MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE FERHAT ABBES
SETIF

MEMOIRE

Présenté à l'institut d'Electronique
pour l'obtention du diplôme de

MAGISTERE

Option: **CONTROLE INDUSTRIEL**

Par

M Abdelaâli BOUMAARAF

THEME

DEVLOPPEMENT DES TECHNIQUES DE TEST DES SYSTEMES NUMERIQUES

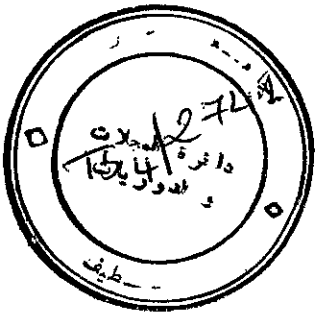
CONCEPTION D'UN PROCESSEUR DE TEST

Soutenu le / /1996
devant la Commission d'examen:

A.KHELLAF	M.C.Université de Sétif	Président
H.CHEMALI	C.C.Université de Setif	Rapporteur
A.BENBELKACEM	M.C.ENST de Kouba	Examinateur
R.E.BEKKA	M.C.Université de Sétif	Examinateur
B.BELHOUKI	C.C. Université de Sétif	Examinateur

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE FERHAT ABBES DE SETIF



MEMOIRE

Présenté à l'institut d'Electronique
pour l'obtention du diplôme de

MAGISTERE

Option: **CONTROLE INDUSTRIEL**

Par

M Abdelaâli BOUMAARAF

THEME

Développement des techniques de test des système numériques

CONCEPTION D'UN PROCESSEUR DE TEST

Soutenu le / /1996
devant la Commission d'examen:

A.KHELLAF	M.C.Université de Sétif	Président
H.CHEMALI	C.C.Université de Setif	Rapporteur
A.BENBELKACEM	M.C.ENST de Kouba	Examinateur
R.E.BEKKA	M.C.Université de Sétif	Examinateur
B.BELHOUKI	C.C. Université de Sétif	Examinateur

Dédicaces

Je dédie ce travail à :

mes chers parents,

mes frères et soeurs,

ma grande-mère,

mes oncles Smail, Moussa et Hamza,

ma tante Dhrifa,

toute la famille

et à tous mes amis.

Abdelaâli

Dédicaces

Je dédie ce travail à :

mes chèrs parents,

mes frères et soeurs,

ma grande-mère,

mes oncles Smail, Moussa et Hamza,

ma tante Dhrifa,

toute la famille

et à tous mes amis.

Abdelaâli

Remerciements

A l'issu de l'élaboration de la présente thèse, je me plie à cette aimable tradition, qui est de remercier toutes les personnes, qui de près ou de loin, par leur compréhension, leur coopération, m'ont facilité la tâche et ont contribué à la mise en forme du thème.

Mes vifs remerciements vont à Mr ***H. Chemali*** qui a proposé et dirigé ce travail. Je lui exprime ma profonde gratitude pour l'aide qui ma apportée durant ce travail par sa compétence, son encouragement et qui a donné sans compter de son temps et son savoir.

Je tiens à exprimer toute ma reconnaissance à Mr ***A. Khellaf***, maître de conférence à l'Institut d'Electronique de Sétif qui a accepté de me faire l'honneur de présider ce jury et juger ce modeste travail.

Je remercie au même titre, Mr ***A. Benbelkacem***, Docteur à l'ENST Kouba, Mr ***R. E. Bekka***, maître de conférence à l'Institut d'Electronique de Sétif et Docteur ***B. Belhouki***, chargé de cours à l'institut d'électronique Sétif, qui ont accepté d'être membres de jury, témoignant ainsi de l'intérêt qu'ils portent à ce travail.

Mes vifs remerciements vont également à Mr ***B. Fortas***, pour son aide matérielle et son encouragement.

Mes remerciements vont également à:

Mr ***D. Slimani***, Mr ***L. Selmani*** et ***W. Kara*** qui ont été pour moi une source de conseils et d'encouragement.

K. Djellal, ***T. Herihiri***, ***L. Assassi***, ***Z. Benghedfa*** et ***S. Baâzizi*** pour leur soutien au long de ce travail, ainsi tous mes collègues et ex-collègues du laboratoire de test ***D. Mabrouk***, ***L. Ziet***, ***H. Lakhlef***, ***A. Mayouf*** et ***H. Haddar***, pour leur collaboration enthousiaste et passionnée.

Enfin, dans le souci de n'oublier personne, que tous ceux qui m'ont aidé, de près ou de loin, que ce soit par leur amitié, leurs conseils ou leur soutien moral et matériel, trouvent dans ces quelques aimables et sincères mots l'expression de ma profonde gratitude.

A. Boumaâraf

Summary

In this work, a programmable test and diagnostic controller **(PTDC)** is developed for printed circuit board (PCB) interconnect testing in the context of boundary scan, scan and built-in-test features. The PTDC is a 60 instruction processing unit designed to take into account a variety of scan mechanisms with different cell's architecture by developing direct micro-control of the IEEE BS 1149.1 Test Access Port (TAP) through the introduction of a comprehensive instruction set.

Transmission of testing data and instructions via the test path for allowing multiple test functions and diagnostic procedures can be accomplished by the PTDC since its architecture is designed to fulfil testing exigency (test generation, response compaction) and overcome difficulties arising from using standard processors (testing time efficiency and pattern application capability).

A complete study of the PTDC and applications is presented. Two PTDC with hardwired and micro programmed control units are developed and validated using PALASM a CAD tool of Advanced Micro Devices.

Key words : Boundary scan, built-in-self test, test generation, compression, test controller.

Résumé

Dans ce travail, un contrôleur PTDC (**Programmable Test and Diagnostic Controller**) exploitant les mécanismes "Scan", Boundary Scan (BS) et "BIST" est développé pour le test et diagnostic des systèmes numériques modernes. Ce contrôleur (PTDC) de 60 instructions est conçu pour la prise en compte d'une grande variété d'architectures Scan et particulièrement celle de IEEE BS 1149.1, grâce à un contrôle direct des signaux TAP du BS. L'architecture du PTDC et son répertoire d'instructions sont adaptés à mieux répondre que les processeurs standards en ce qui concerne les exigences de transmission et traitement de données de test via le chemin Scan. Des instructions complètement inhérentes au test sont ainsi développées. La validation de deux PTDC (câblé et micro-programmé) est réalisée grâce à l'outil de développement PALASM de AMD.

Mots clés : Boundary scan, built-in-self test, génération de vecteurs de test, compression, contrôleur de test.

Sommaire

Illustrations de figures

Figure Titre ..... Page

Glossaire

ASIC (*Applied Specific Integrated Circuit)*
ASIP *(Applied Specific Integrated Processor)*
BEST *(Built-In Evaluation And Self-Test)*
BILBO *(Built In Logic Block Observer)*
BIST (*Built-In Self Test*)
BITE (*Built-In Test Equipment*)
CABSA (*Cellular Automata Based Signature Analyses*)
CFSR (*Complete Feed-back Shift Register*)
CM (*Control Memory*)
CMBD (*Control Memory Data Buffer*).
CN-LFSR (*Control LFSR*)
CP (*Control Points)*
CSBL *(Centralised And Separate Board-Level)*
CSTP *(Circular Self-Test Path)*
CUT (*Circuit under Test*)
DFT (*Design For Testability)*
DR (*Data Register*)
FIBS *(Fault Injection Boundary Scan)*
FID *(Fault-Injection Data)*
FIE *(Fault-Injection Enable)*
FT (*Functional Test*)
GTSA *(Group Theoretic Signature Analysis)*
ICT (*In Circuit Test*)
IR (*Instruction Register).*
LFSR (*Linear Feedback Shift Register*)
LOCST *(LSSD on chip self-test)*
LSSD (*Level Sensitive Scan Design*)
MCM (*Multiple Chip Module*)
MISA (*Multiple Input Shift Analyser*)
MISR (*Multiple Input Signature Register*)
MITAD (Module d'Isolation, de Test et d'Aide au Diagnostic)
MPC (*Micro-program Counter)*
MPC (*Micro-Program Counter)*
MPCU (*Microprogrammed Control Unit*)
OP (*Observation Points)*
PAL (*Plan Array Logic*)
PCB (*Printed Circuit Board*)
PG-LFSR (*Primary Generator LFSR*)
PIs (*Primary Inputs)*
POs (*Primary Outputs)*
PRPG (*PseudoRandom Pattern generator)*
PTDC (*Programmable Test And Diagnostic Controller*)
RTD (*Random test Data)*
RTS (*Random test Data)*
SDI (*Scan Data In*)
SDO (*Scan Data Out*)
SISA (*Single-Input Signature Analyser)*
SISR (*Single-Input Signature Register*)
SMT (*Surface Mount Technology*)
SP (*Scan Path)*
SRSG (*Shift-Register Sequence Generator*)
Stuck at 1/0 (Collage 1/0)
TAP (*Test Access Port*)
TCK (*Test Clock*)
TDC (*Test Diagnostic Control)*
TDI (*Test Data Input*)
TDO (*Test Data Output*)
TMS (*Test Mode Select*)
TPG (*Test Pattern Generation*)
UPG (*Unmatched Pattern Generation*)
VPA (Vecteur Pseudo-Aléatoirs)
WPG *(Weighted Test Generator)*

INTRODUCTION

Introduction

L'émergence de circuits intégrés à très haute densité MCM (*Multiple Chip Module*) et l'exploitation extensive de circuits spécifiques montés en mode SMT (*Surface Mount Technology*), durant ces dernières années, ont imposé aux différentes opérations de la chaîne de conception l'inclusion de certaines structures et règles, pour non seulement des fins de confort d'exploitation et d'assurance de fonctionnement, mais aussi pour une amélioration et simplification des opérations de maintenance. L'acceptation presque unanime, par la communauté micro-électronique, de la DFT structurée (*Design For Testability*) comme activité d'intérêt majeur est principalement dûe à la souplesse des mécanismes et stratégies de test introduits au sein du circuit intégré lui même tout en préservant la forme originale.

Il est à noter, que les actions entreprises sont diverses, mais se convergent à la mise en oeuvre d'outils software et hardware qui allègent les opérations de test dès les premières phases de conception. Les multitudes sources d'erreurs dépendantes du processus de fabrication, de conception et du champ d'application réduisent considérablement les chances de réaliser un diagnostic parfait. Alors il y a eu recours aux mécanismes de chemins de données Scan [1], BS (*Boundary Scan*) [21], BIST (*Built In Self Test*) [1] et BIST-BS [62,63] qui représentent aujourd'hui les techniques les plus répandues et dont les actions au niveau des Cls se traduisent généralement par l'addition de circuiteries supplémentaires pour assurer de meilleures contrôlabilités et observabilités des différents noeuds dans un circuit et donc achever des procédures de test performantes. L'accroissement de vitesse, de densité d'intégration et notamment la diversité de nouveaux produits miniaturisés imposent aussi aux ATEs (*Automatic Test Equipment*) des contraintes très difficiles à prendre en charge pour garantir des tests efficaces. Alors il y a orientation de travaux pour concevoir des ATEs conformes et adaptables aux dernières exigences introduites par la philosophie de test elle-même d'une part, et pour résoudre les restrictions technologiques et évolutives qui l'accompagnent d'autre part.

Nous présentons dans le cadre de ce travail, une étude sur les techniques DFT (*Design For Testability*), BIST (*Built In Self Test*), l'étude du standard de test BS IEEE 1149.1 introduit en 1991 par le JTAG BSS (*Joint Test Action Group Boundary Scan Standard*) et devenu depuis un des maillons indispensables à la conception de circuits testables, les techniques de test par isolation et injection de fautes, pour faire surtout ressortir et recenser les véritables problèmes inhérents à ces méthodes et pouvoir ensuite proposer un contrôleur sous la forme d'un module de test, MCM ou système, qui est appelé à apporter les atouts suivants en mode d'exploitation et de conception :

- son introduction au sein d'un système évite la re-conception pour des fins de test,
- exploite lui même les avantages du BS et d'auto testabilité,
- exécute des routines très adaptées aux exigences du standard BS et recommandées pour le test de systèmes à plusieurs cartes.
- ce processeur peut aussi effectuer les tâches de génération et de compression étroitement liées à la réduction du temps de test et du volume d'information à traiter.

Les différentes tâches accomplies dans ce travail sont résumées ci-dessous :

Dans le premier chapitre, on présente une recherche sur les techniques de test DFT qui concernent les circuits, les cartes et les systèmes modernes et où les cellules de test, l'observabilité, la contrôlabilité, la couverture de test et le temps de test représentent les paramètres de base. Les mécanismes de chemins de données telles que Scan Path, LSSD (*Level Sensitive Scan Design*), Scan Path partiel, Boundary Scan et structure du standard IEEE BS 1149.1 sont aussi expliqués dans ce chapitre.

Dans le deuxième chapitre, on présente une étude détaillée sur les techniques d'auto test comprenant :

L'étude des opérations de compression et de génération ;

l'étude des cellules scan : BIST centralisé et séparé, test aléatoire à prise (RTS), LOCST (*LSSD on Chip Self-Test*), BIST concurrent, test aléatoire des données (RTD), le système d'analyse de test cyclique (CATS), l'auto test à chemin circulaire (CSTP), BILBO (*Built In Logic Block Observer*), CABSA (*Cellular Automata Based Signature Analyser*) ; les techniques de test avancées telles que :

l'association Boundary Scan et BIST,

l'auto test de cartes (PCB) utilisant l'architecture BS,

la cellule BS à injection de fautes

et le MITAD (Module d'Isolation, de Test et d'Aide au Diagnostic).

Une étude sur le contrôle câblé et micro-programmé étant exigée pour développer notre processeur, alors nous avons mis en évidence les avantages et inconvénients des différentes approches dans le troisième chapitre. Après un organigramme, définissant les tâches et fonctions que doit accomplir le contrôleur à concevoir, nous avons présenté les blocs diagrammes de deux processeurs PTDC (*Programmable Test And Diagnostic Controller*). Le processeur à unité de contrôle câblée est présenté au quatrième chapitre alors que le cinquième chapitre est réservé au processeur à unité de contrôle micro-programmée.

Les différents modules qui composent les deux PTDC sont analysés et ensuite développés dans les chapitres 4 et 5.

Une recherche des micro-instructions qui forment le répertoire des PTDC est d'importance capitale, car c'est sur la base de son architecture et son répertoire d'instructions que repose toute la stratégie de test à l'aide de notre processeur conçu. Les architectures et les répertoires d'instructions sont donnés aux chapitres 4 et 5.

Après une définition et explication de toutes les instructions, nous avons déterminé les differents signaux électriques à générer, et donc, le cahier de charge des unités de contrôle. Nous avons ensuite, développé et optimisé la structure des deux unités de contrôle des deux PTDC.

Le sixième chapitre est consacré à la validation des circuits développés grâce à l'outil CAO PALASM *"Advanced Micro devices"*. Nous avons d'abord effectué un pré-traitement de simplification du nombre élevé de conditions de contrôle que doit gérer l'unité de contrôle "UC" d'une part, parce que le PALASM ne permet pas le traitement simultané et multiple d'un nombre élevé, et d'autre part parce que nous avons imposé un certain nombre de partitionnements en groupes fonctionnels pour mieux adapter et prendre en charge les exigences de test.

Après développement des circuits à base de PAL et des programmes de validation des deux UC par PALASM (sixième chapitre), nous avons analysé quelques applications.

Dans ce même chapitre nous avons expérimenté la validation des circuits PRPG (*Pseudo Random Pattern Generator*) et MISA (*Multiple Input Shift Analyser*) en utilisant les PALs de type PAL22v8 comme action de vérification des programmes développés et pour présenter une certaine assurance quand à la réalisation complète du PTDC.

Ce mémoire se termine par une conclusion qui situe la valeur de ce travail et présente les perspectives.

Chapitre 1

TECHNIQUES DE TEST

1. TEST DES CIRCUITS

2. TEST DES CARTES ET SYSTEMES

3. TECHNIQUES D'OBSERVABILITE ET DE CONTROLABILITE

4. TECHNIQUES DE SCAN PATH « CHEMIN DE BALAYAGE »

1. TECHNIQUES DE TEST

1.1 INTRODUCTION

Le processus de test consiste à appliquer un ensemble de valeurs logiques [0,1] successives aux entrées primaires PI (*Primary Inputs*) d'un circuit, et observer les valeurs résultantes aux sorties primaires PO (*Primary Outputs*). Afin d'exprimer le résultat d'un tel test, il est nécessaire de connaître les valeurs de sorties qui devraient apparaître dans le bon circuit, celles-ci sont appelées « *Fault-Free Outputs* » [1,2,3,4,5]. Il existe deux classes de test, le test paramétrique qui s'occupe des mesures électriques (courant, tension, temps, ...etc) et le test logique qui s'intéresse uniquement à la fonctionnalité du circuit [2].

Le problème majeur de test est de trouver l'ensemble des valeurs de test adéquates pour chaque circuit. Ce processus est décrit comme étant la génération de test TPG (*Test Pattern Generation*). Il détermine si effectivement le circuit accomplit correctement sa fonction. L'ingénieur de test ne dispose, en général, que du schéma synoptique du circuit. Sa tâche est donc d'identifier les problèmes inhérents à la fonction de chaque partie du circuit, ainsi il se trouve face à de nombreux problèmes dont :

- Pour un circuit combinatoire, l'utilisation de la table de vérité paraît la solution la plus simple, mais celle ci entraîne le problème de sa construction pour des circuits larges d'une part et le temps de vérification qui peut devenir extrêmement important d'autre part [4].
- Pour un circuit séquentiel, le concept d'appliquer un test exhaustif (application de tous les vecteurs de test possibles) est très difficile. Cette difficulté est liée à la production d'états internes [4].
- Le test des microprocesseurs et les circuits micro-programmés imposent le test de chaque instruction du répertoire pour une multitude d'opérandes et différents séquencements d'instructions.

Le test fonctionnel est, alors difficile à implanter d'une manière systématique. L'approche structurelle est basée sur une liste pré-définie de fautes, dérivées d'un modèle de fautes. Dans le TPG structurel, l'attention est dirigée vers les défauts dans le circuit, on stimule une faute pour la faire manifester à la sortie, en la propageant sur un chemin sensible par application d'un vecteur à l'entrée. Celui ci étant trouvé, on cherche ensuite, par une simulation logique, toutes les autres fautes détectées par ce vecteur. Les principales causes des erreurs observées peuvent être des erreurs de conception, des erreurs de fabrication ou des défauts physiques [2].

On note que la "faute" est le résultat fonctionnel d'un défaut physique. Les fautes peuvent être regroupées en différentes catégories : collage à 0 (*Stuck-at-0*), collage à 1 (*stuck-at-1*), état intermédiaire, circuit ouvert (*Stuck-open*), court circuit, en pont (*Bridge*) et sensible à la configuration (*Pattern Sensitive*) [1,2,3,4,5,6].

La contrôlabilité, l'observabilité et la détection sont les trois facteurs importants qui déterminent la complexité de test d'un circuit [1].

1.2 TEST DES CIRCUITS

Les mécanismes qui permettent de détecter les défauts, moyennant leurs sensibilisations fonctionnelles ou les vérifications de comportement sous des conditions et contraintes forcées, sont utilisés pour le tri de circuits qualifiés « bons » (parfois les résultats sont comparés à ceux fournis par la simulation). Ces tests ne sont pas universels, car ils dépendent de l'application. Alors, il y a évidement intérêt à découvrir, le plus tôt possible, le composant défectueux avant son montage dans un système. Plus tard, le test est effectué sur le système global, plus il est complexe, délicat et coûteux. Les premiers tests appliqués à un produit, à la sortie de la chaîne de fabrication sont destinés à assurer la conformité aux spécifications de la filière technologique (résistivité, tension de seuil, ...etc). Un premier test fonctionnel est généralement appliqué avant l'implantation des produits sur une carte. La décision des opérations de test à chaque pièce est binaire, où les reconnues défectueuses seront marquées et rejetées avant le montage. La soudure dans le circuit et la connexion des pièces aux plots sont des opérations délicates qui doivent être suivies d'un test final [4].

En fin de production, la nature du test est critique car la décision définitive est binaire. Un examen séparé des circuits rejetés fournit des données utiles pour l'amélioration du rendement de fabrication, et donc du processus technologique. Les fautes peuvent être détectées après des tests particuliers appliqués, ou dès l'apparition d'erreurs de fonctionnalité du système. Lorsque le circuit est déclaré bon, on ne peut seulement dire que les réponses aux stimuli du circuit sont bonnes, c'est à dire que la fonction désirée peut être réalisée.

Un circuit, considéré bon après test, peut aussi subir des dommages ultérieurs (électrique ou mécanique). De même, le test des circuits au stade de fabrication est dans certain cas insuffisant (problèmes liés à l'économie), inadéquat ou non performant (connaissance). Un circuit supposé bon, une fois intégré dans un système, peut éventuellement causer un mauvais fonctionnement. L'identification de circuits défectueux individuellement en fin de production est plus facile et moins chère que leur identification une fois intégrés et implantés dans un système [6,7].

Le prix de test croit si le niveau d'abstraction est plus bas. En effet, il est plus pratique de vérifier qu'une porte logique se comporte conformément à sa table de vérité que de vérifier ses caractéristiques courant-tension [8].

Les composants programmables (microprocesseur, micro-contrôleur) sont parmi les CIs les plus complexes qui se trouvent dans la majorité des applications informatiques, industrielles, scientifiques et militaires. Pour ces circuits, on peut considérer le test des paramètres électriques et temporels comme étant un problème presque maîtrisé, mais les tests fonctionnels (vérification de la fonction logique réalisée par le circuit) exigent plus d'actions et de stratégies. En effet, de très grands nombres et diversité de fonctions que les circuits programmables actuels réalisent, rendent critique la tâche de génération de test complet [6,7]. Le test des circuits logiques est subdivisé en deux : test des circuits combinatoires et test des circuits séquentiels.

1.2.1 Test des circuits combinatoires

Le test des circuits combinatoires consiste à appliquer toutes les combinaisons d'entrées primaires (PI) donnant une sortie primaire (PO) ne dépendant que de ces entrées sous test. Les vecteurs de test sont générés à partir d'une table de vérité donnant tous les vecteurs de test possibles pour la détection de fautes [3,4]. La réduction du nombre de ces vecteurs de test se base sur les propriétés suivantes :

- Les défauts détectés par le même vecteur doivent être réduits en un seul défaut.
- Les vecteurs de test en commun à plusieurs défauts peuvent être réduits en un seul vecteur [4].

1.2.2 Test des circuits séquentiels

La réponse d'un circuit séquentiel dépend non seulement de ses entrées primaires mais aussi de son état interne. Alors, le test d'un tel circuit ne peut être spécifié à partir d'une simple table de vérité. On peut échapper aux nombreux problèmes comme :

- Il faut définir l'état présent du circuit (initialisation).
- Spécifier la réponse du circuit pour toutes les séquences d'entrées possibles.
- Tenir compte des temps de propagation, car l'information risque d'être déformée (aléas).

Un circuit séquentiel comporte généralement une ou plusieurs bascules. Pour son test, il est nécessaire d'effectuer les opérations suivantes :

1. Formation de la liste d'erreurs.
2. Initialisation.
3. Test des lignes asynchrones (indépendantes de l'horloge).
4. Utilisation de l'horloge pour valider les lignes à tester.

On remarque que les circuits combinatoires sont faciles à traiter par des systèmes automatiques, alors que les circuits séquentiels sont difficiles, à cause de leurs états qui peuvent êtres multiples et dont seulement certains sont utilisés [1,2,3,4,5].

1.3 TEST DES CARTES ET SYSTEMES

La carte est le milieu d'interconnexion des CIs le plus pratique pour la réalisation de systèmes électroniques. Une fois, les circuits bons identifiés et insérés sur une carte pour former un système, l'opération d'un test de fonctionnalité globale est considérée. Noter que le test des cartes même à la conception reste complexe. En effet, seulement quand les fonctions des cartes se prêtent à des vérifications indépendantes de la fonctionnalité globale qu'il y a possibilité d'écarter les cartes ne répondant pas aux exigences imposées. La stratégie de test des cartes n'étant pas universelle, des variétés se présentent suivant les objectifs tels que : la vérification de la conception, la détection de fautes, le diagnostic, ... etc [9]. Les catégories de test des cartes et systèmes sont multiples, les plus répandues sont le test au niveau circuit ICT (*In Circuit Test*) et test fonctionnel FT (*Functional Test*).

1.3.1 Test au niveau circuit

Le test dit ICT était conçu pour le test des composants tels que résistances, capacités, diodes et transistors décrits par des valeurs spécifiques de connexion, position et orientation sur les cartes (test analogique). ICT est développé ensuite pour le test des CIs complexes, et se base sur l'utilisation d'un « lit de clous » qui produit des accès aux pins d'entrée et de sortie d'un circuit, en mode normal ou en mode isolation. La complexité de la fonctionnalité, augmentant avec la densité et la taille des cartes, exige un accès à un nombre important de noeuds pour assurer un test complet. ICT est appelé à réaliser des tests fonctionnels de chaque composant plutôt qu'un simple contrôle d'une série de paramètres et une vérification de tables de vérité. ICT est utilisé au stade final du processus de test des cartes. Néanmoins, ICT n'offre pas de garantie complète sur la fonctionnalité globale du système, vu les problèmes de contact des clous avec les pins des circuits, le nombre important de clous, leur alignement, positionnement ainsi que la gestion des signaux recueillis pour l'établissement de critères de décision de test [2,4].

1.3.2 Test fonctionnel

La forme classique d'un test fonctionnel (FT) est la vérification de la fonction du circuit. Deux types de test fonctionnel sont possibles : on-line et off-line. Dans le cas du premier type, seulement des entrées et sorties d'un circuit bien particulières sont vérifiées. Les accès et sélections des points de test restent dans ce cas dépendants de l'application [1]. La vérification de la fonctionnalité et la décision de test sont généralement obtenues après comparaison des données de test avec celles d'un « *golden unit* » ou celles obtenues par simulation. Le problème de diagnostic est très délicat du fait que les décisions du test ne fournissent pas d'informations concernant les parties défectueuses. Dans le cas de test fonctionnel de type off-line, généralement, on y introduit des vecteurs de test à la vitesse du testeur et il y a gestion des réponses (individuelles, globales ou comprimées). Ce type de test peut conduire à un diagnostic d'une manière simple mais nécessite parfois une connaissance approfondie du système à tester [2].

Il est à noter, que la stratégie de test des cartes varie suivant plusieurs facteurs tels que technologie, industrie, complexité des circuits, coût, ... etc. Avec les nouvelles technologies, les différents tests cités ci dessus sont insuffisants et parfois inadéquats, alors de nouveaux mécanismes sont introduits.

1.4 PROBLEMES DE TEST DES CARTES

Parmi les problèmes rencontrés, lors du test des cartes, on peut citer :

1. **L'accès physique :** la facilité d'accès au circuit pour des fins de test est très liée à la disposition physique des différents composants sur la carte. Parmi les caractéristiques physiques qui compliquent le processus de test, on trouve :
   - La disposition très dense de composants rend l'accès via des probes de test presque impossible.
   - Les « cartes à multicouches » posent le problème de contact avec les couches internes.
   - La technologie de montage en surface (SMT) et les composants à broches montées sous des boîtiers produisent eux aussi, une difficulté supplémentaire d'accès et surtout d'assurance de contact.

2. **L'interfaçage électrique :** Le fait de poser la probe sur un noeud du circuit, sur le connecteur ou sur l'une des broches implique un chargement électrique, ce qui pourrait entraîner une limitation de sortance et donc affecter le fonctionnement normal du circuit sous les conditions de test.
3. **Structure du système :** Un système peut être partitionné en plusieurs unités montées sur des cartes séparées. Le test de chaque carte isolée du système admet souvent une fonction non identifiable, il est donc difficile de réaliser des opérations de diagnostic par approche pré-définie [1,3].

1.5 TECHNIQUES D'OBSERVABILITE ET DE CONTROLABILITE

A cause de l'importance du test dans l'industrie et sa contribution dans le coût total du produit, il est nécessaire de pouvoir évaluer la qualité du circuit en formulant une mesure de la testabilité. Cette dernière est définie comme le pourcentage du nombre de fautes masquées par rapport au nombre total de fautes. La testabilité d'un circuit dépend principalement de l'observabilité et de la contrôlabilité de ses noeuds internes.

Ainsi on définit ces deux paramètres comme suivent :

1. La contrôlabilité est l'aptitude à contrôler la défaillance au niveau d'un noeud à partir des entrées primaires du circuit.
2. L'observabilité est l'aptitude à propager la valeur d'un noeud défectueux sur une sortie primaire.

La mesure de testabilité commence, en général, par la mesure de la contrôlabilité et de l'observabilité d'un circuit, pour distinguer les structures qui sont plus ou moins difficiles à tester [1,4].

Il existe plusieurs catégories de circuits qui ont une testabilité assez faible, dont on cite : les circuits avec redondance, les circuits asynchrones et les circuits synchrones non structurés [1,3].

Il est important au testeur de pouvoir contrôler les éléments tels que les bascules et les compteurs pour des buts d'initialisation. Alors, les entrées asynchrones de ces composants doivent êtres accessibles au testeur et ne doivent êtres, en aucun cas, liées d'une manière compliquée durant leurs états inactifs. L'accès à ces éléments et aux autres noeuds internes du circuit est possible en utilisant des broches additionnelles, des connecteurs de bord, ou en intégrant des multiplexeurs ou des registres à décalage dans le circuit [1,3,4]. Certains points de contrôle et d'observation sont définis ci-dessous.

- Points de contrôle d'un circuit numérique :
  - Lignes d'horloge.
  - Lignes de sélection de données.
  - Lignes de commande et validation.
  - Lignes d'adresses, de données et de contrôle de bus système.
- Points d'observabilité (points de test) :
  - sorties de bascules.
  - Compteurs.

- Registres.
- Encodeur prioritaire.
- Noeuds redondants.
- Sorties de circuits de contrôle et de validation.
- Boucles de retour.

1.6 MECANISMES D'AMELIORATION D'OBSERVABILITE ET DE CONTROLABILITE

L'application de certaines règles de testabilité est montrée sur les circuits de la figure 1.1. Le problème de contrôlabilité et d'observabilité est corrigé par un ajout de portes logiques, de MUX ou autre circuiterie avec une broche additionnelle d'entrée-sortie pour le test. Les éléments pour lesquels il est nécessaire d'y intervenir pour améliorer la contrôlabilité et l'observabilité sont donnés ci-dessous.

1.6.1 Points de test

Il existe deux types de points de test, points de contrôle CP (*Control Points)* et points d'observation OP (*Observation Points).*

Dans la figure 1.1.b une nouvelle entrée de contrôle CP est ajoutée à la porte G pour former une nouvelle porte G*. Quand CP=0 alors G*=G et le circuit fonctionne en mode normale. Pour CP=1 le point G*=0, c-à-d on force la sortie de circuit à 0. Cette modification de la porte permet une injection de 0 au circuit 0 (0.I). Le nouveau point de test d'observation (OP) permet une observation directe de G*.

La figure 1.1.c montre le circuit d'injection d'un 0/1. G est modifié par l'addition d'un point de contrôle CP1 et une nouvelle porte G' est ajoutée au circuit. Quand CP1=CP2=0 alors G'=G, indique une opération normale. CP1=1, le signale G*=0 et G'=CP, donc G' est facilement contrôlé par 0 ou 1.

La figure 1.1.d montre l'utilisation d'un multiplexeur (MUX) comme un circuit d'injection de 0/1 quand la ligne CP2 est sélectionnée.

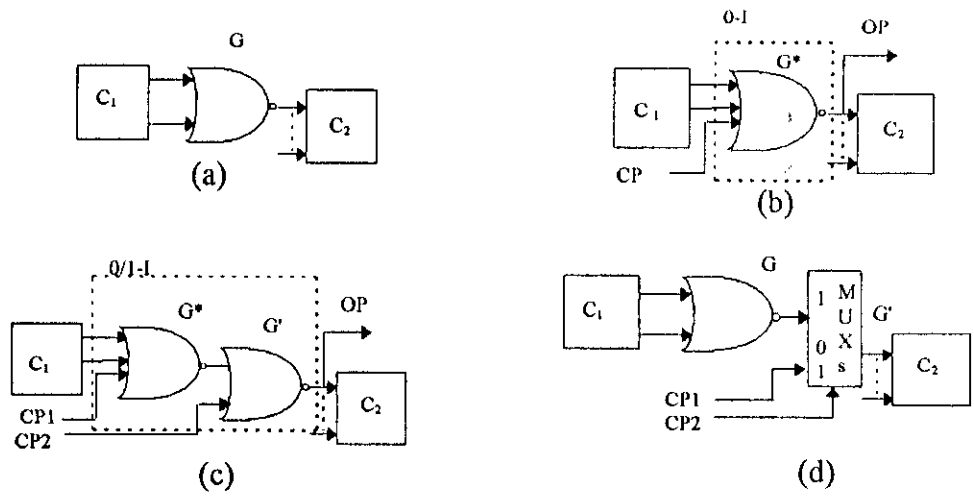


Figure 1.1 : Utilisation de points de test

En général, lorsque G est un signal arbitraire, si on insère une porte AND ou NOR dans cette ligne, on crée un circuit d'injection de 0, et si on insère une porte OR ou NAND on crée un circuit d'injection de 1 [1].

1.6.2 Redondance logique

Un noeud d'un circuit est dit logiquement redondant si toutes ses sorties sont indépendantes de la valeur logique attribuée à ce noeud. Un exemple de redondance logique est montré à la figure 1.2 ; elle est généralement utilisée pour éliminer le hasard statique.

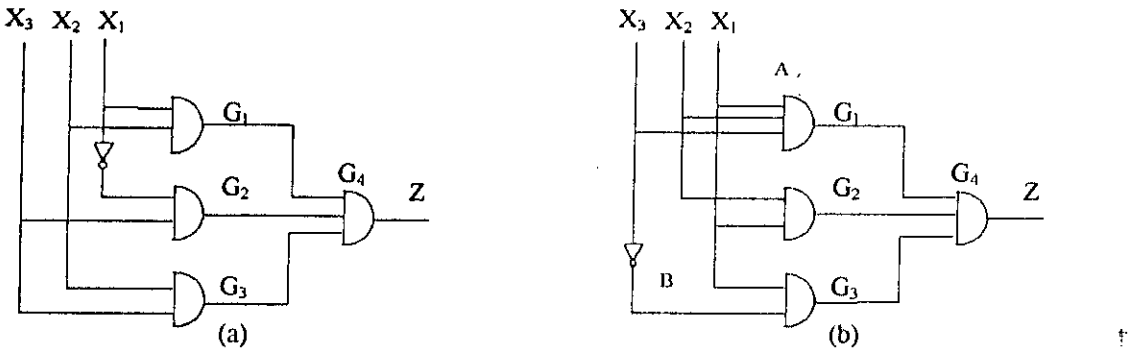


Figure 1.2 : Hasard statique et masquage d'erreurs

Dans l'exemple (a) de la figure 1.2, la porte G_3 est introduite pour éliminer le hasard statique lors de la transition du terme X_1X_2 à X_1X_3. Le collage à '1' de la sortie de G3 ne peut être propagé via la porte G4 pour être observé en Z. De même dans l'exemple (b), le collage à '1' du noeud A ne peut être décelé au niveau de Z à cause des valeurs incompatibles de X_1, X_2 et X_3. Ceci affecte par conséquent la détection du collage à '1' du noeud B.

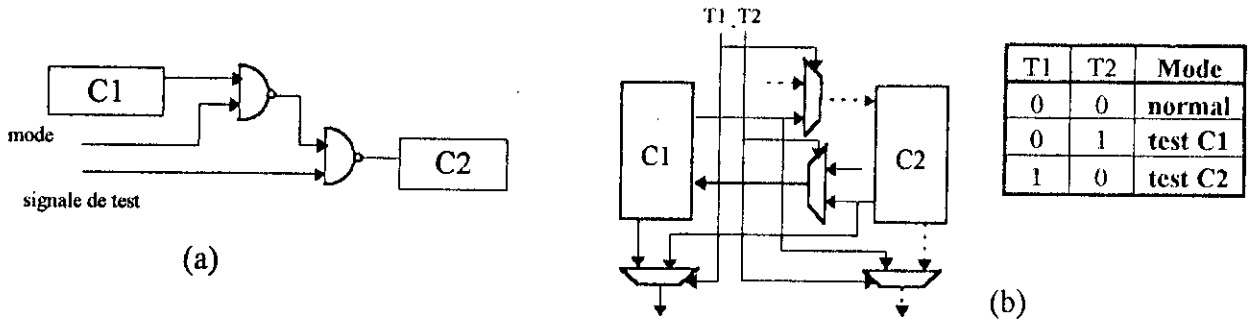
Cependant, une faute introduite au niveau d'un noeud redondant, ne peut être détectée et donne ainsi naissance à deux problèmes : le premier est que la faute peut réintroduire la condition du hasard statique pour laquelle elle est conçue à éliminer ; le second est que la faute peut masquer la détection de fautes d'autres noeuds non redondants [1,4,5].

Il est alors nécessaire de contrôler et d'observer les états par circuiterie supplémentaire.

1.6.3 Partitionnement de circuits complexes

Pour les circuits complexes et de grande taille, il n'est pas simple de réaliser un contrôle et une observation élevés de tous les points. Mais, si le circuit s'apprête à une subdivision (connaissance de fonctionnalité) alors, il est possible de le subdiviser en plusieurs sous-circuits facilement testables par ajout de logiques additionnelles (MUXs, registres, portes logiques, ...etc). La partition dépend généralement de la structure du circuit. Des exemples de subdivision sont illustrés sur la figure 1.3. Pour l'isolation et le test des différents modules issus de la partition, il suffit de valider la ligne de commande 'mode'. Pour l'exemple (a) de la figure 1.3, la première porte isole le circuit C2 lors du test et la seconde insère le signal test. Dans l'exemple (b), il suffit de valider un chemin adéquat pour le transfert des données de

test d'un circuit indépendamment de l'autre. Pour le test du circuit C2 on valide le chemin présenté en tirés. L'effort de test est proportionnel au cube du nombre de circuits existants [1,2,3,4,9].



T1	T2	Mode
0	0	normal
0	1	test C1
1	0	test C2

Figure 1.3 : Partition et isolation de modules

1.6.4 Initialisation

L'observation des éléments mémoires et leur contrôle nécessitent des actions au niveau de chaque bascule, car les réponses des bascules à un vecteur dépendent des entrées et des états présents des bascules. L'initialisation de ces éléments est nécessaire pour l'application d'un programme de test [1,3,4]. Ceci peut être assuré en agissant sur les lignes PR et CLR comme il est illustré sur la figure 1.4. Les lignes PR et CLR sont maintenues à '1' via une résistance commune de « *Pull-Up* ». Cependant, si le noeud X est mis à '0' la sortie sera indéterminée. La séparation de ces deux lignes de commande facilitera leur contrôle.

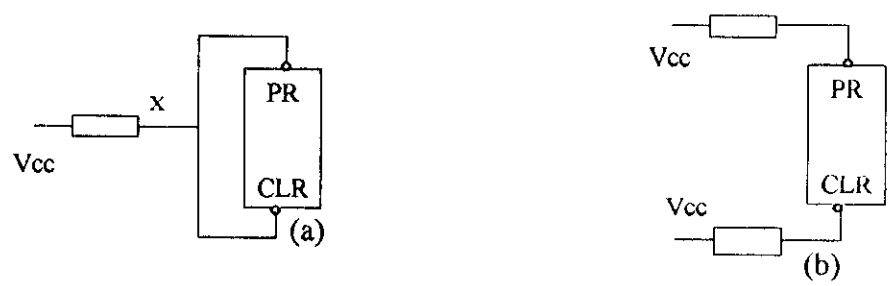


Figure 1.4 : Initialisation des éléments de mémoire

1.6.5 Chaîne de retour

Lorsqu'un circuit possède des contre-réactions (définition des états), l'application du test peut être inefficace, vu les données imposées par les informations de retour ainsi masquant les erreurs. Pour y remédier, plusieurs techniques ont été développées pour assurer le contrôle de ces contre-réactions, mais aucune ne semble satisfaisante. Certaines de ces approches sont illustrées sur la figure 1.5. La première configuration permet seulement l'isolation de la contre-réaction, alors que la seconde facilite, en plus de l'isolation, le contrôle via la deuxième porte [1,2,6].

Figure 1.5 : Contrôle de chaîne de retour

1.6.6 Test des ROM et des PLA

La mémoire est toujours validée par la mise à la masse de la ligne EN figure 1.6, le testeur est incapable de contrôler les lignes d'entrée ; il est donc conseillé de réduire la dépendance du programme de test pour prévoir des modifications à volonté. La validation de la ROM est assurée par la mise à "0" de la ligne de contrôle "EN" comme le montre la figure 1.6. Pour ce faire, on relie la ligne "EN" à la masse via une résistance de « *Pull-Down* » ainsi un accès externe est possible. Le second exemple dans cette figure, permet une grande immunité au bruit mais exige une circuiterie plus grande [1,11]. Le test de ces composants nécessite l'application bien déterminée de vecteurs pour assurer un test physique de chaque case et chaque noeud. Une recherche très approfondie pour le test de ce type de circuits est présentée en [11,12,13].

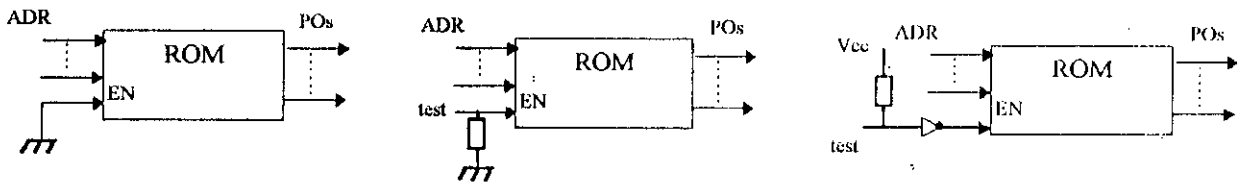


Figure 1.6 : Test des ROMs

1.6.7 Contrôle direct de l'horloge et oscillateur

Il est parfois nécessaire de prévoir un mécanisme de contrôle de l'horloge d'un système afin de pouvoir modifier la fréquence ou introduire une horloge externe. Ceci synchronise les opérations du testeur avec le système et donc facilite généralement le contrôle[1,4]. Un exemple de contrôle direct de l'horloge est illustré sur la figure 1.7.

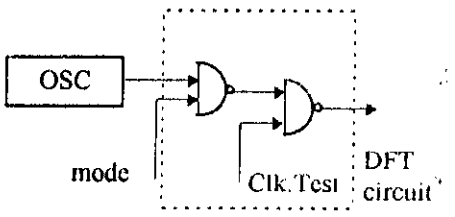


Figure 1.7 : Test logique d'un oscillateur

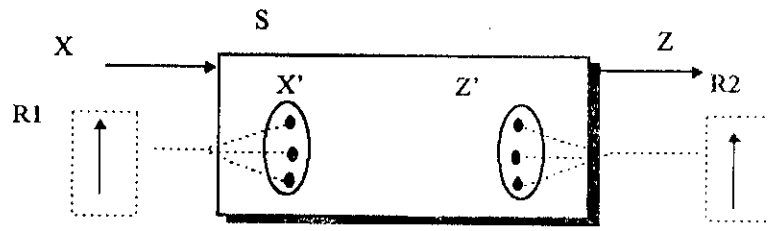


Figure 1.8 : Points de test "Scan"

1.8.1 Principe du SCAN PATH « S.P »

Cette technique permet de réduire la complexité du problème de génération de test pour les circuits séquentielles en assurant de meilleures contrôlabilités et observabilités. On introduit des bascules pouvant fonctionner en mode de chargement parallèle et en mode de décalage sériel [14,15,16,17].

1. Dans le mode de chargement parallèle, les différentes bascules sont chargées en parallèle lors du fonctionnement normal du circuit.
2. Dans le mode de décalage sériel (mode test), on charge les bascules par des valeurs de test désirées (contrôlabilité) et par un simple décalage on peut observer leur contenu en sortie (observabilité).

Pour ce faire, chaque bascule est précédée d'un MUX sous le contrôle d'un signal de sélection comme le montre la figure 1.9. Les bascules sont conçues de manière à fonctionner aussi bien en mode de chargement parallèle à travers Din qu'en mode de décalage sériel (formation d'un registre à décalage par les différentes bascules "FFs"), suivant l'état de la ligne de commande *Scan-select*. Ainsi le circuit peut être facilement contrôlable à travers la ligne d'entrée SDI (*Scan Data In*) et observable à travers la ligne de sortie SDO (*Scan Data Out*) [18,19].

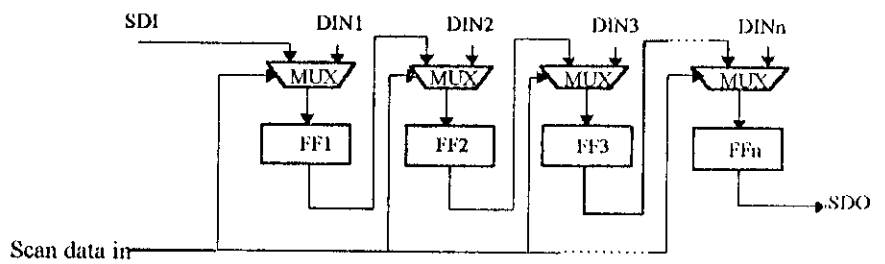


Figure 1.9 : Le schéma de principe du registre SP

Pour accéder au test par cette technique, il est nécessaire de parcourir les étapes suivantes :

1. Test de la chaîne de Scan-Path.
2. Application d'un premier vecteur de test via la chaîne de *San-Path* et des entrées externes du système.
3. Validation d'un cycle d'horloge normal du système pour produire le vecteur d'état futur et les sorties externes.
4. Décalage sériel du contenu du registre Scan vers la sortie.

Les étapes 3 et 4 sont répétées pour les vecteurs de test. La lecture du contenu du registre SP peut évidement être faite en même temps avec l'introduction du vecteur suivant. Les bascules dans la cellule

SP sont de type 'D' maître esclave pour assurer la synchronisation des signaux [3,17]. La figure 1.10 illustre la structure d'une telle cellule.

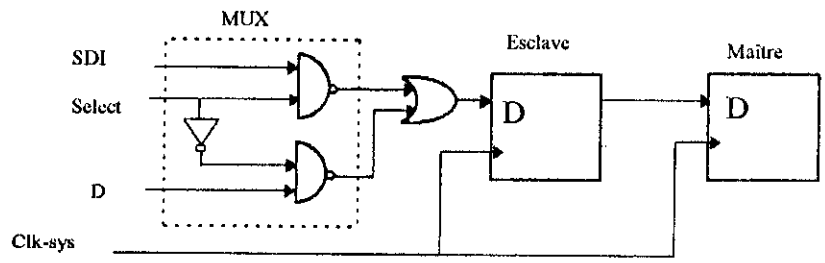


Figure 1.10 : Cellule de Scan Path

1.8.2 Principe du LSSD (*Level Sensitive Scan Design*)

Cette technique vient remédier aux faiblesses du scan Path de base. Dans cette technique les latchs sont utilisés par paires. Leur fonctionnement est contrôlé par des horloges multiples pour améliorer la contrôlabilité et l'observabilité du circuit. La figure 1.11 illustre la configuration d'une telle cellule :

En fonctionnement normal, Clk_C est activée afin de permettre le passage de la donnée Din, alors que Clk_A est désactivée.

En mode test, Clk_A et Clk_B sont activées, en effet A permet le chargement de la donnée SDI provenant de la sortie de la paire de FFs précédante dans L_1 et B permet le transfert du contenu de L_1 dans L_2. Cette configuration exige une circuiterie complexe et plus au moins importante [1,3,4,17].

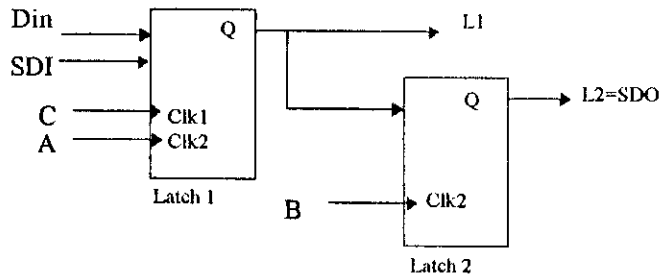


Figure 1.11 : Configuration d'une cellule LSSD

1.8.3 Prince du Scan Path partiel

Dans le principe du SP, le temps de test est entièrement consacré au chargement et au décalage de données dans la chaîne SP. En plus de ça, une grande circuiterie additionnelle est entraînée dans le cas de circuits complexes, ce qui peut ainsi réduire les performances du système.

Pour y remédier à ces problèmes, le registre SP est décomposé en sous registres, afin d'effectuer des opérations de test en parallèle. Ainsi on réalise une réduction du temps de test et de la circuiterie additionnelle. Les séquences de chargement et de décalage deviennent aussi plus petites. Les bascules à introduire dans la chaîne SP sont difficiles à contrôler et représentent une source d'information et non

seulement un point de réception de la donnée. Les bascules contrôlables et observables à partir des PIs et POs ou admettant des lignes de sélection, sont exclues de la chaîne SP [15].

1.9 BOUNDARY-SCAN « B.S »

Parmi les problèmes rencontrés lors du test des cartes comprenant plusieurs CIs on cite :

- L'intégration : en effet, le taux d'intégration étant très élevé alors, plus la disposition des composants devient très dense, plus l'accès via des probes aux points de test est presque impossible.
- L'interfaçage électrique par des probes de test provoque des effets indésirables sur le fonctionnement normal du circuit sous certaines conditions de test.
- Un problème de contact est aussi posé pour les cartes multicouches et la technique de montage en surface (SMT).

Le standard JTAG (*Joint Test Action Group, Boundary-S BS 1149.1*) a été mis au point pour faire face à ces problèmes. Il offre certains avantages évidents et consiste à réaliser un test des interconnexions des circuits, facilite l'isolation des circuits via le bus et réduit le nombre de connexions. Il utilise le mécanisme d'un contrôleur séquentiel pour l'entrée et la sortie de données [20,21].

L'objectif du standard IEEE-1149.1 est de résoudre les problèmes du test d'assemblage des composants sur les cartes et des modules, et ceci moyennant un nombre restreint d'accès (figure 1.12).

Quatre signaux suffisent : TDI est un signal d'entrée série, TDO est le signal de sortie série, TCK est l'horloge de test et TMS un signal de contrôle et de sélection. De plus, un signal d'initialisation TRST peut être présent. Le bus constitué par la connexion en série des composants *Boundary-Scan* de la carte comme un fil distribué sur le circuit imprimé permet la commande de signaux inaccessibles par les voies normales. Pour être compatible *Boundary-Scan*, un composant doit posséder des buffers fonctionnels (registres dédiés) autorisant deux types de fonctionnement qui sont dictés par le standard. Un mode « normal » permet au composant de fonctionner de manière transparente. Ce mode permet aussi d'échantillonner les signaux fonctionnels dans un registre à décalage, utilisé pour lire et écrire pendant le fonctionnement du composant. Un mode « test » assure un fonctionnement dédié au test de la « connectique » en utilisant le registre à décalage. Les détails concernant le BS sont donnés dans les références [1,4,5].

Le bus test et la logique associés opèrent comme suit:

1. Une instruction est envoyée seriellement via TDI au registre d'instruction.
2. La circuiterie de test sélectionnée est configurée pour répondre à l'instruction. Parfois ceci implique l'envoi de données supplémentaires à travers TDI à un registre sélectionné par l'instruction.
3. l'instruction de test exécutée, le résultat du test peut être acheminé vers la sortie du registre sélectionné et transmis via TDO au bus maître. Il est possible d'introduire de nouvelles données par TDI, quand TDO est en train d'envoyer le résultat via le bus.

1.9.1.3 SHIFT-DR

Dans cet état, les registres de données spécifiés par le contenu du registre d'instruction, et situés entre TDI et TDO, sont décalés d'une position. Une nouvelle valeur entre dans le chemin de test via TDI, et une nouvelle valeur est observée via TDO.

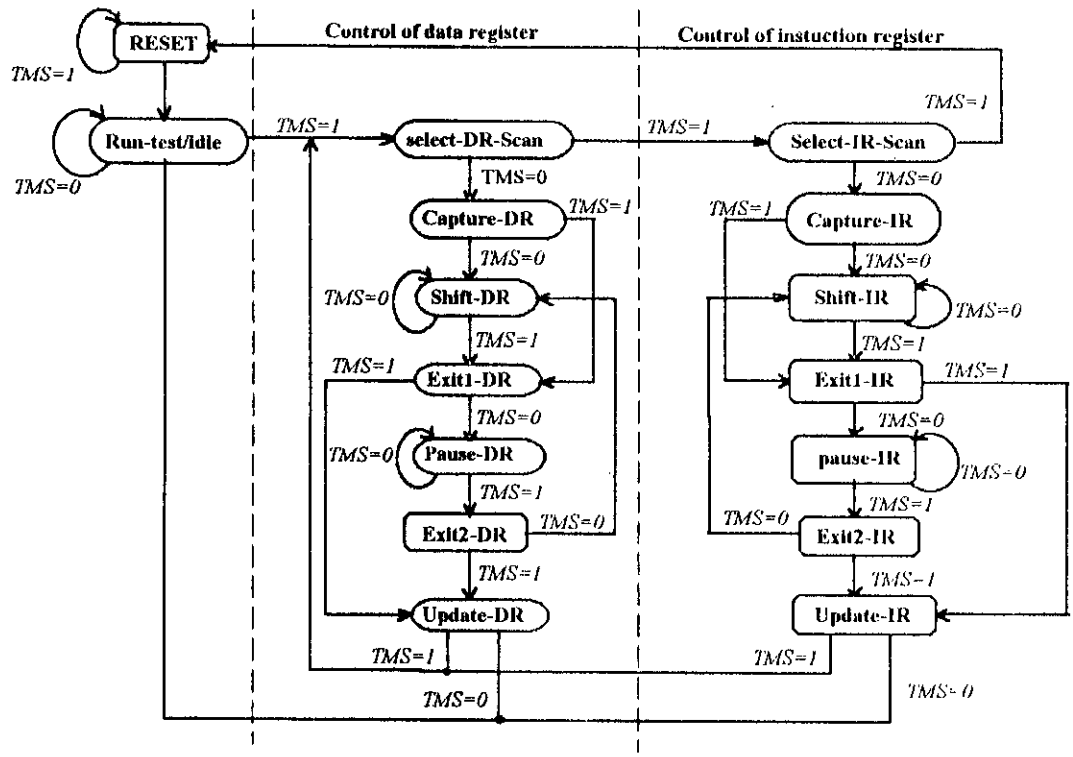


Figure 1.13 : Diagramme des transitions du TAP

1.9.1.4 UPDATE-DR

Quelque registres possèdent des sorties parallèles (latched) de telle manière que la sortie ne change pas de valeur durant le processus de test. Dans ce mode de contrôle, les registres de données ayant cette caractéristique, sont chargés dans les registres à décalage associés. Une fois le registre BS est chargé avec une donnée de test, le contrôleur du TAP est dans l'état *SHIFT-DR* ; ces données peuvent être appliquées ensuite aux entrées de la logique interne quand le contrôleur du TAP est dans l'état Update-DR par simple activation de l'horloge *Update-DR*.

1.9.1.5 CAPTURE-IR, SHIFT-IR et UPDATE-IR

sont analogues à celles du registre de données (*Capture-DR*, *Shift-DR* et *Update-DR*) et on note que TDO est activée seulement pendant les deux états *Shift-DR* et *Shift-IR*.

1.9.1.6 EXIT, PAUSE et SELECT

Durant les huit derniers états, la logique de test n'effectue aucune opération. Les fonctions *Pause-IR* et *Pause-DR* sont définies pour suspendre toute activité (arrêt temporaire) afin de valider une opération

plus urgente. Les six derniers états (*Select-DR, Select-IR, Exit1-IR, EXIT1-DR, Exit2-IR, Exit2-DR*) sont des décisions qui permettent la validation et le contrôle des opération de test. Le diagramme d'état de la figure 1.13 montre le séquencement des états du TAP [20,21,22,23].

1.9.2 Registre d'instruction et commandes

Ce registre a la possibilité d'introduire une nouvelle instruction par décalage pendant qu'il maintient l'instruction courante fixée à son port de sortie. Il est utilisé pour spécifier les opérations à exécuter et sélectionner le registre de données de test.

Chaque instruction permet un seul chemin sériel du registre de données de test entre TDI et TDO. Les instructions : *Extest, Bypass* et *Sample* sont demandées, alors que *Intest* ou *Runbist* sont très fortement recommandées.

1.9.2.1 Instruction BY-PASS

Chaque composant (module) a un registre *BY-PASS*, c'est un registre à une bascule. Si on considère une carte de 100 composants, lorsque on veut tester un seul composant, les données de test doivent transiter à travers tous les composants comme le montre la figure 1.14, donc pour réduire la longueur du chemin de test, il faut mettre les autres composants (les 99 composants) en mode BY-PASS (transparent) où une seule bascule relie TDI à TDO (Gain substantiel pour chargement et lecture des donnés de test).

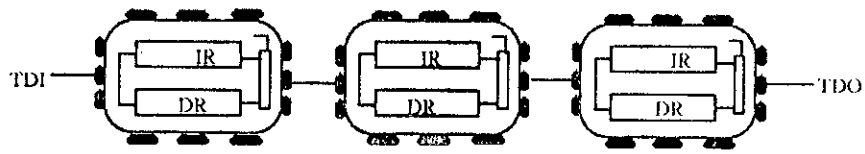


Figure 1.14 : L'acheminement du Bus de test entre les différents modules

1.9.2.2 Instruction EXTEST

Elle est utilisée pour tester les interconnexions entre les différents composants d'une carte. Lorsque cette instruction est exécutée, les cellules d'entrées du BS sont utilisées pour fournir les vecteurs de test, tandis que les cellules de sorties de BS servent à capturer les réponses.

1.9.2.3 Instruction SAMPLE

Dans ce cas on peut prélever des données de test pendant le fonctionnement normal du circuit, et les placer dans le registre BS pour un transfert ultérieur à travers TDO.

1.9.2.4 Instruction INTEST

Cette instruction est utilisée pour appliquer un vecteur de test à la logique interne via le chemin du BS et capturer la réponse correspondante.

1.9.2.5 Instruction RUNBIST

L'exécution du processus d'auto test se fait en appelant l'instruction *Runbist*, et en mettant le contrôleur du TAP à l'état *RUN-TEST/IDLE*. Cette instruction doit sélectionner le registre BS pour le connecter entre TDI et TDO.

1.9.2.6 Instruction IDCODE et USERCODE

Deux instructions optionnelles sont définies dans le TAP. La première permet de lire le code d'identification du composant en question alors que la deuxième sert à la lecture de son programme de test à exécuter. Ces deux codes existent dans le registre d'identification.

1.9.3 Les registres de données

Plusieurs registres de données sont intégrés dans ce standard tels que :

1.9.3.1 Registre BY-PASS

C'est un registre d'un seul bit qui permet de traverser un composant sans l'affecter mais ayant un intérêt de vitesse de test [20,21].

1.9.3.2 Registre BOUNDARY-SCAN

C'est une chaîne de cellules BS connectées seriellement, et qui représente un bus de transfert des données entre TDI et TDO. La figure 1.15 illustre le schéma de principe d'une cellule BS.

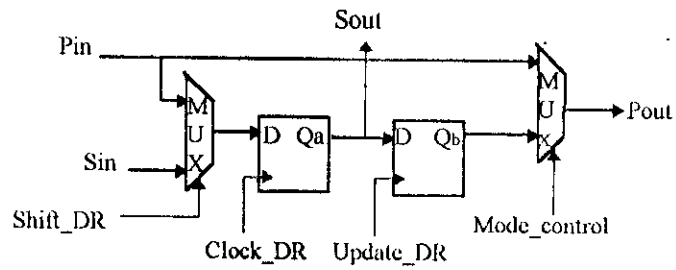


Figure 1.15 : Schéma de principe d'une cellule BS

Chaque cellule a quatre modes de fonctionnement :

1.9.3.2.1 Mode NORMAL

C'est un mode transparent, les données passent du pin Pin vers Pout lorsque la commande Control-Mode est désactivée.

1.9.3.2.2 Mode SCAN

Les cellules BS forment une chaîne de test où Sout de chaque cellule est connectée à Sin de la cellule adjacente. La première cellule est contrôlée par TDI et la dernière cellule contrôle TDO. Dans ce *cas Shift-DR* est activée, et l'horloge est appliquée à *Clock-DR*.

1.9.3.2.3 Mode CAPTURE

La donnée à l'entrée Pin est chargée dans le chemin de test en désactivant *Shift-DR* et en appliquant l'horloge à *Clock-DR*.

1.9.3.2.4 Mode UPDATE

Une fois la bascule A est chargée, soit en mode *Capture* soit en mode *Scan*, on peut faire sortir la valeur QA au port Pout en appliquant une impulsion d'horloge sur *Update-DR*, et en activant *Control-Mode*. Si la cellule *Boundary-Scan* est de type ligne d'entrée, la commande *Control-Mode* est appelée *Input-Control-Mode*, sinon on l'appelle, pour les cellules à broches de sortie, *Output-Control-Mode*.

1.9.3.3 Registre d'états

C'est un registre de test de type *Scan-Path* qui permet de contrôler la logique interne de circuit.

1.9.3.4 Registre d'identification

Un registre de 32 bits divisé en quatre champs comme suit : *header, manufacturer-code, part-number* et *version-number-code*. Ces 4 champs donnent des informations sur la position du composant en question dans la carte et de son programme de test à exécuter [20,21].

1.9.4 Modes de test au niveau circuit et carte

La figure 1.16 montre la stratégie de connexion inter-circuits dans une carte. En effet les différents composants forment une chaîne de test, dans laquelle TDI de chaque circuit est reliée à TDO du circuit voisin, excepté le premier et le dernier circuits qui eux forment TDI et TDO de la carte elle même [23,24]. Utilisant cette configuration, des variétés de test peuvent être appliquées :

- Test des connexions
- Observabilité et contrôlabilité améliorées.
- Test de chaque circuit (isolation).

Pour cela, il existe 3 modes de test qui sont:

1.9.4.1 Mode test externe

Ce mode est prévu pour tester la circuiterie d'interconnexion supportée par le BS. En sélectionnant les données du test d'une manière appropriée, on peut tester les courts circuits, les circuits ouverts ou le collage.

1.9.4.2 Mode de test échantillonné

La donnée d'E/S du circuit peut être échantillonnée durant le fonctionnement normal du système. La donnée échantillonnée peut être testée en dehors, alors que le système continue son fonctionnement normal.

1.9.4.3 Mode de test interne

Dans ce mode, les entrées de la logique interne sont contrôlées par les entrées de la cellule BS et la réponse peut être capturée à la sortie de la cellule. Le chemin de test interne et le BIST associé au circuit peuvent être activés pour tester le circuit. Cette configuration remplace quelques formes de test utilisant le lit de clous « *Bed-of-nails* ».

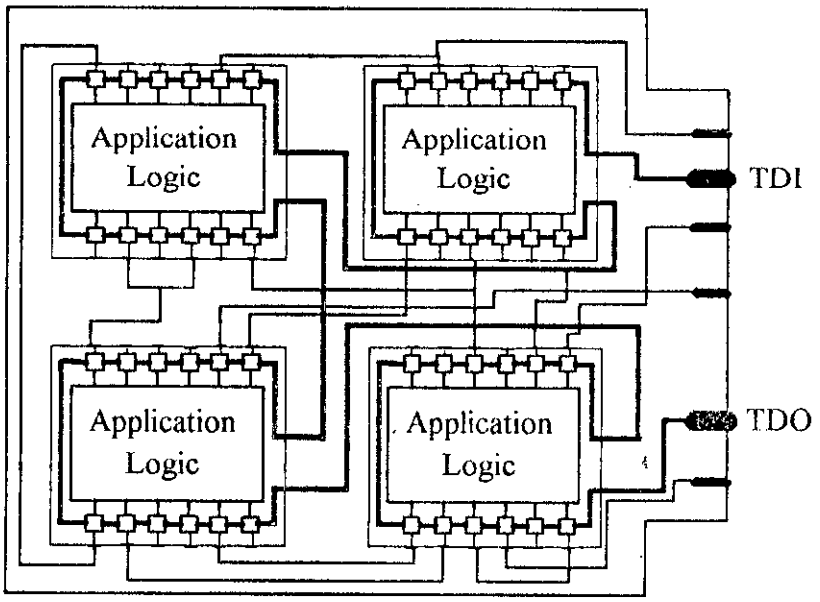


Figure 1.16 : Structure du BS1149.1 au niveau carte

1.10 CONCLUSION

La quantité de données à transférer à travers l'équipement de test est parfois importante. Dans les techniques de Scan Path, le temps de test est presque entièrement consommé pendant le chargement et la lecture sérielle des séquences de test et de leur réponse. S'il était possible de générer de façon interne les vecteurs de test, le gain serait appréciable. Le test interne se montre adéquat pour résoudre les problèmes de stockage (espace mémoire) et réduire le temps de test.

Il ne faut surtout pas imaginer que le BS 1149.1 soit une solution miracle répondant à toutes les questions posées par le test. La démarche qui consiste à tout baser sur cette technique, a un peu freiné son utilisation car quelques limitations proviennent des nouveaux ASIC et ASIP ne respectant pas les règles de testabilité et du fait qu'il n'est pas toujours facile d'adapter cette technique sans toucher à certaines performances qui peuvent devenir critiques [25,26].

TECHNIQUES D'AUTO TEST (BIST)

2. TECHNIQUES D'AUTO TEST (BIST)

2.1 INTRODUCTION

Le « BIST » (*Built-In Self Test*) désigne la technique qui permet à un circuit (carte, système) à s'auto-tester, il est défini comme une combinaison des mécanismes du test incorporé « BIT » (*Built-In Test*) et des procédures d'auto-test (*Self Test*) [1].

Le « BITE » (*Built-In Test Equipment*) se réfère au hardware et, ou software incorporés dans une unité pour assurer un test sans intervention externe [2].

Deux catégories de BIST sont définies On-line BIST (concurrent et non concurrent) et Off-line BIST (fonctionnel et structurel) comme le montre la figure 2.1 :

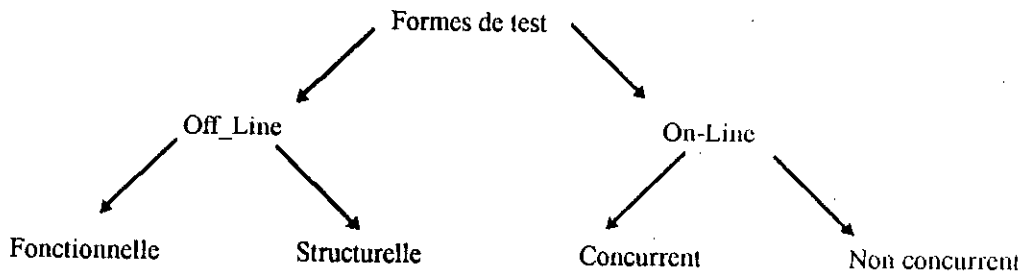


Figure 2.1 : Formes de test

- **ON-LINE BIST**

Le test est réalisé pendant le fonctionnement normal du circuit. Il est subdivisé à son tour en deux modes, le mode concurrent réalise le test simultanément avec le fonctionnement normal du système et le mode non concurrent se produit à l'état *IDLE* du système.

- **OFF-LINE BIST**

Le test est réalisé en dehors de l'état de fonctionnement normal du système. Il est basé sur deux approches : fonctionnelle et structurelle [1].

Les techniques d'auto-test s'appuyant essentiellement sur la production des vecteurs de test et sur l'analyse des réponses. Les éléments les plus importants qui affectent le coût du test sont la génération de vecteurs de test, l'application et la gestion des réponses. En général, des informations doivent accompagner chaque type de circuit pour réaliser son test. Le test compact est usuellement sélectionné pour réduire le volume de données à traiter grâce au mécanisme de production dites signatures. Le principal facteur de mérite de ces signatures obtenues par compression est la précision avec laquelle la présence d'erreurs est déclarée [1,4].

Le test compact comprend généralement trois étapes :

- Génération de vecteurs de test.
- Compression de données en signatures.
- Analyse et diagnostic des données.

Quand ces trois opérations sont adaptées et introduites au sein d'un CI, on obtient alors une structure de type BIT (*Built In Test*) et si le résultat du test est aussi interne on dira que c'est un "auto-test".

2.2 GENERATION DE VECTEURS DE TEST

Les techniques de productions des vecteurs à appliquer au CUT (*Circuit under Test*) sont dites générations de vecteurs de test TPG (*Test Pattern Generation*). Cette dernière est réalisable à base d'un registre à décalage doté d'une boucle de retour qui assure la rétro-réaction : LFSR (*Linear Feedback Shift Register*) [28]. La figure 2.2 montre une génération de vecteurs (S_i) à base d'un LFSR à trois bascules à état initial (011) et une porte ou-ex assurant le signal de retour. Ce PRPG permet une génération des vecteurs S_0, ..., S_6 périodiquement avec une période de 7 (2^3-1) [28].

S_0	0	1	1
S_1	0	0	1
S_2	1	0	0
S_3	0	1	0
S_4	1	0	1
S_5	1	1	0
S_6	1	1	1
S_7	0	1	1

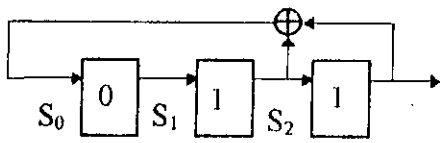


Figure 2.2 : Exemple d'une TPG

Lors de la génération des stimuli, un grand nombre de compromis ont été consentis sur le modèle de fautes, sur le nombre de fautes envisagées et sur l'étendue de la couverture de fautes. Il est alors raisonnable de s'interroger sur la couverture que réaliserait un ensemble de vecteurs de test générés d'une manière aléatoire. Une même performance de couverture exigera probablement un nombre de stimuli supérieur au nombre généré par un algorithme déterministe [1,29]. Plusieurs formes de générations sont résumées ci-dessous :

- Test exhaustif.
  - Génération de vecteur de test exhaustive.
- Test pseudo-aléatoire.
  - Génération pondérée.
  - Génération adaptative.
- Test pseudo-exhaustif
  - Combinaison d'un LFSR et registre à décalage.
  - Combinaison d'un LFSR et porte XOR (CFSR).
  - LFSR condenser.
  - LFSR cyclique.
  - Compteur de syndrome.
  - Génération multiple.

2.2.1 Génération exhaustive

La production des 2^n combinaisons pour un circuit à n entrées est qualifiée de test exhaustif. Un compteur binaire modulo 2^n peut être utilisé comme générateur de ces vecteurs. Un LFSR autonome a longueur maximale est utilisé avec la rétroaction réalisée grâce à un polynôme caractéristique primitif, son circuit peut être aussi modifié pour inclure l'état nul. Ce générateur est inutilisable si n est très élevé à cause du temps de test très élevé qu'il exige [1,4].

2.2.2 Génération pseudo-aléatoire

Les vecteurs générés ont beaucoup des caractéristiques des vecteurs aléatoires, mais sont générés d'une manière déterministe et donc à répétitions. Il est à noter que les 2^n vecteurs ne sont pas tout nécessaires.

Les vecteurs pseudo-aléatoires (VPA) sont générés avec ou sans remplacement, c-à-d le vecteur de test peut être généré plus d'une fois pour la génération avec remplacement et d'une manière unique pour le mode sans remplacement, d'où on peut employer pour leur génération un LFSR autonome.

Le test pseudo-aléatoire est applicable aux circuits combinatoires et séquentiels. Il évite effectivement de longs et complexes algorithmes de procédures de génération de vecteurs de test et c'est ainsi qu'il trouve son utilisation dans les systèmes à BIST. Sa longueur est choisie selon le critère de couverture de fautes [1,4,30,31,32]. Parmi les méthodes de génération on citera deux:

2.2.2.1 Génération pondérée

Le générateur de test pondéré WPG *(Weighted Test Generator)* est un générateur qui produit des vecteurs ayant une distribution des "0" et des "1", non uniforme ; Pour réaliser cette génération on assemble un LFSR autonome et un circuit combinatoire de pondération [33,34,35,36].

2.2.2.2 Génération adaptative

La génération adaptative utilise un "TPG" avec une modification du poids basée sur le résultat de simulation de fautes. Il en résulte donc une ou plusieurs distributions de probabilités pour les vecteurs de test. Une fois ces distributions sont déterminées, un générateur "TPG" approprié peut être conçu. Ce générateur est efficace en terme de longueur de test, mais sa réalisation est très complexe [1].

2.2.3 Génération pseudo-exhaustive

Cette méthode de génération dérive de la génération exhaustive, mais exige beaucoup moins de vecteurs de test. Elle est basée sur la segmentation du circuit et sur le test exhaustif de chaque segment. Il est à noter que les différents segments ne sont pas nécessairement disjoints [1,2,4]. Plusieurs formes de segmentation sont possibles :

2.2.3.1 Segmentation logique

Elle est subdivisée en deux catégories :

- **Segmentation en cône**

Le circuit à m sorties est segmenté logiquement en m cônes. Chaque cône est constitué de toute la logique associée à une sortie. Il est testé exhaustivement si tous les cônes sont testés en mode concurrent. Cette forme de test décrite par McCluskey [1984] est appelée : test de vérification. Si on considère un circuit combinatoire C de X={x_1, x_2,..,x_n}entrées et de sorties Y={y_1, y_2,...y_m}. Soit $y_i=F(X_i)$, ou $X_i \subseteq X$ et soit $w=\max_i\{|X_i|\}$. Une forme de test de vérification produit 2^w vecteurs sur tous les C_w^n sous ensembles de w entrées à C. Le circuit sous test est noté (n, w) CUT, où w<n. Si w=n, alors le test pseudo-exhaustif devient simplement exhaustif [37].

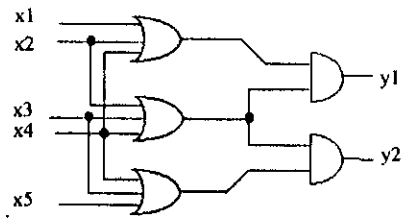


Figure 2.3 : (5, 4) CUT

- **Segmentation par sensibilisation de chemin**

Lorsqu'un circuit est réparti en segments suivant le concept de sensibilisation par chemin [1] comme le montre la figure 2.4 ; on peut tester C_1 exhaustivement par l'application de 2^{n_1} combinaisons au CUT de telle manière que E et F soient forcés à 0 pour ne pas affecter la porte OU et laisser passer D vers G directement (c'est le chemin sensibilisé). Le même procédé est adopté pour C_2 et C_3, d'où on a besoin de $2^{n_1}+2^{n_2}+2^{n_3}-2$ vecteurs au lieu de $2^{n_1+n_2+n_3}$.

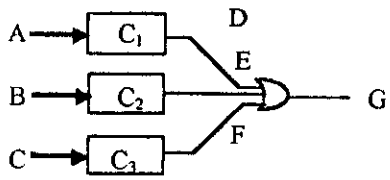


Figure 2.4 : Exemple de sensibilisation de chemin

- **Segmentation physique**

Pour les circuits très larges, les techniques décrites pour le test pseudo-exhaustif conduisent à des ensembles de test très larges. Dans ce cas, le test pseudo-exhaustif peut être utilisé en employant le concept de segmentation physique, c-à-d, en subdivisant le circuit à l'aide des techniques de segmentation hardware [1,37].

2.2.3.2 La Génération complète (CFSR)

Dans cette technique, la modification hardware sur le LFSR est basée sur l'addition de portes logiques supplémentaires permettant la génération de l'état nul, ainsi le LFSR standard devient un CFSR (*Complete Feed-back Shift Register*) de période 2^n. La figure 2.5 illustre un CFSR à quatre étages de

période maximale 2^4. L'insertion de l'état nul est faite par une porte XOR additionnelle, insérée à la sortie de la dernière bascule du CFSR, et la détection de cet état est réalisée par la porte logique NOR. Plusieurs formes de génération modifiant le LFSR sont présentées dans [1] et [38].

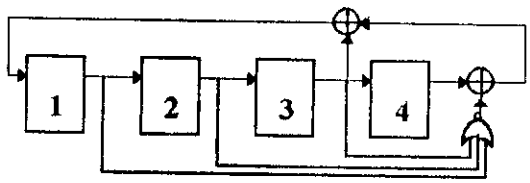


Figure 2.5 : Structure d'un CFSR

2.2.3.3 Génération dynamique

La génération de vecteurs de test se fait à base du LFSR, cependant dans le cas général le nombre de sorties m diffèrent de celui d'entrées n du CUT. Lorsque m<n et les connexions sont fixes, on dit qu'on a un UPG (*Unmatched Pattern Generation*) à connexions statiques. Un UPG exige une circuiterie importante pour sa conception réduisant ainsi la qualité de test [39].

L'UPG à connexions dynamiques est conçu à partir d'un LFSR et d'un circuit d'aiguillage inséré entre ce dernier et le CUT. Il nécessite m bascules pour construire le générateur primaire PG-LFSR (*Primary Generator LFSR*) et exige n/m bascules pour concevoir le circuit de contrôle du générateur CN-LFSR (*Control LFSR*) qui réalise les différentes permutations. L'organisation générale du générateur est illustrée sur la figure 2.6.

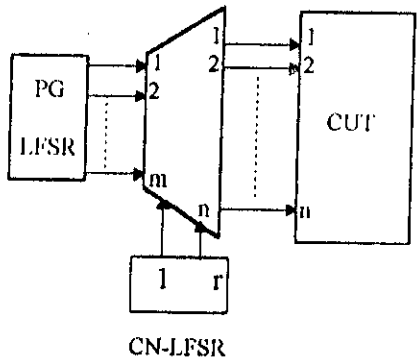


Figure 2.6 : Structure d'un UPG à connexions dynamiques

L'UPG est conçu de telle manière que chaque sélection à m cellules du PG-LFSR forme un registre à décalage ; cette opération est contrôlée par une sortie du CN-LFSR. Cette génération est très efficace, améliore la qualité de test, exige une circuiterie additionnelle acceptable et permet une bonne couverture de fautes [39].

2.2.3.4 Génération multiple

Dans certaines applications pratiques du BIST, les vecteurs de test générés par un LFSR durant un cycle pour un état initial donné, peuvent ne pas couvrir toutes les fautes simulées, et dans d'autres cas, une partie du cycle parait suffisante pour garantir une bonne génération de test. Il est ainsi avantageux de perturber le cycle de génération par la modification de l'état initial du LFSR durant l'opération de

génération. Pour réaliser cette technique on utilise des cellules LSSD comme le montre la figure 2.7, cependant il y a ralentissement du processus de test.

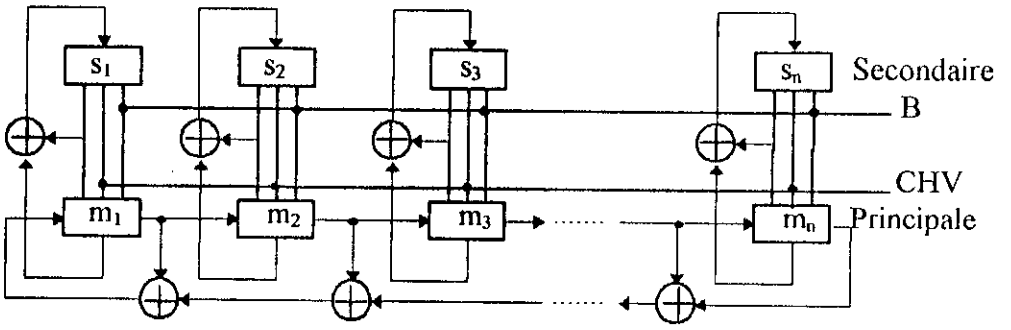


Figure 2.7 : Structure d'un MULTIPLE SEED LFSR.

Le chargement d'un nouvel état est assuré par une paire d'horloge B et CHV, et en imposant le polynôme $(1+x+x^2)$ à chaque étage [40].

2.3 COMPRESSION DE DONNEES (*SIGNATURE ANALYSIS*)

La comparaison de la réponse du circuit sous test avec la réponse de référence est la méthode conventionnelle qui permet de détecter les fautes, mais en pratique, cette méthode est souvent très chère à cause du nombre élevé de données à stocker [1].

La compression de la réponse permet de diminuer considérablement les données à mémoriser, d'où sa large utilisation [41,42]. Parmi les techniques de compression on citera : comptage des '1', [43], le syndrome [44], comptage de transitions [45], le contrôle de parité [46], GTSA *(Group Theoretic Signature Analysis)* [47], compression quadratique [48,49]et la division polynomiale (analyse de signature) [50].

L'analyse de signature a une très large utilisation dans le domaine de la compression vu son taux de couverture de fautes assez élevé comparativement aux autres méthodes de compression [1].

2.3.1 La division polynomiale

Cette technique permet la division de la séquence de données E(x) par le polynôme caractéristique P(x). Q(x) le quotient ou R(x) le reste de la division est considéré comme étant la signature. L'équation résultante est :

$$\mathbf{E(x)= Q(x) \times P(x) + R(x)} \qquad (2.1)$$

Dans cette technique on a utilisé le LFSR comme un élément de base pour la compression. Le LFSR divise le polynôme E(x) par le polynôme caractéristique P(x), le quotient apparaît seriellement à sa sortie et son contenu final constitue le reste appelé « signature ». La probabilité de masquage [50] pour cette technique est donnée par l'équation :

$$\mathbf{P = \frac{2^{k-1}-1}{2^{k+r}-1} \rightarrow 2^{-r}} \qquad (2.2)$$

k : représente la longueur de la séquence.

r : représente la longueur du registre LFSR.

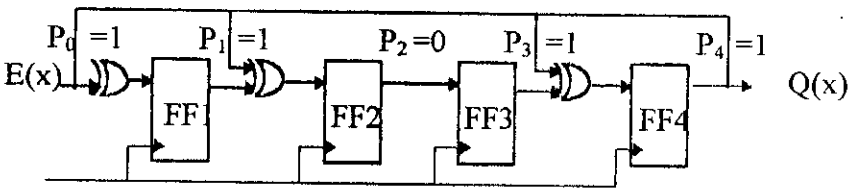
Il y a deux formes de LFSR qui sont : à une seule entrée et à entrées multiples.

2.3.1.1 LFSR à une seule entrée

Les bascules du LFSR sont connectées suivant le polynôme caractéristique [51,52,53]. Ce dernier a deux structures :

- **Structure interne**

Les portes XORs sont placées entre les bascules d'où le nom de la structure comme le montre la figure 2.8.

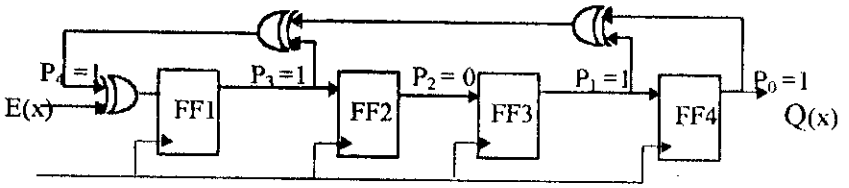


$P(x)=1+x+x^3+x^4$

Figure 2.8 : Exemple d'un LFSR à structure interne

- **Structure externe**

Les portes XORs sont placées extérieurement et l'ordre du polynôme caractéristique est en sens inverse avec celle de la structure interne comme le montre la figure 2.9:

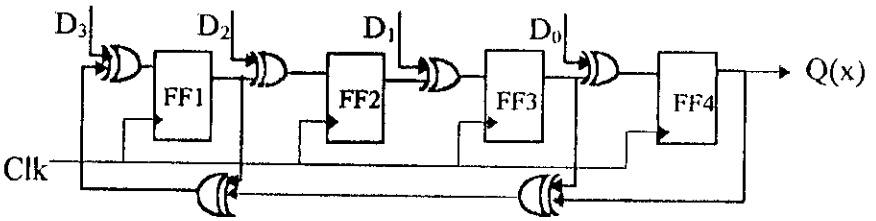


$P(x)= 1+x+x^3+x^4$

Figure 2.9 : Exemple d'un LFSR à structure externe

- **LFSR à entrées multiples (MISR)**

L'utilisation des LFSRs à une entrée simple est limitée aux circuits à sortie unique. Pour des circuits à plusieurs sorties, on peut utiliser un LFSR à multiples entrées MISR (*Multiple Input Signature Register*) [54,55,56,57]. La forme générale du MISR est montrée sur la figure 2.10.



$P(x)= 1+x+x^3+x^4$

Figure 2.10 : Exemple d'un MISR

Parmi les avantages du MISR on cite :

- Obtention d'une signature unique pour n sorties du circuit sous test au lieu de n signatures de la structure LFSR.
- Une seule opération de compression et une seule signature à mémoriser au lieu de n opérations et n signatures.
- Réduction du temps de test.

L'inconvénient principal du MISR est le taux de masquage élevé qu'il peut produire.

2.4 ARCHITECTURES SPECIFIQUES DU BIST

Dans cette partie on présente la description de quelques architectures de BIST développées par plusieurs groupes de recherche.

2.4.1 Architecture du BIST par module séparé et centralisé (CSBL)

Cette architecture CSBL *(Centralised And Separate Board-Level)*, proposée par Benwitzest [1], comme l'illustre la figure 2.11, n'utilise pas le mécanisme Scan, elle est utilisable pour les circuits combinatoires (C) et circuits séquentiels (S).

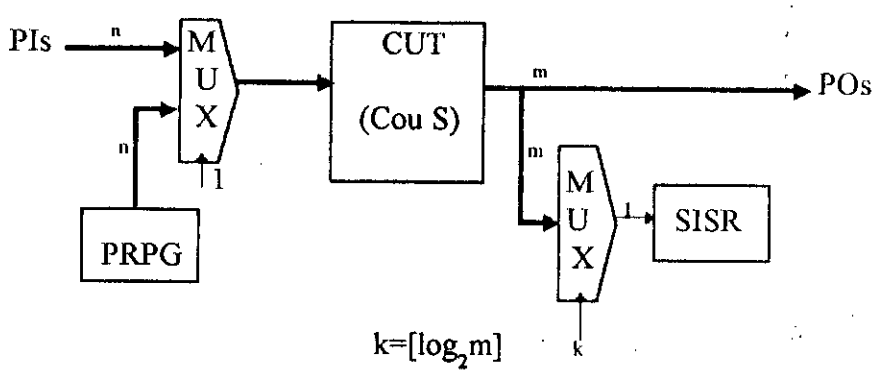


Figure 2.11 : Architecture du BIST centralisé et séparé.

Durant le mode de test OFF-LINE, les entrées sont dirigées par le générateur PRPG et les sorties sont contrôlées par l'analyseur de signature SISR *(Single Input Signature Register)*. Pour réduire les coûts du hardware, le test est répété m temps (1 pour chaque sortie) d'ou le besoin d'un seul analyseur de signature. Cette méthode est la plus commode pour les circuits "*Pipelines*" avec des lignes de rétroaction limitées [1].

2.4.2 Structure d'évaluation et d'auto-test (BEST)

Cette architecture BEST *(Built-In Evaluation And Self-Test)* est une application du CSLB. En général, elle est utilisée pour les circuits séquentiels. Les entrées du CUT sont pilotées par le PRPG et les sorties sont comprimées dans le MISR comme le montre la figure 2.12.

Les détails concernant l'application des entrées primaires et des sorties du PRPG quand on applique au CUT les valeurs normales ou de test ne sont pas définis. On peut utiliser soit un MUX, soit on charge le

PRPG par les entrées primaires et ensuit on les applique au CUT. Ce même concept est appliqué aux sorties. Le coût hardware du BEST est faible, cependant on a besoin d'une simulation extensive de fautes pour achever un équilibre acceptable entre la couverture de fautes et le temps de test.

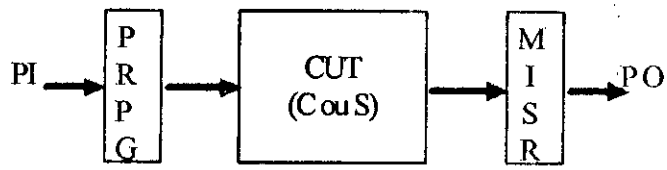


Figure 2.12 : La structure d'évaluation et d'auto-test

Cette architecture a une faible couverture de fautes pour les circuits séquentiels car elle repose sur les vecteurs pseudo-aléatoires. Pour contourner ce problème, on utilise un registre "Scan Path" au sein du circuit pour rendre son format combinatoire [1,2,9].

2.4.3 Test aléatoire à "prise" (RTS)

Cette technique n'est pas une véritable architecture BIST car la circuiterie de test est externe. Elle a les attributs suivants : le test hardware est séparé et distribué, utilise CUT avec SP (LSSD) mais n'utilise pas le BS comme le montre la figure 2.13.

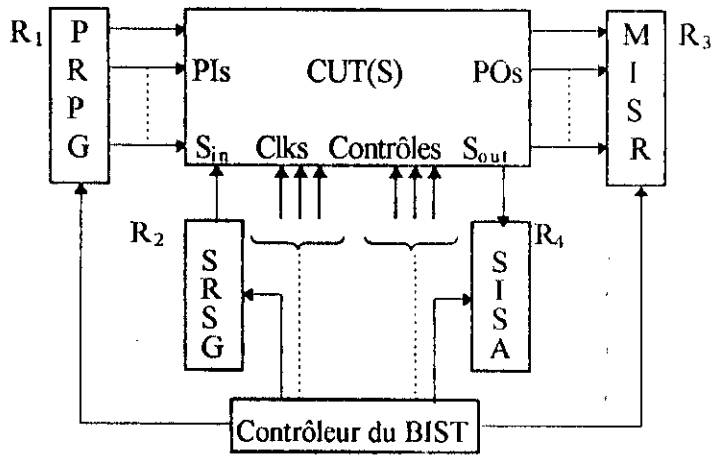


Figure 2.13 : Test aléatoire à prise.

Le CUT est testé selon les étapes suivantes :

1. Initialiser le PRPG.
2. Charger une valeur pseudo-aléatoire dans R_2 en utilisant le chemin scan.
3. Générer un nouveau VPA (Vecteur Pseudo-Aléatoire) avec R_1.
4. Appliquer un top d'horloge sur R_3 pour capter la réponse du CUT.
5. Exécuter le chargement parallèle des cellules du système d'enregistrement pour capter la réponse du vecteur de test aléatoire.
6. Décaler la donnée du registre interne SP du CUT vers l'extérieur et la comprimer dans le SISA *(Single-Input Signature Analyser)* (R_4).

SP. Comme le montre la figure 2.15, le PRPG dirige les registres SP en parallèle et la signature est générée par le MISR à partir de chaque registre de SP, aussi en parallèle.

L'utilité des registres multiples SP est de réduire le temps de test ; pourtant ils exigent que le PRPG s'exécute pendant k cycles d'horloges, où k est la longueur de registre SP le plus long, d'ou pour les petits registres, certaines données générées par le PRPG passent au MISR et donc peuvent affecter la performance de cette architecture [58].

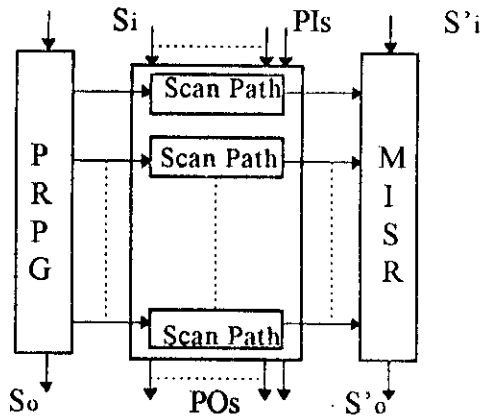


Figure 2.15 : L'auto-test par le MISR et le SRSG parallèle

2.4.6 Architecture du BIST concurrent

Les chercheurs Saluja et al [60] ont étendu le concept du OFF-line BIST pour inclure le ON-line BIST. Une version de leur proposition est l'architecture du BIST concurrent (CBIST) comme le montre la figure 2.16.

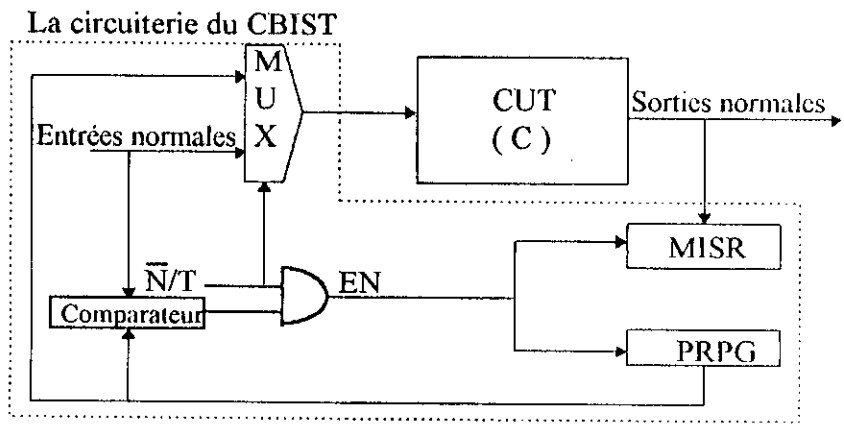


Figure 2.16 : Le BIST concurrent pour les circuits combinatoires.

Cette technique est employée pour les circuits combinatoires et aussi pour les circuits séquentiels qui admettent un partitionnement possible de leur circuiterie en blocs combinatoires testables séparément. Elle n'utilise pas les registres SP et BS ; le hardware du BIST est séparé du hardware du circuit normal. Pour réduire le coût du hardware on utilise l'architecture du BIST centralisé. En mode de test OFF-LINE BIST, le PRPG pilote le circuit et le MISR comprime les réponses. Cependant en mode de test ON-LINE BIST, le circuit est en fonctionnement normal, le PRPG et le MISR sont initialisés et laissés dans leurs états jusqu'à la validation du signal EN. Lorsque l'état du circuit et l'état initial du PRPG sont identiques, le signal EN est validé ; le PRPG changera d'état et le MISR échantillonnera les sorties

Le test s'opère comme suit : R_1, R_2 et R_3 sont mis en mode scan pour être chargés et puis les mettre dans le mode test durant la phase test du circuit. Dans ce mode, R_1 et R_2 génèrent un nouveau vecteur de test et R_2 et R_3 opèrent comme un MISR (où les vecteurs générés par R_2 constituent son état initial). La simulation de fautes est recommandée dans cette technique pour déterminer le nombre de cycles d'horloge d'exécution du mode test afin d'atteindre une couverture de fautes désirée. Cependant le problème est la génération multiple de certains vecteurs alors que d'autres sont ignorés [1].

La plupart des approches précédentes utilisent certaines formes de LFSR pour générer les vecteurs de test et la signature, alors que la RTD assume que la donnée soit normale dans le circuit si elle est combinée avec certaines formes de mécanismes de capture d'erreurs pour tester le circuit. Ce concept est la base des trois architectures de BIST suivantes [1].

2.4.8 L'auto test simultané (SST)

Cette approche utilise l'architecture du BIST distribué et encastré en plus de l'architecture du CUT avec SP, mais n'utilise pas le BS et les LFSRs comme le montre la figure 2.19.

Dans cette technique, les cellules d'enregistrement sont modifiées pour qu'elles soient auto-testables. Ces cellules ont trois modes d'opération : mode test, mode scan et le mode normal.

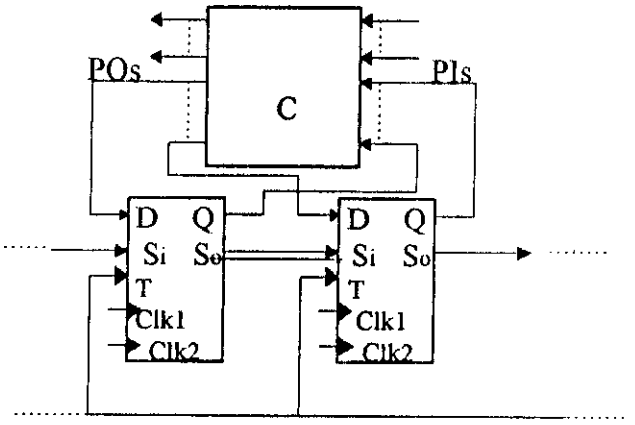


Figure 2.19 : L'architecture de BIST utilisant le SST

Le processus de test est de charger le registre SP par la donnée, positionner le circuit en mode test et valider l'horloge pour un temps suffisant afin d'obtenir un test convenable et ensuite placer le circuit en mode scan pour décaler le contenu du SP vers l'extérieur. Lorsqu'une erreur se produit dans une sortie durant le processus de test ou le contenu final du registre SP est incorrect, l'opération défectueuse est détectée. En mode test, l'auto-test du registre SP collectionne les résultats de test du circuit C et les vecteurs de test générés pour C simultanément. Cette procédure conduit à un test très rapide puisqu'elle ne demande pas de décalage du registre SP. Malheureusement, plusieurs problèmes sont associés à cette approche. Un de ces problèmes se manifeste lors du test d'une logique externe, c'est la production du test et du champ de test. Pour éliminer ce problème de production de test, On peut utiliser un ATE. Cependant pour y remédier au problème du champ de test, On utilise certains moyens tels que le BS, la connexion des sorties du CUT à ses entrées durant l'auto-test. Mais en général, les CUT ont un nombre de sorties qui diffèrent de celui des entrées ; ce qui exige parfois pour les CUT qui ont le nombre de

La forme générale d'un CSTP est montrée sur la figure 2.21.c. Une chaîne à m registres de CSTP est analogue à un MISR avec un polynôme primitif ($1+x^m$). Cette approche réduit la circuiterie mais affecte l'application de test à cause de la non linéarité des séquences de test.

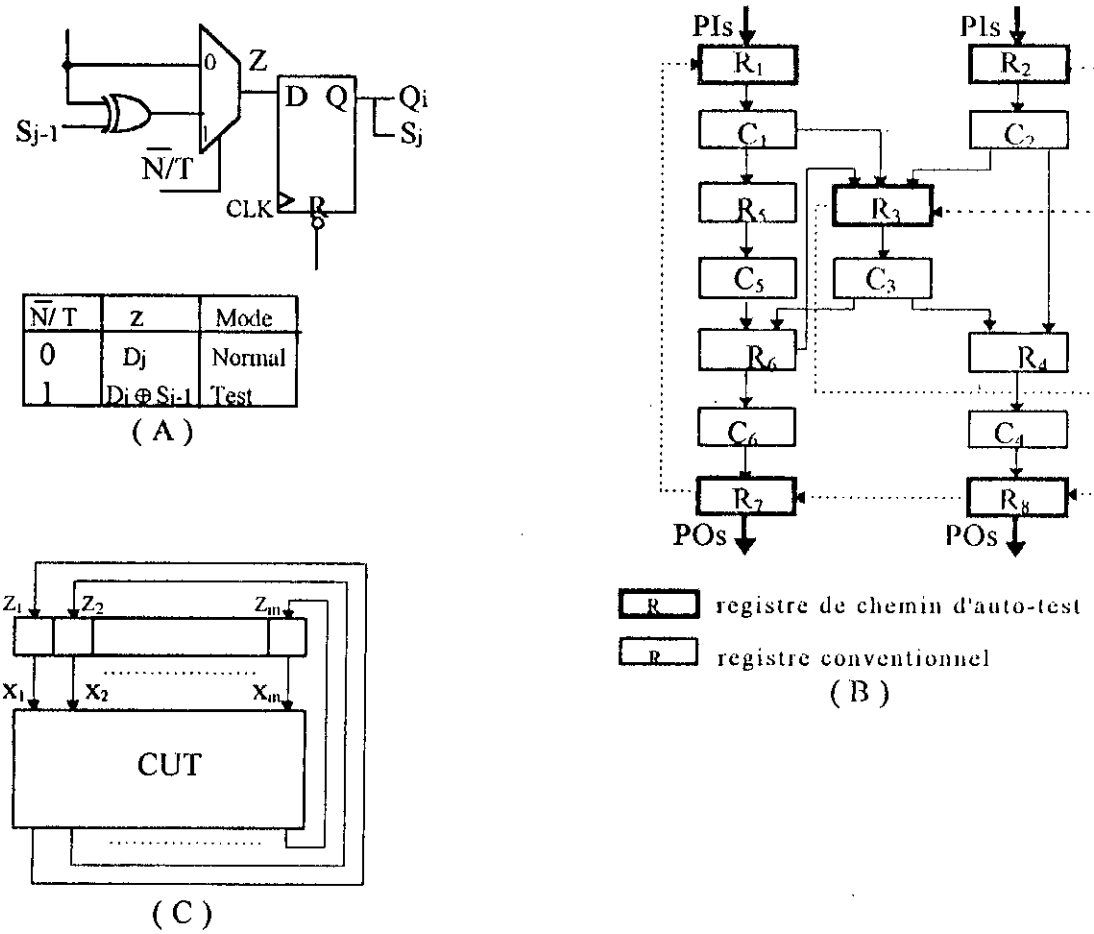


Figure 2.21 : Principe de l'architecture CSTP de BIST

Une autre approche similaire consiste à intégrer un analyseur de signatures SA dans la circuiterie de test afin de simplifier l'opération d'identification des circuits défectueux. La méthode CSTP décrite précédemment n'est pas pratique pour le test des systèmes à microprocesseur, car des opérations de contrôle et de comparaison durant un cycle d'horloge paraissent impossibles.

Les cellules d'auto-test sont modifiées et remplacées par des cellules de BIST automatique dites ABIST comme le montre la figure 2.22. Le registre d'analyse de signature SAR (*Signature Analysis Register*) intégré dans la chaîne de test permet l'action simultanée des mécanismes de compression et de génération.

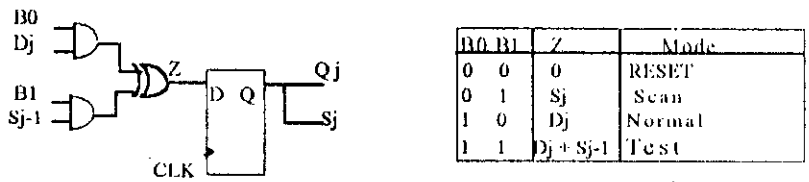


Figure 2.22 : Cellule de ABIST

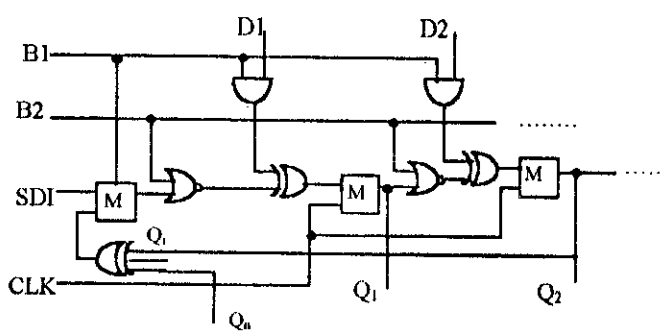
Les différentes opérations de test sont:

- Initialisation des cellules ABIST et du SAR.
- Initialisation de la chaîne de SP interne au circuit.
- Activation du mode test : application des séquences de test au circuit.
- Compression de la réponse du circuit et lecture de la signature.

Le processus se répète jusqu'à l'application de tous les vecteurs de test. On évalue la réponse en la comparant avec la référence. La circuiterie additionnelle est limitée et le temps de test dépend de la couverture de fautes simulées [64,65].

2.4.11 La technique BILBO (*Built In Logic Block Observer*)

Dans les techniques précédentes, des problèmes majeurs sont posés ; un de ces problèmes est la considération de circuits non "partitionnables" ; cependant la technique BILBO consiste à partitionner le circuit en plusieurs modules combinatoires et en un ensemble de registres remplacés par des BILBOs comme le montre la figure 2.23. Les entrées sont pilotées par des registres BILBOs et les sorties activent autres registres BILBOs. On distingue quatre modes de fonctionnement suivant les valeurs attribuées aux lignes de contrôle B1 et B2 [66].

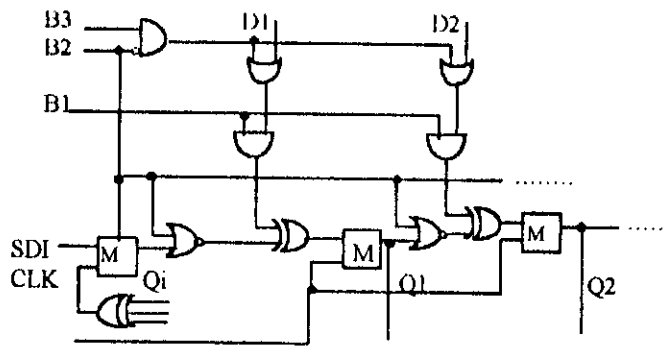


B1	B2	Modes
0	0	SCAN PATH
0	1	Test
1	0	RESET
1	0	NORMAL

M : MUX

Figure 2.23 : Structure et mode de fonctionnement d'une cellule BILBO

L'utilisation de registre BILBO dans une architecture *Pipeline* nécessite l'ajout d'une ligne de contrôle pour séparer les deux configurations (PRPG et MISR) durant le test comme le montre la figure 2.24.



B1	B2	B3	Modes
0	0	0	SCAN
0	1	0	RESET
1	0	0	MISR
1	0	1	PRPG
1	1	0	NORMAL

Figure 2.24 : Structure de CBILBO

Cette méthode est très efficace et permet une réduction en temps et en circuiterie mais ne peut réaliser des opérations de génération et de compression simultanément, de même l'opération d'auto-test nécessite le transfert de séquences de test via les CUTs [1,66].

2.4.12 La technique CABSA (*Cellular Automata Based SA*)

Cette technique consiste à remplacer les bascules par des cellules dites automatiques CA conçues à base de BILBO pour améliorer la testabilité. Une CA se développe en pas discrets en dépendance de la valeur proche d'un emplacement « site » déterminé par sa valeur antérieure et celle d'un ensemble de sites de voisinage. L'étendu de celui ci varie suivant la dimension de CA considérée. L'utilisation d'un PRPG à base de CA paraît très efficace à cause de la réduction significative de la grande corrélation des bits de données ; c'est l'une des principales motivations qui ont amené à utiliser les CA dans les analyseurs.

Les deux méthodes réalisant l'implantation de CA dans un MISR pour former un CMISR (*Cellular Automata Based MISR*) sont décrites par les équations suivantes :

$$R(t+1)=R'(t) \oplus O(t+1) \quad (2)$$

$$R(t+1)=(R(t) \oplus O(t))' \quad (3)$$

R(t) : contenu d'une cellule à l'instant t.

O(t) : sortie du CUT.

R'(t) : valeur incrémentée de CA.

Leur structure est illustrée par la figure 2.25.

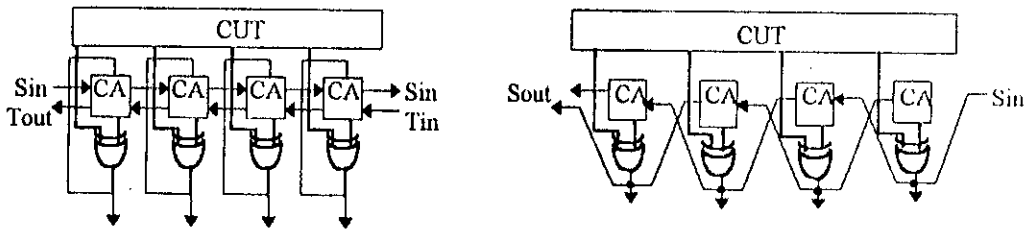


Figure 2.25 : Structures de MISR à base de CA

La figure 2.26 illustre l'utilisation d'un BILBO conçu à base de CA dit CALBO (*Cellular Automaton Logic Block Observer)* dans un système. CALBO peut être configuré soit en PRPG soit en MISR. Durant une phase de test, un CALBO opère en un PRPG et l'autre CALBO en un MISR, assurant ainsi le test d'un bloc. Toutes les applications dans lesquelles sont utilisées les méthodologies de BIST peuvent être donc remplacées par les structures CALBO. L'approche de CABSA garantit une meilleure couverture de fautes et un masquage d'erreurs faible [1,67].

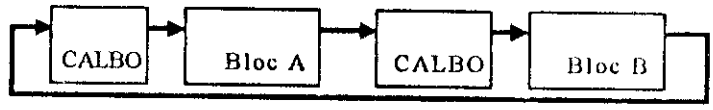


Figure 2.26 : L'utilisation de CALBO dans un système.

2.5 ASSOCIATION BOUNDARY-SCAN ET BIST

L'impact de Boundary-Scan sur les techniques de test est une réalité et apporte une philosophie et stratégie de tests simples. Il permet le test d'un circuit via un connecteur et moyennant l'utilisation d'un registre à décalage adjacent aux pins d'entrées et sorties du circuit; ce registre permet le décalage, l'application et l'acquisition de la donnée de test. Il peut être utilisé non seulement pour un test individuel de circuits, mais aussi pour le test des interconnexions. La figure 2.27 illustre une structure combinée de BIST et BS.

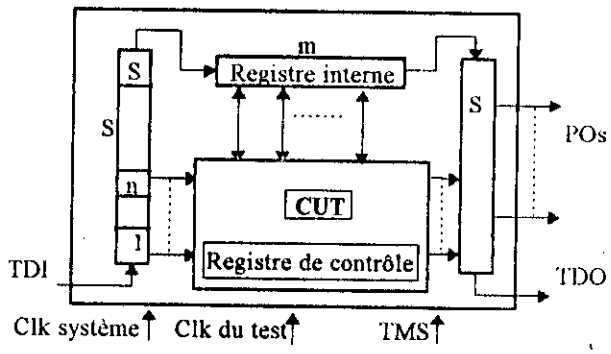


Figure 2.27 : La structure Boundary-Scan-BIST

Dans cette technique les cellules BS sont remplacées par des éléments appelés « template », capables d'être configurés en cellules BS, en plus :

Les éléments adjacents aux Pins d'entrées forment un registre qui peut être configuré en PRPG. Les éléments de sortie peuvent être configurés en MISR ou LFSR

La première étape dans la procédure du test est de configurer le registre d'entrée comme PRPG capable de générer 2^{S-1} séquences non nulles de s bits (ou s est le nombre de cellules d'entrée).

La séquence de test à chaque moment est de r = (n + m) bits, les n premiers bits sont les entrées primaires du circuit tandis que les m derniers bits sont le résultat du SCAN-IN du contenu des dernières flip-flops du générateur aux m bits du registre interne. Les réponses activant les sorties primaires sont appliquées au registre de sortie qui est configuré en MISR. Le contenu du registre scan sera ensuite décalé vers le registre de sortie configuré cette fois en LFSR, dans le même temps le générateur décale les nouveaux m bits dans le registre scan, après ceci un nouveau cycle de test commence.

Finalement la signature produite est observée au niveau du connecteur pour comparaison et diagnostic. Les cellules de scan sont réalisées à base de cellules automatiques [62,63].

Une cellule automatique est une machine à états finis dont l'état prochain est fonction de son état précèdent, de l'état de la cellule précédente et celle de la cellule suivante.

La figure 2.28 illustre une cellule automatique et la figure 2.29 illustre un registre d'entrée à base de cellules automatiques.

L'un des avantages des cellules automatiques est l'élimination de liaison directe entre la première cellule et la dernière. La combinaison des règles 90 et 150 permet d'avoir une génération complète [63].

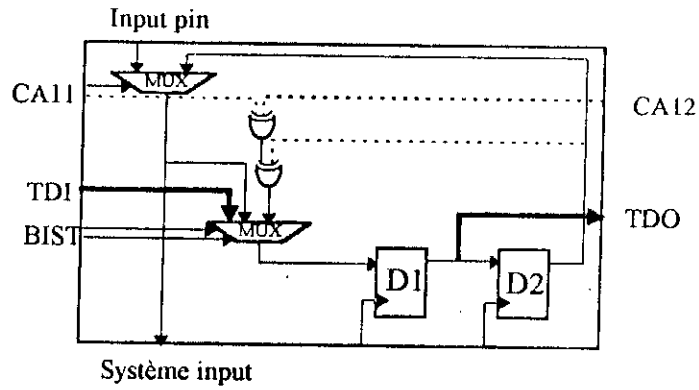


Figure 2.28 : Réalisation d'une cellule règle 90/150

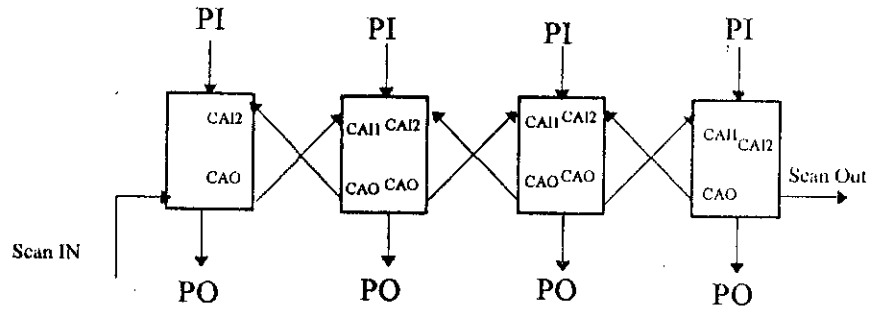


Figure 2.29 : Conception d'un registre d'entrée à l'aide des cellules automatiques

La règle 90 est définie par la relation.

$$a_i(t+1) = a_{i+1}(t) + a_i(t) + a_{i-1}(t) \qquad (2.3)$$

La règle 150 est définie par la relation.

$$a_i(t+1) = a_{i+1}(t) + a_i(t) + a_{i-1}(t) \qquad (2.4)$$

où i est l'ordre de la cellule [63].

Les modes de fonctionnement de BIST-BS sont :

- **Test Externe :** Test des interconnexions de la carte.
- **Test Interne :** Test de la logique interne du circuit.
- **Test d'acquisition :** Les registres d'entrées et sortie sont configurés pour l'acquisition de la donnée durant l'opération normale du circuit.
- **BYPASS :** utilise un MUX pour le transfert direct de la donnée de BS via TDI et TDO.
- **BIST :** Le registre d'entrée est configuré en un PRPG et celui de sortie en MISR.

La couverture de fautes dépend non seulement de l'efficacité des vecteurs de test générés mais aussi de la corrélation entre la période principale du PRPG et de la longueur du registre interne. Cette approche est très efficace, du fait qu'elle permet une large génération de vecteurs de test et une grande couverture de fautes, néanmoins elle exige une circuiterie additionnelle importante pouvant entraîner des délais inacceptables [63].

2.6 CELLULES AVANCEES DE TEST

Plusieurs travaux sont dirigés vers la conception de nouvelles cellules capables de répondre à certaines exigences de test : conception de cellules pour l'auto-test des cartes PCB (*Printed Circuit Board*) et de cellules à injection avec structure BS.

2.6.1 Auto-test de carte (PCB) utilisant l'architecture BS

Dans cette partie, la technique consiste à introduire l'auto-test de PCB en utilisant l'architecture BS. Les techniques d'auto-test de PCB doivent prendre en considération les différentes structures de conception en terme de complexité de génération de vecteurs, de leur application, d'analyse des réponses et résolution dans le diagnostic [68].

2.6.1.1 Extension de Boundary-Scan

Les cellules standards BS sont étendues pour une génération efficace en temps et en compression au sein de la chaîne BS. La compression locale des réponses réduit le temps de décalage vers l'extérieur et simplifie la détection et le diagnostic. Détection et diagnostic sont ainsi rendus indépendants de la topologie et complexité structurelle des réseaux de noeuds sous test.

2.6.1.1.1 Cellule Boundary-Scan du pin d'entrée

Considèrons la "cellule étendue de scan" montrée à la figure 2.30. Cette cellule a toute la fonctionnalité d'une cellule BS standard. La partie marquée X de la cellule montre le circuit logique ajouté à cette cellule. Les deux registres R_1 et R_2 sont définis pour charger et décaler la séquence. Le registre R_3 avec la porte XOR à deux entrées forme le circuit de vérification de parité et de compression.

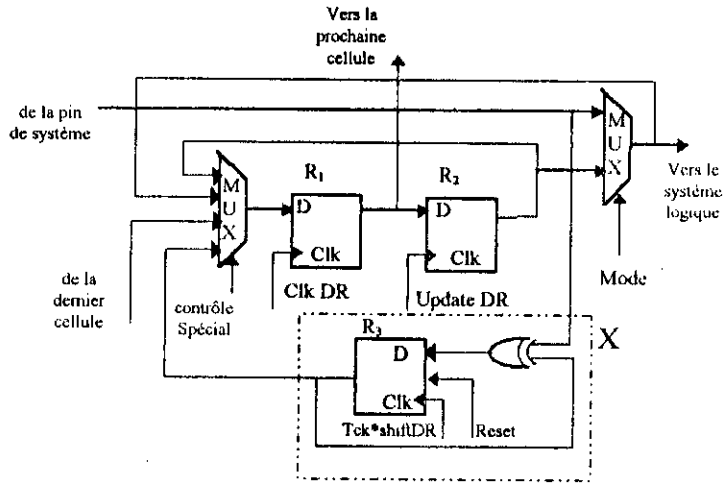


Figure 2.30 : Cellule Scan du pin d'entrée

Avec cet arrangement, la cellule scan réalise la fonction de décalage d'un vecteur et le chargement de vecteurs dans la chaîne de BS en même temps compacte les bits de la cellule scan en provenance du noeud qui lui est connecté. Le point le plus important, ici bien sûr, est que les bits arrivant sur les noeuds au réseau n'affectent pas le contenu de la séquence décalée et introduite dans la chaîne BS. Donc il y a deux opérations, la première est l'obtention des vecteurs de test par le décalage et la

deuxième est la compression de la séquence. Quand la séquence de test est terminée, la donnée de parité compactée en R_3 peut être transférée au registre R_1 et décalée extérieurement vers TDC (*Test Diagnostic Control).*

2.6.1.1.2 Cellule Scan de la broche de sortie

Des séquences de vecteurs sont appliquées en utilisant le mécanisme de cycle suivant : décaler-mise à jour-capter. La cellule est ainsi étendue comme le montre la figure 2.31, pour capter la valeur durant l'opération de capture de donnée de test d'interconnexion ; ceci assure la non destruction de la séquence qui se déplace (*walking sequence*), à cause de cette opération de captage et peut être utilisée comme moyen local de génération de vecteurs.

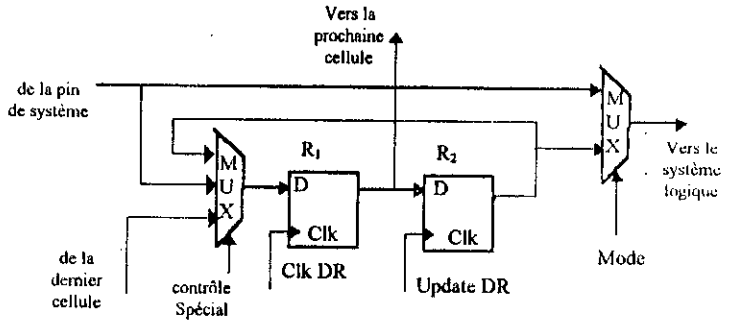


Figure 2.31 : Cellule de Scan du pin de sortie

2.6.1.1.3 Cellule de contrôle

Dans le mécanisme proposé de BIST, les vecteurs de test sont générés par opération interne de décalage de la chaîne de BS. Ceci nécessite d'une part que le contenu de chaque cellule soit modifié après chaque cycle : compaction et décalage, et d'un autre coté, il faut maintenir le contenu de la cellule de contrôle fixe pendant l'application d'une série complète de vecteurs de test. Ces deux exigences sont une extension développée en [68] et représentée en figure 2.32.

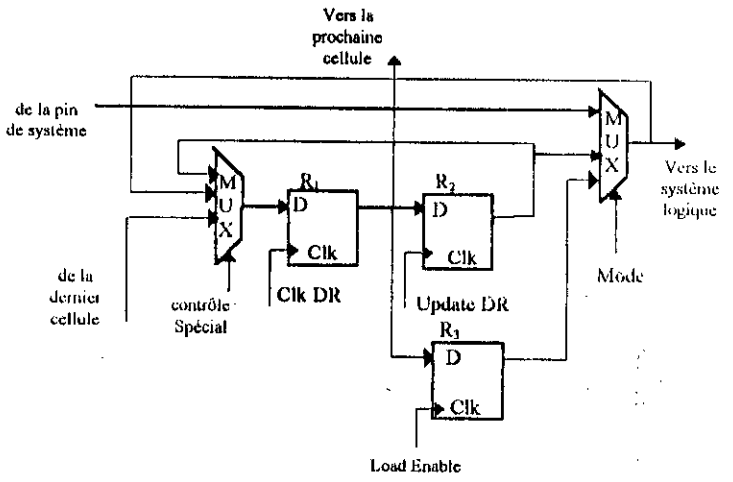


Figure 2.32 : Cellule de contrôle

2.6.2 Cellule BS à injection de fautes

Plusieurs compagnies utilisent la technique d'injection de fautes à travers des pins en *Stuck-at* ou en ***Stuck-open*** mais cette technique est devenue délicate, vu la difficulté d'accès physique à ces pins. On note que l'injection de fautes est une méthode pour vérification de la tolérance de fautes utilisée dans le développement, la vérification et le diagnostic software des systèmes très complexes [69].

La cellule de la figure 2.23 est utilisée pour l'injection de fautes dans la chaîne de boundary scan FIBS *(Fault-Injection Boundary Scan)*. Cette cellule augmente et facilite l'injection de fautes des pins d'entrées et de sorties. Cette nouvelle configuration de la cellule BS a imposé une modification dans la structure du standard BS en additionnant deux champs dans le registre d'instruction : le premier est FIE *(Fault-Injection Enable)* et le deuxième pour la donnée à injecter FID *(Fault-Injection Data)* [69].

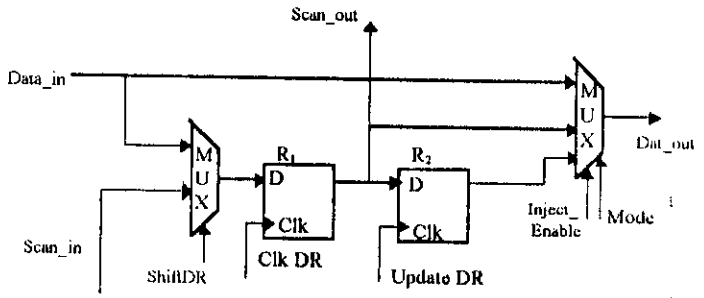


Figure 2.33 : Extension de la cellule BS pour l'injection de fautes

2.7 ORGANISATION ET FONCTIONS DU MITAD

Les concepteurs du MITAD « Module d'isolation, de test et d'aide au diagnostic » ont proposé un circuit capable de réaliser des opérations de test, d'isolation et de diagnostic [70,71], basées sur les techniques de génération et de compression de données. Son intérêt est de :

- Résoudre les problèmes de limitations d'accès aux points de test (circuits montés en surface et sur les deux faces).
- Eviter le recours de re-conception des systèmes pour inclure les techniques BIST ou Boundary-Scan dans tous les composants.
- Favoriser particulièrement le diagnostic au moyen d'un mode d'isolation.
- Réduction du temps de test obtenu grâce à l'action simultanée de génération de vecteurs de test à un CUT et compression de réponses d'un autre CUT en une signature [72].

2.7.1 Organisation du MITAD

La figure 2.34 illustre le schéma synoptique du MITAD.

Ce module est constitué de quatre parties essentielles :

- Un générateur de vecteurs de test pseudo-aléatoires (PRPG).
- Un analyseur de signatures à entrées multiples (MISR).
- Un circuit d'aiguillage à connexions sélectives.

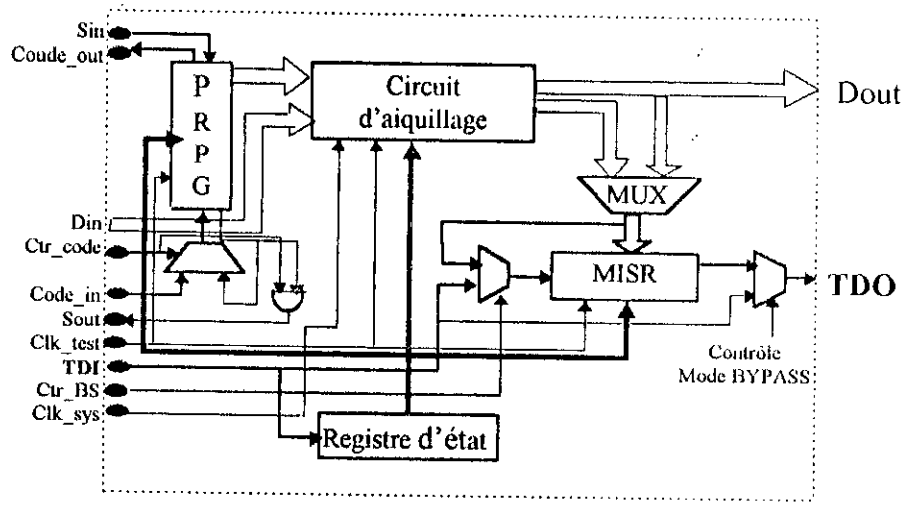


Figure 2.34 : Schéma synoptique du MITAD

2.7.2 MODE DE FONCTIONNEMENT DU MITAD

Les différents modes d'utilisation sont définis suivant l'instruction contenue dans le registre d'état du MITAD. On peut citer le mode normal, génération, analyse, test, Boundary Scan. Tous ces modes sont expliqués dans [72,73,74].

2.8 CONCLUSION

Toutes les techniques de test des circuits séquentiels reposent sur l'introduction de données de test aux bascules pour la formation de l'état du système et l'analyse des réponses de ce circuit soumis aux vecteurs d'entrée. L'optimisation est recherchée au niveau des circuits de cellules, du temps de génération de vecteurs et principalement au niveau de l'efficacité du test.

Chapitre 3

PRINCIPALES APPROCHES DE CONCEPTION DES UNITES DE CONTROLE

1. *ROLE DE L'UNITE DE CONTROLE*
2. *METHODE DE CONCEPTION D'UNE UNITE DE CONTROLE*

3. PRINCIPALES APPROCHES DE CONCEPTION DES UNITES DE CONTROLE

3.1 INTRODUCTION

Pour exécuter les instructions de son répertoire, l'unité centrale doit produire et gérer des signaux de contrôle selon des séquences optimisées en nombre et en temps. L'unité centrale peut être généralement décomposée en deux parties :

- Unité de traitement.
- Unité de contrôle.

La section de traitement contient les éléments matériels tels que ALU, registres à décalages, comparateurs,...etc pour favoriser des activités directes sur les données moyennant des opérateurs spécifiques câblés. Le but de l'unité de contrôle est de fournir les commandes nécessaires pour gérer tout le système d'opération en envoyant les signaux adéquats aux différents éléments qui composent l'unité centrale produisant un contrôle simultané spatial et temporel ; ces signaux sont synchronisés à l'aide d'une horloge principale du système (*Master Clock*).

Les concepteurs d'organes de contrôle ont développé plusieurs techniques dont la plupart appartiennent à l'une des deux catégories suivantes :

- Contrôle par unité câblée.
- Contrôle par unité micro-programmée.

L'unité de contrôle est dite câblée "*Hardwired*" car le circuit final est défini à l'aide des connexions physiques de composants typiques tels que les portes logiques, les bascules, les décodeurs...etc.

Dans l'unité de contrôle dite micro-programmée "*micro-programmed*" toutes les fonctions de contrôle pouvant être simultanément activées sont groupées sous forme de mots de contrôle stockés dans une mémoire séparée et appelée mémoire de contrôle CM (*Control Memory*). Ces mots sont lus séquentiellement à partir de cette mémoire de contrôle et se présentent sous forme de champs de contrôle directement connectés aux unités fonctionnelles pour activer les éléments appropriés. L'activation séquentielle de ces organes réalise les tâches désirées. Cette approche exige l'inclusion d'une mémoire de contrôle en plus de la mémoire principale, i impose donc un autre effort supplémentaire de conception concernant la minimisation de la longueur de la micro-instruction [75].

Il est à noter, que l'unité de contrôle décide des séquences d'opérations en se basant sur l'état du système et sur l'instruction présente. Cet état est matérialisé par des conditions sauvegardées dans des flip-flops et peut être par des signaux produits extérieurement tels que Reset. L'unité de contrôle se présente comme un module ayant pour entrées l'horloge, les signaux d'états, le mot instruction et les signaux spécifiques émanants de l'extérieur et ayant pour sorties les signaux d'activation, de séquencement, de décision et de supervision [76].

8. Spécifier le diagramme d'état de cette unité de contrôle.
9. Spécifier les caractéristiques des éléments "*hardware*" à utiliser dans le contrôleur.
10. Compléter la conception de l'unité de contrôle et dessiner le diagramme logique du circuit final.

De ce fait, une organisation générale de cette approche est montrée sur la figure 3.1 :

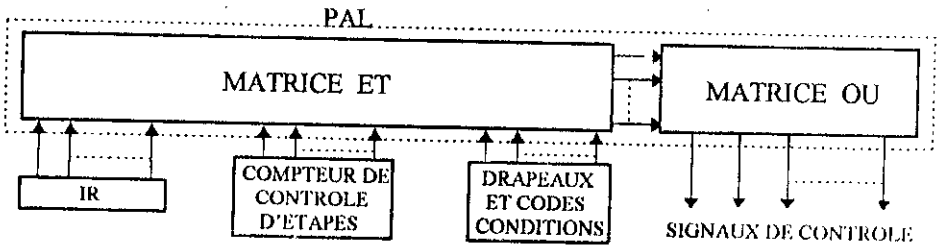


Figure 3.1 : Organisation générale de l'unité de contrôle câblée

Après spécification du diagramme d'états de l'UC, on peut ensuite utiliser les règles de conception de circuits séquentiels (méthode globale, partitionnement, connexions et commandes) :.

a. Une méthode simple est de concevoir un circuit qui, pour tout état donné, génère les signaux de contrôle (sorties) et définit son prochain état et les différents chemins transitoires nécessaires à l'accomplissement de la tâche désirée, en se basant uniquement, sur l'instruction et les conditions (bits d'état). Le bloc diagramme de la figure 3.2 illustre cette méthode.

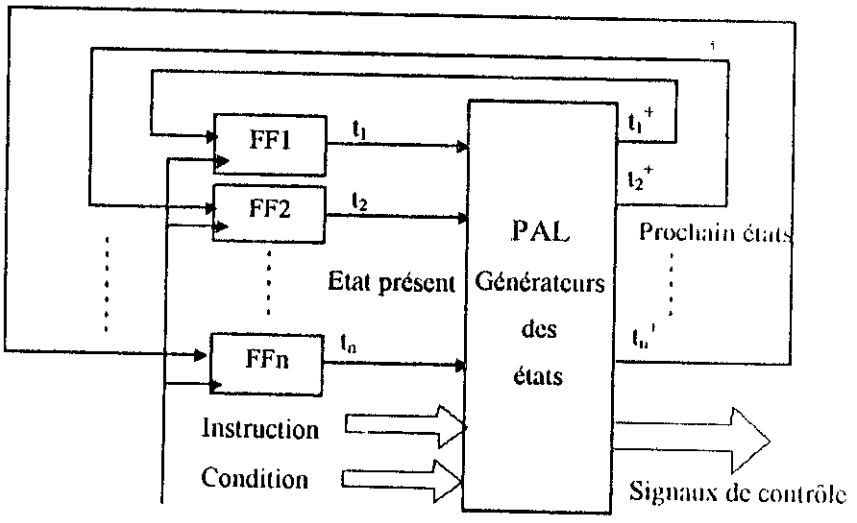


Figure 3.2 : Organisation d'une unité de contrôle par PAL

L'utilisation de circuit PALs allège énormément la conception suivant cette approche. En effet, il suffit uniquement de programmer le PAL et le problème de câblage se trouve évité en plus d'une réduction systématique du nombre de portes logiques [76].

b. Une deuxième méthode est d'utiliser un compteur qui peut produire un nombre supérieur ou égal au nombre total d'état du contrôleur. Pour un état donné, le chargement du compteur n'est activé que si

changements des contenus de la MC, c'est pourquoi cette approche microprogrammée offre plus de flexibilité que celle dite câblée [76].

3.3.2.2 Structure de la micro-instruction

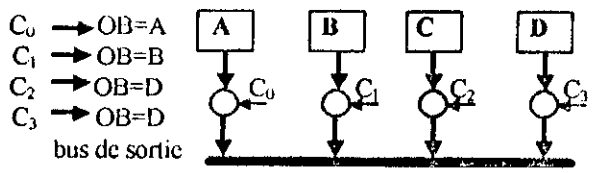
En général, toutes les micro-instructions ont deux champs importants, le champ de contrôle et le champ adresse. Le champ de contrôle indique les lignes de contrôle à activer, alors que le champ d'adresse spécifie l'adresse de la micro-instruction suivante à exécuter.

La longueur de la micro-instuction décide la taille de la mémoire de contrôle et de son coût. Cette longueur est reliée directement aux facteurs suivants:

1. Le degré de parallélisme (le nombre de micro-operations qu'on peut activer simultanément).
2. L'organisation du champ de contrôle.
3. La méthode avec laquelle l'adresse de l'instruction suivante est spécifiée.

La structure de la micro-instruction est similaire à l'instruction d'un ordinateur. Lorsqu'il est possible d'exécuter plusieurs micro-opérations simultanément, toutes ces micro-opérations seront produites en parallèle et donc peuvent être spécifiées par un code opération commun. Ceci permettra d'avoir de petits microprogrammes mais, lorsqu'il y a abondance de parallélisme, la longueur des micro-instructions augmente. Les petites micro-instructions ont une aptitude limitée à la formulation du parallélisme, car elles ne peuvent pas exprimer un parallélisme massif et en définitif, la longueur du microprogramme utilisant ces instructions se trouvera allongée.

L'information de contrôle peut être organisée sous différentes manières. La plus simple est d'organiser le champ de contrôle de telle façon que toutes les lignes de contrôle aient un bit représentatif. C'est le parallélisme parfait qui n'a pas besoin de décodage. Cependant il conduit à une utilisation incomplète de l'espace de la mémoire de contrôle quand il est impossible d'évoquer toutes les opérations de contrôle simultanément ; par exemple ; supposant qu'on à 4 registres A, B, C et D comme le montre la figure 3.4. Puisqu'on à un seul bus de sortie "*output bus*", il est donc impossible d'autoriser plus d'un transfert à la fois. Si on attribue à chaque bit du champ de contrôle une ligne de contrôle, on obtient l'ensemble de commandes décrites dans le tableau de la figure 3.4.



LinVal	C_0	C_1	C_2	C_3
OB=A	1	0	0	0
OB=B	0	1	0	0
OB=C	0	0	1	0
OB=D	0	0	0	1
NOP	0	0	0	0

Figure 3.4 : Fonctionnement d'un BUS

Donc on a 5 combinaisons binaires seulement pour ce format; d'où si on utilise le codage, on a besoin seulement 3 bits comme le montre la figure 3.5. Ce format conduit à un champ de contrôle plus petit et donc à de micro-instructions petites mais un décodeur est exigé [76].

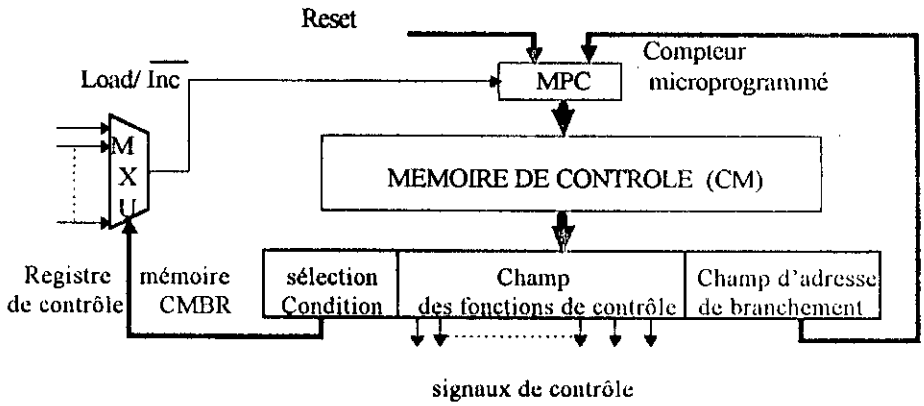


Figure 3.6 : Organisation générale de MPCU

3.3.2.3.1 Le registre mémoire de contrôle "CMBR" (*Control Memory Buffer Register*)

Ce registre a les mêmes fonctions que celles du registre dit MBR de la mémoire principale d'un ordinateur [75]. Il est fondamentalement un latch qui agit comme un buffer pour les micro-instructions ramenées de la MC. Généralement, chaque instruction possède 3 champs distincts comme le montre la figure 3.6.

Le champ "*condition select*" sélectionne les conditions externes à tester; si la condition sélectionnée est juste, la sortie de MUX doit être à 1, ainsi la sortie du MUX est connectée à l'entrée "load" du MPC, donc le MPC doit être chargé par l'adresse spécifiée dans la partie du champ adresse de la micro-instruction. Cependant si la condition externe sélectionnée est fausse, le MPC doit pointer vers la micro-instruction suivante à exécuter. Par conséquent, cet arrangement permet un branchement conditionnel. Le champ de contrôle "*Control Function Field*" de la micro-instruction contient l'information du contrôle sous une forme encodée [76].

3.3.2.3.2 Le compteur de microprogramme (MPC)

Ce registre nommé MPC (*Micro-Program Counter)* jouant le même rôle que le pointeur PC, pour cette raison, il est usuellement incrémenté après chaque micro-instruction de type fetch et donc pointe à l'adresse de la micro-instruction suivante à exécuter, c-à-d initialement il est chargé à travers une source externe pour déterminer l'adresse de démarrage d'un microprogramme. D'après ceci, le MPC s'incrémente après chaque micro-instruction "*fetch*" transférée au CMBR. Lorsque l'instruction de branchement est rencontrée, le MPC doit être chargé par le contenu du champ d'adresse de branchement "*Branch Address Field*" de la micro-instruction contenue dans le CMBR.

3.3.2.3.3 Multiplexeur de sélection des conditions externes (*External Condition Select MUX*)

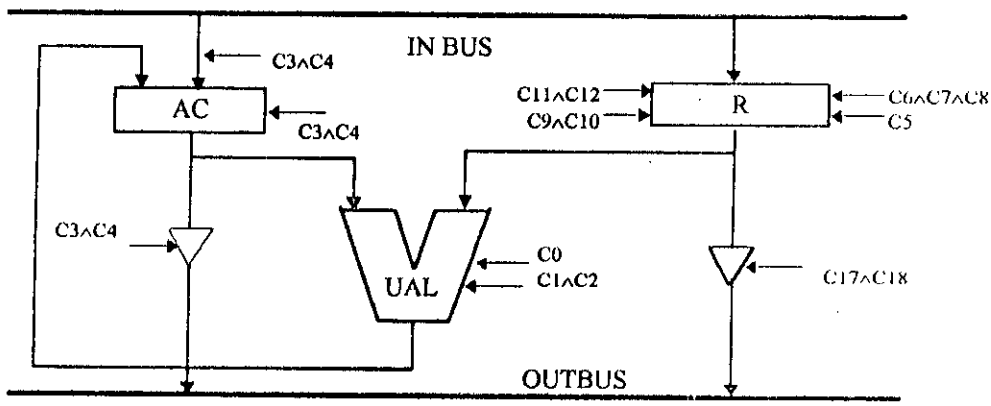
Ce MUX sélectionne une des conditions externes suivant le contenu du champ de sélection de conditions « *Condition Select Field* » de la micro-instruction courante. Si la condition à sélectionner est spécifiée sous forme encodée, les micro-instructions sont étroites et par conséquent on obtient une mémoire de contrôle réduite et donc de faible coût [17].

l'approche encode les fonctions de contrôle d'une manière logique. Premièrement, il y a regroupement suivant des unités fonctionnelles particulières puis encodage par la suite. Pour illustrer cette idée, on donne l'exemple d'une MPCU où des micro-instructions ont un champ de contrôle encodé comme le montre la figure 3.8 ; considérant les fonctions de contrôle suivantes en langage RTL :

C0 : ACC ← ACC+R
C1∧C2 : ACC ← ACC-R
C3∧C4 : ACC ← 0
C5 : R ← 0
C6∧C7∧C8 : R ← R+1
C9∧C10 : R ← R-1
C11∧C12 : R ← INBUS
C13∧C14 : ACC ← INBUS
C15∧C16 : OUTBUS ← ACC
C17∧C18 : OUTBUS ← R

ACC est l'accumulateur et R est le registre de MPCU.

quatre unités fonctionnelles sont exigées comme le montre la figure 3.8 :



Champ des fonctions de contrôle							
Champ ACC		Champ COUNT du BUS		Champ d'entrée du BUS		Champ de sortie	
A1	A2	B1	B2	I1	I2	D1	D2

A1	A2	Signaux activés	B1	B2	Signaux activés	I1	I2	Signaux activés	D1	D2	Signaux activés
0	0	aucun	0	0	aucun	0	0	aucun	0	0	aucun
0	1	C0	0	1	C5	0	1	C11, C12	0	1	C15, C16
1	0	C1,C2	1	0	C6, C7, C8	1	0	C13,C14	1	0	C17, C18
1	1	C3,C4	1	1	C9, C10	1	1	aucun	1	1	aucun

Figure 3.8 : Exemple de structure de L'MPCU selon l'approche de S.R.DAS

Dans cet exemple, la longueur du champ de contrôle est de 8 bits seulement, cependant elle est de 19 bits sans encodage [76].

contrôle (CFC), puisque ce dernier n'est exploité que lorsqu'il y a un branchement et ceci représente l'avantage de cette approche qui se traduit par une réduction de la longueur du microprogramme et devient une caractéristique et une spécificité propre de ce type de MPCU.

Le champ de sélection des conditions (CSC) spécifie les différents types des micro-instructions par action et affectation de conditions sur le MUX qui ordonne à son tour au MPC de s'incrémenter ou d'être chargé par l'adresse de branchement spécifiée dans le CFC. La logique combinatoire LOG COMB n'active les commandes de contrôle que pour les micro-instructions normales (les instructions à opérations) suivant leur adresse dans le CSC.

Le MPC est chargé par le contenu du champ des fonctions de contrôles lorsqu'il y a une micro-instruction de branchement suivant la condition du MUX spécifiée par le contenu du champ de sélection des conditions, cependant si cette condition n'est pas vérifiée, le MPC est incrémenté de 1 qui est le même cas pour l'exécution de micro-instructions normales.

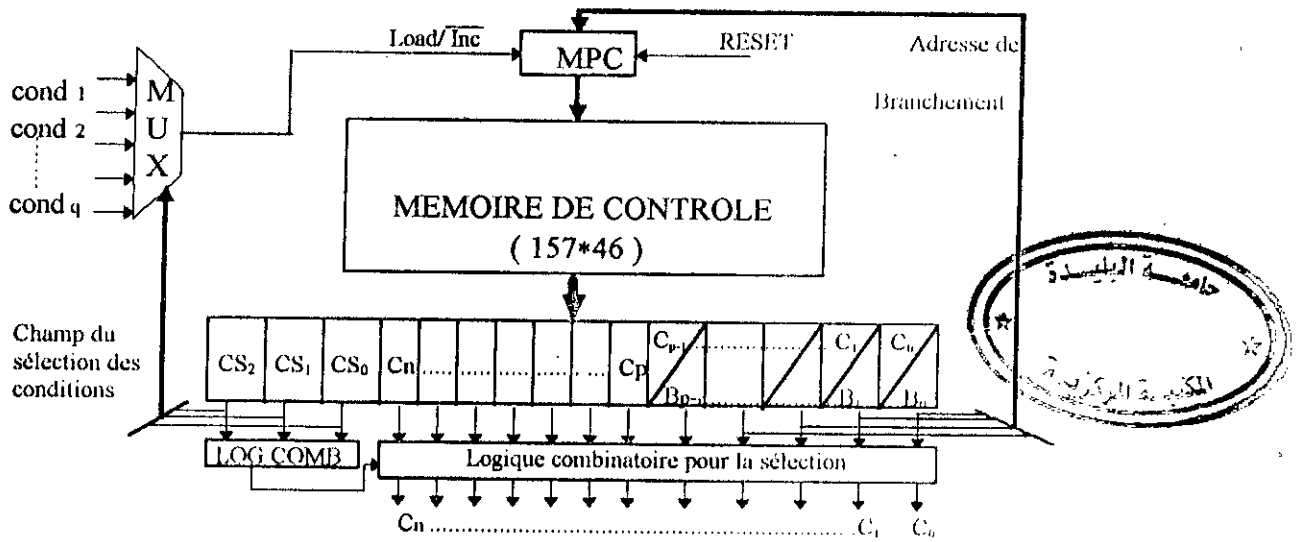


Figure 3.10 : Organisation générale du MPCU a multiple formats des M-INST

Note : On a utilisé cette approche pour réaliser notre unité de contrôle microprogrammée à cause de sa commodité, sa simplicité ainsi que l'avantage qu'elle nous fournit et qui ne se trouve dans aucune autre approche c'est la réduction de la longueur du microprogramme [76].

L'exécution d'une instruction recommande les étapes suivantes :

1. Rechercher l'instruction (*FETCH*).
2. Décoder l'instruction pour identifier l'opération.
3. Si l'opération demandée est l'opération HALT donc on passe à l'étape 6, si non on continue.
4. Chercher l'opérande et l'opération désirée.
5. Faire l'étape 1.
6. Exécution d'une infinité de boucles (loops).

La première étape est dite "*cycle fetch*", et les autres étapes sont appelées "cycle exécution".

PROCESSEUR DE TEST A UNITE DE CONTROLE CABLEE

4. PROCESSEUR DE TEST A UNITE DE CONTROLE CABLEE

4.1 INTRODUCTION

L'introduction des techniques de test de type *Scan* [1], *Boundary-Scan* "BS" [21] et *Built In Self Test* "BIST" [62,63] qui généralement se matérialisent par l'introduction de chemins logiques au sein des circuits intégrés numériques pour favoriser des actions spécifiques de test, constitue une solution attractive pour les cartes et modules numériques de densités élevées. En effet, les problèmes d'accessibilité, de contact et de production de test sont allégés mais le pouvoir de ce mécanisme reste limité à cause de certaines contraintes dont on peut citer :

- Le temps important de test consacré à l'introduction de très longues séquences en mode de décalage série et à la génération d'un nombre important de vecteurs de test pour atteindre des couvertures de fautes élevées dont le seuil de validité reste toujours discutable.
- Compression de réponses du système sous test avec objectif principal la diminution du volume d'information à traiter mais introduisant un masquage incontrôlable.

Le test des systèmes de CIs, ne comportant pas tous des mécanismes de type Scan, BS et BIST ou partiellement, nécessite l'utilisation d'un équipement de test supplémentaire pouvant au moins effectuer l'isolation, la génération, la compression et la collecte de données.

La diversité des pannes, des fautes et des CIs et leur liaison et dépendance technologique exigent l'utilisation d'une variété d'algorithmes de test et imposent en plus l'utilisation d'ATE *(Automatic Test Equipment)* multiples. Le test à travers le mode scan unique est donc incomplet.

La majorité des équipements de tests actuels ne répondent pas aux contraintes et réductions imposées, de plus en plus, par les nouvelles techniques de conception. Toutes les solutions qui peuvent concourir à une amélioration de la qualité, de la réduction des coûts des produits à travers les tests doivent être prises et les systèmes à caractéristiques programmables deviennent la solution inévitable [78].

Dans ce travail, on propose une étude d'un processeur de test PTDC* (Programmable Test And Diagnostic Controller) *sous la forme d'un module ou d'un circuit "chip" capable d'exécuter des programmes spécifiques de test et de présenter des diagnostics éventuels. Les opérations de génération de vecteurs de test, de compression de données et d'isolation sont des mécanismes à adapter aux différentes situations qui se présentent lors du test.

L'action simultanée de génération de vecteurs de test, pour un circuit sous test, et la compression des réponses d'un autre CUT en signatures permet une réduction de temps de test. En plus, ce processeur doit être lui même testable ; son auto-testabilité est impérative car la complexité des traitements qu'il est appelé à réaliser, sous des conditions critiques, lui impose une certaine fiabilité et surtout une certaine assurance de fonctionnalité.

Nous proposons, dans la figure 4.2, le schéma synoptique d'un processeur qui répond aux exigences de test citées et qui offre la souplesse de la programmation pour la production de contrôle des différentes opérations de test.

Ce processeur de test se compose des parties principales suivantes : Une logique de génération, une logique de compression, un circuit de transfert et d'aiguillage d'information, une unité arithmétique et logique "UAL", des registres de travail, une mémoire locale de programmation, une table de données et de vecteurs spécifiques pour le contrôle des mécanismes Boundary-Scan et une unité de contrôle qui produit tous les signaux électriques de commande.

Dans ce travail, on présente l'étude de deux processeurs de test : un PTDC à unité câblée et un PTDC à unité microprogrammée.

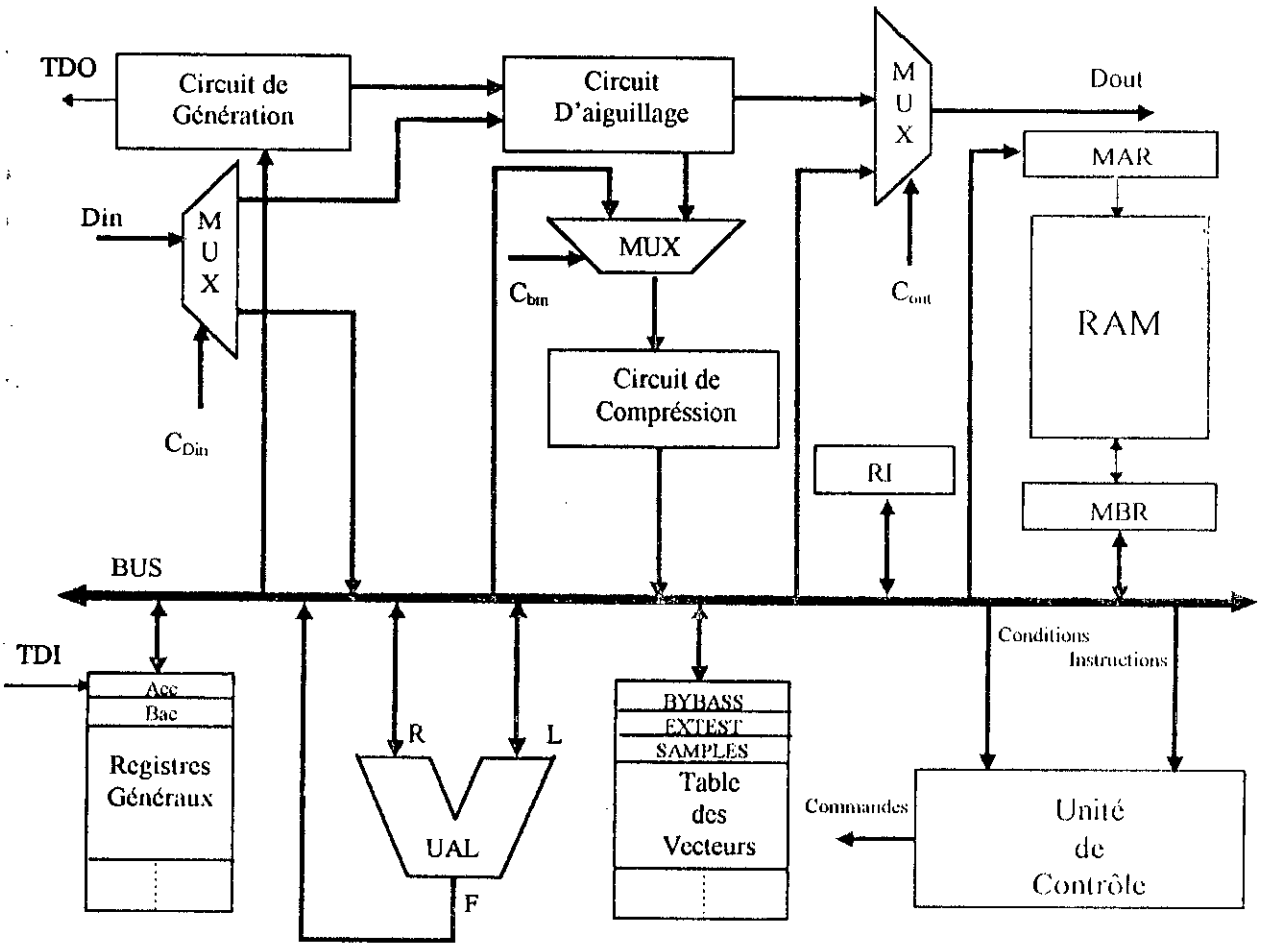


Figure 4.2 : Schéma synoptique du processeur de test

4.3 PROCESSEUR DE TEST A UNITE DE CONTROLE CABLEE

Après une étude des différentes techniques de conception de processeurs et définition de leur jeu d'instructions et particulièrement des modèles RISC *"Reduced Instruction Set Computer"* (ordinateur à jeu d'instructions réduit) [79], nous avons développé un processeur dédié au test de type RISC.

La figure 4.3 montre le bloc diagramme du processeur de test à unité de contrôle câblée et les commandes appliquées aux sous modules.

Il est à rappeler que le contrôle câblé (*hardwired control)* permet, à l'aide d'une logique simple, d'obtenir une exécution rapide des instructions, en un seul cycle machine [79].

Pour développer ce processeur nous avons suivi les étapes suivantes :

1. Détermination des différentes tâches à traiter.
2. Développement de schéma bloc de ce module de test.
3. Définition du graphe d'états de l'unité de contrôle "UC", et réalisation des optimisations et simplifications nécessaires.
4. Développement du schéma logique des différentes parties actives.
5. Validation des différentes parties, vérification des séquencements et production des signaux indispensables au fonctionnement.

Dans les paragraphes qui suivent, les différents modules de ce processeur de test sont expliqués.

4.3.1 Logique de génération

Le circuit générateur des vecteurs de test, à appliquer au CUT, est divisé en deux parties pour assurer une couverture de fautes élevée [31] et réaliser un contrôle simple sur les vecteurs produits.

4.3.1.1 PRPG (*Pseudo Random Test Generator*)

La figure 4.4 présente le circuit logique d'un générateur de vecteurs de test pseudo-aléatoire (PRPG) à base d'un registre à décalage autonome (LFSR) [38], à état initial et polynôme caractéristique programmables.

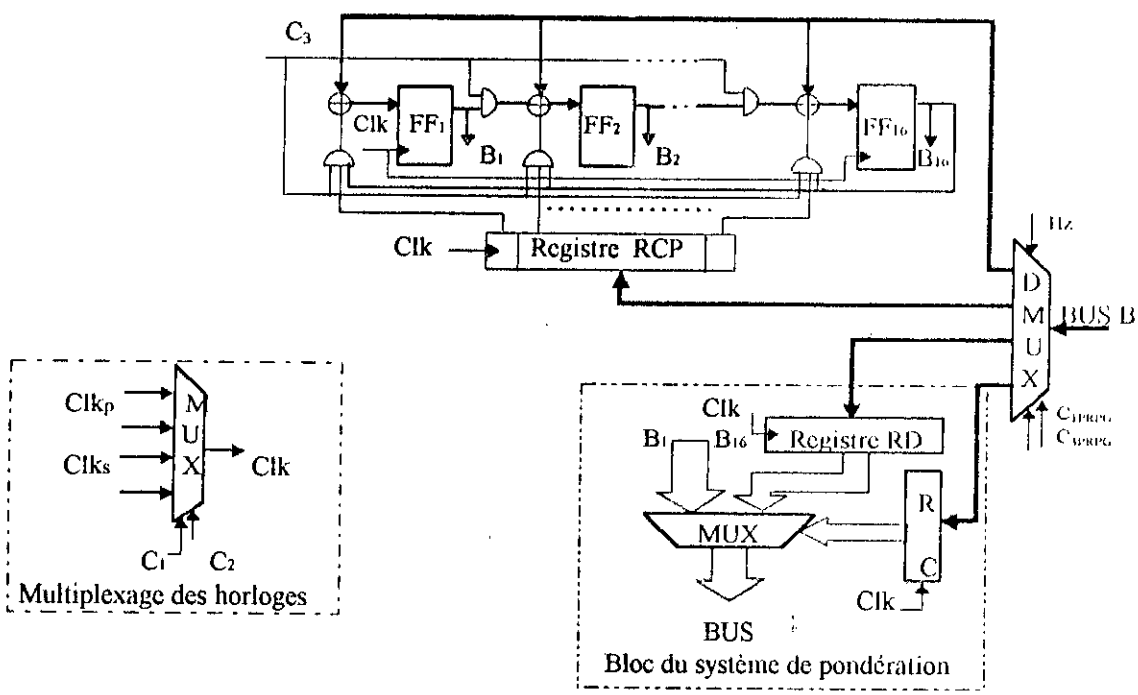


Figure 4.4 : Schéma synoptique du PRPG

Exemple

Pour fixer la ligne 2 à 1 et la ligne 3 à 0 il faut choisir le registre de commande (0110 0000) et le registre de donnée (x10x xxxx) ce qui nous donne un nouveau vecteur V.

$$(B)\begin{pmatrix} V_{11} & \cdots & V_{201} \\ \cdot & \cdots & \cdot \\ \cdot & \cdots & \cdot \\ \cdot & \cdots & \cdot \\ V_{116} & \cdots & V_{2016} \end{pmatrix} \xrightarrow{Après} (V)\begin{pmatrix} V_{11} & \cdots & V_{201} \\ 1 & 1 & 1 \\ 0 & 0\ 0 & 0 \\ \cdot & \cdots & \cdot \\ V_{116} & \cdots & V_{2016} \end{pmatrix}$$

4.3.2 Logique de Compression

Pour réaliser une compression matérielle simple, qui ne présente pas un facteur de masquage important, on a utilisé la technique de la division polynomiale à plusieurs entrées MISR (*Multiple Input Signature Register*) ayant un polynôme caractéristique et un état initial programmables [56]. La compression, des réponses du CUT en signatures, est montrée sur la figure 4.6.

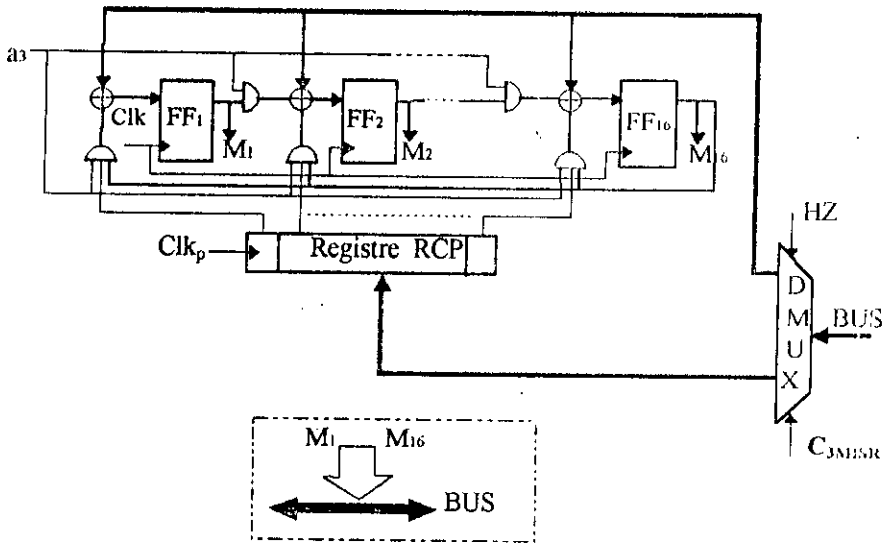


Figure 4.6 : Schéma synoptique du MISR

Le registre RCP est un latch qui maintient la valeur du polynôme diviseur du MISR. Si on place dans ce registre une valeur égale (9111)h, le polynôme caractéristique est alors $x^{16}+x^{12}+x^{8}+x^{4}+1$. La compression se fait soit à la fréquence du système sous test Clks ou à la fréquence du processeur Clkp suivant les deux commandes C_1 et C_2. Avec ces mêmes commandes, on transfert les données vers le registre RCP ou vers le registre contenant l'état initial. Le résultat de la compression se trouve dans le registre MISR (M_1,..., M_{16}) et peut être transféré facilement vers le bus. Le tableau 4.2 montre le fonctionnement complet de ce MISR.

La recherche d'un polynôme qui assure un masquage réduit est recommandée. Les polynômes les plus usités sont consignés dans des tables [6,54].

Après une étude très exhaustive de toutes les spécifiés recommandées pour réaliser le test par Boundary Scan, nous sommes positionnés du côté de la définition d'instructions à format général (pour une exploitation directe des techniques existantes) auxquelles il faut joindre un ensemble de paramètres pré-définis pour suivre le cheminement de tout le graphe d'état du TAP (*Test Acces Port)*, contrôleur du BS. Rappelons que les cellules BS permettent de capter une donnée pour la propager vers d'autres cellules ou la transférer vers la logique interne suivant les commandes de registre d'instructions. Ces cellules sont contrôlées par le TAP qui travaille comme une machine à états finis synchronisée par une ligne de commande TMS (*Test Mode Select)*.

Les données enregistrées pour réduire les accès vers l'extérieur ou vers la mémoire de programme sont montrées dans le tableau 4.3. Quatre bits de sélection d'une position sont définis.

adresse				Contenu de la mémoire
C_{TV1}	C_{TV2}	C_{TV3}	C_{TV4}	
0	0	0	0	0010
0	0	0	1	02DF
0	0	1	0	0002
0	0	1	1	0001
0	1	0	0	Code BYPASS
0	1	0	1	Code EXTEST
0	1	1	0	Code INTEST
0	1	1	1	Code RUN BIST
1	0	0	0	Code IDCODE
1	0	0	1	Code SAMPLE
1	0	1	0	0006
1	0	1	1	0017
1	1	0	0	007D
1	1	0	1	0007
1	1	1	0	AAAA
1	1	1	1	5555

Tableau 4.3 : Adressage et contenu mémoire de la table de données

4.3.5 Registres du processeur

Le test des différents circuits nécessite plusieurs registres tels que des registres pour réaliser le transfert de données, la récupération des réponses, le décalage et l'adressage de la mémoire.

Comme les registres sont des éléments mémoires à très faibles temps d'accès, et leur connexion aux bus ne nécessite qu'un contrôle par mots binaires, le traitement séquentiel des informations et le cheminement de données sont donc rapides et facilement contrôlables. Alors différents types de registres sont exploités. Noter l'introduction de registres pour l'envoi et réception en mode série à travers TDI (entrée sérielle de circuit) et le registre TMS (RTMS) pour commander le signal TMS de TAP. Les registres utilisés sont définis ci-dessous :

4.3.5.1 Accumulateurs

Certaines instructions, telles que Scan-DR-IO et Scan-DR-O, nécessitent deux registres, un pour l'envoi de la donnée en mode série à travers TDI et un autre pour recevoir les réponses de la chaîne de BS.

C_1	C_2	C_3	FONCTION DU COMPTEUR
0	0	0	Aucune opération
0	0	1	Chargement
0	1	0	Lecture (output enable)
0	1	1	Décrementation du compteur suivant l'horloge processeur
1	0	0	Décrementation du compteur suivant l'horloge système
1	0	1	Incrémentation du compteur suivant l'horloge processeur
1	1	0	Incrémentation du compteur suivant l'horloge système
1	1	1	Le chargement du compteur par FFFF(h)

Tableau 4.7 : Mode de fonctionnement du compteur

4.3.5.5 Registres de Programme

Deux registres PC (*Program Counter*) et IR (*Instruction Register*) sont définis pour simplifier le contrôle des instructions et synchronisation. Le registre PC pointe toujours vers l'instruction suivante tandis que le registre IR contient l'instruction en cours d'exécution. L'utilisateur ne peut accéder au contenu de ces deux registres que par programme. Les modes de fonctionnement de ces deux registres sont donnés dans les deux tableaux 4.8.(a) et (b) :

L	INC	RESET	Fonction
1	0	0	Chargement de PC
0	1	0	Incrémentation
x	x	1	PC←0

(a)

L	Reset	Preload	Fonction
1	0	0	Chargement de IR
0	0	1	IR ← FF
0	1	1	IR ← 00

(b)

Tableau 4.8 : Mode de fonctionnement de (a) PC et (b) IR

4.3.6 Unité arithmétique et logique « UAL »

Cette unité est connectée aux bus comme le montre la figure 4.3. Son fonctionnement est résumé sur le tableau 4.9.

$C_{1\,UAL}$	$C_{2\,UAL}$	$C_{3\,UAL}$	Fonction réalisée par UAL (F)
0	0	0	F = 0
0	0	1	F = NEG (R)
0	1	0	F = L (ET) R
0	1	1	F = L (OU) R
1	0	0	F = L.1
1	0	1	F = L+R
1	1	0	F = L.R
1	1	1	F = L (XOR) R

Tableau 4.9 : Mode de fonctionnement de l'UAL

4.3.7 Mémoire de programmation RAM

C'est une mémoire statique de mots binaires de largeur de 16 bits dont la capacité peut s'étendre jusqu'à 64 ko. Elle est utilisée avec un registre d'adresse MAR (*Memory Address Register*) qui pointe vers l'adresse désirée, et un buffer pour le transfert de données et pour le synchronisme des échanges,

Processor) dont le jeu d'instructions est principalement dédié au traitement de séquences de données représentatives des réponses et fonctionnalité de circuits. L'accent est particulièrement mis sur la production d'opérations spécifiques pour le test des éléments basés sur les techniques BS, BIST, SCAN, ...etc.

4.4 JEU D'INSTRUCTIONS DU PROCESSEUR DE TEST

L'exécution de chaque instruction exige un ensemble de micro-instructions qui forment cette instruction. Chaque micro-instruction est définie par un état qui sera exécuté pendant un cycle d'horloge, les variables d'entrée, les variables de sortie et les conditions de transition.

4.4.1 Micro-Instructions

Une micro-instruction est définie comme étant le code qui déclenche l'ensemble des micro-commandes pour l'exécution de micro-opérations pendant une unité du cycle machine. Ces micro-instructions sont expliquées sous format du langage RTL (*Register Transfer Language*) [76,79].

Avant la formation du tableau 4.10 des micro-instructions, sous forme d'états, on a sélectionné toutes les instructions qui sont recommandées pour notre application (instructions logiques, test, BIST, BS, ...etc) conformément et suivant l'organigramme 4.1, ensuite, on a défini toutes les micro-commandes des blocs actifs du processeur. Nous avons rassemblé et optimisé toutes les micro-instructions qui nécessitent les mêmes commandes pendant un état. (Nous avons appliqué le fusionnement des états suivant la méthode de graphe d'états d'une machine synchrone).

Etat	Micro-Instruction	Signaux de contrôle
t0	Reset PC	Rpc
t1	MAR ← PC	Lmar
t2	MBR ← ((MAR)), Inc PC	Incpc, R/W, En, Lmbr
t3	IR ← MBR	Lembr, Lir
t4	MAR ← PC	Lmar
t5	MBR ← ((MAR))	R/W, En, Lmbr
t6	PC ← MBR	Lpc, Lembr
t7	MAR ← Si	Cm_2, Lmar
t8	Aacc ← MBR	Lembr, C_2aac
t9	Bacc ← MBR	Lembr, C_2bac
t10	MAR ← Di	Cm_1,Lmar
t11	MBR ← Aacc	Lmbr, Lembr,C_1aac
t12	MBR ← Bacc	Lmbr, Lembr, C_1bac
t13	((MAR)) ← MBR	En
t14	Cmp ← Aacc	C_1aac, C_3cmp
t15	Cmp ← Bacc	C_1bac, C_3cmp
t16	Dec Si	C_1si, C_2flg, Lflg
t17	Dec Di	C_1fg, C_1di, Lflg
t18	Inc Cmp	C_1cmp, C_2cmp
t19	Dec Cmp	C_2cmp, C_3cip
t20	MAR ← MBR	Cm_1, Cm_2, Lmar
t21	Inc Si	C_2si, C_3si
t22	Inc Di	C_2di, C_3di
t23	Aacc ← MBR	Lembr, C_2aac
t24	Bacc ← MBR	Lembr, C_2bac

t82	Buf2 ← Buf1 OR Bacc	C_1aac, Lflg, Lbuf$_2$, C_2ual
t83	Aacc ← Buf2	C_2aac, Lebuf$_2$
t84	Buf2 ← Buf2-Bacc	Lflg, C_1bac, Lbuf$_2$, C_2ual, C_3ual
t85	MBR ← ((MAR))	R/W, En, Lmbr
t86	SI ← Cmp	C_3si, C_2cmp
t87	Buf2 ← Buf1 & Bacc	Lflg, C_1bac, Lbuf$_2$, C_1ual, C_2ual
t88	Buf2 ← /Buf1	Lflg, Lbuf$_2$, C_1ual, C_2ual, C_3ual
t89	Aacc ← Din	Cdin,C_2aac
t90	TDO ←Shift Aacc, Dec Cmp, Shift Bacc ← TDI	C_1td0, C_1aac, C_2aac, C_2cmp, C_3cmp, C_1bac, C_2bac
t91	DI ← SI	C_1di, C_2si
t92	A=0, Gen, Dec DI, TDO ← TDO PRPG	C_1di, C_2di, C_1prpg, C_2prpg, C_2tdo, C_1flg, Lflg
t93	Gen, Dout ←VT, MISR ←Din, Dec Cmp	A_1, Cdin, C_1misr, C_1prpg, C_2prpg, C_1cmp
t94	Shift Bacc ← TDI, Dec Cmp	C_2cmp, C_3cmp, C_1bac, C_2bac
t95	Dou ← Aacc	C_1aac
t96	MAR ← Pc	Lmar
t97	MBR ← ((MAR)), Inc Pc	Incpc, R/W, En, Lmbr
t98	Halt	
t99	Chargement de la RAM	Cmc

Tableau 4.10 : Etats, instructions et commandes

4.5 REPERTOIRE DES INSTRUCTIONS

Le jeu d'instructions du processeur de test développé est constitué de 63 instructions codées chacune sur 8 bits. Le répertoire d'instructions est défini suivant une étude basée sur la recherche des opérations indispensables au test et diagnostic des circuits et sur les mécanismes et procédures de compactions de l'information et particulièrement sur le traitement de séquences de données de longueurs très variées.

Les différentes instructions définies sous le format RTL et en liaison directe avec l'architecture conçue sont expliquées ci-dessous.

4.5.1 Instructions spécifiques au Boundary-Scan

Ces instructions sont définies à partir du graphe d'états du TAP qui contrôle l'architecture du standard BS 1149.1, pour faciliter la programmation et l'utilisation des différentes options offertes par la technique BS. Les instructions qui suivent nous permettent d'exploiter facilement le TAP : sélection du registre d'instruction ou registre de données, l'envoi des commandes et des données et la récupération des réponses de la chaîne de BS etc... Dans les paragraphes qui suivent on citera les 9 instructions définies et leurs micro-instructions associées.

4.5.1.1 Instruction « INITIALISE TAP du BOUNDARY-SCAN »

Ayant pour mnémonique : I_BS, cette instruction positionne le TAP des circuits sous test à l'état PAUSE_IR en suivant les états (ti) et les signaux de commandes appropriés.

t70 -.CMP ← (0010)h ; *Définir le nombre de décalage de RTMS.*
t79 -RTMS ← (02DF)h ; *Définir le signal de contrôle de TMS.*
t66 -Shift (RTMS), DEC CMP ; *Placer le TAP à L'état Pause-IR après initialisation.*
if ZCMP <> 0 then go to t66
else go to fetch

```
t70 - CMP ← (0010)h              ; Définir le nombre de décalage de RTMS.
t62 - Shift (RTMS), Shift(Aacc),  ; Envoyer le code Run_Bist au registre RI de BS à travers TDI.
   Shift(Bacc), DEC CMP
   if ZCMP <> 0 then go to t62
t71 - RTMS ← (0001)h             ; Définir le signal de contrôle de TMS.
t72 - CMP ← (002)h               ; Définir le nombre de décalage de RTMS.
t73 - Shift (RTMS), DEC CMP
   if ZCMP <> 0 then go to t73   ; Remettre le TAP à l'état Pause-IR.
   else go to fetch
```

4.5.1.5 Instruction Intest

Le code d'INTEST est envoyé au CUT pour effectuer le test interne de l'application.

```
t67 - CMP ← (0002)h              ; Définir le nombre de décalage de RTMS.
t68 - RTMS ← (0001)h             ; Définir le signal de contrôle de TMS.
t66 - Shift (RTMS), DEC CMP      ; Placer le TAP à l'état Schift-IR.
   if ZCMP <> 0 then go to t66
t74 - Aacc ← ((INTEST))
t70 - CMP ← (0010)h              ; Définir le nombre de décalage de RTMS.
t62 - Shift (RTMS), Shift(Aacc),  ; Envoyer le code Intest au registre RI de BS à travers TDI.
   Shift(Bacc), DEC CMP
   if ZCMP <> 0 then go to t62
t71 - RTMS ← (0001)h             ; Définir le signal da contrôle de TMS.
t72 - CMP ← (0002)h              ; Définir le nombre de décalage de RTMS.
t73 - Shift (RTMS), DEC CMP
   if ZCMP <> 0 then go to t73   ; Remettre le TAP à l'état Pause-IR.
   else go to fetch
```

4.5.1.6 Instruction Idcode_User code

Le code ID_CODE / USER _CODE est transmis au CUT pour lire le code d'identification du circuit.

```
t67 - CMP ← (0002)h              ; Définir le nombre de décalage de RTMS.
t68 - RTMS ← (0001)h             ; Définir le signal de contrôle de TMS.
t66 - Shift (RTMS), DEC CMP      ; Placer le TAP à l'état Schift-IR.
   if ZCMP <> 0 then go to t66
t75 - Aacc ← ((ID_CODE / USER_CODE))
t70 - CMP ← (0010)h              ; Définir le nombre de décalage de RTMS.
t62 - Shift (RTMS), Shift(Aacc),  ; Envoyer le code ID_CODE au registre RI de BS à travers TDI.
   Shift(Bacc), DEC CMP
   if ZCMP <> 0 then go to t62
t71 - RTMS ← (0001)h             ; Définir le signal de contrôle de TMS.
t72 - CMP ← (002)h               ; Définir le nombre de décalage de RTMS.
t73 - Shift (RTMS), DEC CMP,     ; Remettre le TAP à l'état Pause-IR.
   if ZCMP <> 0 then go to t73
   else go to fetch
```

```
t62 -Shift (RTMS), Shift (Aacc),          ; Envoyer les données sériellement à travers TDI et
    Shift(Bacc) DEC CMP, DEC BUF2         ; récupérer le contenu de la chaîne de BS.
    if ( ZCMP <> 0 AND Z<> 0 )
    then go to t62
t12 - MBR ← Bacc
t10 - MAR ← DI
t63 - MBR ←((MAR)), INC DI                ; Stocker les données de la chaîne BS.
    if ZCMP <> 0 then go to t7
t64 - RTMS ← (007D)h                      ; Définir le signal de contrôle de TMS.
t65 - CMP  ← (0007)h                      ; Définir le nombre de décalage de RTMS.
t66 - Shift (RTMS), DEC CMP               ; Remettre le TAP à l'état Pause-IR en envoyant la valeur
    if ZCMP <> 0 then go to t66           ; nécesaire à travers TMS.
    else go to fetch
```

4.5.1.9 Instruction Scan_DR_O NBR, ADD1

C'est la même instruction que SCAN_DR_IO sauf que pour SCAN_DR_O l'ancien contenu du registre DR sélectionné par l'instruction BS n'est pas sauvegardé en mémoire.

```
t54 - CMP ← ( 0006 )h                     ; Définir le nombre de décalage de RTMS.
t55 - RTMS ← (0017)h                      ; Définir le signal de contrôle de TMS.
t56 - Shift (RTMS), DEC CMP               ; Sélectionner le registre de donnée et mettre le TAP à l'état
    if ZCMP <> 0 then go to t56           ; Pause-DR.
t4  - MAR ← PC
t59 - MBR ← ((MAR)), INC PC
t46 - CMP ← NBR                           ; Définir le nombre de vecteur de test.
t8  - MAR ← PC
t57 - MBR ←((MAR)), INC PC
t25 - SI ← MBR                            ; Spécifier l'adresse de début de données.
t7  - MAR ← SI
t60 - MBR ←((MAR)), INC SI
t8 - Aacc ← MBR
t61 - buf2 ← (0010)h
t62 -Shift (RTMS), Shift (Aacc),          ; Envoyer la donnée sériellement à travers TDI et
    Shift(Bacc), DEC CMP, DEC BUF2        ; récupération de contenu de la chaîne de BS.
    if (ZCMP <> 0 AND Z <> 0)
    then go to t62
    if (ZCMP <> 0 AND Z = 0)
    then go to t7
t64 - RTMS ←(007D)h                       ; Définir le sigjal de contrôle de TMS.
t65 - CMP ← (0007)h                       ; Définir le nombre de décalage de RTMS.
t66 - Shift (RTMS), DEC CMP CMP           ; Remettre le TAP à l'état Pause-IR en envoyant la valeur
    if ZCMP <> 0 then go to t66           ; nécessaire à travers TMS.
    else go to fetch
```

4.5.2.6 Instruction LIS_MISR

Charge l'état initial dans le MISR à partir de Aacc.

t43 - IS_MISR ←Aacc

4.5.2.7 Instruction LPC_MISR

Charge le polynôme caractéristique dans MISR à partir de Bacc.

t44 - PC_MISR ←Bacc

4.5.2.8 Instruction CMPR Aacc

Elle est utilisée pour calculer la signature d'une séquence qui provient en mode série à chaque étape, 16 bits de cette séquence sont stockés dans Aacc, puis sont aiguillés vers le MISR. Cette opération se répète jusqu'à la fin de cette séquence.

t45 - MISR ←Aacc

4.5.2.9 Instruction Test_Mode_Normal_NBR

Cette instruction permet le calcul de la signature d'un ensemble de vecteurs de sortie provenant d'un CUT en mode de fonctionnement normal.

```
t4  - MAR  ←PC
t59 - MBR  ←((MAR)), INC PC
t46 - CMP  ←MBR          ; Définir le nombre de vecteurs à tester.
t42 - MISR ←DIN, DEC CMP ; Comprime les signaux de réponse de CUT en une signature.
    if ZCMP <> 0 then go to t42
    else go to fetch
```

4.5.2.10 Instruction BIST RAM

Elle permet de tester la RAM du processeur selon un algorithme spécial adapté pour le test de la RAM puis procède au calcul de la signature [11].

```
t27 - CMP ←(FFFF)h       ; Définir la taille de la RAM à tester.
t31 - DI  ←(2)h
t32 - Aacc←(5555)h       ; Définir le mot à stocker dans les cases mémoires d'adresse paire.
t33 - Baac←(AAAA)h       ; Définir le mot à stocker dans les cases mémoires d'adresse impaire.
t34 - SI  ←CMP
t7  - MAR ←SI
t11 - MBR ←Aacc
t35 - ((MAR)) ←MBR, DEC SI ; Mettre le mot 5555 dans les cases paires.
    if ZSI = 0 then go to t13
t37 - MAR ←SI
t36 - MBR ←Bacc, DEC SI
t13 -((MAR)) ← MBR       ; Mettre le mot AAAA dans les cases impaires.
    if ZSI <> 0 then go to t34
t86 - SI  ←CMP
t6  - MAR ←SI
```

t91 - DI ← SI

t92 - A1 = 0, GEN, DEC DI, TDO←TDO_PRPG ; *Génération des vecteurs à mettre dans le registre*
if ZDI <> 0 then go to t92 ; *interne .*

t93 - GEN, DOUT ←VT, MISR ← DIN, DEC CMP ; *Génération et compression des réponses*
if ZCMP <> 0 go to t91 ; *de CUT.*
else go to fetch.

4.5.2.15 Instruction SCAN_IN_Bacc

Elle permet de lire le contenu du registre scan de l'UUT et placer son contenu dans Bacc.

t70 - CMP ←(0010)h ; *Définir le nombre de décalage de RTMS.*
t94 -Shift Bacc ←TDI, DEC CMP ; *Lire le contenu de la chaîne BS par décalage bit par bit.*
if ZCMP <> 0 then go to t94
else go to fetch

4.5.3 Instructions d'entrée-sortie (E/S)

Deux instructions d'entrée-sortie sont utilisées pour une communication simple avec l'extérieur.

Instruction (E/S) LOAD Aacc, DIN : Elle lit le port d'entrée DIN et le charge dans Aacc.

t89 - Aacc ←DIN

Instruction (E/S) OUT_Aacc : Le contenu de Aacc est envoyé vers le port DOUT.

t95 - DOUT ←Aacc

4.5.4 Instruction de chargement LOAD

Ces instructions permettent un transfert simple de données ; l'opérande source est copié dans sa place de destination, sans qu'il soit modifié lui même. 11 combinaisons de transfert sont permises.

- LOAD Bacc, Aacc : Cette instruction place le contenu de Aacc dans Bacc. Elle s'exécute avec l'état t39.
- LOAD Aacc, ADD : Cette instruction charge le contenu d'une case mémoire (ADD) dans Aacc. Elle s'exécute avec l'enchaînement des états t39, t57, t20, t85 et t23.
- LOAD Bacc, ADD : Cette instruction charge le contenu d'une case mémoire (ADD) dans Bacc. Elle s'exécute avec les états t4, t57, t20, t85 et t24.
- LOAD DI, ADD : Cette instruction charge le contenu d'une case mémoire (ADD) dans DI. Elle s'exécute avec les états t4, t57, t20, t85 et t26.
- LOAD SI, ADD : Cette instruction charge le contenu d'une case mémoire (ADD) dans SI. Elle s'exécute avec les états t4, t57, t20, t85 et t25.
- LOAD Bacc, [SI] : Cette instruction charge le contenu d'une case mémoire adressée par SI dans Bacc ; l'adressage dans cette instruction est de type indexé. Elle s'exécute avec les états t7, t5 et t9.
- LOAD Aacc, [SI] : Cette instruction charge le contenu d'une case mémoire adressée par le contenu de SI dans Aacc ; l'adressage est de type indexé. Elle s'exécute avec les états t7, t5 et t8.

- JMP ADD : JMP est une instruction de saut systématique ; dans ce cas le PC est chargé par l'adresse ADD sans condition. Elle s'exécute avec les états t4, t5 et t6.

4.5.7 Instructions aritmétiques et logiques

Ces instructions sont necessaires au traitement de type comparaison, verification,... etc. Les différentes instructions de ce type sont définies ci-dessous :

- XOR Aacc, Bacc : Effectue (bit à bit) un "XOR" logique entre l'opérande destination (Aacc) et l'opérande source (Bacc). Le résultat sera placé dans (Aacc). (Les états t81, t82 et t83 sont utilisés).
- AND Aacc, Bacc : Effectue (bit à bit) un « AND » logique entre l'opérande destination (Aacc) et l'opérande source (Bacc), le résultat sera placé dans (Aacc). Ceci necessite t81, t87 et t83.
- NEG Aacc : NEG effectue l'inversion logique de chacun des bits de son opérande (Aacc). Cette instruction s'éxecute pendant les états t81, t88 et t83.
- SUB Aacc, Bacc : SUB fait la soustraction de Aacc et Bacc ; charge le résultat dans (buf2). Elle est définie par les deux états t81 et t84
- ADD Aacc, Bacc : ADD fait la somme de Aacc et Bacc ; charge le résultat dans (buf2). Elle est définie par les deux états t81 et t85
- COMPAR Aacc, Bacc : L'instruction "COMPAR" compare deux opérandes par soustraction et positionne l'indicateur Z (le flag Z) en fonction du résultat obtenu. Le résultat de la soustraction n'est pas sauvegardé. C'est la même instruction que SUB avec modification du bit Z.

4.5.7.1 Instructions DEC et INC

Les deux instructions DEC et INC sont utilisées pour l'incrémentation la décrémentation des trois compteurs de processeur (CMP, SI et DI) et nécessitent l'exécution d'un seul état. Par exemple pour décrémenter DI on utilise l'état t17 voir tableau 4.10.

t17 - DEC DI *; Soustraction d'un 1.*

4.5.7.2 Instruction HLT

Cette instruction permet l'arrêt de processeur lorsqu'elle est rencontrée.

t98- NOP
if Restart then go to t0
else t98

Dans le tableau 4.11 on présente les différentes instructions développées pour notre processeur de test à unité de contrôle câblée.

Chapitre 5

PROCESSEUR DE TEST A UNITE DE CONTROLE MICRO-PROGRAMMEE

5. PROCESSEUR DE TEST A UNITE DE CONTROLE MICRO-PROGRAMMEE

5.1 INTRODUCTION

La complexité des graphes d'états du contrôleur câblé du processeur de test défini au chapitre 4, nous a amené, à développer un contrôle micro-programmé ayant pour objectifs principaux : la production simple des micro-commandes des différents blocs du processeur, l'émulation de micro-instructions spécifiques au test, une assurance de synchronisation et particulièrement la possibilité de réaliser des micro-routines adaptables aux exigences et restrictions imposées par l'environnement de test (vu le manque de standard en dehors du BS).

5.2 ARCHITECTURE DU PROCESSEUR

Pour palier aux problèmes de vitesse de fonctionnement des techniques micro-programmées [75,76], nous avons adopté la structure à deux bus telle que le montre le synoptique de la figure 5.1. Les parties principales de ce processeur sont :

- Un générateur de vecteurs de test PRPG.
- Un analyseur de signatures à entrées multiples MISR.
- Une unité de contrôle.
- Une mémoire RAM pour stocker les programmes et les données.
- Deux « bus » assurent le cheminement de données entre les différents composants du processeur.
- Des registres spécifiques : RTMS, CX, ACC1, ACC2 et RI .
- Une table de données spécifiques au mécanisme Boundary Scan en mémoire ROM.
- Deux compteurs : COUNT et PC.

Dans les paragraphes ci-dessous on présente l'étude et le développement des différents modules de ce processeur de test.

5.2.1 Générateur de vecteurs de test

Ce générateur de type PRPG est à état initial et polynôme diviseur programmables. Les procédures de diagnostic de type : "tester-tirer certaines conclusions" au cours de test ont imposé l'addition d'un circuit de pondération comme le montre la figure 5.2. Les caractéristiques de ce PRPG sont :

- 16 bascules de type "D".
- Un registre de polynôme caractéristique "RCP" pour le chargement parallèle du polynôme caractéristique et son maintien durant les compressions.

- Un registre de commande "RC" qui contrôle l'activation de la logique de poids à travers les "MUX".
- Un registre de données "RD" pour la fixation de certaines broches de sorties à des valeurs prédéterminées.
- Un circuit de sélection de l'horloge à activer

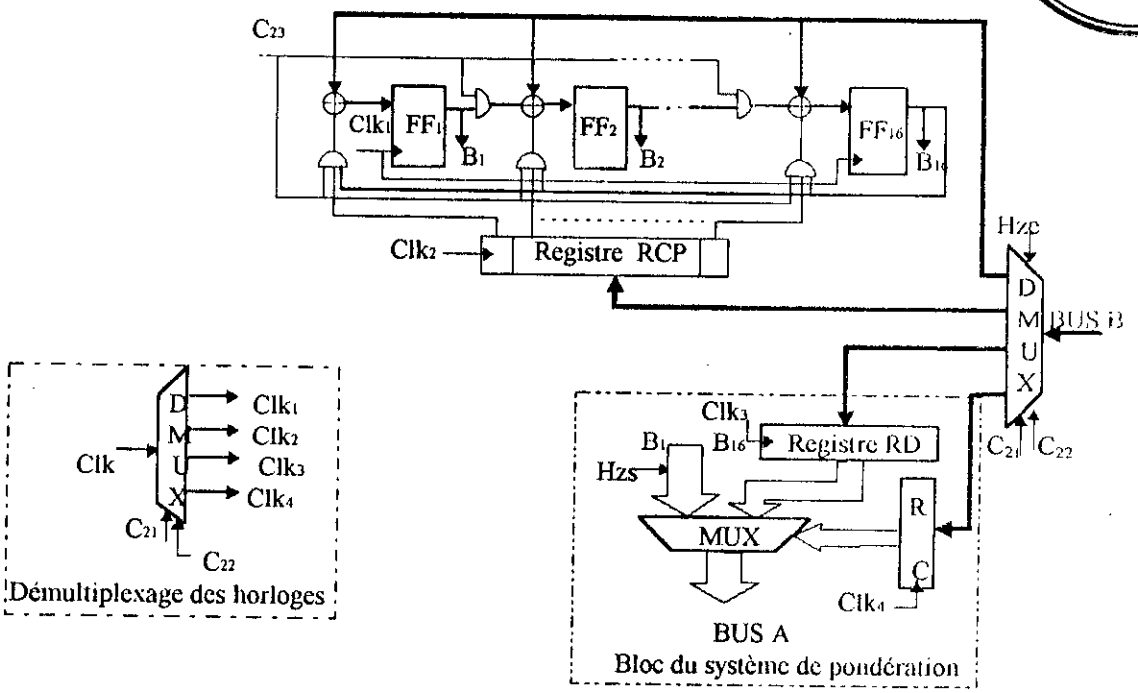


Figure 5.2 : Schéma synoptique du PRPG

Trois commandes sont nécessaires pour le fonctionnement du PRPG (tableau 5.1) :

HZe: haute impédance d'entrée.

HZe = /C_{23}

HZs: haute impédance de sortie.

HZs = /C_{21}+/C_{22} +/Hze

C_{21}	C_{22}	C_{23}	Opérations	HZe	HZs
0	0	0	NOP	1	1
0	0	1	Charger le registre RC	0	1
0	1	1	Charger le registre RD	0	1
1	0	1	Charger le registre RPC	0	1
1	1	1	Charger l'état initial	0	1
1	1	0	Génération	1	0
0	1	0	NOP	1	1
1	0	0	NOP	1	1

Tableau 5.1 : Mode de fonctionnement du PRPG

5.2.2 Analyseur de signatures à entrées multiples (MISR)

Pour tester un circuit à n entrés, on est obligé d'appliquer 2^n vecteurs de test, ce qui nous impose d'analyser 2^n réponses, donc un volume considérable d'information à stocker dans la mémoire (pour un circuit à 16 entrées et une sortie, on a 65536 bits à stocker). Le MISR permet la compression de ce volume en une signature à 16 bits. Il est conçu à base d'un LFSR à état initial et polynôme caractéristique programmables. Son schéma de principe est présenté sur la figure 5.3.

Les caractéristiques du MISR sont :

- 16 bascules de type « D ».
- Un registre RCP pour le chargement parallèle du polynôme caractéristique.

5.2.4 Les accumulateurs

Le fonctionnement et le rôle des deux accumulateurs ACC1 et ACC2 utilisés dans notre contrôleur sont expliqués ci-dessous. On note que ACC1 est dédié au transfert de données à travers TDI et ACC2 est dédié à la réception des données à travers TDO.

5.2.4.1 ACC1

C'est un registre de 16 bits, implanté certainement pour les traitements et sauvegarde de données partielles. Il assure une synchronisation échanges et facilite la programmation, ACC1 fonctionne comme l'indique la table 5.4.

$L = / C_{15} * C_{16}$
$S = C_{15} * / C_{16}$
$HZ = / C_{15} * / C_{16}$

C_{15}	C_{16}	OPERATIONS	L	S	HZ
0	0	NOP	0	0	1
0	1	CHARGEMENT	1	0	1
1	0	DECALAGE	0	1	1
1	1	DMUX ← ACC1	0	0	0

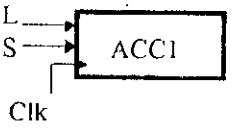


Tableau 5.4 : Commandes et opérations du ACC1

5.2.4.2 ACC2

Les mêmes caractéristiques et le même rôle que ACC1 sont attribués au registre ACC2. Les opérations et les commandes agissant sur lui sont indiquées sur la table 5.5.

$L = / C_{11} * C_{12}$
$S = C_{11} * / C_{12}$
$HZ = / C_{11} * / C_{12}$

C_{11}	C_{12}	OPERATIONS	L	S	HZ
0	0	NOP	0	0	1
0	1	CHARGEMENT	1	0	1
1	0	DECALAGE	0	1	1
1	1	DMUX ← ACC2	0	0	0

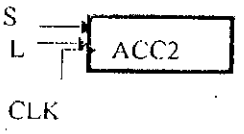


Tableau 5.5 : Commandes et opérations du ACC2

5.2.5 Les registres

Deux registres sont utilisés l'un pour les instructions du programme et l'autre pour commander la ligne TMS de BS.

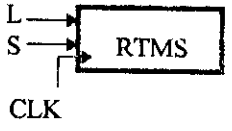
5.2.5.1 Registre d'instruction « RI »

Ce registre à 16 bits maintient la valeur du code objet de l'instruction pendant toute la durée de l'exécution. Sa commande de Chargement est C_8.

5.2.5.2 RTMS « *Register Test Mode Select* »

Ce registre de 16 bits assure la production du signal TMS pour le test des modules BS. La figure 5.4 montre leur commande et opération.

$L = C_{28}$
$S = C_{29}$
$C_{28} = 1$ CHARGEMENT
$C_{29} = 1$ DECALAGE



L: pour le chargement et S : pour le décalage à gauche.

Figure 5.4 : Commande et opération de RTMS

5.2.8 Autres unités de processeur

Le rôle et les caractéristiques des autres unités qui composent le processeur sont expliquées ci-dessus :

5.2.8.1 MAR (*Memory Address Register*)

C'est un registre à 16 bits qui assure la synchronisation entre les différents éléments de l'UC et la mémoire comme l'indique la table 5.9.

C_3	C_4	OPERATIONS
0	0	NOP
0	1	NOP
1	0	CHARGEMENT MAR ← PC
1	1	CHARGEMENT MAR ← BUSB

Tableau 5.9 : Commandes et opérations du MAR

5.2.8.2 MBR (Memory Buffer Register)

C'est un buffer de données à 16 bits qui assure la synchronisation du transfert des données entre la mémoire et les autres unités du processeur.

5.2.8.3 RAM

Cette mémoire est utilisée pour sauvegarder les programmes de test et de diagnostic. Elle peut être externe ou interne. Quand elle est externe, le bus de données et d'adresses permettent son chargement.

C_5	C_6	C_7	OPERATIONS	L	HZs
0	0	1	MBR ← BUS B	1	1
1	1	1	MBR ← ((MAR))	1	1
0	1	0	((MAR)) ← MBR	0	0
1	0	0	BUS B ← MBR	0	0
0	0	0	NOP	0	1
1	1	0	NOP	0	1
1	0	1	NOP	0	1
0	1	1	NOP	0	1

Tableau 5.10 : Opérations et commandes du MAR

5.2.8.4 Liaison du Bus

Le besoin d'accélérer le transfert des données entre les différentes unités du processeur a imposé la nécessité d'utiliser les deux bus A et B. La commande C9 assure la connexion.

C_9=1 : Bus B ← Bus A

5.2.8.5 Port de sortie

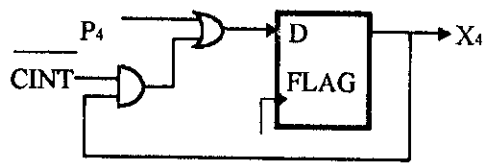
Notre processeur est doté d'un port de sortie qui permet au programmeur d'analyser les résultats de test (16 bits type collecteur ouvert). L'opération est comme suit :

C_{37}=1: PORTs ← BUS B.

On peut aussi connecter une mémoire extérieure au port de sortie munie d'un registre d'adressage RA, chargé initialement par la première adresse où on doit transférer les données. Ce transfert se fait comme suit : La donnée courante est au port de sortie, une commande additionnelle P_5 sert à incrémenter le RA chaque fois qu'on veut transmettre une autre donnée.

5.2.8.6 Mode interruption (Port d'entrée)

Le port d'entrée est utilisé pour introduire des programmes de test dans la RAM et permettre d'apporter des modifications sur les données stockées. Cette modification peut prendre la forme d'une interruption, pour cela il faut agir matériellement sur la commande externe CINT. Une fois CINT est activée comme l'indique la figure 5.6, le FLAG se met à zéro, donc un mode d'interruption sera exécuté. Ce transfert se fait d'une manière simihaire au cas précèdent, seulement la première donnée récupérée au port d'entrée doit obligatoirement représenter le nombre de mots à transférer et la deuxième pointe vers la première adresse où on doit stocker ce bloc de données. Une fois l'interruption est achevée, il faut activer le FLAG à l'aide d'une commande interne P_4.



C_{38}	C_{39}	OPERATIONS	HZ
0	0	NOP	1
0	1	P_5 =1	1
1	0	P_4 =1	1
1	1	BUS B ← PORTe	0

Figure 5.6 : Fonctionnement du mode interruption

5.2.9 Positionnement du circuit sous test « CUT »

Pour contrôler l'acheminement des données du CUT vers le processeur il faut prévoir une commande de contrôle de haute impédance HZ_1 et en plus une autre HZ_2 pour isoler les portes ET ; comme l'illustre la figure 5.7. Les opérations réalisées sur le CUT sont indiquées dans la table 5.11.

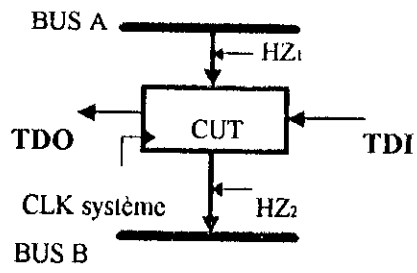


Figure 5.7 : Les commandes agissent sur le CUT

$HZ_1 = \overline{C_{13} * C_{14}}$
$HZ_2 = \overline{C_{13} \oplus C_{14}}$
$P_1 = /C_{13} * C_{14}$
$P_2 = C_{13} \oplus C_{14}$
$P_3 = C_{13} * /C_{14}$

C_{13}	C_{14}	OPERATIONS	P_1	P_2	P_3	HZ_1	HZ2
0	0	NOP	0	0	0	1	1
0	1	CUT ← ACC2, ACC1 ← CUT	1	1	0	1	0
1	0	CUT ← PRPG, ACC1 ← CUT	0	1	1	1	0
1	1	CUT ← BUS A, BUS B ← CUT	0	0	0	0	1

Tableau 5.11 : Opérations et actions sur le CUT

5.2.10 Unité de contrôle

Cette unité sert à générer les commandes nécessaires pour l'exécution de toutes les micro-instructions. L'unité micro-programmée développée est donnée sur la figure 5.8.

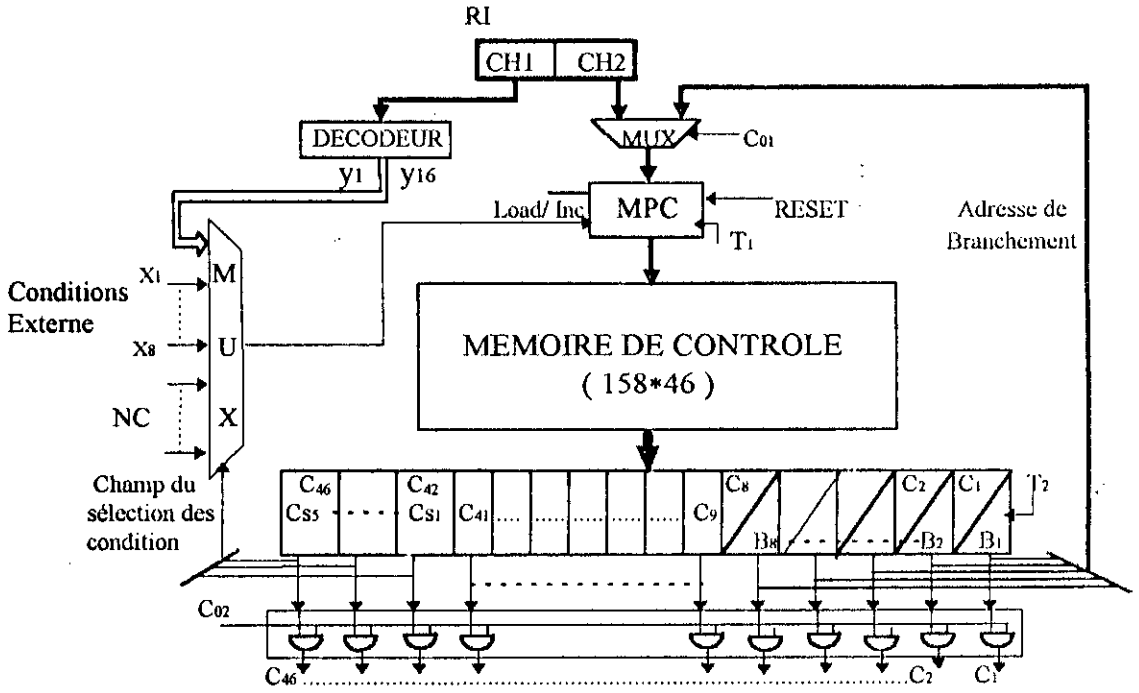


Figure 5.8 : Schéma synoptique de l'unité de contrôle

Cette unité de contrôle se compose des éléments suivants :

- Une ROM de 46 bits *158 micro-instructions.
- Un MUX de 32 entrées commandé par 5 signaux de sélection.
- Un MPC (compteur de microprogrammation) qui pointe vers l'adresse de la micro-instruction à exécuter.
- Un buffer de données CMDB pour la synchronisation et le maintien des commandes et codes.

Le programme résident en mémoire, les adresse et les signaux à générer en hexadécimale sont représentés dans l'ANNEXE 1.

5.2.11 Mode de fonctionnement

L'unité de contrôle opère suivant deux modes définis selon le contenu du champ COND-SELECT comme le montre à la table 5.12.

5.2.12 Mode opération

Si un mode normal est sélectionné alors le contenu du CMDB est une micro-instruction à exécuter.

5.2.13 Mode de Branchement

C'est un saut ordinaire pour le contrôle du séquencement des différentes micro-opérations représentant les étapes d'une instruction. Cette conception est adaptée à la minimisation de la taille de la ROM.

TYPE DE BRANCHEMENT	MODE DE FONCTIONNEMENT	CODE OBJET CH2					CONDITION DE SELECTION				
		H_1	H_2	H_3	H_4	H_5	C_{40}	C_{41}	C_{42}	C_{43}	C_{44}
incrementation du MPC	X1= 0 mode opération	x	x	x	x	x	0	0	0	0	0
branchement interne	X2=1 saut inconditionnel	x	x	x	x	x	0	0	0	0	1
branchement externe CH1	X3=1 saut inconditionnel	x	x	x	x	x	0	0	0	1	0
interruption FLAG=0	X4=1 saut conditionnel	x	x	x	x	x	0	0	0	1	1
JZACC1 ACC1<>0	X5=1 saut conditionnel	x	x	x	x	x	0	0	1	0	0
JNZACC1 ACC1=0	X6=1 saut conditionnel	x	x	x	x	x	0	0	1	0	1
COUN<>0	X7=1 saut conditionnel	x	x	x	x	x	0	0	1	1	1
LACC1 ADD	Y1=1 saut conditionnel	0	1	1	1	1	0	1	0	0	0
LACC2 ADD	Y2=1 saut conditionnel	1	0	0	0	0	0	1	0	0	1
LCOUN ADD	Y3=1 saut conditionnel	1	0	0	0	1	0	1	0	1	0
LISP ADD	Y4=1 saut conditionnel	1	0	0	1	0	0	1	0	1	1
LISM ADD	Y5=1 saut conditionnel	1	0	0	1	1	0	1	1	0	0
LCPP ADD	Y6=1 saut conditionnel	1	0	1	0	0	0	1	1	0	1
LCPM ADD	Y7=1 saut conditionnel	1	0	1	0	1	0	1	1	1	0
LRC ADD	Y8=1 saut conditionnel	1	0	1	1	0	0	1	1	1	1
LRD ADD	Y9=1 saut conditionnel	1	0	1	1	1	1	0	0	0	0
READ ADD	Y10=1 saut conditionnel	1	1	0	0	0	1	0	0	0	1
TEXH ADD	Y11=1 saut conditionnel	1	1	0	0	1	1	0	0	1	0
TEPR ADD	Y12=1 saut conditionnel	1	1	0	1	0	1	0	0	1	1
BIST ADD	Y13=1 saut conditionnel	1	1	0	1	1	1	0	1	0	0
SCAN ADD	Y14=1 saut conditionnel	1	1	1	0	0	1	0	1	0	1
INDAT ADD	Y15=1 saut conditionnel	1	1	1	0	1	1	0	1	1	0
LCX ADD	Y16=1 saut conditionnel	1	1	1	1	0	1	1	0	0	0
CX=0	X8=1 saut conditionnel	x	x	x	x	x	1	0	1	1	1

Tableau 5.12 : Les conditions de branchement

L'exécution d'une micro-instruction se déroule sur trois phases :

1. L'adresse de la micro-instruction doit être chargée dans le MPC pendant une fenêtre T1.
2. Lecture du contenu de cette adresse (CMDB ← ((MPC))) pendant une fenêtre T2.
3. l'exécution se fait pendant une autre fenêtre temporelle T3.

Trois signaux de séquencement assurent le fonctionnement de l'UC. Le circuit séquentiel simple de la figure 5.9 est utilisé pour la production des fenêtres temporelles T_1, T_2 et T_3.

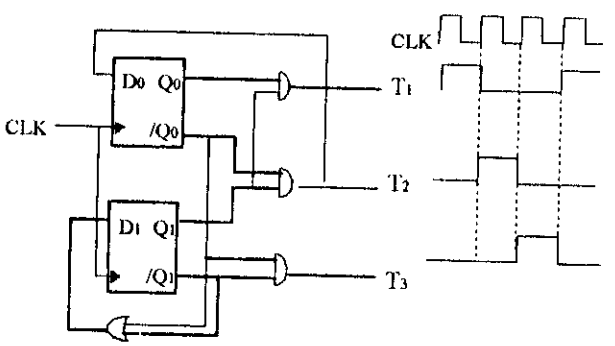


Figure 5.9 : Schéma synoptique du séquenceur

5.2.14 Unité de contrôle à deux ROM

Notre unité de contrôle définie précédemment exige un grand espace mémoire. Pour y remédier, on est amené à utiliser le concept d'unité de contrôle à deux mémoires [76] dont le schéma de principe est indiqué sur la figure 5.10.

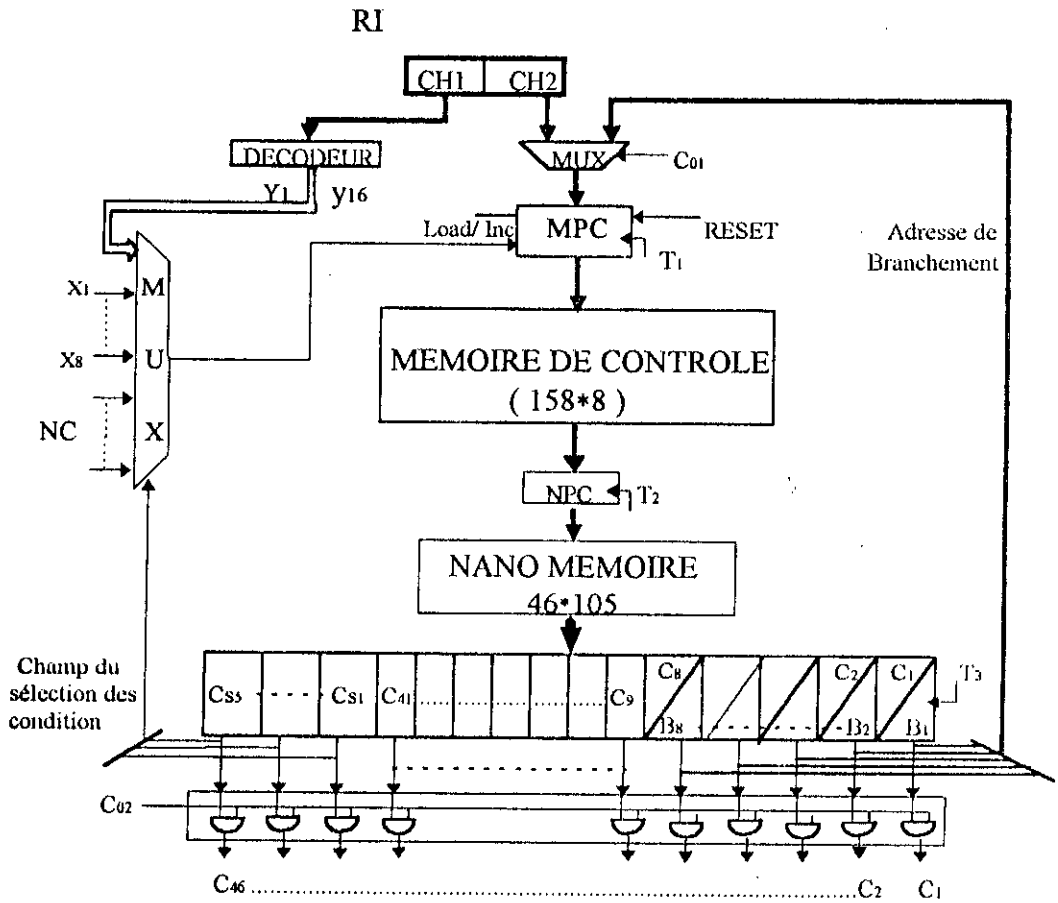


Figure 5.10 : Schéma de principe de l'unité de contrôle à deux mémoires

Cette technique nous permet de gagner 160 octets d'espace mémoire, mais insérer une autre mémoire dans l'unité de contrôle implique des cycles fetch en plus, donc du temps additionnel à perdre, ce qui la rend très lente et donc non souhaitable pour le test.

Le programme qui réside dans la mémoire secondaire « Nano-Memoire » est donné en ANNEXE 2.

5.3 JEU D'INSTRUCTIONS

Comme tous les processeurs du marché, notre modèle est doté d'un jeu d'une cinquantaine d'instructions où chaque instruction individuelle est codée sur 16 bits en deux champs CH1 et CH2 d'un octet chacun :

- **CH1** : Pointe vers la première micro-instruction à exécuter.
- **CH2** : Représente une condition de sélection pour les cas de saut. (Voir tableau 5.13).

Les instructions sont rangées en groupes comme suit :

5.3.1 Instructions de Chargement

Pour que le processeur fonctionne correctement et afin de favoriser les opérations de test, le besoin de charger les différentes unités du processeur étant vital, on a développé les instructions suivantes :

- LCX ADD : le contenu de la case mémoire ADD est transféré dans le registre CX.
- LACC1 ADD : transfère le contenu de la case mémoire ADD dans l'accumulateur ACC1.
- LACC2 ADD : transfère le contenu de la case mémoire ADD dans l'accumulateur ACC2.
- LCOUNT ADD : charge le contenu de la case mémoire ADD dans le compteur COUNT.
- LISP ADD : sert à charger l'état initial du PRPG en transférant le contenu de la case mémoire ADD dans LFSR du PRPG.
- LCPP ADD : chargement du polynôme caractéristique du PRPG, le contenu mémoire en ADD est envoyé au registre RCP du PRPG.
- LRC ADD : charger le registre de commande de la logique de poids dans le PRPG à partir de la case mémoire ADD.
- LRD ADD : charge le registre de données de la logique de poids dans le PRPG avec le contenu de la case mémoire ADD.
- LISM ADD : le contenu de la case mémoire ADD est transféré vers le LFSR du MISR, c'est un chargement d'état initial.
- LCPM ADD : elle charge le polynôme caractéristique dans le registre RCP du MISR.

En plus du mode d'adressage direct utilisé pour les instructions précédentes, un mode immédiat est aussi défini pour les instructions suivantes : LACC1 OPR, LACC2 OPR, LCOUNT OPR, LISP OPR, LISM OPR, LCPP OPR, LCPM OPR, LRC OPR, LRD OPR et LCX OPR.

5.3.2 Instructions de Stockage

Deux instructions de stockage sont incluses dans ce groupe. STACC1 ADD : transfère ACC1 dans la case mémoire ADD. STACC2 ADD : l'exécution de cette instruction stocke le contenu de l'accumulateur ACC2 dans la case mémoire ADD.

5.3.3 Instructions arithmétiques et logiques

Pour fixer des limites, réaliser des décisions sur les valeurs arithmétiques et logiques et surtout permettre une coordination structurée des mécanismes de test (masquage et couverture de fautes), une UAL est exigée. Notre processeur a 7 opérations arithmétiques et logiques :

- ADD : additionne les contenus de ACC1 et ACC2 et place le résultat dans ACC1.
- SUB : soustraction du contenu ACC2 à partir de ACC1 et chargement du résultat dans ACC1.
- AND : c'est un ET logique bit à bit des contenus ACC1 et ACC2 ; le résultat est placé dans ACC1.
- CMP : On compare ACC1 et ACC2 et on modifie les drapeaux (Flag) concernés.

- XOR : c'est le OU-EXCLUSIF de ACC1 avec ACC2, ACC1 contient le résultat de l'opération.
- NOT : Complémentation du contenu de ACC1.
- NEG : Complémentation à deux de ACC1.

5.3.4 Instructions de Sauts

- JMP OPR : l'exécution de cette instruction produit un saut inconditionnel et le compteur programme pointe vers l'adresse de branchement OPR.
- JZACC1 OPR : effectue le branchement vers l'adresse OPR si le contenu de ACC1=0, c'est un saut conditionnel.
- JNZACC1 OPR : Branchement vers l'adresse OPR si le contenu de ACC1 est non nul.
- LOOP OPR : Branchement vers l'adresse OPR si le contenu de CX est non nul, elle exige un chargement au préalable de CX avec le nombre de répétitions, c'est une instruction de boucle.

5.3.5 Instructions de contrôle de sorties

1. REACC1 : envoie le contenu de ACC1 vers le port de sortie.
2. REACC2 : envoie le contenu de ACC2 vers le port de sortie.
3. READ OPR : Cette instruction envoie le contenu d'un bloc mémoire vers le port de sortie à partir de l'adresse OPR ; le nombre de mots à transférer est chargé initialement dans CX.

5.3.6 Instructions de test

Comme ce processeur est destiné au test des circuits numériques, plusieurs instructions spécifiques de test subdivisées en plusieurs groupes sont introduites :

5.3.6.1 Instructions de test ordinaire

Le test ordinaire est défini par l'envoi de vecteurs de test et récupération de réponses. Quatre instructions (TEPR OPR, TEXH OPR, TEPR ADD et TEXH ADD) permettent ce type de test.

5.3.6.2 TEXH OPR

L'exécution de TEXH permet de générer un nombre OPR de vecteurs de test et de récupérer le résultat de compression dans ACC1. Le compteur COUNT sert aussi pour la génération des vecteurs de test à appliquer au CUT ; initialement COUNT est chargé par la valeur OPR et à chaque fois qu'on veut générer un vecteur de test COUNT est décrémenté si COUNT est non nul. On génère ainsi tous les vecteurs de test allant de 1 a OPR. En conséquence on a conçu une instruction qui fait un test exhaustif, en terme de micro-instructions TEXH se résume comme suit :

1. BusB ← MBR COUNT ← BusB.
2. DEC COUNT ; BusA ← COUNT BusB ← CUT ; MISR ← BusB.
3. IF COUNT ≠ 0 THEN goto 2.

4. BusA ← MISR ; BusB ← BusA ; ACC1 ← BusB.
5. go to FETCH.

5.3.6.3 TEPR OPR

Cette instruction lance une opération de test pseudo-aléatoire ; initialement COUNT pointe vers le nombre OPR de vecteurs de test à générer par PRPG et à chaque fois qu'on veut générer un autre vecteur de test et sachant que COUNT est non nul, on décrémentera COUNT. Le PRPG générera ce vecteur et l'envoie au CUT ; simultanément le MISR reçoit la réponse. A la fin de l'opération de test, le résultat de la compression est dans ACC1. En langage RTL ceci se résume comme suit :

1. BusB ← MBR; COUNT ← BusB
2. DEC COUNT, BusA ← PRPG, CUT ← BusA, BusB ← CUT, MISR ← BusB.
3. IF COUNT ≠ 0 THEN goto 2
4. BusA ← MISR, BusB ← BusA, ACC1 ← BusB.
5. go to fetch.

Noter que TEPR ADD et TEXH ADD ne diffèrent des deux autres que par le mode d'adressage utilisé. Dans ce cas ADD représente l'adresse de la case mémoire où se trouve le nombre de vecteurs de test à générer.

5.3.6.4 Instructions d'auto test (BIST OPR, BIST ADD)

Pour une assurance de fonctionnalité et analyse des tâches du processeur, un contrôle additif des éléments et unités du processeur tels que le PRPG et le MISR s'impose. L'instruction BIST offre cet avantage.

BIST OPR, BIST ADD : un nombre OPR (ADD l'adresse de la case mémoire où se trouve ce nombre) de vecteurs de test sont générés par le PRPG et ensuite envoyés au MISR. Le résultat final de la compression est dans ACC1.

5.3.6.5 Instructions de test de type SCAN

Certains CUT configurés selon la stratégie de test SCAN exigent un soin particulier, c.a.d l'envoi de données de test seriellement via le pin TDI et récupération de réponses aussi seriellement via TDO. Les deux instructions suivantes : SCAN OPR, SCAN ADD peuvent accomplir ce rôle. L'exécution de l'instruction SCAN OPR permet de transférer la donnée OPR qui est dans ACC2 vers le CUT via le pin TDI et reçoit la réponse dans ACC1 via le pin TDO. L'instruction SCAN OPR na diffère de SCAN ADD que par le mode d'adressage, ADD pointe vers l'adresse de la case mémoire où se trouve la donnée à transférer.

5.3.6.6 Instructions BOUNDARY-SCAN

Suivant le standard BS 1149.1 et l'organigramme du TAP, on a constaté qu'il faut concevoir les instructions BS suivantes :

5.3.6.6.1 PMTB

Cette instruction est utilisée pour initialiser le TAP du CUT en le mettant à l'état PAUSE-IR du diagramme de transition. En terme de micro-instructions, PMTB se résume à :

1. COUNT ← 000B
2. RTMS ← FB40
3. DEC COUNT RTMS (←)
4. IF COUNT ≠ 0 THEN goto 3
5. go to FETCH.

5.3.6.6.2 INDAT OPR

Elle transfère seriellement la donnée OPR (mot) vers le CUT via TDI et par l'intermédiaire des décalages à gauche de ACC2 et reçoit les réponses par décalage à gauche de ACC1 et via TDO pour les comprimer dans le MISR. La réponse finale est dans ACC1 alors que le résultat de compression du mot précédent est dans ACC2. Les évolutions dans le diagramme de transition du TAP se font comme suit :

Si OPR est un code d'instruction, le TAP est à l'état PAUSE-IR, il évolue vers EXIT2-IR puis SHIFT-IR et y restera pendant 16 tops d'horloges, temps suffisant pour envoyer un mot de 16 bits et revenir vers PAUSE-IR en passant par EXIT1-IR.

Si OPR est une donnée de traitement, le TAP est à l'état PAUSE-DR, alors il évolue vers EXIT2-DR puis SHIFT-DR et restera aussi dans cet état pendant 16 tops d'horloges, et ensuite reviendra à PAUSE-DR en passant par EXIT1-DR.

En terme de micro-instructions INDAT est décrite comme suit :

1. BusB ← MBR ; ACC2 ← BusB.
2. COUNT← 0002.
3. RTMS ← 8000.
4. DEC COUNT RTMS(←).
5. IF COUNT ≠ 0 THEN goto 4.
6. COUNT ← 0010.
7. RTMS ← 0.
8. DEC COUNT, RTMS(←), CUT←ACC2(←), ACC1(←)←CUT, MISR←ACC1(←).
9. IF COUNT ≠ 0 THEN go to 8.
10. COUNT ← 0002.
11. RTMS ← 8000.
12. DEC COUNT, RTMS(←).
13. IF COUNT ≠ 0 THEN goto 12.
14. BusA ← MISR, BusB ← BusA ACC2 ← BusB.
15. go to FETCH.

Dans certains cas, OPR est un code d'une instruction standard (BYPASS, EXTEST, INTEST, IDCODE, USERCODE, SAMPLE et RUNBIST) et pour minimiser le temps d'accès mémoire on a

introduit le concept de la table de données où des codes spécifiques sont stockés. On a utilisé le mode d'adressage direct : INDAT ADD

Notant que les instructions : BYPASS, INTEST, EXTEST, RUNBIST, IDCODE, USERCODE, SAMPLE, SCANBS et INDAT ADD suivent la même description en terme de micro-instructions que INDAT OPR.

Dans le tableau 5.13 on présente toutes les instructions utilisées dans le jeu d'instructions de notre processeur à unité de contrôle micro-programmée.

N°	INSTRUCTION	CH1	CH2	N°	INSTRUCTION	CH1	CH2
1.	LACC1 OPR	19	00	31.	LACC1 ADD	07	0F
2.	LACC2 OPR	1B	00	32.	LACC2 ADD	07	10
3.	LCOUNT OPR	1D	00	33.	LCOUNT ADD	07	11
4.	LCX OPR	9A	00	34.	LCX ADD	07	1E
5.	LISP OPR	1F	00	35.	LISP ADD	07	12
6.	LISM OPR	21	00	36.	LISM ADD	07	13
7.	LCPP OPR	23	00	37.	LCPP ADD	07	14
8.	LCPM OPR	25	00	38.	LCPM ADD	07	15
9.	LRC OPR	27	00	39.	LRC ADD	07	16
10.	LRD OPR	29	00	40.	LRD ADD	07	17
11.	TEXH OPR	2B	00	41.	TEXH ADD	07	19
12.	TEPR OPR	2F	00	42.	TEPR ADD	07	1A
13.	BIST OPR	33	00	43.	BIST ADD	07	1B
14.	SCAN OPR	38	00	44.	SCAN ADD	07	1C
15.	INDAT OPR	64	00	45.	INDAT ADD	07	1D
16.	BYPASS	66	00	46.	INTEST	68	00
17.	EXTEST	6A	00	47.	SAMPLE	6C	00
18.	RUNBIST	6E	00	48.	IDCODE	70	00
19.	USERCODE	72	00	49.	SCANBS	74	00
20.	PMTBS	5E	00	50.	UPDATE	61	00
21.	REACC1	5A	00	51.	REACC2	5C	00
22.	READ OPR	07	18	52.	STACC1 OPR	53	00
23.	STACC2 OPR	56	00	53.	JMP OPR	40	00
24.	JNZACC1 OPR	42	00	54.	JZACC1 OPR	3F	00
25.	LOOP OPR	9C	00	55.	HALT	83	00
26.	ADD	4E	00	56.	SUB	50	00
27.	AND	44	00	57.	CMP	46	00
28.	NEG	48	00	58.	NOT	4A	00
29.	XOR	4C	00	59.	READB OPR	92	00
30.	LRTMS OPR	29	00	60.	LRTMS ADD	07	00

Tableau 5.13 : Répertoire d'instructions du PTDC

5.3.6.6.3 UPDATE

Cette instruction sert à la mise à jour de la donnée où de l'instruction transférée par INDAT en mettant le TAP à l'état PAUSE-DR ; l'évolution se fait soit de l'état PAUSE-IR à l'état PAUSE-DR si UPDATE est précédée d'une instruction INDAT où OPR représente un code d'une instruction, ou bien de PAUSE-DR vers elle même si OPR est une donnée normale. UPDATE se compose d'un ensemble de micro-instructions :

1. COUNT ← 0006.
2. RTMS ← E800.
3. DEC COUNT RTMS(←).
4. IF COUNT ≠ 0 THEN go to 3.
5. go to fetch.

5.4 CONCLUSION

Le PTDC à unité de contrôle micro-programmée posside 60 instructions codées sur 16 bits dont 32 sont de type ''test''.

L'unité de contrôle de ce PTDC génère 46 fonctions de contrôle C_1-C_{46}, analyse 8 conditions externes résultant du traitement en cours et 17 conditions imposées par le type d'instructions (branchement conditionnel). Les dimensions de la mémoire de contrôle sont 158*46 bits.

Un gain de 160*8 bits et une simplification de la traduction des mots de contrôle en mémoire sont les résultats de la conception proposée pour l'UC à 2 mémoires (*Nanomemory*) aux dépends d'un cycle additionnel d'horloge à chaque micro-instruction.

Chapitre 6

VALIDATION DU PTDC PAR PALASM

1. CIRCUITS PROGRAMMABLES

2. LOGICIEL DE VALIDATION « PALASM »

3. VALIDATION DU PROCESSEUR DE TEST PAR PALASM

4. APPLICATIONS

5. COMPARAISON.

6. VALIDATION DU PTDC PAR PALASM

6.1 INTRODUCTION

Dans ce travail, nous avons utilisé le logiciel CAO PALASM (*Advanced Micro Devices 1992*) pour la validation fonctionnelle du circuit du processeur de test développé en conformité avec le cahier des charges. La validation à l'aide des circuits logiques programmables (*PLD, FPGA, PAL,...*) est devenue ces dernières années une méthode privilégiée car elle est rapide et non onéreuse. Elle s'articule particulièrement, sur l'implantation d'expressions booléennes ou fonctions séquentielles qui utilisent les structures logiques génératrices de circuits VLSI qui accomplissent des tâches diverses et parfois même spécifiques. Le nombre de compagnies qui utilise cette méthode de conception ne cesse d'augmenter (*Altera, AMD,...*) et des standarisations sont déjà nées [77].

Le logiciel PALASM permet non seulement de produire des programmes de simulation, de vérification de signaux, optimisation et la sélection des circuits PAL mais aussi de produire l'interface pour la fusion réelle des fusibles qui composent les PAL d'une application donnée. En effet, ce logiciel génère le fichier format "*JEDEC*" pour programmer le PAL et effectuer la simulation électrique.

6.2 CIRCUITS PROGRAMMABLES

La réalisation pratique d'un système logique se distingue par la sélection de composants disponibles sur le marché. Le concepteur doit décomposer un système donné en blocs fonctionnels, et ensuite optimiser les fonctions utilisées suivant des critères de coût, performances, complexité de connexions...etc. L'utilisation de circuits dont la fonction est programmable par l'utilisateur représente une solution élégante.

6.2.1 Circuits combinatoires programmables

La plupart de ces composants sont constitués d'une matrice d'opérateur ET qui génère les produits des variables d'entrées et de leur complément et d'une matrice d'opérateurs OU qui effectue les produits. Selon le type du circuit, l'une ou l'autre ou les deux matrices sont programmables. Dans la figure 6.1 on donne le schéma synoptique des PALs [77].

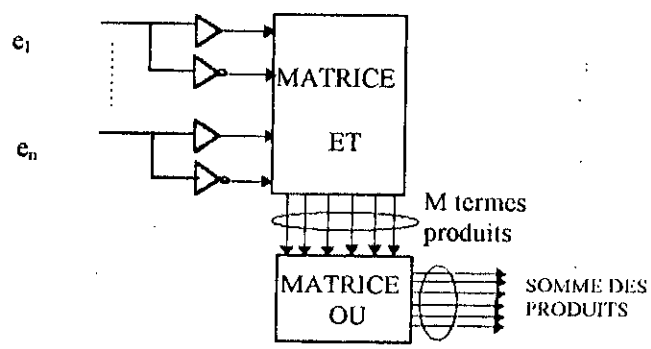


Figure 6.1 : Schéma synoptique d'un PAL

La matrice ET est réalisée à base de portes ET à diodes, la programmation s'effectue grâce à des fusibles placés en série avec ces diodes. La figure 6.2 représente un exemple de fonction logique réalisée avec un PAL.

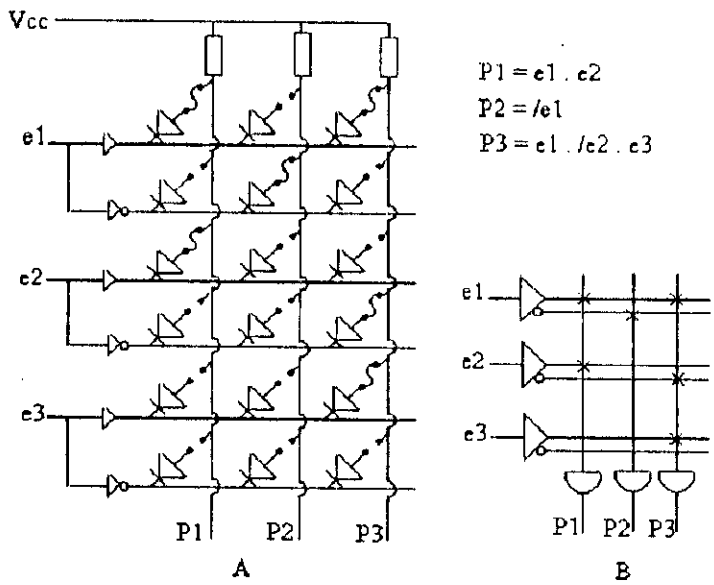


Figure 6.2 : Exemple de PAL. A : Circuit électrique. B : Représentation.

6.2.2 Circuits séquentiels programmables

Dans ce cas, en plus des deux matrices (ET, OU) on trouve à la sortie de ces PALs des bascules dont la fonctionnalité est programmable [77]. La figure 6.3 illustre une sortie séquentielle d'un PAL. La sortie peut éventuellement avoir un circuit à trois états contrôlée par une validation (*Output Enable*).

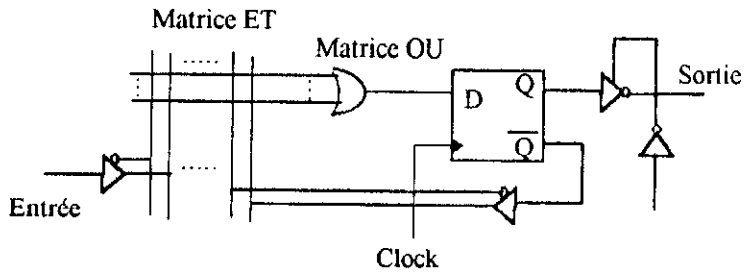


Figure 6.3 : Principe d'un PAL à sortie séquentielle

6.3 LOGICIEL DE VALIDATION « PALASM »

PALASM est un outil CAO développé par la société *Advanced Micro Devices*, Inc, en 1992. Il génère les fichiers JEDEC, HEX ou BINAIRE et ceci à partir de spécifications logiques introduites par le concepteur (équations logiques, équations de transition et équations de sortie). Ce fichier *JEDEC* est utilisé pour programmer le PAL en utilisant un programmateur similaire à celui utilisé pour les mémoires. Le résultat de la simulation est donné sous forme de chronogrammes ou binaire.

Pour programmer un PAL par PALASM il faut suivre les étapes suivantes :

1. Analyser et résoudre le problème posé en faisant ressortir les fonctions combinatoires, séquentielles ou par une combinaison des deux.

2. Déterminer le type du PAL à utiliser en fonction de la solution.
3. Créer le fichier PALASM et programmer la solution du problème en respectant l'organisation de PALASM.
4. Compiler le fichier jusqu'à l'obtention d'un fichier exempt d'erreurs.
5. Simuler le comportement du circuit PAL et s'assurer ainsi du bien fondé de la solution réalisée par rapport au problème posé.
6. Programmer le PAL au moyen d'un programmeur de PALs et ensuite réaliser des tests pour valider le prototype en cours d'étude.

6.3.1 Structure d'un programme PALASM

Un fichier (programme) PALASM est organisé généralement en trois segments [77]:

1. Un segment déclaration.
2. Un segment équation logique.
3. Un segment simulation.

6.3.1.1 Système combinatoire

Pour un système combinatoire la syntaxe d'un programme PALASM est la suivante :

```
TITLE : unité de contrôle                    ; Donner le titre du fichier.
PATTERN : A
REVISION : 5                                 ; Donner la version.
AUTOR :                                      ; Spécifier l'auteur.
COMPANY :
DATE :
CHIP  PAL 22L10

;_____ Segment déclaration _____

PIN 1X combinatorial ;input                  ; Après le mot réservé PIN on donne le numéro de PIN et son
                                             nom ainsi que son type (combinatorial ou registered) pour Vcc et
                                             GND pas de type de PIN à fournir.
PIN 2   Y combinatorial ; output
PIN 3   Z combinatorial ; input

;_____Segment équations booléennes _____

EQUATION                                     ; mot réservé.
/Y = X * Z                                   ; Donner l'équation logique de la sortie Y en fonction des entrées
                                             X et Z.

;_____segment de simulation _____

SIMULATION                                   ; mot réservé.
TRACE_ON X Y Z                               ; spécifie les signaux à tracer.
SETF  /X Z                                   ; donne les valeurs X=0 et Z=1.
CHECK  /Y                                    ; vérifier que Y=0.
TRACE_OFF                                    ; mot réservé.
```

6.3.1.2 Système séquentiel

Pour un système séquentiel la structure du programme devient :

```
;_____Segment déclaration _____
TITLE : unité de contrôle                ; Donner le titre du fichier.
PATTERN : A
REVISION : 5                             ; Donner la version.
AUTOR :                                  ; Spécifier l'auteur.
COMPANY :
DATE :
CHIP  PAL 22L10

;_____Déclaration de PINs _____
PIN 12  GND
PIN 24  Vcc

;_____ Segment de déclaration des états _____
STATE                                    ; mot réservé.
MEALY_MACHINE                            ; mot réservé pour spécifier qu'il s'agit de machine de Mealy
                                         sinon spécifier que la machine est de type Moore.
START_UP :=POWER_UP→ STATE0              ; l'état STATE0 est l'état initial à la mise sous tension.
;_____Codage des états_____
STATEi = SO * S1                         ; pour dire que l'état STATEi correspond à la valeur 11 des
                                         sorties S0 et S1.
;_____ Equations de transition _____
STATEi := ini→ STATEj                    ; à partir de l'état STATEi on passe à l'état STATEj sachant la condition
+inj→STATEi+→statek                      (ini )et on reste dans l'état STATEi si la condition (inj) est vérifiée sur
                                         les entrées et on passe à l'état STATEk dans le cas contraire
;_____Output équations _____
STATEi OUTF= con1→s1*/s2*s3*/s4*/s5      ; sortie s1 au niveau H si on est dans l'état STATEi avec la
                                         condition con1 réalisée.
;_____ Conditions des équations _____
CONDITION                                ; Mot réservé.
CAR1 = /M*A6*/A5

;_____ Simulation segment _____
SIMULATION                               ; Mot réservé.
TRACE_ON clock A0 A1                     ; spécifie les signaux à tracer.
setf   A0  /A1                           ;donne les valeurs A0=0 et A1=1.
clockf  clock
check  STATEi                            ; vérifier que l'étati=1.
TRACE_OFF                                ; Mot réservé.
```

6.3.2 Choix des circuits PAL

La sélection des PALs dépend de plusieurs facteurs : taille, technologie, subdivision, synchronisation, connexions,... La figure 6.4 montre la structure externe du PAL de type PALce 26V12 et ses caractéristiques sont :

- Le nombre des entrées est 12.
- Le nombre des sorties est 13 et seule la sortie de broche N° 28 qui n'est pas synchronisée.

- Les broches 21 et 7 représentent successivement la broche de la masse et celle de l'alimentation.
- Les broches 20 et 22 peuvent accepter jusqu'à 16 maxtermes de 8 mintermes chacun.
- Les broches 19 et 23 peuvent accepter jusqu'à 14 maxtermes de 8 mintermes chacun.
- Les broches 18 et 24 peuvent accepter jusqu'à 12 maxtermes de 8 mintermes chacun.
- Les broches 17 et 25 peuvent accepter jusqu'à 10 maxtermes de 8 mintermes chacun.
- Les broches 16 et 26 peuvent accepter jusqu'à 8 maxtermes de 8 mintermes chacun.
- Les broches 15 et 27 peuvent accepter jusqu'à 6 maxtermes de 8 mintermes chacun [18].

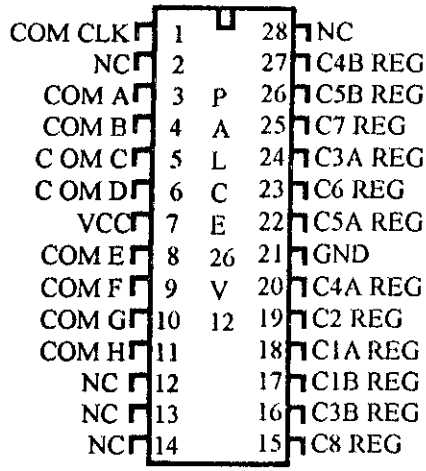


Figure 6.4 : Structure externe d'un PAL (PALce 26v12)

6.4 VALIDATION DU PROCESSEUR DE TEST PAR PALASM

Pour permettre une bonne vérification de la fonctionnalité de notre processeur, on a validé les différents modules avec PALASM.

6.4.1.1 Validation du PRPG

Le circuit logique du PRPG développé à la figures 4.3 et 5.3 peut être simulé à l'aide de PALASM. La validation du ce PRPG nécessite deux PALs :

Le premier PALce 26v12 peut être utilisé pour valider le signal de la boucle de retour des OU exclusifs. Ce PAL possède comme entrées l'état présent du PRPG et les signaux (C_1, C_2,...C_{16}) qui caractérisent le polynôme caractéristique comme l'indique la figure 6.5. Le deuxième PALce 26v12 peut être utilisé pour générer les vecteurs du PRPG et permettre le décalage de données. Il est commandé par 3 signaux : C_{1PRPG}, C_{2PRPG} et C_{3PRPG} (suivant la table 5.4) et par un signal pour la boucle de retour B qui est produit par le premier PAL1. Les entrées (I_0...I_{15}) sont utilisées pour le chargement de l'état initial.

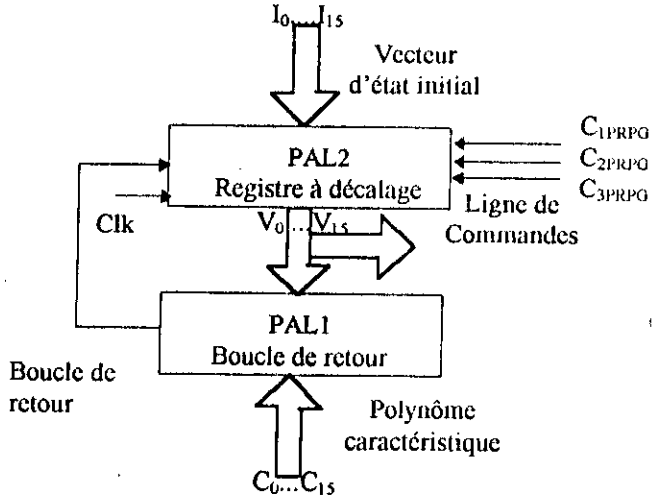


Figure 6.5 : Schéma bloc d'un PRPG validé par les PALs

Pour une simple application on présente sur la figure 6.6 le résultat de la validation d'un générateur de vecteurs de test pseudo-aléatoire avec un état initial I=(0111 1111) et polynôme caractéristique $P(x)=1+X^5+X^8$ dont le programme est présenté en Annexe 3 (la figure 1 présente la structure externe de PAL validé alors que la Figure 3 donne les signaux générer par ce PAL)

```
CIOCK 0110011011011011011011011011011011011011011011011011011011011011
I[1]  1111111111111111111111111111111111111111111111111111111111111111   ;I[1..7] vecteur d'état initial.
I[2]  0000000000000000000000000000000000000000000000000000000000000000
I[3]  0000000000000000000000000000000000000000000000000000000000000000
I[4]  0000000000000000000000000000000000000000000000000000000000000000
I[5]  0000000000000000000000000000000000000000000000000000000000000000
I[6]  0000000000000000000000000000000000000000000000000000000000000000
I[7]  0000000000000000000000000000000000000000000000000000000000000000
C[1]  0000111111111111111111111111111111111111111111111111111111111111   ;C[1] commande de génération.
GND   0000000000000000000000000000000000000000000000000000000000000000
/OE   0000000000000000000000000000000000000000000000000000000000000000
V[1]  1111110000000000001110000001110001110000000000001111110000111      ;V[1..8] vecteurs générés
V[2]  1100001110000000000011100000011100011100000000000011111100000      par le PRPG.
V[3]  1100000001110000000000011100000011100011100000000000011111
V[4]  1100000000001110000000000011100000011100011100000000000000111
V[5]  1100000000000011100000000000011100000011100011100000000000000
V[6]  1100000000000000011100000000000011100000011100011100000000000
V[7]  1100000000000000000011100000000000011100000011100011100000000
V[8]  1100000000000000000000011100000000000011100000011100011100000
VCC   1111111111111111111111111111111111111111111111111111111111111111
```

Figure 6.6 : Résultat de validation de PRPG

6.4.2 Validation de la logique de compression

Pour la validation du circuit de compression MISR, deux PALs sont utilisés. Le premier PAL comprime le vecteur qui entre en parallèle et produit la signature ($S_1,S_2,...S_{16}$) comme signaux de sorties de ce PAL. Le deuxième PAL réalise la boucle de retour du MISR.

Les entrées des deux PALs sont :

C_0, C_1, ...C_{15} : pour fixer le polynôme caractéristique qui est maintenu dans un registre pendant la compression.

I_0, I_2, ...I_{15} : les entrées pour fixer l'état initial du MISR.

D_0, D_2, ...D_{15} :représente les données à traiter comme le montre la figure 6.7

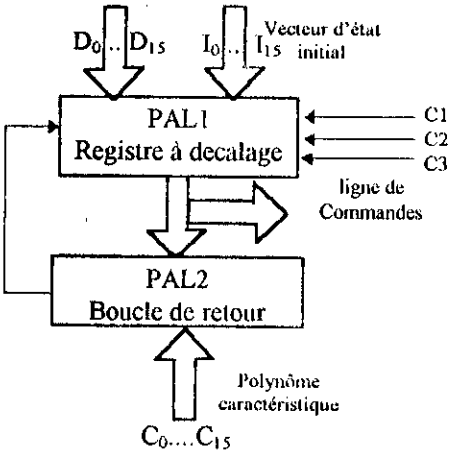


Figure 6.7 : Schéma bloc d'un MISR validé par les PALs

Pour un exemple d'application on présente sur la figure 6.8 le résultat de la compression d'une séquence Di(0...0) à l'aide d'une validation d'un MISR (avec un état initial I=(0001 1111) et polynôme caractéristique $p(x)=1+ X^5+X^8$) validé par PALASM dont le programme de validation est présenté en Annexe 3 (figures 2 et 3 : présentent respectivement la structure externe du PAL validé et les signaux générés par ce PAL). On remarque que le MISR, avec une séquence d'entrée de zéros, devient un générateur de vecteurs.

```
Clock   0110001100110011001100110011001100110011001100110011001100110011
I[1]    0000000000000000000000000000000000000000000000000000000000000000   ;I[1..3] vecteur d'état
I[2]    0000000000000000000000000000000000000000000000000000000000000000   initial.
I[3]    1111111111111111111111111111111111111111111111111111111111111111
D[1]    0000000000000000000000000000000000000000000000000000000000000000   ;D[1..4] vecteur de données
D[2]    0000000000000000000000000000000000000000000000000000000000000000
D[3]    0000000000000000000000000000000000000000000000000000000000000000
D[4]    0000000000000000000000000000000000000000000000000000000000000000
C[1]    0000111111111111111111111111111111111111111111111111111111111111   ;C[1] commande
GND     0000000000000000000000000000000000000000000000000000000000000000   de démarrage.
/OE     0000000000000000000000000000000000000000000000000000000000000000   ;SIG[1..8] résultat de la
SIG[1]  1100000111111111111111111110000000000001111111111111111111110   compression.
SIG[2]  1100000000011111111111111111100000000000011111111111111111
SIG[3]  1100000000000001111111111111111110000000000001111111111111
SIG[4]  1111111000000000000111111111111111111100000000000011111111
SIG[5]  1111111111100000000000111111111111111111100000000000011111
SIG[6]  1111111111111110000000000011111111111111111100000000000001
SIG[7]  1111111111111111100000000000111111111111111111000000000000
SIG[8]  1111111111111111111110000000000011111111111111111100000
VCC     1111111111111111111111111111111111111111111111111111111111111111
```

Figure 6.8 : Résultat de validation de MISR

6.4.3 Validation de l'unité de contrôle

L'unité de contrôle du processeur de test développé présente le noyau principal introduit pour accomplir les tâches spécifiques de test. Pour cela, tous nos efforts sont rassemblés pour obtenir une unité de contrôle performante. Cependant, la connaissance des mécanismes de test et de la synthèse séquentielle représentent la clef de succès.

La génération de commandes de cette unité de contrôle est précédée par deux phases de travail, la simplification des expressions logiques de commandes par un logiciel dit *DIGITAL* de « *WORK BENCH 1992* » qui est basé sur la méthode de « McCluskey » et la recherche de PAL possédant les caractéristiques optimales du nombre de commandes ayant des maxtermes et des mintermes adéquats. Le PAL utilisé dans notre travail est le PALce 26V12, dont les caractéristiques sont données sur la figure 6.4.

6.4.3.1 Validation de l'unité de contrôle câblée du PTDC

Après définition et production du diagramme d'états global de l'unité de contrôle, il s'est avéré qu'un seul PAL ne peut être utilisé par le logiciel PALASM. Afin de montrer que la structure de notre unité de contrôle produise effectivement tous les signaux de contrôle définis par les micro-instructions, On a procédé à la validation par une partition en blocs d'instructions où chaque groupe d'instructions est validé par un seul PAL.

On a subdivisé les instructions en 12 groupes ; utilisant 12 PALs pour la génération des états de l'unité de contrôle, 7 PALs pour le décodage de ces états pour la génération effective des signaux de contrôle ; car il y a des signaux de contrôle qui ne peuvent pas être générés qu'à partir d'une seule PIN (utilisation de PAL 8 comme des ports logiques OR). En total nous avons 59 signaux de contrôle à produire.

La figure 6.9 montre l'association des PALs qui composent l'unité de contrôle validée par PALASM. Les PALs utilisés pour la validation sont le PAL 22V10 et le PALce 26V12 et ceci car ils possèdent des sorties programmables (séquentielles et combinatoires) et ils ont des PINs qui supportent 16 minterms.

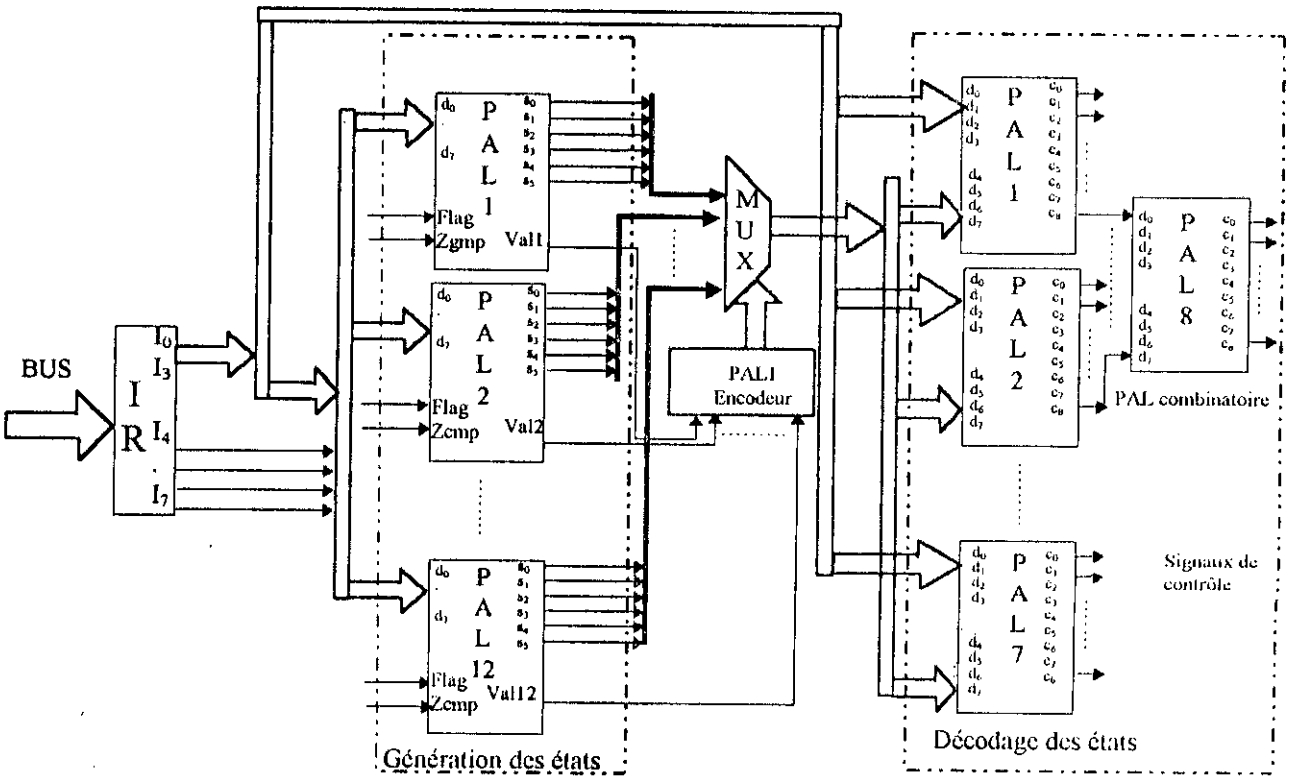


Figure 6.9 : Unité de contrôle câblée à base de PALs

Tout en sachant que les PALs de géneration des états et le PAL 8 de décodage des états sont de type PAL22V10 et le reste des PALs est de type PALce 26V12

6.4.3.2 Validation de l'unité de contrôle micro-programmée du PTDC

La complexité de l'unité de contrôle et la spécificité des signaux de contrôle a nécessite 7 PALs pour sa validation. En effet, après analyse des différents signaux de contrôle, on a utilisé 7 PALs pour piloter les 46 commandes exigées par tous les blocs du PTDC. L'unité de contrôle étudiée a une ROM de taille 46*158 bits. Le nombre de micro-instructions mémorisées est de 158 donc 8 bits d'adresse sont nécessaires.

Les PAL_1 jusqu'à PAL_5 produisent les signaux C_1 à C_{46} par groupe de 8 comme le montre la figure 6.10, le sixième et le septième PAL contrôlent la sortie du MUX et donc le chargement ou l'incrémentation du MPC ($L/\overline{INC}$).

La validation de l'UCMP réalisée suppose que le registre d'instruction RI contient déjà l'instruction. Ce dernier comporte deux champs, un pour l'identification de la micro-instruction de début et l'autre définit la condition interne s'il y en a une.

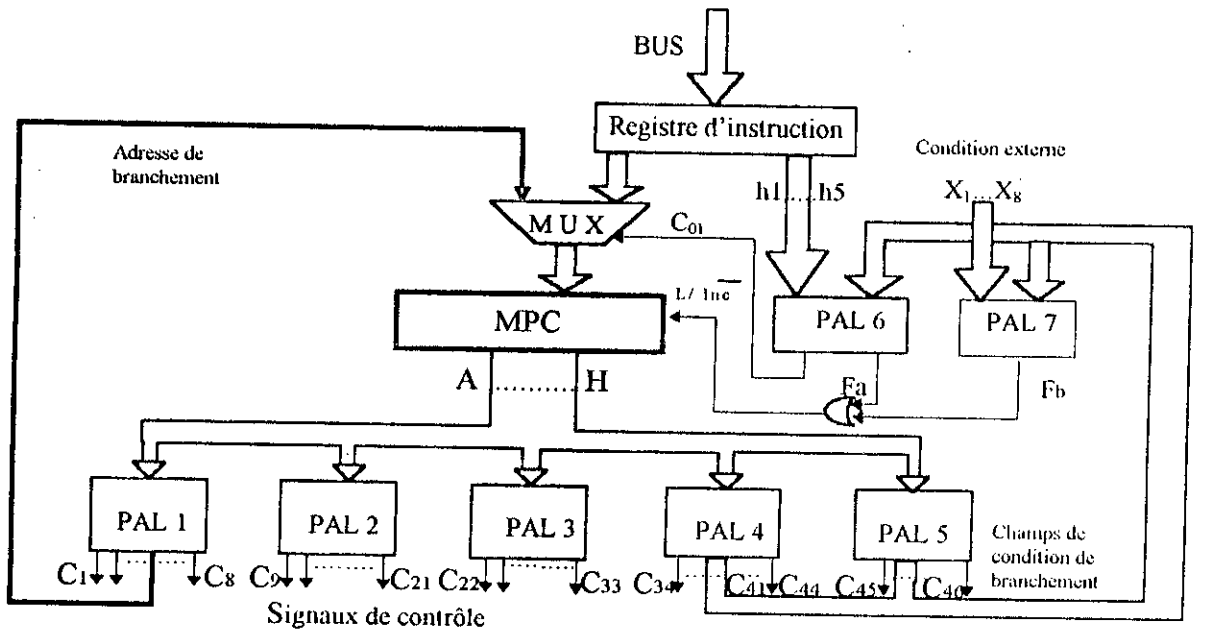


Figure 6.10 : L'organisation générale de MPCU (*Micro-Programmed Control Unite*)

Pour notre processeur on a 17 conditions internes et 8 conditions externes ; ces conditions sont sélactionnées par les bits du " champ de conditions" C_{41}-C_{45} (25 conditions au total ; 5 bits d'adressage du MUX).

Le signal C_{01} contrôle l'adresse de la micro-instruction suivante représentée par les signaux C_1 à C_8 quand il y a branchement. Dans le cas contraire, les signaux C_1 à C_8 forment les fonctions de contrôle des divers blocs du PTDC.

La première vérification du bon fonctionnement de cette unité de contrôle ainsi que de la bonne génération des commandes de test est faite par la validation de trois instructions : LACC1 ADD, INDAT OPR, ADD ADD1, ADD2. Le programme complet est consigné dans une disquette.

6.5 APPLICATIONS

Des exemples d'applications du processeur conçu sont développés ci dessous :

6.5.1 Test des composants possédant le standard BS

Considérons l'exemple montré à la figure 4.5 d'un système composé de deux CIs fabriqués selon le la technique BS. Pour tester ce système il faut vérifier la fonctionnalité de la chaîne BS puis les interconnexions des deux circuits.

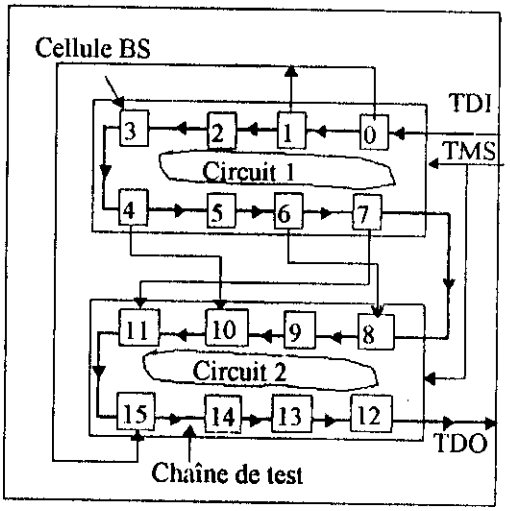


Figure 6.11 : Test des interconnexion

6.5.1.1 Test de la chaîne BS

Pour tester la chaîne BS de ce système, on suit les étapes suivantes :

On envoie en premier lieu un zéro logique au sein d'une séquence de 1 dans la chaîne de BS à travers TDI grâce à l'instruction Scan-DR-O, puis on récupère cette donnée par l'instruction Scan-DR-OI. Puis on procède de la même manière en décalant un 1 logique au sein d'une séquence de zéro préalablement chargée dans la chaîne BS, en suite, on décale la séquence 00110011... pour avoir toutes les possibilités entre deux cellules BS (00,01,10,11).

Le programme ci dessous présente la procédure de test de la chaîne BS.

```
Load Aacc, Add1        ; Définir l'état initial du MISR.
Load Bacc, Add2        ; Définir le polynôme caractéristique du MISR.
L_IS_MISR              ; Charger l'état initial du MISR à partir du Aacc.
L_PC_MISR              ; Charger le polynôme caractéristique du MISR à partir du Bacc.
I_BS                   ; Initialisation du TAP en le ramenant à l'état Shift_IR.
Sample                 ; Envoyer le code Sample vers le registre d'instruction du TAP
```

```
Sample                    .Des deux circuits.
Scan_DR_O 20(h), Add3     ; Envoyer un zéro dans une séquence de un(FFFF 7FFF).
Scan_DR_IO 10(h), Add4    ; Récupération du contenu du registre BS
Load Aacc, Bacc
Scan_DR_O 10(h),Add5      ; Envoyer un 1 dans une séquence de zéro 8000(h).
Scan_DR_IO 10(h),Add6     ; Envoyer une combinaison de zéro et de un 3333(h) et
Load Aacc,Bacc            récupérer la réponse de la 2em séquence.
CMPR Aacc                 ; Compression du contenu de Aacc en une signature.
Scan_Dr_O 10(h), Add5
Load Aacc,Bacc
CMPR Aacc
```

Le même exemple programmé avec le langage du processeur à unité de contrôle micro-programmée est explicité ci-dessous.

```
LISM, Add1                ; Charger l'état initial du MISR.
LCPM, Add2                ; Charger le polynôme caractéristique du MISR.
PMTBS                     ; Initialisation du TAP en le ramenant à l'état Shift_IR.
Sample                    ;Envoyer le code Sample vers le registre d'instruction du TAP
Sample                    des deux circuits.
INDAT, FFFF               ;Envoyer un zéro dans une séquence de un(FFFF 7FFF) à travers TDI.
INDAT, 7FFF
LISM, Add1                ; Charger l'état initial du MISR.
LCPM, Add2                ; Charger le polynôme caractéristique du MISR.
INDAT, 8000               ; Envoyer un 1 dans une séquence de zéro 8000(h)
INDAT, 8000
STACC1, 00FF              ; Stocker la signature de premier test.
INDAT, 3333(h)            ; Envoyer une combinaison de zéro et de un 3333(h) et
INDAT, 3333(h)            récupérer la réponse de la 2em séquence.
STACC1, 0100              ; Stocker la signature de deuxième test.
INDAT, 0000(h)            ; Envoyer une combinaison de zéro et de un 3333(h) et
INDAT, 0000(h)            ; récupérer la réponse de la 2em séquence.
STACC1, 0101              ; Stocker la signature de troisième test.
```

6.5.1.2 Programme de test des interconnexions

Pour le test des interconnexions du système montré à la figure 6.11, on exécute l'instruction Extest et on vérifie si la cellule d'entrée contient la même valeur que celle de sortie alors la connexion est établie.

La procédure de test des interconnexions est exécutée après l'initialisation du MISR, cette procédure est présentée ci-dessous :

```
I_BS                        ; Initialisation du TAP en le ramenant à l'état Shift_IR.
Sample                      ; Envoyer le code Sample vers le registre d'instruction du TAP
Sample                      des deux circuits.
Scan_DR_O 10(h), Add7       ; Envoyer une combinaison de zéro et un (F0F0 h).
I_BS
Extest                      ; Envoi du code Extest vers le registre d'instruction du TAP pour
Extest                      les deux circuits.
Scan_DR_IO 10(h), (Add7+1)  ; Envoyer une combinaison inverser de zéro et un (0F0F h) et
I_BS
Load Aacc, Bacc             récupération des résultat de Extest.
CMPR Aacc                   ; Comprime les réponses.
Scan_Dr_IO 10(h),Add7       ; Récupération des réponses du circuit.
Load Aacc, Bacc
CMPR Aacc                   ; Comprime les réponses de circuits.
```

Le même exemple traité avec le processeur à unité de contrôle micro-programmée donne :

```
PMTBS                       ; Initialisation du TAP en le ramenant à l'état Shift_IR.
Sample                      ; Envoyer le code Sample vers le registre d'instruction du TAP
Sample                      des deux circuits.
IDAT F0F0                   ; Envoyer une combinaison de zéro et un (F0F0 h).
IDAT F0F0
Extest                      ; Envoi du code Extest vers le registre d'instruction du TAP pour
Extest                      les deux circuits.
UPDAT
IDAT 0F0F                   ; Envoyer une combinaison inverser de zéro et un (0F0F h) et
IDAT 0F0F
STACC1, 00FF                ; Stocker la signature de premier test.
UPDAT
STACC1, 00FF                ; Stocker la signature de premier test.
```

On voit ici l'avantage du processeur de test en ce qui concerne le contrôle du TAP par l'envoi simultané des instructions BS et des données de test, l'application de l'horloge tck et de la séquence TMS ainsi que la séquence de donnée sur TDI. Tout ceci nécessite simplement l'exécution d'une instruction de test.

6.5.1.3 Test type Scan

Soit un circuit réalisé selon le mode Scan (CI avec registre scan interne de 16 bits) ayant 16 sorties et 10 entrées comme le montre la figure 6.12.

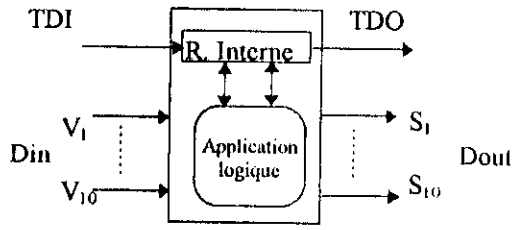


Figure 6.12 : Circuit intégré selon le mode scan

L'instruction Test-Scan-Vect réalise le test de ce type de circuits en exploitant l'avantage du mode Scan qui ramène le test de circuit séquentiels à un test de circuits combinatoires. La procédure de test est présentée ci dessous.

```
Load Aacc, Add1          ; Définir l'état initial du MISR.
Load Bacc, Add2          ; Définir le polynôme caractéristique du MISR.
L_IS_MISR                ; Charger l'état initial du MISR à partir du Aacc.
L_PC_MISR                ; Charger le polynôme caractéristique du MISR à partir du Bacc.
Test_Scan_Vect 10, 40    ; Génération et compression en utilisant le PRPG et le MISR.
```

Ici l'avantage du processeur de test est évident en ce qui concerne l'envoi du nombre important de vecteurs de test à l'application logique et en même temps la compression des réponses en signatures. Il faut noter que les vecteurs de test sont directement générés par le PRPG à chaque cycle d'horloge ce qui est impossible de réaliser à l'aide de processeurs courants.

6.5.1.4 Test par génération de vecteurs pondérées

Soit un circuit à 16 entrées et 10 sorties (figure 6.13) ; pour tester ce circuit on fait appel à la notion de segmentation par sensibilisation de chemins (§ 2.2.3.1). Le circuit est divisé en trois segments S1= 4, S2=4 et S3=8.

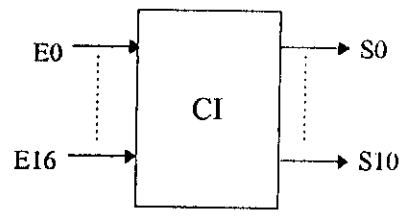


Figure 6.13 : Circuit sans DFT

Pour tester ce circuit on utilse la logique de poids pour fixer deux segments et tester le troisieme segement. Le programme de test est le suivant :

```
Load Aacc, Add1          ; Définir l'état initial du MISR..
Load Bacc, Add2          ; Définir le polynôme caractéristique du MISR..
L_IS_MISR                ; Charger l'état initial du MISR à partir du Aacc.
```

```
L_PC_MISR          ; Charger le polynôme caractéristique du MISR à partir du Bacc.
Load Aacc, 00 FF   ; Définir les 8 entrées à fixer de S3.
Load Bacc, FF FF   ; Définir les valeurs des entrées à fixer.
TPP, FF FF         ; Exécutions de test par poids on applique 2^8 vecteurs.
Load Aacc, 0F 00   ; Définir les 4 entrées à fixer de S2.
TPP, 00 FF         ; Exécutions de test par poids on applique 2^4 vecteurs.
Load Aacc, F0 00   ; Définir les 4 entrées à fixer de S1.
TPP, 00 FF         ; Exécution de test par poids on applique 2^4 vecteurs.
Load cmp, Aacc     ; Récupérer la valeur de la signature.
```

Ici on voit qu'il y a un nombre d'instructions réduit du programme de test. L'instruction TPP accomplit les tâches de génération, de pondération et de compression.

6.6 COMPARAISON

Dans le tableau suivant on présente quelques remarques concernant les 2 PTDC développés et un microprocesseur quelconque du marché. Les remarques sont liées à la structure hardware, software et à l'adaptation aux tests.

	Microprocesseur	**PTDC câblé**	**PTDC micro-programmé**
Hardware E/S et test	complexe (décodage d'adresse du CUT, liaison et emplacement du µP/CUT. -Bus multiple (avantage)	E/S prédéfinie par rapport à CUT. Système d'aiguillage + 1 bus	E/S prédéfinie. système à 2 bus.
Test BS	nécessite une interface de transmission	doté des registres et des tables de données pour le BS	doté de routine pour accomplir ce type de test
Test des différents composants	2 cycles sont nécessaires (lecture et écriture) pour chaque donnée	un seul cycle : envoi et réception d'une donnée	un seule cycle pour l'envoi et la réception de données
Test standard	nécessite une interface et des algorithmes [80].	doté d'un PRPG, logique de pondération et MISR Instructions prédéfnies	doté d'un PRPG, logique de pondération et MISR
Instruction	forme générale	63 instructions où 30 réservées au test	60 instructions où 32 pour le test
Codage et nombre d'instructions	multiple, nombre élevé d'instruction	8 bits, codage simple	16 bits, codage lineaire
Placement / CUT	-extérieur -Dépendance totale de l'application.	Intérieur / extérieur. -Peut être incorporé (MCM) -doté d'un système d'isolation et de By-pass	intérieur / extérieur. -Peut être incorporé (MCM) doté d'un système d'isolation et de By-pass

Tableau 6.1 : Comparaison PTDC microprocesseur « courant »

6.6.1 Test par génération de vecteurs

Exemple : CUT à 8 entrées de type Scan. Le tableau 6.2 résume les résultats de comparaison.

	microprocesseur	PTDC	Observation
Génération : **-déterministe** **-pseudo-aléatoire** **-pondérée**	-programmée -(2^8 ×8 × 2 cycles) -(plus 2^8 cycles)	-programmée. -Hardware (2^8 tops de Clk) -hardware (2 cycles)	La génération programmée par algorithme nécessite un temps supplémentaire >Tgen Hardware (traitement de matrices MISR PRPG)
compression	programmée (2^8 .×16 .× 3 cycles)	hardware (2^8 tops de Clk)	programme d'une seule bascule en compression necessite traitement de : X[i]:=X[i-1]⊕X[N]⊕D[i]
Mémoire de données	Necessite table spécifique	Stockage de signatures de références seulement. Etat démarrage	- Ce processeur de test réalise la compression hardware. -Ne nécessite pas de stockage au préalable.
temps de test	$64\times2^8<t_t<64*2^8+2^8\times161$(p) (cycle)	2^8 (cycle de Clk) + 2 cycle d'initialisation des registre de pondération	$t_t = t_g$ (temps de génération) + t_p (temps de pondération) + t_c (temps de compression)+ t_t (temps de traitement)

Tableau 6.2 : Comparaison pour un CUT-SCAN

6.6.2 Test d'un circuit doté d'un mécanisme Boundry-Scan :

Soit un circuit (à huit cellules d'entrées et huit cellules de sortie) placé dans une chaîne BS au quatrième rang comme le montre la figure 6.14. En supposant que le test des interconnexions est déja réalisé (toutes les connexions sont correctes), la fonctionnalité de ce circuit est vérifiée comme suit :

- On place tous les sept autres circuits en mode « BY-PASS ».
- On applique une procédure de test sur les entrées (chargement de cellules d'entrées).
- On sélectionne des états de sortie (initialisation ou affectation de sorties présentes).
- Vérification (*'mode cumulé') des signatures.*

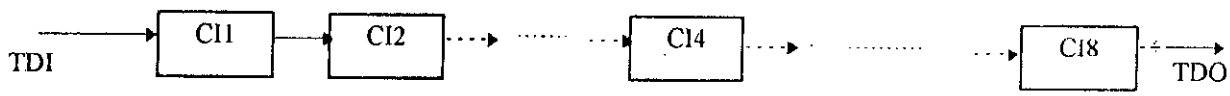


Figure 6.14 : Chaîne de BS

6.6.2.1 Programme de test en langage machine (Intel 8086)

```
        MOV CX, #05
        MOV AX, #FF
ADD1    MOV ADD, #AX
        DEC CX
        JNE ADD1
        MOV AX, #01100 ; Initialisation du TAP.
        MOV CX, #05
ADD2    MOV ADD, AX
        SAL AX
        DEC CX
        JNE ADD2            ;Sélectionner RI.
ADD4    MOV AX, # COD BY-PASS
        MOV CX, # FF FF
        MOV BX, # 04
ADD3    MOV ADD, AX
        SAL AX
        DEC CX
        JNE ADD3        ;Envoer le code Bay Pass.
        DEC BX
        JNE ADD4
ADD5    MOV AX, CODE INTEST
        MOV CX, # FF
        MOV ADD, # AX
        MOV ADD, # AX
        SAL AX
        DEC CX
```

```
        JNE ADD5     ;envoyer le code INTEST.
        MOV DX #07
ADD6  MOV AX, #101 1100
        MOV ADD, AX
        MOV BX, donnée
        MOV BX, 0000
ADD6  MOV ADD, AX
        MOV ADD, BX
        MOV DX, ADD
        SAL AX
        DEC
        JNE ADD6     ; Envoyer le Code By pass
        MOV AX, # 0000
        SAR DX
        DEC DX
        JNE ADD6 ; application de données et
                 récupération de réponses
        END.
```

6.6.2.2 Programme de test en PTDC cablé

```
        I-BS          ; Initialisation du TAP.
        CMP  00 04
ADD BAY PASS
        DEC CMP
        JZ  ADD1      ; Envoi du code By pass.
        INTEST        ; envoi du code Intest
        CMP 00 03
ADD2  BYPASS          ; Envoi du code By pass.
        DEC CMP
        JZ  add2      ; Envoi du code By pass.
        SCAN_DR_01, ADD1, ADD2 ;  application de données et récupération de réponses
HALT.
```

Remarque :

On note une réduction nette du nombre d'instructions du programme à base du PTDC et une diminution du temps d'exécution (utilisation des deux processeurs à la même fréquence).

Les conclusions à tirer sont les suivantes :

Obtention de programmes de test courts pour le PTDC.

Temps de test par circuit est réduit : temps de test global non excessif donc réduction du coût de test et amélioration de fonctionnalité globale pour un système à circuits multiples.

6.7 REMARQUES

Quand une application de test type BS est analysée, l'utilisateur de notre processeur de test n'a pas besoin de positionner le TAP à l'état voulu, il fournit uniquement les instructions spécifiques au test (Bypass, Extest,...). Ainsi une connaissance détaillée du TAP et de son fonctionnement n'est pas impérative alors que tous les mécanismes et orientations doivent être définis pour exploiter les instructions d'un processeur général, en plus des informations et données à définir avec rigueur.

Pour effectuer un test par la méthode la plus répandue à savoir par vecteurs, le processeur général a besoin de prendre des programmes entiers [78] imposant un temps de test très long et parfois inacceptable, donc contraire au but de la génération et compression pseudo-aléatoires. Le processeur de test intègre les deux unités de génération et de compression et offre donc un moyen simple pour la production des programmes de test.

Il est à noter que les spécificités et détails des mécanismes de commande du TAP (noyau du BS) sont pris en compte sous format d'une table de vecteurs pour deux objectifs :

- Réduire les accès à l'extérieur pour rendre les données disponibles (gagner en temps et éviter des adaptations électriques).
- Permettre des améliorations par simple changement de code et information dans cette table (système évolutif).

La validation des deux processeurs nous a éclairée sur les deux techniques de conception des unités de contrôles câblée et micro-programmée. Pour la première, on a besoin de 12 PALs pour la génération des états, 7 PALs pour le décodage de ces états et la génération des signaux de commandes et 2 PALs pour l'aiguillage, figure 6.10. Alors que pour la deuxième unité, on a besoin de 6 PALs pour la génération des signaux de commandes et 2 PALs pour la détermination des conditions et l'adresse de branchement. Une différence de 13 PALs nous donne un aperçu général sur le coût d'un contrôle câblé.

CONCLUSION

Conclusion

Notre contribution dans ce travail qui rentre dans le cadre du projet de recherche pour le "*développement des techniques de test des systèmes numériques*" se résume à l'étude et à la validation, à l'aide de l'outil CAO PALASM de *Advanced Micro Devices*, de deux contrôleurs programmables de test à architecture modulaire. Ces contrôleurs sont principalement définis pour prendre en charge les difficultés liées à l'accès limité aux points de test et présenter des facilités de production et de gestion d'opérations de test dans les environnements et systèmes structurés autour des mécanismes Scan, BIST et exploitant d'une manière générale la DFT.

Malgré la diversité des stratégies de conception orientées test, en pratique, un éventail très large de contraintes économiques et fonctionnelles limitent les seuils de testabilité. Il est devenu clair que, si des équipements automatiques de test doivent suivre l'évolution continuelle des CIs, seulement les équipements qui seront dotés d'un grand pouvoir d'adaptation technologique à l'environnement de test par programmation conduiraient à de solutions exploitables.

Notre travail s'inscrit dans cette perspective, c-a-d la mise en oeuvre d'un processeur de test. Pour que ce dernier, une fois montée dans l'application, ne soit pas lui même une charge supplémentaire de test (générateur de fautes et réducteur de performances des systèmes sous test), nous nous sommes fixés à n'inclure dans ce processeur que les éléments indispensables et justifiant d'un apport de simplicité d'exploitation de procédures de test.

Notre approche à été d'abord, le recensement de toutes les difficultés inévitables accompagnant les actions de test de type Scan, BS, BIST ... et ensuite la définition d'un répertoire d'instructions qui comprend toutes les opérations fortement recommandées pour achever un test de qualité (efficacité, temps, couverture de fautes, ...). Ces actions sont traduites par l'inclusion de modules de compression, de génération et d'aiguillage de données au sein de notre PTDC d'une part, et par la définition d'instructions qui agissent directement sur les signaux de contrôle des mécanismes BS, Scan et auto test des circuits sous test d'autre part.

Malgré l'architecture modulaire de notre PTDC, la validation par PALASM a nécessite un certain nombre de partitionnements. En effet, une optimisation partielle de certaines contraintes dues au nombre élevé de conditions de contrôle que doit gérer le contrôleur du PTDC, est réalisée avant la simulation complète de toute l'UC (noter les limitations de traitement d'équations simultanées et multiples par PALASM).

Nos résultats de simulation explicités dans cette étude montrent la validité de notre approche et les applications étudiées appuient l'utilité de notre PTDC.

Notre PTDC permet de réaliser le test des interconnexions de circuits à base de BS et des tests fonctionnels. Des modes de fonctionnement de type cumulatif, d'isolation et "By-pass" du PTDC sont définis pour favoriser le diagnostic et optimiser le temps de test.

Nous pensons que la réalisation d'un prototype assurerait non seulement, la validité de la technique de test développée dans ce projet mais, fournirait un outil sur lequel une expérimentation pour la production de séquences de synchronisation et des séquences dites *homing sequences* (véritables mécanismes de diagnostic, de vérification et de maintenance à masquages très réduits) est possible.

Pour obtenir des tests performants, à l'aide de processeurs dédies, tels que notre PTDC, nous pensons que les actions à entreprendre sont :

- Déployer tous les efforts pour développer des modèles de fautes très représentatifs des différentes sources d'erreurs.
- Réfléchir à ce que le processeur de test soit utilisé comme un catalyseur et un initiateur d'opérations de test pour échapper aux problèmes de synchronisation et de masquage.
- Définir un répertoire d'instructions très réduit pour faciliter la production de procédures de test.
- Définir des PTDC à partir d'unités expansibles et réfléchir à leur introduction de manière simple au sein des MCM.
- Protéger les entrées-sorties du PTDC pour travailler dans un environnement d'erreurs non coopératif et parfois sévère.
- Etude de la compatibilité des signaux spécifiques aux différentes technologies avec ceux des ports d'E/S du PTDC.

Enfin, après toute l'étude de la philosophie des mécanismes étroitement liés au test, nous tenons à appuyer que l'innovation dans le domaine de développement des techniques de test, avec un impact immédiat, serait dans la recherche et développement de circuits électriques de cellules (noyau des techniques scan) qui apporteraient la nouveauté suivante : "Toute défaillance au sein de la cellule correspond à une erreur qui se dévoile toute seule".

Bibliographie

BIBLIOGRAPHIE

Bibliographie

[1] M. Abramovici, M. A. Breur & A. D. Friedman, "Digital systems testing and testable design," *Computer Science Press England*, 1990.

[2] H. Fijuwara, "Logic testing and design for testability," *The MIT Press Cambridge*, March 1986.

[3] R. G. Bennetts, "Testable logic circuits," *Addition-Wesley Publishers Limited*, 1984.

[4] B. R. Wilkins, "Testing digital circuits : An introduction," *Van Nostrand Reinheld (UK) Co. Ltd*, 1986.

[5] A. Miczo, "Digital logic testing and simulation,", John Wiley & Sons, Inc, 1987.

[6] P. P. Gelsinger, "Design and test of the 80386," *IEEE Design & Test*, 1987,. pp. 42-56.

[7] F. Cathoor, J. Vansas, L. Inze & H. Deuman, "A testability strategy for multiprocessor architecture," *IEEE Design & Test* 1987, pp.12-20.

[8] Th. J. W. Verhallen & A. J. Vande Goor, "Functional testing of modern microprocessors," *IEEE EDAC Brussels*, 1992, pp. 350-354.

[9] H. Chemali & A. Boumaâraf, "Combination scan path for automatic design of testable circuits," Mediterranean conf. Electronics 1995, Grenoble (France) Sept. 1995.

[10] J. A. Abraham, "Design for testability," *IEEE Design & Test of computers*, 1992, pp. 22-26.

[11] T. W. Willams, "VLSI testing," Elsivier Science Publishers B.V., 1986

[12] R. Treuer, V. K. Agrawal & H. Fijuwara, "A new BIST design for PLA's with high fault coverage and low overhead," *IEEE Trans. Comput.* Vol. c-36 N° 3, march 1987, pp. 369-373.

[13] K. Wilken & J. P. Shen, , "Continuous signature monitoring : Efficient concurrent-detection of processor control errors," *IEEE Int. Test Conf.*, 1988, pp. 914-925.

[14] J. A. Abraham, "Test equipment Boards and board level testing," Elsivier Science Publishers B.V. 1986, pp. 239-275.

[15] V. D. Agrawal, K. T. Cheng & D. P. Johnson, "Designing circuits with partial scan," *IEEE Design & Test of computers*, April 1988, pp. 8-15.

[16] G. Grassl, "Design for testability," *SIEMENS rapport D8000 Munchen* 1983, pp. 1-36.

[17] J. A. Abraham, "Design for testability," *IEEE* 1983, pp. 278-280.

[18] E. B. Eichlberger & T.W. Williams, "A logic design structure for LSI testability," *Proc. Design Automation conf.*, 1977, pp. 462-468.

[19] T. W. Pradhan & S. K. Gupta, "A new framework for designing and analyzing BIST techniques and zero aliasing compression," *IEEE Trans. Comput.*, Vol. 40, N° 6, June 1991, pp. 743-763.

[20] C. M. Maunder & R.E. Tulloss, "An Introduction to the boundary scan standard: ANSI/*IEEE* Std 1149.1," *IEEE* Journal of electronic testing: theory and application 1991, pp. 11-26.

[21] R. G. Bennetts & A. Osseyran, "*IEEE* standard 1149.1-1990 on boundary scan: History, literature Survey, and current Status," *Journal of electronic testing: theory and application* Vol. 2 N° 1, March 1991, pp. 11-26.

[22] R. A. Teradyne, "Le Boundary Scan fait ses preuves en production," Mesures 645, Mai 1992, pp. 75-82.

[23] M. F. Lefebvre, "Functional test and diagnosis: a proposed JTAG sample mode scan tester," *IEEE Int. Test* Conf. 1990, pp. 244-303.

[24] S. M. Hassan & V. K. Agarwal, "BIST of PCB interconnects using BS architecture," *IEEE Design & Test of Computers*, April 1989, pp. 19-34.

[25] G. Ouillade, "Améliorer l'aptitude au test avec le boundary scan" *Electronique* N°40, Septembre 1994, pp. 81-83.

[26] G. Ouillade, "Le boundary scan applique a un modèle multipuce " *Electronique* N°40, Septembre 1994, pp. 84-89.

[27] K. P. Parker, "The impact of boundary scan board test," *IEEE Design & Test of computers*, Aug. 1989, pp. 18-30.

[28] D. W. Clark & L. J. Weng, " Maximal and near-maximal shift register sequences : Efficient event counters and easy discrete logarithms," *IEEE Trans.* on *Comput.*, Vol. c-43, May 1994, pp. 560-568.

[29] D. T. Tang, " Logic test pattern generation using linear codes," *IEEE Trans. on Comput.*, Vol. c-33 N° 9, Sept. 1984, pp. 845-849.

[30] M. Abramovici & D. T. Miller, "Are random vectors useful in test generation?," *IEE Int. Test Conf.* 1989. pp. 22-25.

[31] K. D. Wanger, C. K. Chin & E. J. McCluskey, " Pseudorandom testing," *IEEE Trans. Comput.* Vol. c-36 N° 3, march 1987, pp. 332-343.

[32] B. L. Dervisoglu, "Scan Path architecture for pseudorandom testing," *IEEE Design and Test of computers.*, Aug.1989, pp. 32-48.

[33] P. Bardel, W. Mcanney & J. Savir, "Built in test for VLSI: pseudorandom techniques," John wiley & sons New York 1987.

[34] F. Muradali, V. K. Agarwal & B. Nadeau-Dostie, "Anew procedure for weighted random BIST," *IEEE Int. Test Conf.*, 1990, pp. 660-669.

[35] D. Schnurmann, E. Lindbloom & R. G. Carpenter, "The weighted random test-pattern generator," *IEEE Trans. Comput.*, Vol. c-24 N° 7, July 1975, pp. 108-113.

[36] K. Iwasaki & V. Yamaguchi, "Design of signature circuits based on weight distributions of errors correcting codes," *IEEE Int. Test Conf,*. 1990, pp. 779-785.

[37] E. J. McClusky, "Verification testing-a pseudoexhaustive test technique," *IEEE Trans.* on *Comput.* Vol. c-33, June 1984, pp. 541-546.

[38] L. T. Wang & E. J. McClusky, "Complete feedback shift register for BIST," *IEEE ICAAD* 1986, pp. 56-59.

[39] W. Li, C. McCrosky & M. Abd-El-Barr, "The design and analysis of unmatched LFSR pattern generators in pseudorandom testing.," *IEEE Design & Test*, 1993, pp. 217-222.

[40] J. Savir, W. H. McAnney, "A multiple seed LFSR," *Proc. IEEE Int. Test* Conf., 1990, pp. 657-659.

[41] N. R. Saxena, P. Franco & E.J. McCluskey, "Simple bounds on serial signature analysis: Aliasing for random testing," *IEEE Trans. Comput.* Vol. 41, N°5, May 1992, pp. 638-653.

[42] J. P. Robinson & N. R. Saxena, "A unified view of test compression methods," *IEEE Trans. Comput.* Vol. c-36 N° 1, January 1987, pp. 94-99

[43] Z. Bazilai, J. Savir, G. Markowsky & M. G. Smith, "The weight syndrome sums approach to VLSI testing," *IEEE Trans.* on *Comput.*, Vol. c-30, Dec. 1981, pp. 996-1000.

[44] J. savir, "Syndrome testable design of combinational circuits," *IEEE Trans.* on *Comput.*, Vol. c-27, June 1980, pp. 613-620.

[45] S. M. Reddy, "A note on testing logic circuits by transition counting," *IEEE Trans. Comput.*, Vol. c-26, Mar. 1977, pp. 313-314.

[46] J. Savir & W. H. McAnney, " Double-parity signature analysis for LSSD networks," *IEEE Int. Test Conf.*, 1987; pp. 510-516.

[47] B. Bose, "Group theoretic signature analysis," *IEEE Trans. Comput.*, Vol. 39 N° 11, November 1990, pp. 1388-1403.

[48] M. G. Karpovsky & P. Nagvajara, "Optimal robust compression of test," *IEEE Trans. Comput.*, Vol. 39 N° 1, Janu 1990, pp. 138-141.

[49] M. G. Karpovsky & P. Nagvajara, "Optimal time space compression of test responses for VLSI devices," *IEEE Int. Test Conf.*, 1987, pp. 523-529.

[50] J. E. Smith, "Measures of the effectiveness of faults signature analysis," *IEEE Trans. on Comput.*, Vol. c-39, N°6, June. 1980, pp. 510-514.

[51] S. B. Tan, K. Totton, K. Baker & R. Porter, "A fast signature simulation tool for BIST circuits," *IEEE* Design & *Test* of computers, 1987, PP.17-25.

[52] K. K. Saluja, "An efficient signature computation method," *IEEE Design and Test of computers*, 1992, pp. 22-26.

[53] J. Savir & W. H. McAnney, "Identification of failing tests with cycling registers," *IEEE Int. Test Conf.* 1988; pp. 322-328.

[54] T.W. Williams, W. Daehn, M. Gruetzner & C.W. Starke, "Aliasing errors in signature analysis registers," *IEEE Design & Test*, 1987, pp.39-45.

[55] J. C Chan & J. A. Abraham, "A study of faulty signatures using a matrix formulation," *IEEE Int. Test Conf.*, 1990; pp. 553-561.

[56] P. C. Maxwell, "Comparative analysis of different implementation of multiple inputs signature analyzers," *IEEE Trans. Comput.*, vol.37 N°11, Nov. 1988, pp. 1411-1416.

[57] R. David, "Comment on signature analysis for multiple output circuits," *IEEE Trans. Comput.*, Vol. 39, N°2, Feb. 1990, pp. 287-288.

[58] P. H. Bardell & W. H. McAnney, "Self-testing of multichip logic modules," *IEEE Int. Test Conf.*, Nov. 1982; pp. 200-204.

[59] E. B. Eichlberger & E. Lindbloom, "Random-pattern coverage enhancement and diagnosis for LSSD logic Self-test," *IBM journal of research & development*, Vol. 27, N°3, May 1983, pp. 265-272.

[60] K. K. Saluja, R. Sharma & C. R. Kime, "A concurrent testing technique for digital circuits," *IEEE Trans. on Computer-Aided Design*, Vol. 7, N°12, Dec. 1988, pp. 1250-1259.

[61] L. T. Wang, "A self test diagnosis architecture for board using boundary scan," Proc. European *Test* Conf., 1989, pp. 119-126.

[62] P. Nagvajara, M. G. Karpovsky & L. B. Levith, "Pseudorandom testing for boundary scan design with built in self test," *IEEE Design & Test of computers*, Sep. 1991, pp. 58-65.

[63] C. S. Gloster & F. Brglez, "Boundary scan with built in self test," *IEEE Design & Test of Computers*, 1989, pp. 36-55.

[64] A. P. Strole & H. J Wunderlich, "Error masking in self-testable circuits," *IEEE Design & Test of Computers*, 1990, pp. 544-552.

[65] C. E. Stroud, "Automated BIST for sequential logic synthesis," *IEEE Trans. Comput.*, 1988, pp. 22-32.

[66] L. T. Wang & E. J. McClusky, "Concurrent Built-In Logic Block Observer (CBILBO)," *Int. Sump. On Circuit and Systems*, Vol. 3, 1986, pp. 1054-1057.

[67] P. D. Hortensius, R. D. McLeod & H. C. Card, "Cellular Automata Based signature analysis for built in self test," *IEEE Trans. Comput.*, vol.39 N°10, Oct. 1990, pp. 1273-1282.

[68] A. S. M. Hassan, V. K. Agarwal, B, Nadeau-Dostie & J. Rajski, "BIST of PCB interconnects using boundary scan architecture," *IEEE Trans. Computers*, vol. 11, N° 10, Oct. 1992, pp. 1279-1288.

[69] S. Chau, "Fault injection boundary scan design for verification of fault tolerant systems," *IEEE Int. Test Conf.*, 1994, pp. 677-682.

[70] N. Vasanthavada & N. Kanopovlos, "A built in test module for fault isolation," *IEEE Design & Test of Computers*, 1990, pp.58-65.

[71] G. Sapp, "ASSIST (Allied Signal's Standardized Integrated Scan Test)," *IEEE Int. Test Conf.*, 1990, pp. 95-102.

[72] H. Chemali & D. Mabrouk, "A controlled test and diagnostic module for self testable boards," *Proc. of the 6th Int. Conf. On micro-electronics ICM'94, Turkey*, Sept. 1994, pp.158-161,

[73] H. Chemali, D. Mabrouk & A. Boumaâraf, "Module d'isolation de test et d'aide au diagnostic (MITAD)," Proceeding of the CMSES, Skikda May 1994, pp. 177-183.

[74] D. Mabrouk, "Module d'isolation de test et d'aide aux diagnostic (MITAD)," *Thèse de magister à l'institut d'électronique, Université de Sétif*, 1991.

[75] V. C. Hamacher, S. G. Zaky & Z. G. Vranesic, "Structure des ordinateurs," *Mc Graw-Hill*, 1985.

[76] M. Rafiquzzaman & R. Chaudra, "Modern computer architecture," *West publishing company* USA 1988.

[77] P. Guerrange, "Documentation de PALASM," Advanced Micro Devices, 1992.

[78] M. S. Abadir, A. R. Parikh, P. A.Sandborn, K. Drake & Linda Bal "Analyzing multichip module testing strategies," *IEEE Design & Test of computers*, Vol.11 N°.1, Spring 1994, pp. 40-52.

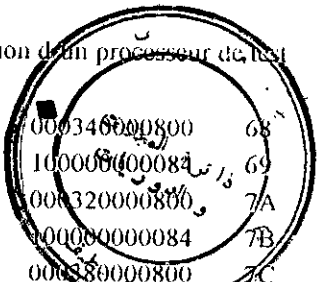
[79] J. C. Heudin & C. Panetto, "Les architectures RISC, Théorie et pratique des ordinateurs à jeu d'instructions réduit," *Edition Dunod*, 1990.

[80] A. Boumaâraf & L. Assassi, "Système de test et de diagnostic par analyse de signature," *Mémoire d'ingeniorat à l'institut d'électronique*, Université de Sétif, 1993.

ANNEXES

ANNEXE 1 : Programme résident dans la mémoire de contrôle

AD	Micro-instructions	Commandes	AD
0	PC←0		00
1	IF FLAG=0 then go to 118	180000000076	01
2	MAR←PC	000000000008	02
3	MBR←((MAR)) PC←PC+1	000000000071	03
4	BUSB←MBR MAR←PC ; RI←BUSB	000000000098	04
5	MBR←((MAR)) PC←PC+1	000000000071	05
6	if X4=1 go to CH1	080000000000	06
7	BUSB←MBR MAR←BUSB	00000000001C	07
8	MBR←((MAR))	000000000070	08
9	If y1=1 then go to 25	020000000019	09
10	If y2=1 then go to 27	12000000001B	0A
11	If y3=1 then go to 29	0A000000001D	0B
12	If y4=1 then go to 31	1A000000001F	0C
13	If y5=1 then go to 33	060000000021	0D
14	If y6=1 then go to 35	160000000023	0E
15	Ify7=1 then go to 37	0E0000000025	0F
16	If y8=1 then go to 39	1E0000000027	10
17	If y9=1 then go to 41	010000000029	11
18	If y10=1then go to 61	11000000003D	12
19	If y11=1 then go to 43	09000000002B	13
20	If y12=1 then go to 47	19000000002F	14
21	If y13=1 then go to51	050000000033	15
22	If y14=1 then go to 56	150000000038	16
23	If y15=1 then go to 100	0D0000000064	17
24	If y16=1 then go to 154	03000000009A	18
25	BUSB←MBR ACC1←BUSB	000000008010	19
26	go to 1	100000000001	1A
27	BUSB←MBR ACC2←BUSB	000000000810	1B
28	go to 1	100000000001	1C
29	BUSB←MBR COUN←BUSB	000004000010	1D
30	go to 1	100000000001	1E
31	BUSB←MBR PRPG(is)←BUSB	000000700010	1F
32	go to 1	100000000001	20
33	BUSB←MBR MISR(is)←BUSB	000000060010	21
34	go to 1	100000000001	22
35	BUS←BMBR PRPG(cp)←BUSB	000000500010	23
36	go to 1	100000000001	24
37	BUSB←MBR MISR(cp)←BUSB	000000020010	25
38	go to 1	100000000001	26
39	BUSB←MBR PRPG(rc)←BUSB	000000400010	27
40	go to 1	100000000001	28
41	BUSB←MBR PRPG(rd)←BUSB	000000600010	29
42	go to 1	100000000001	2A
43	BUSB←MBR COUN←BUSB	000004000010	2B
44	decCOUN BUSA←COUN CUT←BUSA BUSB←CUT MISR←BUSB	000006843000	2C
45	If COUN<>0 then go to 44	1C000000002C	2D
46	go to 54	100000000036	2E
47	BUSB←MBR COUN←BUSB	000004000010	2F
48	dec COUN BUSA←PRPG CUTB←USA BUSB←CUT MISR←BUSB	000002B43000	30
49	If COUN<>0 then go to 48	1C0000000030	31
50	go to 54	100000000036	32
51	BUSB←MBR COUN←BUSB	0000 0400 0010	33
52	dec COUN BUSA←PRPG BUSB←BUSA MISR←FUSB	000002340100	34
53	If COUN<>0 then go to 52	1C0000000034	35
54	BUSA←MISR BUSB←BUSA ACC1←BUSB	000000088100	36
55	go to 1	100000000001	37
56	ACC2←BUSB BUSB←MBR	000000000810	38
57	COUN←10	000025000000	39
58	dec COUN CUT←ACC2(s) ACC1(s)←CUT	000002806400	3A
59	If COUN<>0 then go to 58	1C000000003A	3B
60	go to 1	100000000001	3C
61	BUSB←MBR PORT/S←BUSB	002000000010	3D



62	go to 1	100000000001	3E
63	If ACC2<>0 then go to 1	140000000001	3F
64	BUSB←MBR PC←BUSB	000000000012	40
65	go to 1	100000000001	41
66	If ACC1=0 then go to 1	040000000001	42
67	go to 64	100000000040	43
68	UAL←ACC1 and ACC2	00040000CE00	44
69	go to 81	100000000051	45
70	UAL←ACC1 cmp ACC2	001400000CE00	46
71	go to 81	100000000051	47
72	UAL←NEG(ACC1)	00180000C200	48
73	go to 81	100000000051	49
74	UAL←NOT(ACC1)	00080000C200	4A
75	go to 81	100000000051	4B
76	UAL←ACC1 xor ACC2	000C0000CE00	4C
77	go to 81	100000000051	4D
78	UAL←ACC1+ACC2	001C0000CE00	4E
79	go to 81	100000000051	4F
80	UAL←ACC1-ACC2	00100000CE00	50
81	ACC1←UAL	000000008200	51
82	go to 1	100000000001	52
83	BUSB←MBR MAR←BUSB	00000000001C	53
84	BUSA←ACC1 BUSB ←BUSA MBR ←BUSB	00000000C140	54
85	go to 88	100000000058	55
86	BUSB←MBR MAR←BUSB	00000000001C	56
87	BUSA←ACC2 BUSB←BUSA MBR←BUSB	000000000D40	57
88	((MAR))←MBR	000000000020	58
89	go to 1	100000000001	59
90	BUSB←ACC1 PORT/S←BUSB	00200000C100	5A
91	go to 1	100000000001	5B
92	BUSB←ACC2 PORT/S←BUSB	002000000D00	5C
93	go to 1	100000000001	5D
94	COUN←000B	000045000000	5E
95	RTMS←FB40	0000A9000000	5F
96	go to 142	10000000008E	60
97	COUN←0006	000065000000	61
98	RTMS←E800	0000E9000000	62
99	go to 142	10000000008E	63
100	BUSB←MBR ACC2←BUSB	000000000810	64
101	go to 132	100000000084	65
102	ACC2←code_bypass	000300000800	66
103	go to 132	100000000084	67
104	ACC2←code_intest	000340000800	68
105	go to 132	100000000084	69
106	ACC2←code_extest	000320000800	7A
107	go to 132	100000000084	7B
108	ACC2←code_sample	000380000800	7C
109	go to 132	100000000084	7D
110	ACC2←code_runbist	000360000800	7E
111	go to 132	100000000084	7F
112	ACC2←code_icode	0003A0000800	80
113	go to 132	10000000008446	81
114	ACC2←code_user_code	003C0000800	82
115	go to 132	100000000084	83
116	ACC2←code_scan(bs)	0007C0000800	84
117	go to 132	100000000084	85
118	COUN←0002	000005000000	86
119	decCOUN	000002000000	87
120	BUSA←COUN BUSB←BUSA ACC2←BUSB	000006000900	88
121	BUSB←PORT/E COUN←BUSB	00C004000000	89
122	BUSB←PORT/E MAR←BUSB ACC1←BUSB	00C00000800C	8A
123	BUSB←PORT/E MBR←BUSB	00C000000040	8B
124	((MAR))←MBR decCOUN	00002000020	8C
125	UAL←ACC1+ACC2 avecP5=1	009C0000CE00	8D
126	ACC2←UAL	000000008200	8E
127	BUSA←ACC2 MAR ← USB BUSA ← USB	000000000D0C	8F
128	If COUN<>0 then go to 123	1C000000007B	90
129	FLAG1(P4=1)	004000000000	91
130	go to 1	100000000001	92
131	go to 131	100000000083	93
132	COUN←0002	000005000000	94
133	RTMS←8000	000089000000	95
134	decCOUN ; RTMS(s)	000012800000	96
135	If COUN<>0 then go to 134	1C0000000086	97
136	COUN←10	000025000000	98
137	RTMS←0	0000A9000000	99
138	decCOUN ; RTMS(s) CUT←ACC2(s) ACC1(s)←CUT MISR←ACC1(s)	0000A9000000	9A
139	If COUN<>0 then go to 138	1C0000000084	9B
140	COUN←0002	000005000000	9C
141	RTMS←8000	000089000000	9D
142	decCOUN←RTMS(s)	000012800000	9E

143	If COUN<>0 then go to 142	1C000000008E	9F
144	BUSA←MISR BUSB←BUSA ACC2←BUSB	000000080900	A0
145	go to 1	100000000001	A1
146	BUSB←MBR MAR←BUSB ACC1←BUSB	00000000801C	A2
147	MBR←((MAR))	000000000070	A3
148	decCOUN UAL←ACC1+ACC2	001C0200CE00	A4
149	ACC1←UAL	000000008200	A5
150	BUS A←ACC1 BUS B←BUSA MAR←BUSB avec P5	008020000C10C	A6
151	BUS B←MBR	002000000010	A7
	PORT/S←BUSB		
152	If COUN<>0 then go to 147	1C000000009B	A8
153	go to 1	100000000001	A9
154	CX←BUSB BUSB←MBR	400000000010	AA
155	go to 1	100000000001	AB
156	If CX=0 then go to 1	1D0000000001	AC
157	PC←BUSB decCX BUSB←MBR	200000000012	AD
158	go to 1	100000000001	AE

s : décalage.

is : état initiale.

C.P. : polynôme caractéristique.

PORTs : port de sortie.

PORTe : port d'entrée.

ANNEXE 2 : Programme résident dans la mémoire de contrôle (Nanomemory)

AD	Micro-instructions	Commandes	AD
0	*PC*←0		00
1	IF FLAG=0 *then go to 118*	*1*80000000076	0*1*
2	MAR←*PC*	000000000008	02
3	MBR←((MAR))	00000000007*1*	03
	PC←*PC+1*		
4	BUSB←MBR	000000000098	04
	MAR←*PC* ; RI←BUSB		
5	MBR←((MAR))	00000000007*1*	05
	PC←*PC+1*		
6	*if* X4=*1 then go to CH1*	080000000000	06
7	BUSB←MBR	0000000000*1C*	07
	MAR←BUSB		
8	MBR←((MAR))	000000000070	08
9	If y*1*=*1 then go to 25*	020000000*19*	09
10	If y*2*=*1 then go to 27*	*12*00000000*1*B	0A
11	If y*3*=*1 then go to 29*	0A00000000*1*D	0B
12	If y4=*1 then go to 31*	*1*A00000000*1*F	0*C*
13	If y5=*1 then go to 33*	060000000*21*	0D
14	If y6=*1 then go to 35*	*16*0000000*23*	0E
15	Ify7=*1 then go to 37*	0E0000000*25*	0F
16	If y8=*1 then go to 39*	*1*E0000000*27*	*10*
17	If y9=*1 then go to 41*	0*1*0000000*29*	*11*
18	If y*10*=*1then go to 62*	*11*000000003D	*12*
19	If y*11*=*1 then go to 43*	09000000002B	*13*
20	If y*12*=*1 then go to* 47	*19*000000002F	*14*
21	If y*13*=*1 then go to51*	0500000000*33*	*15*
22	If y*14*=*1 then go to* 56	*15*000000003*8*	*16*
23	If y*15*=*1 then go to 100*	0D0000000064	*17*
24	If y*16*=*1 then go to 105*	03000000009A	*18*
25	BUSB←MBR	000000008*10*	*19*
	ACC*1*←BUSB		
26	*goto1*	*1*00000000001	*1*A
27	BUSB←MBR	00000000*810*	*1*B
	ACC*2*←BUSB		
28	BUSB←MBR	0000040000*10*	*1C*
	COUN←BUSB		
29	BUSB←MBR	000000700*10*	*1*D
	PRPG(*is*)←BUSB		
30	BUSB←MBR	00000060*010*	*1*E
	MISR(*is*)←BUSB		
31	BUS←BMBR	000000500*10*	*1*F
	PRPG(*cp*)←BUSB		
32	BUSB←MBR	0000000200*10*	*20*
	MISR(*cp*)←BUSB		
33	BUSB←MBR	000000400*10*	*21*
	PRPG(*rc*)←BUSB		
34	BUSB←MBR	000000600*10*	*22*
	PRPG(*rd*)←BUSB		
35	*dec*COUN	00006843000	*23*
	BUSA←*COUN*		
	CUT←BUSA		
	BUSB←*CUT*		
	MISR←BUSB		
36	If *COUN*<>0 *then go to* 44	*1C*000000002*C*	*24*
37	*goto* 54	*1*00000000036	*25*
38	*dec*COUN	000002B43000	*26*
	BUSA←*PRPG*		
	CUTB←USA		
	BUSB←*CUT*		
	MISR←BUSB		
39	If *COUN*<>0 *then go to* 48	*1C*0000000030	*27*
40	*dec*COUN	000002340*100*	*28*
	BUSA←*PRPG*		
	BUSB←BUSA		
	MISR←BUSB		
41	If *COUN*<>0 *then go to* 52	*1C*0000000034	*29*
42	BUSA←MISR	000000088*100*	*3A*
	BUSB←BUSA		
	ACC1←BUSB		
43	*COUN*←*10*	000025000000	*3B*
44	*dec*COUN	000002806400	*3C*
	CUT←*ACC2*(*s*)		
	ACC1(*s*)←*CUT*		
45	If *COUN*<>0 *then go to* 58	*1C*000000003A	3D
46	BUSB←MBR	00200000001*0*	3E
	PORT/S←BUSB		
47	If *ACC2*<>0 *then go to 1*	*1*40000000001	3F
48	BUSB←MBR	000000000*12*	40
	PC←BUSB		
49	If *ACC1*=0 *then go to 1*	040000000001	*41*
50	*goto* 64	*1*00000000040	*42*
51	UAL←*ACC1 and ACC2*	00040000*C*E00	*43*
52	*goto* 81	*1*00000000051	44
53	UAL←*ACC1 cmp ACC2*	001400000CE00	45
54	UAL←NEG(*ACC1*)	00180000*C*200	46
55	UAL←NOT(*ACC1*)	00080000*C*200	47
56	UAL←*ACC1 xor ACC2*	000*C*0000*C*E00	48
57	UAL←*ACC1*+*ACC2*	001*C*0000*C*E00	49
58	UAL←*ACC1*-*ACC2*	00100000CE00	5A
59	*ACC1*←UAL	000000008200	5B
60	BUSA←*ACC1*	00000000*C*140	5*C*
	BUSB ←BUSA		
	MBR ←BUSB		
61	*goto* 88	*1*00000000058	5D
62	BUSA←*ACC2*	000000000D40	5E

	BUSB←BUSA		
	MBR←BUSB		
63	*go to 131*	000000000020	5F
64	BUSB←*ACC1*	00200000*C*100	60
	*P*ORT/S←BUSB		
65	BUSB←*ACC2*	002000000D00	61
	*P*ORT/S←BUSB		
66	*C*OUN←000B	000045000000	62
67	RTMS←FB40	0000A9000000	63
68	*go to 142*	10000000008E	64
69	*C*OUN←0006	000065000000	65
70	RTMS←E800	0000E9000000	66
71	*goto 132*	10000000008C	67
72	*ACC2←code_bypass*	000300000800	68
73	*ACC2←code_intest*	000340000800	69
74	*ACC2←code_extest*	000320000800	6A
75	*ACC2←code_sample*	000380000800	6B
76	*ACC2←code_runbist*	000360000800	6C
77	*ACC2←code_icode*	0003A0000800	6D
78	*ACC2←code_user_code*	003C0000800	6E
79	*ACC2←code_scan*(bs)	0007C0000800	6F
80	*C*OUN←0002	000005000000	70
81	*dec*COUN	000002000000	71
82	BUSA←*C*OUN	000006000900	72
	BUSB←BUSA		
	ACC2←BUSB		
83	BUSB←*P*ORT/E	00C004000000	73
	*C*OUN←BUSB		
84	BUSB←*P*ORT/E	00C00000800C	74
	MAR←BUSB		
	ACC1←BUSB		
85	BUSB←*P*ORT/E	00C000000040	75
	MBR←BUSB		
86	((MAR))←MBR	00002000020	76
	*dec*COUN		
87	UAL←*ACC1*+*ACC2*	009C0000CE00	77
	avec P5=1		
88	BUSA←*ACC2*	000000000D0C	78
	MAR ←BUSB		
	BUSA ←BUSB		

89	If *C*OUN<>0 *then go to*	1C000000007B	79
90	FLAG1(*P4=1*)	004000000000	7A
91	MBR ← ((MAR))	100000000083	7B
92	RTMS←8000	000089000000	7C
93	*dec*COUN ; RTMS(*s*)	000012800000	7D
94	If *C*OUN<>0 *then go to*	1C0000000086	7E
95	RTMS←0000	0000A9000000	7F
96	*dec*COUN ; RTMS(*s*)	0000A9000000	80
	*C*UT←*ACC2*(*s*)		
	ACC1(*s*)←*C*UT		
	MISR←*ACC1*(*s*)		
97	If *C*OUN<>0 *then go to*	1C0000000084	81
98	If *C*OUN<>0 *then go to*	1C000000008E	82
99	BUSA←MISR	000000080900	83
	BUSB←BUSA		
	ACC2←BUSB		
100	BUSB←MBR	00000000801C	84
	MAR←BUSB		
	*ACC1*BUSB		
101	*dec*COUN	001C0200CE00	85
	UAL←*ACC1*+*ACC2*		
102	BUSA←*ACC1*	008020000C10C	86
	BUSB←BUSA		
	MAR←BUSB		
	avec P5		
103	If *C*OUN<>0 *then go to*	1C0000000093	87
104	*C*X←BUSB	400000000010	88
	BUSB←MBR		
105	If *C*X=0 *then go to 1*	1D0000000001	89
106	*PC*←BUSB	200000000012	8A
	*dec*CX		
	BUSB←MBR		

s :décalage.
is :état initiale.
cp :polynôme caractéristique.
PORTs :port de sortie.
PORTe :*port d'entrée*.
AD : *Adresse*.

ANNEXE 3 : Programmes de validation par PALASM du PRPG et MISR

Fichier N°1 : Validation d'un PRPG

```
;PALASM Design Description
;-------declaration Segment ------------
TITLE  VALIDATION du PRPG (type externe)
PATTERN UNIVERSITE DE SETIF
REVISION 1.0
AUTHOR  A. Boumaâraf
COMPANY logic_labo
DATE  26/06/95

CHIP temp   PALCE16V8

;----- PIN Declarations ---------------
PIN  1          CLOCK   COMBINATORIAL  ; INPUT
PIN  2..8       I[1..7]  COMBINATORIAL  ; INPUT       ; Pins d'états initiaux.
PIN  9          C[1]     COMBINATORIAL  ; INPUT       ; Pin de commande.
PIN  20         VCC                     ; INPUT
PIN  11         /OE                     ; INPUT
PIN  12..19     V[1..8]  REGISTERED      ; OUTPUT      ; Sorties de vecteurs à générer.
PIN  10         GND                     ; INPUT

;-------- Booleen Equation Segment ------

EQUATIONS
V[1]:=C[1]*(V[8]:+:V[5])+/C[1]*I[1]
V[2]:=C[1]*(V[1])+/C[1]*I[2]
V[3]:=C[1]*(V[2])+/C[1]*I[3]
V[4]:=C[1]*(V[3])+/C[1]*I[4]
V[5]:=C[1]*(V[4])+/C[1]*I[5]
V[6]:=C[1]*(V[5])+/C[1]*I[6]
V[7]:=C[1]*(V[6])+/C[1]*I[7]
V[8]:=C[1]*(V[7])+/C[1]*I[7]

;_________ Simulation Segment _______________
SIMULATION
 trace_on clock C[1] V[1..8]
 setf  OE /C[1] I[1] /I[2..7]          ; Fixation d'état initial.
   clockf clock
 setf C[1]                             ; Commande de génération.
for I:=1 to 32 do                      ; Génération de 2^5 vecteurs.
 begin
   clockf clock
  end
trace_off
□
```

Fichier N°2 : validation d'un MISR

```
;PALASM Design Description

;-------Declaration Segment ------------
TITLE  VALIDATION du MISR (type interne)
PATTERN UNIVERSITE DE SETIF
REVISION 1.0
AUTHOR  A. Boumaâraf
COMPANY logic_labo
DATE  26/06/95

CHIP temp   PALce16V8

;----- PIN Déclarations ---------------
PIN  1        CLOCK       COMBINATORIAL  ; INPUT
pin  2..4     I[1..3]     COMBINATORIAL  ; INPUT     ;Pins d'états initiaux.
PIN  5..8     D[1..4]     COMBINATORIAL  ; INPUT     ;Pins de données.
PIN  9        C[1]        COMBINATORIAL  ; INPUT     ;Pin de commande.
PIN  20       VCC
PIN  12..19   SIG[1..8]   REGISTERED     ; OUTPUT    ;Les sorties de la compression.
PIN  10       GND
PIN  11      /OE

;-------- Boolean Equation Segment ------

EQUATIONS
SIG[1]:=C[1]*(SIG[8]:+:d[1])+/C[1]*I[1]
SIG[2]:=C[1]*(SIG[1]:+:d[2])+/C[1]*I[1]
SIG[3]:=C[1]*(SIG[2]:+:d[3])+/C[1]*I[2]
SIG[4]:=C[1]*(SIG[8]:+:SIG[3]:+:d[4])+/C[1]*I[3]
SIG[5]:=C[1]*(SIG[4]:+:d[4])+/C[1]*I[3]
SIG[6]:=C[1]*(SIG[5])+/C[1]*I[3]
SIG[7]:=C[1]*(SIG[6])+/C[1]*I[3]
SIG[8]:=C[1]*(SIG[7])+/C[1]*I[3]

;_________ Simulation Segment _____________

SIMULATION
trace_on clock  C[1] SIG[1..8]
setf /C[1]  OE /I[1..2] I[3]          ; Fixer l'état initial à [0001 1111].
Clockf CLOCK
setf C[1]  /d[1..4]                   ; Fixer la donnée à 0
for i:=1 to 30 do                     ;Comprime pendant 2^5 tops d'horloge.
Begin                                 ; c-a-d comprime 2^5*3 données.
  clockf clock
 end
trace_off
□
```

```
TITLE:      VALIDATION du MISR (type interne)
PATTERN:    UNIVERSITE DE SETIF
REVISION:   1.0
AUTHOR:     A. Boumaâraf
COMPANY:    logic_labo
DATE:       26/06/95
MACRO:      TEMP
```

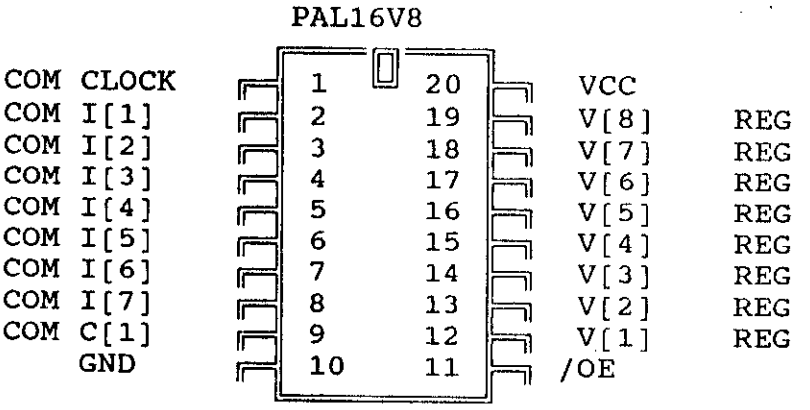


Figure 1 : Circuit du PRPG à base de PAL

```
TITLE:      VALIDATION du MISR (type interne)
PATTERN:    UNIVERSITE DE SETIF
REVISION:   1.0
AUTHOR:     A. Boumaâraf
COMPANY:    logic_labo
DATE:       26/06/95
MACRO:      TEMP
```

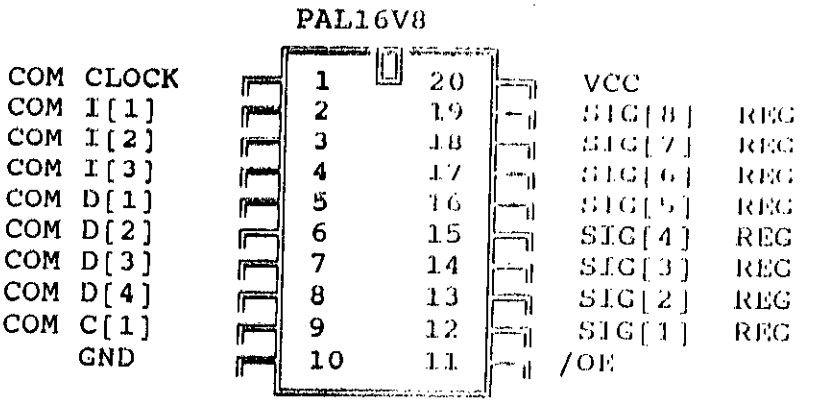


Figure 2 : Circuit du MISR à base de PAL

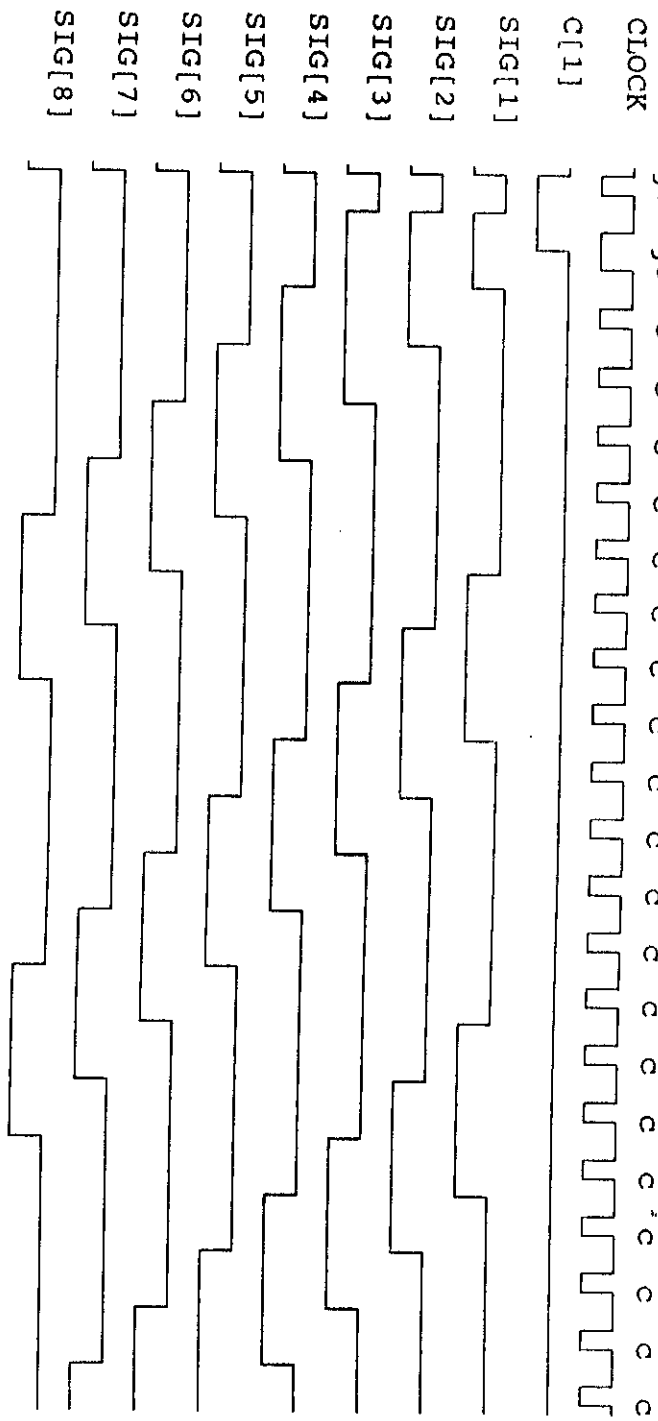


Figure 2 : Résultat de validatin de MISR

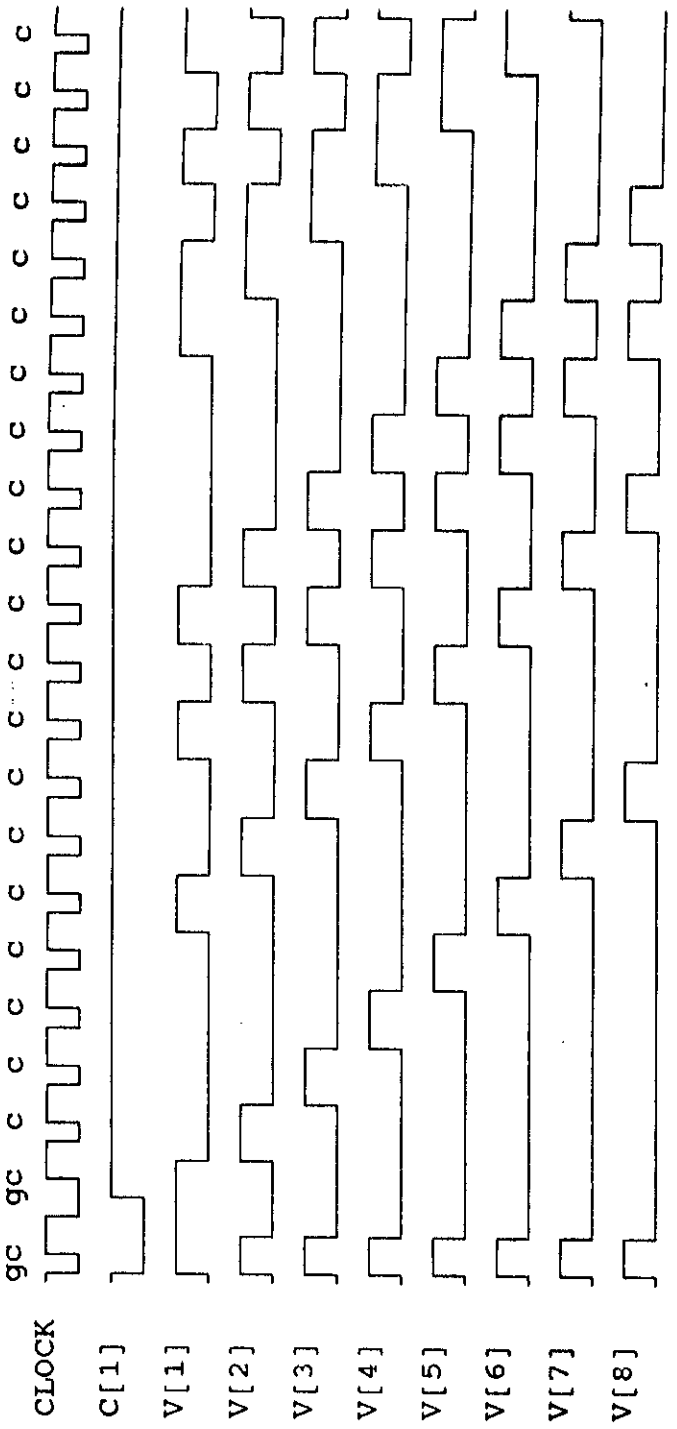


Figure 3 : Résultat de validation du PRPG