THÈSE DE DOCTORAT DE L'UNIVERSITÉ PARIS VI

Spécialité Informatique

Présentée par Walid Maroufi

Pour obtenir le titre de

Docteur de l'Université Paris VI

Aide à la Conception de Systèmes Testables

Présentée le 2 juillet 1999, devant le jury composé de

M. Alain GREINER	Président
M. Christian LANDRAULT	Rapporteur
M. Joao Paulo TEIXEIRA	Rapporteur
M. Patrice DEROUX-DAUPHIN	Examinateur
Mme Meryem MARZOUKI	Examinateur

Remerciements

Je remercie monsieur le professeur Alain Greiner, directeur du département architecture au LIP6, pour m'avoir accueilli dans son laboratoire et avoir accepté de présider le jury de cette thèse.

Ensuite, j'adresse mes remerciements à madame Meryem Marzouki, chargée de recherche CNRS, qui m'a encadré dès mes premiers pas dans mes études et travaux de recherche post-universitaire, et dont les conseils et encouragements m'ont été très utiles dans la préparation de ce travail.

Je remercie aussi messieurs Christian Landrault, directeur de recherche CNRS au LIRMM (Montpellier), et Joao Paulo Teixeira, professeur et responsable de recherche à l'INESC (Lisbonne, Portugal), qui m'ont fait l'honneur d'être rapporteurs de cette thèse.

Je tiens également à remercier monsieur Patrice Deroux-Dauphin, président de TEMENTO SYSTEMS (Grenoble), qui a bien voulu accepter de juger ce travail en participant à mon jury de thèse.

Mes remerciements s'adressent aussi à tous les membres du département architecture du laboratoire LIP6, pour les marques de sympathie qu'ils n'ont cessé de me témoigner. Tout particulièrement, mes remerciements s'adressent aux membres du bureau 214 : Hakim, Mourad, Yann et Yannick ; grâce à leur sollicitude, j'ai pu travailler dans un excellent environnement humain et scientifique.

Je n'oublie pas que mes premiers travaux ont commencé à germer dans le laboratoire TIMA de l'INPG (Grenoble), dont je garde un très bon souvenir. Une pensée particulière à Maher et Hédi pour leurs conseils judicieux, leur gentillesse et leur humour.

À tous ceux qui m'ont aidé de près ou de loin, à un moment ou à un autre, dans la préparation de ce travail, je tiens à adresser mes remerciements les plus sincères.

À mes parents
À ma soeur
À mes grands mères
À mes beaux parents

À ma fiancée

Résumé

Malgré l'intérêt croissant pour les systèmes intégrés sur une seule puce, la testabilité et la maintenabilité des systèmes électroniques discrets revêt toujours une importance capitale dans des domaines comme l'avionique ou les transports terrestres. Néanmoins, les besoins industriels en matière d'outils d'aide à la conception de systèmes testables restent loin d'être satisfaits. Outre qu'ils doivent être performants, de tels outils doivent répondre à des besoins différents, afin d'accompagner toutes les étapes de la vie d'un système, de sa conception à sa maintenance en cours d'utilisation.

Des architectures, souvent standardisées, de conception en vue de la testabilité existent tant pour les circuits intégrés (BIST, SCAN) que pour les cartes (IEEE 1149.1) ou les systèmes (IEEE 1149.5). Les systèmes intégrés sont également pris en compte dans des travaux récents, tels que ceux du consortium VSIA ou du groupe de travail P1500 de l'IEEE. Ces techniques, protocoles, et architectures de test constituent les éléments sur lesquels peuvent se fonder des stratégies globales de testabilité pour les systèmes électroniques ou micro-électroniques.

Toutefois, la diversité des possibilités existantes n'est pas sans causer certains problèmes aux concepteurs qui, vue la complexité croissante des systèmes qu'ils ont à construire, se trouvent souvent confrontés à des choix en matière de stratégies de test pour répondre aux contraintes qu'ils se sont fixées. C'est précisément dans ce contexte que se place cette thèse, dans laquelle nous proposons une méthode d'aide à la conception de systèmes testables, qui vise à guider le concepteur pour l'inclusion de fonctionnalités de test dans son système, quelle que soit l'étape de développement de ce système.

L'objectif de cette méthode est double. Elle permet d'abord d'optimiser au maximum le temps de développement du test en étudiant le plus tôt possible la testabilité du système et en favorisant le rapprochement entre les concepteurs et les spécialistes du test. Le deuxième objectif est de développer une architecture de test capable d'assurer une maintenance ultérieure efficace et fiable.

Mise en oeuvre dans l'outil STA que nous avons développé, cette méthode adopte une approche ascendante, traitant tour à tour la testabilité des circuits, des cartes (ou MCMs), et enfin du système global. À chaque niveau, des stratégies de test sont proposées et des décisions sont prises de façon interactive entre l'outil et l'utilisateur. Une stratégie d'évaluation de la testabilité de cartes, de MCMs et éventuellement de systèmes intégrès est aussi proposée et intégrée dans la méthode. Des passerelles vers des outils de génération de structures de test, d'analyse de testabilité au niveau circuit, de synthèse en vue du test ou de synthèse tout court, assurent l'aspect modulaire de la méthode.

Mots-clé : Systèmes discrets, cartes, MCMs, testabilité, contrôle, évaluation.

Abstract

Despite the growing interest in integrated systems on a chip (SoCs), testability and maintainability of discrete electronic systems is still a critical issue in fields like avionics. However, industrial needs for tools dedicated to system design for testability are far from being satisfied.

Such tools must meet different needs in order to follow all the steps of a system life cycle, from its design to its life time maintenance.

Standardized test architectures exist for circuits (SCAN, BIST), for boards (IEEE-1149.1) as well for systems (IEEE-1149.5). Integrated systems are also taken into account in recent works, such as those of VSIA consortium and P1500 working group. These test techniques, protocols and architectures are the elements on which can be based global strategies for testability of electronic and micro-electronic systems.

With the growing complexity of the systems, the designers have to take into account various constraints, and the choices they have to make become difficult. This is actually the context of this research work. A new method that helps the design of testable systems is proposed. It is meant to drive the designer in the inclusion of test functionalities in all the steps of the system development.

This method has a twofold objective. It first enables to optimize the test development time by studying as soon as possible the system testability in the design process and by getting the designers and the test experts closer. The second objective is to develop a test architecture able to ensure an efficient and reliable further maintenance.

The proposed method has been embedded in the STA tool that we have developed. It uses a bottom-up approach processing. Starting by the circuits, the boards or MCMs, and then the system testability.

At each level, test strategies are proposed and decisions are made interactively between the user and the tool. A strategy for the evaluation of the boards, MCMs and eventually integrated systems testability, is also proposed and integrated in the method. Gateways to test structures generation tools, circuit level testability analysis, synthesis for testability or high level synthesis, makes the method modular.

keywords: Systems, boards, MCMs, testability.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction Générale

La conception des systèmes électroniques discrets demeure un problème d'actualité, malgré l'intérêt récent pour la conception des systèmes intégrés sur une seule puce (SoC: "System on a Chip"). En effet, dans certains domaines tels l'avionique, les télécommunications et la construction automobile, l'utilisation de tels systèmes est inévitable et les progrès en VLSI, y compris ceux accomplis dans le domaines des SoCs, ne peuvent qu'améliorer leurs performances et limiter les problèmes d'encombrement, mais en aucun cas remettre en question leur utilisation.

Si nous nous intéressons aux éléments qui constituent ces systèmes, à savoir les circuits, les cartes et les Modules multipuces (MCM: «Multi-Chip Module»), nous remarquons que la testabilité de chacun d'eux a déjà été largement étudiée, aux différentes étapes du cycle de vie du produit (développement, production et maintenance). Les techniques de test au niveau circuit comme le SCAN et le BIST, ainsi que les standards de la famille IEEE1149.* tel le 1149.1 pour le test des cartes et le 1149.5 pour le test de systèmes, représentent les fondements de toute stratégie de test. Autour de ces techniques et standards, des méthodes de développement (analyse, synthèse et conception) en vue du test, ainsi que des méthodes de génération de vecteurs de test, d'analyse de résultats et de diagnostic se sont multipliées et améliorées au fil du temps.

Dans le monde de l'industrie, on a rapidement compris l'importance du test à toutes les étapes de la vie d'un produit. Ainsi leur demande se fait de plus en plus pressante en ce qui concerne les systèmes de synthèse et de conception en vue du test pour le développement de systèmes testables, les méthodes de test et de diagnostic pour la validation (validation des prototypes), le test de production et les méthodes de diagnostic pour la maintenance.

Le travail effectué dans cette thèse s'inscrit globalement dans ce contexte, et plus précisément dans la coordination entre les différents éléments intervenants lors de la conception des systèmes électroniques discrets. Concrètement, nous présentons une méthode d'aide à la conception de systèmes testables, dont le rôle principal est de proposer des stratégies de test pour les différents modules du système, en tenant compte des caractéristiques de son architecture et des contraintes du concepteur. L'optimisation du temps global de conception en vue du test, et l'efficacité de l'architecture de test dans le cadre de la maintenance future du système, représentent les deux buts principaux de la méthode. Dans un souci de pragmatisme, il a été décidé que cette méthode sera pilotée par un outil (STA: "System Testability Assistant") qui lui est dédié, dont les caractéristiques principales sont sa modularité et son aspect interactif. Il est utile de signaler dès à présent que STA n'est ni un outil de synthèse en vue du test, ni un outil de conception en vue du test, sa tâche principale consistant à générer des stratégies de test dans le but d'aider le concepteur à inclure des fonctionnalités de test lors de la conception de son système. Le choix d'une stratégie ou d'une autre se fait de façon interactive entre l'utilisateur et l'outil. Des passerelles vers des outils de synthèse, de synthèse en vue du test et plus généralement de génération de structures de test sont prévues, concrétisant ainsi l'aspect modulaire de l'outil.

Le deuxième chapitre est consacré à la définition de la problématique à résoudre tout au long de cette thèse.

Dans le troisième chapitre, nous introduisons les concepts de base et l'état de l'art en matière de modélisation des systèmes en vue de leur spécification, de conception en vue du test aux niveaux: circuit, carte et système et d'analyse de testabilité.

Le quatrième chapitre représente le coeur de la thèse. Nous y présentons la méthode d'aide à la conception des systèmes testables que nous proposons, ainsi que les choix de modélisation des systèmes à traiter.

Lors du traitement d'un système, l'évaluation de la testabilité des cartes (ou MCM) qui le composent est souvent indispensable. Pour ce faire, une méthode d'analyse de testabilité est proposée et présentée dans le cinquième chapitre.

Le sixième chapitre est dédié à la présentation de l'outil STA qui pilote la méthode d'aide à la conception des systèmes testables proposée.

Dans la conclusion générale nous commentons la méthode présentée et nous exposons les perspectives futures de ce travail.

En annexe, nous présentons un exemple qui montre les principales transformations que peut subir l'architecture d'un système tout au long du traitement.

Chapitre 2

Problématique

Longtemps, la prise en compte du test lors du développement a été considérée comme préjudiciable au profit escompté pour un nouveau produit, notamment en terme de temps de mise sur le marché (TTM: "time-to-market"). Actuellement, la situation c'est complètement inversée, en effet la réticence des industriels envers la prise en compte du test lors du développement n'a cessé de décroître jusqu'au point que, souvent, le temps de développement du test arrive à dépasser le temps de développement du produit [Tur97]. En effet, il y a eu prise de conscience que le test joue précisément en faveur du TTM dans la mesure où il constitue le seul moyen de valider des prototypes et vérifier des produits dont la complexité ne cesse de s'accroître. La figure 2.1, qui représente le cycle de vie d'un produit, montre qu'une bonne stratégie de test permet d'accélérer et d'améliorer les tests de validation et de production et ainsi réduire le TTM.

Les bénéfices de la conception en vue du test des systèmes électroniques ne concernent pas les seuls constructeurs industriels, mais aussi les utilisateurs, puisque dans certains domaines, comme les télécommunications, l'avionique ou encore la construction automobile, l'utilisation de systèmes testables permet d'assurer une maintenance efficace et rapide. Il est inutile de rappeler ici l'importance de la maintenance dans ces domaines d'activité, où les plus petits problèmes de pannes peuvent causer un manque à gagner important durant l'indisponibilité de service.

Le test revêt donc une importance capitale autant pour le constructeur et vendeur du produit que pour son utilisateur. Tout le challenge consiste à limiter le coût du développement du test, tout en proposant des stratégies de test efficaces afin de garantir une bonne maintenance ultérieure du produit.

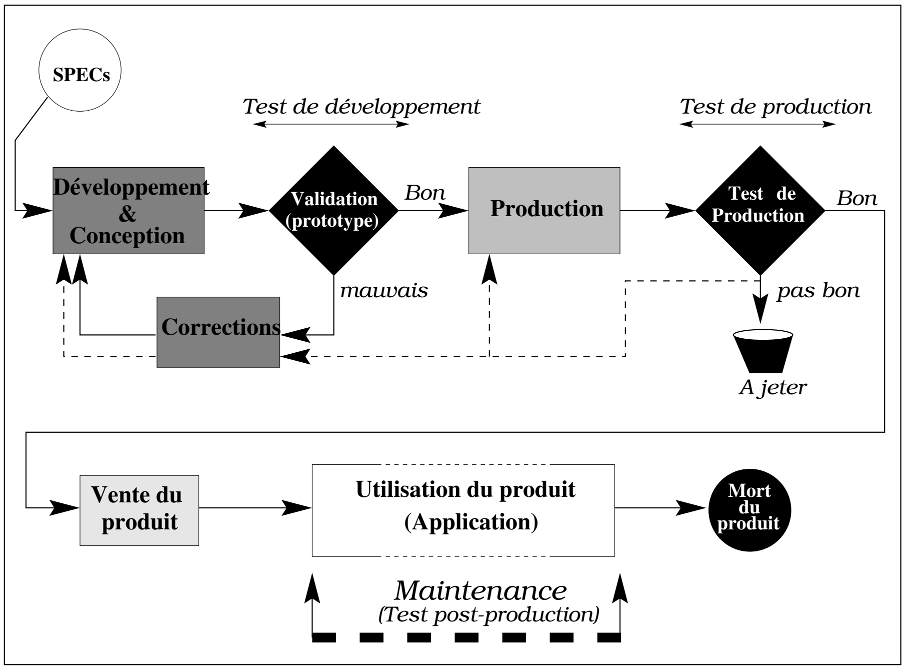

Fig. 2.1 – Cycle de vie d'un produit

Un des problèmes qui augmentent le TTM et alourdissent ainsi le coût du développement est la barrière (virtuelle) qui existe entre ceux qui développent le produit et ceux qui développent son test. En effet, les ingénieurs de test interviennent généralement sur des architectures déjà développées à un niveau assez bas ce qui implique deux inconvénients majeurs:

- Les aller et retour d'une communauté à l'autre induisent une augmentation des temps de développement, et donc de leur coût.
- Les performances de l'architecture de test ne peuvent pas être optimales puisque le test n'a pas été pris en compte assez tôt lors de la conception du produit. Les problèmes de performances surviennent notamment au niveau du surplus en surface dû au test.

Les solutions à ces deux inconvénients doivent cibler les différents niveaux hiérarchiques de conception, à savoir les circuits (y compris les SoCs), les cartes et les MCMs, et les systèmes. Pour les circuits, la solution principale réside dans l'utilisation d'outils de synthèse de haut niveau en vue du test, dont la vocation est de prendre en compte le test très tôt lors du développement du produit. La recherche dans ce domaine est encore jeune, mais nous pouvons imaginer que dans quelques années des produits tout à fait valables seront commercialisés et que leur utilisation sera largement diffusée dans le milieu industriel.

Pour les cartes et les systèmes, l'une des solutions est d'intervenir au niveau des spécifications du système, avant que le partitionnement matériel/logiciel soit effectué [TR95] [AHTR97] [LA97]. Cela dit, nous pensons que la solution idéale doit passer par une approche pragmatique globale visant à aider le concepteur à assurer la testabilité de son système. Cette approche doit être assez flexible pour pouvoir prendre en charge le système et ses différents éléments (circuits, cartes, MCMs, etc.) à n'importe quel niveau de développement. L'idée est de proposer au concepteur des solutions de test à intégrer dans le système, en tenant compte des caractéristiques de son architecture et de celles des éléments qui le constituent, le plus tôt possible lors des étapes de développement. C'est tout simplement la problématique à laquelle nous allons essayer de répondre tout au long de ce mémoire, particulièrement dans le chapitre 4 où une méthodologie d'aide à la conception des systèmes testables est proposée. Il est clair qu'une telle approche doit être pilotée par un outil qui lui est dédié. C'est exactement le rôle de l'outil STA que nous avons développé, et que nous présentons dans le chapitre 6. Les objectifs de STA sont l'aide à l'évaluation de la testabilité d'un système (matériel ou matériel/logiciel) en cours de conception, l'aide à l'évaluation de la cohérence de la testabilité de ses différents modules, ainsi que la proposition de solutions testables pour les modules du système qui le nécessitent.

Afin de garantir une maintenance ultérieure efficace, l'approche doit tenir compte de la testabilité post-production du système lors de la proposition des stratégies de test. Cela représente le deuxième volet de la problématique que nous essayons de résoudre.

Dans la littérature nous pouvons rencontrer les travaux présentés par Richard Sedmak [SE95] concernant une approche hiérarchique hautement automatisée pour la conception en vue du test des systèmes à base de processeur de traitement de signal. Cette approche, bien que répondant en partie à la problématique que nous nous posons, reste très liée au programme RASSP ("Rapid Prototyping of Application-Specific Signal Processors") développé par les laboratoires "Lockheed Martin's Advanced Technology", qui se propose d'améliorer d'un facteur quatre les paramètres temps de cycle, qualité et coût. En effet elle représente l'aspect conception en vue du test de ce programme et se base sur la notion de cycle de test: détection, isolation et correction, durant les différentes phases du cycle de vie du produit. Outre le fait quelle est plutôt dédiée au programme RASSP, cette approche essaie surtout de promouvoir une architecture de test BS/BIST ce qui limite à notre avis le choix en matière d'architecture de test surtout si on veut l'adapter à un type de systèmes discrets autre que celui ciblé par RASSP.

Chapitre 3

Concepts de base et état de l'art

3.1 Introduction

La problématique étudiée dans cette thèse concerne la prise en compte du test lors de la conception des systèmes électroniques discrets. Ce sujet touche différents domaines de recherche à la fois. En effet, dés le départ, trois questions importantes se posent:

- *Si on veut traiter la testabilité des systèmes discrets, alors comment les modéliser ?*
- *Quelles fonctionnalités de test faut-il rajouter dans les différentes cartes du système ?*
- *Comment évaluer et analyser la testabilités des cartes et donc du système ?*

Les réponses à ces trois questions mènent inévitablement à trois axes de recherche différents: la modélisation des systèmes électroniques discrets, la conception en vue de test et l'évaluation de la testabilité. Dans ce chapitre nous allons essayer de présenter ces différents domaines de recherche ainsi que les principales techniques, standards ou encore travaux qui y ont trait. Pour chacun des trois domaines, outre l'historique, un aperçu des travaux en cours sera présenté.

3.2 Méthodologies pour la spécification et/ou la conception des systèmes

Le traitement des systèmes électroniques, qu'il s'agisse de leur spécification ou de leur conception assistée par ordinateur (CAO), nécessite généralement qu'ils soient décrits selon des modèles qui tiennent compte de leurs caractéristiques architecturales, temporelles ou autres. Pour cela il existe dans la littérature plusieurs méthodologies de spécification et/ou de conception se basant sur des modèles de description en entrée et en sortie et qui expriment la façon de procéder pour spécifier et/ou concevoir un système. Des outils sont utilisés comme support pour décrire, analyser, valider et produire.

Dans ce paragraphe nous présentons succinctement la classification et l'historique des principales méthodologies de spécifications des systèmes et nous les comparons.

3.2.1 Classification des méthodes et historique

Dans la présentation chronologique qui va suivre, l'accent a été mis sur les modèles sur lesquels se basent les méthodes structurées de spécification ou de conception. Dans [Hau92] des études plus détaillées de ces dernières sont disponibles.

Méthode SADT ("Structured Analysis and Design Technique SADT")

Cette méthode [MM88] a été développée entre 1972 et 1977 par SOFTECH et se base sur l'utilisation du modèle SADT qui permet de décrire un système à travers l'enchaînement de ses activités (Actigrammes) et la transformation de ses données (Datagrammes) par un ensemble de "boîtes", chacune étant liée par quatre types de liens avec son environnement: Entrées, Sorties, Contrôles et mécanismes.

Le modèle se scinde en deux modèles: modèle Actigramme et modèle Datagramme. Dans chacun des deux modèles une boîte peut se raffiner selon un modèle SADT, ce qui implique l'aspect hiérarchique vertical. Le modèle SADT est l'outil typique d'aide à la spécification.

Méthode d'analyse structurée ("Structured Analysis SA")

Cette méthode [DPJ85] a été développée par DEMARCO et permet de décrire les spécifications d'un système à travers l'ensemble de ses activités et de ses données. Cette méthode admet pour modèle le diagramme de flot de données DFD. Chaque activité du diagramme peut se raffiner elle même selon un diagramme de flot de données favorisant ainsi l'aspect hiérarchique du modèle. D'un autre coté il ne permet ni le contrôle ni le séquencement temporel des actions, ce qui constitue une limitation.

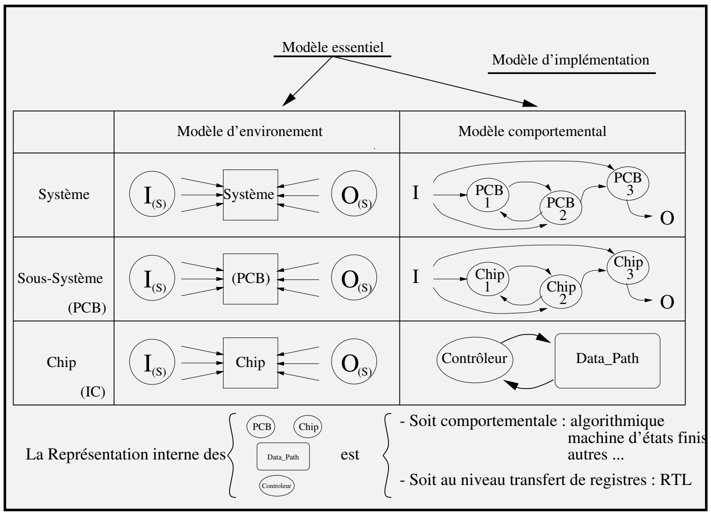

Fig. 3.1 – Modèle de Ward et Mellor pour la représentation des systèmes électroniques

Méthode de Ward et Mellor ("Structured Analysis-Real Time SART")

Les méthodes SART sont l'extension temps réel de la méthode d'analyse structurée SA de DEMARCO. Elles permettent d'aborder le problème de spécification et/ou conception des systèmes suivant deux axes cohérents: l'axe des données (méthode SA) et l'axe des contrôles (extension dite temps réel). Celle de Ward et Mellor a été développée par Paul Ward et Stephan Mellor en 1984-1985 [WM85] et aboutit à une analyse structurée de logiciels pour des applications temps réel. Cette méthode se base sur l'emploi de deux modèles:

- Le modèle essentiel: composé lui même de deux modèles à savoir le modèle d'environnement décrivant l'environnement dans lequel le système va opérer et le modèle comportemental décrivant le comportement du système.

- Le modèle d'implémentation dans lequel le système sera implémenté selon trois niveaux hiérarchiques: les processeurs, les tâches et les modules.

Les deux modèles se basent sur le diagramme de flot de données de DEMARCO auquel est ajouté un diagramme complémentaire exprimant le contrôle de l'évolution du DFD, on parle alors de diagramme de flot de contrôles DFC. Comme le DFD, le modèle essentiel de Ward et Mellor permet une approche progressive de la spécification par raffinement des activités ce qui donne au modèle son aspect hiérarchique.

La figure 3.1 permet de voir un exemple d'application du modèle essentiel de Ward et Mellor aux systèmes électroniques discret.

Méthode de Hatelay et Pirbhai (SART)

Cette méthode [PH87] a été développée par Imtiaz Pirbhai et Derek Hateley en 1986-1987, et est conceptuellement très proche de celle de Ward et Mellor. Elle repose sur un modèle constitué lui même de deux modèles:

- Le modèle de spécification: formé d'un DFD auquel est ajouté un DFC, de façon similaire au modèle de Ward et Mellor. La différence réside dans la notation des activités de contrôles et dans leur signification pour spécifier les états actifs et inactifs des activités de transformation. Tout comme pour le modèle de Ward et Mellor, la spécification est indépendante de l'implémentation.
- Le modèle d'architecture: pour l'implémentation.

Méthode de Lavi et Harel

Cette méthodologie se base sur trois vues de modélisation d'un système: activité, contrôle, implantation. le modèle ECS ("Embedded Computer System") sur lequel repose cette méthode décrit le système selon les trois vues:

- La spécification de l'architecture (Module-chart) décrivant les modules du système et son environnement.
- La spécification des activités (Activity-chart) proche de celle de Ward.
- La spécification du contrôle (State-chart) [Har87].

Les Méthodes orientées objet

Les méthodes orientées objet se basent principalement sur le concept d'encapsulation défini par Parnas puis Guttag en considérant toutes les ressources (données et modules) comme objets. Ceci rend le développement orienté objet typiquement ascendant [Boo94] [Hau92]. Un objet est caractérisé par:

- ses attributs ou variables rémanentes exprimant son état et donc son aspect statique.

- ses actions spécifiques ou méthodes qui sont les seules à pouvoir accéder aux attributs de l'état de l'objet, ils définissent le comportement de l'objet et donc son aspect dynamique.

Un ensemble d'objets qui partagent la même structure et un même comportement forme une classe. Plusieurs types d'associations entre classes peuvent avoir lieu selon des notions: d'association, d'héritage, d'agrégation, d'utilisation, d'instanciation et de métaclasse. Un modèle objet admet différents éléments qui forgent ses spécificités, ils sont de deux catégories: majeurs et mineurs:

- **Eléments majeurs**
  - `Abstraction:` Elle fait ressortir les caractéristiques essentielles d'un objet (qui le distinguent de tous les autres genres d'objets) et donc procure les frontières conceptuelles rigoureusement définies par rapport au point de vue de l'observateur.
  - `Encapsulation:` C'est le procédé de séparation des éléments d'une abstraction qui constituent sa structure et son comportement. Elle permet de dissocier l'interface contractuelle de la mise en oeuvre d'un objet.
  - `Modularité:` C'est la propriété d'un système qui a été décomposé en un ensembles de modules cohérents et faiblement couplés.
  - `Hiérarchie:` C'est un classement ou un ordonnancement des abstractions.
- **Eléments mineurs**
  - `Typage:` C'est le fait d'imposer la classe d'un objet de façon que des objets de types différents ne puissent pas être intervertis ou tout au plus qu'ils ne puissent l'être que de façon très restrictive.
  - `Simultanéité:` C'est la propriété de distinguer un objet actif d'un autre qui ne l'est pas.
  - `Persistance:` C'est la propriété d'un objet à travers laquelle son existence transcende le temps (i.e. l'objet continue d'exister après que son créateur cesse d'exister) ou l'espace (i.e. l'objet se déplace par rapport à l'espace mémoire où il a été créé).

Le modèle objet permet de décrire un système de façon hiérarchique très proche de la réalité. Plusieurs langages évolués permettent de développer ce modèle, notamment le C++ qui est une extension du C et qui introduit les notions du modèle objet dans sa syntaxe.

Comparaison des méthodes, modèles et classification

Toutes les méthodes et les modèles que nous avons présentés permettent de spécifier, décrire et/ou concevoir un système donné. Certains d'entre eux sont plutôt adaptés à la spécification, d'autres à la conception. La figure 3.2 montre une classification de ces méthodes et de certaines autres méthodes que nous n'avons pas présenté, selon leur meilleure adéquation à la spécification ou à la conception.

	Méthode	Concept
Spécifications	SADT (Ross)	Actigrammes/Datagrammes
	SA (DeMarco)	Diagrammes de flot de données DFD
	SART (Ward,Hateley)	DFD, DFD +DFC
	JSD (Jakson)	Diagramme de Jakson
	OOA (Coad)	Modèle orienté Objet
Conception	SD (Yourdon)	DFD et organigrammes structurées
	JSD (Jakson)	Structures de données et réseau de processus
	OOD (Booch)	Encapsulation objet
	OMT (Rumbaugh)	Modèle Objet

Fig. 3.2 – Classification des méthodes

Mis à part le modèle orienté objet tous les autres modèles admettent une approche hiérarchique descendante et sont guidés par la question: "*que doit faire le système?*". Alors que le modèle objet se caractérise par une approche hiérarchique ascendante et répond à la question: " de quoi est constitué le système? ".

Tous les modèles possèdent plus ou moins les trois qualités: abstraction, raffinement et lisibilité, ce qui les rend exploitables. Dans le registre de la lisibilité, nous pensons que le modèle orienté objet est le meilleur, en effet il permet de décrire un système d'une façon claire et facile et permet une souplesse certaine dans la manipulation des objets et des classes. Il reste toutefois à signaler qu'un modèle comme celui de Ward et Mellor permet lui aussi une description parfaitement claire du système grâce à son modèle essentiel.

3.3 La conception en vue du test (DFT: "Design For Testability")

Le but principal de la conception en vue du test est de rendre un système facilement testable et ce à différents niveaux: circuit, carte/MCM (Modules multi-puces) ou système (discret ou embarqué : SoC " system-on-a-chip "). L'idée est d'augmenter la contrôlabilité, l'observabilité et donc l'accessibilité des différents éléments qui le composent, en se basant sur des techniques et standards de test.

Dans cette section nous présentons la conception en vue de test à travers trois de ses aspects les plus importants à savoir: les techniques et les standards de test existants, la synthèse en vue du test et le contrôle du test.

3.3.1 Les techniques et standards de test

Au niveau circuit nous distinguons deux classes de techniques de test, à savoir les techniques "ad hoc" et les techniques structurées. Au niveau carte ou au niveau système, il s'agit de méthodes hiérarchique réutilisant la DFT circuit et se basant sur des standards de test.

La DFT au niveau circuit

Les techniques "ad hoc"

Ce sont des techniques post-conception, simples et peu coûteuses. Elles sont très liés aux circuits auxquels elles sont appliquées. Cela rend l'utilisation de chacune d'elles restreinte à un nombre limité de circuits qui peuvent l'implémenter. Dans [ABF90] et [Ben84] nous pouvons rencontrer certains exemples d'utilisation de telle techniques. Les techniques "ad hoc" les plus importantes sont :

- *Le partitionnement:* L'idée est d'établir un partitionnement de l'architecture du circuit en plusieurs blocs indépendants. Ceci permet de les tester sans qu'il y ait une influence des uns par rapport aux autres [WP83]. Cette technique prône la logique "diviser pour conquérir ".

- *Les points de test:* L'idée est d'ajouter des points de test pour améliorer la contrôlabilité et l'observabilité. L'ajout de points de test se fait par l'intermédiaire de multiplexeurs et permet de couper les boucles combinatoires dans les circuits.

Les techniques structurées

Vu la croissance fulgurante qu'a connue la densité d'intégration, l'utilisation des techniques "ad hoc" pour un test exhaustif n'était plus possible. Dès lors des techniques de test structurées ont vu le jour et ont été associées à des outils de générations de vecteurs de test et de simulation de fautes. Les deux techniques les plus importantes sont :

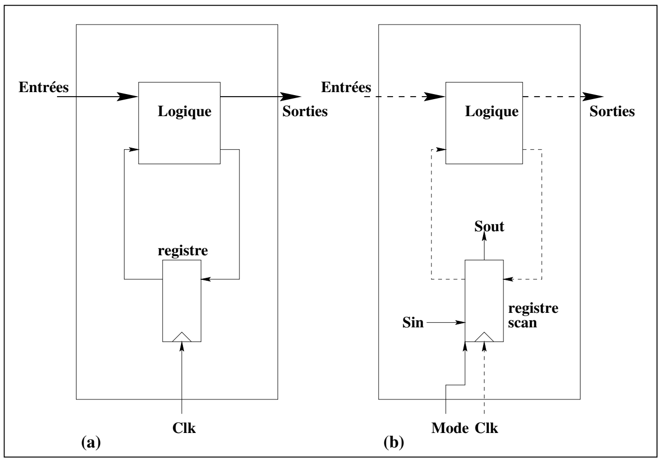

Fig. 3.3 – Exemple d'intégration d'un scan-path complet dans un circuit séquentiel

- **Le scan-path:** L'idée est de remplacer les éléments mémorisants du circuit par des éléments qui jouent le même rôle mais qui peuvent être chaînés les uns aux autres, afin de réaliser un registre à décalage accessible de l'extérieur du circuit (figure 3.3[a] le circuit avant l'inclusion du scan-path, figure 3.3[b] le circuit avec un scan-path complet). C'est à travers ce registre que les données de test seront transmises au circuit et que les résultats seront récupérés. La technique scan-path est utilisée essentiellement dans les circuits séquentiels, puisque le fait que les points mémorisant deviennent complètement contrôlables et observables permet en mode de test de régler le problème de leur initialisation et transformer ainsi (virtuellement) le circuit en circuit combinatoire. Nous distinguons entre autres deux façons de faire :

  - *La technique LSSD:* Elaborée à IBM, elle se base sur l'utilisation de latchs scannables [WE77].
  - *La technique de type NEC :* Le scan-path est réalisé à partir de bascules D [FWY78].

  Le scan-path peut être total, en chaînant tous les éléments mémorisants entre eux ou partiel, en chaînant seulement certains d'entre eux. Si la chaîne du scan-path complet est divisée en plusieurs chaînes, on dit que le scan-path est multiple. Le choix de ces chemins de scan se fait selon des critères de testabilité (voir paragraphe 3.3.4). Pour un circuit séquentiel, l'utilisation d'un scan-path partiel ne garantit pas son aspect combinatoire en mode test, ce qui implique l'utilisation de générateurs de vecteurs de test plus complexe (dit générateurs séquentiels). Toutefois un scan-path partiel reste moins coûteux en matériel qu'un scan-path total (plus de détails dans [ABF90]).

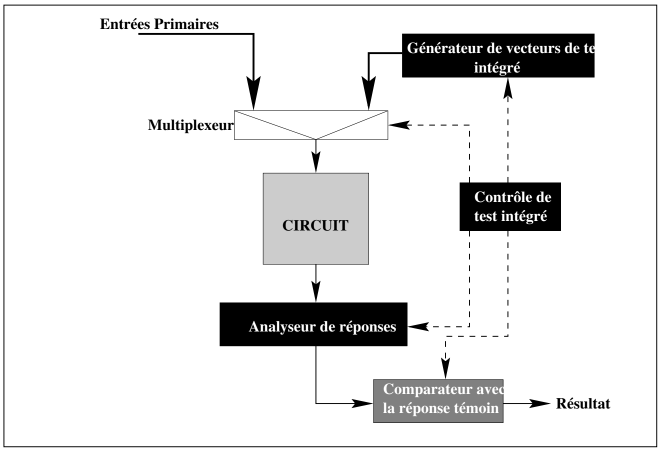

Fig. 3.4 – Architecture d'un test intégré (BIST)

- **Le test intégré BIST ("Built-In-Self-Test"):** Le principe est d'intégrer dans le circuit à la fois le générateur de vecteur de test, l'analyseur de résultat et le contrôle du test (figure 3.4). Cela a pour but d'éviter d'envoyer les vecteurs de test de l'extérieur et donc d'éviter l'utilisation de mémoires de test externes qui peuvent atteindre des tailles très importantes surtout pour les gros circuits. En général nous rencontrons deux types de BIST: le BIST pour mémoire qui nécessite généralement des algorithmes de test déterministes et le BIST pour des structures logiques tels les contrôleurs. Les éléments les plus importants d'une architecture BIST sont :

  - *Le générateur de vecteur de test TPG ("Test Pattern Generator"):* Il se base sur l'utilisation d'un LFSR " Linear Feedback Shift Registers " [BAMS85] qui peut être adaptée pour un test exhaustif [WM86], pour un test pseudo aléatoire [WCM87] ou encore pour un test pseudo exhaustif [McC84] [UM89].

  - *L'analyseur de réponse de sortie SA (" Signature Analyzer "):* L'idée est de compresser les résultats de test en une signature qui peut être facilement comparée à une signature témoin [ABF90]. Un analyseur de signature peut se baser sur l'utilisation d'un simple LFSR ou d'un MISR "Multi-Input Signature Register" qui, par rapport au premier, permet de réduire le coût en matériel relatif à la compression.

  – *Le registre BILBO ("Built-In Logic Bloc Observer"):* Ce registre inclut à la fois les deux aspects de génération de vecteurs et d'analyse de signature. C'est une idée de Koenemann et al [KMZ79] qui, en se basant sur le fait que les TPG et SA peuvent être réalisés par des LFSRs semblables, ont proposé ce registre à double fonction pour les remplacer (plus de détail dans [WP83]).

plus de détails concernant le BIST peuvent être trouvés dans [Ben94].

– **La technique d'utilisation conjointe SCAN-BIST** Les deux techniques SCAN et BIST présentent des aspects très complémentaires. En effet la première règle les problèmes émanant de l'aspect séquentiel des circuits alors que la deuxième se charge de ceux concernant la génération et la sauvegarde des vecteurs de test pour les gros blocs combinatoires et complexes. Ceci fait que ces deux techniques se combinent parfaitement ensemble, ce qui permet de réduire considérablement le coût du test et augmenter sa flexibilité.

Outre le scan-path et le BIST, d'autres techniques de test au niveau circuit peuvent être rencontrées, par exemple la technique de macro-test de PHILIPS [BDS90].

La DFT au niveau carte

Jusqu'à la fin des années 80, le test des cartes (PCB) reposait principalement sur des techniques de test au niveau circuit (telle le scan-path). Le problème principal était d'atteindre les circuits implantés sur la carte pour les tester. Ceci se faisait selon une technique dite technique du "lit à clous" qui consiste à utiliser une planche sur laquelle sont implantés des clous reliés à un testeur. La carte est ainsi placée sur la planche de telle façon que les clous coïncident avec les entrées/sorties des circuits. Cette technique présente trois défauts majeurs à savoir la complexité des testeurs (et donc leur coût), la difficulté de manipulation et les risques d'encombrement des cartes, et surtout le fait que pour tester une carte il faut la débrancher du système. Avec l'évolution technologique qu'à connu le domaine (augmentation du nombre de circuits sur la carte, montage en double face, réduction des échelles, etc.), ces défauts ont pris une ampleur de plus en plus grande et la technique est devenue obsolète.

Entre temps BULL a proposé une technique qui se base sur le principe de chaîner entre eux tous les registres de scan intégrés dans les circuits de la carte. Ceci permettait d'atteindre facilement les différents circuits pour les tester et dans certains cas, même le test d'interconnexions était possible. C'est en quelque sorte une extension du scan path au niveau carte.

Le problème fut finalement résolu avec l'avènement du standard Boundary Scan [Boa90] à la fin des années 80. Ce standard a été proposé par le groupe JTAG (Joint Test Action Group) qui a vu le jour en 1985 grâce notamment à une initiative de PHILIPS. La norme IEEE/ANSI 1149.1 connue sous le nom "Boundary Scan" est une collection de règles applicables au niveau circuit pour permettre un test aisé des cartes, afin d'améliorer leur production et surtout leur maintenance. En effet, l'importante évolution qu'a connue la conception des circuits VLSI a conduit à un besoin de standardisation des protocoles pour faciliter le test à différents niveaux de hiérarchie (circuit, carte, système, etc.). Les travaux effectués par le groupe JTAG ont abouti aux propositions suivantes:

- Faciliter le test et le diagnostic d'éléments numériques.
- Maîtriser et minimiser la complexité des testeurs de cartes.
- Réutiliser les tests fonctionnels des circuits et les dispositifs de test intégré pour le test in-situ.
- L'approche du test sériel qui minimise le nombre de broches de test dans un circuit.

Suite à ces propositions, le standard IEEE1149.1 a vu le jour. Il présente deux modes d'opérations majeurs à savoir :

- Le mode **`"Non invasive"`** dans lequel l'activité du circuit est complètement invisible pour le fonctionnement normal du circuit (ex: lecture des résultats de test).
- Le mode **`"Pin-Permission"`** durant lequel la logique interne du circuit est déconnectée du monde extérieur pour permettre son test.

Architecture générale d'un circuit Boundary Scan

L'architecture standard d'un circuit BS est représentée par la figure 3.5. Le bloc *`logique`* représente la logique interne qui réalise la fonction propre du circuit. Les entrées/sorties de ce bloc sont connectées aux broches du boîtier par l'intermédiaire de cellules Boundary Scan chaînées entre elles pour réaliser le registre Boundary Scan. Outre ce dernier, nous pouvons remarquer la présence de deux autres registres sériels, le registre de "BYPASS" et le registre d'instruction. Optionnellement nous pouvons rencontrer un registre d'identification dont le rôle est d'identifier le circuit lorsque, plus tard, les différents circuits BS d'une carte seront connectés les uns aux autres. Le standard autorise le concepteur à ajouter d'autres registres pour des applications spécifiques.

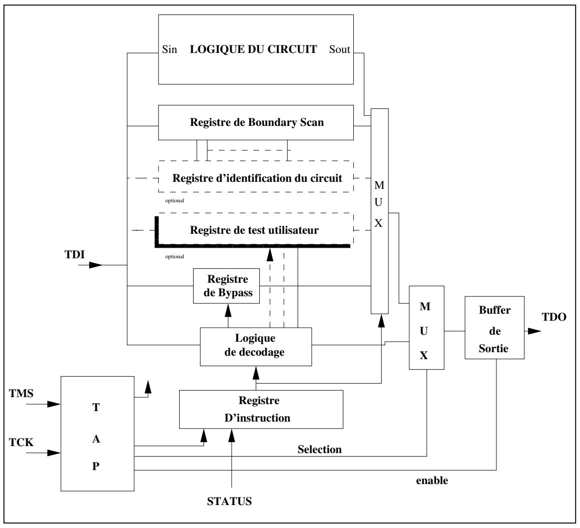

Fig. 3.5 – Architecture Boundary Scan

L'architecture BS présente un bus de test à quatre signaux, (optionnellement cinq) qui sont: l'horloge de test TCK, un signal de mode de test TMS, l'entrée de données de test TDI et la sortie des données de test TDO (le cinquième signal optionnel étant un signal de ré-initialisation TRST). Cette architecture est pilotée par un port d'accès de test TAP (Test Acces Port) qui reçoit les signaux du bus BS.

La cellule Boundary Scan:

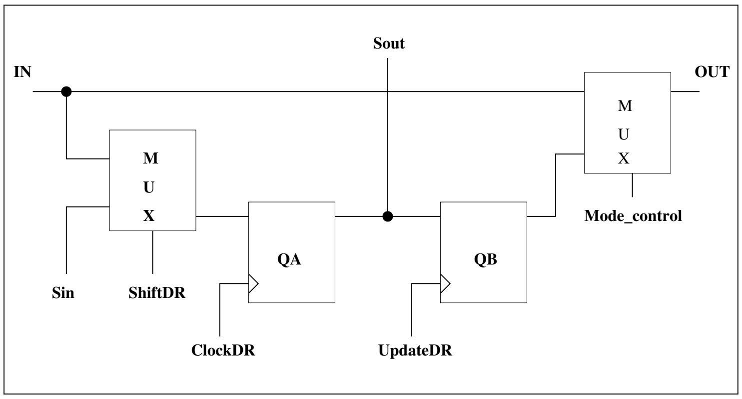

Fig. 3.6 – La Cellule Boundary Scan

La cellule BS présentée par la figure 3.6 admet quatre modes de fonctionnement:

- Le mode normal où **mode_control=0**, les données passent directement de **IN** à **OUT**. C'est le mode de fonctionnement normal du circuit.

- Le mode de SCAN où **`shift_DR=1`**, **`TCK`** est appliquée à **`clockDR`** afin de connecter les cellules BS les unes aux autres et construire ainsi le registre Boundary Scan. La première cellule du registre sera reçoit le signal TDI alors que la dernière fournit le signal TDO.

- Le mode de capture (ou encore échantillonage) où **`shift_DR=0`**, au coup d'horloge les données présentes sur l'entrée **`IN`** sont captées et donc disponibles sur les bascules **`QA`**.

- Le mode de mise à jour où **`mode_control=1`**, **`Update_DR`** applique les données captées ou décalées dans **`QA`** à la sortie **`OUT`**.

Cette cellule, étant unidirectionnelle, ne peut pas être affectée à des entrées/sorties bi-directionnelles. C'est pourquoi une autre architecture basée sur le même principe, mais plus complexe, est proposée dans ce cas.

`Le contrôleur TAP :`

Le contrôleur du bus de test TAP est une machine d'états finis à seize états. Comme entrées, cette machine reçoit le signal TMS et l'horloge TCK. Il est clair que le dia-

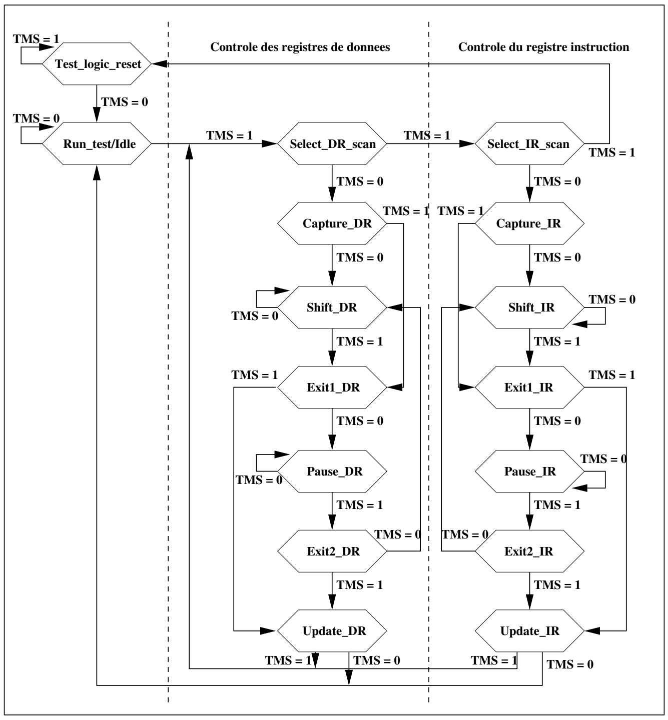

Fig. 3.7 – La machine d'état TAP

gramme d'état présenté par la figure 3.7 se charpente selon deux sous-diagrammes, l'un correspondant au contrôle du registre d'instruction et l'autre à celui du registre de données. L'état de la ligne TMS fixe la condition de transition d'un état à un autre au coup d'horloge.

L'état **`Test_Logic_Reset`** correspond au mode normal de fonctionnement du circuit. L'état **`Run_Test/Idle`** à certaines opérations de décalage, il permet également l'exécution de certains algorithmes de test itératifs tel le test intégré (BIST). Parmi les autres états, on note essentiellement: l'état **`Capture_DR`** (respectivement **`Capture_IR`**) dans lequel les données sont capturées en parallèle dans le registre de données (respectivement le registre d'instruction), l'état **`Shift_DR`** (respectivement **`Shift_IR`**) dans lequel le registre de données situé entre TDI et TDO (respectivement le registre d'instruction) est décalé d'une position et l'état **`Update_DR`** (respectivement **`Update_IR`**) pendant lequel le registre de données (respectivement le registre d'instruction) est chargé à partir du registre sériel correspondant.

Les registres :

- *Le registre d'instruction:* Ce registre comprend deux parties, une partie sérielle et une autre parallèle, ce qui lui permet de stocker une nouvelle instruction tout en maintenant la précédente. L'instruction en question spécifie les opérations à effectuer et le registre de données sélectionné. Il est à signaler que le standard prévoit des instructions obligatoires tel BYPASS, EXTEST et SAMPLE et des instructions simplement recommandées tel INTEST et RUNBIST [Par92].

- *Les registres de données:* Ils sont au minimum au nombre de deux: Le registre de **`BYPASS`** et le registre de **`Boundary Scan`**. D'autres registres optionnels peuvent être intégrés, comme le registre d'identification du circuit et du constructeur.

La chaîne Boundary Scan :

Les circuits intégrés pourvus de la norme IEEE1149.1 sont conçus pour être liés les uns aux autres et former ainsi une chaîne BS. Une chaîne simple BS est un ensemble de circuits Boundary Scan liés les uns aux autres et ayant en commun les deux lignes TCK et TMS (figure 3.8). Pour connecter un circuit à un autre il suffit de relier la sortie TDO de l'un à l'entrée TDI de l'autre. Fréquemment, certains circuits BS sont connectés à des circuits non BS, dans une telle configuration il est possible d'utiliser la chaîne BS de la carte pour le test des circuits non BS. On parle dans ce cas de "clusters BS" [MLT93].

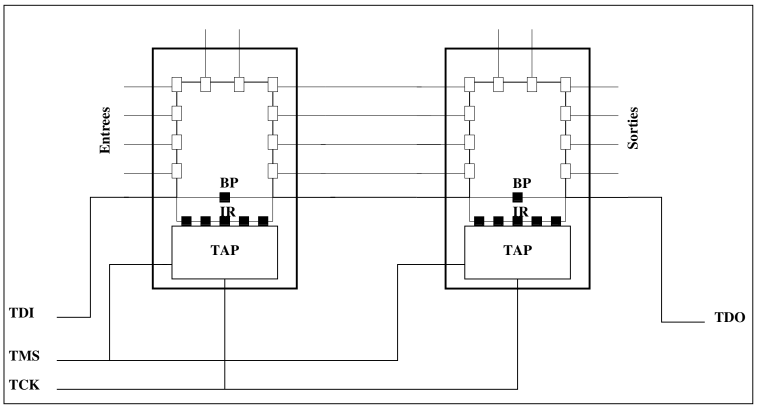

Fig. 3.8 – Chaîne simple Boundary Scan

Une propriété importante des chaînes simples est que les contrôleurs de bus (TAP) de tous les circuits sont à chaque instant dans le même état ce qui implique qu'un seul diagramme d'état d'un TAP est suffisant pour décrire le diagramme de toute la chaîne.

Plusieurs chaînes simples peuvent coexister sur une même carte où les signaux de leurs différents contrôleurs de bus sont complètement indépendants et non partagés. Dans ce cas les opérations de test des différentes chaînes doivent être cordonnées.

Le Test Boundary Scan

Dans ce paragraphe nous présentons les approches de test utilisant la norme IEEE1149.1.

Le Test Boundary Scan de base :

Le test BS de base réclame une utilisation rigoureuse du diagramme du contrôleur de bus (TAP) et des instructions BS. Le séquencement de test [Par92] représente les opérations de base du standard IEEE1149.1:

1- Initialisation de la logique dans l'état **Test_Logic_Reset**.

2- Chargement d'une instruction (**CAPTURE_IR, SHIFT_IR, UPDATE_IR**).

3- Lecture de la réponse aux stimuli précédents et application d'un vecteur de test (**CAPTURE_DR, SHIFT_DR, UPDATE_DR**).

4- Reboucler en 3- ou 2- jusqu'à la fin du test.

Les différents modes de test :

On note essentiellement trois modes de test qui sont: le mode de test interne, le mode de test externe et le mode d'échantillonage.

Le test interne a pour vocation de tester la logique interne des circuits supportant la norme IEEE1149.1. Le test externe lui s'intéresse aux interconnexions et éventuellement de la logique externe. Les données en E/S des circuits BS sont stockées à la volée lors du test d'échantillonage. Différents types de tests se rattachent aux trois précédents modes :

Le test interne d'un circuit : Comme son nom l'indique, il se réfère au mode de test interne et a donc pour but de tester la logique interne d'un circuit même s'il est implanté sur une carte. L'instruction correspondant à ce test est INTEST.

Concrètement, un échantillon de test est appliqué aux entrées logiques, le résultat est ensuite capturé et décalé à l'extérieur pour être comparé aux réponses témoins. La figure 3.9 montre la configuration BS des circuits de la chaîne durant ce test. Le registre BYPASS de tous les circuits non testés est utilisé pour isoler le circuit testé sur la carte.

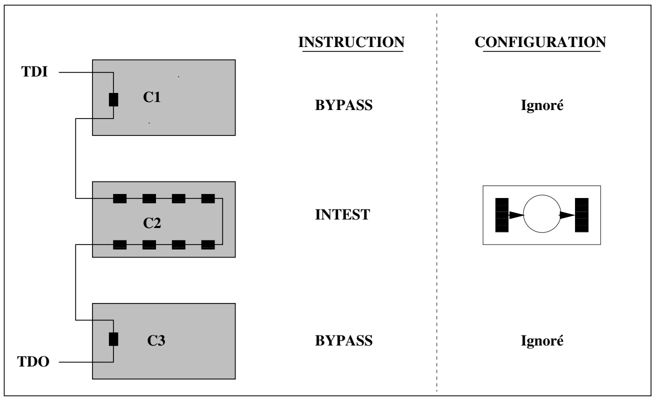

Fig. 3.9 – Test interne "INTEST"

Le test intégré : Certains circuits BS contiennent des dispositifs de test intégré (BIST) et dans ce cas ce genre de test est possible dans le mode de test interne. Une instruction optionnelle RUNBIST permet d'assurer ce genre de test. Il est à noter que ce test s'effectuera dans l'état **Run_Test/IDLE** .

Le test d'interconnexion : Ce test a pour but de tester les interconnexions entre des circuits supportant la norme IEEE1149.1, il se rattache au mode de test externe. L'instruction qui assure ce test est EXTEST. L'idée est d'appliquer un échantillon de test sur les sorties d'un circuit et de capter la réponse sur les entrées d'un autre, de manière à tester interconnexions entres ces deux circuits. La figure 3.10 représente la configuration BS des circuits de la chaîne au cours de ce test.

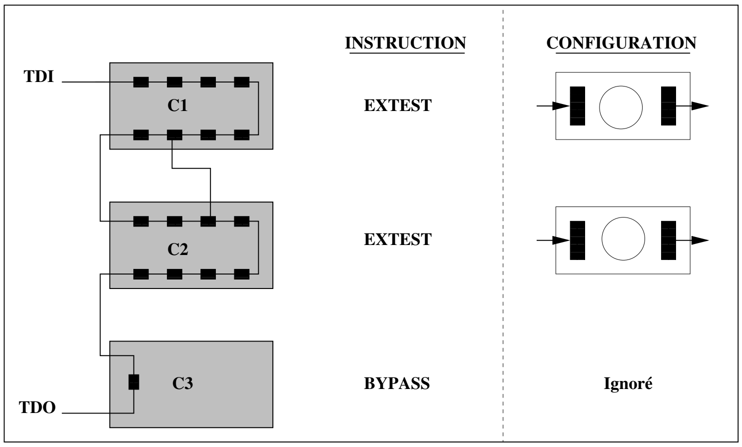

Fig. 3.10 – Test externe "EXTEST"

Le test d'integrité : Un test d'intégrité a pour but de déterminer si un circuit est abîmé, mal placé voire inexistant dans la chaîne. De même il peut vérifier si une connexion TDO/TDI entre deux circuits est détruite ou si l'un des signaux du TAP d'un circuit est mal connecté. L'idée se base sur l'écriture d'un couple de bits "01" dans tous

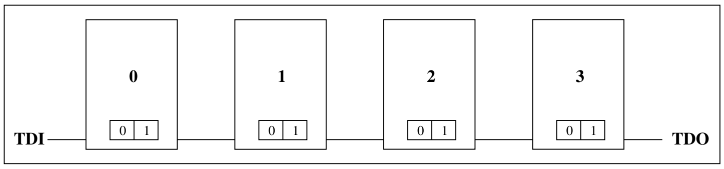

Fig. 3.11 – Test d'intégrité

les registres d'instructions des différents circuits de la chaîne (figure 3.11) et ensuite de récupérer au niveau du dernier TDO de la chaîne le contenu de ces registres. Ceci permet de vérifier s'il y a un défaut dans la chaîne.

Le test de connectivité : Ce test s'occupe de la connectivité des circuits sur la carte, par exemple une connexion circuit/slot. Ce genre de connexion ne peut pas être testé à l'aide d'un test externe classique. Le test de connectivité se rattache au mode d'échantillonage.

Le test d'interaction : Comme tous les circuits d'une carte n'implémentant pas forcément la norme IEEE1149.1, il peut y avoir des interactions entre des noeuds BS et des noeuds non BS. Ce test a pour but de créer des conditions pour que, si une interaction a lieu, elle soit détectable (ex : un court-circuit entre deux noeuds). C'est l'équipement automatique de test externe qui se charge de mettre en oeuvre ces conditions au niveau des vecteurs de test.

Travaux et applications rattachés à la norme

Pour simplifier la description des circuit intégrant la norme BS, un langage de description spécifique a été proposé. Il s'agit du BSDL "Boundary Scan Description Language" [PO90], un sous ensemble de VHDL dédié à la description des fonctionalités BS.

La norme BS est également applicable aux MCMs dont l'architecture s'apparente à celle des cartes. En ce sens, des travaux ont été élaborés et ont porté essentiellement sur les méthodologie pour le test des MCMs incluant des fonctionalités BS [HW92] [LMT94].

L'un des avantages de la norme est qu'elle se combine bien avec les autres techniques de test au niveau circuit. En effet, dans [Zor92] une méthode de test basée sur l'utilisation du BIST combiné au BS est proposée (instruction BS nécessaire : RUNBIST) pour le test des MCMs. De même dans [HK94] le cas d'un circuit BS intégrant en plus un chemin de scan est étudié (instruction BS nécessaire : SCANTEST).

Dans [MT91] les algorithmes de test BS les plus importants sont largement détaillés : EXTEST, INTEST, etc. Par ailleurs, dès l'avènement du standard, des travaux de modélisation de fautes et de diagnostic ont vu le jour [Jon91] [SHM91].

Dans [TP95] Tegethoff et Parker commentent le succès qu'a connu la norme et proposent des thèmes de recherche autour de la norme qui peuvent encore être étudiés pour mieux universaliser l'adoption de la norme dans la conception VLSI.

Enfin il faut noter que les concepteurs se heurtent en général au problème du coût du Boundary Scan, ce qui explique leur hésitation à l'intégrer dans leurs circuits. Toutefois, ils sont de plus en plus conscient des larges possibilités qu'offre cette norme, et son adoption devient de plus en plus fréquente (généralement combinée avec d'autres techniques de DFT au niveau circuit telle le SCAN et surtout le BIST).

Les buffers Boundary Scan

Dans le cas où un circuit est déja fabriqué sans intégrer des fonctionalités BS, il est encore possible de rendre son utilisation sur une carte compatible avec la norme BS. Cette solution consiste à utiliser des buffers (registres) BS qui contiennent les fonctionnalités BS standards et qui peuvent être connectés aux entrées/sorties primaires d'un circuit pour le rendre BScannable. Plusieurs produits industriels ont déja été commercialisés, citons entre autres le circuit SCAN182245A de chez Fairchild [FAI98] qui représente un buffer BS bidirectionnel (18 bits, neuf en entrée et neuf en sortie), et le circuit SN74BCT8244A de Texas Instruments, faisant partie de la famille des circuits SCOPE, il présente quatre ports de quatre bit chacun (deux en entrée et deux en sortie) [Ins98].

Plus de détails concernant la norme BS peuvent être retrouvés dans [Par90] [MT90]

[BO91] [MO96].

La DFT au niveau système

La DFT au niveau système se base principalement sur la réutilisation des techniques de test circuit selon une approche hiérarchique conforme à l'aspect classique d'un système électronique: circuit (ASIC, FPGA, etc.), carte (PCB ou MCM) et système.

Le but principal est que les informations de test (données et commandes) puissent se transmettre facilement et d'une façon flexible du niveau le plus haut (système) au niveau le plus bas (circuit) et vice versa, tout en essayant de limiter les surcoûts en temps et en matériel. Ceci passe inévitablement par l'ajout d'autres fonctionalités de test aux différents éléments du système tel les contrôleurs de test, et surtout par la définition de certaines règles, notamment celles devenues des standards comme le Boundary Scan au niveau carte (IEEE 1149.1) et le standard "Module Test And Maintenance Bus (MTM)" (IEEE 1149.5) au niveau système. Toujours au niveau système et dans [Bha91], une méthode d'extension du standard IEEE1149.1 au niveau système est proposée.

Le Scan Path au niveau système

Il s'agit tout simplement de l'extension de la technique du scan path au niveau système. En effet les registres de scan intégré de chaque circuit d'une carte sont reliés pour former un seul registre par carte (voir paragraphe 3.3.1). Ensuite, ces différents registres sont reliés à un contrôleur système global, qui leur fournit le même signal pour les données de test série SCANin et le même signal pour le mode de fonctionnement (test ou normal), et reçoit le signal SCANout du registre carte qu'il sélectionne (figure 3.12). Tous les signaux qui connectent les registres carte au contrôleur système constituent ainsi un bus de test.

Le BIST au niveau système

Actuellement, les travaux de recherche concernant les méthodes hiérarchiques pour le test des systèmes électroniques s'intéressent particulièrement à l'intégration du BIST au niveau circuit, carte et système. En effet, intégrer le BIST hiérarchiquement au niveau système permet d'éviter de transférer les vecteurs de test du niveau système au niveau circuit, ce qui est non seulement lourd à gérer, mais rend également le test très lent (inconvénient par exemple du scan path hiérarchique). Par ailleurs, le BIST hiérarchique permet de s'affranchir des problèmes que peuvent causer les mises à jour éventuelles du système à tester (par exemple le remplacement d'une carte par une autre), étant donné

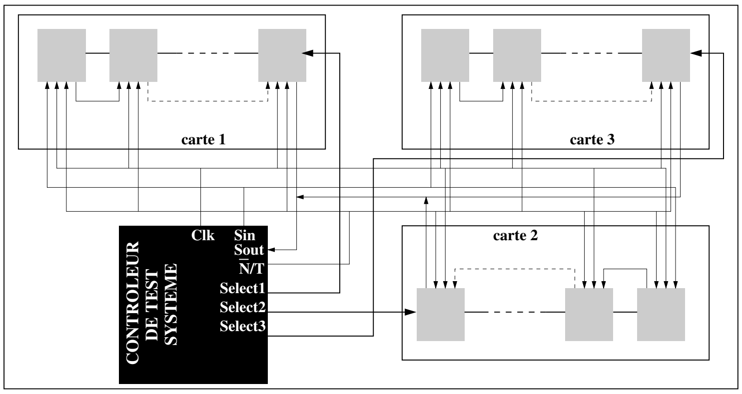

Fig. 3.12 – Architecture d'un scan path hiérarchique au niveau système

que les vecteurs de test sont générés à l'intérieur des sous éléments du système, et ne sont pas fixés dès le départ pour tout le système (comme c'est le cas par exemple pour le scan path hiérarchique).

L'idée générale est d'intégrer du BIST dans chaque circuit du système, et de raccorder ces circuits à un contrôleur de test qui donne l'ordre de lancer le test intégré. Tous les contrôleurs de test carte sont commandés par un contrôleur de test système. La figure 3.13 présente une architecture de BIST hiérarchique qui se base notamment sur l'adoption du standard BS IEEE1149.1 au niveau des cartes et du standard IEEE1149.5 au niveau système. Sur chaque carte, un contrôleur de test BS se charge de lancer le test intégré des circuits de sa carte (plus de détails sur les contrôleurs BS sont disponibles dans le paragraphe 3.3.3). Au niveau système, un contrôleur du bus MTM (IEEE1149.5) permet de piloter les différents contrôleurs BS carte.

Outre les deux avantages déjà cités de rapidité du test et de flexibilité, la technique du BIST hiérarchique au niveau système présente deux avantages très importants:

- La simplification des programmes de test et de la complexité des équipements automatiques de test.
- La simplicité relative des contrôleurs carte et système.

L'inconvénient principal du BIST au niveau circuit, à savoir le surcoût en surface, est complètement compensé au niveau système par la réduction de la complexité des contrôleurs et de la connectique dédiée au test. D'où une réduction globale du coût du test et une amélioration de sa qualité.

Plusieurs travaux de recherches et d'applications ont porté sur l'intégration du BIST hiérarchique dans les systèmes [HK94] [Mau93]. Souvent dans le domaine des télécommu-

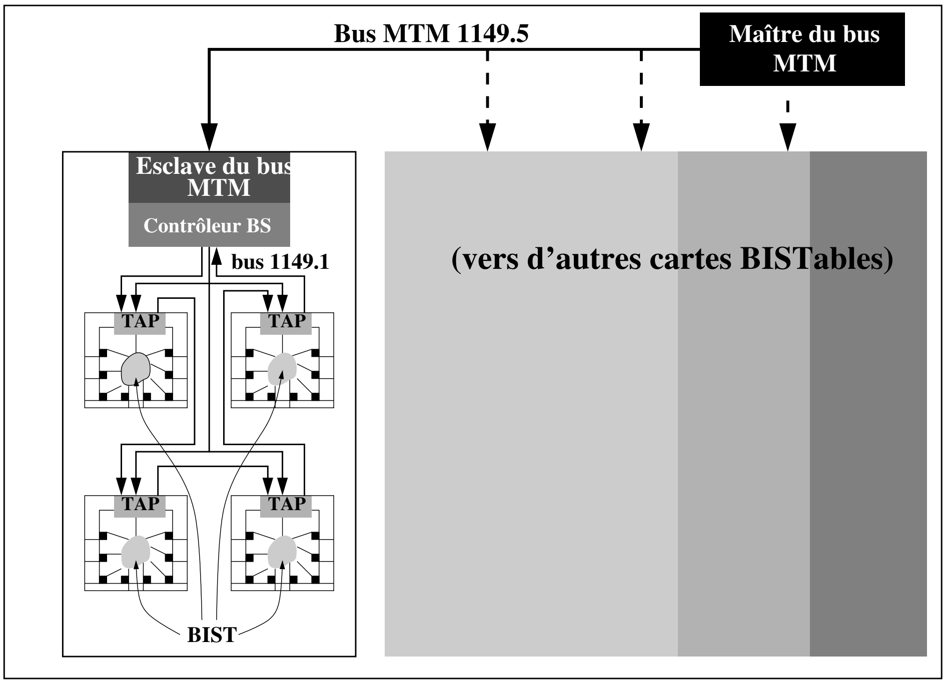

Fig. 3.13 – Architecture d'un test BIST hiérarchique"

nications, cette technique est utilisée pour détecter rapidement les cartes en panne et les remplacer. Dans [HSA95] une méthode qui adapte le logiciel de diagnostic de panne de la société canadienne NORTEL à une architecture de BIST hiérarchique est présentée. Cette méthode a déjà été implémentée dans plusieurs produits de NORTEL.

Enfin, une multitude de travaux ont portés sur les contrôleurs nécessaires à une telle architectures et aux deux niveaux carte et système. Ces contrôleurs sont le plus souvent compatibles aux bus BS IEEE1149.1 pour le niveau carte [JY91] [HK92] et MTM pour le niveau système [HK94].

Le standard MTM-bus IEEE1149.5

Le standard IEEE 1149.5 "Module Test and Maintenance (MTM) bus" définit un bus sériel synchrone de test et de maintenance capable d'intégrer dans un système un ensemble de modules testables. Un module peut être composé d'une carte ou d'un ensemble de cartes [Boa95]. Le but principal de ce bus est de minimiser le coût de l'architecture tout en gardant une bonne qualité de test et de maintenance.

Le standard définit le protocole de communication pour le transfert des commandes et des données de test et de maintenance entre le contrôleur système central et les différents modules à tester. Ce protocole se base sur un modèle de communication maître/esclave où le module maître peut communiquer avec 250 modules esclaves. Un des avantages du bus est que sa topologie "multidrop" (où les signaux sont communs à tous les modules) permet de retirer à tout moment n'importe quelle carte sans que ceci altère le fonctionnement global de l'architecture de test. En effet, l'adressage est conçu de sorte que le contrôleur central puisse communiquer avec un, plusieurs ou tous les modules du système à la fois. Les domaines d'application du standard sont très divers, notamment les télécommunications, l'avionique et la défense.

Structure et caractéristiques du bus

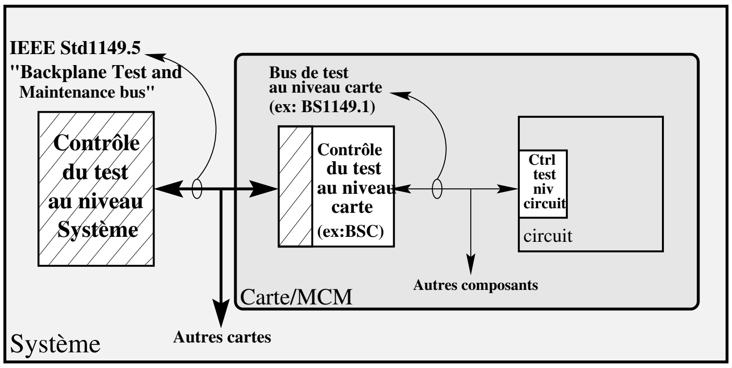

Fig. 3.14 – Hiérarchie des bus de test au niveau système

La figure 3.14 représente une vue hiérarchique des bus de test au niveau système. Le bus MTM IEEE1149.5 a été conçu de sorte qu'il puisse s'interfacer avec différents type de bus au niveau carte, entre autres le bus BS IEEE1149.1, selon une approche centralisée (tout le contrôle se fait au niveau système) ou distribuée (le contrôle est distribué au niveau des cartes).

La structure globale du bus IEEE1149.5 se compose de quatre parties.

– Les signaux: MMD, MSD, MPR, MCTL et MCLK.

– Le maître chargé de la communication entre le processeur de test et la transmission des informations de test aux modules sous test.

– Un ou plusieurs esclaves chargés d'assurer le dialogue avec le maître et de transmettre les informations de test aux éléments constituants les modules sous test, comme de recevoir ces informations en retour.

– Optionnellement, un module esclave-maître qui, en cas de défaillance du maître principal, peut prendre le relais et se comporter comme un maître.

Il est important de signaler que:

- La transmission sérielle des données est synchrone, et se fait par paquets de 17 bits (bit de parité inclus).

- Le bus est capable d'intégrer jusqu'à 250 esclaves.

Les transmissions en série sont synchronisées par un signal d'horloge (MCLK). Cette synchronisation permet de simplifier la conception des automates du maître et de l'esclave.

A un instant donné il ne peut y avoir qu'un seul maître qui contrôle les opérations de transfert sur le bus. Mise à part la transmission de l'entête (premier paquet envoyé par le maître lors d'une nouvelle transmission), tous les échanges sont bidirectionnels. En effet, à chaque fois que le maître envoie une donnée, l'esclave doit lui répondre, même si la réponse est constituée d'un paquet NUL.

Le module esclave-maître a initialement la fonction d'un esclave. Sa présence est indiquée dans le cas où le système doit être tolérant aux pannes. En effet, l'idée est que si une panne intervient au niveau du maître principal, le module esclave-maître prend le relais et assure la gestion du bus à sa place.

Chaque esclave possède son adresse propre codée sur 8 bits (256 adresses). Cela permet au maître d'adresser jusqu'à 250 esclaves, les six dernières adresses (251 - 256) sont réservées comme suit :

1. Une adresse "broadcast" qui permet d'adresser tous les esclaves à la fois.

2. Quatre adresses "multicast" pour créer des groupes d'esclaves.

3. Une adresse dite "amnésique", elle est nécessaire dans le cas où un module esclave n'arriverait pas à déterminer son adresse pour une raison quelconque.

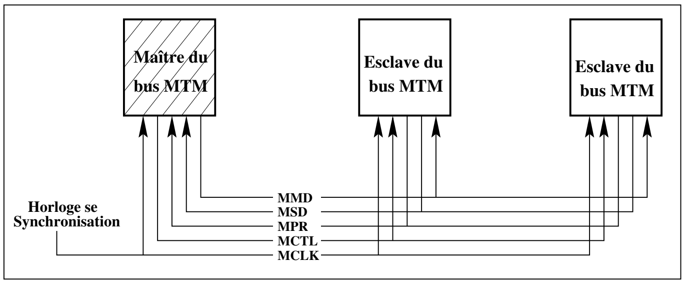

Fig. 3.15 – structure du bus MTM 1149.5

Le bus 1149.5 est constitué de cinq signaux dont quatre obligatoires (figure 3.15):

- MMD *"Master Data"*: Signal de transmission de données du maître.
- MSD *"Slave Data"*: Signal de transmission de données de l'eclave.
- MCTL *"Control"*: Signal de contrôle généré par le maître.
- MCLK *"Clock"*: Signal d'horloge commun au maître et aux esclaves.
- MPR *"Pause request"*: C'est un signal facultatif, mais hautement recommandé par le standard. Il permet aux esclaves de réclamer au maître une pause. Ceci est très utile lorsque l'esclave est lent, en effet, sans recourir au signal MPR, l'esclave en question risque de perdre certains paquets de données. Cette perte de paquets implique une retransmission de l'information, entraînant une forte augmentation de la durée totale de transmissions.

Le protocole 1149.5

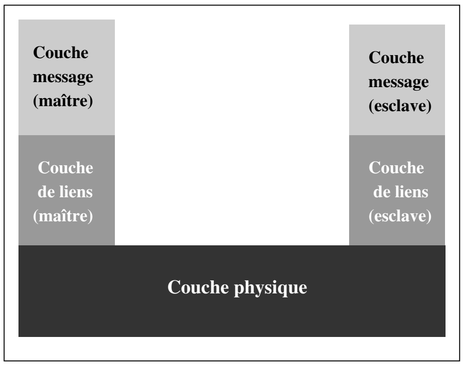

Fig. 3.16 – Les différentes couches du protocole 1149.5

Le standard définit un protocole pour le bus MTM (figure 3.16) qui repose sur trois couches:

- Une **couche physique** qui spécifie les caractéristiques physiques des interconnexions, bus comprit. Cela inclut les caractéristiques des signaux et les caractéristiques temporelles. Evidemment, les caractéristiques physiques relatives à une technologie donnée, celle qui a servi à la conception du système, ne sont pas disponibles dans le standard.
- Une **couche de "liens de données"** qui spécifie les caractéristiques du protocole permettant une transmission de paquets sans erreurs entre le maître et un ou

plusieurs esclaves. Cela inclut la sérialisation et l'empaquetage de l'information, la génération de parité et la reconnaissance d'adresse.

- Une **couche de "liens de message"** qui spécifie la syntaxe des messages et identifie les fonctions devant être réalisées par le maître ou par les esclaves.

Plus de détail, concernant ce protocole sont disponibles dans [Boa95].

Applications du standard IEEE1149.5

Vu les possibilités multiples qu'offre le bus IEEE1149.5 en matière d'intégration et d'implémentation, plusieurs applications au niveau système sont envisageables. Ces applications dépendent de la stratégie globale adoptée pour le test du système, qui peut être centralisée ou distribuée.

Une architecture de test centralisée se caractérise par un contrôleur de test central capable de gérer à lui seul le test des différents modules constituant le système. Ce type d'architectures présente deux avantages importants:

- le coût relativement faible de l'architecture, vu que les éléments d'interfaçage avec les modules du système sont très simples et ne contiennent aucune intelligence de test.

- La simplicité des éléments d'interfaçage.

Toutefois, l'architecture centralisée présente l'inconvénient majeur de la lenteur du test, puisque la parallélisation des tests des différents modules n'est pas possible. Dans [LHM92] un exemple de hiérarchisation des bus MTM IEEE1149.5 et BS IEEE1149.1 selon une approche centralisée est présenté.

Une architecture distribuée se caractérise quant à elle par un ensemble d'éléments d'interfaçage relativement intelligents en matière de test au niveau des modules constituant le système et d'un contrôleur central beaucoup plus simple. Cette architecture présente un certain nombre d'avantages:

- La rapidité du test, étant donné que la parallélisation du test des modules est possible.

- La possibilité d'intégrer des modules en provenance de différentes équipes de conception. Il s'agit là de l'un des objectifs fixés pour le standard.

- La possibilité d'échanger un module défaillant par un autre, ayant la même fonction mais de conception différente, sans pour autant changer les algorithmes ni les vecteurs de test.

– La parallélisation du test en tant que tel et tous les avantages qui en découlent.

Contrairement à une architecture centralisée, une architecture distribuée présente un coût relativement élevé, engendré par la complexité des éléments d'interfaçage qu'elle requiert au niveau des modules constituant le système. Toujours dans [LHM92], un deuxième exemple présente l'adoption du bus MTM IEEE1149.5 pour le test d'une architecture à base de microprocesseur, selon une approche distribuée.

Les travaux actuels utilisant le standard IEEE1149.5 comme solution pour le test système s'intéressent particulièrement à l'aspect contrôle du test. Cet aspect sera étudié ultérieurement dans le paragraphe 3.3.3.

3.3.2 La synthèse en vue du test

La synthèse en vue du test ("Synthesis For Testability" : SFT) peut être considérée comme l'automatisation de la conception en vue du test. En effet, il était concevable d'ajouter les fonctionalités une synthèse logique effectuée, mais actuellement, vu les contraintes de coût et de temps de mise sur le marché qui en résultent, il est nécessaire de tenir compte du test le plus tôt possible lors des étapes de la conception et notamment lors des étapes de synthèse. A partir de ce constat, plusieurs travaux ont été initiés et ont porté essentiellement sur l'inclusion de structures de test aux niveaux logique et RTL. Ces travaux ont déjà abouti à des outils performants de synthèse en vue du test, dont certains sont commercialisés. La recherche en la a ensuite été étendue au niveau comportemental et même au niveau spécification de système, en procédant par modification des algorithmes de synthèse en vue de la testabilité. Dans ce paragraphe nous présentons succinctement la SFT aux différents niveaux hiérarchique et les outils associés.

Les étapes de la SFT

Dans [Ait95], Aitken définit les quatre étapes essentielles de la SFT, que ce soit au niveau logique ou RTL:

– **La sélection de la méthodologie de test:** Elle concerne la sélection des règles et des contraintes de conception ainsi que les structures de test à inclure dans l'élément à synthétiser.

– **L'insertion des structures de test:** C'est une étape essentielle de la SFT qui consiste à définir quelle structure ajouter à une description, selon son niveau hiérarchique, pendant les étapes de conception (logique ou RTL).

- **La vérification:** Cette étape consiste à vérifier si le circuit ainsi que ses fonctionalités de test fonctionneront comme prévu après conception.

- **La préparation du programme de test:** C'est le programme de test nécessaire lors des étapes de fabrication pour assurer la bonne qualité du produit.

La synthèse en vue du test se fait généralement sous certaines contraintes de testabilité (contrôlabilité, observabilité, accessibilité etc.) [Cox95].

La SFT au niveau logique

Les premiers travaux dans le domaine de la SFT ont tous débuté au niveau logique et plusieurs outils ont été développés et commercialisés. Ces outils transforment la description du circuit au niveau portes logiques, afin d'y introduire les structures de test requises. Il s'agit généralement de structures permettant le scan path, total ou partiel, basées sur l'utilisation de bascules D multiplexées ou de "latches" scannables (méthode LSSD). Ces outils sont généralement des modules complétant des outils de synthèse logique. L'insertion de structures de test est un processus post-synthèse logique. La figure

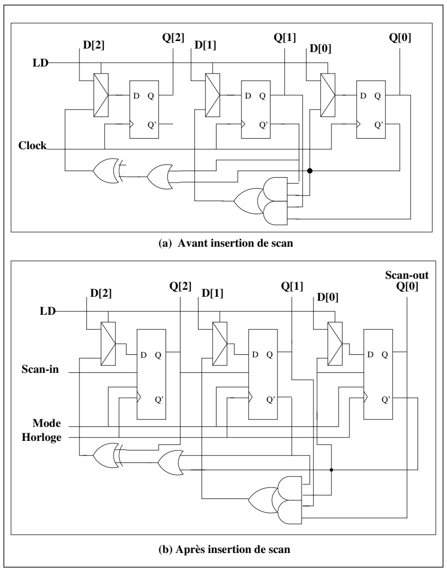

Fig. 3.17 – Insertion de scan au niveau logique

3.17 [Ait95] montre un compteur décrit au niveau logique, avant (a) et après (b) insertion de structures de test suivant la méthodologie de scan path total. Parmi les outils commerciaux de SFT au niveau logique nous pouvons citer:

- L'outil **"Test Compiler"** de Synopsys, qui permet d'insérer du scan path total ou partiel, d'ajouter des structures BS et de créer et formater les vecteurs de test [Syn97].

- L'outil **"DFT Advisor"** de Mentor Graphics qui commence par analyser la testabilité du circuit. Il propose une variété de méthodes de scan partiel et d'identification et de synthèse de point de test [Men97a].

- L'outil **"Test Synthesis Solution"** d'IBM, qui effectue la synthèse de scan path total ou partiel [Ibm98].

La SFT au niveau RTL

Toujours dans le but de prendre en compte le test le plus tôt possible lors des étapes de synthèse, des travaux ont été initiés et ont abouti à des outils de SFT au niveau RTL. Ces outils synthétisent le test pendant l'étape synthèse RTL du circuit ou transforment une description RTL en une description de même niveau mais testable. Globalement l'idée est d'insérer le code RTL des structures de test dans le code du circuit et de synthétiser le tout en tenant compte des contraintes de conception, notamment celles relatives au test. L'étape de vérification revêt dans le cas de la SFT RTL une importance capitale, étant donné qu'il nécessaire de vérifier si, après synthèse du code RTL, la logique de test est conforme aux attentes et si elle fonctionne normalement avec l'architecture de base du circuit. Les structures de test qui sont intégrées à ce niveau de la synthèse sont généralement du BIST, du BIST pour mémoires et du BS. Toutefois, il est toujours possible de décrire du scan path au niveau RTL. Dans [Ait95], un exemple d'inclusion de structures de test dans le code RTL d'un compteur est présenté.

Les outils de SFT RTL commerciaux sont plutôt récents, parmi les plus connus nous pouvons citer:

- L'outil **"LBIST Architect"** de Mentor Graphics [Men97b] qui insère automatiquement au niveau du code RTL des structures de BIST synthétisables.

- L'outil **"MBIST Architect"** de la même société, spécialisé dans la synthèse de BIST pour mémoires intégrées dans des SoC ("System On Chip") [Men97c].

- L'outil **"Test Synthesis Solution"** d'IBM (synthèse de Boundary Scan et de BIST) [Ibm98].

- L'outil **"LogicBIST"** de LogicVision qui automatiquement des modules logiciels intégrés ("embedded software cores") synthétisables au niveau RTL [Log98b]. Ces modules intègrent des structures de BIST (génération, contrôle et récolte de résultats) ou de Boundary Scan.

- L'outil **"MemBIST"** de la même société, spécialisé dans la génération automatique de mémoires intégrées (SRAMs, DRAMs et ROMs) BISTables sous forme de code RTL synthétisable [Log98c].

Combinés aux outils de synthèse classique, tous ces outils représentent une solution de synthèse en vue du test au niveau RTL.

Toutefois, de tels outils ne permettent pas de considérer la testabilité globale, du circuit. Des travaux de recherche, en cours de développement au laboratoire permettent de pallier ces inconvénients. Il s'agit du système IDAT, que nous présentons dans [BMM99], et qui traite des architectures virtuelles enrichies d'un certain nombre de caractéristiques, qui lui donnent un aspect intermédiaire, entre les niveau RTL et comportemental. Ce système traite la testabilité globale de l'architecture, en intervenant au niveau de l'allocation de ressources testables, et en tenant compte des résultats du module d'analyse de testabilité qui lui est associé. De plus, contrairement aux outils cités ci-dessus, IDAT ne requiert pas toutes les informations de contrôle et de synchronisation pour fonctionner. C'est pourquoi il peut être utilisé avant génération du code VHDL RTL.

La synthèse de haut niveau en vue de la testabilité (HL-SFT: "High Level Synthesis for Testability")

Le principe de la synthèse de haut niveau en vue de la testabilité est la génération automatique d'architectures testables à partir d'une description comportementale du circuit. Cette synthèse se fait selon la méthodologie de test choisie et selon des contraintes de coût et de performances. Il est utile de rappeler que toutes les méthodologies de synthèse de haut niveau qui ont été proposées s'articulent en général autour de deux étapes principales l'ordonnancement (ordre des opérations) et l'allocation (allocation de ressources) [CW91] [GDWL92]. Toutes ces méthodologies ont en général un but commun qui est l'optimisation à la fois du délai et de la surface. La HL-SFT se fixe un troisième objectif, qui est la testabilité de l'architecture générée après synthèse.

Concrètement, la prise en compte du test se fait principalement pendant l'étape d'allocation, on parle alors d'allocation en vue de la testabilité. La testabilité du système n'est pas négligée pour autant durant l'étape d'ordonnancement. En effet, la corrélation entre les deux étapes de synthèse est tellement importante qu'il est impossible d'imaginer une allocation en vue de la testabilité sans ordonnancement en vue de la testabilité. Cette corrélation est largement étudiée dans [Lee97] où les règles de synthèse en vue du test sont présentées. Ces règles sont au nombre de quatre dont les trois premières concernent l'allocation et la quatrième (basée sur les trois premières) concerne l'ordonnancement:

- règle1: Allouer, chaque fois que possible, un registre à au moins une entrée primaire ou une sortie primaire.
- règle2: Réduire la profondeur de séquentialité entre un registre d'entrée et un registre de sortie.
- règle3: Réduire le nombre de boucles séquentielles.
- règle4: Ordonnancer les opérations pour permettre l'application des trois premières règles.

A notre connaissance, il n'existe pas encore d'outils de HL-SFT. Toutefois plusieurs travaux lancés dans des laboratoires de recherches universitaires ont abouti à des développements intéressants. Les prototypes ainsi proposés adoptent des stratégies fondées sur la synthèse de BIST ou de scan. On peut citer parmi ces travaux:

- le système SYNTEST [HPCN92] de l'université `Case Western Reserve` qui adopte une méthodologie de BIST pendant l'étape d'allocation tout en considérant que l'ordonnancement a été déjà fait.
- la technique RALLOC [Avr91] de l'université de `Stanford` pour l'allocation de registres en vue d'une approche BIST.
- le système PHITS [Lee97] de l'université de `Princeton` pour la réduction de nombre de boucles selon une approche de scan partiel (PHITS-PS) ou de "non scan" (PHITS-NS).
- Les travaux effectués au laboratoire LIRMM de Montpellier, concernant la synthèse de haut niveau en vue du test, permettant d'améliorer la testabilité globale de l'architecture [FPR98] [FPRV98].

Spécification système en vue de la testabilité

Dans certains travaux, l'étude de la testabilité démarre au niveau des spécifications au niveau système. Ainsi, dans [VSSvR94], Vranken présente une méthode de spécification système en vue du test, dédiée aux systèmes matériels/logiciels. Elle consiste à partitionner ces systèmes en différents modules testables et à insérer dans leur spécification des fonctionnalités de test complètement indépendantes du mode d'implémentation. Ces travaux se situent dans la continuité de l'approche Macro Test de Philips.

Par ailleurs, Al-Hayek, Le Traon et Robach étudient dans [AHTR97] l'importance des critères de coût du test lors du partitionnement logiciel/matériel des systèmes. Un algorithme d'évaluation du coût de test est proposé, qui tient compte à la fois des modules matériels et logiciels.

3.3.3 Le contrôle du test

Le contrôle du test représente le troisième volet important de la DFT après les techniques/standards de test et la synthèse en vue du test. Il se manifeste aux trois niveaux: circuit, carte/MCM et système. Néanmoins, une différence de concept entre le contrôle du test au niveau circuit et celui au niveau carte ou système est à signaler. En effet, au niveau système ou carte, le contrôle du test se base essentiellement sur la gestion de bus standardisés (tels les bus IEEE1149.1 et IEEE1149.5) ou non, alors qu'au niveau circuit on ne peut pas vraiment parler de bus de test interne.

Le contrôle du test au niveau circuit

Etant donné que les techniques de DFT au niveau circuit nécessitent en général un contrôle externe du test (tel est le cas pour contrôler un scan path ou pour déclencher un test BIST), la partie contrôle de test réellement intégrée dans les circuits est souvent très simple voire inexistante. Par exemple, dans le cas d'un scan path total ou partiel, il n'y a pratiquement pas de contrôle de test intégré dans le circuit, tout le contrôle se faisant de l'extérieur, par l'intermédiaire de contrôleurs au niveau de la carte ou du système intégrant le circuit. Quand le circuit intègre des fonctionnalités BIST, un contrôleur très simple est intégré à son tour dans l'architecture du circuit, afin de gérer les opérations de test (passer en mode test, fixer le nombre de vecteurs de test, initialiser les registres, déclencher le processus de BIST, etc.). Dans [KAB86] et [BGC88], l'implémentation de tels contrôleurs est présentée. L'une des entreprises qui présente le plus de produits adoptant une approche BIST est la société américaine "LogicVision", qui propose avec son outil **icBIST** [Log98a] trois contrôleurs de test au niveau circuit (code RTL) pour le contrôle de la logique BISTable ("logicBIST Controller"), pour le contrôle de mémoires SRAM, DRAM et ROM BISTables ("memBIST Controller") et pour la gestion de toute structure BIST à travers le bus BS IEEE1149.1 ("tapBIST (1149.1) Controller").

Le contrôle du test au niveau carte

Le contrôle du test au niveau carte reposait il y a une vingtaine d'années sur l'utilisation de testeurs fonctionnels externes qui, pour assurer un bon taux de couverture, se heurtaient à des problèmes de coût exorbitants étant donné leur complexité.

Actuellement le contrôle du test au niveau carte est étroitement lié à l'architecture du bus BS IEEE1149.1. En effet, depuis l'avènement de la norme, plusieurs travaux ont été élaborés et ont abouti à un ensemble de gestionnaires plus ou moins intelligents du bus BS. Ces gestionnaires peuvent varier de la simple interface parallèle/série et série/parallèle au module complexe capable de générer lui-même les vecteurs de test et d'analyser les résultat pour un test "go/no-go".

Commandés à partir de programmes logiciels implémentant les algorithmes BS, ces gestionnaires permettent, via les TAPs des circuits BScannables de la carte, d'exécuter les test BS classiques tels le test d'interconnexion ou d'intégrité. D'autres programmes plus avancés, se basant sur l'utilisation des instructions BS facultatives telles RUNBIST ou SCANTEST, peuvent faire appel à ces gestionnaires pour contrôler le test des structures de DFT autres que le BS à l'intérieur des circuits.

Dans [LB89], Lien et Breuer présentent un contrôleur de test et de maintenance pour modules et cartes se basant sur l'utilisation du bus BS IEEE1149.1 pour le test de cartes et capable de gérer le contrôle de plusieurs bus à la fois.

Jarwala et Yau ont été aussi parmi les premiers à avoir proposé un tel gestionnaire [YJ90]. Dans [JY91], ces auteurs présentent l'architecture de leur gestionnaire et montrent sa capacité à gérer le test BIST au niveau carte.

Peng et Hâkegard proposent dans [HP97] une approche de synthèse automatique de contrôleurs de test pour cartes intégrant des fonctionnalités BIST et adoptant la norme BS.

Dans [KB92] une architecture de contrôleur reconfigurable est proposée. Ce contrôleur est capable de gérer entre autres le bus IEEE1149.1 et donc le test au niveau carte.

De leur coté, les industriels proposent généralement avec leurs systèmes intégrant des fonctionalités BS, des contrôleurs pour le bus IEEE1149.1. Deux cas sont possibles, ou bien sur chaque carte BS est implanté un contrôleur, ou bien un seul contrôleur central implanté sur l'une des carte, se charge de tester toutes les autres (figure 3.18). Parmi ces contrôleurs, nous pouvons citer:

- Le contrôleur BS **`ASC16`** de "GOEPEL electronics" pour bus PC/AT qui dispose de 16 MO de mémoire scan et qui peut atteindre une fréquence maximale de 8MHz

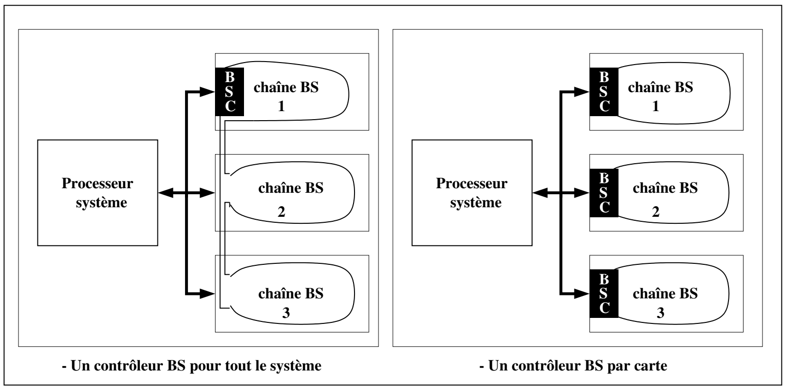

Fig. 3.18 – Les deux cas d'architectures de contrôle pour cartes BS

[Goe99a].

- Les contrôleurs BS **`PSC1149.1-(x)`**, de la même société, mais pour bus PCI [Goe99b].
- Le contrôleur BS intégré **`SCANPSC100F`** de "FAIRCHILD semiconductor" [FAI99]. Doté d'un ATPG interne et d'un analyseur de signature, il peut contrôler jusqu'à deux bus BS à la fois.

Le contrôle du test au niveau système

Avant l'avènement du standard IEEE1149.5, le contrôle du test au niveau système consistait à gérer les fonctionnalités de test au niveau des circuits et des cartes (BS) selon des méthodes ad hoc. En effet, ces méthodes différaient d'une application à l'autre et étaient inspirées des structures de test déjà existantes dans le système. L'une de ces méthodes plus ou moins structurées est celle présentée dans [Bha91], qui se base sur une extension du standard BS IEEE1149.1 au niveau système. Cette méthode a été présentée alors que le projet du standard IEEE1149.5 était déjà bien avancé, mais elle se voulait plus simple et moins onéreuse. Actuellement, alors que le standard IEEE1149.5 est de plus en plus utilisé dans la maintenance et le test des systèmes électroniques, le contrôle du test au niveau système revient à contrôler le bus MTM défini par le standard (voir paragraphe 3.3.1). Rappelons que ce bus se base sur un modèle de communication maître/esclave et qu'il est amené à s'interfacer avec d'autres bus de test tel le bus BS IEEE1149.1. En tenant compte de ces propriétés, il n'est pas possible d'imaginer un contrôleur monobloc pour ce bus comme cela était le cas avec le contrôleur Boundary Scan. La figure 3.19 donne une idée de l'architecture de contrôle d'un tel bus, où l'on note la présence d'une partie dite *"Contrôle Maître"* incluant le maître et toute l'intelligence de test additionnelle que le

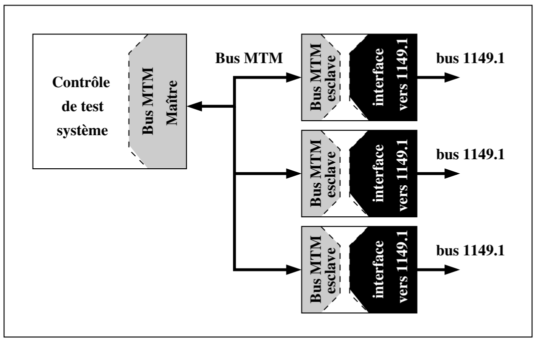

Fig. 3.19 – Contrôle du test au niveau système

concepteur veut intégrer en matériel selon l'application et d'une partie dite `"Interface Esclave"` incluant l'esclave et le bloc d'interfaçage avec un autre bus de test présent dans chaque module testable du système.

Dans [KB92] un contrôleur maître reconfigurable, valable pour plusieurs bus de test à la fois, offre la possibilité du contrôle du bus MTM. Ce contrôleur est constitué d'un bloc de base dit bloc 1149.n qui selon la configuration, se connecte à un sous-bloc dédié à un bus donné (IEEE1149.1, IEEE1149.5, IEEE1149.2 et un bloc interface analogique). Un ensemble de registres de contrôle et d'état, une interface qui, avec le bus VME/SCSI et un générateur d'horloge complètent l'architecture du contrôleur ainsi configuré, est équivalent au contrôleur maître de la figure 3.19.

L'interfaçage entre le bus IEEE1149.5 et le bus IEEE1149.1 est un cas très fréquent dans les architectures hiérarchiques de test. Cela implique la définition de l'architecture du bloc d'interfaçage incluant l'esclave du bus MTM et la partie interface de contrôle du bus BS. Dans [LHM92] une telle architecture est proposée, elle a la particularité de pouvoir s'interfacer à plusieurs bus BS (synchrones entre eux) à la fois.

3.3.4 Analyse de la testabilité

L'analyse de testabilité se base principalement sur l'évaluation de certaines métriques qui obéissent à certaines règles. Son but principal est de guider l'insertion de structures de test pendant la conception, généralement lors des étapes de synthèse. Elle peut être utile à d'autres niveaux, notamment lors de la génération des vecteurs de test (optimisation des vecteurs et de leur coût). Par ailleurs, cette analyse peut être quantitative ou qualitative selon l'approche adoptée et les besoins requis.

La plupart des méthodes d'analyse de testabilité s'intéressent principalement aux circuits, aux différents niveaux hiérarchique: portes logiques, RTL et comportemental. Certains travaux ont porté sur l'analyse de testabilité des cartes et des systèmes, mais ils sont beaucoup moins nombreux que ceux dédiés aux circuits.

Les deux métriques de testabilité les plus importantes sont la contrôlabilité et l'observabilité qui peuvent être définies de la façon suivante:

- La **contrôlabilité** est une grandeur quantitative ou qualitative qui représente la difficulté à initialiser un signal interne du module sous test à une valeur déterminée à partir de ses entrées primaires.
- L'**observabilité** est une grandeur quantitative ou qualitative qui représente la difficulté à observer un signal interne du module sous test à partir de ses sorties primaires.

Presque toutes les autres métriques de testabilité que nous pouvons rencontrer sont fonction de ces deux métriques. L'une des métriques les plus populaires est la **testabilité** dont le calcul diffère d'une approche à l'autre.

L'**accessibilité** est une métrique qui représente la difficulté d'accès à un noeud interne donné du module sous test. Dans le cas d'approches quantitatives, l'accessibilité est souvent calculée de la façon suivante:

Accessibilité = Contrôlabilité + Observabilité

Les premières approches, généralement quantitatives, d'analyse de testabilité au niveau circuit ont vu le jour au début des années 70 [Bre78], se situent toutes au niveau portes logiques. Ces approches ont largement influencé SCOAP [Gol79] [GT80], qui est l'algorithme d'analyse de testabilité le plus utilisé au niveau porte, et qui a servi plus tard comme base pour différents systèmes d'analyse. SCOAP définit deux classes de métriques:

- Les métriques d'ordre combinatoire, elles concernent les noeuds combinatoires du circuit et sont au nombre de trois: la contrôlabilité combinatoire à 0 (CC0), la contrôlabilité combinatoire à 1 (CC1) et l'observabilité combinatoire (CO).
- Les métriques d'ordre séquentiel, elles concernent les noeuds séquentiels du circuit et sont au nombre de trois: la contrôlabilité séquentielle à 0 (SC0), la contrôlabilité séquentielle à 1 (SC1) et l'observabilité séquentielle (SO).

Nous retrouvons ces six métriques dans tous les systèmes d'analyse de testabilité au niveau circuit qui ont suivi SCOAP, tels par exemple CAMELOT [BMR80]. Le calcul de ces métriques pour tous les noeuds du circuit se fait par propagation des contrôlabilités des entrées (supposées complètement contrôlables) vers les sorties primaires et par propagation des observabilités des sorties (supposées complètement observables) vers les entrées primaires.

Au niveau RTL, les approches d'analyse de testabilité décrites ci-dessus ne peuvent pas être appliquées pour diverses raisons, dont la principale est le fait que les signaux au niveau RTL sont du type multi-bit (de 0 à n) et non plus des mono-bit (0 ou 1) comme c'est le cas au niveau logique. Appliquer le même concept que SCOAP reviendrait à calculer pour chaque signal la contrôlabilité à chaque valeur qu'il peut prendre (soit 2^n contrôlabilités différentes). Cela est évidemment est très complexe à mettre en oeuvre et présente un problème de déchiffrage de résultats. D'autres algorithmes, adaptés au niveau RTL, ont donc été proposés, tels TMEAS [Gra79] qui a la particularité d'être aussi applicable au niveau portes, et l'algorithme proposé par Gu dans [Gu96]. Ce dernier se rapproche de SCOAP avec l'idée de classes de métriques (combinatoire et séquentielle), mais évidemment fait abstraction des valeurs que peuvent prendre les signaux.

L'analyse de testabilité ne s'arrête pas seulement au niveau RTL, en effet des algorithmes d'analyse de testabilité au niveau comportemental ont déjà été proposés tels BETA "Behavioral Testability Analysis" présenté par Chen dans [CWS91]. Ces algorithmes adoptent généralement une approche qualitative, basée sur une classification des registres en éléments contrôlables, non contrôlables, observables et non observables.

A la différence des travaux d'analyse de testabilité au niveau circuit, ceux dédiés aux cartes sont très peu nombreux. Un des rares algorithmes est celui proposé dans [PLL97], qui adopte une approche qualitative pour l'analyse de testabilité de cartes ou de MCMs.

Dans [TR97], Le Traon et Robach présentent un algorithme d'analyse de testabilité de systèmes logiciels/matériels applicable très tôt au niveau spécification hiérarchique de flot de données. Le but est d'utiliser cet algorithme pour prendre en compte le test très tôt lors de la conception d'un système.

Dans [DTT98], une méthode pour l'évaluation de la qualité d'architectures de systèmes matériel/logiciel est présentée par l'INESC au Portugal. Se basant sur des critères et des métriques de qualité, cette méthode met l'accent sur les deux aspects de diagnosabilité et de testabilité.

3.4 Conclusion

Dans ce chapitre nous avons présenté, le plus souvent selon leurs évolutions chronologiques, les différents travaux de recherche ayant un lien avec la conception de systèmes discrets testables. Cette étude bibliographique représente une base de références pour la méthodologie d'aide à la conception des systèmes testables que nous proposons dans le chapitre 4.

Ces travaux tournent autour de trois domaines principaux:

- Les méthodologies de spécification et/ou de conception des systèmes.
- La conception en vue du test (DFT) aux trois niveaux circuit, carte et systéme. La DFT est détaillée selon ses trois aspects les plus importants:
  - Les techniques et standards de test: principalement, le SCAN et BIST au niveau circuit, le standard BS IEEE1149.1 au niveau carte et les méthodes hiérarchiques ainsi que le standard IEEE1149.5 au niveau système.
  - La synthèse en vue du test (SFT): présentée surtout à travers les outils déja existants (de recherche ou commercialisés).
  - Le contrôle du test aux niveaux circuit, carte et système.
- L'analyse de testabilité surtout au niveau circuit (selon le type de description: logique, RTL ou comportemental) où la recherche est la plus avancée.

Chapitre 4

Méthodologie proposée : Aide à la conception de systèmes testables

4.1 Introduction

Ce chapitre présente la méthodologie que nous proposons pour l'aide à la conception des systèmes testables. Cette méthode s'intéresse aux systèmes discrets matériels ou matériel/logiciel en cours de conception. Les composantes essentielles de ces systèmes sont les cartes (PCB) et les circuits sur ces cartes. Les cartes du système peuvent aussi contenir des modules multipuces (MCM) qui, durant toutes les étapes de cette méthodologie, seront assimilés à des cartes, vu les similitudes architecturales qui existent entre les deux.

Les éléments du système peuvent être décrits à différents niveaux d'abstraction. Les circuits peuvent être décrits au niveau comportemental, structurel (RTL) ou logique; il est possible de considérer des circuits existants ou des puces préconçues pour les MCMs. Cela confère à notre méthode un aspect plutôt pragmatique, à la différence des méthodes classiques de conception de systèmes testables qui démarrent à partir des spécifications du système avant que le partitionnement logiciel/matériel ne soit effectué, comme nous l'avons déjà évoqué dans la problématique.

4.2 Description de la méthodologie

Etant donné la complexité croissante des systèmes matériel/logiciel, leur testabilité est un problème de plus en plus complexe à résoudre. Par conséquent les techniques de synthèse en vue de la testabilité doivent être en mesure de résoudre tous les problèmes que peut poser la testabilité des systèmes, pour permettre une maintenance ultérieure flexible et efficace. Actuellement, l'idée est d'introduire les fonctionnalités de test au plus tôt, lors des étapes de conception des systèmes, mais le choix des fonctionnalités les mieux adaptées à chaque élément du système demeure un problème à résoudre. C'est à cette question que nous allons tenter de répondre à travers la méthodologie que nous proposons.

L'idée générale est d'adopter un traitement ascendant de la testabilité du système, partant du composant le plus petit, à savoir le circuit, pour aboutir au système lui-même. Il est important de noter que ce que nous proposons n'est pas une méthode de synthèse traditionnelle, mais une façon de guider le concepteur à introduire du test dans son système de manière cohérente, dans un souci d'efficacité et d'économie.

A chaque étape du traitement, il est très important d'évaluer la testabilité de l'élément traité, que ce soit un circuit, un MCM ou une carte. Cette évaluation permet de juger l'état de testabilité de l'élément et de détecter où il faut ajouter des fonctionnalités de test; elle fournit donc les informations nécessaires au noyau central de traitement pour qu'il puisse, en les combinant avec d'autres (telles les contraintes imposées par l'utilisateur et les spécifications du système lui-même), proposer au concepteur les stratégies adéquates à adopter pour introduire du test dans le système. Cette évaluation représente le premier principe sur lequel se base notre méthodologie.

La volonté des concepteurs d'introduire du test dans leur système se heurte en général à des contraintes en matière de surface et donc de coût. Trop souvent, le test a été sacrifié très tôt pendant la conception parce qu'un compromis n'a pas été trouvé. Dans notre cas nous considérons qu'un tel compromis ne peut être atteint qu'après avoir consulté le concepteur plusieurs fois et à différentes étapes de la conception. Il peut accepter ou refuser les propositions qui lui sont faites, et reste le seul maître de la procédure. Cet aspect interactif représente le second principe sur lequel se base notre méthodologie.

Le module principal, responsable de la proposition des stratégies de test et de la communication avec le concepteur, est en relation avec plusieurs de sous-modules d'évaluation, de génération de structures de test, de génération de contrôleurs de test, de synthèse, etc. Ceci confère à la méthode un aspect modulaire qui représente le troisième principe sur lequel elle se base.

La réutilisation ("Re-use") des ressources est le quatrième et dernier principe de la méthode que nous présentons. Elle se manifeste à deux niveaux:

– La réutilisation de structures de test relatives à certains modules particuliers pour le test d'autres modules (exemple la "clusterisation" dans le cas des cartes Boundary Scan (paragraphe 3.3.1)).

– La réutilisation de contrôleurs de test prédéfinis pour circuits ou cartes (existants ou sous forme de description structurelle).

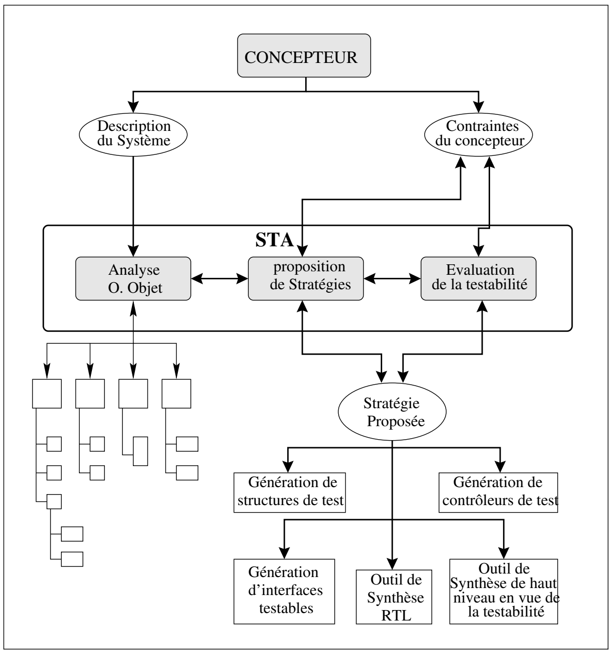

Fig. 4.1 – Vue fonctionnelle de la méthode

la figure 4.1 représente la vue fonctionnelle de la méthode que nous proposons. Le concepteur introduit la description de son système qui est analysée selon un modèle orienté objet, dans le but de créer la structure de donnée objet sur laquelle sera effectué le traitement. Le modèle orienté objet que nous avons adopté sera présenté dans la section 4.3. Le module principal de proposition de stratégies de test prend en compte la structure de donnée du système, les contraintes imposées par le concepteur et les résultats que lui fournit le module d'évaluation. Une fois la stratégie proposée acceptée par le concepteur, des passerelles permettent d'atteindre des modules de génération de structures de test, de génération de contrôleurs de test, de synthèse, etc. Ces modules ont pour rôle d'appliquer la stratégie acceptée sur la structure de données.

L'organigramme principal du traitement que subira chaque système est représenté par la figure 4.2. Une fois la description du système introduite et analysée selon le modèle objet, le traitement proprement dit commence au niveau des circuits. Carte par carte, la testabilité des circuits est évaluée et, le cas échéant, l'insertion de structures de test (pour circuit non testable) ou de complément de structures de test (pour circuit partiellement testable) est proposée. Une fois le traitement de tous les circuits d'une carte terminé, une pré-étude de la testabilité de la carte commence. Ce traitement a pour but d'effectuer une première évaluation de la testabilité de la carte et de détecter tous les éléments de test qu'elle contient: circuits testables, circuits testables Boundary Scan, contrôleurs Boundary Scan ou autre, etc.

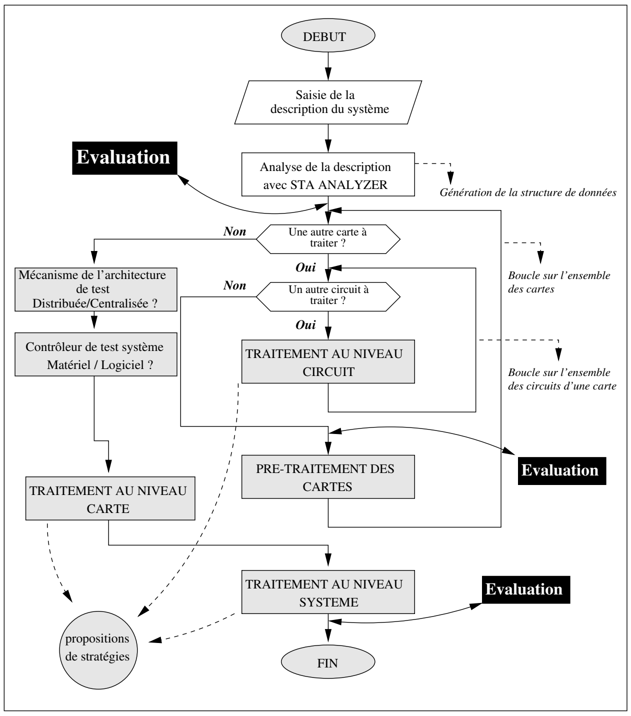

Fig. 4.2 – Organigramme principal de la méthode

Une fois les pré-études de toutes les cartes réalisées, deux traitements importants ont lieu. Le premier concerne la prise de décision sur la nature de l'architecture globale de test à adopter: distribuée ou centralisée. Dans le cas d'une architecture distribuée, l'intelligence de test, en ce qui concerne le contrôle, est repoussée dans les cartes où des contrôleurs plus ou moins autonomes sont capables de dérouler à eux seuls les algorithmes permettant de tester les circuits de la carte. Un contrôleur système minimal (par opposition à intelligent) permet de transmettre les commandes et les données entre les contrôleurs carte et le processeur de test externe. Dans le cas d'une architecture centralisée, toute l'intelligence de test est concentrée dans le contrôleur système, alors que des contrôleurs carte minimaux permettent de transmettre les données et les commandes de test aux circuits et de récupérer les résultats de test. Ces contrôleurs minimaux jouent le rôle d'une simple interface. Le second traitement concerne la prise de décision sur la nature du contrôleur système: logiciel ou matériel. Dans le cas d'un contrôleur logiciel, c'est le processeur externe de test qui exécute un algorithme sous forme de plan de test pour tester tous les éléments du système. Dans le cas d'un contrôleur système matériel, le plan de test est implanté dans un circuit dédié, avec un protocole de communication particulier pour pouvoir gérer le test des différentes cartes du système.

Les deux derniers traitements de prises de décisions sont étroitement liés à la structure du système: nombre de cartes, complexité des cartes, etc. et aux choix et préférences du concepteur: rapidité du test, coût, tolérance aux pannes, etc.

Décision prise concernant le type de l'architecture globale de test et la nature du contrôleur système, le traitement final de la testabilité des cartes peut avoir lieu. Ce traitement n'aurait pas pu avoir lieu plus tôt, étant donné que nous n'avions aucune idée sur la nature des contrôleurs carte que nous devions générer: minimaux ou intelligents. Ce traitement consiste essentiellement en la construction des chaînes Boundary Scan ou autres chaînes de test sur les cartes, en la génération de contrôleurs de test et en l'harmonisation de la structure de test de la carte lorsque un ou plusieurs MCM sont présents. Il est important de noter que les circuits des MCMs subissent les mêmes traitements que ceux des cartes, que des contrôleurs de test MCM peuvent être générés si ce dernier est en cours de conception et que des chaînes de test Boundary Scan ou autre peuvent y être construites. Un MCM subit les mêmes traitements qu'une carte, jusqu'à l'étape d'harmonisation qui a pour rôle d'organiser le test dans une carte en présence de MCMs: hiérarchisation des contrôleurs, jonctions entre les chaînes de test, etc.

Après avoir achevé la testabilité des cartes, le traitement qui consiste à finaliser le test au niveau système peut avoir lieu. Sa tâche principale est de générer le contrôleur système, logiciel ou matériel, et d'assurer les liens avec les contrôleurs cartes, autrement dit à définir le bus de test de l'architecture globale.

Le détail de tous les traitements que nous venons d'introduire constitue la suite de ce chapitre.

4.3 Modélisation et spécification des systèmes

Afin de pouvoir traiter tous les éléments d'un système, ce dernier doit être décrit et modélisé selon un modèle particulier, qui soit le plus flexible possible et qui puisse le représenter en gardant au mieux ses spécificités structurelles et fonctionnelles. Plusieurs modèles sont disponibles dans la littérature (voir section 4.3) et notre choix s'est porté sur le modèle orienté objet.

4.3.1 Justification des choix de modélisation

Rappelons tout d'abord que dans un modèle orienté objet, toutes les ressources sont considérées comme étant des objets, définis à travers leurs *attributs* spécifiant l'aspect statique de l'objet et leurs *méthodes* qui sont des fonctions sur les attributs et qui spécifient l'aspect dynamique de l'objet. Une *classe* est un groupe d'objets partageant la même structure (attributs/méthodes). L'abstraction, l'encapsulation, la modularité, l'héritage, l'instanciation et la hiérarchie sont les principes de base d'un modèle orienté objet (MOO, voir paragraphe 3.2.1).

Mis à part le modèle orienté objet qui adopte une stratégie ascendante partant des constituants du système, tous les autres modèles de la littérature adoptent une stratégie descendante qui essaie de répondre à la question: *"que doit faire le système ?"*. Etant donné que la méthodologie que nous présentons suit un traitement ascendant partant de la testabilité des circuits pour atteindre celle des cartes, des MCMs et du système en sa totalité, le MOO peut être a priori recommandé pour les systèmes auxquels nous voulons ajouter du test. De plus, trois raisons essentielles nous ont conduit à choisir définitivement le MOO:

- Sa bonne lisibilité: un MOO est très réaliste étant donné que chaque élément du système peut être représenté par un objet (ex Circuit ->Objet Circuit). Donc la structure du modèle reste très proche de celle du système dans la réalité.

- Sa programmabilité: plusieurs langages informatiques supportent la programmation orientée objet. Essentiellement le langage C++ qui introduit les concepts OO dans sa syntaxe.

- Le MOO facilite l'adoption des techniques de réutilisation qui représentent l'un des principes de notre méthodologie.

4.3.2 Le modèle OO adopté

Le modèle OO que nous avons adopté considère avant tout une super-classe nommée `Administrative_Class` (figure 4.3) de laquelle tous les objets du système héritent certaines informations comme: le nom de l'objet, sa date de création et son type. Trois classes principales: `System_Class`, `PCB/MCM_Class` et `Circuit_Class` dérivent de la classe `Administrative_Class` et instancient respectivement les objets: `System Object`, `PCB/MCM Object` et `Circuit Object`.

Nous avons décidé que les cartes (PCBs) et les MCMs ont la même classe de référence, à savoir la `PCB/MCM_Class` étant donné la grande ressemblance de leurs structures respectives et des techniques de test qui leur sont réservées.

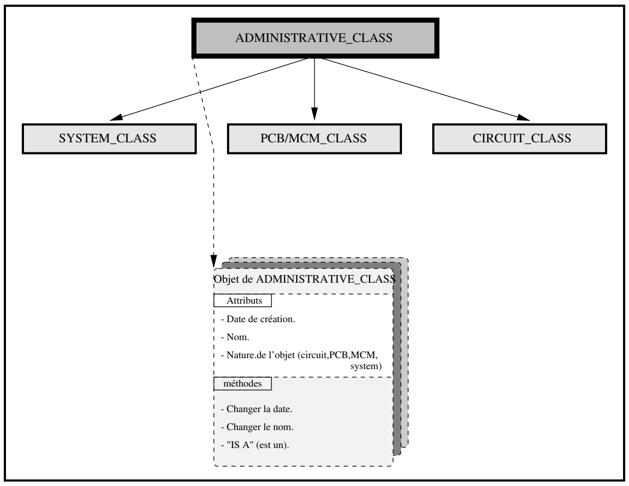

Fig. 4.3 – La classe Administrative

Pour tous les objets du modèle, les attributs concernent de près ou de loin la testabilité de l'élément. En effet, des attributs comme la *Complexité* ou la *Criticité* peuvent influencer la stratégie de test à adopter pour l'élément en question. Les méthodes sont des opérations associées aux attributs, pour assurer la testabilité de l'élément, comme par exemple l'ajout de structures de test dans son architecture (ex: ajouter du BS dans la description structurelle d'un objet circuit), l'évaluation de sa complexité, l'évaluation du surplus en surface dû à l'ajout de structures de test, ou encore la génération de contrôleur de test pour l'élément en question. Dans ce qui suit nous présentons le détail des différentes classes du modèle.

La classe `System_Class`

Un objet de cette classe représente le système entier à traiter. Parmi ses attributs nous citons principalement: la *liste d'éléments* qui est une liste chaînée d'objets PCB, la complexité du système et la criticité du système (importance du système dans son environnement) (figure 4.4).

Une première ramification de cette classe est la classe *System+BS1149.5_Class* où sont instanciés les systèmes adoptant la norme IEEE1149.5 et son bus MTM (Module Test and Maintenance bus) et dont certaines cartes contiennent du BS. Parmi les méthodes de cette sous-classe, la fonction *générer BSM* (BSM: "BS Master") permet de descendre plus bas dans la hiérarchie de la classe pour atteindre la (sous-sous) classe *System+BS1149.5+BSM* où sont instanciés les systèmes contenant un contrôleur de bus 1149.5 dit "BS Master" capable de gérer le test des cartes BSintégrées dans le système.

La deuxième ramification est la classe *System+BS_Class* où sont instanciés les systèmes intégrants des cartes BS mais qui n'adoptent pas la norme IEEE 1149.5. Parmi les méthodes de cette sous-classe, la fonction *générer SBSC* (SBSC: "System BS Controller") permet de descendre plus bas dans la hiérarchie de la classe pour atteindre la (sous-sous) classe *System+BS+SBSC* où sont instanciés les systèmes contenant un contrôleur de test BS au niveau système (pour toutes les cartes).

Enfin, nous avons prévu une troisième hiérarchisation de la classe source *System_Class* à savoir la sous-classe *System+PrefTest_Class*. Cette classe permet d'instancier des systèmes pour lesquels le concepteur indique ses préférences en matière de testabilité. Elle permet d'une part d'instancier les différents objets du système de façon tout à fait normale puisqu'elle hérite de la classe *System_Class* tous ses attributs et toutes ses méthodes et d'autre part d'introduire les préférences de l'utilisateur sous forme d'attributs.

Pour résumer, la classe *System_Class* admet trois hiérarchisations différentes:

– *System >System+BS1149.5 >System+BS1149.5+BSM*

– *System >System+BS >System+BS+SBSC*

– *System >System+PrefTest*

La classe `PCB/MCM_Class`

Cette classe instancie les cartes et les MCMs du système, puisque comme nous l'avons indiqué précédemment, ils sont traités de la même façon par notre méthode vu leurs similitudes structurelles. Les principaux attributs de cette classe sont: la *liste d'éléments*

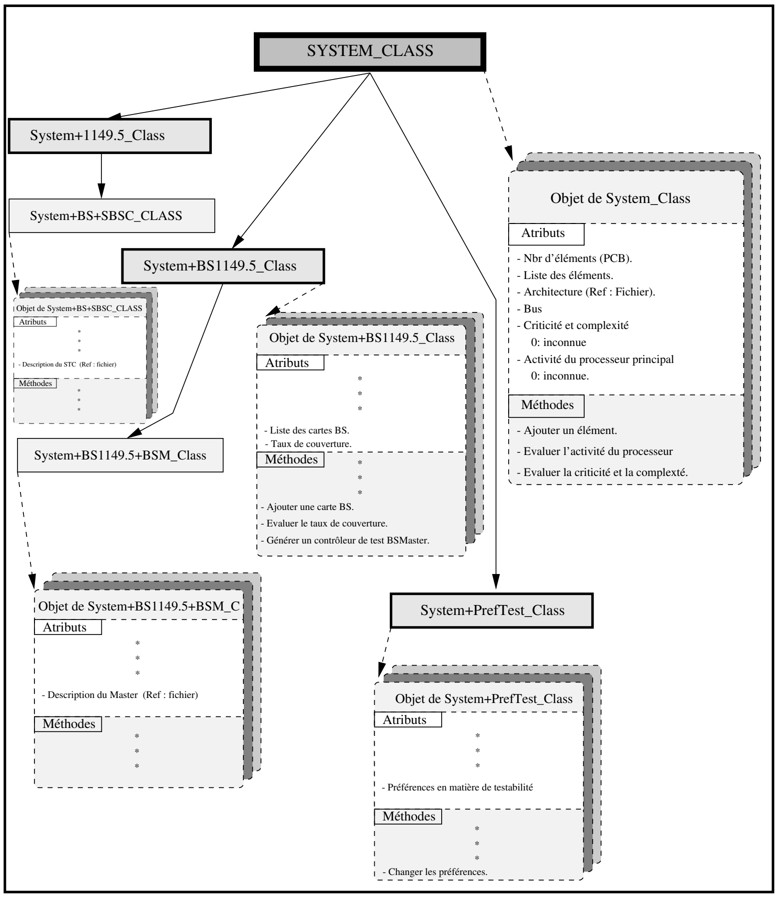

Fig. 4.4 – La classe *System_Class*

qui est une liste chaînée d'objets circuits, la *complexité* de la carte ou du MCM et sa *criticité* (importance de l'élément dans le système) (figure 4.5).

Une première hiérarchisation de cette classe nous amène à la sous-classe *PCB/MCM+BS1149.1_Class* dans laquelle sont instanciées les cartes ou les MCMs contenant des fonctionalités Boundary Scan. En plus des attributs et des méthodes qu'elle hérite de la classe mère *PCB/MCM_Class*, elle admet de nouveaux attributs comme par exemple la liste d'éléments BS (circuits BS) et la liste d'instructions BS qui sont prévues dans ces circuits, et de nouvelles méthodes dont la plus importante est la fonction *générer BSC* qui a pour rôle de générer un contrôleur Boundary Scan pour l'élément en question. Cette dernière fonction nous permet de descendre plus bas dans cette première hiérarchisation pour atteindre la (sous-sous) classe *PCB/MCM+BS1149+1+BSC_Class* dans laquelle sont donc instanciés les éléments (carte ou MCM) contenant des circuits BS et un contrôleur Boundary Scan local pour les contrôler.

Une deuxième hiérarchisation de la classe mère *PCB/MCM_Class* est la sous-classe

PCB/MCM+BTC_Class. En effet une méthode de la classe mère dite "générer BTC" ("Board Test Controller") permet de générer un contrôleur de test pour carte ou pour MCM. A la différence du BSC, le contrôleur BTC ne contrôle pas les circuits BS mais les circuits testables non BS, c'est-à-dire les circuits contenant des fonctionnalités BIST ou SCAN ou les deux à la fois. Il est toutefois possible d'avoir un contrôleur "universel" (BSC+BTC), qui est en effet un contrôleur BSC avec des modules pour le contrôle des circuits BIST ou SCAN. Dans cette dernière configuration l'élément (carte ou MCM) contenant ce type de contrôleur est instancié dans la (sous-sous) classe *PCB/MCM+BS1149.1+BSC_Class* de la première hiérarchisation.

Comme pour la classe système, une sous-classe *PCB/MCM+PrefTest_Class* est prévue et sert à instancier les cartes ou les MCMs pour lesquels le concepteur a des préférences en matière de testabilité. Elle hérite directement de la classe mère *PCB/MCM_Class* tous ses attributs et ses méthodes, constituant ainsi la troisième hiérarchisation de cette classe.

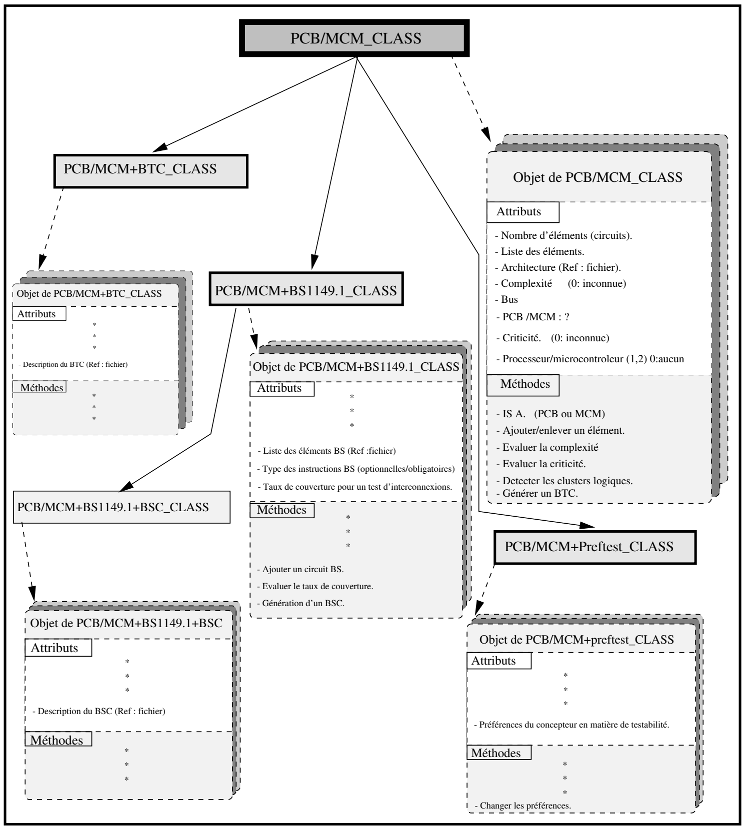

Fig. 4.5 – La classe *PCB/MCM_Class*

Pour résumer, la classe *PCB/MCM_Class* admet trois hiérarchisations différentes:

– *PCB/MCM >PCB/MCM+BS1149.1 >PCB/MCM+BS1149.1+BSC*

– *PCB/MCM >PCB/MCM+BTC*

– *PCB/MCM >PCB/MCM+PrefTest*

La classe `Circuit_Class`

Chaque circuit du système est instancié dans cette classe ou dans les classes qui en dérivent. Elle se ramifie en quatre sous-classes principales: la sous-classe *CircuitDig_Class* pour les circuits digitaux, la sous-classe *CircuitAna_Class* pour les circuits analogiques et la sous-classe *CircuitMix_Class* pour les circuits mixtes. La classe *CircuitSoft_Class* instancie des modules logiciels, dans le cas d'une co-conception matériel/logiciel. Dans ce qui suit nous allons présenter principalement la hiérarchisation de la classe *CircuitDig_Class* (figure 4.6).

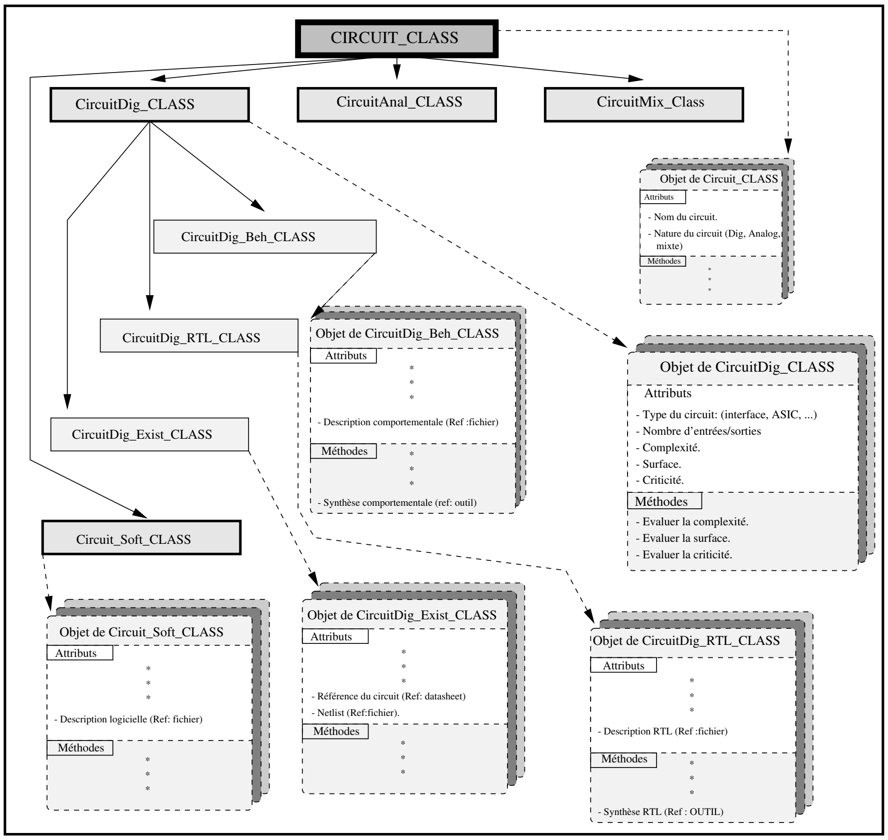

Fig. 4.6 – La classe `Circuit_Class`

La classe *CircuitDig_Class* (que nous appellerons désormais classe mère dans le restant de ce paragraphe) admet trois ramifications différentes à savoir les (sous-sous) classes: *CircuitDig_Beh_Class* pour l'instanciation des circuits digitaux décrits en

VHDL comportemental ("Behavioral"), *CircuitDig_RTL_Class* pour les circuits décrits en VHDL RTL structurel et *CircuitDig_Exist_Class* pour instancier les circuits existants ("off-the-shelf") ou les circuit dont on a la "netlist" (une méthode de la classe, dite "ISA", permet de savoir de quel type d'objet il s'agit).

Descendant plus bas dans la hiérarchie de la classe *CircuitDig_Class* nous rencontrons deux autres classes (figure 4.7):

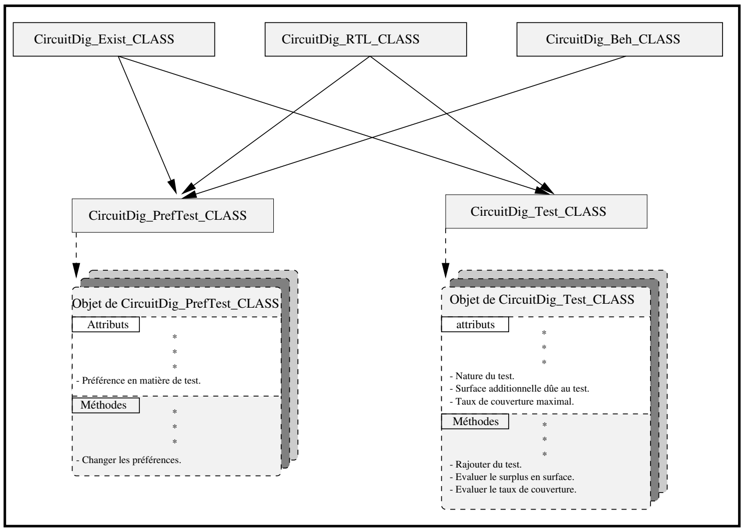

Fig. 4.7 – Suite classe *Circuit_Class*

- Une première (sous-sous-sous) classe dite *CircuitDig_Test_Class* dans laquelle sont instanciés les circuits testables du système. Elle hérite de deux classes différentes (héritage multiple) à savoir les classes *CircuitDig_RTL_Class* et *CircuitDig_Exist_Class*. Comme jusqu'à présent on ne sait pas rajouter du test au niveau comportemental, cette classe ne peut pas hériter de la classe *CircuitDig_Beh_Class*.

- Une deuxième (sous-sous-sous) classe dite *CircuitDig_PrefTest_Class* héritant des trois classes *CircuitDig_RTL_Class*, *CircuitDig_Exist_Class* et *CircuitDig_Beh_Class*, et dans laquelle sont instanciés les circuits pour lesquels le concepteur fournit ses préférences en matière de testabilité.

Exemple de Déclaration d'une classe en C++

```
class System_Class : public Administrative_Class
{
      public:
       char*    name;
       List*    elements;           /* Liste des cartes */
       char*    ArchitecFile;      /* Fichier contenant l'architecture*/
       char*    bus;
       int      complexity;
       int      criticity;
       int      availability;      /* Activité du processeur principal */
       prtype  processorType;

       System_Class();          /* Constructeur de la classe */
       ~System_Class();       /* Destructeur de la classe */

       ObjectType    IsA()             {return SystemNode;}
       int           IsBase()           {return AdministrativeNode;}
       int           EvalAvail()       {return availability;}
       int           EvalSysComp()  {return complexity;}
       int           EvalCrit()        {return criticity;}
       void          print();
};
```

Fig. 4.8 – Déclaration C++ de la classe `System_Class`

4.3.3 Implémentation du modèle

Le modèle ainsi présenté peut être implanté en C++. Les classes sont déclarées selon leurs attributs, leurs méthodes et leurs héritages. La figure 4.8 représente la déclaration C++ de la classe `System_Class`. Toutes les autres classes du modèle sont déclarées de Manière équivalante.

4.4 Etude de la testabilité des circuits

Ce module concerne l'étude de la testabilité des circuits du système. La méthode que nous présentons n'étant pas une méthode de synthèse en vue de la testabilité, l'idée est donc de guider l'incorporation de fonctionnalités de test dans les circuits en proposant des stratégies et des choix de test au concepteur.

Les stratégies qui peuvent être proposées diffèrent en deux points essentiels, à savoir les techniques de test à adopter (BS, BIST, SCAN,...) et la façon dont les fonctionnalités de test seront introduites dans le circuit. Le fait que les circuits peuvent être instancié à différents niveaux d'abstractions allant du comportemental à l'existant, implique que ce module peut avoir recours à différents outils de synthèse et à différents générateurs de structures de test au niveau circuit. La figure 4.9 résume les principales configurations qui peuvent se présenter au concepteur.

Supposons qu'on soit en présence d'une description comportementale d'un circuit, comme il n'est pas possible de rajouter des structures de test directement à une telle description (paragraphe 3.3.2) alors deux possibilités s'offrent au concepteur:

- Effectuer une synthèse de haut niveau classique à l'aide de l'outil de son choix, commercial ou universitaire, dans le but d'obtenir une description RTL pouvant contenir ultérieurement des structures de test.

- Effectuer directement une synthèse de haut niveau en vue de la testabilité en faisant appel à un outil pour le faire. Il n'existe pas, à notre connaissance, d'outil commercial, mais en revanche plusieurs développement universitaires sont en cours (voir paragraphe 3.3.2). On peut aussi, dans le cadre d'une solution mixte, faire appel à l'outil IDAT en cours d'élaboration au sein de notre laboratoire [BMM99]. Lors de cette synthèse et principalement durant l'étape d'allocation, des unités fonctionnelles testables peuvent être allouées pour obtenir une description RTL testable. Cette allocation en vue de la testabilité obéit à la fois aux contraintes du concepteur et à des études d'évaluation de testabilité et de compromis surface/temps/testabilité [BMM98b] [BMM98a].

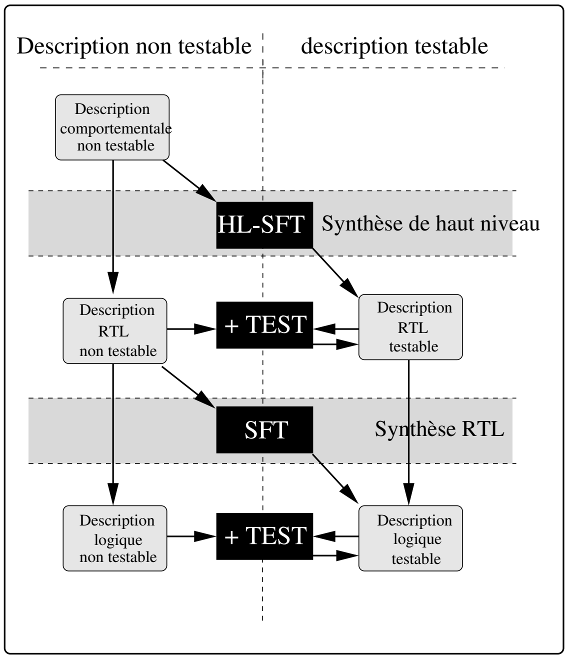

Fig. 4.9 – Orientation de la testabilité au niveau circuit

Supposons maintenant qu'on soit en présence d'une description RTL. Etant donné que l'incorporation de structures de test dans une telle description est un problème parfaitement résolu (paragraphe 3.3.2), la tâche principale consiste donc à imaginer quelles structures incorporer. Un module externe dédié à cette tâche peut être considéré, il aura à effectuer des recommandations au concepteur, en tenant compte des spécificités de la description du circuit et des contraintes du concepteur. Des générateurs de structures de test sont prévus à cet effet, ils ont pour but de rajouter du SCAN, du BIST ou les deux, en minimisant l'extension en surface qu'ils occasionnent. Ce module est considéré externe puisque la méthode que nous présentons n'intervient pas au niveau circuit mais au niveau système, c'est-à-dire les cartes et le système en sa globalité. En ce qui concerne l'ajout de fonctionnalités Boundary Scan, cela fait partie des tâches du corps principal de la méthode, étant donné que c'est un problème au niveau carte. En effet une évaluation de la testabilité de la carte (chapitre 5) permet de détecter sur une carte quels sont les circuits prioritaires quant à l'incorporation de structure BS. Cette étude BS a lieu avant le commencement de l'étude de testabilité des circuits proprement dits de telle façon que pour chaque circuit on sache avant d'entamer l'étude de sa testabilité s'il serait intéressant d'y rajouter du BS (figure 4.10).

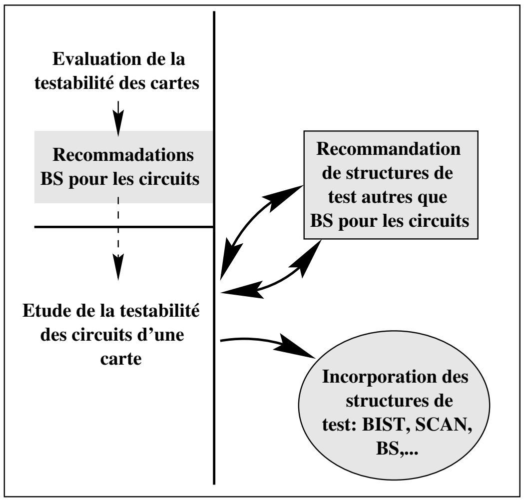

Fig. 4.10 – Rajout de structures de test BS dans un circuit

Il est à rappeler que pour les descriptions de circuits déjà testables, il est possible d'étudier l'ajout de complément de test ; le cas le plus fréquent est l'introduction de structures BS dans un circuit contenant déjà du BIST ou du SCAN.

Le concepteur a par ailleurs la possibilité de synthétiser les descriptions RTL des circuits pour obtenir des descriptions au niveau logique. Là encore des générateurs de structures de test au niveau logique (paragraphe 3.3.2) sont disponibles et permettent d'introduire les fonctionnalités qui auraient été recommandées par un module externe (pour le BIST et le SCAN comme c'était le cas pour les descriptions RTL) ou suite à l'évaluation de la testabilité des cartes (pour le BS).

4.4.1 Cas des circuits existants

La possibilité d'ajouter des fonctionnalités Boundary Scan à un circuit sous forme de registres externes (" BUFFER BS ") règle le cas des circuits existants. En effet, on trouve sur le marché des registres spéciaux constitués de cellules Boundary Scan qui peuvent être connectés aux entrées/sorties de circuits non BS pour les rendre testables. Le concepteur connaît à priori le nombre exact de registres BS qu'il peut ajouter sur chacune de ses cartes, ainsi que leurs tailles. Par ailleurs, grâce à l'étape d'évaluation, les noeuds des cartes les moins faciles à tester sont détectés. En tenant compte des deux dernières informations, on peut décider à quels circuits les registres BS disponibles peuvent être raccordés en se basant sur le principe suivant: maximiser le nombre de circuits 100 % BS (toutes les broches d'entrées/sorties du circuit sont connectées à un ou plusieurs registres BS) sur la carte. En effet, avoir un circuit dont toutes les entrées/sorties sont "bufferisés" en BS permet de profiter de toutes les possibilités de l'architecture BS contenue dans les registres. Dans le cas contraire (le circuit est partiellement "bufferisé"), certains test BS ne seront plus possibles (ex : le test interne avec INTEST). Cela revient à dire qu'il faut éviter de raccorder systématiquement les registres BS aux entrées/sorties des noeuds les moins testables de la carte (par ordre de testabilité) puisque cela nous ramène à des circuits partiellement BS et très difficilement testables. Le module qui gère l'affectation des entrées/sorties de ces " buffers " adopte par défaut le principe que nous venons de citer, toutefois le concepteur a la possibilité de favoriser d'autres critères:

- **`Traitement par défaut:`** le module récupère le classement des circuits en terme de testabilité (classement des différents circuits de la carte, du moins testable au plus testable, issu du module d'évaluation). Commençant par le moins testable, il vérifie si l'un des registres BS disponibles permet de connecter toutes ses broches d'entrées/sorties (si plusieurs registres peuvent faire l'affaire, celui à choisir sera celui qui aura le nombre de broches égal ou le plus proche de celui du circuit). Si c'est le cas, il l'affecte au circuit en question, sinon il passe au circuit suivant et refait le même traitement. La figure 4.11 montre un exemple d'un tel traitement.

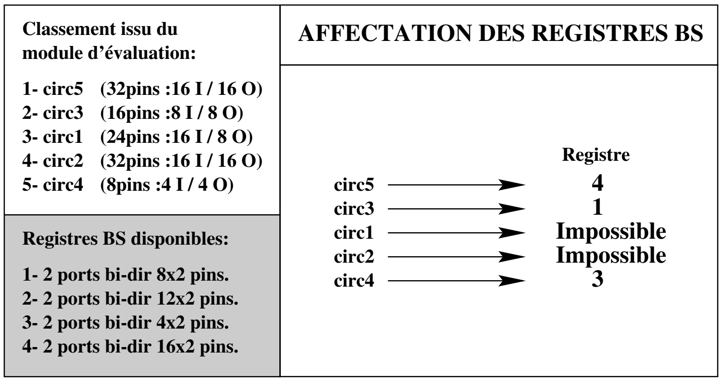

Fig. 4.11 – Exemple de traitement par défaut

- **`Traitement par compensation :`** Ce traitement est très proche du traitement par défaut (affectation par ordre de testabilité des circuits), la seule différence réside dans le fait qu'un registre BS peut être partagé entre deux ou plusieurs circuits. La figure 4.12 représente un exemple simple d'un tel traitement: il s'agit d'affecter des registres BS (trois registres identiques ayant chacun deux ports à 4 pins bidirectionnels sont disponibles) à deux circuits (circ1: 8 entrées et 4 sorties et circ2: 4 entrées et 8 sorties). L'idée est d'affecter : le premier registre aux 8 entrées de circ1 (les deux ports en entrées), le troisième registre aux sorties de circ2 (les deux ports en sorties) et le deuxième est partagé entre les deux circuits (un port pour les 4 sorties de circ1 et l'autre pour les 4 entrées de circ2). Ce traitement a l'avantage de compenser le manque de ressources qui peut se présenter dans certains cas (pour le même exemple, aucun registre n'aurait été affecté si nous avions utilisé le traitement par défaut). En revanche, il présente un inconvénient majeur, celui d'être obligé d'appliquer le même test à tous les circuits partageant un même registre BS (dans l'exemple présenté, le deuxième registre fixe le type de test à appliquer aux deux circuits, le premier et le troisième doivent suivre pour éviter les conflits). Il est certain qu'un tel traitement limite les performances de l'architecture de test BS. C'est pour cela que ce traitement est optionnel, et le concepteur peut y faire appel lorsqu'il estime pouvoir s'en contenter.

- **`Le traitement exhaustif :`** C'est un traitement qui ne s'intéresse pas aux circuits en tant que tels, mais à leurs broches d'entrées/sorties et plus précisément à leur classement par ordre de testabilité. En effet, il utilise le classement des entrées/sorties des circuits de la carte en matière de testabilité, fourni par le module d'évaluation. Partant du moins testable, il affecte une à une les broches des dif-

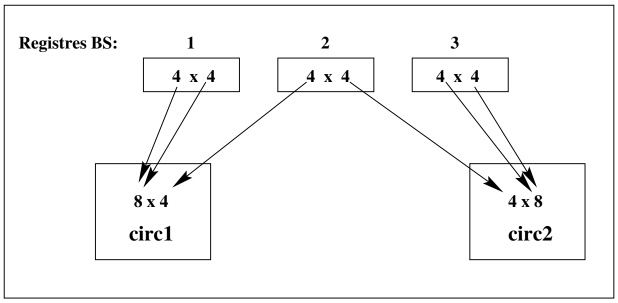

Fig. 4.12 – Exemple de traitement par compensation

férents circuits jusqu'à épuisement de tous les ports des registres BS disponibles. Un calcul préalable permet de positionner les ports bidirectionnels des registres BS de façon optimale en tenant compte du nombre de broches d'entrée et de broches de sortie sur la carte. La figure 4.13 donne un exemple simple traité de la sorte. Un seul registre BS de 16 bits (2 ports de 8 bits) est disponible, les 8 broches de sortie et les 8 broches d'entrée les moins testables sont affectés aux deux ports du registre. Il est clair que la broche 7 du circuit **circ2** ne peut pas être affectée puisqu'elle représente une sortie d'un circuit, alors que les huit cellules du port réservé aux sorties sont déjà toutes affectées. Ce traitement présente le même inconvénient que le traitement précédent, étant donné qu'un registre peut être affecté à plusieurs circuits à la fois. On peut même dire que cet inconvénient est plus flagrant avec ce traitement qu'avec le traitement par compensation, vu l'importante interdépendance qui se crée entre les différents circuits du point de vue du test BS, ce qui restreint le concepteur à un seul type de test BS, en général le test d'interconnexions avec EXTEST. En revanche, il présente l'avantage d'augmenter au maximum la testabilité globale d'une carte, étant donné qu'il affecte de façon prioritaire les broches les moins testables. Ce traitement est optionnel, et c'est au concepteur de décider de son application.

– **Le traitement manuel :** Avec ce type de traitement, c'est le concepteur qui affecte manuellement les différents registres aux différents circuits. En général, c'est un traitement qui succède aux trois autres présentés plus haut, dans le but de permettre au concepteur d'ajuster affectations si cela s'avère nécessaire.

Notons que lorsque nous parlons de registres BS disponibles, cela inclut la garantie d'un emplacement réel sur la carte.

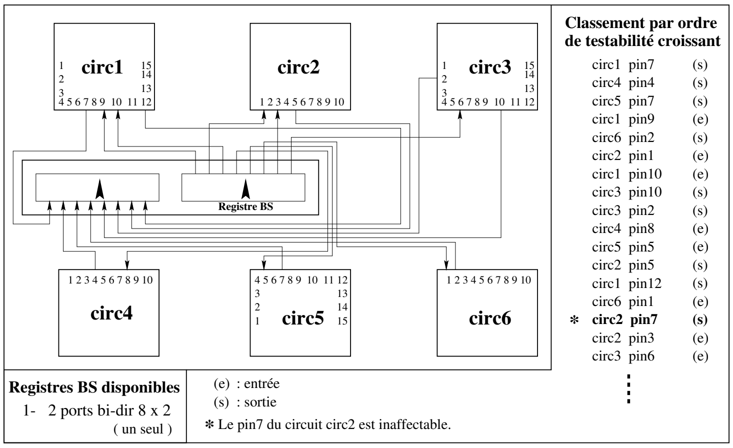

Fig. 4.13 – Exemple de traitement exhaustif

Pour résumer, mis à part les recommandations concernant les circuits qui devraient contenir de façon prioritaire du BS et qui sont gérées au sein des modules principaux de la méthode que nous présentons, tous les autres traitements relatifs à la testabilité des circuits telles la recommandation de stratégies de test pour les circuits (au niveau RTL ou logique), la génération de structures de test ou encore la synthèse de haut niveau en vue de la testabilité (cas des descriptions comportementales) sont effectués par des modules externes auxquels les modules principaux font appel.

4.5 Pré-traitement des cartes

Une fois la testabilité de tous les circuits du système étudiée, il aurait été logique d'entamer directement l'étude de la testabilité des cartes qui englobe la construction des chaînes SCAN ou BS, et si besoin la génération des contrôleurs de test. Toutefois, pour accomplir ces tâches, il faut avoir une idée du type d'architecture de test du système à adopter (distribuée ou centralisée) et sur la nature du contrôleur système (matériel ou logiciel) (paragraphe 4.2). En effet, d'une part le choix du type d'architecture influence directement le choix du type des contrôleurs de test cartes (minimaux ou intelligents) et d'un autre coté le choix de la nature du contrôleur système influence la nature des interfaces de communication entre les contrôleurs carte et le contrôleur système.

Cela dit, une étape intermédiaire de pré-traitement des cartes s'avère nécessaire, notamment pour effectuer une réévaluation de leur testabilité après les modifications subies par les circuits lors de la précédente étape. Cette réévaluation permet de donner une idée sur la complexité des cartes, qui représente un paramètre important dans la prise de décision concernant le type d'architecture de test au niveau système à adopter et la nature du contrôleur système (paragraphes 4.6 et 4.7). Cela explique pourquoi cette étape intermédiaire a lieu avant les deux étapes de prise de décision. Les principales tâches accomplies durant cette étape sont:

- Mise à jour du modèle orienté objet représentant le système, suite aux modifications apportées aux circuits. En effet, les objets PCB (ou MCM) comportant des objets circuits modifiés (auxquels du test a été ajouté) peuvent migrer d'une classe à une autre. A titre d'exemple, supposons que du BS a été incorporé dans les descriptions des circuits d'une carte, l'objet PCB représentant cette carte aura à migrer de la classe *PCB/MCM_Class* à la classe *PCB/MCM_BS_Class*.

- Détection de la présence de contrôleurs de test au niveau carte et de leurs types (intelligents ou simples interfaces). Cette information peut s'avérer importante dans la suite du traitement, puisqu'elle peut influencer le choix de l'architecture globale de test et le choix de la nature du contrôleur système. En effet, prenons l'exemple d'un système où un grand nombre de cartes contient déjà des contrôleurs de test intelligents, il est tout à fait logique que le choix se porte sur une architecture de test distribuée avec un contrôleur système central minimal.

- Détection des différentes instructions supportées par les circuits BS présents sur une carte (ou un MCM). Cette détection peut s'avérer une information très utile lors de l'étape de génération de contrôleurs de test BS (détails dans le paragraphe 4.8.2).

Pour résumer, il est clair que cette étape intermédiaire sert à effectuer une "check-list" globale de toutes les cartes (ou MCM) du système, ce qui permet de réunir un maximum d'informations utiles pour les étapes suivantes du traitement.

4.6 Architecture de test: Choix du type

4.6.1 Centralisée/distribuée: avantages, inconvénients.

Comme indiqué précédemment, ce module est responsable du choix du type de l'architecture de test à adopter pour le système: distribuée ou centralisée. Pour rappeler ce qui a été dit dans le paragraphe 4.2, une architecture distribuée du point de vue du test voit l'essentiel de l'intelligence de test distribuée entre les contrôleurs carte. Ces derniers sont capables de réaliser à eux seuls le test des cartes sur lesquelles ils sont implantés. Ainsi, un simple contrôleur système minimal suffit à gérer le test de toutes les cartes du système (figure 4.14). D'un autre coté, une architecture centralisée se caractérise par un contrôleur de test niveau système doté d'une intelligence qui lui permet de gérer à lui seul le test de tous les éléments du système, ce qui revient à dire que les contrôleurs carte ne jouent que le rôle de simples interfaces (figure 4.15). Ces derniers peuvent même être inutiles lorsque le contrôleur système a la possibilité de fournir directement aux cartes les signaux de test.

Les deux architectures admettent l'une comme l'autre des avantages et des inconvénients qui sont plus ou moins mis en évidence en fonction de certains paramètres tels le coût du matériel ou encore la complexité des cartes.

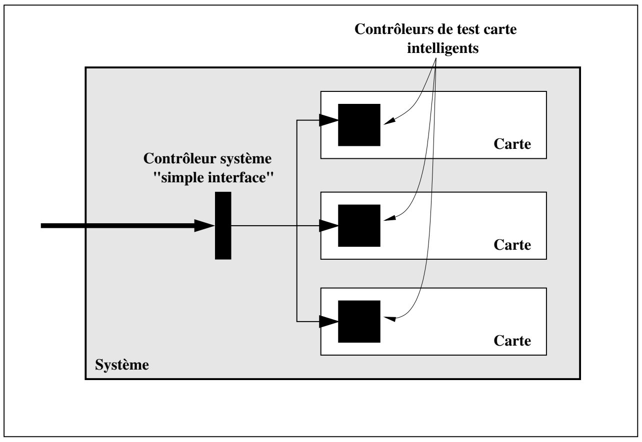

Fig. 4.14 – Architecture de contrôle distribué

Une architecture distribuée favorise la rapidité du test, étant donné que sa parallélisation au niveau des cartes est tout à fait possible. En effet, il suffit que le contrôleur système principal donne l'ordre aux contrôleurs carte de déclencher le test pour que ceux-ci commencent à dérouler en parallèle les algorithmes nécessaires pour le test des cartes qu'ils contrôlent. La rapidité du test représente l'avantage principal d'une architecture distribuée.

Supposons maintenant que le contrôle au niveau système se fait d'une manière complètement logicielle, le processeur principal du système est lui-même responsable de la gestion du test (cas très fréquent). Le fait que l'architecture soit distribuée lui permet de déclencher le processus de test et de se libérer ainsi pour d'autres tâches. Il est donc possible dans ce cas d'envisager une configuration dans laquelle une partie du système fonctionne normalement alors qu'une autre est en train d'être testée.

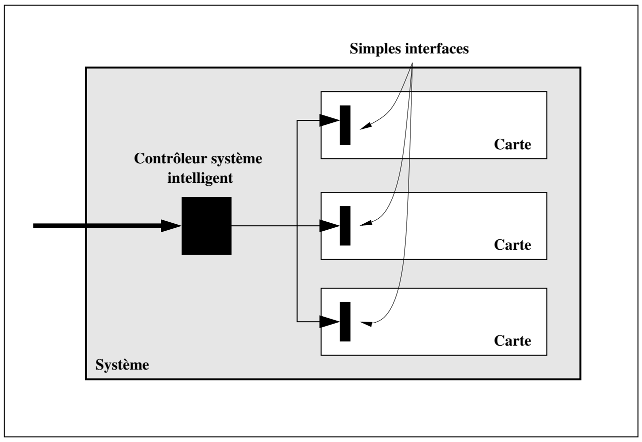

Fig. 4.15 – Architecture de contrôle centralisé

Le principal inconvénient d'une architecture distribuée est le coût, en effet l'utilisation systématique de contrôleurs carte intelligents rend le coût du dispositif de test assez élevé, surtout si on est en présence d'un grand nombre de cartes. De même, et dans le cas d'un MCM, la génération de contrôleur de test sous forme de puce occasionne un surcoût important.

Le deuxième inconvénient d'une telle architecture est la difficulté de sa mise en place. En effet, un encombrement au niveau des cartes peut rendre dans certains cas l'ajout d'un contrôleur presque impossible. Ce problème de mise en place de l'architecture de test se manifeste principalement dans le cas de systèmes déjà conçus, ou partiellement conçus et pour lesquels on s'apprête à ajouter du test.

Pour une architecture centralisée, les caractéristiques sont inversées. En effet, le principal avantage qu'elle présente est le fait qu'elle ne coûte pas cher et qu'elle évite les problèmes d'encombrement. Mieux encore, dans le cas où il serait décidé que le contrôleur système soit implanté en logiciel, le coût de l'architecture de test serait encore plus bas.

Le revers de la médaille est qu'une architecture centralisée présente un inconvénient majeur, à savoir la lenteur qui peut caractériser le processus de test. Comme le contrôleur système central a la lourde tâche de tester tous les éléments du système, il est donc impossible de paralléliser le test des cartes puisque le contrôleur est dans l'obligation de les tester une à une (dans le cas où les cartes sont pourvues d'interfaces pour le test) ou toutes ensembles (exemple de la figure 4.16), ce qui dans un cas comme dans l'autre allonge énormément la durée de test. Cette durée est d'autant plus longue que le système est grand, c'est-à-dire qu'il comprend un grand nombre de cartes. D'un autre coté, sachant que la durée de test d'une carte (exemple test d'interconnexions) est d'autant plus longue que la carte est complexe, il est clair qu'une architecture centralisée ne peut être recommandée pour un système contenant un grand nombre de cartes complexes. La complexité d'une carte est une grandeur qui quantifie la difficulté qu'on peut avoir à atteindre tous les noeuds d'une carte pour un contrôle ou une observation. Les facteurs qui influent le plus sur la complexité d'une carte sont le nombre de circuits qu'elle contient et le nombre de boucles d'interconnexions qui confèrent à la carte un caractère séquentiel (donc complexe). Le calcul de la complexité des cartes du système se fait lors des étapes d'évaluation, et cette complexité est inversement proportionnelle à la testabilité de la carte (voir paragraphe 5).

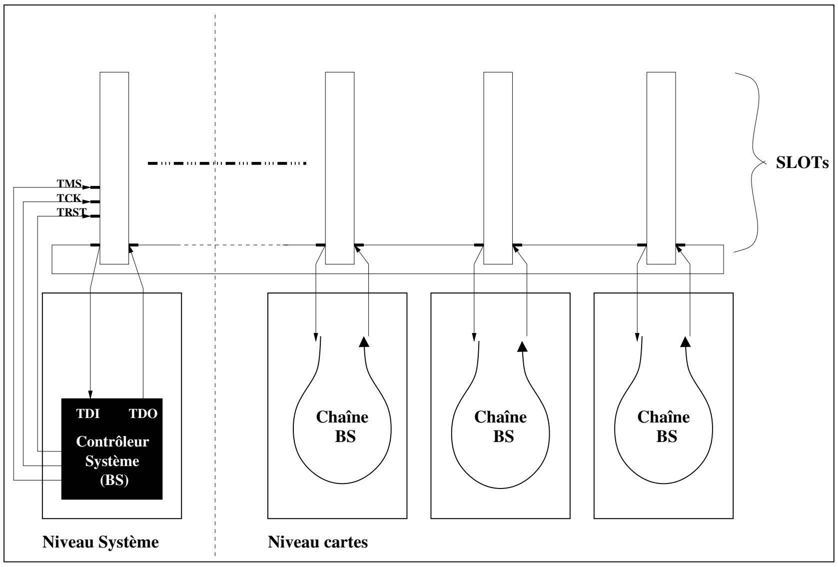

Fig. 4.16 – Architecture centralisée sans interfaces sur les cartes (Test=Boundary Scan)

Un autre inconvénient des architectures centralisées, est qu'elles ne sont pas robustes du point de vue du test. Un simple défaut au niveau du contrôleur système central suffit pour que l'architecture de test se trouve bloquée dans son ensemble. Pour cette raison, une telle architecture ne peut pas être indiquée pour des systèmes appelés à s'auto-tester régulièrement durant leur fonctionnement.

4.6.2 Les paramètres influant sur le choix

D'après ce qui a été dit dans le paragraphe précédent, on peut conclure que certains paramètres entrent en jeu dans le choix du type d'architecture de test à adopter. Ces paramètres peuvent être classés en deux parties: ceux qui relèvent des contraintes et des choix du concepteur et ceux qui sont en relation avec la structure du système lui-même.

Dans la première classe de paramètres on peut distinguer:

- La **rapidité de test** souhaitée par le concepteur qui influence, comme indiqué plus haut, le choix d'une architecture distribuée.
- Le **coût du matériel** qui représente le coût de fabrication d'un circuit sur le marché. Ceci concerne principalement la génération et la fabrication de contrôleurs de test dédiés pour les cartes ou pour le système en entier.

L'importance relative accordée par le concepteur à ces deux paramètres est modulée par des caractéristiques du système lui-même.

La seconde classe de paramètres inclut:

- La **complexité** du système, autrement dit la complexité moyenne des cartes le constituant. Cet attribut dépend donc de la structure interne des cartes et se calcule pendant les étapes d'évaluation de testabilité.
- Le **nombre de cartes** dans le système. Supposons que le coût ne représente pas un handicap particulier pour le concepteur, pour un système comprenant un grand nombre de cartes il serait intéressant d'adopter une architecture distribuée, ceci dans l'optique de paralléliser le test et de diminuer ainsi la durée du processus de test. En revanche, dans le cas où le coût de l'architecture de test revêt une importance capitale, pour un même système (nombre de cartes élevé), il serait préférable d'adopter une architecture centralisée surtout si le coût du matériel est élevé. En effet la génération de contrôleur intelligent dédié à chaque carte implique un surcoût important surtout que le nombre de cartes est élevé.

4.6.3 Evaluation des paramètres par calcul flou

Afin de proposer à l'utilisateur un choix d'architecture, nous avions le choix, au départ, entre deux solutions. La première consistait à utiliser un système expert dédié; cette solution a été abandonnée du fait de sa trop grande complexité. La deuxième solution, qui ne fournit par réellement un choix clair à l'utilisateur, consistait à générer des constatations, plus ou moins indépendantes, à partir des caractéristiques du système (complexité, nombre de cartes) et des contraintes de l'utilisateur (rapidité, coût). Un exemple de constatation pourrait être: *Vous souhaitez un test rapide du système, alors il faut une architecture du type distribuée*. Cette solution a été aussi abandonnée, puisqu'elle n'obéit pas au principe premier de la méthode, qui consiste à proposer à chaque fois une solution claire à l'utilisateur et lui laisser libre choix de l'accepter ou de la refuser. Cela dit, une telle méthode peut être utile dans l'interprétation d'un résultat obtenu par une autre solution.

Il est clair qu'il nous fallait donc une solution simple, tenant compte des différents paramètres entrant en jeu, et proposant un choix précis à l'utilisateur.

Cette solution ne pouvait pas être, a priori, numérique, étant donné que la quantification numérique exacte des paramètres entrant en jeu dans le choix du type d'architecture est très difficile à obtenir (ex: complexité), voire impossible dans certains cas (ex: rapidité du test). On a donc utilisé une quantification plus adaptée, basée sur la logique floue. Les valeurs des paramètres sont donc des valeurs floues, comprises entre le zéro et le un, et exprimant l'importance du paramètre plutôt que le paramètre lui-même. Nous considérons une subdivision de l'intervalle flou [0,1] en huit sous-intervalles. Ces sous-intervalles représentent les valeurs floues que peut prendre un paramètre pour exprimer son importance. Cette dernière peut varier de *pas du tout important* à *très important*.

De plus, cette quantification permet de s'affranchir d'un problème complexe, incontournable dans le cas d'une solution numérique. En effet, chacun des paramètres à quantifier s'exprime dans une unité propre, ce qui rend toute opération numérique mettant en commun certains d'entre eux pratiquement impossible. Le fait qu'une quantification floue exprime plutôt l'importance des paramètres, permet de considérer qu'ils ont tous la même unité "floue".

Ainsi, par exemple, on ne présente plus le nombre de cartes dans le système comme un nombre entier mais comme une valeur floue exprimant l'importance de ce nombre, c'est-à-dire s'il est très grand ou très petit ou entre les deux. De même pour la complexité, dont l'intervalle flou est subdivisé en huit sous-intervalles correspondant aux poids: *Pas du tout complexe*, *Non complexe*, *Plutôt non complexe*, *Moyennement non complexe*, *Moyennement complexe*, *plutôt complexe*, *complexe*, *Très complexe*.

Par ailleurs, concernant les paramètres exprimant les contraintes de l'utilisateur, à savoir la rapidité du test et le coût du matériel, les quantifier en flou facilite sa tâche. En effet, il peut les décrire de façon simple, commwe par exemple: *très coûteux*, *plutôt rapide*, etc.

Un moteur de calcul flou, élaboré dans [Moh97], nous permet, à travers une combinaison des différents paramètres, de déduire un type d'architecture, qui est lui aussi exprimé sur un intervalle flou dont les poids varient de *forcément centralisée* à *forcément* *distribuée*, en passant par les valeurs *plutôt centralisée* et *plutôt distribuée*.

Une fois cette étape de calcul flou terminée, il y a lieu de comparer le résultat obtenu avec les préférences du concepteur. L'étude peut le conforter dans ses choix, dans le cas où le résultat flou obtenu coïncide avec ses préférences ou confirme leur tendance (ex: préférence: *plutôt distribuée* et résultat: *forcément distribuée*). Dans le cas où le résultat est complètement à l'opposé des préférences du concepteur (ex: préférence: *distribuée* et résultat: *centralisée*), le concepteur aura à prendre lui-même la décision finale. Il sera guidé dans sa décision par différentes interprétations du résultat, qui seront générées automatiquement à la fin du calcul flou (voir paragraphe suivant). La prise de décision se fait donc de façon interactive.

4.6.4 Module de génération d'interprétations

Ce module sert à générer automatiquement des interprétations à la fin du calcul flou. Ces interprétations servent à expliquer le pourquoi d'un résultat. Il prend en entrée la préférence du concepteur, les paramètres principaux du système et le résultat flou calculé. En se basant sur une banque de données (dans ce cas une banque d'interprétation), il essaie de fournir un maximum d'interprétations pour le résultat obtenu. La figure 4.17 présente le classement hiérarchique des interprétations au sein de la banque. Elles sont au départ classées selon le résultat obtenu, ensuite selon les préférences de l'utilisateur (type de l'architecture, importance du coût [c], importance de la rapidité [r]) et enfin selon le reste des paramètres (rapidité [R], complexité [CP], nombre de cartes [N], coût du matériel [C]). Dans le but de simplifier cette banque, nous avons décidé de diminuer le nombre de subdivisons de l'intervalle flou de huit à quatre subdivisions. Par exemple, pour le résultat, les poids des quatre subdivisions seront respectivement : *centralisée*, *plutôt centralisée*, *plutôt distribuée* et *distribuée*. Cette banque est construite selon un modèle orienté objet où les classes sont définies par les poids des différents paramètres et un objet d'une classe est une interprétation. Un système d'héritage relie les classes adjacentes et cela en partant du centre (figure 4.17). Etant donné le nombre important de classes qui peuvent exister, il fallait simplifier le modèle en éliminant certaines ramifications de classe. Par exemple, à chaque fois que le résultat coïncide avec la préférence du concepteur, il est inutile de descendre plus bas dans la hiérarchie et un objet de la dernière classe serait par exemple l'interprétation : *Le résultat obtenu est conforme à vos préférences : ARCH distribuée*.

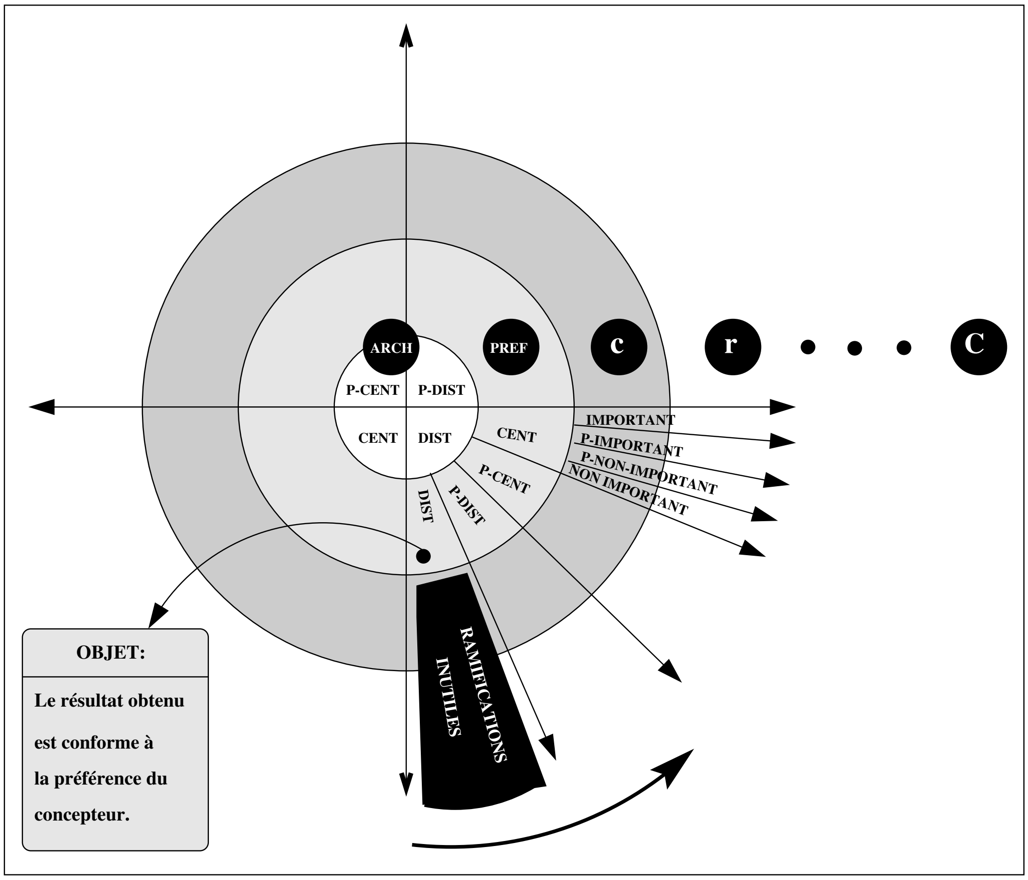

Fig. 4.17 – Structure de la banque d'interprétations

4.7 Contrôleur de test système: Choix du type

4.7.1 Logiciel/matériel: avantages, inconvénients

Ce module est responsable du choix du type du contrôleur qui assurera la gestion du test de toutes les cartes du système. Ce contrôleur système peut être implanté en matériel sous forme de circuit ou en logiciel. Comme dans le cas du choix du type de l'architecture de test, chaque type de contrôleur présente des avantages et des inconvénients.

Le principal avantage d'un contrôleur matériel est la rapidité qu'il peut offrir dans l'implémentation des tâches qui lui incombent. La rapidité de ce contrôleur est d'autant plus importante lorsqu'il s'agit d'un contrôleur intelligent (cas d'une architecture centralisée), puisqu'il aura à contrôler toutes les cartes et pour cela il doit être assez rapide pour avoir un temps global de test raisonnable. Le coût représente l'inconvénient majeur d'un contrôleur matériel (comparé à un contrôleur logiciel).

En ce qui concerne les contrôleurs logiciels, ils se caractérisent avant tout par un avantage principal, la robustesse. En effet, un contrôleur logiciel ne tombe presque jamais en panne (évidemment s'il ne contient pas de fautes), ce qui lui vaut une robustesse certaine comparée à un contrôleur matériel. Cette robustesse est d'autant plus importante dans le cas d'une architecture centralisée, où un contrôleur système défaillant implique le blocage total de l'architecture de test. D'autres avantages du contrôleur logiciel peuvent être considérés comme le coût faible (quasi nul) et l'aspect évolutif. En effet, il est toujours possible d'ajouter des modifications à un contrôleur logiciel, contrairement au cas de matériel où chaque modification passe par la conception d'un nouveau circuit.

Cela dit, un contrôleur logiciel présente l'inconvénient majeur d'être lent. Il est presque inconcevable de confier le contrôle du test d'un grand nombre de cartes à un seul contrôleur système logiciel (architecture centralisée) vu le temps que cela pourrait impliquer.

4.7.2 Les paramètres influant sur le choix du contrôleur

Comme dans le cas du choix du type de l'architecture, les paramètres influant, sur le choix du type du contrôleur système peuvent être classés en deux classes : ceux qui relèvent des contraintes et des choix du concepteur et ceux qui sont en relation avec la structure du système.

Dans la première classe de paramètres nous distinguons :

- **La rapidité de test** souhaitée par le concepteur qui influence le choix d'un contrôleur matériel.
- **Le coût du matériel**, de même que le coût de conception de ce matériel.
- **La robustesse souhaitée** pour le contrôleur choisi.

Comme lors de l'étape de choix du type d'architecture de test, l'importance relative de ces trois paramètres peut être modulée par les caractéristiques du système lui-même. Toutefois, l'importance du coût par rapport aux autres paramètres n'est pas capitale, puisqu'il y a un seul contrôleur système à générer. Dans la suite, nous ne tiendrons donc pas compte de ce paramètre coût.

La seconde classe de paramètres inclut :

- La **complexité** moyenne des cartes dans le système. Plus le système est complexe, plus le test est long et complexe et plus on a besoin d'un contrôleur matériel pour assurer un contrôle rapide et performant.

– Le **nombre de cartes** dans le système. Plus ce nombre est grand, plus le test de l'architecture et long et plus le besoin d'un contrôleur matériel se fait sentir.

Outre ces deux classes, un nouveau paramètre va entrer en jeu, à savoir le type de l'architecture choisi dans l'étape précédente, qui peut influencer largement le choix du type du contrôleur. L'architecture de test fixe la nature du contrôleur système (intelligent ou simple interface), qui représente un facteur primordial dans la prise de décision. Supposons, en effet, que nous sommes en présence d'une architecture de test centralisée, le contrôleur de test est nécessairement intelligent et donc complexe. Si on l'implante en logiciel, on risque donc d'aboutir à une architecture de test très lente et donc non performante. Dans le cas contraire (architecture distribuée), le contrôleur système joue le rôle d'une simple interface qui lorsqu'elle est implantée en logiciel peut ne pas porter préjudice aux performances de l'architecture de test.

4.7.3 Evaluation des paramètres par calcul flou

Tout comme pour la prise de décision concernant le type d'architecture de test à adopter, le moteur de calcul flou, présenté dans [Moh97], est utilisé pour déduire la nature du contrôleur système, en combinant les différents paramètres entrant en jeu et qui sont décrits en valeurs floues. Le résultat obtenu est exprimé dans un intervalle flou dont les poids varient de *forcément logiciel* à *forcément matériel*, en passant par les valeurs *plutôt logiciel* et *plutôt matériel*.

Une fois l'étape de calcul flou achevée, une étape de prise de décision finale basée sur un processus interactif commence. Le concepteur aura à finaliser sa décision en tenant compte des interprétations qui sont générées automatiquement à la fin du calcul flou (comme c'était le cas lors de la prise de décision concernant le choix du type d'architecture de test). Une banque d'interprétations est dédiée à cet effet, elle a la même structure que la banque présentée dans le paragraphe 4.6.4.

4.8 Traitement final des cartes

A ce niveau, tous les circuits ont été traités et nous savons exactement quel type de test est implanté dans chacun d'entre eux. D'autre part, nous savons quel doit être le type de l'architecture de test à adopter (distribuée ou centralisée) et quel type de contrôleur assurera le test au niveau système (matériel ou logiciel). Il est donc possible de finaliser la stratégie de test au niveau de chaque carte du système. Pour cela, deux tâches importantes restent à accomplir, à savoir la génération de contrôleurs de test au niveau carte et l'agencement des structures de test sur les cartes (ex : construction des chaînes Boundary Scan et scan).

Toutefois, avant d'entamer la définition des contrôleurs et l'agencement des structures de test, un autre traitement optionnel peut avoir lieu, dans le cas où il a été décidé que l'architecture globale du test est du type `distribuée`. Ce traitement, dit de `pseudo-centralisation`, a pour rôle de centraliser dans certains cas le contrôle de quelques cartes, même si l'architecture globale est distribuée.

4.8.1 La Pseudo-Centralisation

Ce traitement concerne donc les architectures de test distribuées où l'on peut rencontrer des cartes avec différents degrés d'intégration de fonctionnalités de test. Le but est de diminuer le nombre de contrôleurs de test carte dans le système, en centralisant le contrôle de certains groupements de cartes où le degré d'intégration de fonctionnalités de test est faible. L'idée est d'utiliser des contrôleurs carte intelligents, capables de contrôler plus d'une seule carte (figure 4.18). Chacun de ces contrôleurs est en général placé sur l'une des cartes traversées par la chaîne qu'il contrôle, et doit être autonome (intelligent) pour respecter l'architecture globale de test qui est distribuée.

Le traitement de `pseudo-centralisation` fournit donc comme résultat l'ensemble des groupements de cartes dont le test doit être centralisé. Une fois le traitement achevé, la définition des contrôleurs peut commencer, aussi bien pour les groupements formés que pour le restant des cartes.

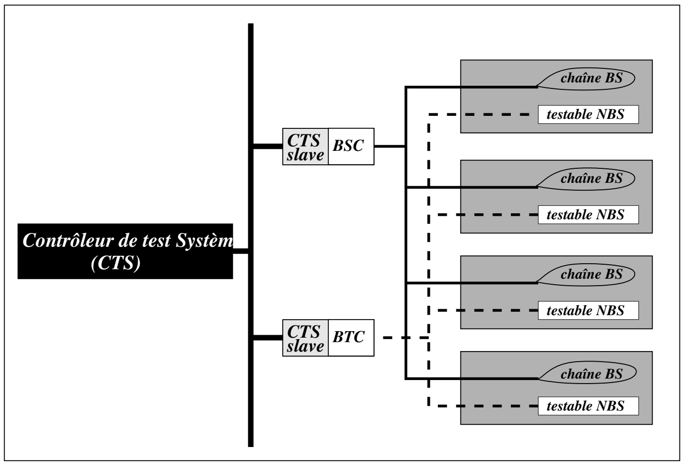

Fig. 4.18 – La `Pseudo-Centralisation`

La décision de centraliser le contrôle d'un groupement de cartes dépend d'un certain nombre paramètres de différents niveaux d'importance :

- Le premier paramètre concerne évidemment le taux d'intégration de fonctionnalités de test sur la carte. En effet il faut éliminer toute carte ayant un taux assez important d'intégration de fonctionnalités de test, nécessitant ainsi à elle seule un contrôleur dédié.

- Le deuxième paramètre concerne les contraintes techniques au niveau de la réalisation réelle du système. Ce paramètre influe sur le nombre maximal de groupements qu'on peut atteindre. En effet, dans la majorité des cas, les producteurs de systèmes fixent certaines règles d'implémentation. L'une des règles les plus utilisée consiste à dire que tous les "slots" pour cartes doivent être identiques: Chaque carte doit pouvoir être placée sur n'importe quel "SLOT" sans que ceci n'affecte son fonctionnement. Or, prenant par exemple le cas d'un test BS, la `pseudo-centralisation` consiste essentiellement à chaîner entre elles des chaînes BS de différente cartes partiellement BS: la règle des "slots" identique ne permet pas d'utiliser des nappes de fils entre les cartes, ni d'intervenir au niveau des "slots" puisque ceci va altérer leur caractère standard (chaque carte ou presque aura un "SLOT" qui lui est réservé exclusivement). Dans ce cas, une des possibilités est de réserver sur tous les "slots" des couples de broches (TDI et TDO) chaînés entre eux (figure 4.19).

  Nous pouvons rencontrer ce genre de chaînage sur les "slots" des architectures construites autour du bus PCI qui fournit tous les signaux du bus BS IEEE1149.1. Toutefois, pour la `pseudo-centralisation`, les couples de broches, chaînées entre elles et présentes sur les "slots", ne doivent pas être gérés par le processeur principal (comme c'est le cas avec le bus PCI), mais plutôt par le contrôleur de test. En effet, ce genre chaînage doit rester indépendant du bus système.

  Le nombre de groupements possibles sera égal au nombre de couples de broches sur chaque "slot". Les cartes concernées par une `pseudo-centralisation` et faisant partie d'un même groupement connectent leur chaîne BS à des couples d'un même chaînage, les autres doivent court-circuiter les deux broches des couples de leurs "slots" (figure 4.20). Le système représenté par la figure comporte sept cartes BS (le processeur système et le contrôleur de test système ne sont pas représentés), trois groupement de chaînes sont formés (carte 1 et carte 5 ; carte 2 et carte 3 ; carte 4, carte 6 et carte 7). Les trois contrôleurs BSC relatifs aux trois groupements sont placés sur les cartes 1, 3 et 7.

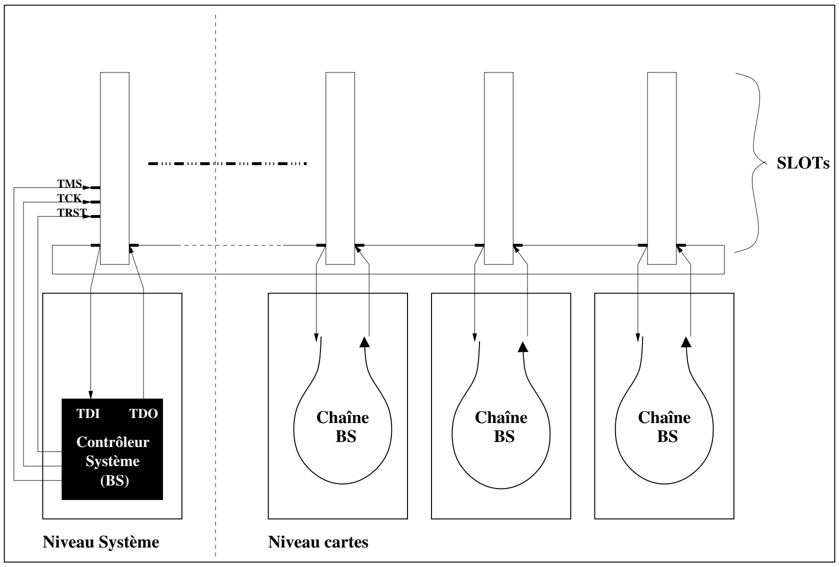

Fig. 4.19 – Le chaînage des couples TDI/TDO sur les SLOTs

Pour résumer, le concepteur doit indiquer le nombre de groupements de cartes possibles dans l'éventualité d'un traitement de `pseudo-centralisation`. Dans le cas où il indiquerait qu'aucune restriction n'est faite sur le nombre de groupements (pas de règles d'implémentation), le traitement peut en former autant que nécessaire.

- La présence de contrôleurs déjà existants qu'on retrouve dans des MCMs déjà conçus ou sur des cartes complètement ou partiellement conçues et qui sont surdimensionnés pour les éléments qu'ils gèrent. Ce paramètre est très important puisque, à lui seul, il peut motiver une `pseudo-centralisation`. Toutefois, le concepteur doit indiquer si techniquement il est possible d'intervenir au niveau de ces contrôleurs et des chaînes qu'ils contrôlent. Cela est nécessaire car on se place dans une configuration de systèmes partiellement conçus où les contrôleurs peuvent être déjà montés et le câblage des cartes concernées n'est donc pas modifiable.

- Le quatrième paramètre concerne les caractéristiques de la chaîne (BS, SCAN ou autre). Plus exactement, le concepteur doit définir la taille maximale de la chaîne, le nombre maximal de circuits qu'elle peut chaîner et les caractéristiques limites des contrôleurs à générer éventuellement (tailles maximale des différents registres, tailles maximales des mémoires internes, algorithmes intégrables, etc.). Ceci doit être fait pour chaque type de test présent sur les cartes.

En tenant compte de ces quatre paramètres, un moteur de traitement forme les

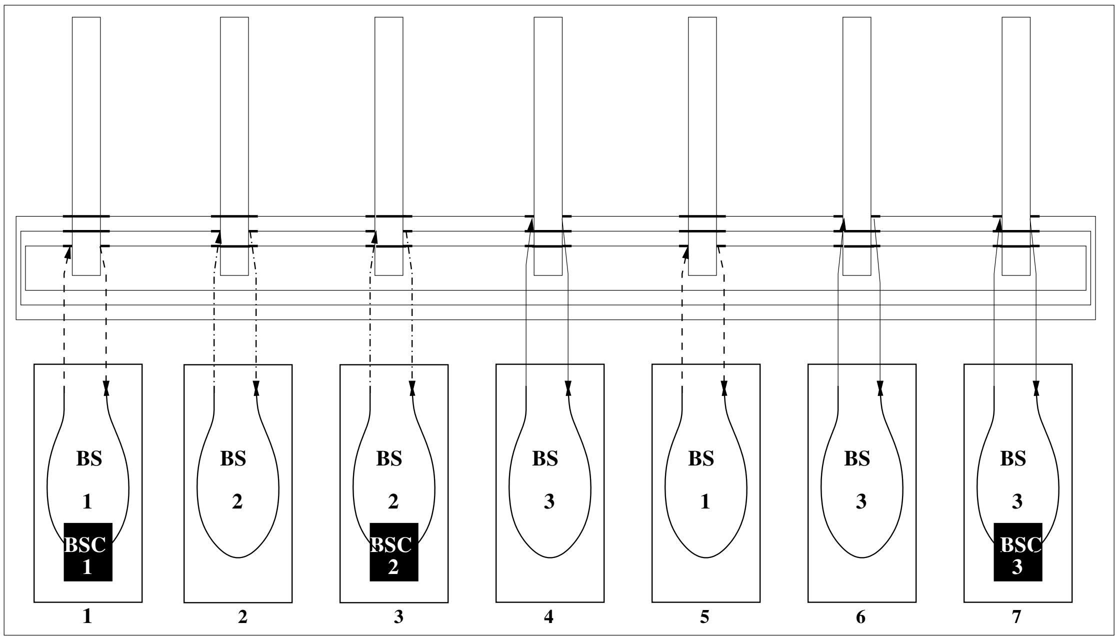

Fig. 4.20 – Exemple de groupements

groupements de cartes en essayant à chaque fois d'utiliser au maximum les possibilités permises par le concepteur concernant les contrôleurs à générer. Si des contrôleurs existants sont présents sur certaines cartes (ou MCMs) prédisposées à une `pseudo-centralisation` et que des modifications réelles sont encore possibles au niveau de leurs câblages, le traitement les considère de façon prioritaire.

Enfin, il à noter qu'en plus du fait qu'elle ne s'applique qu'aux architectures distribuées, la `pseudo-centralisation` reste une étape facultative que le concepteur peut ignorer s'il estime qu'elle ne revêt aucun intérêt pour la testabilité de son système.

4.8.2 Génération des contrôleurs de test pour cartes

Le rôle principal des contrôleurs de test carte est d'atteindre les structures de test sur la carte et de se comporter comme une interface vis à vis du contrôle système. Cela ne les empêche pas d'intégrer dans certains cas une certaine intelligence de test qui leur permet de gérer de façon autonome le test des cartes sur lesquelles ils sont implantés. Deux types de contrôleurs sont donc à distinguer : les contrôleurs minimaux (simples interfaces) adaptés aux architectures centralisées et les contrôleurs intelligents (interface plus intelligence de test intégrée) adaptés aux architectures distribuées.

La diversité des techniques et des structures de test sur une carte nous oblige par ailleurs à définir deux autres classes de contrôleurs carte, à savoir les contrôleurs de test Boundary Scan et les contrôleurs de test intra-circuit tels le SCAN et le BIST. Nous expliquons cela par le fait que le Boundary Scan est avant tout un standard de test dédié au test des cartes, principalement celui de leur interconnexions, et donc il se trouve naturellement à un niveau hiérarchique supérieur au SCAN et au BIST qui sont des techniques de test au niveau circuit. Cela dit, le Boundary Scan peut supporter le test circuit en s'associant aux techniques SCAN et BIST, mais ce n'est pas sa vocation première.

Le contrôleur Boundary Scan: BSC " Boundary Scan Controller "

Ce contrôleur doit être avant tout conforme au standard IEEE1149.1 (paragraphe 3.3.1). Il offrira dans tous les cas le bus du standard comme port pour contrôler les chaînes BS qui existent sur la carte. Dans le paragraphe 3.3.3, nous avons présenté certains travaux de recherche qui ont porté sur le sujet et surtout un nombre important de produits commerciaux qui ont vu le jour. Ils avaient tous pour mission première la conversion des données parallèles provenant du processeur testeur en données série compatibles avec le bus IEEE1149.1. Ils se distinguent par les types d'applications qu'ils visent (carte, systèmes embarqués, etc.) et par le type de bus d'interfaçage (bus PC/AT, bus PCI, bus PCMCIA, etc.). Outre la faculté de conversion parallèle/série et série/parallèle, certains produits commerciaux offrent la possibilité d'intégrer une certaine intelligence de test (génération automatique de vecteurs de test, analyse de résultats, signatures, etc.), mais vu l'aspect confidentiel de ces produits et le fait qu'ils sont généralement accompagnés d'outils logiciel dédiés, il n'était pas possible de les utiliser dans le cadre de notre méthode.

Afin de pouvoir répondre aux différentes configurations qui peuvent se présenter, on a décidé de définir un contrôleur Boundary Scan modulaire (figure 4.21) avec différentes possibilités d'interfaçage (bus processeur SRAM ou bus du standard IEEE1149.5) [MMAA97] [Mar97]. Outre l'ATPG et l'analyseur de signature, ce contrôleur peut intégrer les algorithmes de test BS de telle façon que celui-ci puisse dérouler de façon tout à fait autonome n'importe quel processus de test BS. Ces algorithmes peuvent utiliser des vecteurs de test générés automatiquement par l'ATPG intégré dans le contrôleur, ou bien utiliser des vecteurs pré-générés et stockés dans une mémoire de test interne. Une image de la chaîne BS doit être initialisée dans le contrôleur, cette image comprend les circuits traversés par la chaîne, la taille de leurs registres BS, la taille de leurs registres de données, etc. Ceci confère au contrôleur un aspect programmable. L'intégration de

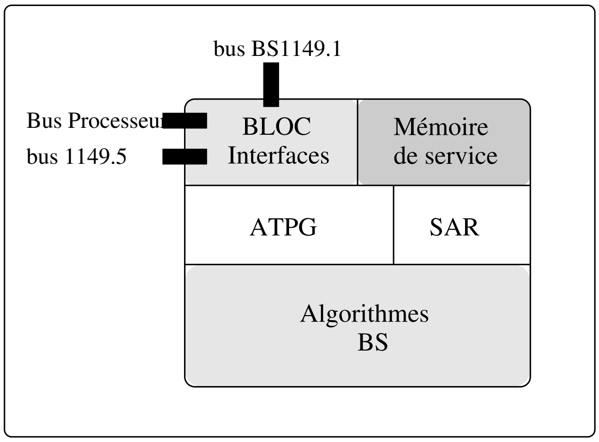

Fig. 4.21 – Contrôleur BS modulaire

ces algorithmes représente l'originalité du contrôleur que nous proposons. L'idée a été motivée par différentes raisons :

- En l'absence de tels algorithmes, le processeur testeur ou le contrôleur système auront à exécuter eux même les différentes phases d'un test BS, passant par le chargement des instructions, le chargement des données et la récupération des résultats, ce qui alourdit considérablement leurs tâches, surtout si le nombre de chaînes à tester est grand.
- Les programmes de test qu'aura à exécuter le processeur de test seront considérablement simplifiés.
- Le temps économisé lors des étapes de contrôle peut être utilisé pour les étapes de diagnostic.

Tous les modules que peut intégrer ce contrôleur ont été définis. Un contrôleur programmable à été synthétisé, il intègre les modules de conversion série/parallèle et parallèle/série, une mémoire de test (taille : 1 KB) et les algorithmes de test BS les plus importants: INTEGRITY, EXTEST, INTEST et RUNBIST.

Ce contrôleur admet deux ports : l'interfaçage avec le processeur testeur se fait à travers un port du type SRAM (Bus d'adresses: 8 bit, Bus de données: 32 bit) et le deuxième port est celui du bus IEEE1149.1. En ce qui concerne la taille d'un tel contrôleur, elle reste étroitement liée à la taille de la mémoire de test. Pour l'exemple synthétisé (mémoire 1KB), elle correspond à 10000 portes.

L'architecture du contrôleur synthétisé est représentée en figure 4.22. Le contrôleur d'adresse a pour rôle l'adressage de tous les registres internes du contrôleur ainsi que celui de la mémoire de test. On note principalement la présence d'un registre de contrôle qui, une fois initialisé, définit le processus à exécuter (simple interfaçage ou algorithme de test spécifique). Un registre d'état "STATUS" est aussi prévu, il contient à tout moment l'état des différents registres du contrôleur.

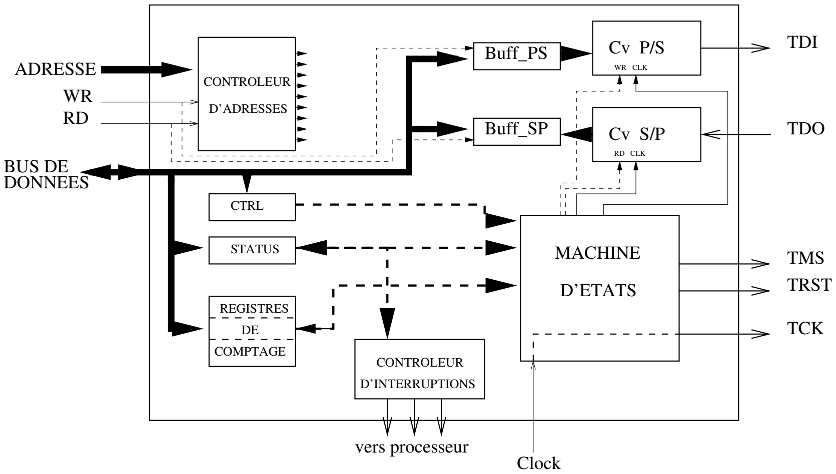

Fig. 4.22 – Architecture du contrôleur BS synthétisé

Par ailleurs, un module de génération d'interruptions en liaison permanente avec le registre d'état a pour rôle d'indiquer au processeur externe:

- la fin d'un processus de test (interruption END_TEST).
- l'échec d'un test d'intégrité (interruption INTEG_FAIL).
- une demande d'écriture de vecteurs de test dans la mémoire de test (pour un algorithme intégré) ou dans le registre de transmission parallèle/série (pour une conversion directe) (interruption IMPORT).
- que le registre de transmission parallèle/série est vide (interruption BUF_PS_EMPTY).

Concernant le bloc "machines d'états ", il est constitué de six machines d'états complètement indépendantes. La première est dédiée à la partie interfaçage (33 états) et les cinq autres respectivement dédiées aux cinq algorithmes de test intégrés dans le contrôleur. Il est à noter que ces machines sont conçues en tenant compte de la structure des machines d'état "TAP" (définies par le standard IEEE1149.1 et contenues dans les circuits BS: voir paragraphe 3.3.1) avec lesquelles elles sont synchronisées. Selon les cas, le signal TMS est généré par l'une des machines d'états.

Intégration du module d'interfaçage

Les deux algorithmes des figures 4.24 et 4.25 complètent l'algorithme principal (figure 4.23) de la partie interface. Dans cet algorithme, tant qu'il n'y a pas de nouvelle opération à effectuer, la machine d'état reste en état d'attente (état 0) correspondant à l'état IDLE des différents contrôleurs TAP des circuits de la chaîne. Dès que le registre de contrôle est initialisé, l'opération à exécuter est décodée (transmission de données ou transmission d'instructions).

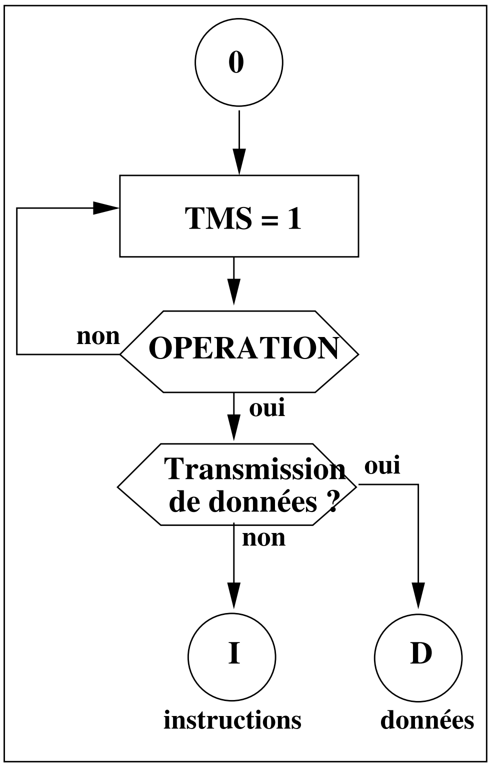

Fig. 4.23 – Algorithme principal du contrôleur

Dans les deux algorithmes de transmission de données et d'instructions nous distinguons essentiellement l'utilisation des trois variables : "N-bit", "N-vect" et "compteur" qui représentent respectivement le nombre de bits de données ou d'instruction, le nombre de vecteur(égal à 1 dans le cas d'une transmission d'instruction) et le nombre de cycles d'horloge entre deux vecteurs successifs. Les signaux SHFT-PS et SHFT-SP sont les signaux de décalage des convertisseurs parallèle/série et série/parallèle.

Deux registres BUF-PS et BUF-SP sont disponibles. Le premier a pour rôle d'alimenter le convertisseur P/S et le second permet de stocker le résultat de la conversion S/P. Les signaux IMPORT, BUF-FULL et END-TEST représentent les 3 bits de poids faible du registre STATUS. BUF-FULL est à '0' lorsque BUF-PS est vide et passe à '1' dès que le processeur y écrit une information. Ce signal repasse à '0' lorsque le contenu de BUF-PS est transmis au convertisseur. Le signal IMPORT est activé lorsque le gestionnaire demande au processeur d'écrire une information dans BUF-PS et il est désactivé dès que l'information est écrite. END-TEST à '1' indique la fin d'une transmission d'instruction ou de données.

Nous remarquons une légère différence entre les deux algorithmes de transmission de données et d'instructions. En effet, dans le cas d'instructions, l'algorithme n'est parcouru qu'une seule et unique fois à la différence du cas des données où il y a plusieurs vecteurs de test à transmettre. Il est aussi à signaler que deux états supplémentaires sont nécessaires au début de l'algorithme de transmission d'instructions, afin de diriger les machines d'état des contrôleurs TAP des différents circuits de la chaîne vers l'état Select-IR-Scan.

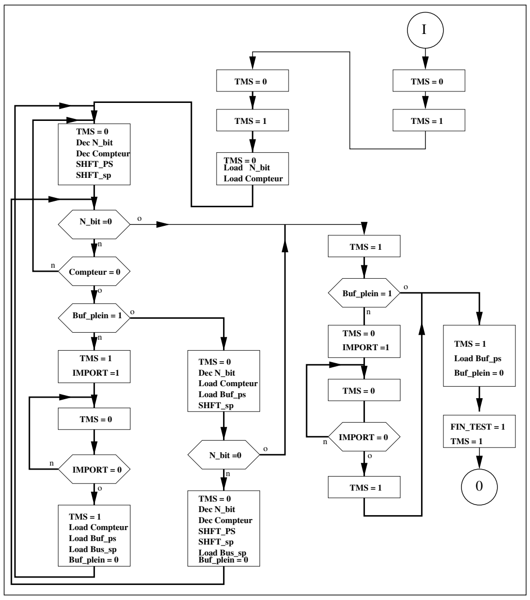

Fig. 4.24 – Algorithme de la transmission d'instructions

Intégration de l'algorithme du test d'intégrité :

Comme il a été mentionné dans le paragraphe 3.3.1, c'est un test qui permet de détecter si un circuit est abîmé, mal placé ou même inexistant dans la chaîne. En général, c'est le premier algorithme de test BS à implémenter. Pour cet algorithme de test, comme pour tous les autres algorithmes qui seront présentés ultérieurement, les caractéristiques de la chaîne BS à tester (nombre de circuits, tailles des registres BS, etc.) sont initialisées dans la mémoire de test du contrôleur avant tout autre opération.

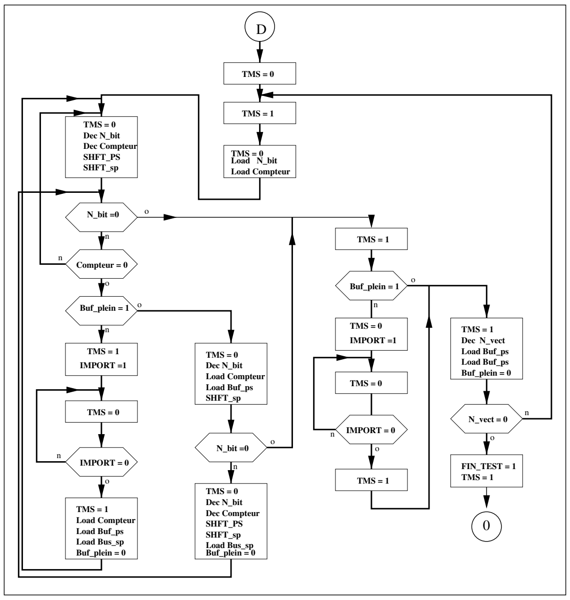

Fig. 4.25 – Algorithme de la transmission de données

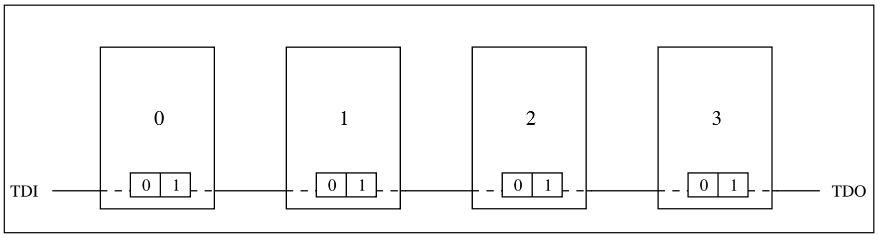

Fig. 4.26 – Test d'intégrité

Au niveau de l'état CAPTURE-IR des contrôleurs TAP des différents circuits de la chaîne, une donnée est inscrite dans les registres d'instructions, cette donnée a toujours ses deux bits de poids le plus faible à la valeur "01". L'idée est de lire en série toutes les données inscrites dans les registres d'instructions (figure 4.26) des circuits de la chaîne et de vérifier ensuite si les deux bits sont à leur place. Pour intégrer cet algorithme, nous avons prévu une mémoire qui contient la taille des différents registres d'instructions et un compteur qui est initialisé à chaque fois au numéro du circuit à vérifier. L'algorithme commence par tester le registre du dernier circuit en vérifiant les deux bits de poids faible, s'il n'y a pas de problème il termine en décalant le restant des bits, avant de tester le prochain circuit et ainsi de suite jusqu'à la vérification de toute la chaîne. Si toute la chaîne ne contient aucun problème alors une interruption de fin de test "END_TEST" est activée, si en revanche une faute est détectée, alors une interruption "INTEG-FAIL" est activée avec l'interruption "END_TEST" et le numéro du circuit défaillant est inscrit sur le bus. La machine d'état relative à cet algorithme contient 20 états.

Intégration de l'algorithme de test interne :

Le test interne permet de tester la logique interne d'un circuit de la chaîne BS. En plus des caractéristiques de la chaîne BS, le processeur doit charger dans la mémoire du contrôleur les vecteurs de test et indiquer le numéro du circuit à tester ainsi que le nombre de vecteurs nécessaires pour le test.

Le contrôleur commence par mettre en mode BYPASS les circuits non concernés par le test et envoyer une instruction SAMPLE/PRELOAD au circuit à tester afin de précharger un vecteur d'initialisation. Une fois cette première étape achevée, une deuxième étape commence et consiste à mettre le circuit en mode INTEST, ce qui sélectionne le registre BS et le connecte entre TDI et TDO. Le contrôleur envoie les vecteurs de test (stockés au préalable dans une mémoire de test) un à un, les applique (en contrôlant le signal TMS) à la logique interne du circuit. En décalant un nouveau vecteur de test dans le registre BS, le contrôleur récupère par la même occasion le vecteur résultat du dernier échantillon appliqué et le stocke dans la mémoire de test. Une fois tous les vecteurs de test envoyés, le contrôleur renvoie au circuit une instruction SAMPLE/PRELOAD pour charger un vecteur " SAFE " et éviter ainsi tout risque de conflits avec les autres composants de la carte. Durant toute la durée du test, les circuits BS non concernés par le test en question restent en mode BYPASS.

Le contrôleur peut inviter le processeur externe de test à récupérer des résultats de test dans le but de libérer de la mémoire ou à envoyer d'autres vecteurs de test si le stock a été épuisé, par l'intermédiaire d'interruptions.

Intégration de l'algorithme du test externe :

Le test externe permet de tester soit les interconnexions entre les différents circuits BS d'une carte soit un ensemble de circuits non BS entourés par la chaîne BS (les "CLUSTERS"). Le processeur aura à initialiser dans le contrôleur les caractéristiques de la chaîne, les circuits concernés par le test et les vecteurs de test.

Le contrôleur commence par mettre en mode BYPASS les circuits qui ne sont pas concernés par les interconnexions à tester, et par envoyer aux autres des instructions SAMPLE/PRELOAD pour charger le premier vecteur de test. Il est à noter que les stimuli concernent seulement les sorties des circuits impliquées dans les interconnexions (têtes d'interconnexions). Un circuit ayant une ou plusieurs sorties comme tête d'interconnexion est dit circuit " DRIVER ". Les résultats sont captés par les entrées des circuits vers lesquels les interconnexions aboutissent (queues d'interconnexions).

La deuxième étape consiste à envoyer des instructions EXTEST aux circuits concernés par le test, tout en laissant les autres en mode BYPASS. Le contrôleur transmet les vecteurs de test sur la chaîne tout en récupérant les résultats des précédents tests qui sont stockés en mémoire.

Pour que le contrôleur soit complètement autonome, il doit connaître les circuits concernés par le test d'interconnexions. Pour cela on a défini un registre "SWITCHS" qui doit être initialisé dès le départ par le processeur externe de test. Chaque bit de ce registre est attribué à un circuit et définit son état : s'il est à un, le circuit en question est concerné par le test sinon il ne l'est pas. Un deuxième registre nommé "DRIVERS" permet de préciser quels sont les circuits dont les sorties représentent des têtes d'interconnexions. Ce deuxième registre permet d'économiser de la mémoire dans la phase de préchargement de vecteurs de test, en effet, le processeur externe n'aura à charger dans la mémoire de test du contrôleur que les vecteurs relatifs aux circuits "DRIVER" et évitera ainsi de charger des vecteurs dans les autres circuits qui sont inutiles en phase de stimulation. Ce deuxième registre augmente la complexité du contrôleur ainsi que sa taille, mais l'économie qu'il permet plus tard en matière de mémoire de test reste plus attrayante. A la fin de chaque processus de test, des vecteurs "SAFE" sont chargés dans les circuits de la chaîne, comme dans le cas du test interne.

Qu'il s'agisse d'un test d'interconnexions ou un test de "CLUSTER", les étapes suivies par le contrôleur sont les mêmes. La différence réside plutôt au niveau du diagnostic où le test de "CLUSTER" reste largement le plus complexe.

Pour le contrôleur synthétisé, les deux registres "SWITCHS" et "DRIVERS" ont chacun une taille de 64 bit, ce qui limite le nombre de circuits BS sur une carte à 64.

Intégration de l'algorithme du test intégré :

Cet algorithme permet d'exécuter l'instruction optionnelle RUNBIST pour l'élaboration du test intégré (" BIST ") d'un des circuits de la chaîne BS. Ceci implique évidemment que le circuit en question contient déjà les fonctionalités de test requises à cet effet.

Le contrôleur se comporte au départ comme dans le cas d'un test interne: Tous les circuits en BYPASS sauf celui à tester, dans lequel est initialisé l'instruction SAMPLE/PRELOAD. Un premier vecteur de données est chargé dans le registre BS, il sert principalement à définir les signaux des sorties des circuits qui doivent rester inchangés tout au long de l'exécution de l'instruction RUNBIST. Les signaux d'entrées non "clocked" peuvent être chargés de valeurs constantes pour éviter éventuellement les conflits.

Après avoir chargé l'instruction RUNBIST dans le circuit cible, le contrôleur contrôle le signal TMS dans le but de diriger la machine d'état TAP vers l'état "run-test/Idle" dans lequel les opérations d'auto-test accessibles à travers l'instruction RUNBIST peuvent s'exécuter.

L'instruction RUNBIST place entre les deux pins TDI et TDO un registre de données spécifique (qui peut, dans certains cas, être le registre BS lui-même) dans lequel les résultats du test sont chargés.

Afin de pouvoir exécuter un tel algorithme, le contrôleur doit connaître au préalable la durée exacte du test. En d'autres termes, le nombre de coups d'horloges à fournir durant l'état " run-test/Idle " doit être initialisé avec les caractéristiques de la chaîne (à l'instar des autres algorithmes) avant de lancer le test.

Le contrôleur BTC " Board Test Controller "

C'est le contrôleur qui assure le test des circuits contenant des fonctionnalités autre que le Boundary Scan (principalement SCAN ou BIST). Ce genre de contrôleur est nécessaire lorsqu'il n'est pas possible d'atteindre les fonctionalités de test internes à un circuit à travers un bus IEEE1149.1, puisque tous les circuits testables ne contiennent pas nécessairement du BS, et même si c'était le cas, rien n'assure que le concepteur ait prévu un lien entre les structures BS et les autres structures de test tel le SCAN ou le BIST.

Comme le contrôleur BSC précédemment décrit, le BTC peut être intelligent ou minimal. Dans sa forme minimale, il doit jouer le rôle d'interface entre le processeur de test externe ou le contrôleur système de test et les circuits à tester, la tâche principale étant la conversion parallèle/série et série/parallèle des données. Dans la version intelligente, l'idée est d'y incorporer des fonctionnalités supplémentaires telles un ATPG, une mémoire de test et un analyseur de signature. Pour ce qui des algorithmes de test, comme il n'y a pas de standard fixe (à la différence du contrôleur BSC), deux scénarios sont envisageables :

- Charger les routines de test à exécuter dans la mémoire du contrôleur, ce qui lui confère un aspect programmable. Ces routines auront accès aux différentes fonctionnalités du contrôleur : convertisseurs, ATPG, mémoires, etc. Ce scénario implique la définition et la conception d'un contrôleur universel programmable qui peut être disponible en bibliothèque. Le premier avantage d'un tel scénario est la flexibilité du contrôle, en effet il sera toujours possible de changer ou de modifier l'algorithme de test. Le deuxième avantage est la réutilisation (" REUSE ") : ce genre de contrôleur peut être utilisé sur n'importe quelle carte, quelle que soit sa configuration.

– Fournir au concepteur une description synthétisable du contrôleur universel, auquel il pourra greffer ses algorithmes de test ou d'autres éléments de test et éventuellement retirer certaines fonctionnalités, s'il le juge nécessaire, pour enfin synthétiser le tout et obtenir un contrôleur intelligent programmable, dans le sens où l'utilisateur aura à initialiser avant chaque processus de test les caractéristiques de la carte et des circuits qu'il va contrôler. L'avantage principal est la rapidité de test et l'optimisation en taille. Dans certains cas, le concepteur peut même implanter au sein du contrôleur les caractéristiques de la carte et des circuits à tester pour obtenir un contrôleur dédié. Ce scénario est surtout recommandé dans le cas des MCMs où la taille du contrôleur doit être minimale.

Dans le cas où un contrôleur BS (BSC) est déjà prévu sur la même carte, il est toujours intéressant d'implanter le BSC et le BTC en un même circuit. En effet l'idée est de greffer sur la version synthétisable du BSC qu'on a déjà défini certains modules qui pourront utiliser ses fonctionnalités et se comporter comme un BTC. Un projet de générateur de contrôleurs BSC ou BTC ou les deux combinés (minimaux ou intelligents) est à l'étude.

Interfaçage avec le contrôle système

Rappelons que jusqu'ici, mis à part sa nature (logiciel ou matériel), aucune décision n'a été prise concernant le contrôleur système. Dans le cas où il s'agit d'un contrôleur matériel, on ne sait pas s'il s'agira d'un contrôleur pour le bus IEEE1149.5 ou d'un contrôleur pour un autre type de bus. Cela implique que la décision concernant l'interfaçage avec le contrôle système ne peut pas encore se faire à ce stade du traitement.

4.8.3 Agencement des structures de test

C'est l'étape durant laquelle les différentes structures de test, sur lesquelles le choix du concepteur s'est porté, seront reliées pour former l'architecture de test au niveau de la carte. Ceci passe par la réservation d'emplacements pour les contrôleurs, la réservation d'emplacements pour les éventuels registres BS et la construction des chaînes Boundary Scan et SCAN.

En ce qui concerne les contrôleurs, tout dépend de l'état d'avancement de la conception de la carte. En effet, si la carte est déjà existante, l'ajout d'un circuit de contrôle peut s'avérer parfois un exercice difficile (cas d'une carte encombrée). De toutes façons, ces problèmes d'encombrement doivent être pris en compte avant de prendre des décisions concernant la génération des contrôleurs de test. Ceci est d'autant plus évident dans le cas des MCMs où rien ne peut être ajouté à un MCM existant.

Tout se passe de la même façon avec les registres BS externes dont l'emplacement est a priori garanti dès le départ par le concepteur (voir paragraphe 4.4).

La construction de chaîne BS consiste simplement à relier les broches TDI et TDO des différents circuits BS entre eux. Ceci doit quand même être effectué de façon à respecter l'emplacement des circuits sur la carte. Dans le cas où des MCMs contenant du BS existent sur la carte, alors deux possibilités sont envisageables (figure 4.27) :

- Le MCM contient déjà un contrôleur BS (MBSC dans la figure): dans ce cas ce dernier est directement connecté au "slot" de la carte (sur le bus système) tout comme le contrôleur de la carte elle même. Dans ces cas-là, on a fréquemment une architecture globale de test distribuée.

- Le MCM ne contient aucun contrôleur: dans ce cas la chaîne BS du MCM est reliée à la chaîne BS de la carte. Dans ce cas, on a plutôt une architecture globale de test centralisée.

En ce qui concerne les chaînes SCAN (internes au circuits), deux possibilités sont envisageables:

- relier les chaînes SCAN des différents circuits de la carte entre elles (comme s'il s'agissait d'une chaîne BS). Cette façon de faire a comme principal avantage sa simplicité et la facilité de sa réalisation. Toutefois, elle présente un inconvénient majeur, qui est l'impossibilité de tester un circuit tout seul. Le testeur doit toujours faire transiter ses vecteurs de test à travers tous les circuits (qui doivent être en mode test). Cette contrainte limite considérablement les performances de l'architecture de test de la carte.

- Afin de pouvoir sélectionner le ou les circuits SCANables à tester sur une carte, on est obligé de passer par l'utilisation de multiplexeurs pour aiguiller les données de test (figure 4.28). Pour N circuits SCANables, il faut N-1 multiplexeurs 2 vers 1, 1 multiplexeur N vers 1, 1 démultiplexeur 1 vers N sorties et un décodeur à 1 parmi N-1 sorties. Cette deuxième méthode, bien que résolvant le problème de

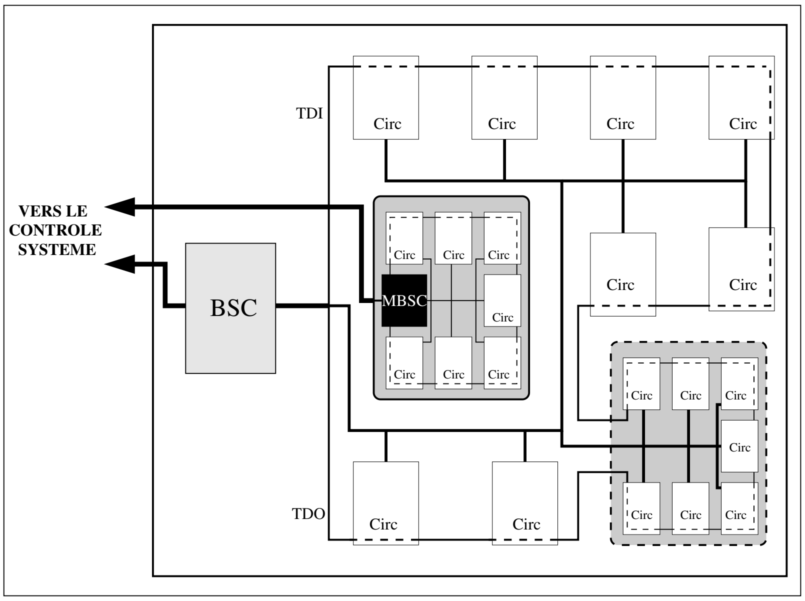

Fig. 4.27 – Construction de la chaîne BS

la première (flexibilité du test), présente l'inconvénient d'être difficile à mettre au point, vu l'encombrement qu'elle peut engendrer sur la carte. Dans certains cas elle est impossible à mettre en oeuvre (pas d'emplacement vacant sur la carte). Ce problème d'encombrement peut être minimisé si toute la partie multiplexage et décodage est intégrée à l'intérieur du contrôleur de test dédié à la carte. Ceci doit être considéré lors de la génération du contrôleur (voir la partie " génération de contrôleur BTC " au paragraphe 4.8.2). Si le contrôleur est une simple interface ou s'il est déjà existant (rien ne peut y être ajouté), on est obligé d'utiliser des éléments discrets.

4.9 traitement final au niveau système

A ce niveau du traitement, des décisions ont été prises concernant la testabilité des circuits, la testabilité au niveau des cartes et les contrôleurs carte que ce soit BSC ou BTC. Cependant, dans le cas où on doit générer ces contrôleurs, une décision définitive concernant la nature du contrôleur système est nécessaire. En effet, il reste à définir le bloc d'interfaçage avec le contrôle système. En revanche, dans le cas où il a été décidé que le contrôleur système sera de type logiciel, l'interface d'un contrôleur carte est fixée dès le départ (de type SRAM) et rien n'empêche sa génération avant la définition du

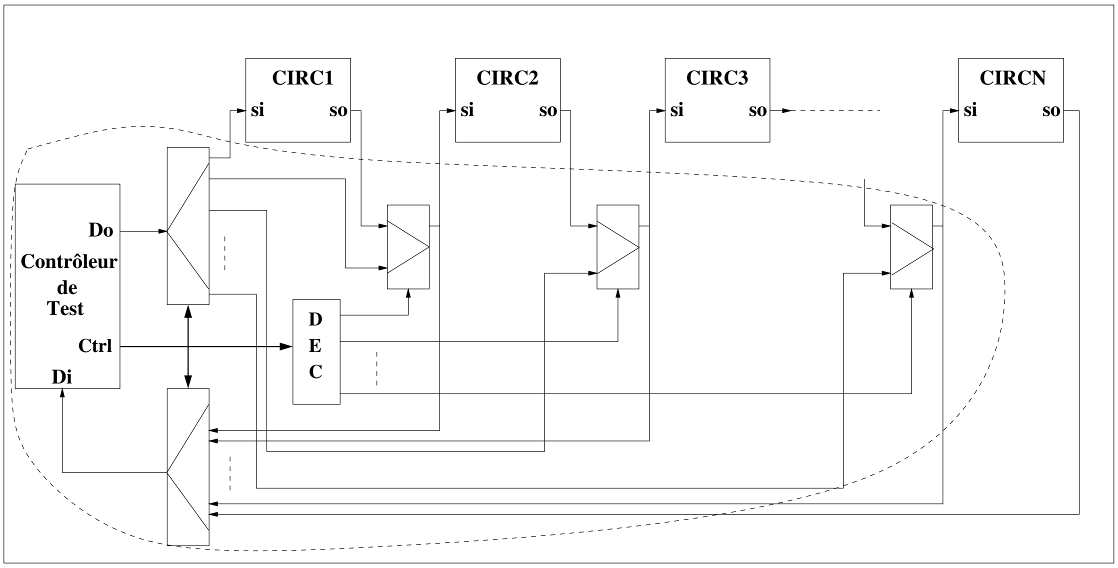

Fig. 4.28 – Utilisation des multiplexeurs pour aiguiller les données vers les circuits SCAN

contrôleur système.

Cela dit, la tâche principale à ce niveau du traitement est la définition de la nature du contrôleur système qui peut se baser sur le standard IEEE1149.5 (voir paragraphe 3.3.1) ou sur une extension du standard IEEE1149.1 (voir paragraphe 3.3.3), ou encore être défini par le concepteur.

4.9.1 Génération du contrôleur système

C'est la tâche principale du traitement au niveau système. Une fois achevée, elle doit permettre de finaliser toutes les décisions concernant la testabilité du système. Comme nous l'avons déjà indiqué, le contrôleur système a le rôle de gérer le test des différentes cartes qui constituent le système, il peut être implémenté en logiciel ou en matériel et il peut être minimal (cas d'une architecture de test distribuée) ou intelligent (architecture centralisée).

Dans le cas où il a été décidé que le contrôle système est du type logiciel, c'est le processeur système qui a la tâche d'exécuter les algorithmes de contrôle de test. Deux configurations peuvent se présenter :

- L'architecture globale de test est distribuée, et dans ce cas le logiciel dédié au contrôle de test doit pouvoir communiquer avec les contrôleurs carte à travers le bus système. Ces derniers sont généralement munis d'interfaces du type SRAM, ce qui ne pose aucun problème au processeur système pour communiquer avec eux.

- L'architecture globale est centralisée : nous distinguons trois cas:

  - Toutes les cartes sont munies de contrôleurs minimaux (jouant le rôle de simples interfaces). Ces contrôleurs sont normalement munis d'interfaces du type SRAM connectées au bus système ce qui permet au processeur de communiquer avec eux sans difficulté.

  - Aucune carte testable n'est munie d'un contrôleur minimal. Ce qui va poser problème dans ce cas est l'émission et/ou la réception des données (ou commandes) de test en série entre le processeur système et les modules testables sur les cartes. Rappelons que généralement les données de test sont sérielles (BS: émission/réception, SCAN: émission/réception, BIST: principalement la réception des résultats) et que les circuits testables d'une même carte sont habituellement chaînés entre eux selon le type de test qu'ils adoptent (chaînes BS, chaînes SCAN etc.).

    En général, si le processeur n'a pas la possibilité de traiter des informations sérielles directement sur le bus, il faut prévoir un module d'interfaçage parallèle/série et série/parallèle en matériel pour ce faire. Le concepteur doit prévoir de chaîner entre elles toutes les chaînes de même type à travers les "slots" en y réservant des couples de broches chaînées entre elles (BS : TDI-TDO, SCAN : SI-SO ou autre) ou autrement si les contraintes d'implémentation réelle le permettent. Dans le cas où il y a plus d'un seul type de chaînes de test, le module d'interfaçage à prévoir doit fournir un port sériel pour chaque type de test ou bien le concepteur doit prévoir un dispositif de multiplexage pour aiguiller les données de test vers les chaînes concernées (figure 4.29: deux types de chaînes : BS et SCAN). Dans le cas où le bus système est un bus PCI et les cartes testables sont exclusivement BS, comme le processeur système a la possibilité de gérer directement sur le bus les signaux BS, il n'y a aucun besoin d'utiliser des modules d'interfaçage: Toutes les cartes BS sont chaînés entres les deux broches TDI et TDO disponibles sur bus.

  - Si au moins une carte testable est munie d'un contrôleur minimal, il est très utile, quand les contraintes matérielles le permettent, de relier la chaîne qu'il contrôle à celles des autres cartes non munies de contrôleurs (figure 4.30). Ceci n'augmente pas énormément les performances temporelles de l'architecture de test (vu qu'elle reste toujours centralisée avec un contrôle système logiciel),

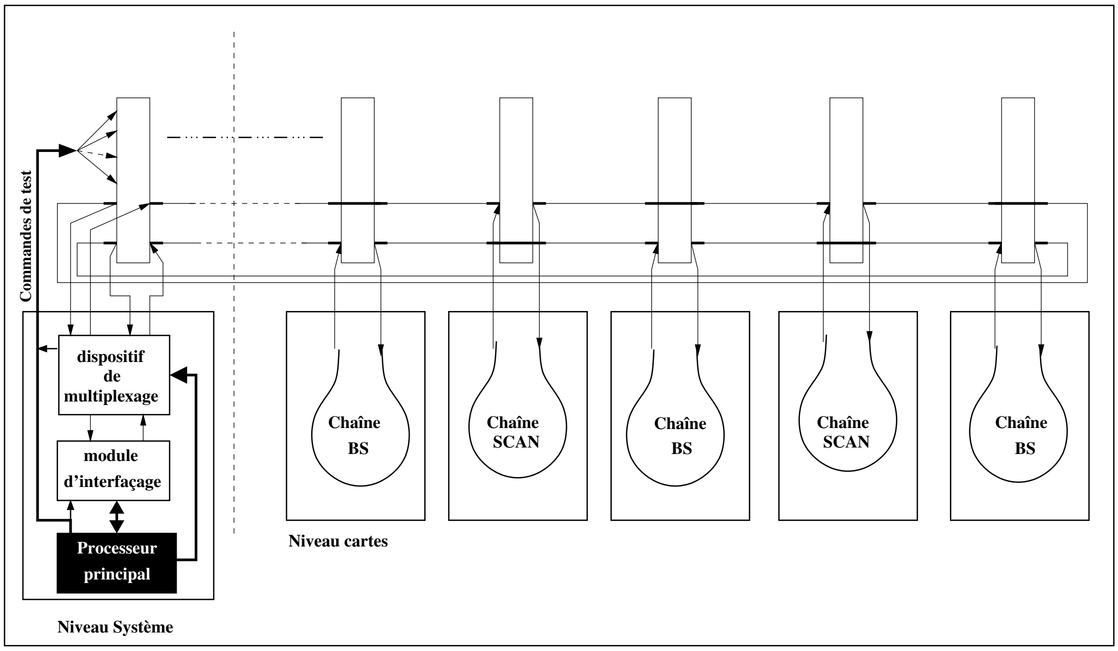

Fig. 4.29 – Contrôle système avec dispositif de multiplexage

mais permet en revanche, grâce à un programme de test bien élaboré, d'alléger la tâche du processeur principal. Si les contrôleurs minimaux existants sur certaines cartes ne peuvent pas assurer l'interfaçage avec toutes les chaînes du système, il faut prévoir un module d'interfaçage additionnel pour les chaînes restantes.

Dans le cas où le contrôle système doit être implémenté en matériel, Le concepteur a le choix entre trois façons de le définir : soit en se basant sur le standard IEEE1149.5, soit en étendant les possibilités du bus du standard IEEE1149.1 au niveau système, soit encore en concevant un contrôleur dédié à l'application et ne se basant sur aucun des deux standards. En choisissant la troisième solution, le concepteur a toute liberté dans les choix à faire, puisqu'il n'a pas de contraintes particulières relatives à un standard. Le choix de l'une des deux premières solutions peut être motivé par la nature des décisions déjà prises concernant les différents modules du système (type du test intégré, type de l'architecture de test, etc.).

Le contrôleur SBSC "System Boundary Scan Controller"

Dans le cas où toutes les cartes testables du système sont exclusivement BS et que l'architecture globale de test est centralisée, il est fort intéressant d'utiliser un contrôleur

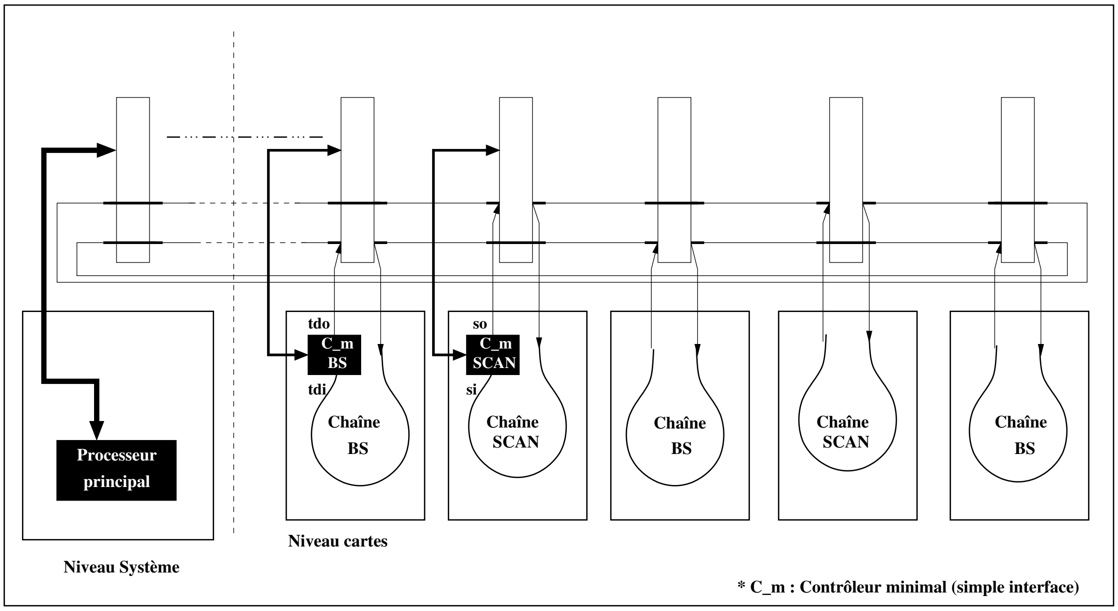

Fig. 4.30 – Présence de contrôleurs minimaux sur certaines cartes

système basé sur l'exploitation du bus BS IEEE1149.1. Une telle approche a été présentée dans [Bha91] (voir paragraphe 3.3.3).

Ce contrôleur SBSC est très semblable au contrôleur carte intelligent (BSC) que nous avons présenté dans le paragraphe 4.8.2. La seule différence est que le SBSC doit être capable de fournir plus d'un port lorsqu'il y a plus d'une chaîne BS dans le système. Un dispositif de multiplexage (pour aiguiller les données de test vers l'une des chaînes) peut être intégré :

- dans le contrôleur lui-même lorsque ceci est possible. Toutefois, le nombre de ports BS sera figé. Si l'application doit évoluer par présence d'une nouvelle chaîne BS dans le système, il n'y a pas de moyens d'assurer le contrôle de la nouvelle extension avec le même contrôleur.

- sous forme d'éléments discrets à l'extérieur du contrôleur. Ceci permet de garder une certaine flexibilité, très utile en cas d'évolution de l'architecture de test.

Dans un cas comme dans l'autre, le SBSC doit intégrer en plus une certaine logique programmable (si le SBSC est programmable) ou micro-programmée (si le SBSC est dédié à l'application et donc complètement autonome) pour la commande du dispositif de multiplexage (figure 4.31). D'autre part, le SBSC présenté doit obligatoirement fournir une interface au bus processeur du système sur lequel il est implanté (en général une interface du type SRAM).

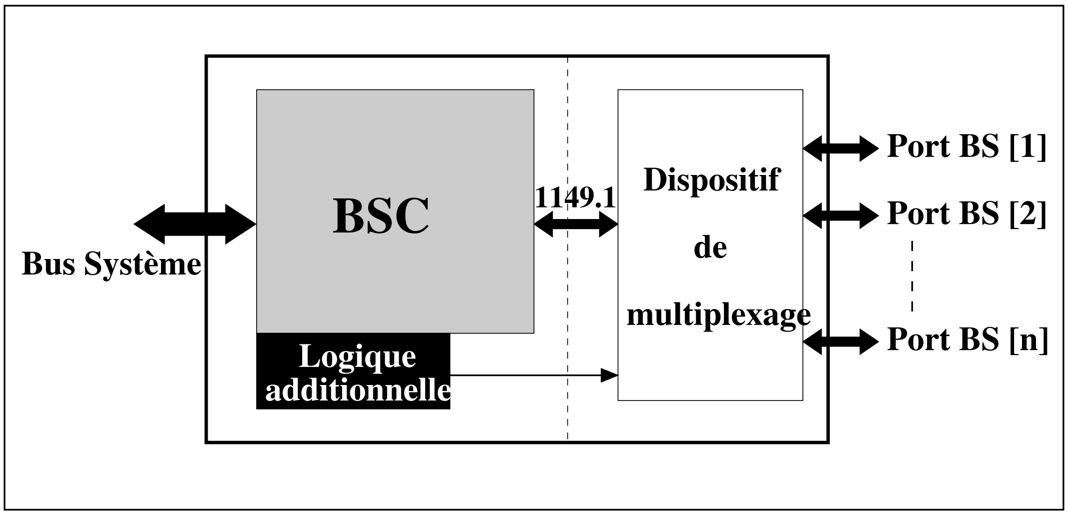

Fig. 4.31 – Le contrôleur SBSC

Le contrôleur du bus IEEE1149.5

Le standard IEEE1149.5 (voir paragraphe 3.3.1) définit un bus sériel de test et de maintenance permettant d'intégrer, dans un système, un ensemble de modules eux mêmes testables. Pour contrôler ce bus, nous nous sommes proposés de définir et développer un module de gestion et de contrôle de ce bus selon les normes fixées par le standard IEEE1149.5, ainsi que les interfaces permettant de le relier à d'autres bus de test (ex. le bus IEEE1149.1). Ce module peut intégrer des algorithmes de test (comme cela était le cas avec le contrôleur BSC intelligent) pour les exécuter de façon plus ou moins autonome. Ceci peut être très utile dans le cas d'une architecture globale de test centralisée.

Ce module se base sur un modèle de communication synchrone maître/esclave comme le montre la figure 4.32. Le maître présente deux ports: un port de type SRAM pour communiquer avec le processeur de test (qui peut être le processeur système lui-même) et un deuxième port pour le bus IEEE1149.5. Les modules esclaves, qui eux sont en relation avec les différents modules à tester (les cartes par exemple), viennent se connecter sur le bus IEEE1149.5. Chaque esclave est constitué de deux sous-modules:

– L'esclave générique, qui a une architecture fixe indépendante du module à tester.

– L'élément d'interfaçage entre l'esclave générique et le module à tester. Cette interface dépend de la nature du module à tester. Par exemple, pour une carte BS, il faut une interface capable de convertir les données de test traitées par l'esclave au format BS IEEE1149.1.

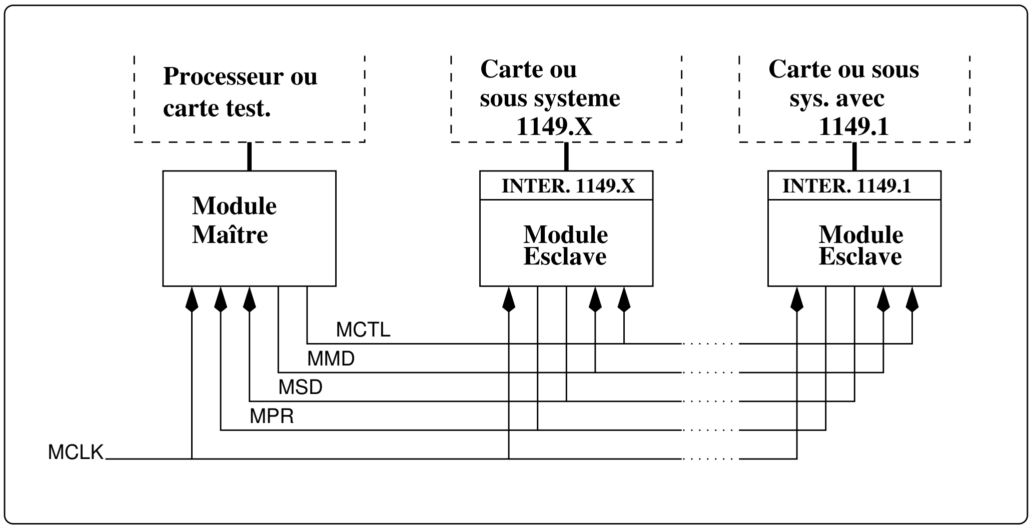

Fig. 4.32 – Le bus 1149.5

Rappelons que le bus IEEE1149.5 est sériel, qu'il peut supporter jusqu'à 250 modules esclaves et que la transmission des données de test se fait par paquet. Les signaux qui composent ce bus sont: le signal de transmission de données du maître (MMD), le signal de transmission de données de l'esclave (MSD), le signal de contrôle généré par le maître (MCTL), le signal d'horloge (MCLK) et le signal de pause généré par l'esclave (MPR) (ce signal est facultatif mais recommandé par le standard, il est très utile dans le cas d'esclave(s) lent(s)).

Les deux architectures du maître et de l'esclave générique ont été définies en tenant compte des contraintes et des recommandations du standard IEEE1149.5. Une première implémentation en C a été réalisée et simulée. Une description VHDL a été ensuite synthétisée à l'aide des outils de la chaîne ALLIANCE [LA98] et simulée avec l'outil de simulation ASIMUT de la même chaîne. Il est à noter que le même outil ASIMUT a été utilisé pour la simulation des modèles C ce qui nous a permis d'utiliser les fichiers de test pour la description VHDL. Dans ce qui suit nous présentons les architectures des deux modules synthétisés :

Le module Maître

L'architecture du maître est représentée par la figure 4.33. Les principaux blocs de l'architecture sont :

– `L'automate de contrôle :` c'est le coeur du maître, il est responsable du contrôle des étapes d'émission (réception) des données ou commandes de test vers les (en

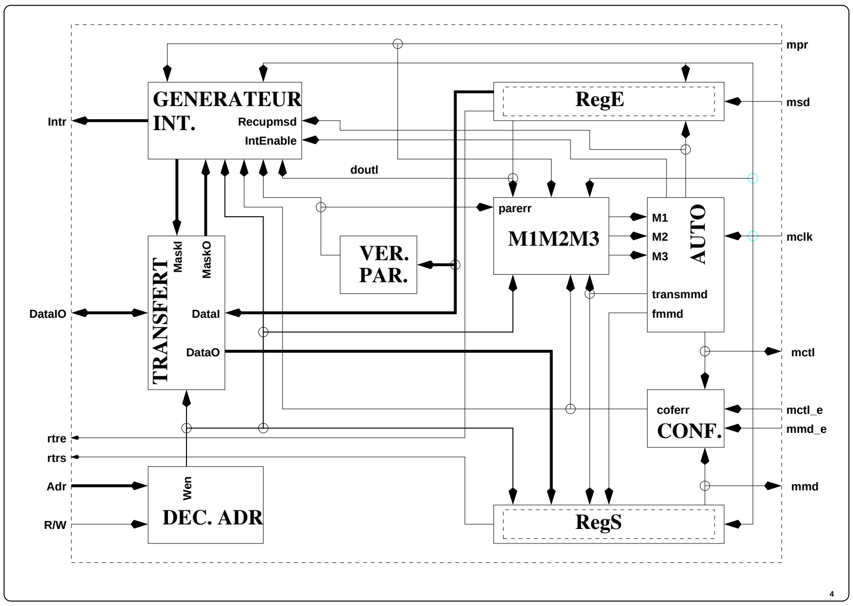

Fig. 4.33 – Le module maître

provenance des) esclaves. Il est contrôlé par trois variables d'états M1, M2 et M3 qui sont générées dans le maître lui-même. Ces variables ont notamment pour rôle d'indiquer la présence de conflits sur le bus (M1), la fin de transmission (M3) ou encore la réclamation d'une pause par l'un des esclaves (M2).

- `Le bloc RegS :` commandé par l'automate, il est en charge d'émettre les données en série vers les esclaves à travers MMD. Il fournit le signal MMD.

- `Le bloc RegE :` commandé par l'automate, il est en charge de réceptionner les données en provenance des esclaves. Il reçoit le signal MSD.

- `Le block de contrôle de parité :` il a pour rôle de détecter une éventuelle erreur de parité dans les paquets en provenance des esclaves. En cas de détection d'erreur, il le signale au générateur d'interruptions qui le signale à son tour au processeur testeur sous forme d'interruption. Il est à noter que dans un tel cas l'automate n'arrête pas la transmission.

- `Le bloc de détection de conflits :` étant donné que le seul maître de l'architecture IEEE1149.5 est susceptible de tomber en panne, le standard a prévu qu'au moins l'un des esclaves soit muni de certaines fonctionalités lui permettant de prendre le relais et de se comporter comme maître en cas de panne du vrai maître. Néanmoins, l'esclave en question peut lui-même connaître des défaillances comme par exemple prendre le contrôle du bus alors que le maître principal est en parfait état de fonctionnement. Ceci déclenche inévitablement des conflits sur le bus et plus précisément au niveau des signaux MMD et MCTL. D'où l'utilité de ce bloc qui a pour rôle de détecter d'éventuels conflits pour les signaler à l'automate via la variable M1 et au processeur de test via le générateur d'interruption.

- *Le générateur d'interruptions :* il est en charge d'avertir le processeur de test (via des interruptions) des événements qui peuvent survenir :
  - une erreur de parité,
  - un conflit sur le bus au niveau de MMD ou MCTL,
  - un esclave défaillant,
  - la réclamation d'une pause par l'un des esclaves,
  - un paquet en provenance d'un esclave est disponible à la lecture. Il est toujours possible de masquer une ou plusieurs interruptions en programmant le masque à partir du processeur.
  - *Le décodeur d'adresse :* il permet d'atteindre les registres internes du maître. Nous l'avons intégré comme bloc à part parce qu'il est prévu que le maître puisse inclure d'autre fonctionnalités telles les algorithmes de test qui nécessitent un adressage pour leur contrôle.
  - *Le bloc transfert de données :* c'est le bloc par lequel transitent toutes les données venant ou partant vers le processeur. Son bus de données est de 17 bits.

Le module esclave générique

Ce module est beaucoup plus complexe que le maître, étant donné qu'en plus de la tâche de transfert de données vers les éléments à tester et la récupération des résultats, il est en charge d'exécuter un certain nombres de commandes, en provenance du maître. Ces commandes sont présentées dans le standard, et nous pouvons les classer en quatre catégories :

- Les commandes de lecture d'état que le processeur envoie à l'esclave via le maître, pour lire l'état de l'application et de l'esclave lui-même.
- Les commandes de programmation qui permettent d'initialiser ou de modifier les paramètres internes de l'esclave.

– Les commandes liées à l'application, telles son initialisation, son lancement ou encore son arrêt.

– Les commandes diverses dans lesquelles nous pouvons inclure toutes les fonctions ajoutées par le concepteur. En effet, le standard prévoit une plage spéciale de commandes réservée à l'usage du concepteur.

Il est à signaler que pour l'esclave générique que nous avons synthétisé, seules des commandes des trois premières classes ont été intégrées. D'autres commandes sont à prévoir pour contrôler les éléments d'interfaçage qui peuvent se greffer sur l'esclave générique.

L'architecture de l'esclave générique présentée dans la figure 4.34 est constituée de quatre blocs principaux :

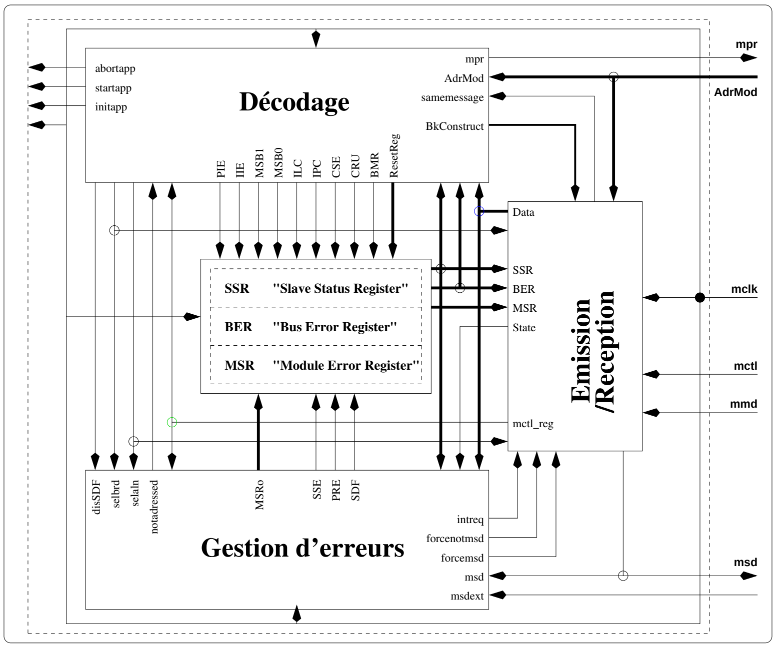

Fig. 4.34 – Synoptique de l'esclave générique

Le bloc d'émission/réception : Sa tache principale est l'émission et la réception des données et des commandes sur le bus IEEE1149.5. De plus, il est en charge de la construction des paquets à transmettre éventuellement au maître et du calcul de leur parités.

L'architecture de ce bloc est présentée dans la figure 4.35. Les principaux éléments qui le constituent sont :

– Le sous-bloc RegE qui contient deux registres, dont l'un récupère les données sérielles en provenance du maître (MMD) et l'autre récupère le paquet ainsi lu dans le premier registre, lorsqu'une fin de transmission est détectée (front descendant de MCTL). Ce sous-bloc a en outre une autre fonction qui consiste à générer, à partir des deux signaux MMD et MCTL, les signaux MCTL_Reg et MMD_Reg qui sont les variables d'états de l'automate de contrôle de l'esclave. Il est important de signaler que ce bloc fonctionne en permanence, même si l'esclave n'est pas concerné par l'opération.

- Le sous-bloc RegS qui a la tache de transmettre les paquets formés dans l'esclave vers le maître, via le signal MSD.
- Le sous-bloc de construction de paquets qui construit et fournit à RegS les paquets à transmettre au maître, suite à une demande de ce dernier. Pour se faire, il a besoin du type du paquet à transmettre, qui lui est fourni par le bloc de décodage de commandes, des différentes données nécessaires à la construction du paquet, qu'il récupère dans les registres internes de l'esclave et de l'adresse de l'esclave, qui peut être codée ou disponible comme entrée de l'esclave (connectée à des "switchs"). Ce sous-bloc est de plus en charge du calcul du bit de parité des paquets qu'il transmet.
- Le sous-bloc CSMG "Control Same_Message Generation" qui est responsable de la génération du signal (interne à l'esclave) **`"Same_Message"`**. Ce message indique au bloc de décodage de l'esclave si le dernier message reçu est une en-tête. Rappelons que seules les en-têtes sont décodées.
- L'automate de contrôle qui représente le moteur de l'esclave. Il est le responsable de la transmission des données. Du fait que les variables d'états sont directement générées à partir des signaux du bus MMD et MCTL, l'automate se comporte toujours comme si les données arrivant sur le bus lui étaient adressées. Evidemment ces données ne sont véritablement prise en compte par l'esclave que si le bloc de décodage arrive à détecter son adresse dans l'en-tête du message.

`Le bloc de gestion d'erreurs :` Il entre réellement en jeu dans le cas où une erreur intervient au niveau de l'esclave. Les trois sous-blocs qui constituent son architecture sont :

- Le sous-bloc de vérification de parité, qui calcule la parité des seize derniers bits du paquet reçu (de 1 à 16) et la compare au contenu du bit 0. En cas d'inégalité, il signale l'erreur au sous-bloc d'interprétation d'erreurs et de gestion d'interruptions.
- Le sous-bloc de détection de collision, qui compare le signal MSD généré à l'intérieur de l'esclave, pris directement à partir du bloc de transmission, au même message

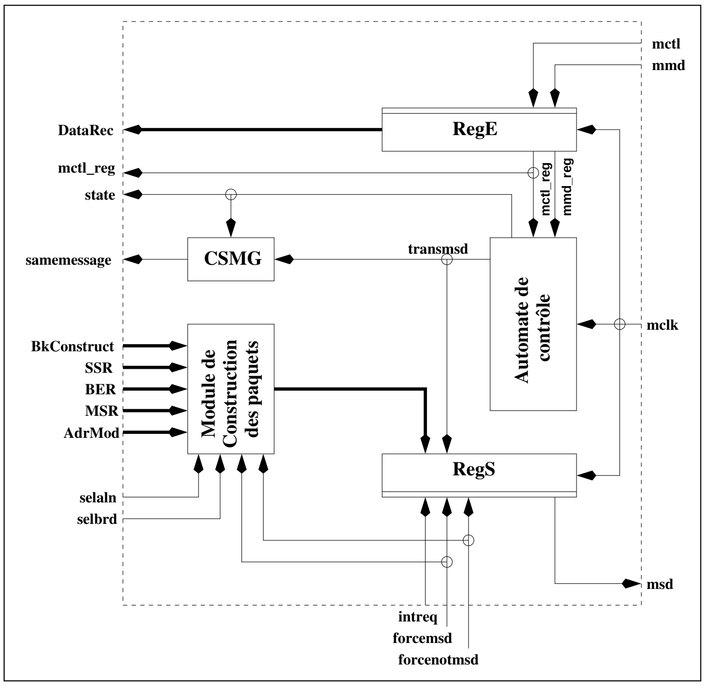

Fig. 4.35 – Le bloc d'émission/réception

mais pris cette fois ci sur le bus. Ceci permet évidemment à l'esclave de détecter une éventuelle collision entre son propre signal MSD et celui d'un autre esclave. En cas de collision, ce sous-bloc informe le sous-bloc d'interprétation d'erreurs et de gestion d'interruptions.

– Le sous-bloc d'interprétation d'erreurs et de gestion d'interruption, qui représente le centre de contrôle du bloc de gestion d'erreurs. Il a pour rôle de récupérer les informations concernant les erreurs de parité, de collision ou d'état incorrect, de les interpréter et de les sauvegarder dans les registres d'état, si la commande en cours le permet. En outre, au vu des erreurs qu'il peut interpréter, il est en charge de générer certaines interruptions internes, pour en informer les différents blocs constituant l'esclave.

Le synoptique de ce bloc est représenté dans la figure 4.36.

Le bloc de décodage : Ce bloc a trois taches principales, à savoir le décodage d'adresses contenues dans les en-têtes des paquets, le décodage des commandes et le comptage des paquets. Son architecture est donc naturellement constituée de trois sous-

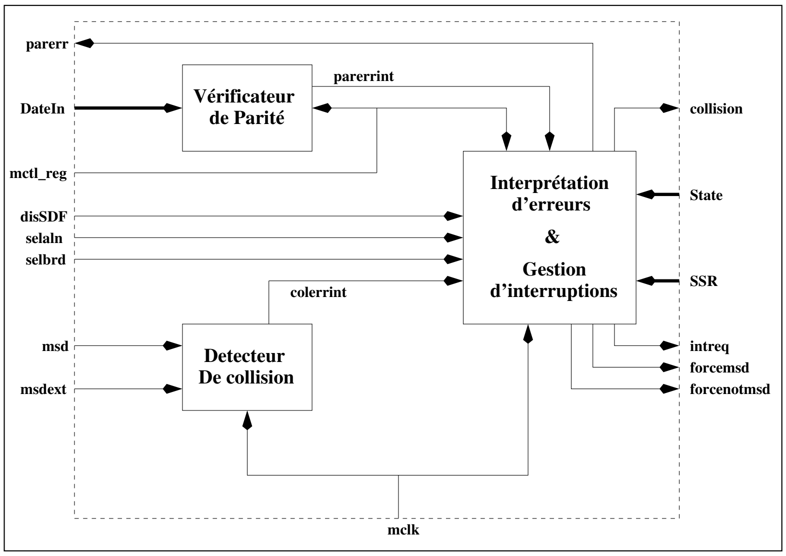

Fig. 4.36 – Le bloc de gestion d'erreurs

blocs (figure 4.37):

– Le décodeur d'adresses qui, dans le cas où le paquet reçu est une en-tête, récupère l'adresse de l'esclave (ou du groupe d'esclaves) adressé par le maître et vérifie si l'esclave dont il est le décodeur est concerné ou non par le paquet.

– Le décodeur de commandes qui, dans le cas où l'esclave et bien adressé par le maître, décode et exécute la commande transmise dans le paquet. La commande est interprétée et le comportement de l'esclave est orienté en tenant compte de la nature de la commande, de la façon dont l'esclave est adressé (seul ou en groupe) et des erreurs qui peuvent survenir. L'esclave fournit un paquet de réponse (paquet "ACK") au maître si la commande l'exige. Signalons que le le standard spécifie 18 commandes de base.

– Le compteur de paquets, qui se charge de récupérer dans l'en-tête le nombre de paquets à échanger avec le maître et de vérifier si, à la fin d'une réception, le nombre de paquets reçu est le bon.

Le bloc registres : C'est le bloc contenant les trois registres d'états de l'esclave:

– Le registre SSR " Slave Status Register ": c'est le registre d'état de l'esclave, il contient des informations concernant son état courant (ex. présence d'une erreur).

– Le registre MSR " Module Status Register ": il concerne particulièrement l'application ou les éventuelles extensions de l'esclave. De ce fait, il n'intervient pas dans

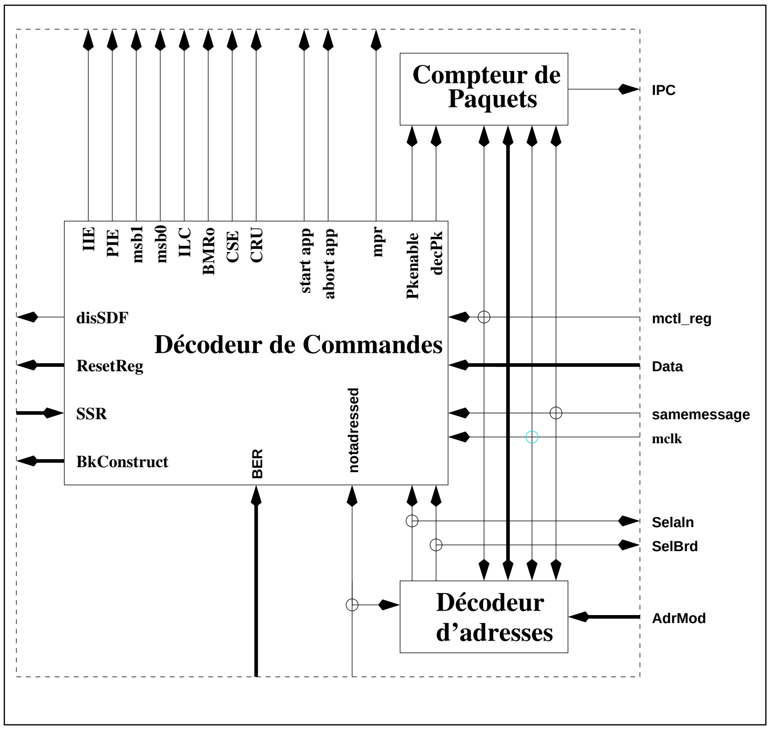

Fig. 4.37 – Bloc de décodage

le fonctionnement de l'esclave générique. Le standard permet de définir, au besoin, plus d'un seul registre MSD.

– Le registre BER " Bus Error Register ": il contient le détail de l'erreur qui peut survenir (parité, collision sur le bus ou autres).

Le contrôleur système dédié

C'est un contrôleur dédié à un système donné, qui ne se base sur l'utilisation d'aucun standard connu. En effet, dans certains cas le concepteur peut préférer ce type de contrôleur, pour ne pas être limité par les contraintes d'un standard. Comme l'architecture d'un tel contrôleur est très liée au dispositif de test du système auquel il est dédié, on ne peut pas la définir en détail dans ce paragraphe.

Une fois le choix du contrôleur système effectué, le traitement système se termine par la définition des blocs d'interfaçage au niveau des contrôleurs carte. L'architecture globale de test du système se trouve ainsi complètement définie.

4.10 Conclusion

La méthodologie présentée dans ce chapitre se veut une approche pragmatique pour l'aide à la conception des systèmes testables. En effet, nous avons essayé de rester toujours très proche de l'aspect pratique des systèmes discrets. Ces derniers se caractérisent par certaines contraintes réelles dont il faut tenir compte avant de prendre une quelconque décision: encombrement, compatibilité, etc.

Elle se base principalement sur quatre principes fondamentaux qui, en quelque sorte, définissent son aspect pragmatique. Trois de ces principes ont été traités dans ce chapitre:

- Le principe d'*interactivité*, qui caractérise l'intervention du concepteur lors des différent traitements de la méthode, dans le but d'aboutir à des compromis entre les contraintes qu'il impose et les propositions qui lui sont présentées.

- Le principe de *modularité*, qui découle de l'aspect modulaire de la méthode. Plusieurs modules (évaluation, génération de structures de test, génération de contrôleurs de test, etc.) sont accessibles à partir d'un module central qui représente l'organigramme principal du traitement.

- Le principe de *réutilisation* ("Reuse"), qui se manifeste à deux niveaux: la réutilisation des structures de test d'un module donné pour le test d'un autre module (ex : la "clusterisation" sur les cartes BS) et la réutilisation de contrôleurs de test standards pour circuits ou cartes pré-générés.

Le quatrième principe est l'*évaluation* de la testabilité du système et de ses modules. Vu l'importance de ce principe, nous avons décidé de lui consacrer le prochain chapitre, dans lequel nous présentons le module d'évaluation proposé pour l'étude de la testabilité des cartes ou des MCMs d'un système. Pour l'évaluation de la testabilité des circuits, nous considérons qu'il s'agit d'un problème au niveau circuit et non au niveau système; des modules externes se basant sur des méthodes connues d'évaluation de la testabilité (comme SCOAP [Gol79]) peuvent être appelés lors du traitement de la testabilité des circuits (paragraphe 4.4).

Enfin, la méthodologie présentée a été concrétisée par le développement de l'outil STA ("System Testability Assistant") qui sera décrit au chapitre 6.

Chapitre 5

Méthode d'évaluation de la testabilité des cartes

5.1 Introduction

Ce chapitre présente une méthode d'évaluation de la testabilité des modules principaux d'un système discret, à savoir les cartes et les MCMs (que nous considérons de la même façon que les cartes). Cette méthode représente une partie importante de la méthodologie d'aide à la conception des systèmes testables exposée dans le chapitre précédent. En effet, pour proposer des solutions globales de conception en vue de la testabilité, il est nécessaire de disposer d'informations concernant la testabilité des éléments constituants le système: circuit, cartes et MCMs. En ce qui concerne les circuits, nous considérons que le problème est résolu étant donné que plusieurs méthodes d'évaluation de la testabilité existent dans la littérature, à différents niveaux d'abstraction: logique (SCOAP) [Gol79] [GT80], RTL (TMEAS) [Gra79] et même comportemental (BETA) [CWS91] (voir paragraphe 3.3.4). Cependant, au niveau carte, ces méthodes ne sont plus applicables, pour diverses raisons d'ordre technique et conceptuel (voir paragraphe 3.3.4). Entre autres, le fait que les architectures internes des circuits ne sont pas toujours connues ou disponibles implique que la mise à plat de l'architecture de la carte à un niveau d'abstraction donné (logique, RTL ou comportemental) est généralement impossible. Le principe de notre méthode est donc d'évaluer la testabilité des noeuds d'une carte sans tenir compte de l'architecture interne des circuits qui la constituent.

Il est possible d'adapter cette méthode aux systèmes intégrés ("System-on-a chip" ou SoC), seules quelques informations externes à propos des composants intégrés dans de tels systèmes étant nécessaires. En effet, nous pouvons toujours considérer un SoC comme étant une carte et les composants qui le constituent comme étant les circuits de la carte. Pour les SoC intégrant des "IP cores", notre méthode d'évaluation reste toujours valable. Une telle extension est possible parce que les seules informations concernant les "cores" et requises par la méthode sont des informations d'ordre externe qui ne violent pas le secret de la propriété intellectuelle (voir les différentes expériences décrites dans le rapport "Testing embedded cores" [Soc98]).

5.2 But de l'évaluation

L'idée générale est d'évaluer la testabilité des noeuds des cartes (ou SoCs) afin de détecter ceux qui sont difficiles à tester. Les informations ainsi obtenues sont ensuite utilisées lors de la proposition de stratégies de test pour les circuits (ou les composants virtuels des SoCs). En effet, pour rendre un noeud d'une carte complètement contrôlable et observable (et donc testable), il suffit d'ajouter du BS aux circuits auxquels il est connecté. Rendre tous les noeuds difficiles à tester complètement testables améliore considérablement la testabilité de la carte. Ce principe reste valable pour les SoCs incluant des "IP cores", dans ce cas c'est à l'intégrateur du système de prévoir les structures BS à ajouter autour des "IP" concernés.

D'autre part, l'estimation de la testabilité de chaque carte (ou MCM) d'un système est une donnée très importante lors des prises de décisions concernant le type de l'architecture de test à adopter pour le système (centralisée ou distribuée, paragraphe 4.6) et le type du contrôleur système de test (logiciel ou matériel, paragraphe 4.7). En effet nous assumons que la testabilité d'une carte est équivalente à sa complexité qui elle, influence les deux choix que nous venons de citer.

Le processus d'évaluation de testabilité peut avoir lieu à différentes étapes du processus d'aide à la conception de systèmes testables 4.2. Il est particulièrement nécessaire de l'implémenter à trois niveaux précis:

- Au début du processus, pour détecter dans chaque carte ou MCM les noeuds difficiles à tester et donc déterminer les lieux prioritaires à l'adjonction de test, afin d'améliorer la testabilité globale du système.

- Pendant l'étape de pré-traitement des cartes (paragraphe 4.5), pour évaluer l'amélioration de la testabilité de la carte après le traitement des circuits. C'est cette dernière évaluation qui sera prise en compte plus tard, lors du choix de l'architecture globale de test et du type du contrôleur système.

– A la fin du processus, pour avoir une idée de la testabilité globale du système et de son évolution par rapport à sa testabilité initiale.

Outre ces trois niveaux d'évaluation, le processus d'évaluation de la testabilité peut être utilisé par le concepteur à la demande, à n'importe quel étape du traitement.

Dans le paragraphe 3.3.4, nous avons cité une méthode d'évaluation de la testabilité d'une carte basée sur une approche qualitative [PLL97]. Cette méthode, bien que fournissant des résultats intéressants, ne peut pas être utilisée dans notre cas. En effet, notre approche nécessite des résultats très précis concernant la testabilité des noeuds et des circuits d'une carte. De tels résultats ne peuvent être obtenus qu'en adoptant une approche quantitative et c'est ce qui a motivé l'élaboration de la méthode que nous allons présenter.

Concrètement, l'approche consiste à calculer la contrôlabilité et l'observabilité des noeuds d'une carte, à détecter les noeuds difficilement testables, à classer ces noeuds selon leur testabilité et enfin à classer les circuits de la carte selon la moyenne des testabilité de leurs entrées/sorties. Ces deux classements sont très importants étant donné qu'ils guident l'ajout de structures de test sur la carte (principalement le BS). En effet, l'expérience a montré qu'ajouter du BS dans les circuits critiques permet d'augmenter considérablement la testabilité globale de la carte, sans qu'il soit nécessaire de doter tous les circuits du standard.

Sachant que dans notre cas les valeurs des métriques de testabilité (contrôlabilité et observabilité) varient de 0 à 1, le calcul des contrôlabilités à travers les noeuds d'une carte se fait en propageant les contrôlabilités des entrées primaires (supposées toutes égales à 1) vers les sorties primaires. Inversement, le calcul des observabilités se fait en propageant les observabilités des sorties primaires (égales à 1) vers les entrées primaires de la carte. Il est important de signaler que nous considérons que la testabilité d'un noeud est égale à son accessibilité qui n'est autre que la moyenne de sa contrôlabilité et de son observabilité.

D'autres approches quantitatives d'évaluation de la testabilité, en vue de guider l'inclusion de structures de test existent déjà (voir paragraphe 3.3.4), mais s'intéressent toutes au niveau circuit et restent non applicables au niveau carte. Ceci nous permet de supposer que la propagation des contrôlabilités et des observabilités à travers les circuits est une information connue et disponible.

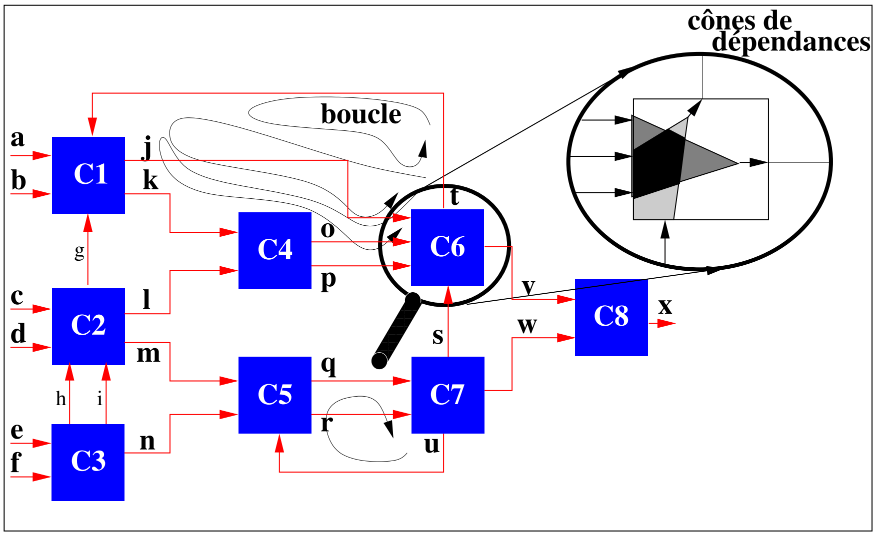

Fig. 5.1 – Boucles et cônes de dépendances sur une carte

5.2.1 Les boucles et les cônes de dépendances

Deux paramètres essentiels interviennent dans le processus d'évaluation, à savoir les boucles au niveau de l'architecture de la carte et les cônes de dépendances au niveau des circuits.

bien que nous considérons les circuits comme des boites noires, certaines informations les concernant sont néanmoins nécessaires. Ces informations sont d'ordre externe et concernent la dépendance entre les entrées primaires et les sorties primaires d'un circuit. Pour modéliser cette dépendance, nous utilisons la notion de **cônes de dépendances**. En effet, un cône de dépendance est défini par une sortie primaire et l'ensemble des entrées primaires dont elle dépend (figure 5.1). Si deux ou plusieurs sorties primaires dépendent des mêmes entrées, elles sont alors modélisées par le même cône. Outre ses entrées et ses sorties, un cône de dépendance est défini par deux facteurs importants qui spécifient la propagation des métriques de testabilité à travers lui. Le premier est le DPCF ("Dependency Cone Factor"), qui représente la contrôlabilité d'une sortie d'un cône si celles de ses entrées sont toutes égales à 1. Le second est l'OBF ("Observability Factor"), qui représente l'observabilité d'une entrée d'un cône si celle de sa sortie ou de l'une au moins de ses sorties est égale à 1. Chaque entrée d'un cône admet son propre OBF. Toutes ces informations font appel à l'étude de testabilité du circuit et constituent des constantes qui peuvent être fournies au concepteur sans entrer dans le détail de l'architecture du circuit (ou d'un IP "core" dans le cas d'un système intégré). Dans [Gu96] par exemple, une méthode est proposée pour le calcul de ces facteurs au niveau RTL.

Dans certains cas, la définition des cônes de dépendances dans un circuit devient très difficile. En effet, pour certains circuits complexes, il arrive que toutes les sorties primaires soient dépendantes d'un très grand nombre d'entrées primaires, compliquant ainsi la définition des cônes. Autre exemple, celui des processeurs et des micro-contrôleurs, pour lesquels il arrive souvent que des sorties primaires ne dépendent d'aucune entrée primaire, ce qui rend la notion de cône de dépendance inutilisable. Dans un cas comme dans l'autre, on est obligé de fixer des heuristiques pour s'affranchir des problèmes que peuvent causer ces cas de figure. Ces heuristiques consistent généralement à considérer que les sorties de ces circuits, principalement celles qui causent problème, sont prioritairement testables, ce qui nous permet de les considérer comme étant des entrées primaires pour la carte. Concrètement, en décidant par exemple de doter les circuits en question des fonctionnalités BS, leurs sorties (respectivement entrées) deviennent complètement testables et agissent comme des entrées primaires (respectivement sorties primaires) sur la carte, dont les contrôlabilités (respectivement observabilités) sont égales à 1.

Le deuxième paramètre qui influe largement sur le calcul des métriques de testabilité sur une carte est la présence de boucles dans l'architecture de la carte (figure 5.1). En effet, la propagation des contrôlabilités des entrées vers les sorties et des observabilités des sorties vers les entrées ne peut plus se faire naturellement puisque la contrôlabilité et l'observabilité des noeuds de retour au niveau des boucles constitueront des inconnues. La solution consiste à ouvrir (virtuellement) ces boucles au niveau de ces noeuds, tout en tenant compte de l'effet de boucle. Pour ce faire, on injecte dans le calcul des "facteurs boucle", qui seront décrits dans la suite de ce chapitre.

Les algorithmes de détection de boucles, de détection des noeuds de retour, de calcul des "facteurs boucle" et enfin de propagation des métriques de testabilité à travers la carte ont tous été complètement automatisés. Plus de détails concernant ces algorithmes sont présentés dans le chapitre 6 dédié à l'outil STA, dans lequel cette méthode d'évaluation a été intégrée.

5.2.2 Calcul de la contrôlabilité et de l'observabilité

Lors de la propagation des contrôlabilités à travers une carte, la **contrôlabilité** d'une sortie primaire d'un circuit est calculée en tenant compte du DPCF du cône de dépendance auquel elle appartient et des contrôlabilités des entrées de ce cône, selon les

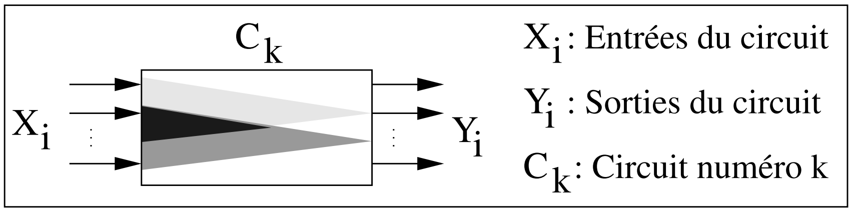

Fig. 5.2 – Cônes de dépendances dans un circuit

formules (a) et (b).

$$(a)\ C_{Y_J} = DPCF_l.C_{IN}\ \ \&\ \ (b)\ C_{IN} = \prod_{p \le I \le q} C_{X_I}$$

Où l représente le cône de dépendance de la sortie et X_p ... X_q les entrées du cône (figure 5.2). L'utilisation d'un produit au niveau de la formule (b) s'explique par le fait que le contrôle d'une sortie nécessite le contrôle de toutes les entrées dont elle dépend.

Lors de la propagation des observabilités, l'**observabilité** d'une entrée primaire d'un circuit est calculée en tenant de son OBF et des observabilités des sorties des différents cônes dont elle constitue une entrée, selon la formule (c).

$$(c)\ O_{X_I} = OBF_{X_I}\ .\ \frac{\sum_{p \le J \le q} O_{Y_J}}{q-p+1}$$

Où Y_p ... Y_q représentent les sorties des cônes dont X_i est définie comme entrée. L'utilisation de l'opération somme dans cette formule s'explique par le fait qu'il n'est pas nécessaire d'observer toutes les sorties concernées pour observer l'entrée en question, mais que l'observation de chacune d'elles peut s'avérer utile.

Tenant compte des trois formules (a), (b) et (c), du fait que la contrôlabilité d'une entrée primaire (EP) est toujours égale à 1 et du fait que l'observabilité d'une sortie primaire (SP) est toujours égale à 1, nous pouvons alors propager les contrôlabilités des EPs et les observabilités des SPs à travers la carte et calculer ainsi la testabilité de chaque noeud.

Cependant, ces formules ne sont plus suffisantes quand des boucles se présentent au niveau de l'architecture de la carte (cas très fréquent, figure 5.3). Pour ce cas on définit deux "facteurs boucle", à savoir le facteur LCF ("Loop Controlability Factor") et le facteur LOF ("Loop Observability Factor"). Ces deux facteurs sont calculés selon les formules (d) et (e).

$$(d)\ LCF_L = \prod_{I}^{J(I)} DPCF_{IJ}\ \ \&\ \ (e)\ LOF_L = \prod_{I}^{K(I)} OBF_{IK}$$

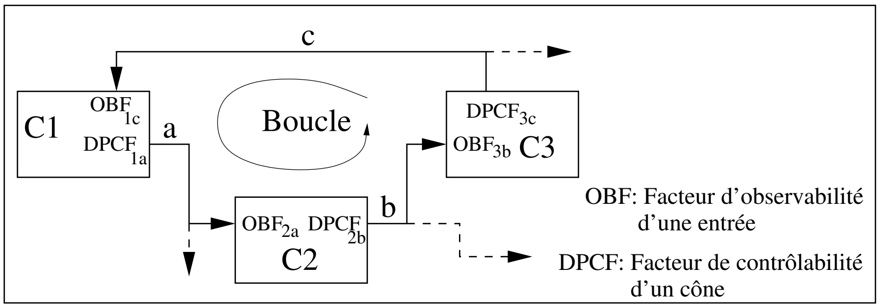

Fig. 5.3 – Exemple de boucle

Où I parcourt les circuits traversés par la boucle et J(I) et K(I) sont respectivement les entrées et sorties des circuits I par lesquelles passe la boucle. Pour la boucle de la figure 5.3, nous aurons donc:

$$LCF = DPCF_{1a}.DPCF_{2b}.DPCF_{3c}$$

et

$$LOF = OBF_{1c}.OBF_{2a}.OBF_{3b}$$

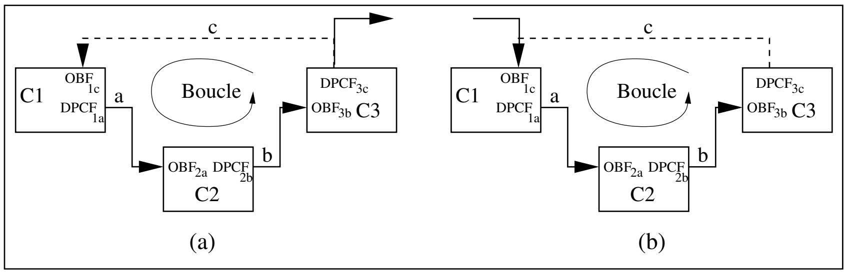

Fig. 5.4 – Ouverture virtuelle des boucles

La façon dont les boucles sont virtuellement ouvertes est présentée dans les figures 5.4 (a) et 5.4 (b). En effet, cette méthode coupe les liens de retour dans les boucles et permet de propager normalement les contrôlabilités et les observabilités à travers la carte. Une fois cette propagation faite, les métriques finales sont calculées selon les formules (f) et (g).

$$(f)\ C_{Y_J} = C^*_{Y_J}.CF_L \ \ \& \ \ (g)\ O_{X_I} = O^*_{X_I}.OF_L$$

Où $C^*_{Y_J}$ et $O^*_{X_I}$ sont les contrôlabilité et observabilité calculées selon les formules (a), (b) et (c), en ouvrant les boucles.

5.2.3 Les métriques de testabilité d'un noeud

Une fois les contrôlabilités et observabilités des EPs et SPs des différents circuits calculées, nous pouvons déduire celles des noeuds de la carte. On définit la contrôlabilité d'un noeud (figure 5.5) comme étant la contrôlabilité de la branche entrante. Dès lors, la contrôlabilité du noeud se propage sur les différentes branches sortantes :

$$(h)\ C_{I_1} = C_{I_2} = C_{I_3} = C_O$$

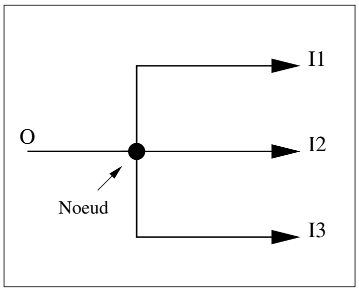

Fig. 5.5 – Exemple d'un noeud sur la carte

L'observabilité du noeud est égale au maximum des observabilités des branches de sorties. Cette observabilité se propage directement vers la branche entrante du noeud :

$$(i)\ O_O = max(O_{I_1}, O_{I_2}, O_{I_3})$$

Etant donné que la testabilité est, pour nous, égale à l'accessibilité alors elle peut être calculée pour un noeud donné selon la formule :

On évalue la testabilité d'un circuit en calculant la moyenne des testabilités de ses entrées/sorties. Notons que la testabilité d'une broche d'un circuit est égale à la moyenne de sa contrôlabilité et de son observabilité.

Enfin la testabilité globale d'une carte (BTA "Board Test Average") est calculée comme la moyenne des testabilités des différents noeuds de la carte. Le BTA sera donc considéré comme la complexité de la carte, grandeur très importante dans la prise décision concernant le type de l'architecture globale de test et le type du contrôleur système lors du traitement.

Une fois tous les calculs de testabilité finis, trois classements sont dressés :

– Classement des circuits par ordre croissant de testabilité. Il peut être très utile lorsque le concepteur a besoin de connaître quels circuits devraient inclure prioritairement des fonctionnalités BS.

- Classement des broches des différents circuits par ordre croissant de testabilité. Il peut être utile lorsque le concepteur est en présence d'une carte déjà conçue et que l'ajout de fonctionalités BS se fait à l'aide de registres BS (voir paragraphe 3.3.1). Plus précisément, ce classement est utile lorsqu'on est dans une configuration de traitement exhaustif, où un registre BS peut être connecté à des broches provenant de différents circuits.

- Classement des noeuds par ordre croissant de testabilité. Il est très utile dans l'évaluation de la testabilité de la carte et dans la détection des noeuds difficiles à tester.

Cas particulier: les signaux bidirectionnels

Souvent sur les cartes, nous nous retrouvons en présence de signaux bidirectionnels (figure 5.6 (a)), pour lesquels les formules que nous venons de présenter ne sont plus applicables. Cela est dû au fait que les notions de contrôlabilité et d'observabilité s'appliquent dans les deux sens pour un même signal bidirectionnel. Plus précisément, l'étude de la contrôlabilité (respectivement observabilité) d'un signal bidirectionnel doit se faire dans un sens comme dans l'autre. Le problème se complique lorsque ces signaux forment entre eux des noeuds bidirectionnels multiples (figure 5.6 (b)).

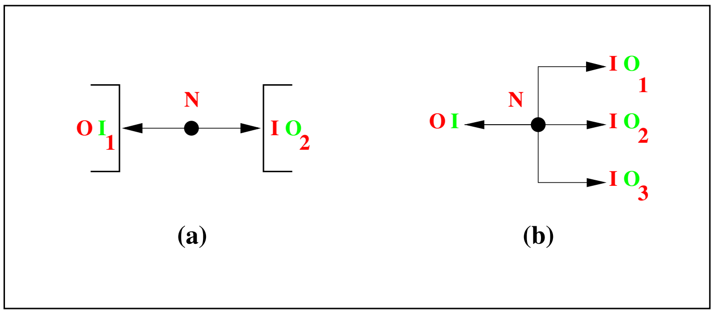

Fig. 5.6 – Signal et noeud bidirectionnels

La solution que nous proposons consiste à éclater ces signaux de façon à les rendre unidirectionnels, créant ainsi de nouvelles entrées/sorties unidirectionnelles virtuelles pour les circuits concernés. L'éclatement de ces signaux provoque par ailleurs la création de nouveaux noeuds unidirectionnels virtuels (figure 5.7). Selon la nature du noeud éclaté, un certain nombre de noeuds virtuels est créé, relatant les différentes configurations de fréquentation du noeud (figure 5.8). Les entrées/sorties virtuelles ainsi que les noeuds virtuels sont ajoutés au niveau de la "netlist" de la carte, et le calcul de leurs testabilités se fait, de façon normale, selon les formules précédentes (de (a) à (j)).

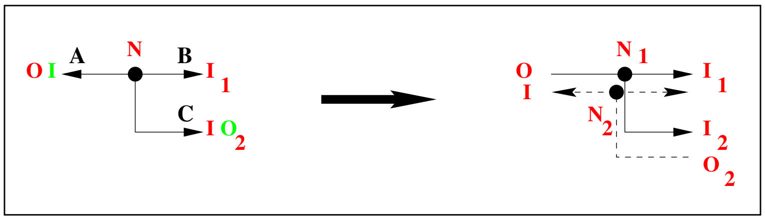

Fig. 5.7 – Exemple d'éclatement d'un noeud

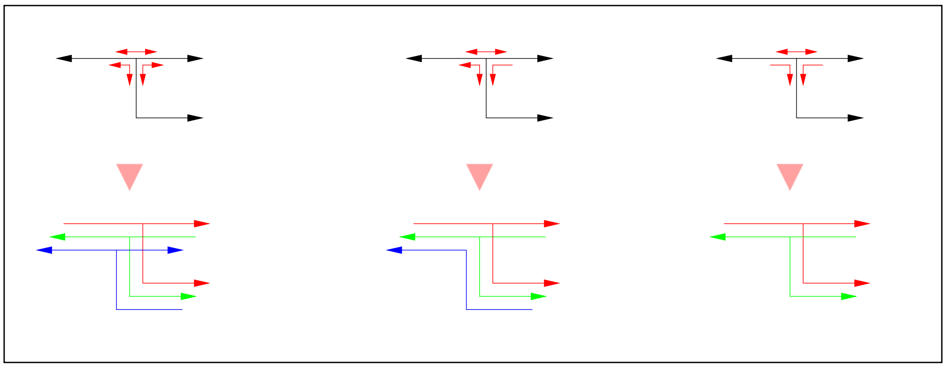

Fig. 5.8 – Différentes configuration d'éclatement de noeuds

La contrôlabilité, l'observabilité et la testabilité du noeud initial et des entrées/sorties initiales, sont fonction des résultats obtenus pour les noeuds et entrées/sorties virtuelles issus de l'éclatement. Ainsi pour le noeud N de la figure 5.7, nous avons:

$$T_N = F(T_{N1}, T_{N2})$$

Les contrôlabilité, observabilité et testabilité des entrées/sorties A, B et C, se calculent de la même façon.

La façon la plus simple de définir la fonction F est de la considérer comme étant une moyenne. Cela dit, nous pouvons imaginer d'autres moyens pour définir cette fonction, précisément en tenant compte du degré de fréquentation du noeud dans une configuration ou dans une autre.

5.2.4 Validation de la méthode

En l'absence d'ATPG et d'un environnement efficace pour le test de cartes dans notre laboratoire, nous avons décidé de tester notre méthode au niveau circuit et de comparer

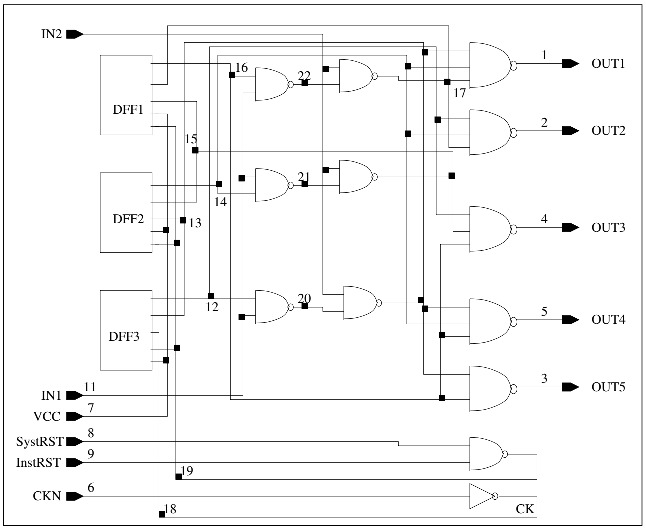

Fig. 5.9 – Architecture du circuit traité

les résultats à ceux obtenus par l'outil SCOAP (dédié aux circuits). Pour cette expérience, nous avons traité le circuit représenté par la figure 5.9 comme une carte et les portes et les bascules ont été considérées comme étant les circuits de cette carte. Les DPCFs et les OBFs des portes sont tous égaux à 1 et nous supposons que ceux des bascules sont de 0,8. Les résultats obtenus par SCOAP sont rapportés sur le tableau de la figure 5.10, alors que ceux obtenus avec notre méthode sont rapportés sur celui de la figure 5.11. Pour chacune des deux méthodes, une classification par ordre de testabilité des noeuds du circuit a été faite.

Noeuds	CC^0	CC^1	CO	SC^0	SC^1	SO	Classification
1	2	6	0	0	1	0	6
2	2	5	0	0	2	0	6
3	2	6	0	0	1	0	6
4	2	5	0	0	2	0	6
5	2	5	0	0	2	0	6
6	1	1	8	0	0	3	1
7	1	1	4	0	0	2	1
8	1	1	4	0	0	2	1
9	1	1	4	0	0	2	1
10	1	1	7	0	0	2	1
11	1	1	6	0	0	1	1
12	1	1	0	1	1	1	3
13	2	4	3	0	0	1	5
14	1	1	0	1	1	0	3
15	2	4	3	0	0	1	5
16	1	1	0	1	1	0	3
17	2	4	3	0	0	1	5
18	2	3	7	0	0	3	2
19	2	3	2	0	0	2	2
20	2	3	5	0	1	1	4
21	2	3	5	0	1	1	4
22	2	3	5	0	1	1	4

Fig. 5.10 – Résultats fournis par SCOAP

Noeuds	Contrôlabilité	Observabilité	Testabilité	Classification
1	0.692381	1.000000	0.846191	11
2	0.638073	1.000000	0.819036	12
3	0.407149	1.000000	0.703575	16
4	0.612544	1.000000	0.806272	14
5	0.520210	1.000000	0.760105	15
6	1.000000	0.959990	0.979990	3
7	1.000000	0.959990	0.979995	2
8	1.000000	0.959990	0.979995	2
9	1.000000	0.959990	0.979995	2
10	1.000000	1.000000	1.000000	1
11	1.000000	1.000000	1.000000	1
12	0.884674	1.000000	0.942332	5
13	0.959971	1.000000	0.979981	4
14	0.849278	1.000000	0.924634	6
15	0.721259	1.000000	0.860625	10
16	0.751334	1.000000	0.875662	9
17	0.849261	1.000000	0.924626	6
18	1.000000	0.959990	0.979990	3
19	1.000000	0.960000	0.979995	2
20	0.849261	0.959962	0.904612	7
21	0.782656	0.842278	0.815967	13
22	0.921554	0.884674	0.903114	8

Fig. 5.11 – Résultats fournis par notre méthode pour l'exemple du circuit

En comparant les deux classifications nous remarquons qu'elles sont presque semblables, malgré l'existence de certaines différences au niveau de quelques noeuds (spécialement les noeuds 13, 16 et 21). L'explication de ces différences découle du fait que notre méthode est principalement conçue pour des cartes et non pour des circuits. Prenant par exemple le cas de la contrôlabilité, modélisée dans SCOAP par quatre métriques différentes : CC^0, CC^1, SC^0 et SC^1 (CC : contrôlabilité combinatoire à 1 ou à 0, SC : contrôlabilité séquentielle à 1 ou à 0). Pour une carte, une telle modélisation est très difficile à mettre au point puisque les composants élémentaires (circuits) ont une fonctionnalité beaucoup plus complexe que celle d'une porte. Toutefois, entre les deux méthodes une observation plus précise des classifications obtenues, montre que la différence entre les deux méthodes réside principalement dans le classement des observabilités. En effet, SCOAP considère que pour observer l'entrée d'une porte, il faut non seulement observer sa sortie, mais aussi contrôler ses autres entrées, ce qui est traduit dans les formules de calcul de l'observabilité (CO : observabilité combinatoire et SO : observabilité séquentielle). Dans notre cas, vu la complexité que peut atteindre un cône de dépendance, nous avons préféré l'utilisation d'un facteur OBF pour chaque entrée qui, étant calculé au niveau circuit, contient déjà l'information concernant la contrôlabilité des différentes entrées autres que celle dont il est le facteur. Ceci permet de s'affranchir de l'introduction de la contrôlabilité des entrées d'un cône dans le calcul de l'observabilité de l'une d'elles, étant donné que même si elles font partie du même cône, elles peuvent avoir des degrés d'influence très différents, ce qui n'est pas le cas pour les entrées d'une porte d'un circuit.

Il est clair que notre méthode n'est pas vraiment adéquate au niveau circuit, mais les résultats qu'elle peut fournir pour des petits circuits peuvent être tout à fait raisonnables. Mais son intérêt réel est de résoudre les problèmes que peuvent rencontrer d'autres méthodes dédiées aux circuits quand on veut les appliquer au niveau carte (ex SCOAP).

5.2.5 Utilisation de la méthode

Circuit	Entrées	Sorties	DPCF
C_1	a, b, t a, b, t, g	j k	0.8
C_2	c, d, h, i	g, l, m	0.8
C_3	e, f	h, i , n	0.8
C_4	k, l	o, p	0.8
C_5	m, n, u m, n	q r	0.8
C_6	j, o, p o, p, s	t v	0.8
C_7	q, r	s, w, u	0.8
C_8	v, w	x	0.8

Fig. 5.12 – Cônes de dépendances des circuits de la carte

Ce paragraphe montre comment notre méthode d'évaluation peut être utilisée au niveau carte pour améliorer la testabilité globale. L'exemple que nous allons traiter est représenté par la figure 5.1, c'est un exemple simple mais qui inclut tous les éléments ayant une influence particulière lors de l'étape d'évaluation (boucles, boucles imbriquées, cônes de dépendances à sorties multiples, etc.). Comme il ne s'agit pas d'un exemple réel, nous avons dû fixer nous même les facteurs DPCF et OBF (ils ont été tous fixés à 0.8). La figure 5.12 présente les cônes de dépendances à travers les circuits de la carte. Rappelons que, du fait qu'elles sont complètement contrôlables (respectivement observables), les entrées (respectivement sorties) primaires de la carte ont chacune une contrôlabilité (respectivement observabilité) égale 1. Au début du traitement, aucun circuit ne contient de fonctionalités de test.

Les résultats obtenus après l'évaluation de la testabilité sont présentés dans la table de la figure 5.13. En parcourant la classification des noeuds de la carte selon leur testabilité, nous remarquons que le noeud (t) est le moins testable de tous. Ceci était prévisible et s'explique par le fait qu'il représente un noeud de retour pour trois boucles à la fois, ce

Noeuds	Contrôlabilité	Observabilité	Testabilité	Classification des noeuds
(a)	1	0.14	0.57	18
(b)	1 primary	0.14	0.57	19
(c)	1 inputs	0.179	0.589	21
(d)	1	0.179	0.589	22
(e)	1	0.212	0.606	23
(f)	1	0.179	0.589	24
(g)	0.64	0.128	0.384	10
(h)	0.8	0.118	0.459	13
(i)	0.8	0.3	0.55	17
(j)	0.26	0.21	0.235	7
(k)	0.21	0.16	0.185	4
(l)	0.21	0.168	0.189	5
(m)	0.64	0.377	0.508	16
(n)	0.8	0.377	0.588	20
(o)	0.14	0.21	0.175	2
(p)	0.14	0.21	0.175	3
(q)	0.26	0.368	0.314	8
(r)	0.41	0.576	0.49	14
(s)	0.085	0.64	0.362	9
(t)	0.04	0.14	0.09	1
(u)	0.11	0.294	0.202	6
(v)	0.0013	0.8	0.4	11
(w)	0.085	0.8	0.442	12
(x)	0	1 primary output	0.5	15

Fig. 5.13 – Résultats de l'évaluation de la testabilité de la carte

qui altère sa contrôlabilité et son observabilité. La testabilité globale de la carte (BTA) a été calculée et est égale à 0.406.

	Circuits Boundary Scan	Classement des circuits selon un ordre croissant de la testabilité								BTA
(a)	AUCUN	4	6	1	7	8	5	2	3	0.406
(b)	Circ4	7	8	5	6	3	1	2	4	0.67
(c)	Circ6	5	7	8	2	3	4	1	6	0.736
(d)	Circ1	7	8	5	6	4	3	2	1	0.683
(e)	Circ6, Circ4	5	7	8	3	1	2	4	6	0.788
(f)	Circ6, Circ7	2	3	4	1	5	8	6	7	0.868
(g)	Circ6, Circ5	2	4	1	3	8	7	6	5	0.882
(h)	Circ6, Circ5, Circ2	8	1	4	7	3	2	5	6	0.964

Fig. 5.14 – Influence de l'inclusion de structures BS dans les circuits

En observant la classification 5.14(a) représentant le classement des circuits par ordre de testabilité, nous remarquons que les circuits les moins testables sont, dans l'ordre, C_4, C_6 et C_1. Si C_6 et C_1 sont tous deux impliquées dans trois boucles, C_4 lui n'est impliqué que dans deux d'entre elles. Toutefois, sa position sur la carte (loin de la seule sortie primaire de la carte) affecte largement l'observabilité de ses broches et explique le fait qu'il est le moins testable de tous. Il est donc important de noter que les boucles ne sont pas le seul paramètre influant sur la testabilité des noeuds et des circuits.

En incluant dans ces circuits (un à un) des fonctionalités BS, nous obtenons les trois classifications (b), (c) et (d) de la figure 5.14. Il est intéressant de signaler que les meilleurs BTAs sont obtenus lorsque C_6 ou C_1 sont dotés de BS et non C_4 comme on aurait pu l'imaginer, étant donné qu'il est le mois testable dans la classification (a). Ceci s'explique par le fait que rajouter du BS dans C_6 ou C_1 permet d'ouvrir virtuellement trois boucles (le noeud de retour (t) devient complètement testable), alors qu'en rajouter dans C_4 ne permet d'en ouvrir que deux. On voit donc, que lorsqu'on doit choisir un seul circuit à modifier, priorité doit être accordée au circuit le moins testable et susceptible d'ouvrir virtuellement un maximum de boucles sur la carte. Dans ce cas, le circuit à choisir est C_6.

Supposons maintenant que les contraintes de conception permettent d'inclure du BS dans deux circuits à la fois. Nous commençons par le faire avec les deux circuits C_6 et C_4 (5.14-e), mais les meilleurs BTAs sont obtenus avec les paires C_6-C_7 (5.14-f) et C_6-C_5 (5.14-g). En effet l'ajout de BS dans C_5 ou dans C_7 permet d'ouvrir virtuellement une quatrième boucle ce qui ne peut être obtenu avec C_6 ni avec C_4. On observe donc à présent que, lorsqu'on peut transformer plus d'un seul circuit, il faut accorder la priorité à ceux permettant d'ouvrir virtuellement le plus grand nombre de boucles, et non seulement aux moins testables.

D'autre part, le BTA de la paire C_6-C_5 est supérieur à celui de la paire C_6-C_7, étant donné qu'une interconnexion existe entre C_6 et C_7, et que l'ajout de deux cellules BS sur la même interconnexion n'est pas vraiment utile, surtout qu'un test d'interconnexion n'est pas vraiment envisageable sur une carte comprenant uniquement deux circuits BS. On peut donc déduire que lorsque le nombre de circuits BS est suffisant pour permettre un test d'interconnexions sur la carte, il est préférable d'ajouter du BS dans les circuits éloignés les uns des autres, afin d'augmenter la testabilité globale de la carte.

Outre la paire C_6-C_5 nous avons tenté d'ajouter du BS à un troisième circuit. Le meilleur résultat est obtenu pour la combinaison C_6-C_5-C_2. Rappelons que C_2 était le moins testable dans la classification 5.14-g (après transformation des circuits C_6 et C_5 en circuits BS). On peut en déduire qu'une fois le problème d'ouverture virtuelle des boucles résolu, et en tenant compte du problème d'espacement des circuits BS sur la carte, le fait de donner la priorité aux circuits les moins bien placés dans les classements intermédiaires de testabilité reste un critère important.

Les quatre observations mises en évidence à partir de cette étude de cas représentent une interprétation de l'influence de l'inclusion de structures de test BS dans un circuit en tenant compte de l'étude de testabilité. Elles constituent des heuristiques efficaces pour déterminer en priorité les lieux d'ajout de fonctionnalités de test, lorsqu'il n'est pas possible, ou souhaitable, de le faire de façon exhaustive.

5.3 Conclusion

Ce chapitre détaille une méthode d'évaluation de testabilité dédiée aux cartes et aux MCMs. Toutefois, cette méthode peut être étendue au niveau des systèmes intégrés surtout s'ils incluent des composants virtuels ("IP cores"). En effet, les seules informations (concernant les constituants du module à tester : circuits, composants virtuels, puces d'un MCM ou autres) que réclame cette méthode sont d'ordre externe. L'élaboration de cette méthode a été motivée par la nécessité d'évaluer la testabilité globale d'une carte (ou d'un MCM), et de détecter les noeuds difficiles à tester dans leurs architectures. Néanmoins la méthode d'évaluation que nous proposons peut être utilisée indépendamment du processus d'aide à la conception de systèmes testables.

Pour valider cette méthode, en l'absence d'environnement efficace pour le test des cartes et surtout d'un ATPG pour cartes, nous l'avons testée au niveau circuit (description logique) en la comparant à SCOAP. La comparaison a montré que les résultats obtenus concernant la contrôlabilité des noeuds sont presque équivalents. Néanmoins, des différences apparaissent au niveau de l'observabilité, parce que les deux méthodes utilisent des concepts différents pour son calcul. Globalement, les deux méthodes ont fourni des résultats comparables. Pour l'exemple présenté, les portes du circuit ont été considérées comme étant des circuits et le circuit lui-même comme étant une carte.

Un exemple d'utilisation de la méthode au niveau carte a été présenté, il permet de voir combien il peut être utile d'évaluer la testabilité avant ou pendant un traitement de conception en vue de la testabilité d'une carte ou d'un système. En effet, cette évaluation permet de détecter les noeuds les plus difficiles à tester, pour y inclure prioritairement des fonctionalités de test (ex: BS) et améliorer ainsi la testabilité globale.

Chapitre 6

Présentation de l'outil STA

6.1 Introduction

Dans ce chapitre, nous présentons l'outil STA ("System Testability Assistant") [MM98a] dont le rôle est de mettre en application la méthodologie d'aide à la conception de systèmes testables discrets que nous avons présentée dans le chapitre 4. En plus du fait qu'il est en charge du contrôle du déroulement des différentes étapes de la méthodologie, STA essaie le plus possible de concrétiser les quatre principes sur lesquels elle repose, à savoir l'interactivité, la modularité, la réutilisation et l'évaluation.

Le principe d'interactivité est mis en évidence notamment grâce au système de boîtes de dialogues qui représente l'ossature de l'outil. En effet, tout au long du traitement, les boîtes de dialogues représentent le principal moyen de communication entre l'outil et l'utilisateur. Ce dernier se trouve régulièrement amené à prendre des décisions et à faire des choix à partir des propositions qui lui sont fournies.

La possibilité d'interfacer le noyau de STA avec des outils externes de DFT, de SFT et d'analyse de testabilité, ou internes tels les modules d'évaluation de testabilité et de génération de contrôleurs de test (carte ou système), confère à l'outil son aspect modulaire.

L'ensemble des contrôleurs de test que nous avons définis et synthétisés, tels le contrôleur du bus IEEE1149.1 (de la simple interface au contrôleur complet, voir paragraphe 4.8.2) ou encore celui du bus IEEE1149.5 (maître et esclave, voir paragraphe 4.9.1) peuvent s'intégrer sous forme de bibliothèques dans STA, confirmant ainsi le principe de réutilisation. Dans la même optique, les algorithmes internes de STA, dont le rôle est de proposer des stratégies de test au concepteur, privilégient souvent la réutilisation de structures de test dédiées à certains modules du système pour le test d'autres modules.

Le module d'évaluation met en oeuvre la méthode d'analyse de la testabilité que nous avons proposée dans le chapitre 5, concrétisant ainsi le quatrième et dernier principe de l'outil, à savoir l'évaluation.

6.2 Architecture globale de STA

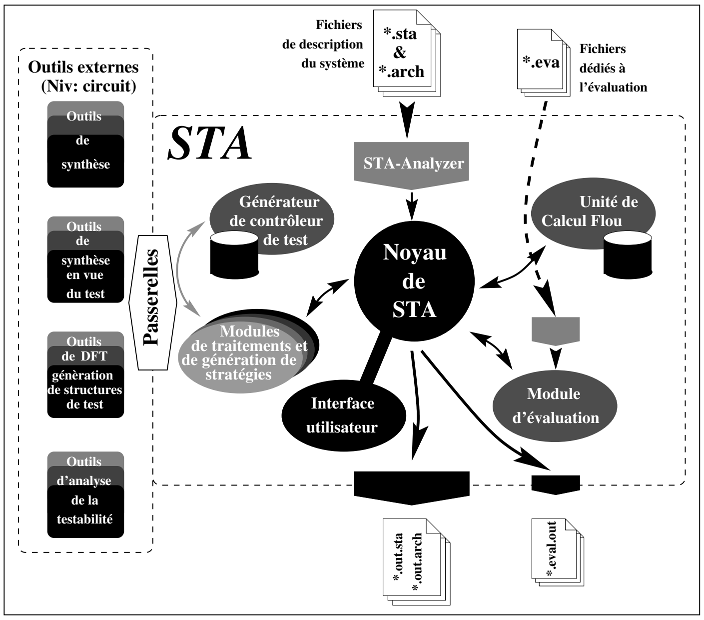

Fig. 6.1 – Architecture globale de l'outil

L'architecture globale de l'outil est représentée par la figure 6.1. Nous distinguons essentiellement:

– Le *noyau de l'outil*, qui communique avec tous les autres modules et qui contrôle l'implémentation de l'algorithme général de la méthode, présenté dans le paragraphe 4.2. Une bibliothèque d'interprétations et de messageries lui est dédiée.

– Le module *"STA-Analyzer"*, dont le rôle est d'analyser les fichiers ".sta" contenant la description du système à traiter, et ".arch" contenant les architectures des cartes et MCMs qui le constituent. Cette analyse lui permet de générer la structure de données qui subira le traitement, selon le modèle orienté objet présenté dans le paragraphe 4.3.2.

- L'*unité de calcul flou*, dont le rôle est de guider l'utilisateur à prendre des décisions concernant le type de l'architecture globale de test à adopter (distribuée/centralisée) et la nature du contrôleur de test système (matériel/logiciel). Cette unité implémente la méthode présentée dans les paragraphes 4.6 et 4.7 et basée sur la modélisation en logique flou des différents paramètres influant sur ces décisions. Cette unité est accompagnée d'une bibliothèque d'interprétations (voir paragraphe 4.6.4).

- Le module *d'évaluation*, dédié à l'analyse de la testabilité des cartes et des MCMs du système selon la méthode présentée dans le chapitre 5. Ce module prend en entrée, entre autres, les fichiers ".eval" qui contiennent les informations concernant la testabilité des circuits et surtout la description des cônes de dépendances.

- Un module *générateur de contrôleur de test*, dont le rôle est de générer les contrôleurs carte ou système selon les choix faits par l'utilisateur. Ce générateur est accompagné d'une bibliothèque de modules élémentaires de contrôle pré-définis et pré-générés.

- Les modules de *traitement et génération de stratégies*, qui sont au nombre de quatre: le module de traitement au niveau circuits, le module de pré-traitement des cartes (MCMs), le module de traitement au niveau carte (MCMs) et le module de traitement au niveau système (présentés dans le chapitre 4). Ces modules communiquent avec les autres modules de STA à travers le noyau. Ils sont également responsables de toutes les transformations à apporter au niveau de la structure de données (ex. changement de l'état d'un circuit, d'une carte, d'un MCM, etc.)

- L'*interface utilisateur*, qui est une interface graphique permettant d'établir le dialogue entre l'outil et l'utilisateur.

- Des *"drivers"* pour la génération, à la fin du traitement, des fichiers de structure (".out.sta") et d'architecture (".out.arch") de la version testable du système, ainsi que les fichiers relatifs au module d'évaluation (".eval.out", ".Cclass" et ".Nclass").

- un ensemble de passerelles, pour communiquer avec d'autres outils de CAO (synthèse, SFT, DFT, analyse de testabilité, etc.) qui s'intéressent généralement au niveau circuit.

6.3 description des différents modules

Actuellement, mis à part le générateur de contrôleurs et les passerelles vers les outils circuit, tous les autres modules ainsi que le noyau central ont été implémentés et sont fonctionnels. Dans cette section nous allons donc essayer de présenter les modules les plus importants de façon concise et selon les fonctions qu'ils exécutent. Afin de rendre la description des différents éléments de l'outil plus claire, nous commencerons pas une présentation rapide de l'interface graphique.

6.3.1 Présentation de l'interface graphique

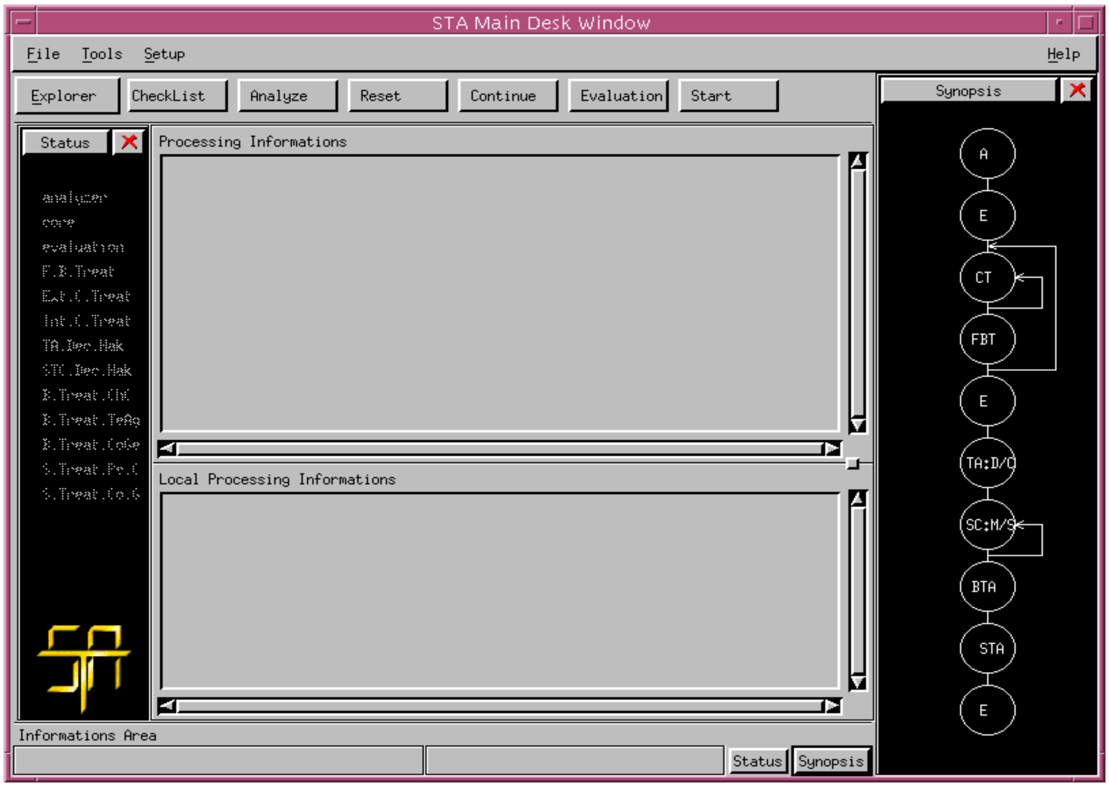

Fig. 6.2 – Fenêtre principale de STA

L'interface graphique est un élément essentiel de l'outil STA, puisque elle permet à l'utilisateur de mieux contrôler le déroulement du traitement, notamment grâce aux différentes fenêtres d'information et de contrôle qu'elle fournit. La figure 6.2 représente une copie de l'écran principal de l'interface, dans lequel nous distinguons essentiellement:

- La fenêtre "Processing Informations" dans laquelle sont affichées de façon séquentielle les informations d'ordre général, comme: l'étape actuelle de traitement, la carte sous traitement, le circuit sous traitement, etc.

- La fenêtre "Local Processing Informations" dans laquelle sont affichées (séquentiellement) les informations d'ordre local, c'est-à-dire les informations relatives à une étape particulière du traitement.

- La fenêtre "Synopsis" dans laquelle est représenté un synopsis de l'algorithme principal. A chaque début d'étape, le cercle la désignant s'allume, permettant ainsi à l'utilisateur de visualiser l'avancement du traitement. La signification des initiales inscrites dans les cercles est la suivante:

  - A : "Analyzer" : Analyzeur de fichier ".sta" et ".arch".
  - E : "Evaluation" : Module d'évaluation.
  - CT : "Circuit Treatment" : Traitement des circuits.
  - FBT : "First Board Treatment" : Pré-traitement des cartes et des MCMs.
  - TA:D/C : Architecture de Test: distribuée ou centralisée?
  - SC:H/S : Contrôleur Système: logiciel ou matériel?
  - BTA : "Board Treatment Achievement" : Traitement final des cartes.
  - STA : "System Treatment Achievement" : Traitement final du système.

- La fenêtre "Status" dans laquelle sont indiqués les modules (ex. TA:D/C) et sous-modules (ex. calcul flou) en fonction. C'est le même principe que celui adopté dans le "Synopsis" mais avec plus de précision.

- Les boutons de commandes tel le bouton "RESET" qui sert à annuler toutes les opérations faites sur le système et à revenir au point de départ, ou encore le bouton "CONTINUE" qui sert à reprendre le traitement là ou il a été arrêté la dernière fois. Les fonctionnalités des autres boutons seront décrites au fur et à mesure de la présentation des modules de l'outil.

6.3.2 Le noyau de l'outil

Le noyau est le module qui gère le fonctionnement général de l'outil: déroulement de l'algorithme principal, communication avec les différents modules, gestion de la structure de données, etc. Son déroulement se manifeste à l'utilisateur à travers la mise à jour des fenêtres "STATUS" et "SYNOPSIS", et surtout à travers les messages qui peuvent apparaître dans la fenêtre "Processing Informations" et certaines boîtes d'informations tel les avertissements ("Warning!") ou les rappels.

L'activation du noyau se fait par le bouton "START", qui enclenche le traitement à partir de l'étape de traitement des circuits. Il est toujours possible de suspendre le traitement pour effectuer d'autres opérations (telle une évaluation ou simplement une consultation de fichiers), et de le reprendre à l'aide de la commande "CONTINUE".

6.3.3 Le module STA-A "STA-Analyzer"

Ce module est responsable de l'analyse des fichiers contenant les descriptions des différents éléments du système et leurs architectures, et de la génération de la structure de données objet selon le modèle objet décrit précédemment. Tous les traitements ultérieurs seront subis par cette structure de données objet qui, une fois le traitement finalisé, sert d'entrée pour les drivers qui génèrent les fichiers de sorties décrivant le système dans sa nouvelle configuration testable.

Les fichiers d'entrée sont de deux types, les fichiers ".sta" décrivant la structure du système en cohérence avec les caractéristiques des classes et objets du modèle adopté, et les fichiers ".arch" décrivant les architectures des cartes et des MCMs. Ces fichiers sont écrits dans deux grammaires propres à l'outil. Les figures 6.3 (a) et 6.3 (b) présentent deux exemples de fichiers ".sta" et ".arch". Les fichiers d'entrée, tout comme les fichiers de sortie, peuvent être consultés à travers l'interface graphique à l'aide d'un explorateur (bouton: "EXPLORER").

Actuellement, la structure d'un système doit être décrite dans un seul fichier "nom-système.sta", toutefois il est possible d'imaginer une analyse hiérarchique d'un ensemble de fichiers du type: "nom-système.sta", "nom-carte1.sta", "nom-carte1-mcm1.sta", ...,"nom-carte2.sta",... "nom-carteN.sta".

Les architectures des cartes peuvent être décrites dans un ou plusieurs fichiers ".arch". L'architecture d'un MCM d'une carte donnée peut être décrite dans un fichier à part ou dans le fichier dédié à la carte. Etant donné qu'ils sont déclarés dans le fichier ".sta", les différents fichiers d'architectures sont analysés de façon automatique lors de l'analyse de celui-ci.

L'analyseur effectue entre autres des vérifications de cohérence entre le fichier ".sta" et les fichiers ".arch" et entre ces derniers eux-même. En cas d'anomalie, il le signale à

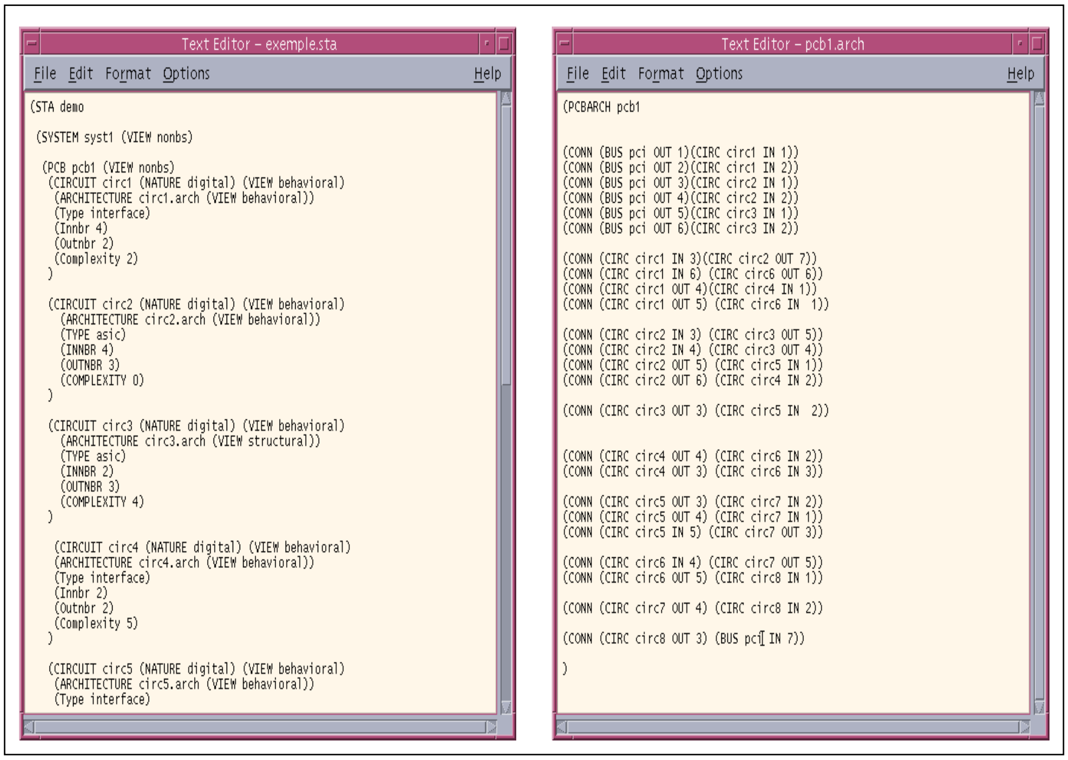

Fig. 6.3 – Exemples de fichiers ".sta" et ".arch"

l'utilisateur à travers un message d'erreur indiquant à quel niveau celle ci s'est produite.

En ce qui concerne la génération des structures de données, il important de signaler que l'analyseur génère deux structures complètement indépendantes: la structure principale (orientée objet) décrivant le système (figure 6.4) et la structure de données représentant les architectures (connectique), qui est beaucoup plus simple.

Le lancement de STA-A se fait après l'ouverture du fichier à analyser (FILE/OPEN "nom-système.sta") en cliquant sur le bouton "ANALYZER". Pour vérifier si la description du système a été bien analysée, il suffit d'effectuer un listing de la structure en cliquant sur le bouton "CHECK-LIST". Le listing en question apparaît dans la fenêtre "Processing Informations", il indique le nombre de cartes dans le système, les circuits de chaque carte, les puces de chaque MCM (assimilées à des circuits), etc. (figure 6.5)

L'analyseur a été implémenté en C++ et est actuellement complètement opérationnel. Outre sa fonction principale d'analyse, il inclut les drivers qui génèrent les fichiers de sortie après traitement (à partir de la structure de données traitée). Plus de détails concernant ce module sont disponibles dans [MM97].

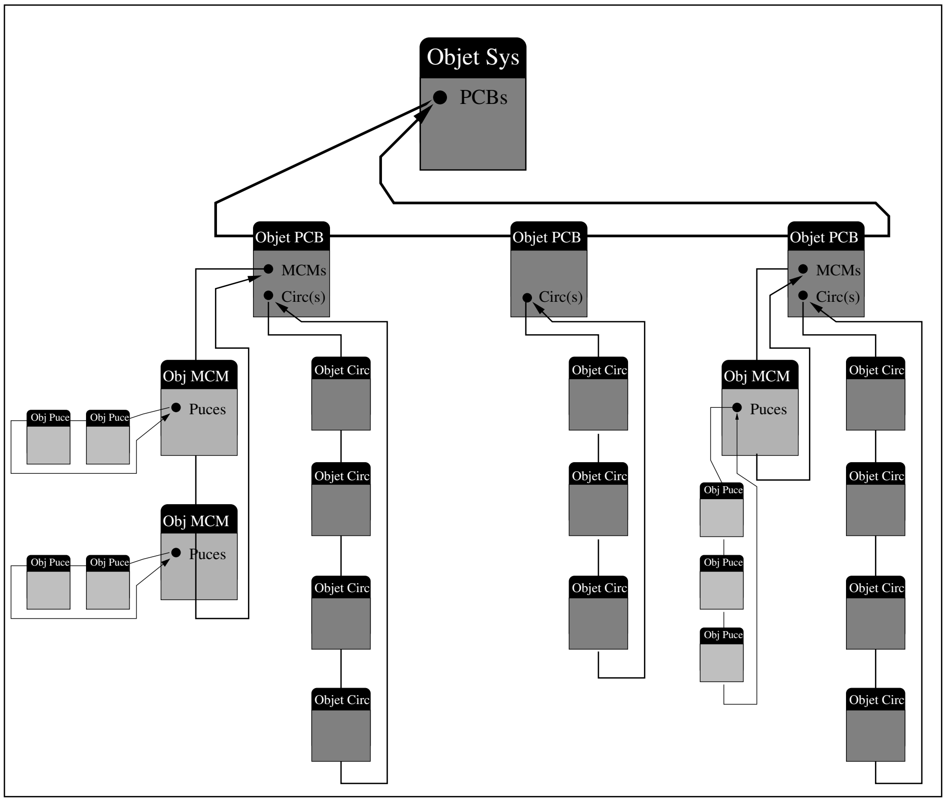

Fig. 6.4 – Structure objet générée par STA-A

6.3.4 Le module d'évaluation

C'est le module responsable de l'analyse de testabilité des cartes et des MCMs. Il prend en entrée les fichiers ".eval" dans lesquels sont indiqué les informations concernant la testabilité interne des circuits (à travers les facteurs OBF et DPCF présentés dans le paragraphe 5.2.1) et la description des cônes de dépendances des différents circuit. Un exemple de fichier ".eval" est donné dans la figure 6.6.

Le résultat de l'évaluation de la testabilité est fournit sous forme de trois classes de fichiers:

- Les fichiers ".eval.out" contenant les valeurs des métriques de testabilité de chaque noeud d'une carte ou d'un MCM.
- Les fichiers ".Nclass" indiquant la classification des noeuds d'une carte (ou MCM) selon leurs testabilités.
- Les fichiers ".Cclass" indiquant la classification des circuits d'une carte (ou MCM) selon leurs testabilités.

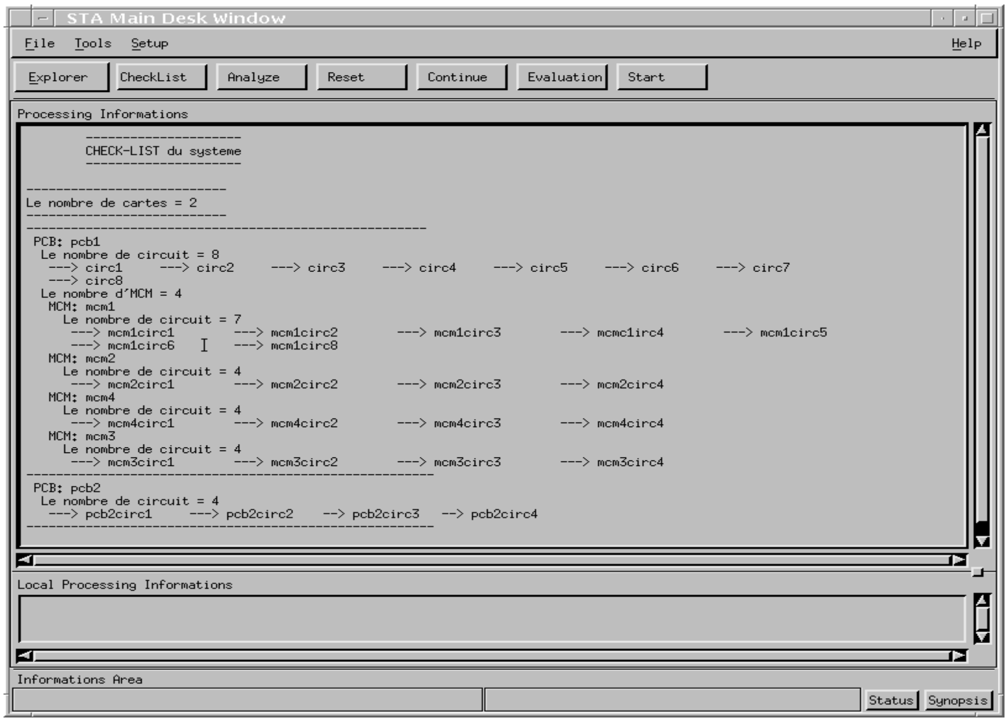

Fig. 6.5 – Exemple de "CHECK-LIST"

Des fichiers intermédiaires de moindre utilité sont générés au cours de l'étape d'évaluation, ils concernent essentiellement la détection de boucles au niveau des architectures des cartes ou des MCMs à évaluer.

Dans le cas où une carte intègre un ou plusieurs MCMs, l'utilisateur a le choix entre une mise à plat des architectures des MCMs avant l'analyse de testabilité ou bien une analyse hiérarchique où les résultats de l'analyse des MCMs (considérés donc comme des circuits) sont rapportés automatiquement au niveau de l'analyse de la carte. Ce choix doit être indiqué au niveau des entêtes des fichiers ".eval".

Au cours du traitement, en plus des trois cas où le lancement du processus d'évaluation est automatique (voir l'algorithme principal du traitement présenté dans le paragraphe 4.2), l'utilisateur peut réclamer l'évaluation d'une ou de plusieurs cartes, en cliquant sur le bouton: "EVALUATION".

Ce module est actuellement complètement implémenté et intégré dans STA. Plus de détails sont disponibles dans [MM98b].

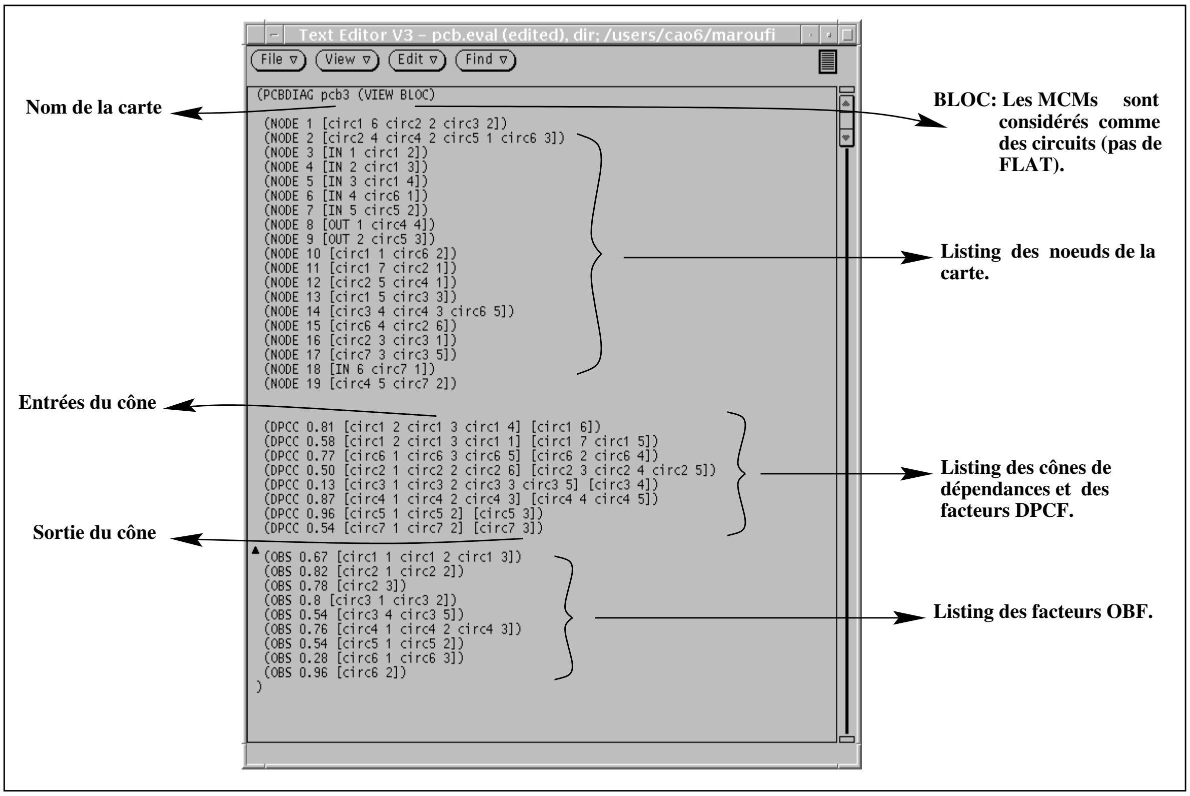

Fig. 6.6 – Exemple de fichier d'évaluation

6.3.5 L'unité de calcul flou

Cette unité intègre les deux mécanismes de prise de décision concernant le type de l'architecture globale de test (distribuée/centralisée) et la nature du contrôleur système (matériel/logiciel). Son lancement se fait après les étapes de traitement circuit et de pré-traitement des cartes. L'utilisateur aura à saisir (à travers des boîtes de dialogues) les paramètres qui entrent en jeu tels la rapidité souhaitée pour le test, le coût du matériel, etc., ainsi que ses préférences (voir paragraphes 4.6 et 4.7). Certains paramètres, telles la complexité moyenne des cartes ou l'importance de leur nombre, sont évidemment estimés de façon interne.

En ce qui concerne l'implémentation, hormis la banque d'interprétations, toutes les autres parties de ce module sont complètement implémentées et fonctionnelles.

6.3.6 Les modules de traitement et de génération de stratégies

Ces modules sont, comme nous l'avons déjà signalé, au nombre de quatre: le module de traitement au niveau circuit, le module de pré-traitement des cartes, le module de traitement au niveau carte et le module de traitement au niveau système.

Le module de traitement au niveau circuit

Ce module sert à guider l'utilisateur pour l'insertion des fonctionnalités de test dans les circuits de son système. L'état de chaque circuit est affiché dans la fenêtre "Local Procssing informations", et selon cet état (description RTL, logique, circuit existant, circuit déjà testable, etc.), une succession de boîtes de dialogues permet de mettre en oeuvre la méthode présentée dans la section 4.4. La figure 6.7 présente quelques exemples de boîtes de dialogues. Dans le cas des circuits existants auxquels du BS sera rajouté sous

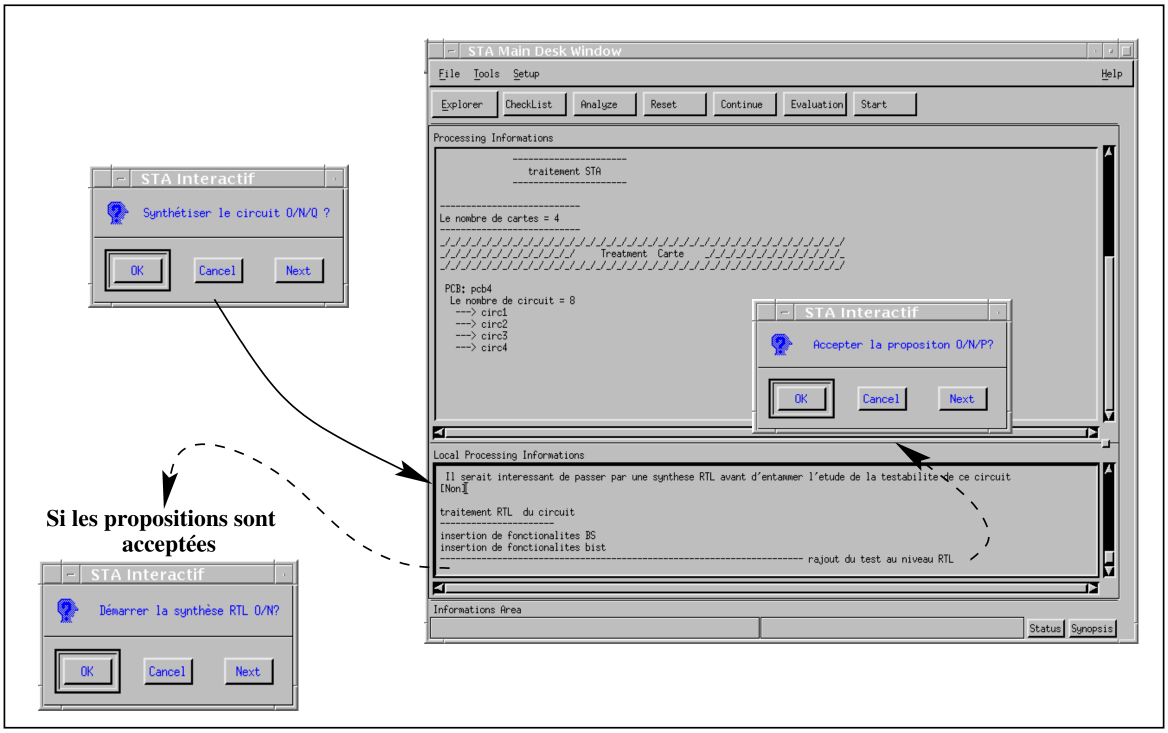

Fig. 6.7 – Exemple de boîtes de dialogue relatives au module traitement au niveau circuit

forme de registres BS externes, l'utilisateur doit indiquer le nombre et les caractéristiques des registres dont il dispose et qu'il peut rajouter sur la carte sans l'encombrer. Suite à des opérations internes relatives aux trois traitements: par défaut, par compensation et exhaustif présentés dans le paragraphe 4.4.1, un rapport lui sera fourni, indiquant l'affectation des registres selon chaque traitement. Ce rapport est accompagné de trois fichiers correspondants aux trois traitements et indiquant comment les registres doivent être connectés sur la carte. L'utilisateur aura donc le choix d'accepter la solution qu'il pense la plus appropriée et même d'y introduire des modifications s'il le juge nécessaire, en intervenant directement sur le fichier correspondant (qui se transforme en fichier d'entrée pour l'outil). Cette partie est en cours d'implémentation.

Le module Pré-traitement des cartes

Ce module effectue, d'une manière tout à fait interne (invisible pour l'utilisateur), l'ensemble des opérations présentées dans le paragraphe 4.5, à savoir la mise à jour de la structure de données, la détection des contrôleurs de test déjà prévus par le concepteur et l'ensemble des instructions BS supportées par les circuits BS implantés sur la carte. Ce module est actuellement complètement implémenté et fonctionnel.

Le module de traitement des cartes

La `pseudo-centralisation` représente la première tâche dont ce module est responsable. Le sous-module qui réalise cette tâche, qui est actuellement complètement défini, et en cours d'implémentation. Outre la `pseudo-centralisation`, les deux tâches principales de ce module sont, comme nous l'indiquons dans la section 4.8, l'agencement des structures de test au niveau d'une carte et la génération des contrôleurs de test. Ces deux opérations se font de façon complètement interactive entre l'outil et l'utilisateur, grâce un ensemble de boîtes de dialogue. Ces boîtes de dialogue (dont quelques exemples sont présentés dans la figure 6.8) indiquent à l'utilisateur quel type de contrôleur il faut implanter sur la carte (simple interface ou contrôleur intelligent/ contrôleur BS ou autre, si c'est un contrôleur BS alors quelles caractéristiques doit il avoir, etc.). Le problème d'agencement des structures de test est, lui, largement lié à celui de génération de structures de test, étant donné que pour un contrôleur généré, il faut prévoir son emplacement (traiter le problème d'encombrement), construire les chaînes de test et les relier, tenir compte des chaînes intégrées dans certains circuits (essentiellement les MCMs et les SoCs) et qu'il doit contrôler, etc. Ceci est aussi facilité par un ensemble de boites de dialogues ainsi que des messages d'attention ("Warning") ou d'erreur.

Tout au long du traitement, des informations concernant les opérations effectuées sont affichées dans la fenêtre "Local Processing Informations". A la fin, les structures de données sont mise à jour (structure et architecture de la carte).

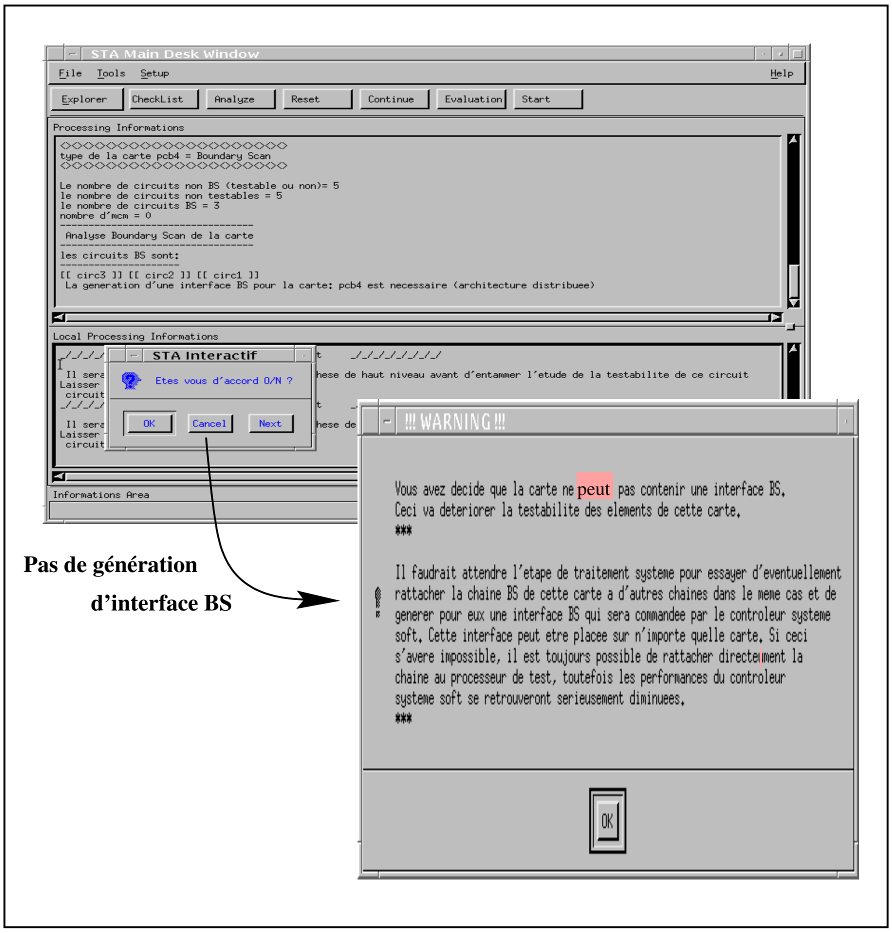

Fig. 6.8 – Exemple de boîtes de dialogue relatives au module traitement au niveau carte

Concernant l'implémentation, mis à part la génération des contrôleurs et le traitement des problèmes d'encombrement, toutes les autres parties du module sont actuellement fonctionnelles. En ce qui concerne le problème de génération de contrôleurs, nous prévoyons deux possibilités: générer du code VHDL (particulièrement du code RTL) ou bien générer directement des "netlist". Un environnement complet de génération d'architecture propre à STA est envisageable.

Le module de traitement système

Ce module est, entre autres, en charge de la tâche principale du traitement au niveau système, à savoir, la génération du contrôleur système dans le cas où il a été décidé qu'il soit intégré en matériel (paragraphe 4.9.1). Tout comme le module précédent, celui ci fonctionne de façon interactive avec l'utilisateur (boîtes de dialogues, messages d'attentions, message d'erreur).

Le sous-module responsable de la génération du contrôleur système sera semblable à celui du module de génération de contrôleur carte (ils seront vraisemblablement implémentés ensemble, au sein d'un même environnement pour la génération de contrôleurs de test). Le module de traitement système n'est donc pas encore complètement fonctionnel.

6.4 Conclusion

Dans ce chapitre, nous avons donc essayé de décrire succinctement l'outil STA qui met en oeuvre la méthodologie d'aide à la conception des systèmes testables que nous présentons dans les chapitres 4 et 5.

STA concrétise l'aspect interactif de la méthodologie, grâce à un système de boîtes de dialogues et de fenêtres d'informations et de mises en garde. Par son aspect modulaire, STA tient plus de l'environnement que du simple outil, en outre, il ne peut en aucun cas être assimilé un outil de DFT ou de SFT. L'originalité de STA est sa capacité à étudier la testabilité d'un système à n'importe quel stade du processus de conception et donc la possibilité de l'utiliser très tôt afin de prévoir quelles fonctionnalités de test il faut inclure dans les différents éléments et selon quelle stratégie ceci doit être fait. L'apport d'un outil comme STA dans le milieu industriel peut être très intéressant étant donné que le fait de définir assez tôt l'architecture de test d'un système permet de limiter le nombre de boucles de retour que peut causer la conception du test dans la chaîne complète de conception et donc d'économiser du temps et par conséquent de limiter les coûts de conception.

L'outil n'est pas actuellement complètement implémenté, certains modules sont encore en cours de développement et l'état d'avancement peut être estimé à 80 %.

Trois modules d'extensions peuvent être envisagés:

- Un module pour la gestion de la testabilité des éléments analogiques que peut intégrer le système.
- Un module pour la gestion des éléments logiciels relatifs aux structures matérielles de test intégrées dans le système.
- Un module dédié à l'étude de la testabilité des SoCs et principalement à la proposition d'architectures de test.

Mis à part les deux premiers modules d'extension qui ne sont pas encore à l'ordre du jour, nous avons déjà commencé à nous intéresser au troisième, dans le cadre du projet européen MEDEA AT403. Plus précisément, nous nous intéressons aux solutions de test (architectures) autour du bus "PI-bus".

Un exemple complet d'un système traité par STA est disponible en annexe. Bien qu'il s'agisse d'un exemple que nous avons conçu, étant donné qu'un exemple vraiment réel ne peut être traité que dans un environnement industriel où les caractéristiques de chaque élément et surtout les circuits sont disponibles, cet exemple sert à montrer quelles sont les transformations que l'architecture du système subit tout au long du traitement.

Chapitre 7

Conclusion Générale

Nous présentons dans cette thèse une méthode d'aide à la conception des systèmes testables pilotée par l'outil STA ("System Testability Assistant"), dont le rôle est de guider l'utilisateur à faire ses choix en ce qui concerne les fonctionalités de test à inclure dans son système. La méthode que nous proposons essaie de résoudre une problématique à deux volets: le premier est de proposer à l'utilisateur des solutions de test au plus tôt lors du développement du système, afin d'optimiser au maximum le TTM ("Time To Market"), et le second est de garantir une architecture finale de test permettant une maintenance efficace du système.

Pour modéliser les systèmes à traiter, notre choix s'est porté sur un modèle orienté objet, pour les avantages qu'il procure en matière de lisibilité, de programmabilité et de facilité à adopter les techniques de réutilisation.

Le traitement s'effectue de façon ascendante partant de l'étude de la testabilité du circuit pour atteindre celle du système global.

Au niveau circuit, la tâche principale est de guider l'utilisateur vers des outils de synthèse, d'analyse de testabilité ou de générations de structures de test ; toutefois, l'inclusion de structures BS dans un circuit échappe à la règle, étant donné qu'elle se décide de façon interne au traitement, suite à une évaluation de la testabilité de la carte dans laquelle le circuit en question est implanté.

Au niveau carte, deux tâches principales sont à signaler, à savoir le contrôle du test et l'agencement des structures de test. De façon générale, les stratégies proposées sont fondées sur le standard BS IEEE1149.1 et ses dérivés, toutefois d'autres types de stratégies sont aussi envisageables.

Au niveau système, le traitement consiste principalement à assurer le contrôle global du test ce qui revient souvent à contrôler le bus IEEE1149.5 ou une extension du bus IEEE1149.1. Néanmoins, la stratégie proposée au niveau système peut ne pas reposer sur l'un ou l'autre des deux standards.

Deux phases importantes du traitement consistent à prendre une décision concernant le type de l'architecture globale de test, qui peut être distribuée ou centralisée, et la nature du contrôle au niveau système qui peut être logiciel ou matériel. Une analyse basée sur un calcul flou permet de guider l'utilisateur dans ces choix.

Une méthode d'évaluation de la testabilité des cartes (ou MCM) a aussi été présentée, et se base sur une approche quantitative qui permet de fournir des classements précis des noeuds ou des circuits d'une carte selon leur testabilité.

La méthode que nous avons présentée, tout comme l'outil STA qui la pilote, se charpentent autour de quatre principes fondamentaux à savoir:

- L'évaluation: Omniprésente lors du traitement, l'évaluation peut être externe (analyse de la testabilité des circuits) ou interne (analyse de la testabilité des cartes et des MCMs).

- L'interactivité: Toutes les prises de décisions se font de façon interactive entre l'outil et l'utilisateur.

- La modularité: La méthode suit une approche modulaire, qui se traduit au niveau de la structure même de l'outil, où le module principal est capable de communiquer avec d'autres modules d'évaluation, de synthèse, de génération de structures de test, de génération de contrôleurs de test, etc.

- La réutilisation: La méthode proposée privilégie la réutilisation de ressources préconçues et disponibles en bibliothèque, surtout au niveau du contrôle du test. D'un autre coté, il est possible que, dans certains cas, la stratégie proposée favorise l'utilisation des fonctionalités de test d'un élément du système (tel un contrôleur de test carte) pour le test d'un ou d'autres éléments (ex : la pseudo-centralisation), ce qui représente en soi une deuxième forme de réutilisation.

Achever l'implémentation de l'outil STA dont l'état d'avancement peut être estimé à 80 %, représente la perspective principale à court terme. A moyen et long termes, la définition et éventuellement l'implémentation de deux autres modules pour la gestions du test des éléments analogiques du système et pour la génération des éléments logiciels nécessaires au contrôle des fonctionnalités de test intégrées, sont envisageables.

Par ailleurs, vu que la tendance actuelle est surtout à la conception des systèmes intégrés (SoCs), nous allons essayer d'adapter certains modules de STA à ce type de systèmes. Actuellement, nous nous intéressons plus particulièrement au module d'évaluation de la testabilité prévu au départ pour les cartes et les MCMs, mais qui peut être parfaitement adaptable au SoCs incluant des "IP cores", qui peuvent être considérés comme étant des cartes et les "IP" comme étant des circuits. A long terme, un système équivalent à STA peut être envisagé pour ce type de systèmes. En ce sens, des travaux ont déjà démarré au sein du laboratoire LIP6/ASIM, et portent précisément sur la définition d'architectures testables pour systèmes intégrés autour du bus "PI bus".

Annexe A

Exemple complet

A.1 Introduction

Cette annexe détaille un exemple de traitement effectué par l'outil STA. L'objet de ce traitement est un système, composé de trois cartes, et ayant une architecture à bus. Ce système n'est pas un système réel, mais nous l'avons choisi pour notre exemple car il incarne la plupart des caractéristiques que nous pouvons rencontrer dans différents systèmes réels, à savoir:

- *Un bus système*: données, adresses et contrôle.
- *Des cartes*: compatibles avec le bus système.
- *Des circuits*: munis d'interfaces du type SRAM.
- *Des signaux*: unidirectionnels et bidirectionnels.
- *Des boucles fonctionnelles*: existent sur les cartes.

Les informations telles que les cônes de dépendance à travers les circuits, les facteurs DPCF et les facteurs OBF, sont supposées connues.

Le fait que STA nécessite certaines informations concernant les circuits (cônes de dépendances, testabilité, etc.) ne nous a pas permis d'étudier un exemple réel, ayant une fonctionnalité bien définie. En effet, ce type d'information n'est disponible que dans un environnement industriel, auquel nous n'avons pas facilement accès.

A.2 Présentation du système avant et au démarrage du traitement

Dans cette section nous présentons le système à travers les schémas de ses trois cartes, la structure de données générée par le module "**`STA-Analyzer`** au moment de la saisie de la description du système et la première évaluation de la testabilité que STA effectue de façon automatique avant le début du traitement.

A.2.1 Description schématique du système

Schéma de la première carte

La première carte, dont le schéma est présenté dans la figure A.1, joue le rôle de carte mère. En effet, elle contient le processeur et les blocs mémoire principaux du système. Pendant le traitement, nous ne nous intéresserons pas à la testabilité de ces derniers, que nous considérons testables. Ainsi, les signaux que fournit le processeur au bus sont considérés complètement contrôlables et observables. La partie de la carte qui nous intéresse est donc celle placée à gauche de la ligne interrompue dans la figure A.1. Dans cette partie, qui contient six circuits, **`circ1`** et **`circ6`** sont considérés comme étant des circuits existants, alors que nous supposons ne disposer que des descriptions RTL des quatre autres (de **`circ2`** à **`circ5`**).

Schéma de la deuxième carte

La deuxième carte, dont le schéma est présenté dans la figure A.2, contient cinq circuits. Mis à par **`circ1`**, qui est considéré comme circuit existant, nous supposons disposer des descriptions RTL des quatre autres (de **`circ2`** à **`circ5`**). Cette carte est connectée au bus de la carte mère (carte 1).

Schéma de la troisième carte

La particularité de cette carte, elle aussi connectée au bus de la carte 1, est qu'elle contient un MCM. Nous supposons que ce MCM est en cours de conception et que nous disposons des descriptions RTL des trois puces qu'il intègre. Par ailleurs, le circuit **`circ1`** est considéré existant.

A.2.2 Génération de la structure de données

La description du système, écrite dans le format de données de STA (fichiers **`*.sta`**), a été analysée par le module "**`STA-Analyzer`**". La structure de données, générée selon le modèle Orienté Objet (OO) adopté par STA, est schématisée dans la figure A.4.

Nous pouvons constater que le système, n'incluant pas encore de structures de test (sauf pour le processeur et les mémoires de carte 1), est instancié dans la classe *`System_Class`*. Il en va de même pour les cartes et le MCM de la carte 3, qui sont non testables, et tous instanciés dans la classe *`PCB/MCM_Class`*. Les circuits considérés comme existants, sont du type (" off-the-shelf ") et sont instanciés dans la classe *`CircDig_EXIST_Class`*. Tous les autres circuits sont instanciés dans la classe *`CirDig_RTL_Class`*. Aucun circuit n'inclut des caractéristiques de testabilité.

A.2.3 Première évaluation de la testabilité du système

Une première évaluation de la testabilité des différentes cartes a été effectuée automatiquement par STA. Pour chaque carte, deux tables sont dressées, la première présente la testabilité des différents noeuds et la deuxième celle des circuits. Le facteur **`BTA`** ("Board Testability Average") représente la testabilité globale de la carte.

Notons que les fichiers d'architecture (**`*.arch`**) ainsi que les fichiers d'évaluation (**`*.eval`**) de chaque carte ou MCM sont préalablement fournis à l'outil.

Evaluation de la testabilité de la carte 1

Testabilité des noeuds

Les résultats de l'évaluation de la testabilité de la carte 1 sont reportés dans la table A.1. Pour chaque noeud nous fournissons sa contrôlabilité, son observabilité et sa testabilité. Les résultats sont donnés dans un ordre croissant de testabilité. Nous constatons que les noeuds les moins testables sont les noeuds **29**, **30**, **32** et **33**. cela s'explique par le fait que toutes les boucles de cette carte transitent précisément par ces quatre noeuds. Ces boucles, qui sont au nombre de dix, mettent en jeu les trois circuits **`circ3`**, **`circ4`** et **`circ5`**.

Testabilité des circuits

La testabilité des circuits, dressée dans la table A.2, confirme les résultats obtenus pour la testabilité des noeuds. En effet, nous constatons que les circuits les moins testables sont: **circ3**, **circ4** et **circ5**, par lesquels transitent les dix boucles fonctionnelles de la carte.

Testabilité globale de la carte

La testabilité globale de la carte a été calculée: **BTA** = 0.550418.

Evaluation de la testabilité de la carte 2

Testabilité des noeuds

Les résultats de l'évaluation de la testabilité des noeuds de la carte 2 sont reportés dans la table A.3. Nous constatons que les noeuds les moins testables sont les noeuds **25** et **22** par lesquels transitent les deux boucles de la carte. Les noeuds **23** et **24**, sont légèrement plus testables, en effet, par chacun d'eux ne transite qu'une seule boucle. Il est à signaler que les deux boucles de la carte mettent en jeu les circuits: **circ3**, **circ4** et **circ5**.

Testabilité des circuits

La testabilité des circuits de la carte 2, dressée dans la table A.4, montre que **circ3** et **circ5** sont les moins testables de la carte. Toutefois, on pouvait s'attendre à ce que le circuit **circ4** soit classé avec eux, parmi les trois circuits les moins testables, mais ce n'est pas le cas. En effet, **circ4**, même s'il entre en jeu dans la constitution des boucles, admet plusieurs entrées ou sorties primaires, ce qui augmente sa testabilité.

Testabilité globale de la carte

La testabilité globale de la carte a été calculée: **BTA** = 0.606172.

Evaluation de la testabilité de la carte 3

Testabilité des noeuds

Avant d'entamer l'évaluation de la carte, STA met à plat l'architecture du MCM et l'intègre dans celle de la carte. Ainsi, seuls les noeuds **28**, **29** **30** et **31** du MCM constituent des noeuds supplémentaires de la carte. Les résultats de l'évaluation de la testabilité sont reportés dans la table A.5. Nous constatons que les noeuds les moins testables sont les noeuds **24**, **25 26** et **27**. En effet, parmi les quinze boucles fonctionnelles que compte cette carte, neuf boucles transitent par chacun de ces noeuds. Les quinze boucles de la carte mettent en jeu les trois circuits **circ1**, **circ5** et **circ6**.

Testabilité des circuits

La testabilité des circuits, dressée dans la table A.6, montre, comme prévu, que les circuits **circ5**, **circ6** et **circ1**, sont les moins testables de la carte. Cela s'explique, évidemment, par le fait qu'ils voient transiter toutes les boucles de la carte.

Testabilité globale de la carte

La testabilité globale de la carte a été calculée: **BTA** $= 0.544103$.

A.3 Evolution du système pendant le traitement

Dans cette section, nous présentons l'évolution du système tout au long du traitement, à travers la testabilité de ses différents modules et les transformations que subit sa structure de données OO.

Nous nous somme fixé certaines contraintes à respecter pendant le traitement:

- La testabilité globale des cartes doit être au minimum égale à 0.95 à la fin du traitement.
- Il n'est pas possible d'utiliser des "**buffers**" Boundary Scan externes pour les circuits existants.
- L'ajout de structures BS doit se faire sur un nombre minimal de circuits.

On comprend aisément les raisons induisant les contraintes 1 et 3. La deuxième contrainte nous permet de mieux montrer les qualités du systéme STA, puisqu'elle évite les solutions de facilité.

A.3.1 Etude de la testabilité des circuits

Avant d'entamer l'étude de la testabilité des circuits à l'aide d'un outil externe, nous avons effectué un traitement préalable pour chaque carte. Ce traitement a pour but de détecter où il faut ajouter de façon prioritaire du BS, afin d'obéir aux contraintes fixées au départ. L'idée est de trouver quelle combinaison de circuits, auxquels on a ajouté BS, donnerait la meilleure testabilité globale à la carte.

Ajout de structures BS dans la carte 1

Nous avons procédé à l'ajout de structures BS dans les circuits, à raison d'un seul circuit BS sur la carte, puis deux circuits BS, et ainsi de suite, jusqu'à ce que tous les circuits soient BS (sans compter bien sûr les circuits existants **circ1** et **circ6**). Pour chaque configuration, nous avons effectué une analyse de testabilité. Les résultats obtenus sont reporté dans la table A.7.

La principale constatation concerne les configurations où un seul circuit contient du BS. Le meilleur résultat est obtenu pour le circuit **circ3**, alors qu'il n'est pas le moins testable dans la configuration sans BS. Cela s'explique par le fait que, d'une part **circ3**, tout comme **circ5** et **circ4**, ouvre virtuellement toutes les boucles de la carte lorsqu'il est doté du BS, et que d'autre part, c'est **circ3** qui contient le plus de broches d'entrée/sortie. Cela permet, en effet, d'améliorer la testabilité d'un plus grand nombre de signaux sur la carte.

La contrainte de testabilité globale supérieure à 0.95 n'est satisfaite que pour une configuration à quatre circuits BS. Ce qui revient à dire qu'il faut ajouter du BS dans tous les circuits (sans compter les circuits existants).

Ajout de structures BS dans la carte 2

Nous avons effectué le même traitement, subi par la carte 1, à la carte 2. Le circuit **circ1**, supposé existant, n'a pas été touché. Les résultats sonr reporté dans la table A.8.

La première constatation concerne **circ4**, qui en incluant du BS donne un meilleur résultat que le circuit **circ2**, mieux classé que lui dans la configuration sans BS. En effet, **circ4**, tout comme **circ3** et **circ5** et contrairement à **circ2**, ouvre virtuellement toutes les boucles de la carte en incluant du BS. Par ailleurs, comparé aux circuit **circ3** et **circ5**, **circ4** admet plus de broches d'entrée/sortie, ce qui explique le fait qu'il donne un meilleur résultat que les deux autres circuits.

La contrainte de testabilité globale supérieure à 0.95 est satisfaite dès les configurations à trois circuits BS. Le triplet qui donne le meilleur résultat est composé de: **circ3**, **circ4** et **circ5**. Il n'est donc pas nécessaire d'ajouter du BS dans le circuit **circ1**.

Ajout de structures BS dans la carte 3

Tout comme les cartes 1 et 2, la carte 3 a subi le même traitement d'évaluation de la testabilité globale selon les combinaisons de circuits BS. **circ1**, étant existant, n'a pas subi le traitement. Les résultats du traitement sont reportés dans la table A.9.

Au niveau des configurations à un seul circuit BS, le fait que **circ6** donne un meilleur résultat que **circ5** peut s'expliquer par le constat que, en lui ajoutant du BS, on améliore largement la testabilité des deux noeuds les moins testables de la carte, à savoir les noeuds **24** et **25**.

Au niveau des configurations à deux circuits BS, il est intéressant de constater que le meilleur résultat est obtenu pour la paire **circ6-Mcm1circ2**. Cela peut être expliqué par le fait que ces deux circuits, n'ayant qu'un seul noeud en commun, permettent d'augmenter la testabilité d'un plus grand nombre de signaux sur la carte.

La contrainte de testabilité globale supérieure à 0.95 est obtenue dès les configurations à quatre circuits BS. Le meilleur résultat est obtenu pour le quadruplet: **circ6**, **Mcm1circ2**, **Mcm1circ4** et **Mcm1circ3**. Il n'est pas donc nécessaire d'ajouter du BS dans le circuit 5.

A.3.2 Prise de décision concernant le test des circuits

Comme le prévoit la méthode, l'étude de la testabilité de chaque circuit est faite par un outil externe. Cela dit, grâce aux trois derniers traitement, STA guide l'utilisateur dans ses choix, en lui indiquant les recommandations concernant l'ajout de structure BS dans les circuits. Dans ce cas de figure, les recommandations sont donc:

- *Pour la carte 1*: ajout de BS dans les circuits **circ2**, **circ3**, **circ4** et **circ5**.
- *Pour la carte 2*: ajout de BS dans les circuits **circ3**, **circ4** et **circ5**.
- *Pour la carte 4*: ajout de BS dans les circuits **circ6**, **Mcm1circ2**, **Mcm1circ3** et **Mcm1circ4**.

Ces recommandations sont transmises à des outils externes à STA qui se chargent de les mettre en oeuvre, et de les compléter si nécessaire. Par exemple, l'outil externe responsable de l'étude de la testabilité des circuits peut recommander à son tour l'intégration de fonctionnalités BIST dans chacun d'eux.

D'autre part, les instructions BS, qui vont être permises par les structures BS des nouveaux circuits testables, peuvent être: **INTEGRITY**, **EXTEST** et **RUNBIST** pour permettre de lancer le test BIST à partir du bus IEEE1149.1.

La figure A.5 représente les transformations subies par la structure de données OO à la fin du traitement au niveau circuit et du pré-traitement des cartes.

Les tables A.10, A.11 et A.12 représentent la testabilité des circuits de chaque carte à ce stade du traitement, ainsi que les testabilité globales des cartes.

A.3.3 Choix de l'architecture globale de test

Une prise de décision concernant l'architecture globale du test est à présent nécessaire.

Les paramètres évalués directement par STA sont (il s'agit de valeurs obtenues par le calcul en logique floue, transcrites en termes qualitatifs pour une meilleure compréhension):

- La **complexité** moyenne du système: *moyennement complexe*.
- Le **nombre de cartes**: *pas du tout grand*.

Les paramètres introduits par le concepteur (notre choix dans le cas de cet exemple) sont:

- La **rapidité de test** souhaitée: *rapide*.
- Le **coût du matériel**: *moyennement coûteux*.
- l'**importance relative** coût/rapidité: *La rapidité est plus importante*.
- **Préférence d'architecture**: *distribuée*.

Résultat du calcul flou: Architecture *plutôt distribuée*.

Décision finale: **Architecture distribuée**.

Nous remarquons que le résultat du calcul flou n'est pas tranché (**plutôt distribuée**). Cela s'explique par le fait que la complexité moyenne n'est pas très importante et que le matériel est moyennement coûteux.

Concrètement, il s'agira de générer des contrôleurs de test intelligents pour chaque carte.

A.3.4 Choix de la nature du contrôleur système

Le choix de l'architecture globale de test étant effectué, celui de la nature du contrôleur système peut commencer.

Les paramètres évalués directement par STA sont:

- La **complexité** moyenne du système: *moyennement complexe*.

- Le **nombre de cartes**: *pas du tout grand*.
- L'**architecture globale de test**: *distribuée*

Les paramètres introduits par le concepteur (notre choix):

- La **rapidité de test** souhaitée: *rapide*.
- Le **coût du matériel**: *moyennement coûteux*.
- l'**importance relative** coût/rapidité: *La rapidité est plus importante*.
- La **robustesse**: *robuste*.
- **Préférence de contrôleur**: *forcément logiciel*.

Résultat du calcul flou: type contrôleur *logiciel*.

Décision finale: **logiciel**.

Le fait que l'architecture est distribuée implique déjà que le contrôleur système va être minimal. Ajouté au fait que le nombre de cartes n'est pas du tout élevé, cela explique le résultat du calcul flou qui penche vers la solution logicielle. Autrement dit, la rapidité du test ne sera pas vraiment affectée par la nature logicielle du contrôleur système.

Concrètement, il s'agira d'un programme logiciel qui s'exécute sur le processeur de la carte 1. Il aura à contrôler les contrôleurs cartes intelligents.

A.3.5 Traitement au niveau des cartes

L'étape de pseudo-centralisation est ignorée dans cet exemple, étant donné qu'elle ne revêt aucun intérêt ici.

Les registres BS des trois circuits du MCM de la carte 3, ainsi que celui du circuit **circ6** de la même carte, sont connectés les uns aux autres. Du fait qu'un seul circuit de la carte est BS, STA ne recommande pas l'intégration d'une nouvelle puce pour le contrôle du test BS dans le MCM (bien que l'architecture est distribuée). En effet, un seul contrôleur au niveau carte contrôlera à la fois le MCM et le circuit BS.

Les contrôleurs que STA propose pour les cartes sont des contrôleurs de type BSC autonomes (voir chapitre 5) intégrant:

- Des modules d'interface parallèle/série et série/parallèle.
- Une mémoire interne de service (la taille est à définir par l'utilisateur).

- Les algorithme de test BS: test d'intégrité, test externe et test BIST.

- Une interface système du type SRAM, pour communiquer avec le processeur sur le bus.

La taille approximative d'un tel contrôleur, muni d'une mémoire de 1KB, avoisine les 8000 portes.

A.3.6 Traitement final au niveau système

Du fait de la nature logicielle du contrôleur système, le traitement final au niveau système ne revêt plus beaucoup d'importance (pas de contrôleur système à générer).

La figure A.6 représente la structure de données OO définitive, à la fin du traitement. Nous constatons, entre autres, la migration des objet cartes vers la classe des cartes BS intégrant un contrôleur BS: *PCB/MCM+BS1149.1+BSC_Class*.

A.4 Conclusion

L'exemple que nous avons présenté, bien qu'il n'a pas une fonction réelle, permet de suivre les différentes étapes d'un traitement STA, à travers:

- Les transformations que subit la structure de donnée du système traité.

- Les évaluations de testabilité des différents éléments qui composent le système: avant, pendant et après le traitement.

- Les choix et les décisions pris pendant différentes étapes du traitement.

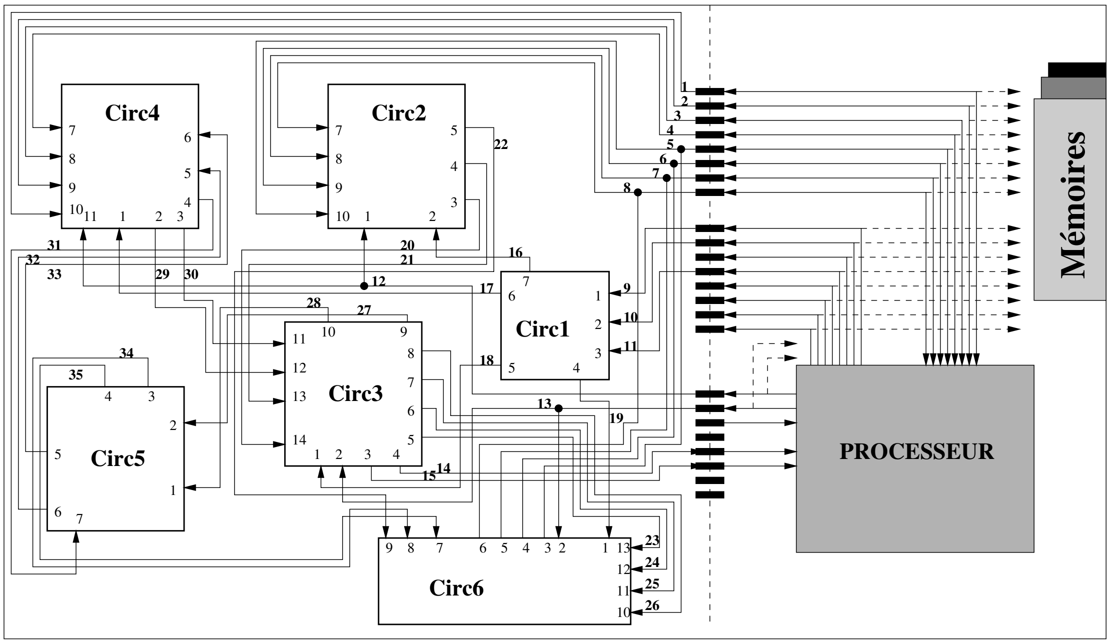

Fig. A.1 – Carte numéro 1

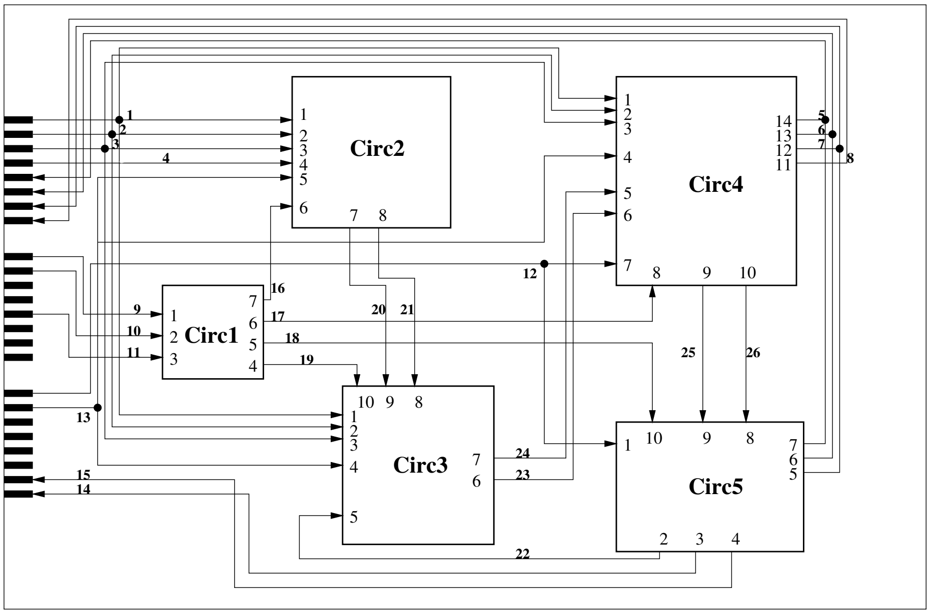

Fig. A.2 – Carte numéro 2

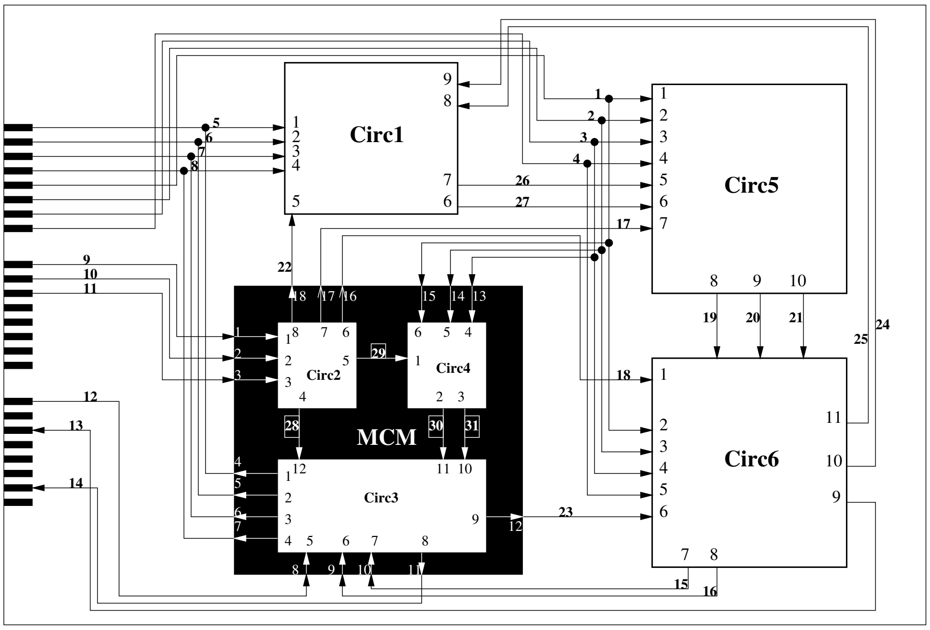

Fig. A.3 – Carte numéro 3

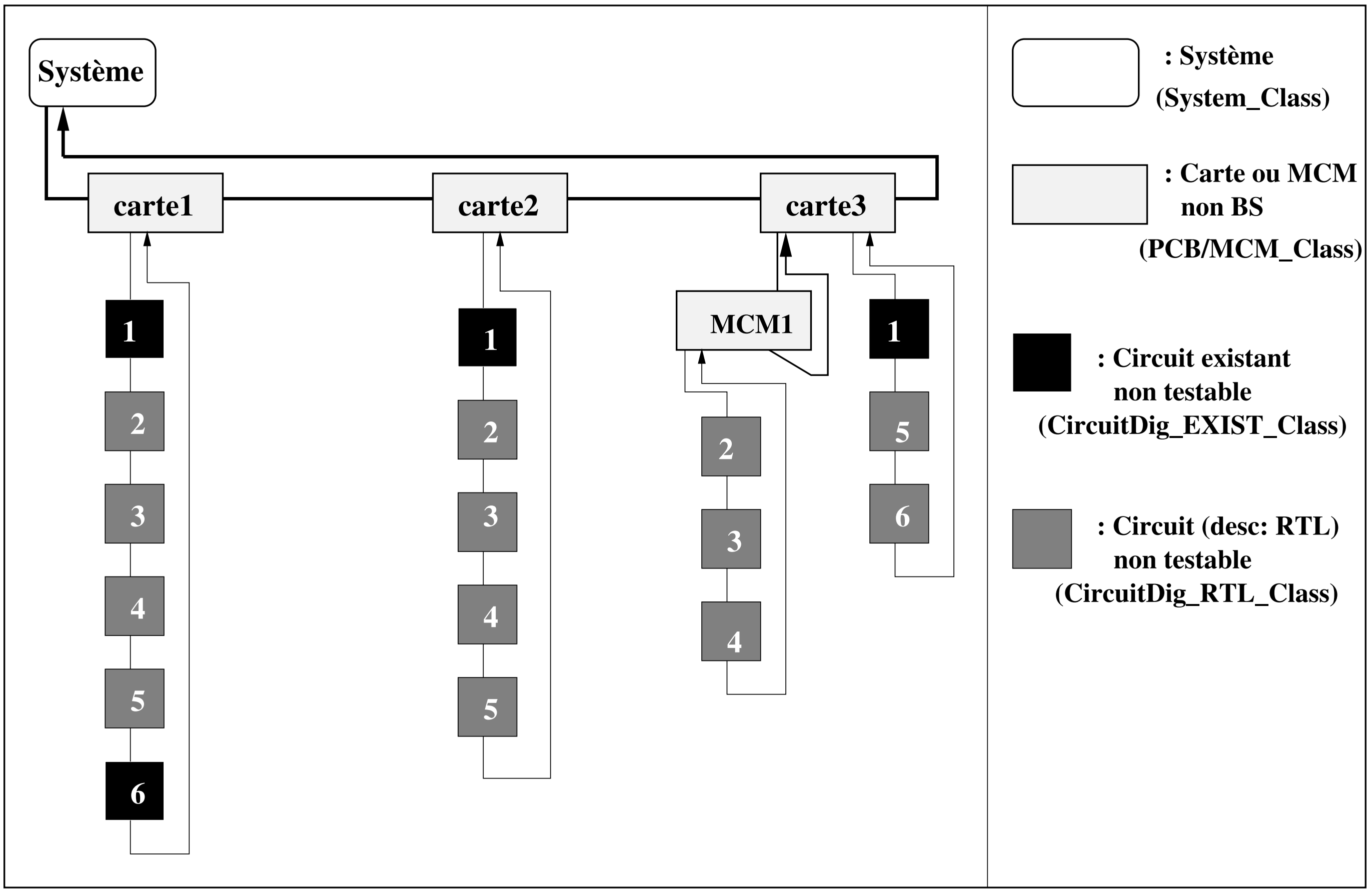

Fig. A.4 – La structure de données au départ du traitement

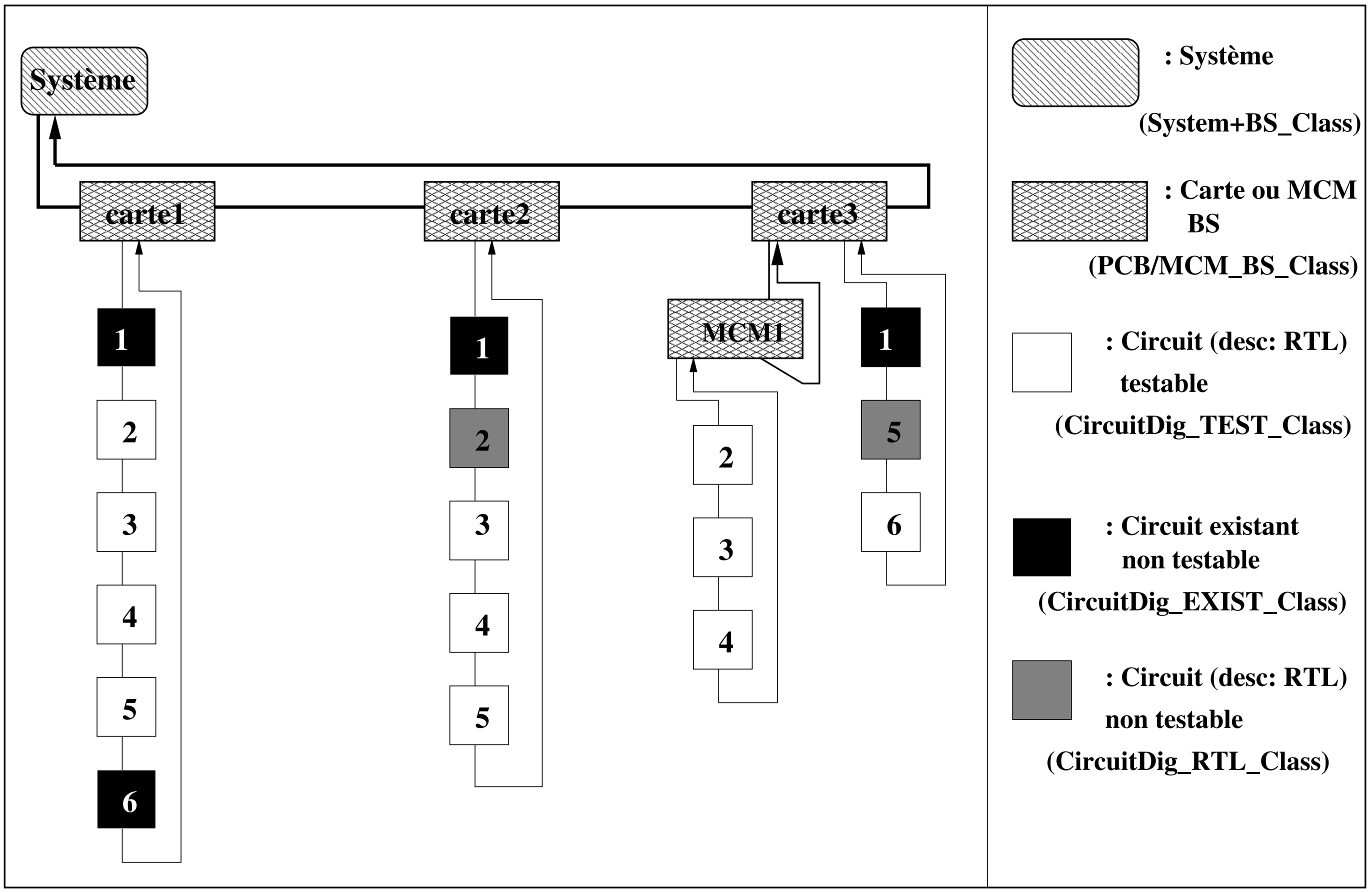

Fig. A.5 – La structure de données après transformation des circuits

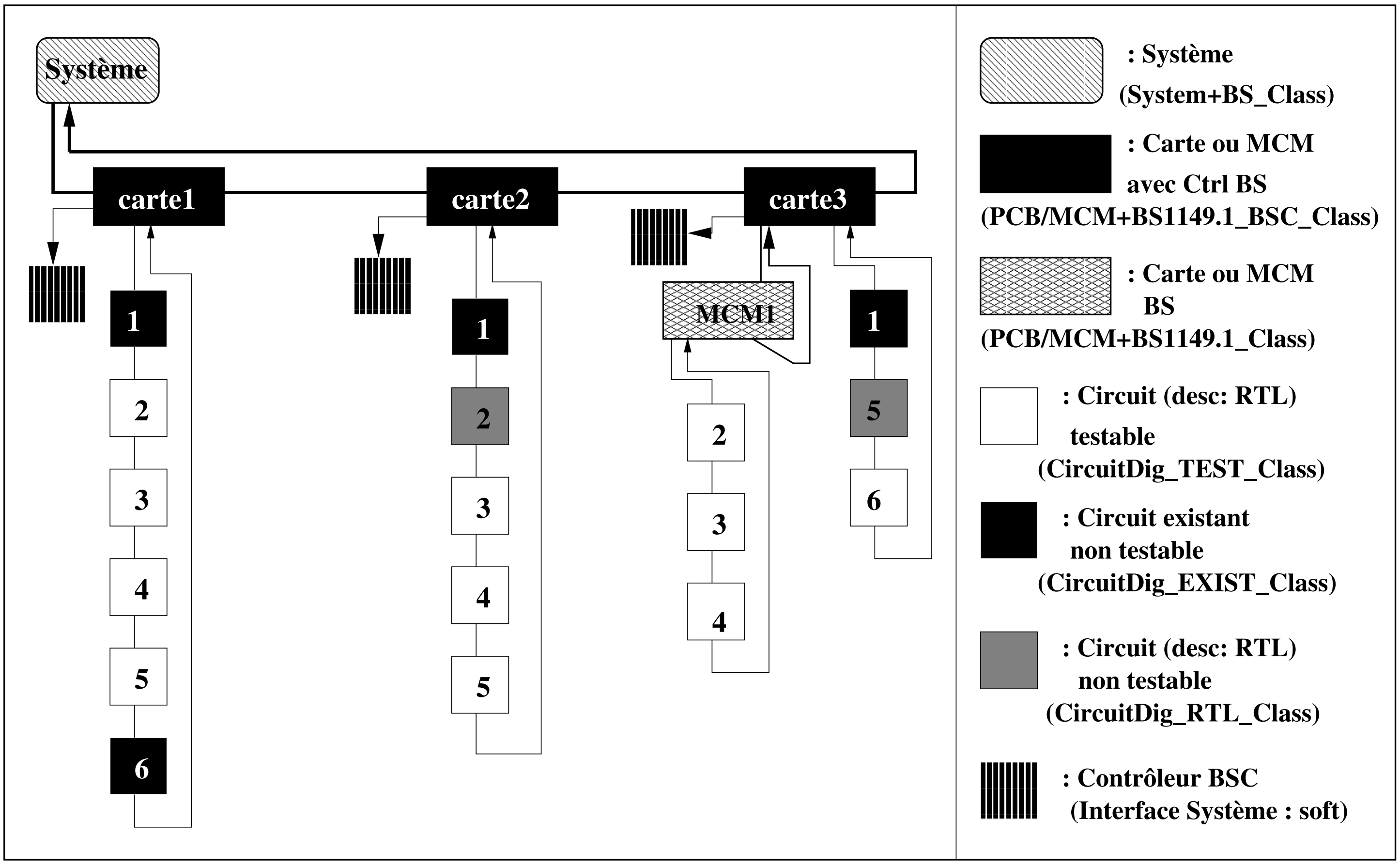

Fig. A.6 – La structure de données à la fin du traitement

Noeud	Contrôlabilité.	Observabilité.	Testabilité.
29	0.003778	0.010843	0.007310
30	0.003778	0.010843	0.007310
32	0.011529	0.014412	0.012970
33	0.014412	0.043980	0.029196
27	0.085899	0.009526	0.047713
28	0.085899	0.009526	0.047713
34	0.043980	0.800000	0.421990
35	0.043980	0.800000	0.421990
31	0.640000	0.528796	0.584398
17	0.800000	0.423037	0.611518
20	0.640000	0.601905	0.620953
21	0.640000	0.601905	0.620953
14	0.262144	1.000000	0.631072
15	0.262144	1.000000	0.631072
16	0.800000	0.534349	0.667175
5	0.500619	0.837173	0.668896
6	0.500619	0.837173	0.668896
7	0.534360	0.837173	0.685766
8	0.534360	0.837173	0.685766
18	0.800000	0.620953	0.710476
22	0.640000	0.800000	0.720000
23	0.640000	0.800000	0.720000
24	0.640000	0.800000	0.720000
25	0.640000	0.800000	0.720000
26	0.640000	0.800000	0.720000
9	1.000000	0.475668	0.737834
10	1.000000	0.475668	0.737834
11	1.000000	0.475668	0.737834
1	1.000000	0.534349	0.767175
2	1.000000	0.534349	0.767175
3	1.000000	0.534349	0.767175
4	1.000000	0.534349	0.767175
12	1.000000	0.534349	0.767175
19	0.800000	0.800000	0.800000
13	1.000000	0.800000	0.900000

Tab. A.1 – Testabilité des noeuds de la carte 1 selon un ordre croissant

Circuit	Testabilité.
circ5	0.223710
circ4	0.482992
circ3	0.501075
circ6	0.631841
circ1	0.714667
circ2	0.718328

Tab. A.2 – Testabilité des circuits de la carte 1 selon un ordre croissant

Noeud	Contrôlabilité.	Observabilité.	Testabilité.
25	0.000990	0.281313	0.141152
22	0.327680	0.150995	0.239337
23	0.054976	0.460800	0.257888
24	0.054976	0.460800	0.257888
26	0.001934	0.800000	0.400967
20	0.640000	0.368640	0.504320
21	0.640000	0.368640	0.504320
16	0.800000	0.294912	0.547456
19	0.800000	0.368640	0.584320
4	1.000000	0.294912	0.647456
5	0.320619	1.000000	0.660309
6	0.320619	1.000000	0.660309
7	0.320619	1.000000	0.660309
9	1.000000	0.424737	0.712368
10	1.000000	0.424737	0.712368
11	1.000000	0.424737	0.712368
18	0.800000	0.686799	0.743400
17	0.800000	0.773333	0.786667
8	0.640000	1.000000	0.820000
14	0.640000	1.000000	0.820000
15	0.640000	1.000000	0.820000
13	1.000000	0.720000	0.860000
12	1.000000	0.773333	0.886667
1	1.000000	0.800000	0.900000
2	1.000000	0.800000	0.900000
3	1.000000	0.800000	0.900000

Tab. A.3 – Testabilité des noeuds de la carte 2 selon un ordre croissant

Circuit	Testabilité.
Circ3	0.508535
Circ5	0.545351
Circ2	0.599172
Circ4	0.683669
circ1	0.685564

Tab. A.4 – Testabilité des circuits de la carte 2 selon un ordre croissant

Noeud	Contrôlabilité.	Observabilité.	Testabilité.
24	0.000792	0.001934	0.001363
25	0.000792	0.001934	0.001363
26	0.167772	0.000028	0.083900
27	0.167772	0.000028	0.083900
19	0.167772	0.014412	0.091092
20	0.167772	0.014412	0.091092
21	0.167772	0.014412	0.091092
17	0.800000	0.011529	0.405765
13	0.003022	1.000000	0.501511
23	0.262144	0.746667	0.504405
15	0.209715	0.800000	0.504858
16	0.209715	0.800000	0.504858
14	0.035184	1.000000	0.517592
30 (MCM)	0.640000	0.759467	0.699733
31 (MCM)	0.640000	0.759467	0.699733
29 (MCM)	0.800000	0.607573	0.703787
9	1.000000	0.476571	0.738286
10	1.000000	0.476571	0.738286
11	1.000000	0.476571	0.738286
1 *	0.604800	0.873000	0.738900
2 *	0.604800	0.873000	0.738900
3 *	0.604800	0.873000	0.738900
4 *	0.604800	0.873000	0.738900
5 *	0.631070	0.900000	0.765535
6 *	0.631070	0.900000	0.765535
7 *	0.631070	0.900000	0.765535
8 *	0.631070	0.900000	0.765535
28 (MCM)	0.800000	0.759467	0.779733
18	0.800000	0.800000	0.800000
22	0.800000	0.800000	0.800000
12	1.000000	0.759467	0.879733

Tab. A.5 – Testabilité des noeuds de la carte 3 selon un ordre croissant (* bidirectionnel)

Circuit	Testabilité.
Circ5	0.286990
Circ6	0.500244
Circ1	0.507836
Mcm1Circ3	0.634578
Mcm1circ2	0.713018
Mcm1circ4	0.752436

Tab. A.6 – Testabilité des circuits de la carte 3 selon un ordre croissant

Circuits BS	BTA
aucun	0.550418
circ5	0.800600
circ4	0.740342
circ3	0.829086
circ2	0.622365
circ3, circ4	0.912787
circ3, circ5	0.912545
Circ3, circ4, circ5	0.932423
Circ3, circ4, circ2	0.948087
Circ3, circ4, circ2, circ5	0.969180

Tab. A.7 – Evolution de la testabilité globale de la carte 1

Circuits BS	BTA
aucun	0.606172
circ3	0.852374
circ5	0.800523
circ2	0.707419
circ4	0.859065
circ3, circ4	0.929336
circ3, circ5	0.921170
Circ4, circ5	0.902371
Circ3, circ4, circ5	0.960816
Circ3, circ4, circ2	0.954126
Circ3, circ4, circ5, circ6	0.985714

Tab. A.8 – Evolution de la testabilité globale de la carte 2

Circuits BS	BTA
aucun	0.544103
circ5	0.778123
circ6	0.846121
Mcm1circ3	0.669475
Mcm1circ2	0.638129
Mcm1circ4	0.621873
Circ6, circ5	0.896586
Circ6, Mcm1circ3	0.903945
Circ6, Mcm1circ2	0.914325
Circ6, Mcm1circ4	0.896546
Circ6, Mcm1 total	0.957905
Circ5, Mcm total	0.955001
Circ5, Circ6, Mcm1 total	0.990496

Tab. A.9 – Evolution de la testabilité globale de la carte 2

Circuit	Testabilité.
Circ1	0.877143
Circ6	0.921538
Circ2	1.000000
Circ3	1.000000
Circ4	1.000000
Circ5	1.000000
BTA	0.969180

Tab. A.10 – Testabilité finale des circuits de la carte 1

Circuit	Testabilité.
Circ1	0.867500
Circ2	0.877143
Circ4	1.000000
Circ3	1.000000
Circ5	1.000000
BTA	0.960816

Tab. A.11 – Testabilité finale des circuits de la carte 2

Circuit	Testabilité.
Circ5	0.846800
Circ1	0.875556
Circ6	1.000000
Mcm1Circ4	1.000000
Mcm1circ2	1.000000
Mcm1circ3	1.000000
BTA	0.957905

Tab. A.12 – Testabilité finale des circuits de la carte 3

Bibliographie

[ABF90] M. Abramovici, M.A. Beuer, and A. D. Freidman. Digital systems testing and design for testable designs. Computer Science Press, 1990.

[AHTR97] G. Al-Hayek, Y. Le Traon, and C. Robach. Impact of system partitioning on test cost. IEEE Design and Test of Computers, pages 64–74, July 1997.

[Ait95] R. C. Aitken. An overview of test synthesis tools. IEEE Design and Test of Computers, 12(2):8–15, 1995.

[Avr91] L. Avra. Allocation and assignment in high-level synthesis for self-testable data paths. In International Test Conference, pages 463–472, Nashville, August 1991.

[BAMS85] P. H. Bardell, J. A. Abraham, WW. H. McAnney, and J. Savir. Built-In Test Theory and Implementations. Tutorial notes, 1985.

[BDS90] F. Beenker, R. Dekker, and R. Stans. Implementing macro test in silicon compiler design. IEEE Design and Test of Computers, pages 41–51, April 1990.

[Ben84] R. G. Bennetts. Design of Testable Logic Circuits. Addison Wesley, Massachusetts, 1984.

[Ben94] B. Bennetts. Progress in design for test: a personal view. IEEE Design and Test of Computers, 11(1):53–59, 1994.

[BGC88] M. A. Breuer, R. Gupta, and J. C>Lien. Concurrent control of multiples bit structures. In International Test Conference, pages 431–442, September 1988.

[Bha91] D. Bhavsar. An architecture for extending the ieee standard 1149.1 test access port to system backplanes. In International Test Conference, pages 768–776, Nashville, 1991.

[BMM98a] P. Bukovjan, M. Marzouki, and W. Maroufi. Allocation for testability in high-level synthesis process. In 5th Electronic Devices and Systems Conference, Brno, Czech Republic, 1998.

[BMM98b] P. Bukovjan, M. Marzouki, and W. Maroufi. Cost/quality trade-off in high-level synthesis for testability. In 2nd International Workshop on Design and Diagnostics of Electronic Circuits and Systems, Szczyrk, Poland, September 1998.

[BMM99] P. Bukovjan, M. Marzouki, and W. Maroufi. Testability analysis and cost/quality trade-off in synthesis for testability. In IEEE European Test Workshop, Constance, Germany, May 1999.

[BMR80] R. G. Bennetts, C. M. Maunder, and G. D. Robinson. Camelot: A computer aided measure for logic testability. In International Conference on Circuits and Computers, pages 1162–1165, 1980.

[BO91] R. G. Bennetts and A. Osseyran. Ieee standard 1149.1-1990 on boundary scan. Journal of Electronic Testing: Theory and Applications, 2 (1):11–25, Mars 1991. (Special issue on Boundary Scan).

[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.

[Boa95] IEEE Standard Board. IEEE std 1149.5-1995,"module test and maintenance (mtm) bus protocol", 1995. [1-55937-558-2][SH94333-NYL].

[Boo94] Grady Booch. Analyse et Conception Orientées Objets. Adisson Wesley, 1994.

[Bre78] M. A. Breuer. New concepts in automated testing of digital circuits. In EEC Symposium on CAD of Digital Electronic Circuits and Systems, pages 62–92, Brussels, 1978.

[Cox95] H. Cox. Synthesizing circuits with implicit testability constraints. IEEE Design and Test of Computers, 12(2):16–22, 1995.

[CW91] R. Camposano and W. H. Wolf. High-Level VLSI Synthesis. Kluwer Academic Publishers, Boston, 1991.

[CWS91] C. H. Chen, C. Wu, and D. G. Saab. Beta: Behavioral testability analysis. In IEEE Internationa Conference on Computer-Aided Design, pages 202–205, 1991.

[DPJ85] Demarco, T. Plauger, and P. J. Structured Analysis and System specification. Prentice Hall, November 1985.

[DTT98] O. P. Dias, L. C. Teixeira, and J. P. Teixeira. Metrics for quality assessment of testable hw/sw systems architectures. In European Test Workshop, pages 205–209, Sitges, Spain, May 1998.

[FAI98] FAIRCHILD. Non-inverting transreceiver with 25ohm series resistor. http://www.fairchildsemi.com/pf/SC/SCAN182245A.html, 1998. FAIRCHILD semiconductor.

[FAI99] FAIRCHILD. Scanpsc100f: Embedded boundary scan controller. http://www.fairchildsemi.com/pf/SC/SCANPSC100F.html, 1999. FAIRCHILD semiconductor.

[FPR98] M. L. Flottes, R. Pires, and B. Rouzeyre. Guided hardware sharing procedure for improving testability. In European Test Workshop, pages 113–117, Sitges, Spain, May 1998.

[FPRV98] M. L. Flottes, R. Pires, B. Rouzeyre, and L. Volpe. Low cost partial scan design: A high level synthesis approach. In 16th IEEE VLSI Test Symposium, pages 332–340, Monterey, California, April 1998.

[FWY78] S. Funatsu, N. Wakatsuki, and A. Yamada. Designing digital circuits with easily testable consideration. In Test Conference, pages 98–102, September 1978.

[GDWL92] D. D. Gajski, N. D. Dutt, A. C. H. Wu, and S. Y. L. Lin. High-Level Synthesis: Introduction to chip and System Design. Kluwer Academic Publishers, Boston, 1992.

[Goe99a] Goepel. Asc 16-(x). http://www.goepel.com/english/prod/bsc/asc16e.htm, 1999. GOEPEL Electronics.

[Goe99b] Goepel. Psc 1149.1-(x). http://www.goepel.com/english/prod/bsc/psce.htm, 1999. GOEPEL Electronics.

[Gol79] L. H. Goldstein. Controllability/observability analysis of digital circuits. IEEE Transaction on Circuits and Systems, CAS-26(9), September 1979.

[Gra79] J. Grason. Temas, a testability measurement program. In Design Automation Conference, pages 156–161, 1979.

[GT80] L. H. Goldstein and E. L. Thigpen. Scoap: Scandia controllability/observability analysis program. In 17th Design Automation Conference, Minnesota, June 1980.

[Gu96] X. Gu. RT Level Testability Improvement by Testability Analysis and Transformations. PhD thesis, Linkoping University, S-581 83 Linkoping, Sweden, 1996. Dissertation No. 414.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. Science of Computer Programming, pages 231–274, July 1987.

[Hau92] P. Haulent. Ge'nie Logiciel: Les me'thodes. Armand Colin, 1992.

[HK92] O. F. Haberl and T. Kropf. A chip solution to hierarchical and boundary scan compatible board level bist. In Second Great Lakes Symposium on VLSI, pages 16–21, Kalamazoo, February 1992.

[HK94] O. F. Haberl and T. Kropf. Self testable boards with standard ieee 1149.5 module test and maintenance (mtm) bus interface. In IEEE European Design and Test Conference, pages 220–225, 1994.

[HP97] J. Håkegård and Z. Peng. Design and synthesis of a generic board-level test controller. In 23rd Euromicro Conference, September 1997. short contribution.

[HPCN92] H. Harmanani, C. A. Papachristou, S. Chiu, and M. Nourani. Syntest: An environment for system-level design for test. In European Design Automation Conference, pages 402–407, 1992.

[HSA95] H. Hulvershorn, P. Soong, and S. Adham. Linking diagnostic software to hardware self test in telecom systems. In International Test Conference, pages 986–992, 1995.

[HW92] J. K. Hagge and R. J. Wagner. High-yield assembly of multi-chip modules through known-good ids and effective test strategies. Proceedings of the IEEE, 80(12):1965–1994, December 1992.

[Ibm98] Ibm. Testbench system. http://www.chips.ibm.com/services/testbench/, 1998.

[Ins98] Texas Instrument. Scan test devices with octal buffers. http://www.ti.com/sc/docs/products/logic/sn74bct8244a.html, 1998. SCOPE Testability Integrated-circuit Family.

[Jon91] Frans De Jong. Boundary scan test, test methodology, and fault modeling. Journal of Electronic Testing, 2(1):77–88, March 1991.

[JY91] N. Jarwala and C. W. Yau. Achieving board-level bist using the boundary scan master. In International Test Conference, pages 649–658, 1991.

[KAB86] J. Kalinowski, A. Albicki, and J. Beausang. Test control line distribution in self-testable vlsi circuits. In International Conference on Computer-Aided Design, pages 60–63, November 1986.

[KB92] K. T. Kornegay and R. W. Brodersen. An architecture for reconfigurable ieee 1149.n master controller board. In International Test Conference, pages 975–983, 1992.

[KMZ79] B. Koenemann, J. Much, and G. Zwiehoff. Built-in logic block observation techniques. In International Test Conference, pages 37–41, October 1979.

[LA97] M. W. Lin and J. R. Armstrong. Test planning for vhdl dsp models. In 1st IEEE European Test Workshop, Cagliari, Italy, May 1997.

[LA98] Lip6-ASIM. Alliance cad system. http://www-asim.lip6.fr/alliance/, 1998. University Paris 6 - France.

[LB89] J. C. Lien and M. A. Breuer. A universal test and maintenace controller for modules and boards. IEEE transaction on Industrial Electronics, 36:231–240, May 1989.

[Lee97] M. T. C. Lee. High-Level Test Synthesis of Digital VLSI Circuits. Artech House, Boston - London, 1997.

[LHM92] D. Landis, C. Hudson, and P. McHugh. Apllications of the ieee p1149.5 module test and maintenance bus. In International Test Conference, pages 984–992, 1992.

[LMT94] M. Lubaszewski, M. Marzouki, and M. H. Touati. A pragmatic test and diagnosis methodology for partially testable mcms. In MultiChip Module Conference, pages 108–113, 1994.

[Log98a] LogicVision. icbist: The most complete test solution for very deep submicron. http://www.logicvision.com/solution/icbist.htm, 1998.

[Log98b] LogicVision. Logicbist: The chip level solution. http://www.logicvision.com/solution/logicbist.htm, 1998.

[Log98c] LogicVision. Membist: The chip level solution. http://www.logicvision.com/solution/membist ic.htm, 1998.

[Mar97] W. Maroufi. Ordonnancement et contrôle du test dans un environnement de synthèse de haut niveau en vue de la testabilité. In Colloque CAO de circuits intégrés et systèmes, pages 234–237. Grenoble, France, 1997.

[Mau93] C. Maunder. A universal framework for managed built-in test. In International Test Conference, pages 21–29, 1993.

[McC84] E. J. McCluskey. Verification testing - pseudoexhaustive test technique. IEEE Transaction On Computers, C-33(6):541–546, June 1984.

[Men97a] MentorGraphics. Dft advisor. http://www.mentorgraphics.com/dft/catalog/dfta.html, 1997. DFT Products.

[Men97b] MentorGraphics. Lbist architect. http://www.mentorgraphics.com/dft/catalog/lbista.html, 1997. DFT Products.

[Men97c] MentorGraphics. Mbist architect. http://www.mentorgraphics.com/dft/catalog/mbista.html, 1997. DFT Products.

[MLT93] M. Marzouki, M. Lubaszewski, and M. H. Touati. Unifying test and diagnosis of interconnects and logic clusters in partial boundary scan boards. In International Conference on Computer Aided Design, pages 654–657, November 1993.

[MM88] D. Marca and D. McGowen. SADT: Structured Analysis and Design Technique. McGraw-Hill, 1988.

[MM97] W. Maroufi and M. Marzouki. Specification and early evaluation of system testability through object oriented modeling. In 1st IEEE European Test Workshop, Cagliari, Italy, May 1997.

[MM98a] W. Maroufi and M. Marzouki. Sta: a system testability assistant. In European Test Workshop, pages 237–238, Sitges, Spain, May 1998.

[MM98b] W. Maroufi and M. Marzouki. System testability evaluation with sta. In 10th International Conference on Microelectronics, Monastir, Tunisia, December 1998.

[MMAA97] M. Marzouki, W. Maroufi, V. C. Alves, and A. R. Antunes. Global test scheduling and control in a hl-sft environment. In 1st IEEE High-Level Design Validation and Test Workshop, November 1997.

[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.

[Moh97] F. Mohamed. Approche à base de logique floue pour le test et le diagnostic des circuits analogiques. Grenoble, France, 1997. Thèse de doctorat.

[MT90] C. Maunder and R. Tulloss. The test access port and boundary scan architecture. IEEE Computer Science Press, 1990.

[MT91] C. M. Maunder and R. E. Tullos. An introduction to the boundary scan standard: Ansi/ieee std 1149.1. Journal of Electronic Testing, 2(1):27–42, March 1991.

[Par90] M. Parot. Norme jtag/ieee 1149.1 standard: Port d'acce's de test et architecture boundary-scan. In SMTique, Connectique, ASIC, pages 187–199, Mars 1990.

[Par92] K. P. Parker. The Boundary Scan Handbook. Kluwer Academic, 1992.

[PH87] I. A. Pirbhai and D. J. Hateley. Strategies For Real-Time System Specification. Dorset House Publishing CO, 1987.

[PLL97] M. Perbost, L. L. Lan, and C. Landrault. Testability analysis of digital boards and mcms. In European Test Workshop, Cagliary, Italy, May 1997.

[PO90] K. P. Parker and S. Oresgo. A language for describing boundary scan devices. In International Test Conference, pages 222–234, 1990.

[SE95] R. Sedmak and J. Evans. A hierarchical design-for-testability (dft) methodology for the rapid prototyping of application-specific signal processors (rassp). In International Test Conference, pages 319–327, Washington D.C., USA, 1995.

[SHM91] D. Sterba, A. Halliday, and D. McClean. Atpg and diagnostics for boards implementing boundary scan. Journal of Electronic Testing, 2(1):89–98, March 1991.

[Soc98] IEEE Computer Socity. IEEE P1500 Standard for Embedded Core Test. Technical report, IEEE, http://grouper.ieee.org/groups/1500, 1998.

[Syn97] Synopsys. Test compiler family. http://www.synopsys.com/products/test/test comp cs.html, 1997. Synopsys, Inc.

[TP95] M. V. Tegethoff and P. Parker. Ieee std 1149.1: Where are we? where from here? IEEE Design and Test of Computers, 12(2):53–59, 1995.

[TR95] Y. Le Traon and C. Robach. From hardware to software testability. In International Test Conference, Washington D.C., USA, 1995.

[TR97] Y. L. Traon and C. Robach. Testability analysis of embedded data flow systems. In European Test Workshop, Cagliary, Italy, May 1997.

[Tur97] J. Turino. Test economics in the 21st century. IEEE Design and Test of Computers, pages 41–44, July 1997.

[UM89] J. G. Udell and E. J. McCluskey. Pseudoexhaustive test and segmentation: Formal definitions abd extended fault coverage results. In Digest of Papers, 19th International Symposium on Fault-Tolerant Computing, pages 292–298, June 1989.

[VSSvR94] H. P. E. Vranken, M. P. J. Stevens, M.T.M. Segers, and J. H. M. M van Rhee. System-level testability of hardware/software systems. In International Test Conference, pages 134–132, 1994.

[WCM87] K. D. Wagner, C. K. Chin, and E. J. McCluskey. Pseudorandom testing. IEEE Transaction on Computers, C-36(3):332–343, March 1987.

[WE77] T. W. Williams and E. B. Eichelberger. A logic design structure for lsi testability. In 14th ACM/IEEE Design Automation Conference, pages 462–468, 1977.

[WM85] P. T. Ward and S. Mellor. Structured Development For Real-Time Systems. Prentice Hall Englewood Cliffs, 1985.

[WM86] L. T. Wang and E. J. McCluskey. Complete feedback shift register design for built-in self-test. In International Conference On Computer-Aided Design, pages 56–59, November 1986.

[WP83] T. W. Williams and K. P. Parker. Design for testability - a survey. In Proceedings of the IEEE, pages 98–112, January 1983. Vol 71, No 1.

[YJ90] C. W. Yau and N. Jarwala. The boundary-scan master: Target applications and functional requirements. In International Test Conference, pages 311–315, 1990.

[Zor92] Y. Zorian. A universal testability strategy for multi-chip modules based on bist and boundary scan. In International Conference on Computer Design, pages 59–66, 1992.