données de test et les appliquer à des coeurs boundary scan avec la même instruction.

Une solution permettant d'intégrer les deux approches consisterait à tester d'une part les coeurs boundary scan individuellement à l'aide de l'instruction TAP_CONFIG puis de tester les coeurs wrappés en utilisant la décompression avec l'instruction DECOM-

PRESS_TEST. Cependant avec cette solution l'architecture globale perd de sa flexibilité, les coeurs boundary scan n'étant plus intégrés dans le processus d'optimisation du temps de test. Ces coeurs étant testés à part, les routeurs TAPCAS ne sont alors d'aucune utilité.

4.6 Conclusion

Nous avons présenté dans ce chapitre une nouvelle méthode de compressions/décompression/expansion de données. Cette méthode est prévue pour être appliquée à des données de test très faiblement corrélées. Nous avons optimisé cette méthode pour améliorer le taux de compression en tenant compte de ces faibles corrélations. Au niveau du temps de test, le gain moyen théorique obtenu par rapport à une simple expansion est de 33%, le gain maximal étant de 66%.

Pour traiter le cas des SoCs ayant un nombre de broches de test réduit, nous avons implémenté cette méthode pour l'intégrer dans l'architecture CAS-BUS. Les éléments ajoutés et modifiés ont été décrits en VHDL générique. Des outils ont été développés et sont présentés dans le chapitre suivant.

Cette nouvelle architecture s'adresse plus particulièrement aux systèmes contenant des coeurs munis de wrapper. Les avantages et fonctionnalités de l'architecture de base sont conservées.

Elle permet également de tester les systèmes intégrant les coeurs boundary scan. Mais dans ce cas, ces coeurs ne peuvent être testés en même temps que les autres. La flexibilité de l'architecture et le routage dynamique des données ne peut s'appliquer alors qu'aux coeurs munis de wrapper.

Chapitre 5

Outils et Résultats

5.1 Introduction

Dans les deux chapitres précédents nous avons présenté l'architecture CAS-BUS. Cette architecture a évolué pour apporter une réponse spécifique aux problèmes liés à certains SoC. Les problèmes que nous avons soulevés et auxquels nous proposons une solution sont d'une part le test des SoC intégrant des coeurs boundary scan et d'autre part le test des SoC au nombres de broches de test limitées.

Les architectures que nous avons proposées sont scalables et modulaires. Pour faciliter la conception d'une architecture adaptée aux SoC visés, nous avons développé un ensemble d'outils logiciels, paramétrables, générant les différents éléments qui constituent cette architecture.

Nous présenterons dans ce chapitre le générateur de modules CAS et TAPCAS. Nous décrirons ensuite le générateur de wrapper P1500 compatible avec la description actuelle de la norme. Des outils implémentant la technique de codage présentée précédemment sont aussi détaillés. Les résultats de synthèse correspondant aux différents modules générés seront présentés et discutés. Pour finir nous présenterons une première évaluation de l'architecture dans le cas d'un exemple de système intégré.

5.2 Générateur de CAS et de TAPCAS

5.2.1 Description de l'outil

L'outil que nous avons développé génère automatiquement le module CAS désiré en fonction de certains paramètres (figure 5.1). Ces paramètres sont :

- N et P. Largeur du bus de test et nombre d'entrées/sorties de l'IP wrappé.

- CAS ou TAPCAS. Il faut spécifier si le routeur va être associé à un coeur wrappé ou à un coeur boundary scan.

- Type de Switch. En fonction des contraintes en surface et en temps de test, l'utilisateur doit choisir le type de switch qui sera contenu dans le module CAS.

- Mux ? Ce paramètre est utilisé si l'outil doit générer un TAPCAS. Selon qu'il se trouve en série avec un CAS ou avec un autre TAPCAS, le module TAPCAS contiendra ou non un multiplexeur en entrée.

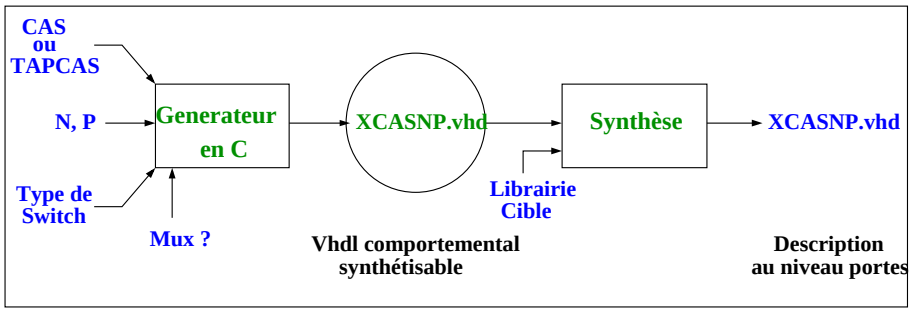


Figure 5.1: Entrées/Sorties du générateur

L'outil génère donc un routeur spécifique en VHDL comportemental synthétisable. Ce modèle comportemental est ensuite synthétisé avec un outil de synthèse dans une bibliothèque spécifique. Nous obtenons en sortie un routeur décrit en VHDL sous forme d'une netlist, décrite au niveau portes,optimisée. Un script a été développé pour que le processus de génération et de synthèse soit complètement automatisé.

La figure 5.2 rappelle brièvement les différences architecturales entre les deux types de switch disponibles. Elle précise aussi les différences entre un CAS et un TAPCAS. Cette figure détaille le générateur en précisant que celui ci a été développé de façon modulaire. Chacun des éléments du routeur est généré individuellement. L'assemblage des différents éléments nous donne un modèle comportemental.

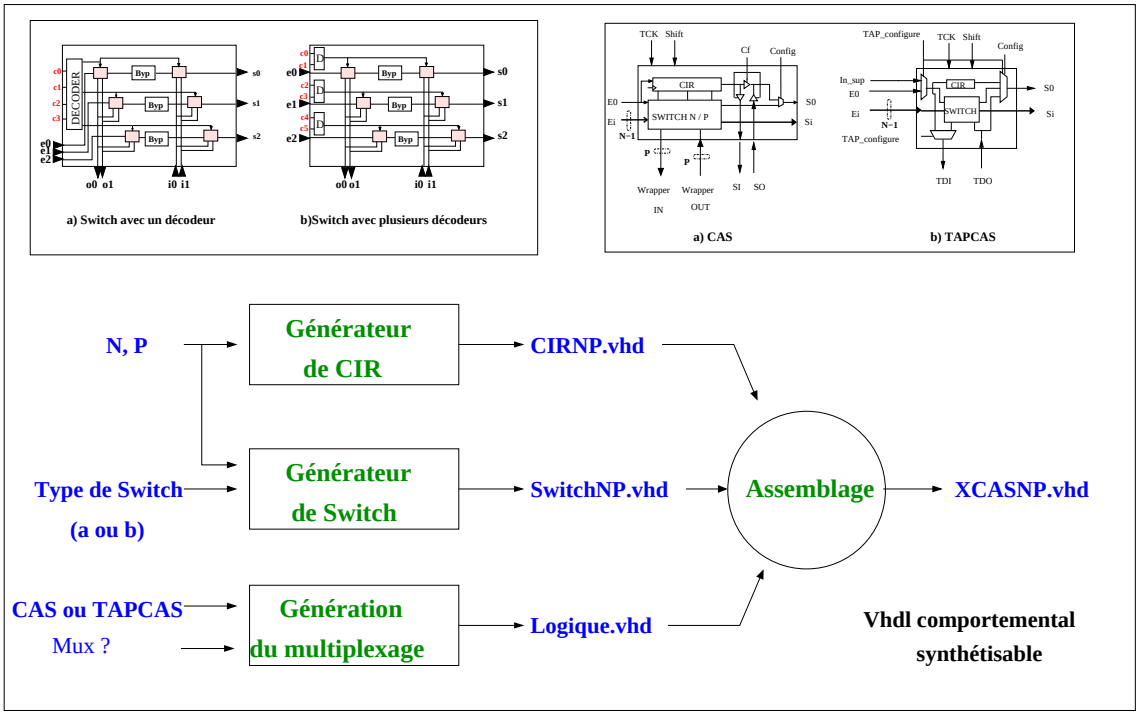


Figure 5.2: Génération modulaire

5.2.2 Résultats

Nous avons généré un ensemble de CAS et de TAPCAS en faisant varier les paramètres d'entrée. L'outil de synthèse que nous avons utilisé est Synopsys Design Compiler. Pour valider les routeurs générés, nous les avons associés à des modules contenant uniquement des chaines de scan. Ces modules ont été ajoutés pour représenter des coeurs wrappés. La validation s'est faite à travers des simulations au niveau porte utilisant le simulateur de Synopsys VSS et la bibliothèque de cellules SXLIB. Cette bibliothèque est développée et utilisée par le laboratoire LIP6 dans la chaîne de CAO ALLIANCE ([LA]).

Les résultats de synthèse sont rassemblés dans le tableau 5.1. En fonction de N et P, nous présentons dans ce tableau la longueur du registre d'instruction (k) et la surface en nombre de transistors (tr) des CAS et des TAPCAS. Pour un même couple (N,P) chaque CAS et TAPCAS a été généré avec les deux types de switch. L'indice (a) signifie que le CAS possède un switch avec N décodeurs, l'indice (b) correspond à un CAS intégrant un switch avec un seul décodeur.

On rappelle que pour un switch avec N décodeurs (a) :

$k = N.\lceil log_2(P+1) \rceil$ et m=P+1

		CAS (a)		CAS (b)		TAPCAS (a)		TAPCAS (b)	
N	P	k	tr	k	tr	k	tr	k	tr
5	1	5	696	3	626	5	684	3	614
5	3	10	1.224	6	2.026	–	–	–	–
5	4	15	1.636	7	2.634	–	–	–	–
8	1	8	1.084	4	912	8	1.072	4	900
8	3	16	1.952	9	7.286	–	–	–	–
8	4	24	2.548	11	39.990	–	–	–	–
8	6	24	2.948	15	>100.000	–	–	–	–
10	1	10	1.338	4	1.044	10	1.326	4	1.032
10	4	30	3.236	13	>100.000	–	–	–	–

TAB. 5.1 – CAS synthesis results

Pour un switch avec un seul décodeur (b) :

$k = \lceil log_2(m) \rceil = \lceil log_2(N!/((N-P)!) + 2) \rceil$ et m=N !/((N-P) !) + 2

m correspond aux nombres de combinaisons implémentées dans le switch.

Surface

La première constatation que l'on peut faire à la lecture du tableau est que k et tr varient en fonction du type de switch. La surface des CAS de type (b) a tendance à très fortement augmenter avec N et P. Ceci provient du fait que le nombre de combinaisons pour ce type de switch grandit en N !/(N-P) !. Nous n'avons pas pu synthétiser les CAS(b) pour des valeurs élevées de N et P à cause de la taille mémoire nécessaire à Synopsys pour effectuer la synthèse.

Par contre, pour les CAS(a) la complexité des décodeurs varie linéairement avec P. Les CAS et TAPCAS intégrant ce type de switch nécessitent moins de surface.

Longueur des CIR

Nous remarquons aussi dans ce tableau que la longueur du registre d'instruction (k) grandit plus rapidement pour les CAS(a) que pour les CAS(b). Cette différence d'évolution provient du fait que pour les switchs de type (a) k évolue en fonction de $N.\lceil log_2(P+1) \rceil$ alors que pour l'autre type de switch k varie en fonction de $\lceil log_2(N!/((N-P)!) + 2) \rceil$.

Règles induites

Lorsque l'intégrateur système a déterminé la largeur de son bus de test (N), il doit

ensuite sélectionner le type de switch qu'il utilisera dans son routeur. A partir des constatations précédentes nous pouvons énoncer trois règles à suivre concernant le choix du type de switch.

- Lorsque P vaut 1, quel que soit N, il vaut mieux générer un CAS utilisant un switch avec un seul décodeur. Pour les TAPCAS (P=1) ce type de switch est préférable.

- Lorsque P est faible, le choix ne peut être fait qu'en fonction des contraintes de surface et de temps de test.

- Pour des valeurs de N et P élevée, les CAS utilisant les switch avec N décodeurs sont à privilégier. La longueur du CIR augmente mais la surface n'explose pas.

5.3 Générateur de Wrapper P1500

A partir des spécifications fonctionnelles données par le groupe IEEE P1500 ([p15]) nous avons réalisé un générateur de wrapper qui vient encapsuler un coeur donné.

5.3.1 Description de l'outil

Ce générateur d'interface prend en entrée (figure 5.3) une instance du coeur à encapsuler. Cette instance correspond à une description en VHDL des entrées/sorties du coeur. Il génère en sortie un modèle en VHDL structurel du coeur wrappé.

Pour générer le wrapper adéquat, le générateur prend en entrée un ensemble de fichiers et de paramètres :

- Une bibliothèque de cellules.
- Le type de cellule choisi.
- Un fichier CTL (Core Test Language) contenant les informations sur l'interface et la testabilité du coeur.

Nous avons développé les différents éléments définis par le groupe P1500. Ces éléments ont été écrits en VHDL comportemental. Cette bibliothèque est constituée du registre de bypass, du registre d'instruction, de la logique de décodage associée et des cellules du registre boundary (WBR). Les cellules du registre WBR sont disponibles sous trois version différentes suivant la fonctionnalité désirée.

La norme prévoit l'utilisation au choix de trois types de cellules comme nous l'avons décrit dans le chapitre 2 : une cellule simple, une cellule type boundary scan et une cellule plus complexe permettant le mode "transfert". N'ayant pas assez d'information sur

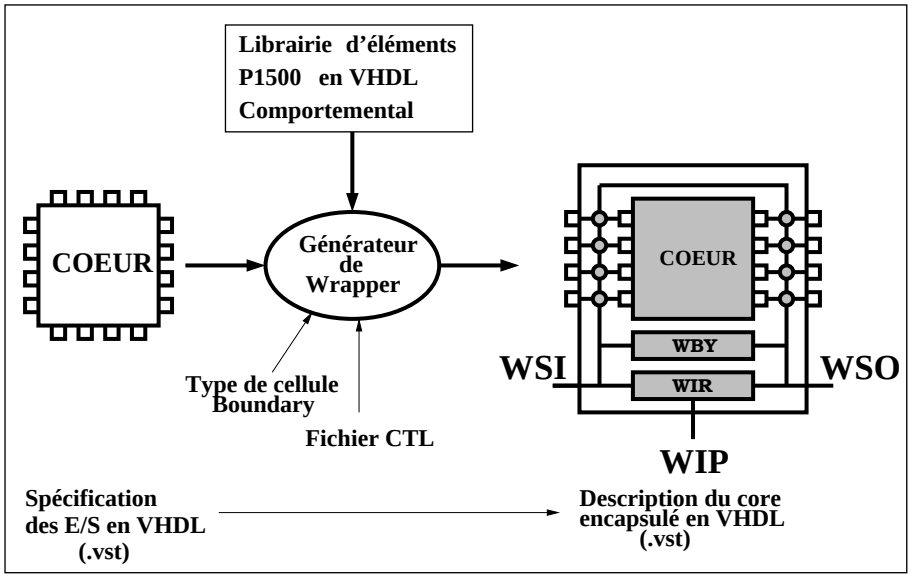


Figure 5.3: Entrées/Sorties du générateur de wrapper

cette dernière nous avons implémenté les deux premières auxquelles nous avons ajouté une cellule spéciale. Cette cellule est une cellule INOUT qui se positionne sur les plots bidirectionnels du coeur. L'implémentation des différents éléments de la bibliothèque est donnée en annexe.

Le deuxième point d'entrée du générateur est le paramètre correspondant au choix du type de cellule du registre WBR. L'utilisateur peut choisir entre une cellule simple et une cellule de type boundary scan, cette dernière intégrant le mode de mise à jour. L'ajout d'une cellule INOUT est automatique et dépend de la description faite des entrées/sorties.

Le standard définit actuellement deux types de compatibilité IEEE P1500. Les fournisseurs d'IP pourront soit délivrer un IP nu auquel ils devront associer un fichier CTL nécessaire à la génération du wrapper, soit délivrer l'IP wrappé avec le fichier CTL correspondant. Le fichier CTL contient toutes les informations nécessaire au test du coeur et/ou les informations nécessaires à la génération du wrapper.

Le lange CTL n'étant pas encore complètement défini, nous avons conçu notre générateur en prévoyant une utilisation ultérieure des informations contenues dans ce fichier. Le fichier que nous utilisons pour l'instant contient les informations relatives aux plots bidirectionnels et aux entrées/sorties de test. En effet pour les entrées/sorties dédiées au test, d'après la norme, il n'est pas nécessaire de connecter une cellule.

Ce générateur est écrit en langage C et doit manipuler des netlists en VHDL. Cela implique la définition d'une structure de données et la mise en place d'analyseurs et de pilotes

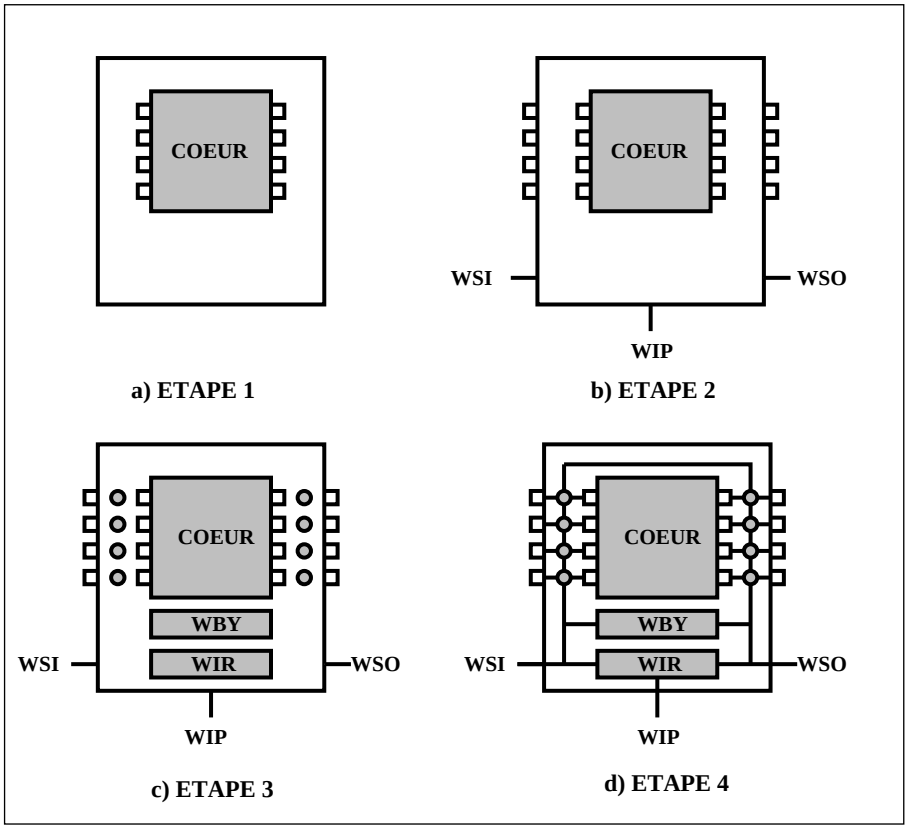


Figure 5.4: Etapes de conception du générateur

("parser/driver"). Pour simplifier la réalisation de ce logiciel, nous avons utilisé les facilités offertes par la chaine de CAO ALLIANCE utilisée dans notre laboratoire. Nous avons utilisé les formats vst (netlist) et vbe (modèle comportemental) qui sont des sous-ensembles du VHDL. Dans ALLIANCE il existe une structure de données qui permet de manipuler ces formats. Il s'agit de la structure "lofig". Dans ALLIANCE des fonctions permettant la manipulation de cette structure de données ont été définies. Notre générateur utilise la structure de donnée lofig et les fonctions associées. Des informations supplémentaires sur cette structure sont données en annexe.

Pour faciliter la portabilité de la bibliothèque, les différents éléments sont décrit en VHDL comportemental (vbe). L'outil VASY de la chaine de CAO ALLIANCE permet de transcrire les fichiers vst et vbe en vhdl standard synthétisable par les outils commerciaux.

5.3.2 Etapes de conception du générateur

La figure 5.4 décrit les étapes de conception du générateur. Dans un premier temps nous créons la structure de données lofig du core encapsulé. La seconde étape consiste à ajouter les différents connecteurs au coeur wrappé. A chaque plot du coeur correspond un connecteur du wrapper. A ces connecteurs viennent s'ajouter les connecteurs correspondant à WSI, à WSO et aux signaux du WIP. La troisième étape consiste à inclure dans la structure de données les instances P1500 puisées dans la bibliothèque. Pour finir, la dernière étape permet de générer les signaux internes et de faire les connexions adéquates.

5.3.3 Résultats

La validation du générateur s'est faite à travers la simulation des architectures générées. Cette simulation a été faite en utilisant l'outil **ASIMUT** de la chaine ALLIANCE [LA]. Cette validation s'est faite en deux phases. La première concernait la simulation des éléments de la bibliothèque du wrapper. Chaque élément a été validé individuellement.

La seconde phase de validation correspond à la simulation des coeurs wrappés générés. Nous avons pris dans un premier temps comme coeurs exemples des boîtes noires de spécifications différentes. Nous avons ensuite encapsulé un processeur AMD2901. Cet AMD a été utilisé pour simuler une session de test interne. A cette première session nous avons ajouté une seconde session de test correspondant au test des interconnexions entre deux AMD. Ces deux sessions nous ont permis de valider le fonctionnement des différents modes et instructions du wrapper.

Pour donner une idée du coût de la surface additionnelle induite par le wrapper, nous avons synthétisé les différents éléments de la bibliothèque. Cette synthèse s'est faite en utilisant l'outil BOOG d'ALLIANCE avec comme bibliothèque cible la bibliothèque de cellules SXLIB. Le tableau 5.2 rassemble les résultats de synthèse correspondant à la surface de chaque élément.

On constate qu'il faut un minimum de 500 transistors environ pour implémenter un wrapper P1500. La surface va croître en fonction de la longueur du registre WBR. Il y aura donc autant de cellules de test que de connecteurs sur le core. Le choix d'une cellule de test simple par rapport à une cellule implémentant le mode update peut devenir crucial si l'IP possède beaucoup d'entrées/sorties. Nous n'avons pas défini la cellule "transfert" mais on peut d'ores et déjà dire, au vu de ses fonctionnalités, que sa surface sera supérieure à celle de la cellule "update".

Cellule	Nombre de transistors
WIR	240
cellule de test	64
cellule de test(update)	90
étage de décodage	110
WBY	38

TAB. 5.2 – Surface des éléments du wrapper P1500

La surface du wrapper dépend donc du nombre d'entrées/sorties du coeur qu'il va encapsuler et de la complexité des cellules du WBR.

5.4 Compression / Décompression

Dans le chapitre précédent nous avons présenté une méthode de compression/décompression/expansion de données. Le module de décompression écrit en VHDL comportemental générique et synthétisable a été présenté à cette occasion. Pour compléter et valider cette méthode nous avons développé un logiciel de compression de données de test écrit en C.

5.4.1 Présentation de l'outil

Cet outil (figure 5.5) prend en entrée un ensemble de fichiers correspondant aux vecteurs à appliquer sur chaque entrée du TAM. Pour un TAM de largeur N nous avons donc N fichiers de vecteurs. Dans un premier temps nous avons considéré que chaque fichier pat.vhd décrivait les vecteurs sous forme d'affectations temporelles. Cet outil génère en sortie un fichier stimul.vhd correspondant aux valeurs compressées des N fichiers pat.vhd. Il est constitué des valeurs à affecter à l'horloge TCK, à TMS et à TDI. Sur TMS on retrouve les informations correspondant aux sauts à effectuer. Les valeurs affectées à TDI représentent, selon la valeur de TMS, soit une donnée compressée soit une indication sur le sens du saut. Le fichier généré est prévu pour être appliqué à l'entrée du décompresseur lors de la simulation.

Il faut noter que les vecteurs décrits par les fichiers pat.vhd correspondent aux valeurs à appliquer sur le bus. Le fichier généré stimul.vhd correspond donc, par rapport au test global, seulement à la phase de décompression et de décalage. Nous supposons donc que l'instruction DECOMPRESS_TEST est chargée et que nous partons de l'état *Shift_DR*

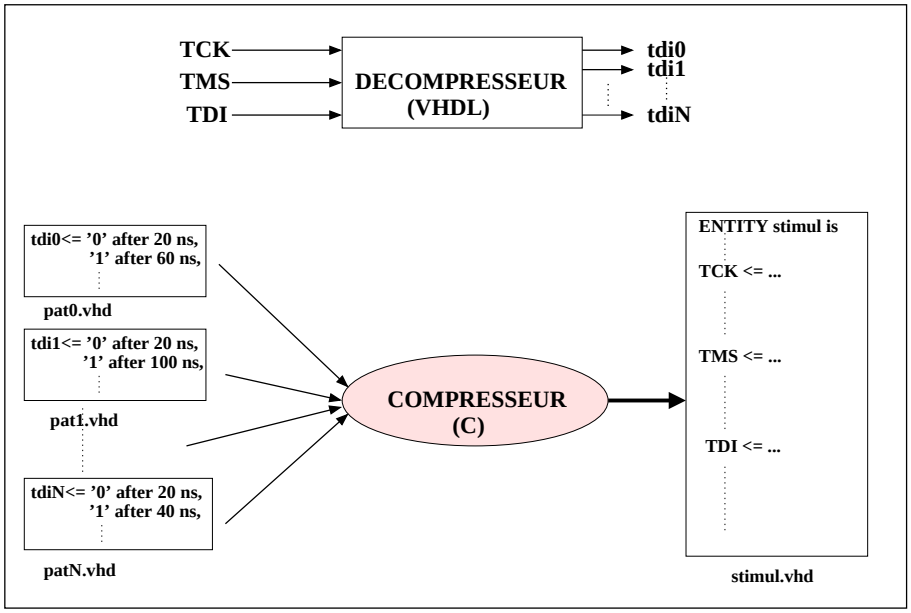


Figure 5.5: Entrées/Sorties du compresseur

dans le TAP modifié. La dernière valeur affectée à TMS dans le fichier stimul.vhd correspond à la fin de la décompression qui est caractérisée dans le controleur par l'entrée dans l'état *Exit1_DR*.

5.4.2 Description de l'outil

Le logiciel est constitué de trois modules (figure 5.6) : un module de sérialisation, un module d'optimisation et un module de compression. Nous avons implémenté l'outil pour deux valeurs de n (longueur de l'échantillon) : n=3 et n=4. Par souci de clarté nous décrivons ce qui suit avec n=4.

Sérialisation

Ce module fait d'abord une analyse syntaxique des différents fichiers pat.vhd. Il génère ensuite pour chacun de ces fichiers un fichier binaire bin_i (bin_1, bin_2,..., bin_N) constitué d'un train de bits de 0 et de 1. Cette suite de valeurs correspond aux données à appliquer après expansion à une ligne du TAM.

Nous faisons ici l'hypothèse que le nombre de bits à appliquer sur chaque entrée du bus est pratiquement identique. Cela revient à considérer que les longueurs des chaînes de scan du bus ont été auparavant équilibrées grâce à un routage adapté des données de test.

L'analyse syntaxique détermine le nombre de bits de chaque fichier bin_i et repère le

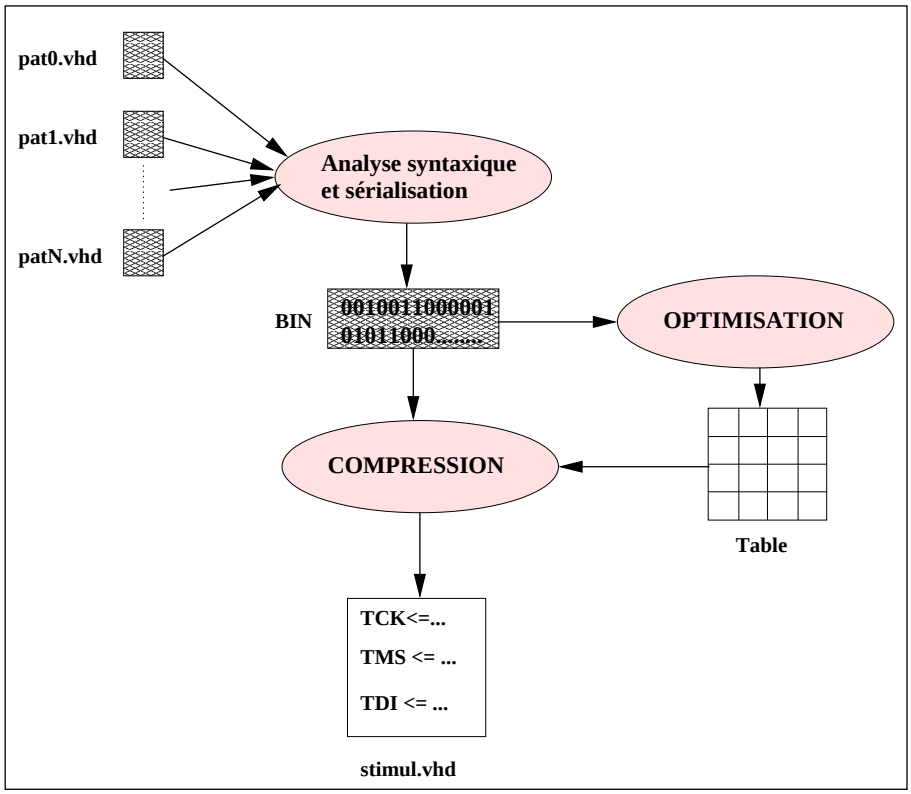


Figure 5.6: Structure du logiciel de compression de vecteurs de test

fichier le plus long. Les fichiers binaires sont ensuite complétés pour que non seulement ils aient tous la même taille mais aussi pour que le nombre de bits soit un multiple de 4, 4 étant la longueur de l'échantillon à compresser.

Chaque fichier bin_i est donc complété avec une suite de X (don't care). Ces X sont insérés au début du fichier bin_i car une fois décompressées ces valeurs seront les premières insérées dans les chaînes de scan du TAM et seront éliminées par les dernières valeurs chargées.

La dernière tâche de ce module consiste à générer un fichier binaire BIN à partir des différents fichiers bin_i. On prélève successivement quatre bits dans chaque fichier bin_i qu'on place en série dans le fichier BIN. On réitère ce processus jusqu'au prélèvement du dernier quartet du dernier fichier bin_N.

Nous avons vu auparavant que les échantillons étaient répartis entre quatre groupes différents. Le premier quartet du fichier BIN détermine donc le groupe de départ. Pour éviter de perdre des cycles au moment de la décompression, les valeurs "X" sont remplacées par un 0 ou un 1. La substitution se fait de façon à minimiser le nombre de sauts entre deux échantillons successifs.

Optimisation

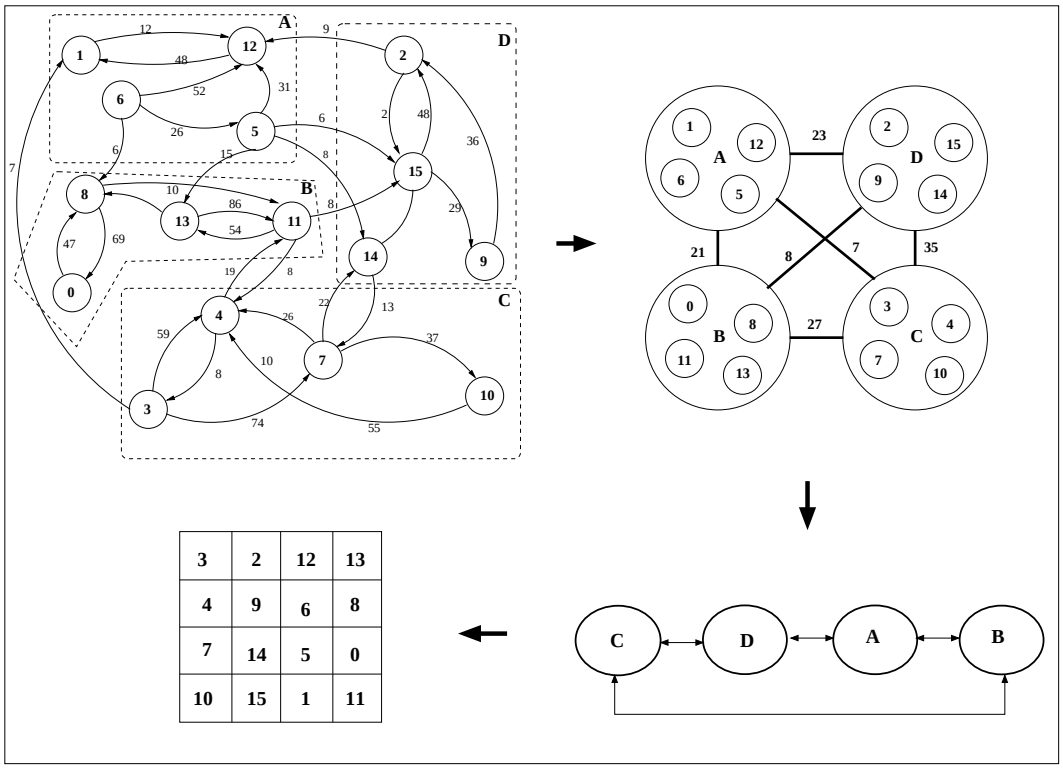


Figure 5.7: Partitionnement : recherche de la table optimale

Le module d'optimisation sert à générer ce que l'on appelle la table optimale à partir du fichier BIN. Nous avons vu qu'un échantillon de quatre bits donnait lieu à 16 combinaisons différentes. Cette table correspond à la répartition des différentes combinaisons dans quatre groupes ordonnés. Nous décrivons dans la figure 5.7, brièvement, le principe de partitionnement des différentes combinaisons.

A chaque groupe de quatre bits (quartet) est associé sa représentation décimale. A partir de cette représentation, un graphe composé de 16 noeuds est construit. Le poids des arcs correspond au nombre de fois où l'on passe d'un quartet à l'autre. Le logiciel lit une première fois le fichier de vecteurs pour construire ce graphe. Les quartets sont lus un à un de facon linéaire.

Il effectue ensuite un partitionnement de ce graphe pour obtenir quatre groupes de quatre noeuds. La fonction de coût du partitionnement est égale à la somme des poids des arcs reliant les quatre groupes. Les arcs internes à un groupe de quatre noeuds n'entrent pas en compte dans le calcul de la fonction de coût. La recherche du partitionnement optimal s'effectue de manière exhaustive, la complexité et le temps de calcul n'étant pas très

élevés (environ 30 secondes).

Une fois le ou les partionnements déterminés, il s'agit d'ordonner les quatre groupes. Leur classement permet de déterminer le nombre de cycles nécessaires à l'application des vecteurs de test. La table optimisée correspond au partitionnement optimal dont le résultat est ordonnancé et dont le premier groupe contient le premier quartet du fichier BIN.

Compression

A partir du fichier BIN et de la table optimale, le module de compression génère les échantillons compressés et les informations relatives aux sauts et aux sens des sauts. Ce module produit en sortie le fichier stimul.vhd affectant les bonnes valeurs sur TCK, TMS et TDI.

Ce fichier de sortie est constitué de trois phases d'affectation. La première correspond à la transmission sur TDI, vers le décompresseur, de la table de codage optimisée. Lorsque l'échantillon est de longueur 4, il faut charger les 16 combinaisons correspondantes. Cela représente 64 cycles d'initialisation. Dans le cas d'un échantillon de 3 bits, cette initialisation se fait en 24 cycles. La deuxième phase correspond à l'envoi du premier quartet déterminant le groupe de départ. La troisième phase est la phase de décompression proprement dite, on génère sur TDI les échantillons compressés et les indications de saut.

5.4.3 Résultats

Décompresseur

La validation du décompresseur s'est faite à travers la synthèse et la simulation du module pour différentes valeurs de N et de n, avec les outils de Synopsys, en utilisant la librairie SXLIB d'ALLIANCE. La synthèse a été faite avec Design Compiler, la simulation avec VSS. Les résultats en termes de surface sont rassemblés dans le tableau 5.3. La

N	3	4	5	10	20	50	100
nb de transistors (n = 3)	7940	8120	8304	9160	11856	16048	23972
nb de transistors (n = 4)	10032	10238	10454	11594	13728	20888	30258

TAB. 5.3 – Taille du décompresseur en fonction de N et n

surface est donnée en nombre de transistors en fonction de N (nombre de sorties du décompresseur) et de n (longueur de l'échantillon décompressé). Ces résultats nous montrent que le décompresseur possède un coût de départ relativement important, de l'ordre de 8000

transistors pour n=3 et de 10000 transistors pour n=4. Cependant l'ajout de sorties supplémentaires provoque une faible variation de la surface. Pour un n donné la fonction de coût surface=f(N) varie comme une droite d'équation y=ax+b. Pour n=3, b=7960 et a=165, pour n=4, b=10032 et a=208. La variation de surface en fonction de N provient du fait que le module de décalage contient N registres. Cette surface varie aussi en fonction de n car le module table contient 2^n échantillons et le module décalage se compose de N registres de longueur n.

Du point de vue de la surface ajoutée par ce module de décompression/expansion, il apparaît qu'un module réalisé pour n=3 est préférable au même module développé pour n=4.

Compresseur

Pour ce qui est du module logiciel, la validation s'est faite en plusieurs phases. Nous avons compressé un ensemble de fichiers de vecteurs pat (pat0.vhd, pat1.vhd..., patN.vhd). Après vérification, la sortie du compresseur (stimul.vhd) a servi de point d'entrée à la simulation du décompresseur correspondant. Un petit outil a permis de comparer les vecteurs présents sur les N sorties du décompresseur avec les N entrées du compresseur.

Pour évaluer l'efficacité de la technique de compression, nous l'avons appliquée à plusieurs fichiers de vecteurs. Nous l'avons appliqué aux vecteurs de test de PCIDDC , un circuit développé au laboratoire ASIM/LIP6 ([WDS+97]). Ce circuit est une interface réseau/bus PCI. Il est constitué de 200K transistors et intègre 4 chaînes de scan de longueur 1028. Nous avons aussi appliqué la technique de compression aux vecteurs de test des circuits benchmarks de ITC99 ([Dav]). Les taux de compressions obtenus pour ces différents circuits sont regroupés dans le tableau 5.4.

Les différentes compressions ont été obtenues pour différentes valeurs de n. La théorie nous précise que pour un échantillon de longueur 4, le gain maximal est égal à 50% et le gain moyen est de 25%. Pour n=3, que nous avons défini comme longueur optimale, le gain maximal est égal à 66% et le gain moyen de 33%.

Les gains que nous présentons dans ce tableau correspondent à une compression qui, selon le cas, intègre ou non la technique d'optimisation de la table de décodage. Lorsque ce n'est pas le cas cette table est initialisée avec 2^n échantillons. Ces échantillons sont répartis dans les quatres groupes dans l'ordre croissant de leur équivalent décimal.

Ces résultats nous permettent de voir que les gains effectifs sont, pour des fichiers assez volumineux, voisins des gains moyens donnés par la théorie (33% et 25%). Cela provient du fait que les données de test sont faiblement corrélées. L'optimisation de la table de codage

	Gain (%) pour n=3		Gain (%) pour n=4		nb cycles total
Circuit	Sans Opt.	Avec Opt	Sans Opt.	Avec Opt.	
b01	33.7	36.7	23.7	32.7	480
b02	29.1	44.8	28.1	49	96
b03	33.2	36.5	26.2	28.5	996
b04	31.3	38.9	28.2	34.4	308
b06	27.3	41.5	27.2	33.6	234
b09	31.9	36.1	23.2	29.7	717
b10	33.3	33.9	25	26	15048
b13	33.1	33.6	24.9	25.4	76380
b14	33.3	33.5	25	25.3	246816
b20	33.3	33.5	24.9	25.3	349728
PCIDDC	34.8	36.7	25.1	26.9	12336

TAB. 5.4 – Taux de compressions obtenus avec notre technique de codage

permet d'améliorer le taux de compression des données quand une corrélation existe.

Nous avons compressé les même fichiers en utilisant le codage de Huffman qui fait intervenir la notion de corrélation. Cette technique de codage possède un bon taux de compression lorsque le nombre d'occurences des échantillons est non uniforme.

Les différentes compressions ont été réalisées en prenant à chaque fois une longueur d'échantillon différente. Cette longueur fait varier le nombre d'occurences et possède donc une conséquence directe sur le codage des échantillons.

Les taux de compression des différents circuits sont présentés dans le tableau 5.5. Dans ce tableau les gains sont donnés pour des longueurs de mot différentes. Les gains ne commencent à être intéressants que pour des valeurs de n supérieure à 8. Pour les circuits b13, b14 et b20 le codage ne devient positif que pour n=16.

Ces gains ne tiennent pas compte du nombre de cycles nécessaires pour transmettre la table de codage. Bien évidemment plus la longueur du mot est grande plus le nombre de cycles nécessaires à la transmission des informations de décodage est élevé. Parmi ces informations à transmettre on retrouve la table de codage mais aussi l'arbre d'Huffman qui varie en fonction du fichier compressé. Dans le cas des circuits présentés, nous avons mesuré le taux de compression corrigé qui inclut ces cycles supplémentaires. Pour tous les circuits il est négatif.

	Gain (%)				
Circuit	n = 3	n =4	n = 8	n = 16	nb cycles total
b01	0	3.1	30.4	69.2	480
b02	4.2	7.3	54.2	82.3	96
b03	0	0.2	18.5	62.7	996
b04	2.6	2.3	33	72.5	308
b06	0	6.9	38.8	75.4	234
b09	0	0.1	20	65.1	717
b10	0	0	0.9	38.1	15048
b13	0	0	0	23.7	76380
b14	0	0	0	14.3	246816
b20	0	0	0	11.4	349728
PCIDDC	0	0	1.3	40	12336

TAB. 5.5 – Gains obtenus en utilisant le codage d'Huffman

Pour ce qui est de la technique de codage que nous avons développée, nous n'avons pas non plus tenu compte du nombre de cycles supplémentaires nécessaires pour transmettre la table de codage. Cependant ce nombre de cycles est constant et ne dépend pas du fichier à compresser. Pour n=3, il faut tenir compte des 3.2^3 (24) cycles supplémentaires. Pour n=4 l'initialisation du décompresseur nécessite 64 cycles.

Il n'est pas question ici de comparer notre technique de codage à celle d'Huffman. Ces deux techniques ne sont pas destinées à compresser les même types de fichiers. Nous avons utilisé ici la méthode d'Huffman pour montrer que les gains en compression que nous avons obtenus se basaient sur des vecteurs de test faiblement corrélés. En termes de gain, pour les circuits présentés, notre méthode s'avère donc plus intéressante qu'une méthode utilisant la corrélation des données. Pour les données de test fortement corrélées, les techniques de codage entropique de type Huffman peuvent être beaucoup plus efficaces. Pour des corrélations moyennes une étude des deux méthodes doit être menée. Cette étude doit tenir compte du taux de compression mais aussi de la surface du décompresseur correspondant.

5.5 Premières évaluations sur un benchmark

N'ayant pas de système intégré réel sur lequel nous pourrions évaluer notre architecture CAS-BUS, nous avons décidé d'obtenir quelques résultats sur un des benchmarks devant être présentés à ITC2002 ([MIC]). Ces circuits sont des SoC pour lesquels les informations suivantes sont disponibles :

- nombre de modules (IPs).

- niveau de hiérarchie des différents modules.

- Nombre d'entrées/sorties/plots bidirectionnels pour chaque module.

- nombre de chaînes de scan internes à chaque module.

- longueur de chacune de ces chaînes de scan.

- nombre de vecteurs de test à appliquer à chacun des modules.

Les différents benchmarks mis à disposition ne contiennent pour l'instant aucune information concernant la surface du SoC et des coeurs qu'il contient. De même les informations sur la puissance consommée pendant l'exécution du test ne sont pas fournies.

Nous ne pouvons donc pas obtenir de résultats en terme de surface ajoutée relative à la surface totale. Cependant pour avoir un premier ordre de grandeur en surface absolue, nous avons généré les différents éléments de l'architecture CAS-BUS pour un de ces benchmarks : le SoC "g1023". Ce SoC est fourni sous la forme d'un unique fichier, un fichier texte qui contient les informations précitées. Ce fichier est présent en annexe.

Ne pouvant aller jusqu'aux étapes de placement/routage et de génération de layout, nous présentons les résultats des différentes synthèses en nombre de transistors.

Le SoC g1023 contient 14 modules. Le nombre de chaînes de scan de chaque module varie de 0 à 14. Selon le module, la longueur des différentes chaînes de scan est comprise entre 9 et 84 et le nombre de vecteurs de test à appliquer s'échelonne entre 15 et 1024. D'autres informations sont présentées dans le fichier en annexe.

Nous avons donc généré une architecture de test CAS-BUS de base pour ce SoC. Pour cela nous avons considéré que tous les modules devait être équippés d'un wrapper P1500. De plus, comme nous n'avons pas d'information sur la puissance consommée pendant le test, nous avons considéré qu'il n'y avait pas de limitation.

A chaque module nous avons associé un routeur CAS de paramètre N et P. La valeur de P pour chaque module est égale au nombre de chaînes de scan plus le registre boundary WBR du wrapper.

Pour établir une surface minimum et une surface maximum induites par cette architecture, nous avons déterminé un Nmax et un Nmin. Chaque CAS dépendant de N et P, P étant fixé, la surface totale varie donc seulement en fonction de N. La valeur maximale que peut prendre la largeur du bus de test (Nmax) correspond à un test en parallèle des différents modules. Nmax est donc égal à la somme des P. Nmin correspond par contre à la valeur maximale des différents P.

Lorsque N=Nmax la surface des différents routeurs CAS sera maximale mais le temps de test sera minimal puisque dans ce cas les modules sont testés en parallèle.

Par contre lorsque N=Nmin, la surface de l'architecture CAS-BUS est minimale mais le temps de test est maximal. Le temps de test peut être optimisé en équilibrant les chaines de scan présentes sur un fil du bus. Le nombre de vecteurs de test à appliquer aux différents modules n'étant pas le même, il faut prévoir différentes sessions de test. Pour chacune de ces sessions une configuration du routage des données de chaque CAS doit être trouvée.

Une architecture a été générée pour N=Nmax et une autre pour N=Nmin. Bien évidemment une architecture avec N=Nmax n'est pas réaliste puisque dans ce cas la présence des modules CAS est inutile, nous avons donc généré cette architecture seulement pour déterminer une borne supérieure de la surface ajoutée.

Le choix du type de CAS s'est fait en fonction de P. Lorsque P vaut 1, le CAS généré intègre un switch avec un décodeur. Dans tous les autres cas, on utilise l'architecture avec N décodeurs.

Nous avons utilisé le générateur de wrapper P1500 que nous avons présenté précédemment pour encapsuler chacun des modules. Deux types de wrapper ont été générés. Le premier intègre des cellules du WBR que nous qualifions de "simple". Ces cellules correspondent à des cellules minimales du point de vue fonctionnel et de la surface. Le deuxième type de wrapper généré contient des cellules "update" qui en plus des fonctions délivrées par la cellule simple autorise le mode de mise à jour.

Pour le SoC g1023, Nmax=49 et Nmin=15. Le tableau 5.6 rassemble les résultats de synthèse obtenus pour les différents coeurs du SoC. S_CAS représente la surface du routeur CAS en nombre de transistors et k correspond à la longueur du registre de configuration interne (CIR) de ce CAS. Dans ce tableau on retrouve donc la surface des CAS en fonction de N et P mais aussi la surface en nombre de transistors de chaque type de wrapper.

Ces résultats nous montrent que selon la valeur de N la surface totale des CAS peut varier du simple au triple (52.026 à 168.028). Le choix de N est donc primordial par rapport

Module	P	N=49		N=15		Wrapper	
		S_CAS	k	S_CAS	k	Simple	Update
1	15	29284	196	9014	60	26756	37468
2	3	11596	98	3582	30	28292	39628
3	2	10116	98	3128	30	23620	33058
4	5	16524	147	5102	45	19588	27388
5	5	16524	147	5102	45	4164	5698
6	3	11596	98	3582	30	2820	3808
7	3	11596	98	3582	30	2820	3808
8	3	11596	98	3582	30	9540	13258
9	2	10116	98	3128	30	6148	8488
10	2	10116	98	3128	30	43780	61408
11	2	10116	98	3128	30	21892	30628
12	2	10116	98	3128	30	20740	29008
13	1	4366	6	1420	5	8196	11368
14	1	4366	6	1420	5	16644	23248
Total		168.028		52.026		235.000	328.262

TAB. 5.6 – Synthèse des différents éléments de DFT

à la surface ajoutée. Mais ce qui est encore plus frappant c'est de voir que pour ce SoC, la surface ajoutée due au wrapper P1500 est bien supérieure à celle ajoutée par les routeurs CAS. La surface totale due aux wrappers augmente de 40 % selon qu'on utilise une cellule simple ou une cellule update. Le choix d'utilisation de l'une ou l'autre des cellules par l'intégrateur système est donc là aussi primordial. La norme P1500 en cours de développement prévoit d'associer un wrapper à chaque coeur, le coût en surface dû au wrapper est donc inévitable.

Selon les modules CAS générés, les longueurs des CIRs (k) sont comprises entre 6 et 196. La configuration du schéma de routage de chaque CAS prendra donc entre 6 et 196 cycles. Ces valeurs restent raisonnables en comparaison du nombre de cycles nécessaires à l'application des vecteurs de test. En effet nous rappelons que le nombre de vecteurs à appliquer à un coeur varie entre 15 et 1024. Ces valeurs restent raisonnables à condition de ne pas avoir trop de sessions de test car chaque session nécessite une nouvelle configuration. k peut donc devenir un paramètre important pour la détermination du nombre de sessions et par conséquent du temps de test.

	N=49		N=15	
	Simple	Update	Simple	Update
Surface CAS-BUS	182.764	182.764	66.762	66.762
Surface Wrapper	235.000	328.262	235.000	328.262
Surface totale	417.764	511.026	301.762	395.024

TAB. 5.7 – Surface totale dûe aux éléments de test

Le tableau 5.7 résume cette première évaluation que nous avons faite sur le benchmak g1023. Dans ce tableau la surface de l'architecture CAS-BUS présentée inclut la surface des routeurs CAS, la surface de la partie contrôle et la surface du boundary scan (cellules aux entrées/sorties du SoC). La surface totale (CAS-BUS + Wrapper) varie entre 301.732 et 511.026 transistors. L'architecture CAS-BUS prend donc une surface comprise entre 17 % et 44 % de la surface de test totale. Il faut bien noter cependant que c'est une surface en nombre de transistors et qu'il faut relativiser ces différentes surfaces par rapport à celle du SoC.

5.6 Conclusion

Nous avons présenté dans ce chapitre les outils nécessaires à la génération des éléments des trois architectures CAS-BUS.

Pour générer une architecture complète et scalable selon le SoC visé, nous avons développé plusieurs outils :

- un générateur de routeurs CAS et de routeurs TAPCAS.

- un générateur de wrapper P1500. Les wrappers générés sont conformes à la norme P1500 telle qu'elle est définie actuellement.

- un outil de compression destiné à la compression de données de test faiblement corrélées.

Pour compléter l'architecture, nous avons développé un décompresseur générique, décrit en VHDL comportemental synthétisable. Les parties contrôle des trois architectures, incluant les trois type de controleurs TAP, ont été modélisés séparemment, en VHDL comportemental synthétisable.

Les différents éléments constituant chacune des trois architectures ont été validés par synthèse et simulation. Les résultats de synthèse nous ont permis de définir quelques

règles à suivre en ce qui concerne le choix du type d'implémentation d'un élément de l'architecture.

Nous avons utilisé ces outils pour générer une architecture CAS-BUS spécifique au système intégré g1023, un des circuits benchmark d'ITC02 [MIC]. Nous avons fait une première évaluation de notre architecture sur ce SoC. Cette évaluation nous a permis de montrer que la surface ajoutée (en nombre de transistors) de l'architecture CAS-BUS peut varier du simple au triple selon le compromis temps de test/surface ajoutée qui a été fait. Cette étude a fait ressortir aussi l'importance de la surface additionnelle due au wrappers P1500.

Cependant les différents résultats de surface obtenus (CAS, wrapper...) doivent être relativisé au SoC et à l'IP auxquels ils sont associés. Certains IPs dépassent le million de transistors et contiennent des dizaines de chaînes de scan de longueur supérieure à 500.

Cela dit la surface de certains routeurs CAS générés est trop élevée et réduit le champ d'utilisation de l'architecture CAS-BUS à des SoC contenant des IPs de paramètres N et P peu élevé. Or la tendance est à l'intégration toujours plus grande d'IPs de plus en plus complexes sur une même puce. L'architecture CAS-BUS doit donc évoluer pour permettre le test de ce type de SoC. Sa surface peut être réduite de façon significative. Nous proposons donc un moyen d'optimiser la taille des routeurs CAS dans le chapitre suivant concernant les perspectives à ce travail.

Chapitre 6

Perspectives

6.1 Introduction

Nous avons présenté dans les chapitres précédents l'architecture CAS-BUS et les outils nécessaires à sa génération. Nous allons voir dans un premier temps quels sont les points à développer pour que cette architecture ne pâtisse plus des limites d'utilisation qui sont les siennes aujourd'hui. Nous présenterons ensuite une approche du test des systèmes sur silicium, basée sur l'utilisation du microprocesseur présent dans le SoC. Nous pensons que cette approche nouvelle, différente de celle que nous avons décrite jusqu'ici, devrait permettre de résoudre certains problèmes liés au test des SoC.

6.2 Evolution de l'architecture CAS-BUS

Pour parfaire les différentes architectures que nous avons présentées certains points doivent être traités.

6.2.1 Incompatibilité décompression/présence de coeurs boundary scan

Dans le chapitre quatre, nous avons vu que le test d'un SoC contenant des coeurs IEEE P1500 et de coeurs IEEE 1149.1, et intégrant notre méthode de décompression, n'était pas possible simultanément. Cela provient du fait que le contrôleur central contient 18 états et que les contrôleurs des coeurs boundary scan ne contiennent que 16 états. Pendant la phase de décompression ces derniers doivent suivre fidèlement le comportement du contrôleur central. Avec deux états de moins cela n'est pas possible. Le contrôle doit donc être revu

pour permettre à la fois de décompresser des vecteurs de test et les appliquer aux coeurs boundary scan.

Le décompresseur pour fonctionner a besoin de trois signaux en entrée : TDI, un indicateur de saut et un indicateur d'activation. Jusqu'ici, l'indicateur de saut est TMS et l'indicateur d'activation est généré par les états ONE_JUMP, TWO_JUMPS et SHIFT_DR de l'automate à 18 états utilisé pour la décompression. On peut envisager d'utiliser une nouvelle architecture utilisant le controleur à 32 états et l'instruction CAS_TEST, à condition de prévoir au niveau du SoC deux broches supplémentaires dédiées à la décompression, permettant d'indiquer la présence de sauts et l'activation/désactivation du compresseur. Cette solution nécessiterait donc deux broches de test supplémentaires.

6.2.2 Compression/Décompression

Un autre point à développer concerne l'application de méthodes de compression/décompression différentes de la nôtre à l'architecture CAS-BUS. Il serait intéressant de voir dans quelle mesure les techniques de codage/décodage statistiques et différentiels peuvent être implémentées et intégrées dans l'architecture CAS-BUS. Nous avons décrit dans cette thèse comment on pouvait décompresser et expanser des vecteurs de test faiblement corrélés. Pour mener une étude plus complète, il faudrait implémenter aussi d'autres techniques destinées cette fois-ci à des vecteurs fortement corrélés et voir les contraintes en surface que cela impliquerait.

Pour ce qui est du compresseur, une amélioration possible consisterait à modifier ses formats d'entrée et de sortie. Jusqu'ici, l'outil prend en entrée N fichiers patN.vhd correspondant chacun aux vecteurs à appliquer à un fil du TAM. Il génère en sortie un fichier stimul.vhd décrivant une succession d'affectations sur TDI, TMS et TCK. L'évolution naturelle de cet outil consisterait à prendre en entrée un seul fichier au format industriel STIL [Soc99] et de générer en sortie un fichier là aussi au format STIL, directement applicable.

6.2.3 Réduction de la surface des routeurs

Le principal frein à l'application de l'architecture CAS-BUS à des SoC importants provient de la surface des routeurs CAS. En effet cette surface dépend de N et P. Les coeurs sont de plus en plus grands et contiennent de plus en plus de chaines de scan. Il faut donc prévoir une façon de réduire la surface des routeurs afin qu'elle ne dépende plus de ces deux variables.

La surface qui explose provient du nombre de combinaisons à implémenter dans le Switch du CAS. Or une très faible partie de ces combinaisons sera réellement utilisée. L'idée serait donc de modifier le générateur de CAS pour qu'il génère un routeur avec comme paramètres N, P et les combinaisons qui seront effectivement utilisées.

Cela implique de connaître pour chaque CAS, avant la génération, les combinaisons qui seront effectivement utilisées. La détermination des combinaisons utiles se fait donc après la définition du plan de test.

Pour fournir ces combinaisons, une perspective à nos travaux serait de développer un outil qui produirait un plan de test optimisé et les différentes sessions correspondantes. L'optimisation des sessions de test permettrait de définir les configurations de routage de données et donc les combinaisons à implémenter. Pour générer ces configurations l'outil pourrait prendre en entrée des paramètres comme :

- les informations relatives au test de chaque coeur : type de test (BIST, scan...), nombre et longueurs des chaînes de scan, nombre de vecteurs à appliquer...

- les contraintes de temps de test.

- la puissance consommée maximale autorisée pour une session de test.

- le nombre de broches de test disponibles au niveau du SoC.

Le fichier CTL fourni avec le coeur pourrait être utilisé par ce nouvel outil.

6.3 Une nouvelle approche purement logicielle du test des SoC

L'architecture CAS-BUS que nous avons présentée dans cette thèse est un TAM scalable permettant d'acheminer les vecteurs de test du testeur externe vers les différents coeurs. Dans cette approche le programme de test s'exécute sur le testeur et l'accès aux différents coeurs se fait à l'aide d'un bus supplémentaire. Cette approche nécessite des broches de test supplémentaires dont le nombre est variable.

Nous avons commencé à réfléchir à une nouvelle approche du test des SoC où l'exécution du programme de test ne se ferait plus par le testeur mais par le microprocesseur interne du SoC. Le testeur ne serait plus alors qu'une immense mémoire contenant les données de test à appliquer. Cette nouvelle approche du test permettrait de s'affranchir de l'utilisation de testeurs de plus en plus couteux.

La figure 6.1 présente l'architecture de test que nous comptons développer pour valider cette nouvelle approche.

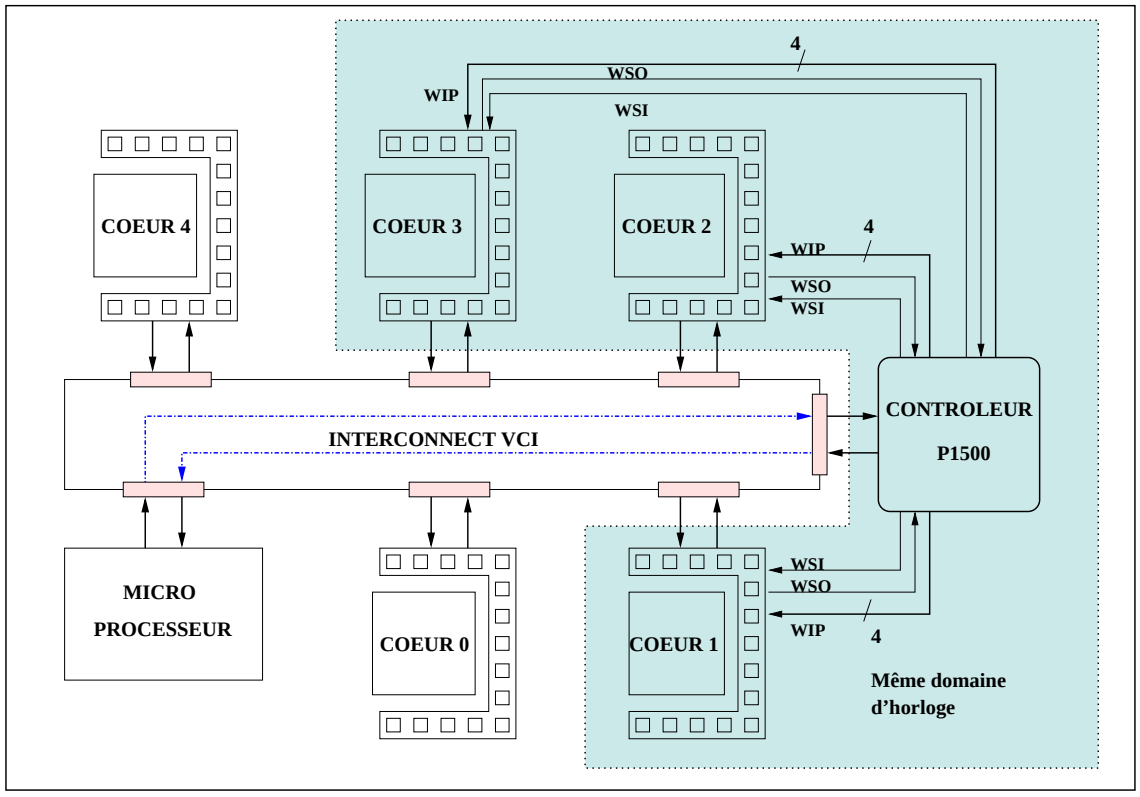


Figure 6.1: Nouvelle architecture visée

Dans cette architecture, les coeurs sont encapsulés dans des wrappers P1500 à interface série et sont connectés au bus système à travers une interface VCI (Virtual Component Interface). Cette interface est une interface standard, largement utilisée, mise au point par le consortium VSIA [vsi]. Les données échangées entre les différents modules du SoC à travers le bus sont disponibles sur 32 bits.

L'idée de départ est de définir un coprocesseur capable de recevoir des données de test du microprocesseur et de les distribuer aux différents coeurs.

Ce coprocesseur que nous avons appelé contrôleur P1500 possède donc d'une part une interface VCI et une ou plusieurs interface P1500 d'autre part. Il doit donc permettre le passage du protocole VCI au protocole P1500 et réciproquement. Les données présentes sur l'interface VCI sont sur 32 bits et l'alimentation d'un wrapper se fait de façon sérielle. Le coprocesseur doit pouvoir faire des conversions série/parallèle.

Les résultats de test des wrappers sont envoyés au microprocesseur via l'interconnect

pour être traités.

L'un des points importants visé par cette architecture est que le test des coeurs se fait à la fréquence d'horloge du système. Nous considérons dans un premier temps que le controleur P1500 alimente en vecteurs de test les IPs fonctionnant à la même fréquence. Si dans le système certains IPs fonctionnent à des fréquences différentes, il faudra mettre en place alors autant de controleurs P1500 qu'il y a de domaines d'horloge.

Cette nouvelle approche, si elle est validée présenterait les avantages suivant :

- utilisation de testeurs plus simple dit "low cost".

- exécution du programme de test par le microprocesseur à la vitesse de fonctionnement du circuit, ce qui réduirait considérablement le temps de test.

- le nombre de broches de test serait réduit et fixe quelque soit le nombre et le type de coeurs présents dans le SoC. Les données de test peuvent être stockées soit dans la mémoire du testeur soit dans une autre mémoire externe au circuit. Le nombre de broches pourrait correspondre aux nombres de broches de cette RAM.

Cette approche est en cours d'étude actuellement.

Conclusion générale

Avec l'apparition des systèmes intégrés sur une même puce et face à une densité d'intégration de plus en plus élevée, les techniques de conception en vue du test classiques ne suffisent plus à assurer un test efficace des circuits. L'industrie des semiconducteurs a tendance à se répartir entre fournisseurs et intégrateurs d'IPs. Ainsi la complexité croissante des System on a Chip (SoC) et la diversité des coeurs à intégrer a pour conséquence de positionner le test comme un goulot d'étranglement dans le développement d'un circuit intégré.

Le test des SoC faisant face à de nouveaux problèmes, de nouvelles techniques de DFT doivent être mises en place. Une harmonisation de l'interface de test fournie avec le coeur est en cours de développement grâce aux iniatives du groupe IEEE P1500 et du consortium VSIA. L'application de ces standards permettra sans doute de résoudre une partie des problèmes liés au test des SoC.

Cependant, les problèmes liés à l'intégration de ces coeurs restent entiers. Parmi ces problèmes, on retrouve celui du choix du mécanisme d'accès au test des coeurs (TAM). Ce choix est crucial car il détermine la surface du SoC, le temps de test, l'équipement de test nécessaire, ..., autant de facteurs ayant un impact direct sur le coût du circuit.

Les SoC et les IPs pouvant être différents, il faut pouvoir définir soit des TAM différents ciblés pour un type de SoC, soit des TAM que l'on qualifie de paramétrables et qui s'ajustent en fonction du SoC visé.

Quelques architectures de test ont été développées pour permettre le transport et l'application des données de test aux broches des différents IPs. Cependant ces architectures ne permettent de tester qu'un type de coeur donné, soit un coeur équippé d'un wrapper soit un coeur boundary scan. De plus, si certaines sont scalables, elles restent fortement liées au type de wrapper utilisé.

Nous avons présenté dans cette thèse une nouvelle architecture de TAM appelée CAS-

BUS dont les spécificités sont les suivantes :

- généricité : l'architecture est paramétrable et peut s'appliquer à différents SoC.

- reconfigurabilité dynamique : le routage des données n'est pas figé même lorsque le circuit est fondu sur silicium. Les configurations de routage se font par programmation.

- modularité : l'architecture est modulaire, ce qui simplifie sa conception et son intégration dans le SoC.

- indépendance vis à vis du type de wrapper utilisé.

- compatibilité avec le standard IEEE P1500 au niveau coeur, tel qu'il est défini dans l'état actuel, et compatible avec le standard IEEE 1149.1 au niveau SoC.

- "scalabilité".

Nous avons fait évoluer cette architecture pour qu'elle puisse permettre le test de SoC contenant deux types de coeurs différents, des coeurs équipés de wrappers et des coeurs boundary scan. Cette extension de l'architecture a permis en plus de la rendre complètement hiérarchique.

Nous avons aussi présenté dans cette thèse comment on pouvait adapter l'architecture CAS-BUS pour le test de SoC disposant de peu de broches de test. Une méthode de compression/décompression/expansion de données de test a été développée pour cela. Cette méthode permet d'obtenir en moyenne un gain de 33 % en temps de test par rapport à une expansion sans compression. Cependant cette méthode ne produit ses meilleurs résultats que pour l'application de données de test faiblement corrélées.

Nous avons développé dans cette thèse un ensemble d'outils permettant de générer une architecture de test complète. Parmi ces outils on retrouve un générateur de routeurs CAS et un générateur de wrappers P1500. Les éléments générés sont disponibles en VHDL comportemental synthétisable. Pour valider la technique de codage développée nous avons aussi conçu un outil permettant la compression des données de test.

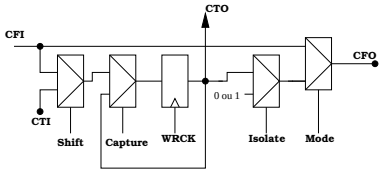
Le fait que l'architecture CAS-BUS soit scalable et reprogrammable permet à l'intégrateur système de générer une architecture qui corresponde au compromis surface ajoutée/temps de test qu'il s'est fixé.

Nous avons pour finir effectué une première évaluation de l'architecture CAS-BUS sur un système exemple, un SoC benchmark. Cette étude a fait apparaître les limites d'utilisation de l'architecture telle qu'elle est définie actuellement. La limite principale est due à une forte augmentation de la surface totale pour des SoC importants.

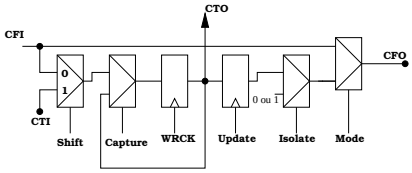
Cependant, dans le cadre de travaux futurs nous proposons un moyen de réduire considérablement la surface due à l'architecture CAS-BUS. Dans ce même cadre une nouvelle approche, une approche logicielle du test des SoC, est actuellement en développement.

Annexes

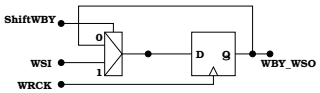
Annexe A : Les éléments de la bibliothèque du générateur de wrapper P1500 et la structure LOFIG



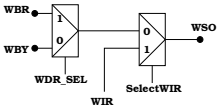
a) Cellule du registre WBR simple



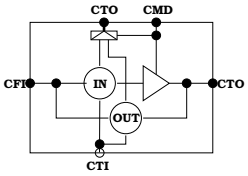
b) Cellule du registre WBR de type boundary scan



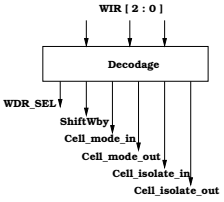
c) Registre de bypass



d) Multiplexeurs de sortie connectés à WSO

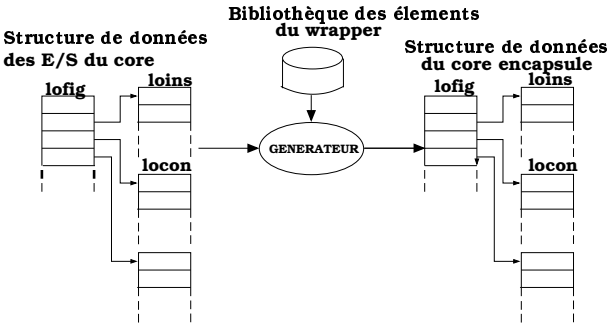


e) Cellule INOUT



f) Logique de décodage associée au WIR

g) La structure de donnée LOFIG



h) Utilisation de la structure LOFIG par le générateur

Annexe B : fichier décrivant le SoC g1023 "benchmark" d'ITC02

SocName g1023 TotalModules 15 Options Power 0 XY 0

Module 0 Level 0 Inputs 4 Outputs 63 Bidirs 53 ScanChains 0 :

Module 0 TotalTests 0

Module 1 Level 1 Inputs 139 Outputs 273 Bidirs 0 ScanChains 14 : 43 43 43 43 42 42 42 42 42 42 42 42 42 42

Module 1 TotalTests 1

Module 1 Test 1 ScanUse 1 TamUse 1 Patterns 134

Module 2 Level 1 Inputs 221 Outputs 215 Bidirs 0 ScanChains 2 : 84 83

Module 2 TotalTests 1

Module 2 Test 1 ScanUse 1 TamUse 1 Patterns 74

Module 3 Level 1 Inputs 192 Outputs 171 Bidirs 0 ScanChains 1 : 53

Module 3 TotalTests 1

Module 3 Test 1 ScanUse 1 TamUse 1 Patterns 57

Module 4 Level 1 Inputs 145 Outputs 155 Bidirs 0 ScanChains 4 : 54 54 54 54

Module 4 TotalTests 1

Module 4 Test 1 ScanUse 1 TamUse 1 Patterns 268

Module 5 Level 1 Inputs 32 Outputs 27 Bidirs 0 ScanChains 4 : 32 32 31 32

Module 5 TotalTests 1

Module 5 Test 1 ScanUse 1 TamUse 1 Patterns 51

Module 6 Level 1 Inputs 20 Outputs 18 Bidirs 0 ScanChains 2 : 47 47

Module 6 TotalTests 1

Module 6 Test 1 ScanUse 1 TamUse 1 Patterns 36

Module 7 Level 1 Inputs 20 Outputs 18 Bidirs 0 ScanChains 2 : 47 47

Module 7 TotalTests 1

Module 7 Test 1 ScanUse 1 TamUse 1 Patterns 34

Module 8 Level 1 Inputs 63 Outputs 80 Bidirs 0 ScanChains 2 : 52 52

Module 8 TotalTests 1

Module 8 Test 1 ScanUse 1 TamUse 1 Patterns 31

Module 9 Level 1 Inputs 56 Outputs 34 Bidirs 0 ScanChains 1 : 64

Module 9 TotalTests 1

Module 9 Test 1 ScanUse 1 TamUse 1 Patterns 68

Module 10 Level 1 Inputs 301 Outputs 377 Bidirs 0 ScanChains 1 : 13

Module 10 TotalTests 1

Module 10 Test 1 ScanUse 1 TamUse 1 Patterns 29

Module 11 Level 1 Inputs 145 Outputs 191 Bidirs 0 ScanChains 1 : 9

Module 11 TotalTests 1

Module 11 Test 1 ScanUse 1 TamUse 1 Patterns 15

Module 12 Level 1 Inputs 157 Outputs 161 Bidirs 0 ScanChains 1 : 13

Module 12 TotalTests 1

Module 12 Test 1 ScanUse 1 TamUse 1 Patterns 16

Module 13 Level 1 Inputs 58 Outputs 64 Bidirs 0 ScanChains 0 :

Module 13 TotalTests 1

Module 13 Test 1 ScanUse 0 TamUse 1 Patterns 512

Module 14 Level 1 Inputs 140 Outputs 114 Bidirs 0 ScanChains 0 :

Module 14 TotalTests 1

Module 14 Test 1 ScanUse 0 TamUse 1 Patterns 1024

Bibliographie

[A+99] Saman Adham et al. Preliminary Outline of IEEE P1500 Scalable Architecture for Testing Embedded Cores. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 483–488, Dana Point, CA, April 1999. IEEE Computer Society Press.

[ARM] ARM Web Site. AMBA Specifications. http ://www.arm.com/armtech/.

[BBG+98] A. Benso, G. Borgonovo, D. Grassi, M. Lobetti-Bodoni, and A. Pricco. An Industrial Approach to Core Testing. In *Digest of Papers of IEEE International Workshop on Testing Embedded Core-Based Systems (TECS)*, pages 1.3–1–4, Washington, DC, October 1998.

[BCC+99] Alfredo Benso, Silvia Cataldo, Silvia Chiusano, Paolo Prinetto, and Yervant Zorian. HD-BIST : A Hierarchical Framework for BIST Scheduling and Diagnosis in SOCs. In *Proceedings IEEE International Test Conference (ITC)*, pages 1038–1044, Atlantic City, NJ, September 1999. IEEE Computer Society Press.

[Bha98] D. Bhattacharya. Hierarchical test access architecture for embedded cores in an integrated circuit. In *IEEE VLSI Test Symposium (VTS)*, pages 8–14, Dana Point (CA), USA, April 1998.

[Boa90] IEEE Standard Board. IEEE std 1149.1-1990, "standard test access port and boundary scan architecture". 345 East 47th Street, New York, NY 10017-2394, 1990.

[CC01] A. Chandra and K. Chakrabarty. Efficient test data compression and decompression for system-on-a-chip using internal scan chains and Golomb coding. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 145–149, Munich, Germany, March 2001. IEEE Computer Society Press.

[Cha00] Krishnendu Chakrabarty. Design of System-on-a-Chip Test Access Architectures Under Place-and-Route and Power Constraints. In *Proceedings*

ACM/IEEE Design Automation Conference (DAC), pages 432–437, Los Angeles, CA, June 2000.

[Cha01] Krishnendu Chakrabarty. Optimal Test Access Architectures for System-on-a-Chip. *ACM Transactions on Design Automation of Electronic Systems*, 6(1) :26–49, January 2001.

[CP96] R. Chandramouli and Stephen Pateras. Testing Systems on a Chip. *IEEE Spectrum*, pages 42–47, November 1996.

[Dav] Scott Davidson. ITC'99 SOC Test Benchmarks Web Site. http ://www.cerc.utexas.edu/itc99-benchmarks/bench.html.

[DMZ99] Sujit Dey, Erik Jan Marinissen, and Yervant Zorian. Testing System Chips : Methodologies and Experiences. *Integrated System Design*, Vol. 11(No. 123) :36–48, September 1999.

[FW98] Chris Feige and Clemens Wouters. Integration of Structural Test Methods into a Architecture Specific Core-Test Approach. In *Digest of Papers of IEEE International Workshop on Testing Embedded Core-Based Systems (TECS)*, pages 5.2–1–8, Washington, DC, October 1998.

[GDJ98] Indradeep Ghosh, Sujit Dey, and Niraj K. Jha. A Fast and Low Cost Testing Technique for Core-based System-on-Chip. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 542–547, San Francisco, CA, June 1998. Association for Computing Machinery, Inc.

[GJD97] Indradeep Ghosh, Niraj K. Jha, and Sujit Dey. A Low Overhead Design for Testability and Test Generation Technique for Core-Based Systems. In *Proceedings IEEE International Test Conference (ITC)*, pages 50–59, Washington, DC, November 1997. IEEE Computer Society Press.

[GM01] Sandeep Kumar Goel and Erik Jan Marinissen. TAM Architectures and Their Implication on Test Application Time. In *Digest of Papers of IEEE International Workshop on Testing Embedded Core-Based Systems (TECS)*, pages 3.3–1–10, Marina del Rey, CA, May 2001.

[HR96] Merrill Hunt and James A. Rowson. Blocking in a System on a Chip. *IEEE Spectrum*, pages 35–41, November 1996.

[ICM01] Vikram Iyengar, Krishnendu Chakrabarty, and Erik Jan Marinissen. Test Wrapper and Test Access Mechanism Co-Optimization for System-on-Chip. In *Proceedings IEEE International Test Conference (ITC)*, pages 1023–1032, Baltimore, MD, October 2001. IEEE Computer Society Press.

[JGDT99] Abhijit Jas, Jayabrata Ghosh-Dastidar, and Nur Touba. Scan Vector Compression/Decompression Using Statistical Coding. In *Proceedings IEEE VLSI Test Symposium (VTS)*, Dana Point, CA, April 1999. IEEE Computer Society Press.

[JT98] Abhijit Jas and Nur Touba. Test Vector Decompression Via Cyclical Scan Chains and Its Application to Testing Core-Based Designs. In *Proceedings IEEE International Test Conference (ITC)*, pages 458–464, Washington, DC, October 1998. IEEE Computer Society Press.

[KNS98] Jake Karrfalt, Zainalabedin Navabi, and Casper Stoel. A Novel Approach to Optimization IEEE 1149.1 for Systems with Multiple Embedded Cores. In *Digest of Papers of IEEE International Workshop on Testing Embedded Core-Based Systems (TECS)*, pages 2.2–1–10, Washington, DC, October 1998.

[KW97] Bernd Koenemann and Ken Wagner. Test Sockets : A Test Framework for System-On-Chip Designs. http ://grouper.ieee.org/groups/1500/pastmeetings.html#970427, April 1997. Presentation at IEEE P1500 Working Group Meeting, Monterey, CA, April 1997.

[LA] LIP6-ASIM. Alliance CAD system. http ://www-asim.lip6.fr/alliance/. University Paris 6 - France.

[LH87] D. A. Lelewer and D. S. Hirschberg. Data compression. *ACM Computing Surveys (CSUR)*, vol. 19, No (3) : pp 261–296, September 1987.

[LP01] Erik Larsson and Zebo Peng. An Integrated System-on-Chip Test Framework. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 138–144, Munich, Germany, March 2001. IEEE Computer Society Press.

[M+98] Erik Jan Marinissen et al. A Structured And Scalable Mechanism for Test Access to Embedded Reusable Cores. In *Proceedings IEEE International Test Conference (ITC)*, pages 284–293, Washington, DC, October 1998. IEEE Computer Society Press.

[MA98] Erik Jan Marinissen and Joep Aerts. Test Protocol Scheduling for Embedded-Core Based System ICs. In *Digest of Papers of IEEE International Workshop on Testing Embedded Core-Based Systems (TECS)*, pages 5.3–1–9, Washington, DC, October 1998.

[Mar99] Walid Maroufi. *Aide à la conception de systèmes testables*. PhD thesis, Universite' Pierre et Marie Curie Paris France, Juillet 1999. Spe'cialite' informatique.

[MIC] Erik Jan Marinissen, Vikram Iyengar, and Krishnendu Chakrabarty. ITC'02 SOC Test Benchmarks Web Site. http ://www.extra.research.philips.com/itc02socbench/.

[ML97] Erik Jan Marinissen and Maurice Lousberg. Macro Test : A Liberal Test Approach for Embedded Reusable Cores. In *Digest of Papers of IEEE International Workshop on Testing Embedded Core-Based Systems (TECS)*, pages 1.2–1–9, Washington, DC, November 1997.

[MNBB98] Sobhan Mukherji, Loc Nguyen, Dwayne Burek, and Steve Baird. IP/VC-Based Test Methodology (Part-1) : A Case Study. In *Digest of Papers of IEEE International Workshop on Testing Embedded Core-Based Systems (TECS)*, pages 1.2–1–9, Washington, DC, October 1998.

[MO96] M. Marzouki and A. Osseiran. The ieee boundary scan standard : A test pradigm to ensure hardware system quality. *Quality Engineering Journal*, 8(4) :635–645, 1996.

[p15] IEEE P1500 Web Site. http ://grouper.ieee.org/groups/1500/.

[RT98] Janusz Rajski and Jerzy Tyszer. Modular Logic Built-In Self Test for IP Cores. In *Proceedings IEEE International Test Conference (ITC)*, pages 313–321, Washington, DC, October 1998. IEEE Computer Society Press.

[SIA99] Semiconductor Industry Association. International Technology Roadmap for Semiconductors., 1999.

[Soc99] IEEE Computer Society. IEEE standard test interface language (stil) for digital test vector data - ieee std. 1450". New York, 1999.

[VB98] Prab Varma and Sandeep Bhatia. A Structured Test Re-Use Methodology for Core-Based System Chips. In *Proceedings IEEE International Test Conference (ITC)*, pages 294–302, Washington, DC, October 1998. IEEE Computer Society Press.

[vBvH99] Jos van Beers and Harry van Herten. Test Features of a Core-Based Co-Processor Array for Video Applications. In *Proceedings IEEE International Test Conference (ITC)*, pages 638–647, Atlantic City, NJ, September 1999. IEEE Computer Society Press.

[vsi] VSI Alliance Web Site. http ://www.vsi.org/.

[WDS+97] F. Wajsbürt, J.L. Desbarbieux, C. Spasevski, S. Penain, and A. Greiner. An integrated PCI component for IEEE 1355. In *European Multimedia Microprocessor*

Systems and Electronic Commerce Conference and Exhibition, Florence, Italy, November 1997.

[Whe97] L. Whetsel. An IEEE 1149.1 based test acces architecture for ics with embedded cores. In *IEEE International Test Conference (ITC)*, pages 69–78, Washington (DC), USA, November 1997.

[Whe99] Lee Whetsel. Addressable Test Ports : An Approach to Testing Embedded Cores. In *Proceedings IEEE International Test Conference (ITC)*, pages 1055–1064, Atlantic City, NJ, September 1999. IEEE Computer Society Press.

[WNC87] Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, vol. 30, No (6) : pp 520–540, June 1987.

[WR01] Lee Whetsel and Mike Ricchetti. Tapping into IEEE P1500 Domains. In *Digest of Papers of IEEE International Workshop on Testing Embedded Core-Based Systems (TECS)*, pages 3.2–1–7, Marina del Rey, CA, May 2001.

[ZBC97] Yervant Zorian, Dwayne Burek, and R. Chandramouli. A 2-Step Strategy tackles System-on-a-Chip Test. In *Digest of Papers of IEEE International Workshop on Testing Embedded Core-Based Systems (TECS)*, pages 3.2–1–5, Washington, DC, November 1997.

[ZM99] Yervant Zorian and Erik Jan Marinissen. Tutorial on Embedded-Core Based System Chips. In *Proceedings IEEE European Test Workshop (ETW)*, Constance, Germany, May 1999.

[ZMD98] Yervant Zorian, Erik Jan Marinissen, and Sujit Dey. Testing Embedded-Core Based System Chips. In *Proceedings IEEE International Test Conference (ITC)*, pages 130–143, Washington, DC, October 1998. IEEE Computer Society Press.

[Zor97] Yervant Zorian. Test Requirements for Embedded Core-Based Systems and IEEE P1500. In *Proceedings IEEE International Test Conference (ITC)*, pages 191–199, Washington, DC, November 1997. IEEE Computer Society Press.

Publications personnelles

Revues ou journaux

Mounir Benabdenbi, Walid Maroufi, Meryem Marzouki. *CAS-BUS : A Test Access Mechanism and a toolbox Environment for Core-based System Chip Testing.* Journal of Electronic Testing : Theory and Applications (Special issue on Plug-and-Play Test Automation for System on a Chip), Volume 18, Numéro 4, Aout 2002 (à paraitre).

Cet article devrait aussi paraître dans un livre dédié au Test des SoC, édité par Kluwer, au cours de l'été 2002.

Conférences internationales avec comité de lecture et actes

Mounir Benabdenbi, Walid Maroufi, Meryem Marzouki. *Testing TAPed Cores and Wrapped Cores With The Same Test Access Mechanism.* IEEE Design Automation and Test in Europe (DATE) Conference.Mars 2001. Munich, Allemagne.

Walid Maroufi, Mounir Benabdenbi, Meryem Marzouki. *Solving the I/O Bandwidth Problem in System on a Chip Testing.* XIII Symposium on Integrated Circuits and Systems Design (SBCCI). Septembre 2000.Manaus (AM), Brésil.

Mounir Benabdenbi, Walid Maroufi, Meryem Marzouki. *CAS-BUS : A Scalable and Reconfigurable Test Access Mechanism for Systems on a Chip.* IEEE Design Automation and Test in Europe (DATE) Conference. Mars 2000. Paris, France.

Colloques internationaux avec comité de lecture et actes

Walid Maroufi, Mounir Benabdenbi, Meryem Marzouki. *Controlling the CAS-BUS TAM with IEEE 1149.1 TAP : A Solution for Systems-On-a-Chip Testing.* 4th IEEE International Workshop on Testing Embedded Core-based Systems (TECS). May 2000. Montreal, Canada.

Colloques nationaux avec comité de lecture et actes

Mounir Benabdenbi. *Conception en vue du test de systèmes intégrés.* 3emes Journées Nationales du Réseau Doctoral de Microélectronique (JNRDM). Mai 2002. Montpellier, France.