Contribution a l'évaluation de l'efficacité du test fonctionnel de microprocesseurs

Bernard Martinet

THESE

Présentée à
l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Pour obtenir le titre de DOCTEUR
(Arrêté ministériel du 23 Novembre 1988)
Spécialité : INFORMATIQUE

par

MARTINET Bernard

Contribution à l'évaluation de l'efficacité du test fonctionnel de microprocesseurs

---

Thèse soutenue le 24 Novembre 1992 devant la commission d'examen

Composition du jury :

Guy MAZARE	Président
Alain COSTES	Rapporteurs
Christian LANDRAULT	
Yves-Jacques VERNAY	Examinateurs
Raoul VELAZCO	

Remerciements

Je voudrais tout d'abord remercier Monsieur Guy MAZARE Professeur à l'INPG, d'avoir accepté de présider le jury de cette thèse.

J'adresse tous mes remerciements à Messieurs Alain COSTES Professeur à l'Institut National Polytechnique de Toulouse, et Christian LANDRAULT Directeur de recherche au CNRS, pour m'avoir fait l'honneur de rapporter sur mes travaux.

Je remercie également Monsieur Yves-Jacques VERNAY Ingénieur au Centre d'Etude des Télécommunications (Centre Norbert Segard) d'avoir accepté de faire partie de ce jury.

Que Raoul VELAZCO, sans les encouragements de qui cette thèse n'aurait jamais vu le jour, soit remercié ici. Qu'il reçoive par ces quelques mots l'expression de toute mon amitié ; le travail à ses côtés s'est toujours réalisé dans la bonne humeur malgré les embûches.

Je tiens également à remercier les ingénieurs du CNET, Messieurs Geoffroy AUVERT et Patrick RIVOIRE, et les ingénieurs de Thomson TMS, Marc SPALANZANI, Guillaume JOSSE, Eric VERLHE pour leur concours dans toute la partie technique de ces travaux. Que soit aussi remercié ici, Monsieur Jean BEAUSSE de Thomson TMS sans l'aide de qui cette expérimentation n'aurait jamais existé.

Je ne peux refermer cette page sans dire merci à tous mes collègues du Laboratoire de Génie Informatique, notamment Chantal ROBACH pour son aide dans la rédaction de ce manuscrit, Catherine BELLON avec qui j'ai souffert sur GAPT, Sylvie PONS et Solange ROCHE, pour leur aide dans la frappe de ce document mais surtout pour leur grande disponibilité et leur bonne humeur. Que tous soit assuré de ma profonde gratitude, surtout ceux avec qui j'ai pu tenir ces grandes discussions qui rendent souvent le travail si agréable… ils se reconnaîtront
.

SITUATION PARTICULIERE

PROFESSEUR D'UNIVERSITE

DETACHEMENT

ENSERG	BLIMAN	Samuel	Mutation	
ENSPG	BLOCH	Daniel	Recteur	21/12/93
ENSIMAG	LATOME	Jean-Claude	Détachement	01/05/93
ENSHMG	PIERRARD	Jean-Marie	Disponible	

RETRAITE

ENSEEG	BONNETAIN	Lucien	Mutation

PERSONNE AYANT OBTENU LE DIPLOME D'HABILITATION A DIRIGER DES RECHERCHES

BALESTRA	Francis
BALME	Louis
BECKER	Monique
BIGEON	Jean
BINDER	Zdeneck
BOE	Louis-Jean
CANUDAS DE WIT	Carlos
CHOLLET	Jean-Pierre
COEY	Jean-Pierre
CORNUEJOLS	Gérard
COURNIL	Michel
CRASTES DE PAULET	Michel
DALLERY	Yves
DESCOTES-GENON	Bernard
DUGARD	Luc
DURAND	Madeleine
FERRIEUX	Jean-Paul
FEUILLET	René
FREIN	Yannick
GAUTHIER	Jean-Paul
GHIBAUDO	Gérard
GUILLEMEOT	Nadine
GUYOT	Alain
HAMAR	Sylviane
HAMAR	Roger
HORAUD	Patrice
JACQUET	Paul
LATOMBE	Claudine
LE HUY	Hoang
LE GORREC	Bernard
LOZANO-LEAL	Rogelio
MACOVSCHI	Mihaïl
MAHEY	Philippe
METAIS	Olivier
MONMUSSON-PICQ	Georgette
MORY	Mathieu
MULLER	Jean
MULLER	Jean-Michel
NGUYEN TRONG	Bernadette
NIEZ	Jean-Jacques
PERRIER	Pascal
PLA	Fernand
RECHENMANN	François
ROGNON	Jean-Pierre
ROUGER	Jean
ROUX	Jean-Claude
TCHUENTE	Maurice

DIRECTEURS DE RECHERCHE CNRS

ABELLO	Louis
ALDEBERT	Pierre
ALEMANY	Antoine
ALLIBERT	Colette
ALLIBERT	Michel
ANSARA	Ibrahim
ARMAND	Michel
AUDIER	Marc
AUGOYARD	Jean-François
AVIGNON	Michel
BERNARD	Claude
BINDER	Gilbert
BLAISING	Jean-Jacques
BONNET	Roland
BORNARD	Guy
BOUCHERLE	Jean-Xavier
CAILLET	Marcel
CARRE	René
CHASSERY	Jean-Marc
CHATILLON	Christion
CIBERT	Joël
CLERMONT	Jean-Robert
COURTOIS	Bernard
CRIQUI	Patrick
CRISTOLOVEANU	Sorin
DAVID	René
DION	Jean-Michel
DOUSSIERE	Jacques
DRIOLE	Jean
DUCHET	Pierre
DUGARD	Luc
DURAND	Robert
ESCUDIER	Pierre
EUSTATHOPOULOS	Nicolas
FINON	Dominique
FRUCHARD	Robert
GARNIER	Marcel
GIROD	Jacques
GLANGEAUD	François
GUELIN	Pierre
HOPFINGER	Emil
JORRAND	Philippe
JOUD	Jean-Charles
KAMARINOS	Georges
KLEITZ	Michel
KOFMAN	Walter
LACROIX	Claudine
LANDAU	Ion
LAULHERE	Jean-Pierre
LEGRAND	Michel
LEJEUNE	Gérard
LEPROVOST	Christian
MADAR	Roland
MARTIN	Jean-Pierre
MERMET	Jean
MEUNIER	Gérard
MICHEL	Jean-Marie
NAYROLLES	Bernard
PASTUREL	Alain
PEUZIN	Jean-Claude
PHAM	Antoine
PIAU	Monique
PIQUE	Jean_paul
POINSIGNON	Christiane
PREJAN	Jean-Jacques
RENOUARD	Dominique
SENATEUR	Jean-Pierre
SIFAKIS	Joseph
SIMON	Jean-Paul
SUERY	Michel
TEODOSIU	Christian
VACHAUD	Georges
VAUCLIN	Michel
WACK	Bernard
YAVARI	Ali-Reza
YONNET	Jean-Paul

PERSONNE AYANT OBTENU LE DIPLOME
DE DOCTEUR D'ETAT INPG

ABDEL-RAZEK	Adel
AKSAS	Haris
ALLA	Hassane
AMER	Ahmed
ANCELLE	Bernard
ANGENIEUX	Gilbert
ATMANI	Hamid
AYEDI	Hassine Feri
A.BADR	Osman
BACHIR	Aziz
BALANZAT	Emmanuel
BALTER	Roland
BARDEL	Robert
BARRAL	Gérard
BAUDON	Yves
BAUSSAND	Patrick
BEAUX	Jacques
BEGUINOT	Jean
BELLISSENT née FUNEZ	Marie-Claire
BELLON	Catherine
BEN RAIS	Abdejettah
BERGER-SABBATEL	Gilles
BERNACHE-ASSOLANT	Didier
BEROVAL	Abderrahmane
BERTHOD	Jacques
BILLARD	Dominique
BLANC ép. FOULETIER	Mireille
BOCHU	Bernard
BOJO	Gilles
BOKSENBAUM	Claude
BOLOPION	Alain
BONNARD	Bernard
BORRIONE	Dominique
BOUCHACOURT	Michel
BRINI	Jean
BRION	Bernard
CAIRE	Jean-Pierre
CAMEL	Denis
CAPERAN	Philippe
CAPLAIN	Michel
CAPOLINO	Gérard
CASPI	Paul
CHAN-TUNG	Nam
CHASSANDE	Jean-Pierre
CHATAIN	Dominique
CHEHIKIAN	Alain
CHIARAMELLA	yves
CHILO	Jean
CHUPIN	Jean-Claude
COLONNA	Jean-François
COMITI	Jacques
CORDET	Christian
COUDURIER	Lucien
COUTAZ	Jean-Louis
CREUTIN	Jean-Dominique
DAO	Trongtich
DARONDEAU	Philippe
DAVID	Bertrand
DE LA SEN	Manuel
DELACHAUME	Jean-Claude
DENAT	André
DESCHIZEAUX née CHERUY	Marie-Noëlle
DIJON	Jean
DOREMUS	Pierre
DUPEUX	Michel
EL ADHAM	Karim
EL OMAR	Fovaz
EL HENNAWY	Adel
ETAY	Jacqueline
FABRE ép. MAXIMOVITCH	Suzanne
FAURE-BONTE ép. MARET	Mireille
FAVIER	Denis
FAVIER	Jean-Jacques
FELIACHI	Movloud
FERVAL	Haj Hassan
FLANDRIN	Patrick
FOREST	Bernard
FORESTIER	Michel
FOSTER	Panayolis
FRANC	Jean-Pierre
GADELLE	Patrice
GARDAN	Yvon
GENIN	Jacques
GERVASON	Georges
GILORMINI	Pierre
GINOUX	Jean-Louis
GOUMIRI	Louis
GROC	Bernard
GROSJEAN	André
GUEDON	Jean-Yves
GUESSOUS	Anas
GUIBOUD-RIBAUD	Serge
HALBWACHS	Nicolas
HAMMOURI	Hassan
HEDEIROS SILIVEIRA	Hamilton
HERAULT	Jeanny
HONER	Claude
HUECKEL	Tomasz
IGNAT	Michel
ILIADIS	Athananios
JANIN	Gérard
JERRAYA	Ahmed Amine
JUTTEN	Christian
KAHIL	Hassan
KUONG QUANG	Dong
KILLIS	Andreas
KONE	Ali
LABEAU	Michel

LACAZE	Alain
LACROIX	Jean-Claude
LANG	Jean-Claude
LATHUILLERE	Chantal
LATY	Pierre
LAUGIER	Christian
LE CADRE	Jean-Pierre
LE GARDEUR	René
LE NEST	Jean-François
LE THIESSE	Jean-Claude
LAMAIGNAN	Clément
LEMUET	Daniel
LEVEQUE	Jean-Luc
LONDICHE	Henry
L'HERITIER	Philippe
MAGNIN	Thierry
MAISON	François
MAMWI	Abdullah
MANTEL ép. SIEBERT	Elisabeth
MARCON	Guy
MARTINEZ	Francis
MARTIN-GARIN	Lionel
MASSE	Dominique
MAZER	Emmanuel
MERCKEL	Gérard
MEUNIER	Jean
MILI	Ali
MOALA	Mohamed
MODE	Jean-Michel
MONLLOR	Christian
MONTELLA	Claude
MORET	Frédéric
MRAYATI	Mohamed
M'SAAD	Mohamed
M'SIRDI	Kouider Nace
NEPOMIASTCHY	Pierre
NGUYEN	Trong Khoi
NGUYEN-XUAN-DANG	Michel
ORANIER	Bernard
ORTEGA MARTINEZ	Roméo
PAIDASSI	Serge
PASSERONE	Alberto
PEGON	Pierre
PIJOLAT	Christophe
POGGI	Yves
POIGNET	Jean-Claude
PONS	Michel
POU	Tong Eck
RAFINEJAD	Paiviz
RAGAIE	Harie Fikri
RAHAL	Salah
RAMA SEABRA SANTOS	Fernando
RAVAINE	Denis
RAZBAN-HAGHIGHI	Tchanguiz
RAZZOUK	Micham
REGAZZONI	Gilles
RIQUET	Jean-Pierre
ROBACH	Chantal
ROBERT	Yves
ROGEZ	Jacques
ROHMER	Jean
ROUSSEL	Claude
SAAD	Abdallah
SAAD	Youcef
SABRY	Mohamed Nabi
SALON	Marie-Christine
SAUBAT ép. MARCUS	Bernadette
SCHMITT	Jean-Hubert
SCHOELLKOPF	Jean-Pierre
SCHOLL	Michel
SCHOLL	Pierre-Claude
SCHOULER	Edmond
SCHWARTZ	Jean-Luc
SEGUIN	Jean
SIWI	Jacques
SKALLI	Abdellatif
SKALLI HOUSSEYNI	Abdelali
SOUCHON	Alain
SUETRY	Jean
TALLAJ	Nizar
TEDJAR	Farouk
TEDJINI	Smail
TEYSSANDIER	Francis
THEVENOD-FOSSE	Pascale
TMAR	Mohamed
TRIOLLIER	Michel
TUFFELIT	Denis
TZIRITAS	Georges
VALLIN	didier
VELAZCO	Raoul
VERDILLON	André
VERMANDE	Alain
VIKTOROVITCH	Pierre
VITRANT	Guy
WEISS	François
YAZAMI	Rachid

Tables des matières

Index des figures

Index des tableaux

Introduction

Depuis l'apparition des premiers microprocesseurs, jusqu'à nos jours, on a assisté à une énorme croissance de leur complexité. En effet l'évolution permanente de la technologie rend actuellement possible d'implanter près d'un million de transistors dans une "puce", ce qui permet d'intégrer l'équivalent d'un ordinateur complet sur un seul circuit. En parallèle avec l'évolution de la technologie, la nécessité de nouvelles techniques de test s'est fait sentir, aussi bien du point de vue des équipements que du point de vue des stratégies de test.
Quand au milieu des années 60, étaient posées les bases de la méthode de test par chemins sensibles (D-Algorithme), la principale préoccupation se situait au niveau des connexions. Tout mauvais fonctionnement était assimilé à un collage à 0 ou à 1 d'une des lignes du circuit. Au vu de la complexité des circuits actuels, il semble illusoire de vouloir appliquer globalement une telle méthode, à cause du nombre important d'éléments (transistors, connexions,…) à considérer. De plus, la connaissance actuelle des mécanismes conduisant à des mauvais fonctionnements montre la faible représentativité du modèle de collages.
A la fin des années 70 apparaît un nouveau concept : *bâtir un test à partir seulement des fonctions réalisées par le circuit à tester en s'affranchissant donc complètement de sa structure*.
La formidable prolifération des microprocesseurs aussi bien dans leur type que dans leur utilisation a mis ces circuits au centre du débat suivant : *comment tester de tels circuits sans s'appuyer sur leur structure ?*
De nombreuses méthodes, dites de test fonctionnel, ont été proposées par les chercheurs et les ingénieurs de test ; cependant il nous semble qu'un aspect important a été négligé : la mesure de l'efficacité de ces méthodes. Les rares tentatives effectuées dans ce domaine l'ont été à partir d'injection de collages à 0 ou à 1 dans des modèles de simulation des circuits à tester. Dans cette thèse l'efficacité du test fonctionnel sera étudiée par une méthode expérimentale basée sur l'injection de défauts dans des circuits réputés bons.

Dans la première partie sont présentées les principales méthodes de test fonctionnel de microprocesseur, ainsi que la méthode GAPT que nous avons développée. Les outils matériels et logiciels qui lui sont associés sont également présentés. Le problème de la modélisation de défauts à différents niveaux d'abstraction est abordé afin de dégager un ensemble représentatif de fautes à injecter.
La deuxième partie est consacrée à la méthode d'injection de défauts employée et à la présentation des résultats des diverses expériences réalisées sur des microprocesseurs 8 bits et 16 bits du commerce.
Au vu de ces résultats expérimentaux, dans la troisième partie, sont tirées des conclusions sur la méthode d'injection de défauts et sur l'efficacité du test fonctionnel par rapport au test structurel du fabricant. Ceci permettra d'identifier ce qui est, à notre avis, le rôle véritable du test fonctionnel : l'aide à la validation de conception et au diagnostic. Enfin une réflexion sur ce que devrait être "l'outil idéal" de test fonctionnel terminera cette étude.

Chapitre 1

Le Test Fonctionnel

I. DEFINITIONS

Vu le grand nombre de termes employés dans le domaine du test, il convient tout d'abord d'en rappeler quelques uns :

I.1. Les différents types de test

I.1.1. Qu'entendons nous par test ?

Tester est le processus visant à déterminer si un dispositif est apte à fonctionner correctement. Pour le concepteur d'un circuit intégré, ce sera le moyen d'assurer que le circuit réalisé correspond à ses spécifications initiales. Pour réaliser ce test il dispose de toutes les informations nécessaires (topologiques, électriques,…), et il a à sa disposition les équipements adéquats (microscopes optiques et électroniques, simulateurs...) lui permettant d'étudier le comportement de chaque élément du circuit.
Pour le fabricant, le test va assurer que le circuit à la sortie des chaînes de fabrication est exempt de tout défaut dû au processus même de fabrication. Il va vérifier qu'il est apte à fonctionner dans les plages prévues pour divers paramètres tels que la température, la fréquence, la consommation. Il doit donc vérifier le bon comportement à un niveau logique mais également paramétrique. Il dispose de matériels très sophistiqués pour réaliser cela (testeurs logiques/analogiques, simulateurs d'environnement sévère,…).

On distingue 3 types de test appliqués à un circuit intégré :

- le test paramétrique qui vérifie les caractéristiques statiques d'un circuit (courant, tension, fréquence…) ;
- le test dynamique qui vérifie l'évolution dans le temps des caractéristiques électriques (temps de commutation, stabilité, …) ;
- le test logique qui vérifie le comportement logique du circuit.

Sous l'étiquette test logique on regroupe en fait deux types de test de nature différente :

- Un test structurel qui prend en compte les connaissances sur l'architecture et la topologie du circuit. Défini par les fabricants, il exploite des fonctionnalités internes (autotest, test en ligne,…) souvent inaccessibles à l'usager.
- Un test comportemental ou fonctionnel, tel que peut le définir un utilisateur du circuit, c'est à dire un test de "boite noire" dont on ne connaît le fonctionnement qu'en présence de certains stimuli et que l'on ne peut vérifier qu'en contrôlant son comportement en fonction de l'application de ces stimuli.

Dans ce travail nous ne parlerons pas de test fonctionnel structurel, et sauf spécification particulière, on assimilera les termes fonctionnel et comportemental.

I.1.2. Défauts, fautes, erreurs

En ce basant sur la terminologie développée en [Lap85] nous allons ici définir le vocabulaire employée dans le reste de cette thèse.
Un défaut est la cause phénoménologique d'un mauvais fonctionnement.
Une faute est la modélisation d'un défaut à quelque niveau que ce soit. Au chaque niveau de modélisation : physique, logique, fonctionnel correspondra donc une faute physique, une faute logique ou une faute fonctionnelle. Afin de replacer ces termes dans le vocabulaire couramment employé, le terme "panne" utilisé pour désigner la manifestation d'un défaut au niveau logique correspond donc à une faute logique.
Une erreur est la manifestation du défaut à l'extérieur du circuit.

Pour ne pas trop alourdir la terminologie, le défaut et la faute physique qu'il induit sont souvent confondus. Par exemple dans le cas d'une coupure de piste (faute physique), le défaut est en fait la poussière qui à empêché localement de dépôt du matériau conducteur de la connexion. Généralement, on considère que la coupure est le défaut.
Si on prend un additionneur au sein d'un processeur par exemple, la coupure d'une piste est un défaut, le collage à zéro (ou a un) qui pourrait en résulter sur une ligne particulière est la faute logique induite (que l'on appelle souvent panne logique), le mauvais résultat en sortie de l'additionneur est un faute fonctionnelle, et le mauvais adressage par exemple, induit par cette faute d'addition sera l'erreur.

Nous faisons encore une distinction supplémentaire entre les fautes paramétriques et les fautes booléennes. On considère comme paramétrique toute faute qui ne donnera lieu à une erreur que sous certaines conditions d'environnement (courant, tension, température, contrainte mécanique,…). Par exemple, pour une porte NOR, si la vérification de sa table de vérité donne des résultats corrects en appliquant les valeurs appropriées sur ses entrées, elle sera bonne du point de vue fonctionnel, même si pour d'autres conditions de tension ou d'intensité ses sorties seront erronées. Dans ce cas, elle sera fautive du point de vue paramétrique.
Dans notre cas on ne s'attachera donc qu'a vérifier le bon fonctionnement d'un microprocesseur à des fréquences moyennes, sans s'intéresser aux fréquences limites hautes ou basses. Il en sera de même pour d'autres paramètres (tensions d'alimentations, température,…).

I.2. Efficacité d'un test

I.2.1. Généralités

Pour un lot de circuits intégrés, le but d'un test est de séparer les circuits qui fonctionnent correctement : les circuits bons, des circuits qui ne réalisent pas la fonction pour laquelle ils ont été conçus : les circuits défectueux. La question que l'on doit se poser est : *quel degré de confiance peut-on accorder à ce tri ?*
Afin d'établir quelques notions importantes nous allons nous servir de certaines des définitions données en [Jac89] et illustrées par la figure 1.1.

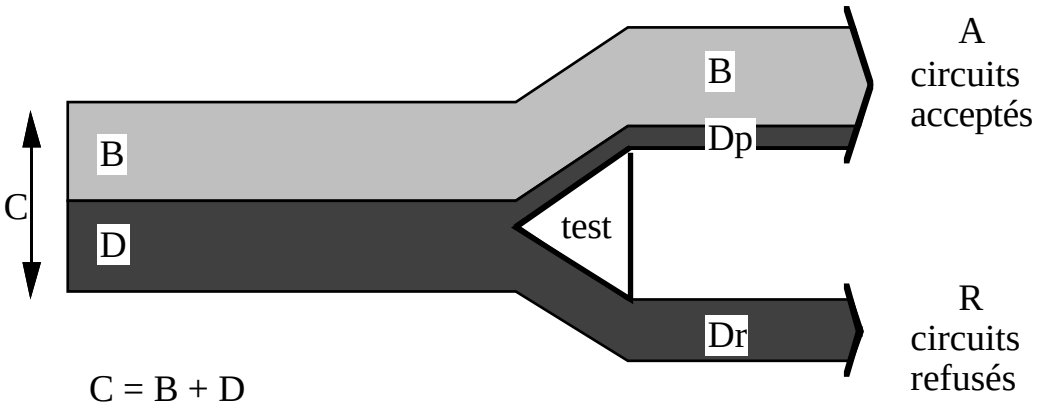


Figure 1.1 : le principe du test de circuits intégrés

Soit un ensemble de C circuits à tester. Il est en fait composé des deux sous-ensembles constitués respectivement de B circuits bons et de D circuits défectueux. Le test va donc essayer de séparer correctement ces deux sous-ensembles.

Nous considérons les hypothèses suivantes qui sont en pratique souvent vérifiées.

Hypothèse 1 : la séquence de test est imparfaite.

Hypothèse 2 : aucun circuit bon ne sera reconnu défectueux par la séquence de test.

Hypothèse 3 : les équipements de test (testeurs) ne sont pas défectueux.

Ces hypothèses étant admises, nous pouvons faire ressortir plusieurs relations de la figure 1.1. L'imperfection du test (hypothèse 1) fait qu'une partie Dp des D circuits défectueux va être acceptée par le test, l'autre partie Dr étant normalement refusée.

A = B + Dp : l'ensemble des A circuits acceptés par le test est formé des B circuits bons et du sous ensemble des Dp circuits défectueux reconnus à tort comme bons.
R = Dr : l'ensemble des R circuits refusés est uniquement constitué du sous-ensemble des Dr circuits défectueux effectivement rejetés par le test.

Deux autres rapports sont généralement calculés, ce sont les rapports suivants :
- **B+Dr / C** soit le taux de circuits correctement testés,
- **B / A** soit le taux de circuits acceptés réellement bons.

C'est ce dernier rapport qui concerne surtout les utilisateurs. Il faut rajouter un autre indicateur qui intéresse les fabricants : le rendement qui est le rapport **A / C**

L'imperfection du test est caractérisée par Dp ; plus Dp est petit, plus le test est efficace. Le rapport **Dp / C** appelé taux d'échappement (escape rate) est utilisé généralement par les fabricants pour qualifier leurs tests. Cette valeur est donnée en nombre de pièces par million (p.p.m.)[1]. L'objectif des fabricants est bien entendu le "zéro défaut".

Malheureusement, le degré de confiance que l'on peut accorder à un test ne peut être correctement évalué qu'**a posteriori**, lorsque l'on connaît exactement le taux de retour client , c'est à dire le rapport entre le nombre de pièces commercialisées comme bonnes détectées défectueuses par un utilisateur et le nombre total de pièces commercialisées. Le fabricant assimile cette valeur au taux d'échappement **Dp / C** alors que cette valeur ne peut représenter qu'une borne minimale, si on admet que tous les circuits défectueux commercialisés ne font pas

[1] *Les valeurs couramment citées vont de quelques dizaines à quelques centaines de p.p.m.*

forcément l'objet d'un retour client. Il suffit pour cela de penser à certaines fautes paramétriques qui ne sont mises en évidence que sous des conditions particulières.

Afin de fixer **à priori** une mesure de confiance dans l'efficacité d'un test, on calcule généralement le taux de couverture de fautes défini comme le rapport entre le nombre de fautes détectées par une séquence de test et le nombre total de fautes. Ce rapport est calculé à partir d'hypothèses d'injection de fautes sur un modèle du circuit à tester. Ce que l'on va mesurer en fait, par des simulations sur ces modèles, c'est *la fraction détectée de l'ensemble des fautes hypothétiques que l'on aura créé*.
Pour pouvoir corréler le taux de couverture de fautes avec le taux d'échappement il manque certaines valeurs difficiles à évaluer, à savoir : les probabilités d'apparition des différentes fautes. Les calculs, lorsqu'ils existent, se basent généralement sur deux hypothèses : l'équiprobabilité des fautes et l'unicité des fautes. Bien que très intéressantes pour les calculs théoriques, dans la pratique ces deux hypothèses n'ont en fait que peu de chance d'être vérifiées[2].

Pour illustrer ceci, prenons un exemple avec des valeurs hypothétiques. Quelle confiance faut-il accorder à un test qui détecte 99% de fautes de collage, si on suppose que les fautes de collage ne représentent que 50% des fautes possibles ? Pour estimer cette confiance, nous sommes devant des inconnues du type : quel est le taux d'apparition des défauts conduisant ou pas à des fautes modélisable par des collages ? De même, pour les 1% de fautes de collage non détectées, quelle est la probabilité pour qu'un défaut se traduise effectivement par ce type de collage particulier ?
Même si l'on admet qu'on peut posséder un modèle parfait d'un circuit et un modèle parfait des fautes possibles du circuit, seuls des calculs basés sur la probabilité d'apparition des différentes fautes pourraient donner une idée exacte du taux d'échappement. La seule possibilité qui nous est offerte afin de donner des valeurs à ces probabilités serait l'analyse statistique d'un très grand nombre de circuits défectueux issus des ensembles Dr (défectueux refusés) et Dp (défectueux acceptés).
Les probabilités effectives d'apparition des différents défauts n'auront plus d'importance sur le calcul du taux d'échappement quand le "zéro défaut" sera effectivement atteint. Encore faudra-t-il être sûr que toutes les fautes possibles sont effectivement contrôlées.

I.2.2. Problème du test fonctionnel

Malgré les réserves émises ci-dessus, on évalue de nos jours l'efficacité de tests par des taux de couverture mesurés sur des modèles. Puisque le nombre de fautes possibles est trop important, seule une modélisation des fautes les plus représentatives peut permettre d'établir des calculs. Pour les fautes électriques ceci est fait en utilisant des modèles de bas niveau (électrique et topologique par exemple), proches de l'implantation réelle du circuit. Pour les fautes logiques on définit un modèle logique du circuit et un ensemble de fautes logiques. Si dans les années 70 on se contentait de raisonner en logique 0 ou 1, les simulateurs actuels introduisent en plus des niveaux 0 et 1, des niveaux transitoires X (inconnu) ou φ (indéterminé) et même tout un système de pondération (0 faible, moyen, fort,…) permettant de s'approcher le plus près possible du comportement électrique du circuit. Les modèles de fautes qui en découlent sont de plus en plus complexes et essaient le plus souvent de prendre en compte des notions de topologie.

Pour un test fonctionnel le problème est plus complexe. On peut, bien sûr, envisager le même type de réponse, à savoir : l'efficacité d'un test fonctionnel est le nombre de fautes fonctionnelles mises en évidences par rapport au nombre total de fautes fonctionnelles d'un

---

[2] *Les types de fautes dépendent fortement du processus de fabrication, certaines phases étant plus délicates que d'autres .*

modèle. Mais si au niveau physique les phénomènes sont à peu près connus, comment déterminer l'ensemble total de fautes au niveau fonctionnel ? De nombreux travaux ont été consacrés à la définition de modèles de circuits et de modèles de fautes, mais en se servant d'hypothèses toujours difficiles à vérifier si on se place devant une boîte noire. Si nous reprenons l'exemple de l'additionneur, comment juger de l'ensemble des fautes fonctionnelles possibles sans prendre en compte des caractéristiques de son architecture (structure parallèle ou série, avec ou sans anticipation de retenue,…) ? La seule démarche sûre serait de vérifier que le résultat de l'addition de A avec B donnera bien A+B pour toutes les valeurs possibles de A et de B. Il restera encore à résoudre le problème des séquences d'additions puisque l'on sait que pour certaines technologies (CMOS par exemple), un circuit combinatoire peut se transformer en circuit séquentiel en présence de certains défauts [Wad78].

II. LE TEST FONCTIONNEL DE MICROPROCESSEURS

Depuis la fin des années 1970, le test fonctionnel a été proposé comme une solution possible au problème de la génération de test pour les circuits complexes (VLSI) et en particulier pour les microprocesseurs. Depuis, certains auteurs l'ont même présenté comme la solution de remplacement des méthodes structurelles pour la détection de fautes de conception et de fabrication des VLSI [Lai83] ; d'autres le considéraient comme la seule solution d'avenir au test des microprocesseurs futurs [Cor87]

II.1. Les méthodes de test fonctionnel

Le test fonctionnel est un test comportemental, c'est à dire un test généré a partir des fonctions que le circuit réalise et ce, indépendamment de sa structure interne. Un tel test est censé utiliser les description du jeu d'instructions et les renseignements d'architecture donnés par le manuel utilisateur du microprocesseur fourni par le constructeur. Ce type de test a été très largement étudié depuis 1978 ; les méthodes proposées visent à permettre le test d'un microprocesseur bien que sa structure fine soit inconnue.
On peut distinguer deux familles de méthodes :
- les méthodes de test par distinction, basées sur des hypothèses de fautes fonctionnelles ;
- les méthodes de test par identification, inspirées du principe de vérification expérimentale ("checking experiment"), qui vérifient que l'exécution des fonctions spécifiées est correcte, sans hypothèse sur les erreurs possibles.

II.1.1.Test comportemental par distinction

Ces méthodes visent à distinguer le circuit sans fautes d'un ensemble de circuits fautifs possibles. Les circuits fautifs sont déduits d'un ensemble d'hypothèses de fautes fonctionnelles. La génération d'un test consiste à trouver les ensembles d'activations qui mettent en évidence chacun des comportements fautifs.
Ce type de méthode suit l'approche classique d'un test au niveau schéma en portes logiques qui utilise un modèle de collages en tant que modèle de fautes (D-Algorithme). A un niveau fonctionnel, les modèles de fautes de bas niveau vont être remplacés par des modèles de fautes fonctionnelles. Les grandes lignes de cette approche ont été données par les travaux de Thatte et Abraham [Tha78]. Cette approche a d'abord conduit à un modèle d'erreur fonctionnelle, et un ensemble de procédures de test ayant pour but la mise en évidence de ces erreurs. Une modélisation du processeur sous forme de graphe (S-graph) a été ensuite proposée [Tha80]. Ce graphe modélise le flot de données dans un microprocesseur et peut être utilisé pour obtenir un ensemble ordonné d'activations de test, qui correspond aux hypothèses de fautes.
De nombreux travaux visant a appliquer partiellement ou totalement cette méthode à des circuits hypothétiques ou standards ont été développés. De nombreuses approches dérivées, dont certaines utilisent les mêmes modèles de circuit et de fautes ont également été développées . Ces méthodes ont pour objectif soit la réduction de la longueur du test, soit l'exploration de variantes des modèles ou des algorithmes des séquences de test [Lin80] [Lin82] [Min82] [Sal83] [Fra84] [Kil86].
Parmi ces travaux on peut remarquer ceux de Thevenod-Fosse et David, qui portent sur le test aléatoire de microprocesseurs. A partir du S-graph et des hypothèses de fautes proposées par Abraham et Thatte ces auteurs déterminent la faute fonctionnelle la plus difficile à détecter afin de calculer la longueur de la séquence aléatoire de test mettant en évidence cette faute avec une probabilité fixée [The81] [The83].
D'autres niveaux de modélisation ont été proposés, notamment la modélisation du fonctionnement du microprocesseur au niveau instruction, micro-instruction, ou de transfert de

registres, toujours en accord avec un modèle d'erreur défini au niveau correspondant (la contribution la plus importante à ce niveau a été celle de Su [Su82] [Su84] [Su85]. Dans toutes ces approches le centre d'intérêt a été la détermination d'un test minimal (le test minimum étant la vérification de la correction des accès en lecture et écriture aux registres internes du processeur à tester) [Su84][Bra84].

II.1.2.Test comportemental par identification

Les méthodes par identification ont pour objectif la vérification de la conformité du circuit sous test avec ses spécifications fonctionnelles. Aucun modèle de fautes n'est nécessaire. La génération d'un test par identification consiste à déterminer l'ensemble des fonctionnalités représentatives du circuit sous test et de construire une séquence de test pour chacune d'elles. Pour un microprocesseur par exemple, les fonctionnalités représentatives sont en fait fixées par les diverses instructions possibles.
Cette méthode est l'extension de la méthode de test par identification d'automates ("checking experiment") étudiée pour les machines d'états finis élémentaires [Bel84a]. En [Jai84] Jain et Susskind ont montré que l'on pouvait étendre la méthode pour traiter des circuits complexes tels que les microprocesseurs.
Les méthodes par identification avaient été au préalable proposées par Robach & Saucier [Rob80] puis Annaratone et Sami [Ann82]. Ces différents auteurs proposent une modélisation pour chaque type d'instruction, permettant de déduire un ordonnancement des séquences de test, par rapport à des critères donnés ("start big" ou "start small").

II.1.3. Méthodes de test Ad hoc

Parallèlement à ces méthodes, des approches ad hoc ont été développées. Toutes ces approches nécessitent une compréhension préalable de l'architecture du circuit à tester avant la définition de la stratégie de test. En cela, elles ne rentrent pas vraiment dans notre définition de départ c'est-à-dire le test à partir d'informations fournies uniquement par le manuel utilisateur ; cependant, le but recherché ici est également la bonne exécution des fonctions décrites dans le manuel du circuit. A partir d'une décomposition en sous-ensembles fonctionnels (UAL, bus, registres ...) [Chi76] ou à partir d'un partitionnement en fonction de la complexité des instructions [Rob80], ces méthodes essaient d'obtenir une activation la plus complète possible du circuit indépendamment de toute hypothèse d'erreur. Des techniques appliquées à la construction de programmes pour des exemples particuliers ont été données en [Mar88][Hen86].

II.2. Les problèmes liés a ces méthodes

II.2.1. Test par distinction

Pour le test par distinction deux problèmes se posent : celui de la validité de la représentation du circuit et celui des modèles de fautes employés.

II.2.1.a Les modèles fonctionnels

Depuis la fin des années 1980, la diminution du nombre de publications sur le test fonctionnel parues dans les congrès spécialisés montre que l'attraction exercée par celui-ci a fortement décru. Ceci peut s'expliquer par plusieurs raisons :
• une analyse des publications parues à ce jour montre que l'utilisation d'un modèle fonctionnel formel sous forme de graphe (S-graph [Tha80], graphe d'exécution réduite [Rob80]...) ne visait qu'à :

- exprimer la contrôlabilité et l'observabilité des registres internes du circuit à tester [Tha80] [Lin80] [Sal83] [Fra84] [Kil86] ;

- permettre de déterminer l'ordonnancement des instructions en vue du test [Rob80] [Ann82].

En général, ce n'est généralement pas ce type de difficulté que l'on rencontre lorsque l'on établi un test pour un microprocesseur.

- L'instruction, ou la séquence d'instructions la plus simple permettant de charger ou de lire le contenu de chaque registre interne est très simple à trouver. L'évolution des microprocesseurs récents va vers une standardisation de l'accès aux registres internes, ceci étant encore plus flagrant lorsque le nombre de registres augmente.
- Vouloir ordonner en vu du diagnostic, en utilisant un modèle du circuit simplifié basé sur un nombre réduit d'hypothèses, semble illusoire. En effet, ces modèles ne prennent en compte généralement que la partie chemin de données, alors qu'il est facile de se rendre compte, au vu du plan de masse d'un circuit, de la grande proportion du silicium occupée par la partie contrôle (environ 50 % pour les processeurs CISC).

• Bien que très attractifs, les modèles proposés (que ce soit sous forme de graphe, de réseaux de Pétri ou de descripteur de type transfert de registre) sont difficilement applicables aux microprocesseurs 32 bits actuels, la quantité d'information étant prohibitive.

Pour le test aléatoire, le problème qui se pose est de trouver la faute la plus difficile à tester. Pour cela, les mêmes critiques que pour la représentativité du S-graph peuvent être avancées.

II.2.1.b Les hypothèses d'erreurs

En ce qui concerne les hypothèses d'erreurs fonctionnelles, les principales réserves s'appliquent à la nature même de ces hypothèses.
Si le principe d'une modélisation de haut niveau est généralement admis, l'introduction de fautes de bas niveau dans ces modèles est plus controversée [Mal87]. En effet, un ensemble de résultats obtenus lors d'expériences concrètes de diagnostic [Vel85] [Vel87a] [Vel88] a mis en évidence, pour des cas particuliers, la non représentativité des fautes physiques avec les modèles couramment proposés. Les informations sur lesquelles se basent les modèles ne prenant pas en compte la topologie du circuit, il est difficile de déduire les conséquences à un niveau fonctionnel d'une hypothèse de faute faite à un niveau physique [Gal80] [Hay85]. Miczo en [Mic82], a montré que la validité de résultats obtenus par simulation de fautes sur des schémas logiques équivalents est contestable (le test de 100 % des collages d'une implantation NAND/NAND d'un multiplexeur 4 bits ne détecte que 50 % des collages d'une même fonction en NOR/NOR). Ces aspects seront plus longuement développés au paragraphe 2.4.

Au niveau des contrôleurs des microprocesseurs, les hypothèses concernant l'exécution des instructions proposées par Thatte et Abraham [Tha80] correspondent probablement à une grande partie des mauvais fonctionnements :

- exécution d'une instruction I à la place d'une instruction J ;
- exécution d'une instruction I en plus d'une instruction J ;
- aucune exécution d'instruction au lieu de l'instruction J.

Néanmoins, au vu du grand nombre d'instructions des microprocesseurs actuels, il est pratiquement impossible de trouver une séquence de test garantissant l'absence de ces erreurs. Enfin, ces hypothèses ont été critiquées parce qu'elles ne prennent pas en compte l'exécution

partielle d'instructions (exécution d'une partie d'instruction à la place d'une autre partie, de la même ou d'une autre instruction ...).

En ce qui concerne le test aléatoire, l'applicabilité de telles méthodes à des microprocesseurs actuels reste à prouver. En effet, lorsque l'on sait qu'une détection proche de 100 % pour un microprocesseur 8 bits tel que le 6800 nécessite un test de $6,5 \times 10^6$ instructions [Fed84], l'extension à des microprocesseurs de complexité plusieurs fois supérieures tels que 68000 ou 68020 laisse sceptique. De plus il faut développer des machines de test dédiées, si l'on veut utiliser au mieux ce type de test et tirer quelques bénéfices de la détection d'erreurs notamment au niveau du diagnostic.

II.2.2. Le test ad hoc

Dans toutes les approches de test ad hoc connues à ce jour [Hen86][Mar88], la nécessité d'activer plusieurs fois chacune des instructions est une constante, bien que la signification accordées au terme "instruction" reste souvent très vague. Si pour le 6800, le jeu d'instructions complet (combinaison de tout code opération et mode d'adressage possible) représente 197 instructions, ce nombre croît très rapidement pour les microprocesseurs 32 bits. Pour illustrer ceci on notera que le seul mode d'adressage "address register indirect with index (base displacement)" du 68020 représente 3487 cas ; le développement complet de l'instruction d'addition ADD avec toutes les tailles d'opérandes, dans tous les modes d'adressage, avec tous les registres (base et index) et toutes les échelles (scaling) possibles conduit à environ 13×10^6 instructions. On ne pourra donc dans ce cas que tester un sous-ensemble de toutes les combinaisons possibles ; le problème de le constitution de ce sous ensemble reste entier. Aucune solution générale n'est applicable et chaque processeur est un cas spécifique dont l'analyse va croître avec la complexité.

II.2.3. Le test par identification

Les critiques précédentes s'appliquent également au test par identification ; on se heurte généralement à la longueur excessive des programmes de test et au manque de généralité des méthodes proposées.

II.3. La méthode de test GAPT

La méthode GAPT pour le test fonctionnel de microprocesseur à été déjà largement décrit [Bel82] [Bel84a], c'est uniquement en temps qu'outil support d'expérience qu'elle sera considérée ici.

La méthode GAPT est une méthode de test par identification dont les points forts sont :

- l'absence d'hypothèse de fautes fonctionnelles, comme pour toute méthode par identification.
- une possibilité de test pseudo-exhaustif des divers blocs architecturaux du circuit,
- l'existence de moyens logiciels et matériels opérationnels associés [Bel84c] [Bel85].

II.3.1. Présentation de la méthode GAPT

Cette méthode a comme caractéristiques :

- de prendre en compte l'ensemble du fonctionnement du microprocesseur, bien que pour les signaux asynchrones cela s'avère parfois très délicat [Bel84b] ;
- de pouvoir être étendue pour traiter tout circuit programmable, c'est à dire tout circuit possédant un jeu d'instruction (processeur, coprocesseur, …)
- de permettre la génération automatisée des programmes de test.

On vérifie le bon fonctionnement d'un microprocesseur en s'appuyant sur le partitionnement du microprocesseur en deux parties : un contrôleur et des ressources constituant la partie opérative. On applique donc deux tests :

- le test de la partie contrôle, qui active les fonctions du circuits (instructions, signaux asynchrones), qui est appelé : test de conformité ;
- le test de la partie opérative, qui vérifie les différents modules de la partie opérative (registres, opérateurs, bus,…), qui est appelé : test de balayage

II.3.1.a Test de conformité

Le test de conformité vérifie la bonne exécution d'un ensemble de fonctionnements représentatifs du microprocesseur. On s'intéresse aux fonctions exécutables par le microprocesseur entre la lecture de deux codes-opération successifs.
L'état comportemental d'un microprocesseur est défini par la valeur des éléments de mémorisation interne décrits dans le manuel utilisateur. Ces éléments de mémorisations seront représentés par :

- les registres adressables (registres données, registres adresses, registre d'état) ;
- le registre instruction et le compteur programme.

L'état comportemental est défini à la fin de l'exécution d'une instruction ou après la prise en compte d'un signal asynchrone externe.
Chaque fonctionnement à tester est vérifié par un module élémentaire de la forme :

- initialisation de l'état comportemental du microprocesseur ;
- activation du fonctionnement à tester et observation des sorties ;
- observation de l'état comportemental du microprocesseur.

L'observation systématique de l'état comportemental permet de vérifier que :

- les éléments de mémorisation qui doivent être modifiés le sont correctement ;
- les autres éléments de mémorisation ne sont pas altérés.

Les opérandes à charger dans les éléments de mémorisation au cours de l'initialisation et à utiliser au cours du fonctionnement à tester pourront être aléatoires ou déterministes (déduits d'hypothèse sur le fonctionnement à tester).

II.3.1.b Test de balayage

Le but du test de balayage est de vérifier la partie opérative du microprocesseur. Celle-ci sera partitionnée en un ensemble de blocs fonctionnel constitué par :

- les registres programmables ;
- les opérateurs : unité arithmétique et logique, unité de calcul d'adresse, décaleurs, unité d'incrémentation des différents registres compteurs,…;
- l'interconnexion entre les blocs (bus).

Les opérandes pour tester chaque bloc pouvant être là encore aléatoires ou déterministes.
Au cours du test de balayage on cherche uniquement à vérifier le bon fonctionnement d'un bloc et non l'exécution d'une fonction. Seuls les registres source seront initialisés et on n'observera que les registres qui mémoriseront le résultat.
Le module de base du test de balayage aura donc la forme suivante :
- initialisation des registres sources ;
- activation de l'instruction permettant de vérifier au mieux le fonctionnement du bloc ;
- observation des registres destination.

Né dans son concept au début des années 1980, le système GAPT est opérationnel depuis 1985 tant sur le plan matériel que logiciel. Sa flexibilité a permis son amélioration au cours des années au vu des différentes expériences de test de microprocesseur ou de diagnostic de circuits fautifs réalisées.
Le système GAPT (figure 1.2) se compose d'un logiciel de génération automatique de programme de test, et d'un ensemble de testeurs développés spécialement pour des applications de test fonctionnel. Quel que soit le testeur utilisé, le développement d'une carte d'interface est nécessaire pour chaque nouveau circuit à tester.

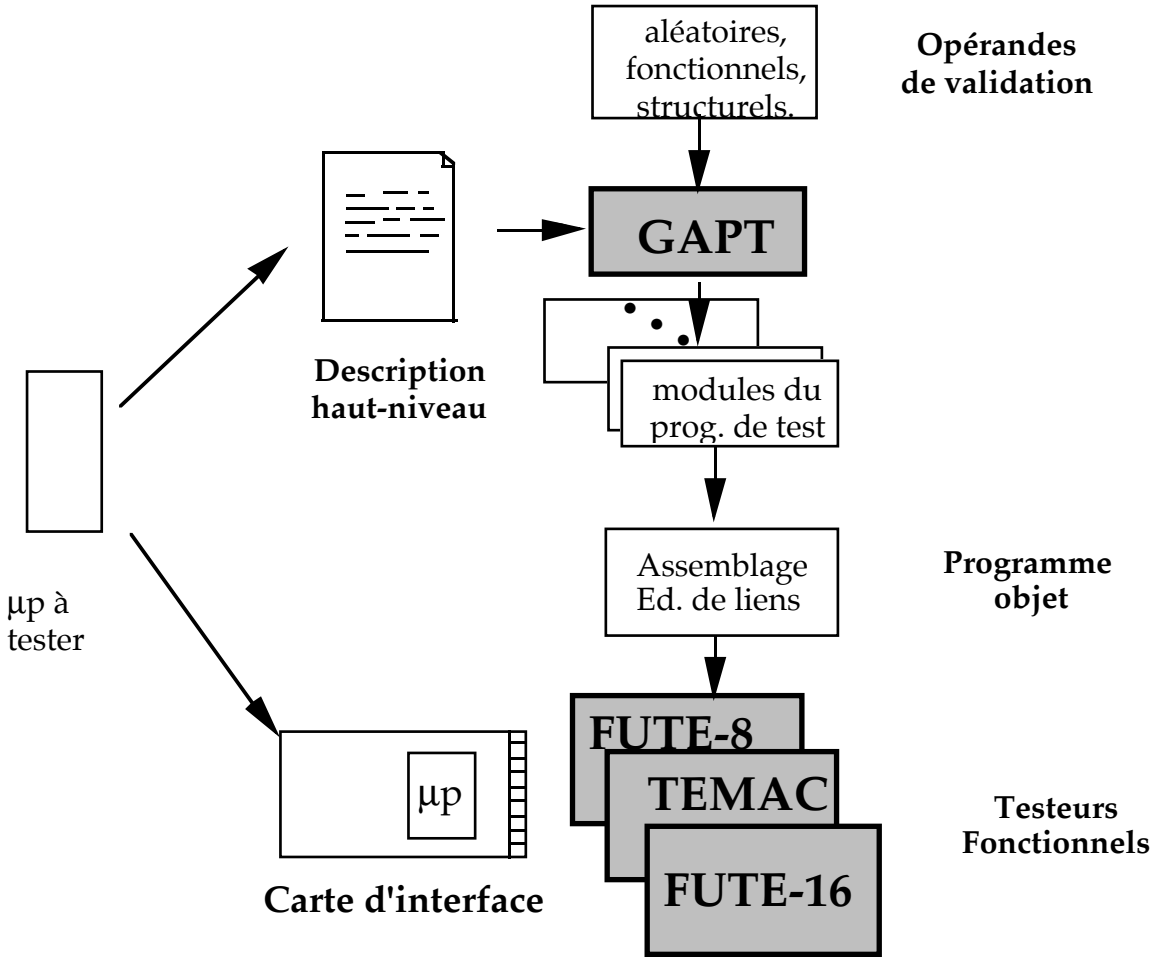


Figure 1.2 : la chaîne de test fonctionnel GAPT

II.3.2. Description du logiciel GAPT[3]

A partir d'une description compacte du circuit à tester, établie à partir du manuel utilisateur, le logiciel GAPT permet la génération de programmes de test en langage assembleur (figure 1.3). Cette définition se fait dans un langage permettant une description "haut niveau" des différents éléments constituant le microprocesseur (le langage GAPT).

[3] *On trouvera en annexe une partie du manuel utilisateur [Mar89]*

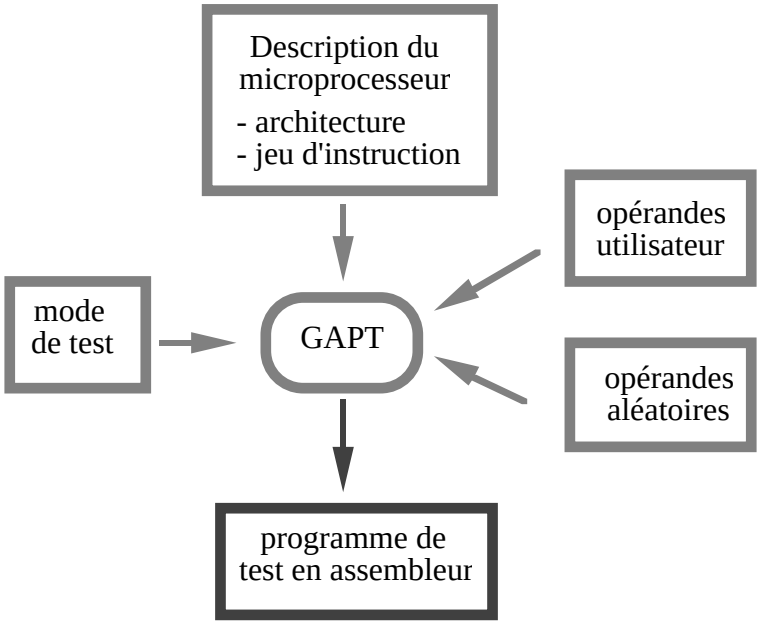


Figure 1.3 : fonctionnement du logiciel GAPT

Une description GAPT d'un microprocesseur contient toutes les informations nécessaires à la génération d'un programme de test.
Ce sont des informations sur :

- "l'architecture" du circuit (registres, groupe de registres, adressages ...),
- le jeu d'instruction.

L'utilisateur se voit offrir la possibilité de définir des opérateurs et des opérandes ad hoc pour tester des instructions particulières, ou d'utiliser des opérandes aléatoires ou aléatoires par parties (parties fixes - parties aléatoires). Plusieurs modes de test permettent de s'adapter au microprocesseur sous test.
Les programmes générés se présentent sous forme de concaténation de modules élémentaires de la forme :

- initialisation des registres internes ;
- activation d'une instruction ;
- observation des registres internes.

Lorsque l'on réalise un test à l'aide de GAPT, plusieurs modes de génération sont à la disposition de l'utilisateur. Chacun de ces modes à une puissance et un domaine d'utilisation différent.

II.3.2.a Le mode conformité

Le test de la partie contrôle, qui active les fonctions du circuit (instructions) est réalisé en mode **conformité**.
Dans ce mode **tous** les registres internes seront initialisés avant l'exécution de l'instruction, et **tous** les registres internes seront observés après. Ce mode, qui conduit à des programmes de taille très importante lorsque le nombre de registres internes augmente, est difficilement utilisable. Par contre, il présente un énorme avantage dans le cadre d'un diagnostic. En effet, chaque module élémentaire est totalement indépendant des autres modules. Il sera dans ce cas plus aisé, lors d'une phase de diagnostic, de proposer des hypothèses de fautes, au vu des résultats obtenus, et d'essayer de les confirmer par de nouveaux test (hypothèse a posteriori)[Abr82][Vel85].

Exemple du 68000
instruction d'addition en format Word - adressage : indirect mémoire, registre données ;
forme du module élémentaire de conformité :

```
        MOVEA.L     #H'B00A41CF,A7      ⎤
        MOVEA.L     #H'92BCC411,A6      ⎥
        MOVEA.L     #H'A031C9B8,A5      ⎥
        MOVEA.L     #H'5E09D830,A4      ⎥   initialisation de tous les registres d'adresses
avec
        MOVEA.L     #H'96BBD41A,A3      ⎥   une valeur aléatoire
        MOVEA.L     #H'B0FD5B54,A2      ⎥
        MOVEA.L     #H'994F285E,A1      ⎦
        LEA         (ZDL).L,A0              initialisation du registre
source avec l'adresse
    ;
        mémoire du 2ème opérande
        MOVE.L      #H'9A2341E2,D7      ⎤
        MOVE.L      #H'FFA5A75C,D6      ⎥
        MOVE.L      #H'0D37C4A6,D5      ⎥
        MOVE.L      #H'E359B840,D4      ⎥   initialisation de tous les registres
de données avec
        MOVE.L      #H'A68B94AA,D3      ⎥   une valeur aléatoire
        MOVE.L      #H'9F4D0064,D2      ⎥
        MOVE.L      #H'5A1FD5EE,D1      ⎦
        MOVE.L      #H'00005555,D0          initialisation du registre source
avec la valeur
    ;
        du 1er opérande (ici valeur déterministe)
        MOVE        #H'D321,CCR             initialisation du registre code
condition
    ;
        (valeur aléatoire)
        ADD.W       (A0),D0                 instruction à
tester
        MOVE        SR,(ZDL+2).L            observation du registre d'état
        MOVE.L      A7,(ZDL+4).L        ⎤
        MOVE.L      A6,(ZDL+8).L        ⎥
        MOVE.L      A5,(ZDL+12).L       ⎥
        MOVE.L      A4,(ZDL+16).L       ⎥
        MOVE.L      A3,(ZDL+20).L       ⎥
        MOVE.L      A2,(ZDL+24).L       ⎥
```

```
        MOVE.L      A1,(ZDL+28).L   |
        MOVE.L      A0,(ZDL+32).L   |
        MOVE.L      D7,(ZDL+36).L   |   observation de tous les registres de données
        MOVE.L      D6,(ZDL+40).L   |   et d'adresses
        MOVE.L      D5,(ZDL+44).L   |
        MOVE.L      D4,(ZDL+48).L   |
        MOVE.L      D3,(ZDL+52).L   |
        MOVE.L      D2,(ZDL+56).L   |
        MOVE.L      D1,(ZDL+60).L   |
        MOVE.L      D0,(ZDL+64).L   ⌋
```

total : 35 instructions
taille : 204 octets de programme et 68 octets de données ou de résultats

II.3.2.b Le mode de pseudo_conformité

Un mode de **pseudo-conformité** présentant la même puissance de détection que le test de conformité mais générant des programmes de taille inférieure est en fait employé pour le test de la partie contrôle.

Dans ce mode, le module élémentaire se présente sous la forme suivante :

- observation de **tous** les registres source ou destination de l'instruction à tester,
- initialisation des registres source,
- activation de l'instruction.

Exemple du 68000
Instruction d'addition en format Word - adressage : indirect mémoire, registre données ;
forme du module élémentaire de pseudo-conformité :

```
        MOVE        SR,(ZDL+2).L        ⎤  observation des registres source ou
destination
        MOVE.L      A0,(ZDL+4).L        ⎥  de l'instruction
        MOVE.L      D0,(ZDL+8).L        ⎦
        LEA         (ZDL).L,A0          ⎤  initialisation des registres
source de l'instruction
        MOVE.L      #H'00005555,D0      ⎦
        ADD.W       (A0),D0                 instruction à
tester
```

total : 6 instructions
taille : 32 octets de programme et 12 octets de données ou de résultats

Par ce biais on réduit donc le nombre de registres initialisés ou observés. L'observation d'une erreur éventuelle se fera dans un module plus éloigné, lorsqu'un registre comportant une valeur erronée sera de nouveau observé.

Exemple du 68000 :

Soit une erreur d'addition qui ne se manifeste qu'avec le mode d'adressage indirect par registre

```
        MOVE        SR,ZDL+96,L         module comprenant l'instruction d'addition
        MOVE.L      A5,ZDL+98,L             donnant un résultat faux
        MOVE.L      D6,ZDL+102,L
        LEA         ZDL+94,A5
        MOVE.L      #H'027378F2,D6
        ADD.W       (A5),D6             D6 contient une
valeur fausse
    ;                                   ----------
        MOVE        SR,ZDL+108,L        ensemble de modules testant d'autres
        …           …
        configurations d'addition (sans se servir de D6)
        …           …
        MOVE.L      #H'ACDE92B1,D4
        ADD.W       D4,-3(A3),A0.W
    ;                                   ----------
        MOVE        SR,(ZDL+156).L      module dans lequel l'erreur sera visible
        MOVE.L      A2,(ZDL+158).L
```

```
        MOVE.L      D1,(ZDL+162).L
        MOVE.L      D6,(ZDL+166).L      observation de D6 qui met en évidence
l'erreur
        LEA         (ZDL-1994).L,A2
        MOVE.L      #H'00001061,D1
        MOVE.L      #H'62D377D2,D6
        ADD.W       D6,-123(A2,D1.L)    première instruction utilisant D6
    ;                                   ----------
```

Le module manifestant l'erreur se situe plus ou moins loin en aval du module contenant l'instruction qui cause réellement l'erreur. Ceci fait que dans une phase de diagnostic, la recherche d'erreur est un peu plus délicate. Mais la grande flexibilité du logiciel GAPT permet une génération dans le mode conformité des instructions suspectes à partir de la même description de base.
La seule possibilité de ne pas détecter une erreur est que le registre contenant la valeur erronée se trouve modifié intempestivement avant sa séquence d'observation, et que la modification lui redonne la valeur attendue.

II.3.2.c Le mode balayage

Le test de la partie opérative, qui vérifie les différents blocs de la partie opérative (registres, opérateurs, connexions ...) se fera à l'aide d'un test de **balayage**. Dans ce cas le module élémentaire aura la forme suivante :

- initialisation des registres source de l'instruction ;
- activation de l'instruction ;
- observation des registres destination de l'instruction.

Exemple du 68000
Instruction d'addition en format Word - adressage : indirect mémoire, registre données ;
forme du module élémentaire de balayage :

```
        LEA         (ZDL).L,A0          ⎫  initialisation des
registres source de l'instruction
        MOVE.L      #H'00005555,D0      ⎭
        ADD.W       (A0),D0                 instruction à
tester
        MOVE        SR,(ZDL+2).L        ⎫  observation des registres
destination de l'instruction
        MOVE.L      D0,(ZDL+4).L        ⎭
```

total : 5 instructions
taille : 26 octets de programme et 8 octets de données ou de résultats

Ce type de test vérifie que les résultats des diverses fonctions sont corrects, mais ne vérifie pas que d'autres registres ne soient modifiés à tort. On se sert donc de ce test pour vérifier, par exemple, toutes les valeurs d'opérande possibles pour un branchement conditionnel lorsque l'on soupçonne une erreur au niveau d'un PLA de décodage de branchement.

La puissance de test de tels programmes provient de leur exhaustivité. Pour chaque instruction considérée, on peut générer **toutes** les combinaisons possibles de code opération, mode d'adressage et registre avec des opérandes prédéterminés ou aléatoires. Sa principale qualité est également son principal défaut. L'exhaustivité entraîne une taille de programme qui, si elle reste acceptable dans le cas d'un processeur 8 bits (exemple M6800 : 5,5 Ko), devient complètement prohibitive dans le cas d'un processeur 32 bits (exemple M68000 : plus de 3 Mo, M68020 estimé à plusieurs centaines de MegaOctets). C'est pourquoi, au vu des expérimentations, de nombreuses améliorations ont été successivement apportées au logiciel GAPT afin de faire face au problème de la taille, tout en essayant de conserver la puissance de détection.
Pour réduire la taille des programmes générés et permettre ainsi de tester des microprocesseurs tels que le 68020, la prise en compte de nouvelles primitives du langage de description, permettant de contrôler les itérations lors de la phase de génération du programme de test, a été nécessaire (tirage aléatoire de nom de registre ...).
L'utilisation du langage assembleur comme langage cible du programme de test est également un avantage et un inconvénient :

- inconvénient car elle nécessite de disposer d'un assembleur (ou d'un cross assembleur) pour le circuit à tester,
- avantage car elle permet une grande portabilité sur différentes machines de test. En outre, la compréhension et leur utilisation dans le cadre de diagnostic est d'autant facilitée.

II.3.3. Description du matériel

Nous allons faire ici une description rapide des divers systèmes de test utilisés dans le système GAPT.
Indépendamment du coût ou de la sophistication des environnements de test développés, un testeur doit assurer deux fonctions :

- stimuler le circuit sous test,
- observer les réponses.

L'observation des réponses se fait en observant toutes les sorties du circuit sous test. Par contre au niveau de l'acquisition des stimuli deux solutions se présentent :

- soit le circuit sous test est passif, et dans ce cas c'est le testeur qui porte dans sa mémoire le programme de test et qui impose les valeurs d'entrée sur les broches adéquates du circuit,
- soit le circuit sous test est actif et c'est lui qui va lire dans une mémoire son propre programme de test.

Un second critère de distinction entre les différents environnements de test est le type de référence utilisé. Deux stratégies s'opposent : une comparaison avec des résultats prédéterminés, ou une comparaison temps réel avec un circuit de référence.

Dans le système GAPT, le méthode employée dans les différents testeurs développés (FUTE8, TEMAC, FUTE16/32) est l'utilisation d'un système actif avec comparaison à des résultats prédéfinis.

Ce système comporte trois avantages :

- la possibilité d'utiliser directement, lors du test, des programmes en langage d'assemblage du microprocesseur cible avec comme seule transformation l'assemblage et l'édition de lien de ces fichiers ;
- la possibilité d'appliquer de longues séquences de test, la seule limitation étant le chargement en mémoire de test des programmes stockés en mémoire de masse (phase qu'il est facile d'automatiser) ;
- la facilité du diagnostic dans la mesure ou l'on raisonne en langage d'assemblage et non en binaire.

II.3.3.a Le testeur TEMAC

Décrit dans la littérature en [Bel84b], [Bel85] [Zia86]] nous en donnons un schéma de principe dans la figure 1.4.

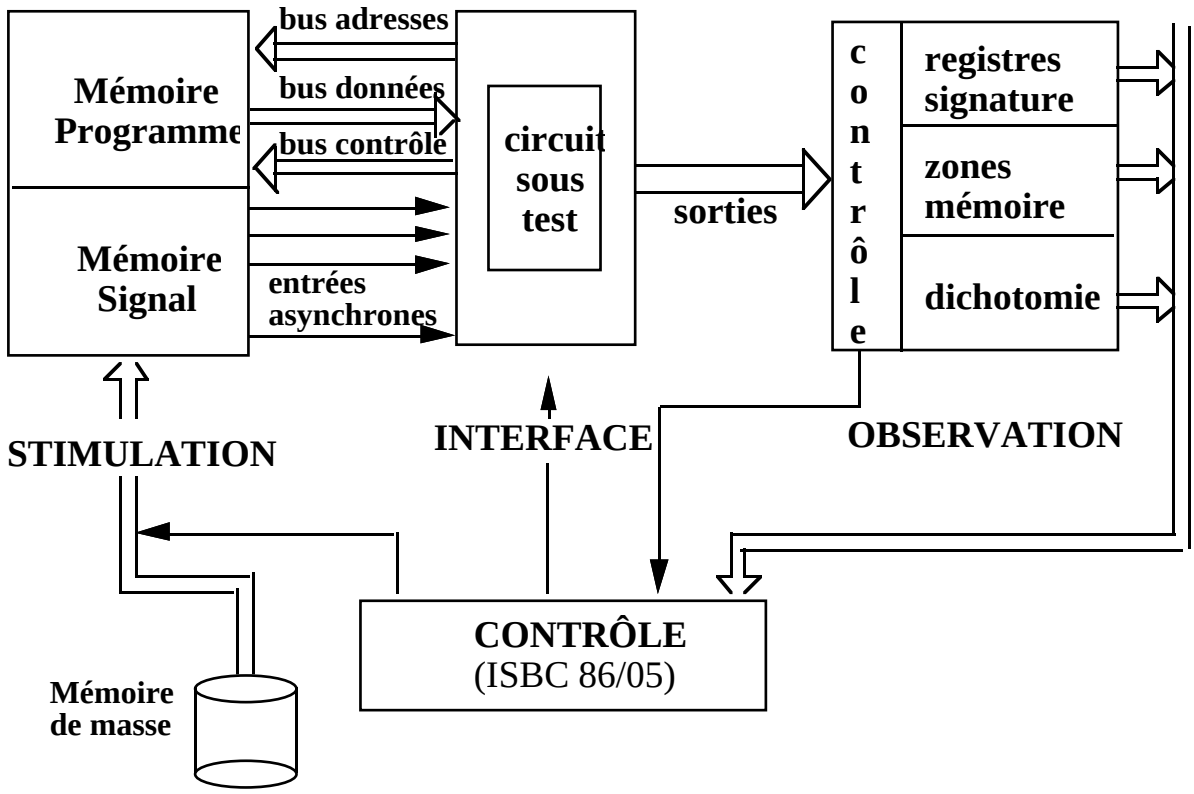


Figure 1.4 : principe du testeur fonctionnel TEMAC.

Basé sur le principe "circuit sous test actif avec observation de tous les bus par signature" (données, adresses et contrôle), ce testeur permet d'obtenir un contrôle très fin des observations. Ceci est réalisé grâce à l'emploi de registres à décalage rebouclés [Dav78], enregistrant une combinaison de leurs anciennes valeurs et des nouvelles portées par les différents bus, et ce, chaque fois qu'elles sont valides. La comparaison des résultats se faisant par rapport à des signatures de référence obtenues par exécution du même programme sur un ou

plusieurs circuits de référence. Les résultats de recherches sur l'efficacité de l'analyses de signatures ont été présentés en [Cas91].
Un système de localisation d'erreur (basé sur un principe dichotomique) permet de déterminer l'instruction qui a manifesté l'erreur. Ce qui est impossible au vu de la seule signature erronée. L'analyse de l'erreur est rendue plus aisée par la possibilité d'observer des échanges entre le circuit sous test et la mémoire du testeur (mémoire pile, afficheurs ...).
Deux types de programmes peuvent être employés en même temps :

- des instructions et des données stockées dans une mémoire non accessible en écriture afin d'éviter les modifications de programme intempestive ;
- des signaux stockés dans une seconde mémoire et préparant l'activation parallèle de signaux d'environnement externe (reset, halt, interruption...)[Bel84b] ou interne (VMA, BG, DTACK...).

II.3.3.b Les testeurs FUTÉ

Famille de testeurs plus légers que TEMAC, les testeurs FUTÉ, dont le premier prototype FUTE8, a été développé pour les circuits 8 bits [Vel82], sont maintenant opérationnels pour les circuits 16 et 32 bits [Pro89][Kar92]. La figure 1.5 en donne le schéma de principe.

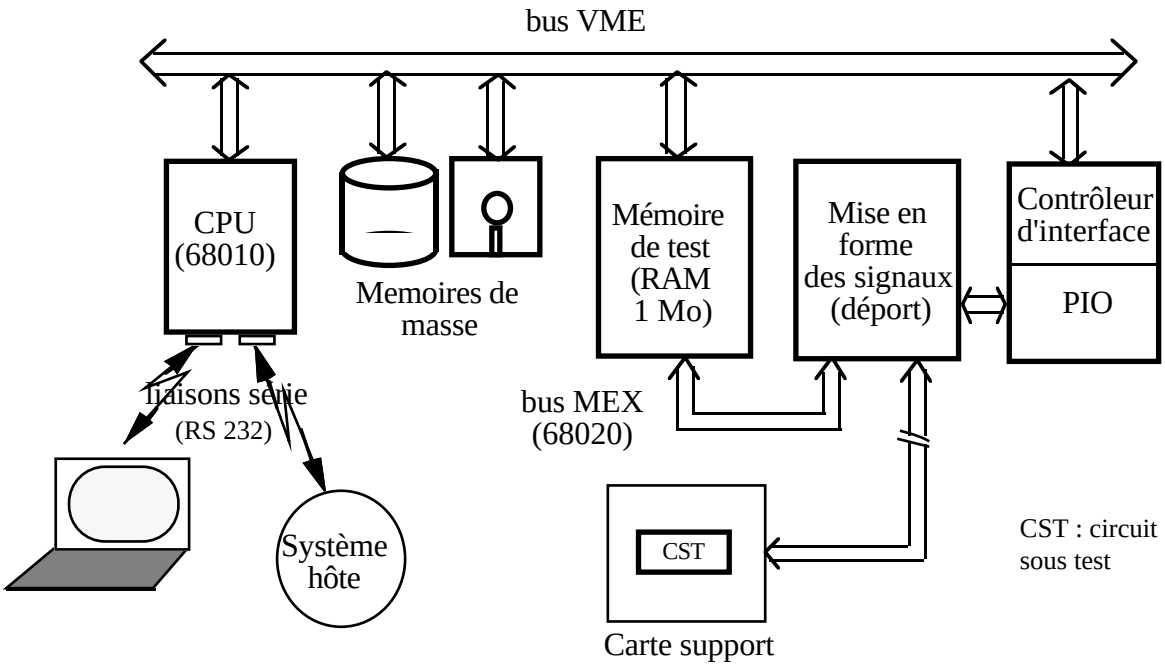


Figure 1.5 : principe du testeur fonctionnel FUTE16.

Le testeur actuel FUTE16/32 est une version "pseudo-industrielle" bâtie autour d'un bus VME. Le circuit sous test exécute un programme stocké dans la mémoire du testeur, l'observation des sorties est réalisée via des écritures dans la même mémoire.
La gestion de la mémoire est donc plus délicate, de même que les diagnostics en cas de modification du programme suite à l'occurrence d'une erreur (écriture intempestive, déséquencement...). Le seul moyen d'observation étant des écritures mémoire, les observations des bus adresses ou des signaux de contrôle ne pourront se faire que de manière indirecte.
Ceci entraîne certaines limitations :

- les activités erronées ayant lieu en dehors des cycles d'écriture ne peuvent pas être détectées directement.
- seule l'activation des signaux d'interruption ou de réinitialisation est possible.

Néanmoins, ce testeur est très largement utilisé notamment pour le test de circuits programmables en environnement sévère [Vel87b][Pro89][Vel89] [Kar91].

II.4. Les défauts et leur modélisation

II.4.1. Les défauts physiques

On peut classer les défauts intervenant durant la phase de fabrication d'un circuits en deux familles :

- Les *défauts globaux* ("global defects") qui vont concerner une surface très large du circuit et qui seront souvent répétitifs. Ils sont dus à des mauvaises manipulations des plaquettes, à des défauts d'alignements des masques, à des gravures trop fortes ou trop faibles, etc. Ces défauts peuvent être limités par une meilleure surveillance des processus de fabrication et ils sont facilement identifiés pendant la fabrication.
- Les *défauts ponctuels* ("spot defects") qui sont des défauts très localisés de petite taille ; leur surface est comparable à celle d'un élément simple (transistor, contact,…) et leur emplacement est aléatoire. Il sont dus à des perturbations locales du processus de fabrication. Généralement ce sont de petites particules chimiques (gouttelettes) ou des poussières en suspension dans l'air qui se déposent durant les diverses phases du processus. Elles vont conduire à du matériau supplémentaire ou à une absence de matériau dans une zone restreinte au sein de l'une des couches du circuit (conductrice, semi-conductrice, isolante).

Dans la suite de ce travail, nous ne nous intéresserons qu'aux défauts ponctuels. De nombreux travaux de recherche ont montré que ce type de défauts étaient à l'origine de la plupart des fautes non détectées par les programmes de tests fonctionnels (au sens comportemental ou structurel) [Mal87] [Kor90].
Les manifestations électriques des défauts ponctuels peuvent être classées comme suit :

- Courts-circuits ("bridges") occasionnés par du matériau conducteur ou semi-conducteur supplémentaire dans des zones de conduction (courts-circuits horizontaux), ou par l'absence de matériau isolant (courts-circuits verticaux).
- Coupures ("opens") provoquées par l'absence de matériau conducteur ou semi-conducteur dans des zones de conduction (coupures horizontales), ou par dépôt supplémentaire de matériau isolant dans les zones de passage prévues dans les couches isolantes (coupures verticales).

- Dispositifs parasites (“extra devices”) ; par exemple du Silicium polycristallin ou de la diffusion supplémentaire peuvent créer un transistor parasite.
- Dégradation de performances des dispositifs existants (modification de la taille d’un transistor, courant de fuite,…)

La figure 1.6 donne des exemples des différentes catégories de défaut couramment citées [Tew91].

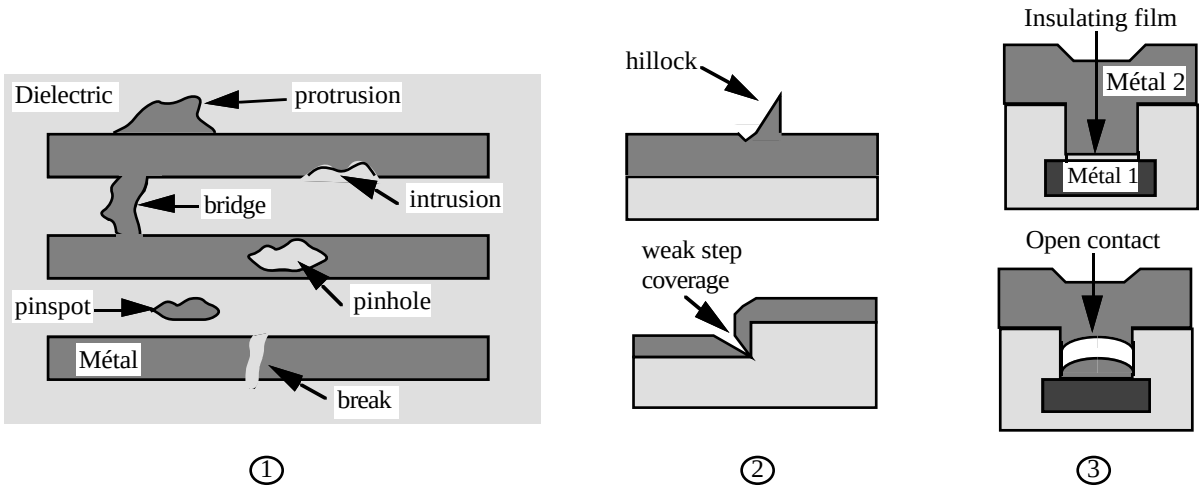


Figure 1.6 : défaut locaux résultant des processus de fabrication :
(1)- défauts horizontaux. (2)- défauts verticaux. (3)- défauts de contacts

A ces défauts, conséquences de problèmes durant la phase de fabrication d’un circuit, il faut ajouter les défauts qui vont apparaître durant la vie du circuit. Ce sont des défauts dus à des phénomènes de corrosion ou d’électromigration par exemple. Les plus connus sont les défauts appelés sous le termes anglais de “gate oxide shorts”[Sod86][Hon91]. Ces courts-circuits peuvent apparaître soit pendant la fabrication, soit durant la vie du circuit. Ils sont dus soit à des défauts dans l’oxyde de grille, soit à l’application d’un trop fort voltage au travers de l’oxyde de grille. Les divers courts-circuits possibles sont illustrés par la Figure 1.7.

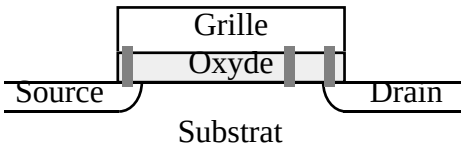


Figure 1.7 : types possibles de courts-circuits d’oxyde de grille.

II.4.2. Problèmes de modélisation.

Comme nous l’avons vu au paragraphe 1.2, le principe couramment employé pour évaluer l’efficacité d’un test est d’établir une liste de fautes abstraites traduisant le mieux possible, au niveau d’un modèle, les effets des fautes physiques au niveau du circuit ; puis pour chacune de ces fautes déterminer la séquence d’activation nécessaire a leur mise en évidence par une erreur

visible aux sorties du circuit. L'efficacité du test est mesurée par la couverture de fautes du modèle.

Il est de nos jours clairement admis que le modèle de collages simples à 0 ou à 1 ne peut pas remplir correctement son rôle dans les simulateurs de fautes et ce, pour deux raisons :

- tous les défauts ne peuvent pas être modélisés par des fautes de collage simple ;
- la représentation des circuits sous forme de diagrammes logiques (ne prenant pas en compte tout les aspects de la topologie) conduit à des impossibilités.

Ces deux affirmations ont été notamment mises en évidence en [Gal80].

Considérons la porte donnée figure 1.8, qui réalise la fonction z=((a+b).(c+d)+(e.f)). Si l'on étudie les deux courts-circuits notés 1 et 2, et les deux coupures notées 3 et 4, on constate que :

- le court-circuit **1** peut être modélisé par un collage à 1 de l'entrée **e** ;
- la coupure **3** peut être modélisée par un collage à 0 de l'entrée **e** (ou de l'entrée **f**, ou des deux) ;
- le court-circuit **2** et la coupure **4** ne peuvent pas être modélisés par un collage car ces défauts entraînent une modification de la fonction réalisée par le circuit. Pour la même raison, un court-circuit entre 2 sorties de 2 portes ne peut être modélisé par aucune faute de collage.

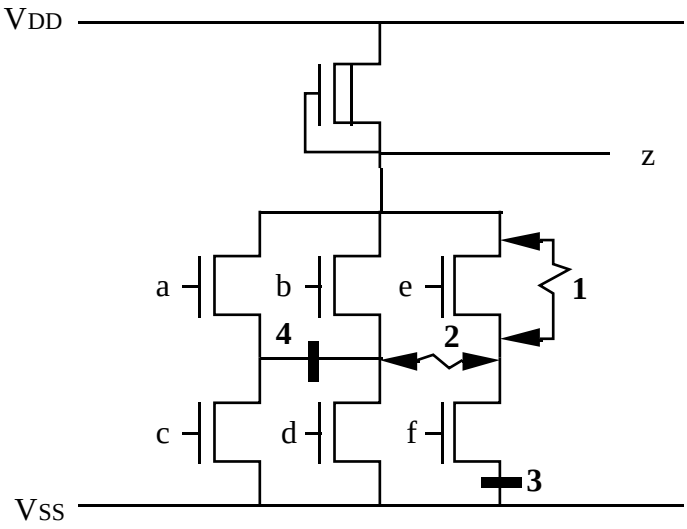


Figure 1.8 : exemples de fautes possibles sur une porte logique.

Reprenons notre exemple, en considérant en plus le diagramme logique (figure 1.9).

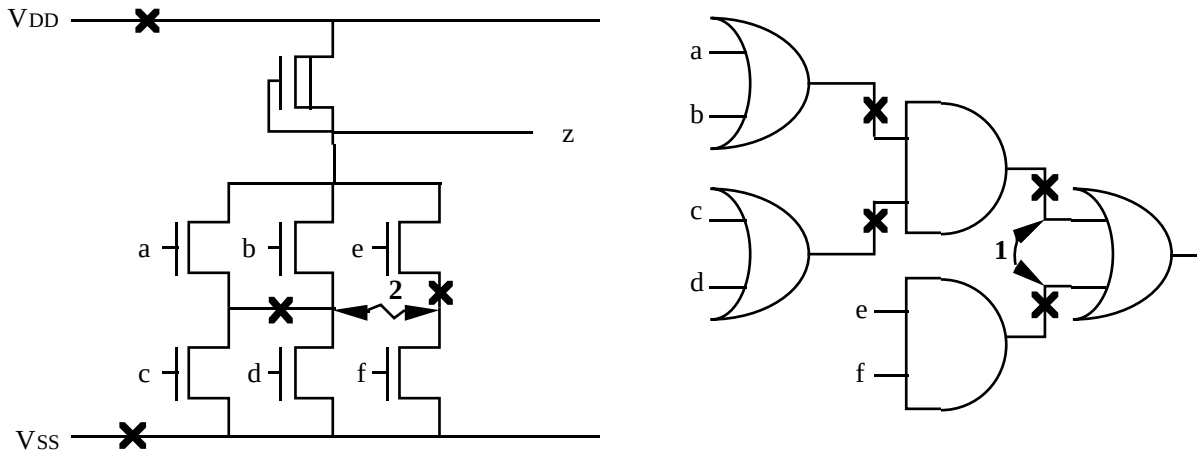


Diagramme électrique Diagramme logique

Figure 1.9 : correspondance entre diagramme électrique et diagramme logique.

Aucune des connexions marquées d'une croix sur le diagramme électrique n'a de représentation sur le diagramme logique. De même aucune des connexions marquées d'une croix sur le diagramme logique n'a de réalité sur le modèle électrique.
De la même façon, le court-circuit possible marqué **2** sur le diagramme électrique ne peut être représenté logiquement ; le court-circuit marqué **1** sur le diagramme logique n'a aucune réalité physique.
En conclusion de cet exemple nous pouvons dire que sur des modèles logiques nous allons tester des fautes qui n'existent pas, alors que nous n'allons pas pouvoir représenter des fautes qui peuvent exister physiquement.

En [Mal87] sont rapportés les résultats de travaux de recherche sur la compréhension et la modélisation de nombreux défauts ponctuels créés par addition ou absence de matériau dans un exemple de circuit. Ce qui suit reprend une partie des exemple développés dans ce travail afin d'étayer les conclusions apportées.

Considérons la fonction booléenne simple suivante appelée F :
$z1 = (\overline{x}\,1 \bullet \overline{x}\,2 + \overline{x}\,3 + x4) \bullet x5$,
$z2 = \overline{x}\,5$,
$z3 = \overline{x}\,4$,
$z4 = \overline{x}\,3$,
$z5 = \overline{x}\,2$.

Le diagramme logique donné figure 1.10 en donne une implémentation minimale en portes NOR.

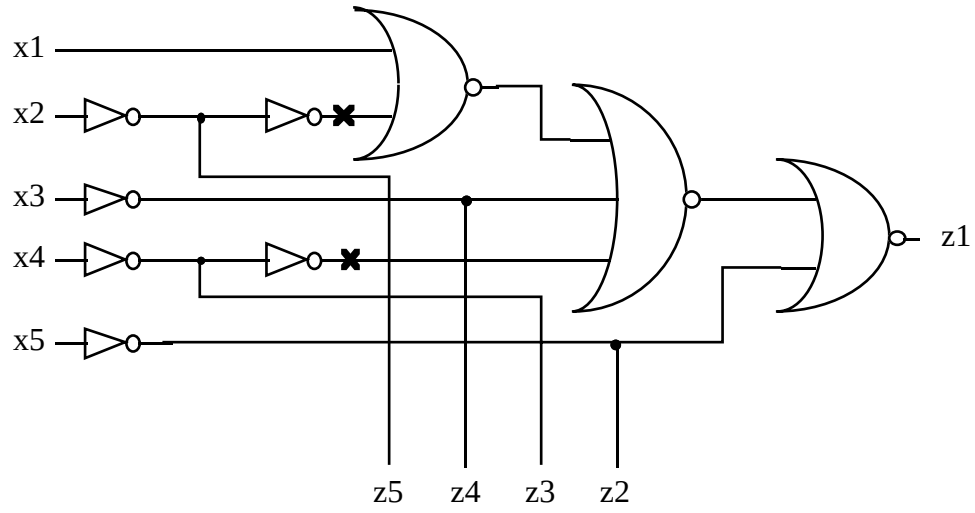


Figure 1.10 : schéma logique en portes NOR de la fonction booléenne F.

La figure 1.11 montre une implémentation en NMOS d'une partie de ce circuit.

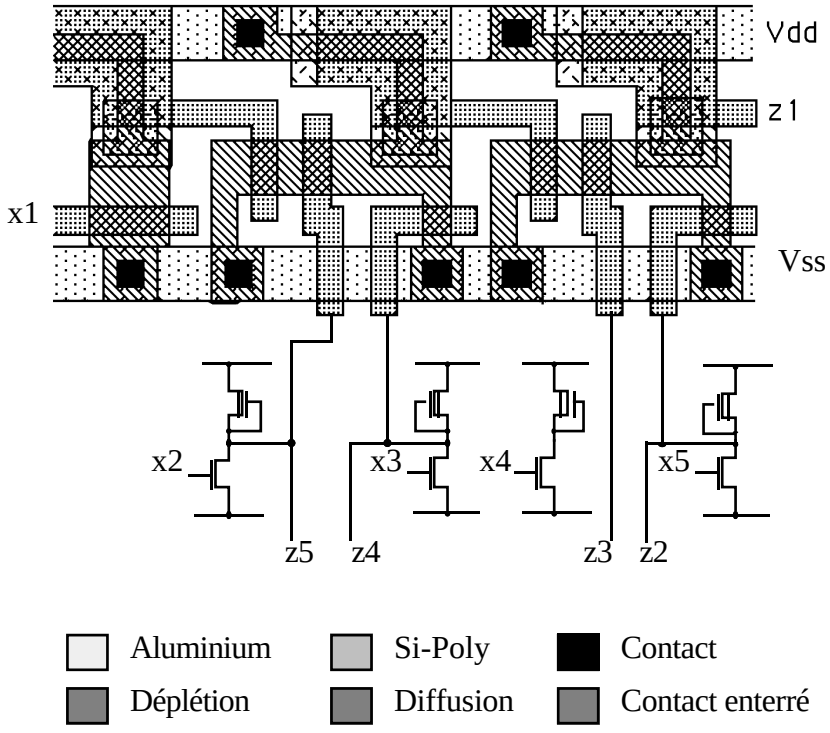


Figure 1.11 : implantation NMOS d'une partie de la logique aléatoire de la fonction F.

Enfin le schéma électrique est donné par le diagramme suivant (figure 1.12) :

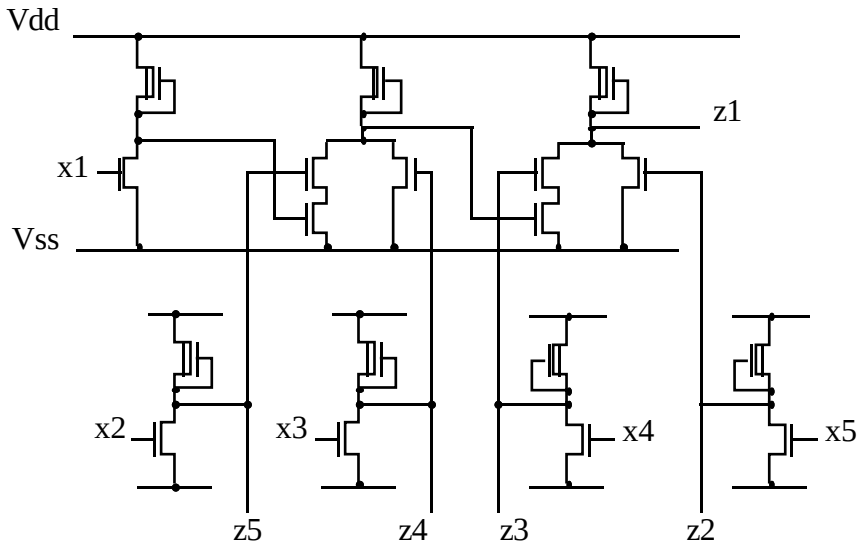


Figure 1.12 : diagramme électrique de la partie de la fonction F implantée.

Ces trois représentations du même circuit sont différentes, et on peut constater que certains défauts visibles au niveau logique n'auront pas de réalité au niveau électrique (par exemple les deux collages marqués d'une croix sur le schéma logique). Pour avoir une bonne corrélation entre les défauts physiques et les défauts logiques, la représentation du circuit au niveau logique doit être proche de l'implantation réelle.
Cette remarque conduit à proposer le diagramme logique suivant (figure 1.13) :

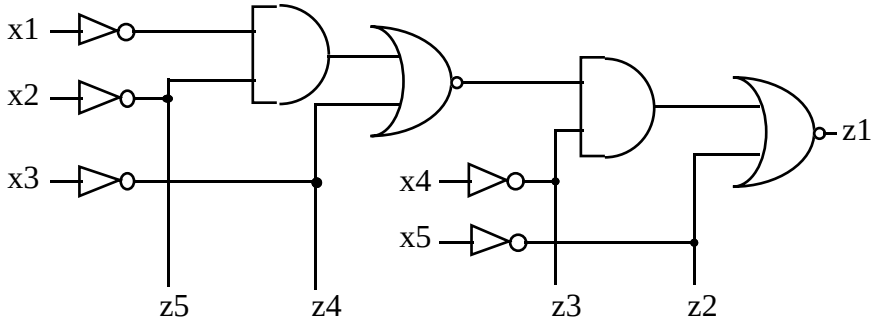


Figure 1.13 : nouvelle représentation logique de la fonction F.

Considérons maintenant les deux défauts ponctuels marqués 1 et 2 sur le schéma de l'implantation donné figure 1.14.

Le défaut 1 est une absence d'oxyde de grille qui va causer un court-circuit entre la grille du transistor et la masse.
Le défaut 2 est un dépôt supplémentaire de diffusion qui va induire un transistor supplémentaire et un court-circuit avec la ligne d'alimentation.

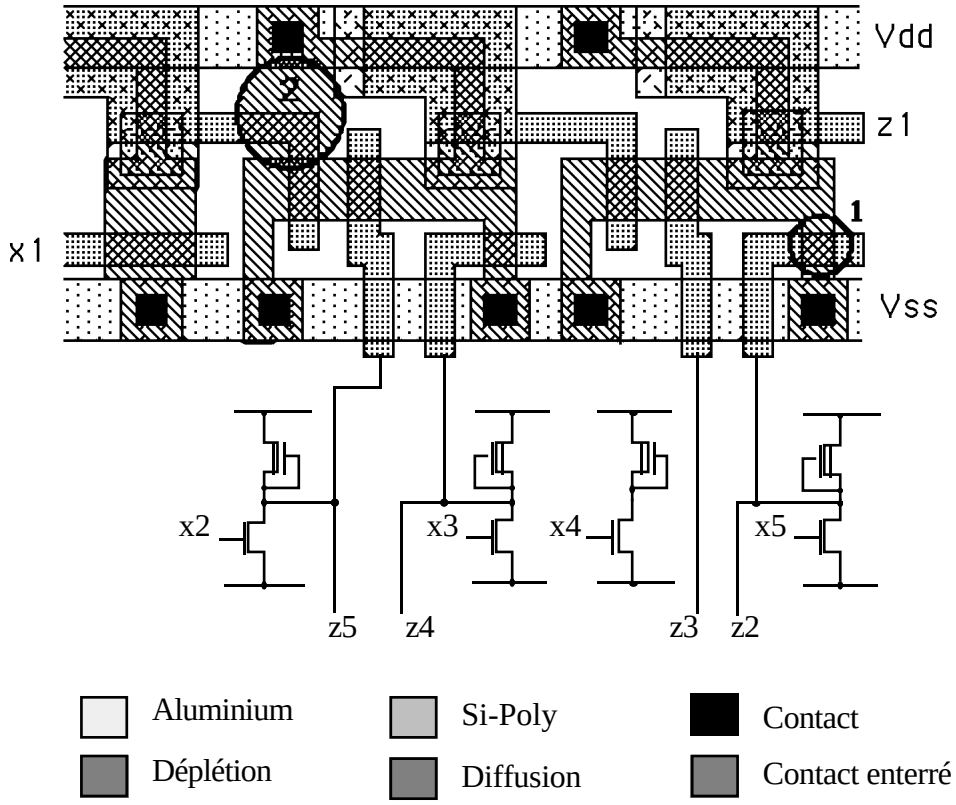


Figure 1.14 : implantation avec prise en compte de défauts
(1) dépôt supplémentaire de diffusion, (2) absence d'oxyde de grille.

Défaut 1 :
Les diagrammes logiques et électriques de la fonction F en présence du défaut 1 sont donnés figure 1.15.
L'analyse montre que le défaut 1, peut être modélisé par deux collages simultanés, un collage à 0 et un collage à 1.

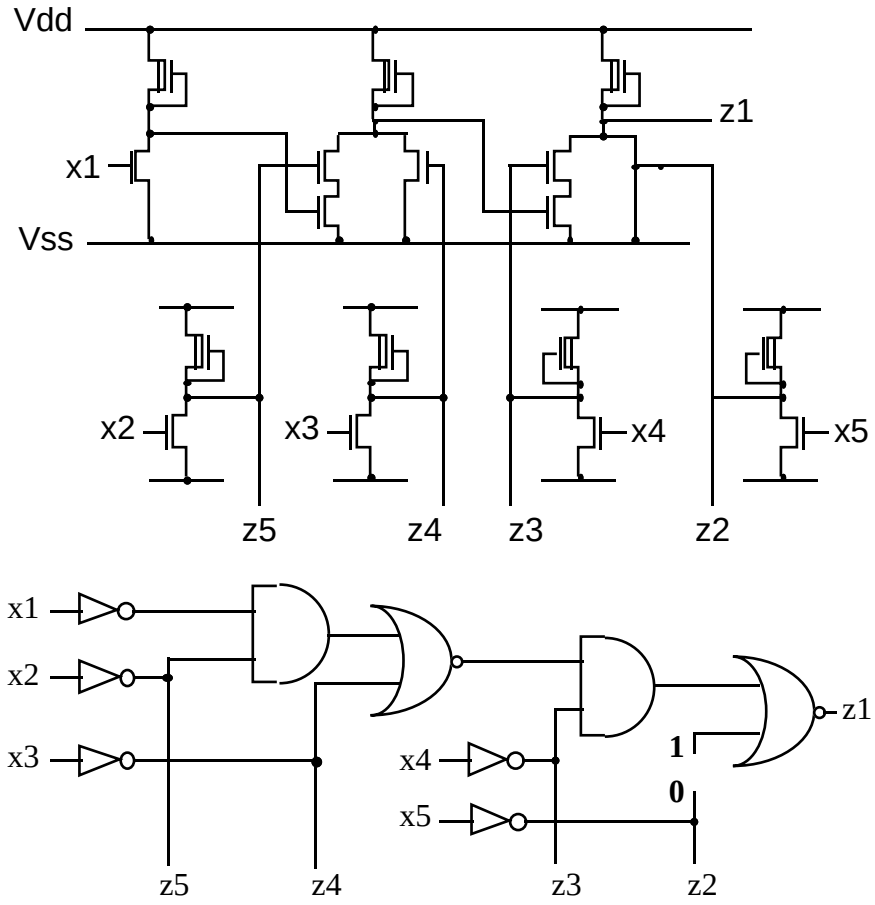


Figure 1.15 : schémas électriques et logiques de la fonction F en présence du défaut 1

Défaut 2 :
Le dépôt supplémentaire de diffusion entraîne la création d'un transistor parasite connectant la ligne d'alimentation et la ligne de masse, et en même temps court-circuite la source de l'un des deux transistors de la seconde rangée avec la ligne d'alimentation.

En présence du défaut 2, la fonction obtenue est fortement modifiée. La figure 1.16 en donne le nouveau diagramme électrique.

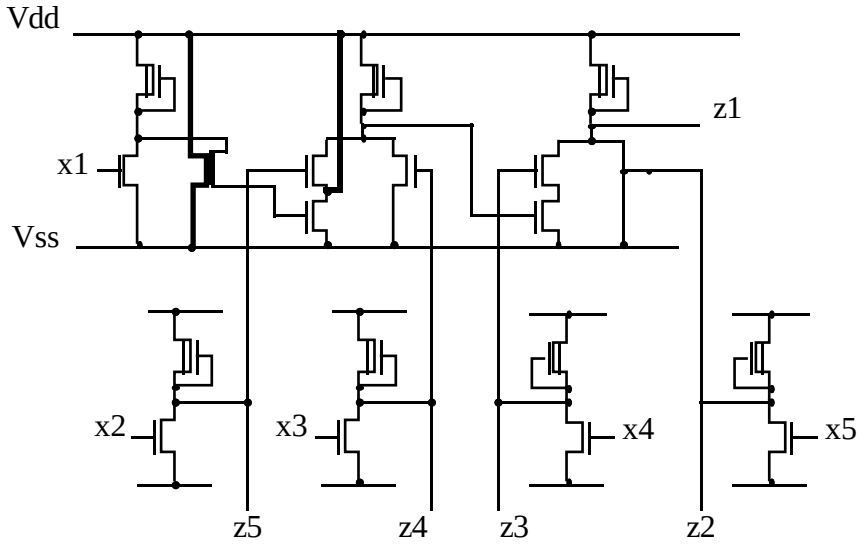


Figure 1.16 : diagramme électrique de la nouvelle fonction en présence du défaut 2.

De toutes ces expériences Maly tire les conclusions suivantes :

- Les fautes qui sont prises en compte à partir du niveau d'abstraction logique n'existeront pas toutes dans le circuit réel.
- Les collages à 1 et les collages à 0 sont des fautes qui existent réellement dans les circuits intégrés VLSI.
- Les fautes de collages multiples sont également présentes dans les circuits intégrés.
- Les fautes de court-circuit peuvent altérer la fonction logique réalisée par le circuit en bouleversant sa topologie.
- Les transistors parasites et les courts-circuits de noeuds internes des portes peuvent transformer profondément la fonction réalisée et ne peuvent pas être modélisés par les modèles de fautes traditionnels.

Donc, l'application des techniques traditionnelles de modélisation de fautes, qui utilisent une représentation logique du circuit et des modèles de collages simples conduisent à des séquences de test qui :

- seront utilisés pour détecter certaines fautes qui n'existent pas :
- ne détecteront pas certaines fautes qui existent ;
- seront utilisés de façon inefficace en ne tirant pas avantage des différentes probabilités d'occurrence de diverses fautes et des combinaisons de fautes multiples.

Dans le cas de modèle fonctionnel la non représentativité des fautes modélisées par rapport aux fautes réelles a été prouvée [Mic82] [Vel88], ce qui rend totalement illusoire le fait de calculer une couverture par rapport à des hypothèses de fautes. L'évaluation de la couverture de collages de la méthode proposée par Abraham et Thatte donnait, suivant ces auteurs, une couverture de 96 % de fautes de collage par simulation sur un modèle au niveau portes logiques d'un microprocesseur 8 bits [Tha80]. L'étude réalisée en 1988 par Klug [Klu88] montre que cette même méthode appliquée à un processeur de traitement du signal CMOS donne pour un modèle de collages 94 % pour la partie chemin de données et 44 % pour la partie contrôle avec un programme de 2300 instructions, développé en 3 hommes/mois. Dans le même article le test aléatoire est évalué et donne des résultats de 94 % pour la partie chemin de données et 64 % pour la partie contrôle et ce, pour un programme de 2500 instructions pseudo-aléatoires. D'autres évaluations de couverture de tests fonctionnels donnent des résultats avoisinant les 70 %. Ces résultats étant obtenus pour des modèles de collages qui ne représentent qu'une faible partie des fautes réelles, notamment dans les circuits CMOS.

En [Gal80] les auteurs ont observés que, sur des échantillons de processeurs 4 bits défaillants, la grande majorité des fautes physiques étaient des coupures ou des courts-circuits. Ces résultats ont été confirmés par les travaux décrits en [Fer88]. Dans cette publication, l'auteur fait part de résultats de simulation de fautes sur deux types de circuits : compteurs et multiplieurs (4

x 4). Les chiffres montrent que près de 90 % des fautes peuvent être modélisées par des coupures ou des courts-circuits. On notera que ces expériences fixent la limite d'une modélisation par collage simple à 0 ou à 1 à moins de 50 % des fautes injectées.

II.4.3. Injection de fautes

Un échantillon de circuits ayant des fautes connues, mélangés à des circuits réputés bons qui pourraient constituer une base de mesure de l'efficacité réelle d'un test est pratiquement impossible à obtenir. Le nombre de circuits défaillants devant être analysés afin de construire un échantillon représentatif des fautes physiques possibles est trop important.

L'injection de fautes, c'est à dire l'introduction délibérée de fautes, peut être réalisée dans un composant selon deux méthodes dépendant du type de cible.
Si la cible est un modèle simulant le comportement ou la structure d'un circuit, on parle de *simulation de fautes.*
Si la cible est un prototype matériel ou le circuit lui-même, on parle d'*injection physique de fautes* [Arl88].

Dans le domaine de la validation du matériel des systèmes informatiques, ces deux méthodes sont souvent appliquées [Mor82][Sch86][Fin87][Arl90]. On notera cependant que l'on travaille généralement sur des prototypes logiciels et matériels des systèmes permettant d'injecter plusieurs types de fautes mais de façon non destructive. Les connexions entre les différents modules du système sont généralement remplacées par des dispositifs permettant d'imposer des valeurs autres que la valeur attendue, simulant ainsi des erreurs dans les modules.
Lorsqu'on travaille sur un circuit intégré, l'injection physique d'une faute est généralement destructive et donc irréversible. On peut néanmoins noter les travaux de Gunneflo [Gun89] [Kar91] qui injecte des fautes transitoires par le biais de bombardements avec des ions lourds. Cette méthode reste limitée de par la nature restreinte des fautes injectées (modifications des bits dans des éléments de mémorisation[4]).

Si on admet que l'injection de fautes au niveau d'un modèle fonctionnel est très contestable, une solution possible pour évaluer l'efficacité du test fonctionnel est l'injection de fautes physiques dans le circuit cible lui-même.
Outre les difficultés techniques posés par les dimensions réduites des éléments, l'injection physique de fautes dans un circuit intégré soulève un problème économique non négligeable lorsqu'un nombre important de fautes doit être considéré[5].

La grande variété des défauts qui peuvent intervenir dans un processus de fabrication implique que le nombre de type de fautes injectables physiquement dans un circuit ou dans un modèle est trop important. De plus il est clair que la reproduction physique de certains défauts est très difficile voire impossible. En effet si on reprend l'exemple du transistor donné dans le paragraphe II.4.2, comme le montre la figure 1.17, on peut reproduire l'équivalent d'un court-circuit drain/grille ou d'un court-circuit source/grille mais le court-circuit grille/substrat est difficilement reproductible.

---

[4] *Ce phénomène est connu sous le nom de phénomène de S.E.U. ("Single Event Upset")*

[5] *Sur des modèles il n'est pas rare de simuler plus de 10.000 fautes ; le même travail sur des processeurs coûtant 100 F pièces reviendrait à 1 MF, sans compter le temps passé à l'injection des fautes soit 1700 heures à raison de 10 minutes par pièce (près de 10 mois à temps plein).*

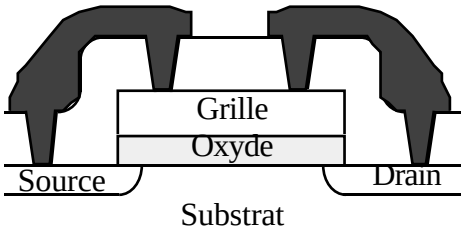


Figure 1.17 : défauts équivalents aux courts-circuits d'oxyde de grille.

II.4.4. Conclusion

Toutes les remarques émises dans les paragraphes précédents nous ont incités à ne pas travailler sur une modèle de fautes et une représentation fonctionnelles du circuit, car ce qui est vrai au niveau logique le sera d'autant plus à un niveau d'abstraction supérieur.
Au vu des résultats annoncés dans le paragraphe II.4.2, nous avons choisi de nous limiter, dans un premier temps, à l'injection physique de coupures puis de coupures et de courts-circuits.
C'est la possibilité de travailler en collaboration avec la société Thomson qui nous à fait choisir le type de circuit étudié : les microprocesseurs 6800 et 68000.

Notre expérimentation décrite dans la partie suivante va donc être résumée comme suit : injection physique de coupures et de courts-circuits dans des circuits du commerce.

Chapitre 2

Expérimentation

Dans nos expériences nous nous intéressons uniquement aux circuits défectueux, par conséquent le schéma du principe général du test donné chapitre 1 est modifié. La figure 2.1 illustre les nouvelles notions employées.

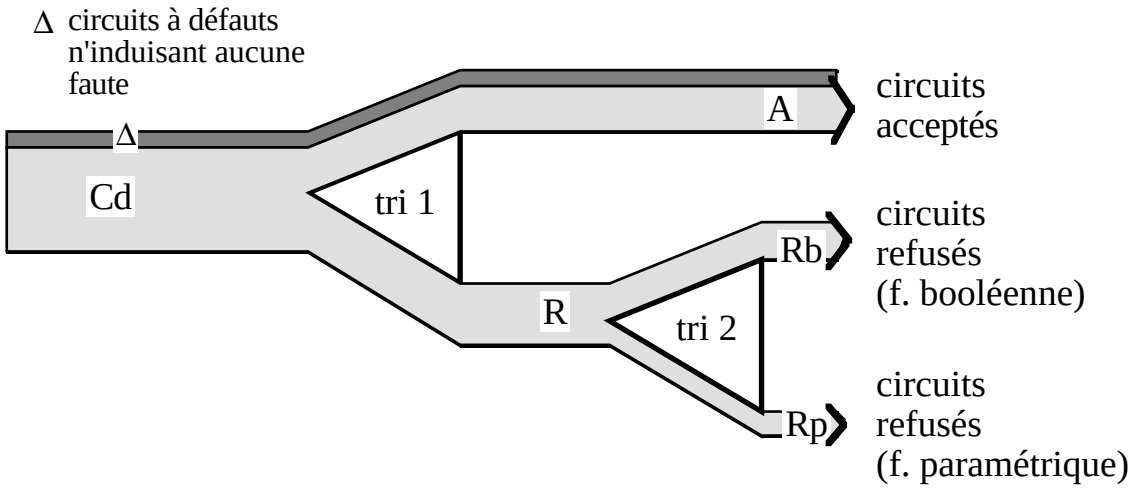


Figure 2.1 : le principe expérimental appliqué

Soit l'ensemble de Cd circuits tous défectueux par injection d'un défaut par circuit. Un premier tri sépare cet ensemble de Cd circuits en deux sous ensembles respectivement de A circuits acceptés et de R circuits refusés. Un second tri sépare l'ensemble des R circuits refusés en deux sous-ensembles suivant le type de faute induite par le défaut. Ces ensembles sont constitués respectivement des Rp circuits refusés pour faute paramétrique et des Rb circuits refusés pour faute booléenne. Un troisième tri intervient pour corriger les ensembles Cd et A en ôtant les circuits portant un défaut qui n'induit aucune faute (noté Δ). Dans les notations ci-dessous ce sont les valeurs corrigées de Cd et A qui seront prises en compte.
Ce que nous allons mesurer sera appelé taux de couverture de fautes simples pour le rapport **R / Cd** et taux de couverture de fautes booléennes pour le rapport **Rb / Cd - Rp**
Dans notre cas les hypothèses d'unicité des fautes et d'équiprobabilité sont garanties par la méthode expérimentale employée.

Par rapport à la figure 1.1 du chapitre 1, notre ensemble de départ de Cd circuits ne représente qu'un sous-ensemble des D circuits défectueux. Cette proportion est difficile à évaluer pour les mêmes raisons que celles avancées plus tôt : nous ne connaissons pas la probabilité d'apparition des défauts que nous créons. Nous ne pouvons donc pas évaluer le nombre D de circuits défectueux (et donc le nombre C de circuits bons et défectueux) qu'il faudrait en fait considérer pour que notre ensemble de départ soit inclus dans un ensemble réellement obtenu à la sortie d'une chaîne de fabrication.
Ce que nous avons cherché à faire, c'est de se doter d'un échantillon de quelque centaines de pièces, présentant un type de défaut reconnu, afin d'obtenir des valeurs concrètes (pour tel ou tel rapport…). Ceci aurait probablement nécessité la fabrication de plusieurs millions de pièces pour en obtenir un ensemble de taille similaire par les voies du hasard.

Trois expériences successives ont eu lieu.
Dans la première expérience nous avons voulu valider la méthode d'injection de défauts et comparer différentes méthodes de tests. Nous avons travaillé sur le microprocesseur 8 bits 6800 et ceci pour deux raisons : d'abord parce qu'il est simple, ensuite parce que nous pouvions disposer de plusieurs programmes de test d'origines différentes, notamment des programmes de test d'un fabricant.
Dans la seconde expérience, nous avons voulu confirmer les résultats obtenus sur un processeur simple, en nous intéressant à un processeur 16 bits, donc de complexité supérieure. Nous avons choisi le 68000 car nous pouvions là encore bénéficier d'aide extérieure pour effectuer les tests du fabricant.

La troisième expérience a également été menée autour du 68000. Le but recherché étant ici l'extension des défauts aux courts-circuits, afin de couvrir un ensemble de fautes représentatif. Des mesures sur la perte d'efficacité du test fonctionnel par réduction de la taille des programmes, déjà esquissées dans la phase précédente, ont été effectuées ici.

I. L'INJECTION PHYSIQUE DE DEFAUTS

La solution que nous avons adoptée pour ces expérimentations a été de nous servir d'un équipement laser développé pour réparer ou modifier des circuits pendant leur phase de conception[6] .
L'équipement employé est composé de deux systèmes centrés autour d'un faisceau laser à Argon :

- Le premier est un laser de découpe qui utilise un faisceau laser focalisé sur une surface ponctuelle d'environ 1 micron. Un faisceau laser de 0,4 W coupe localement des lignes d'aluminium avec une puissance de 2 watts. Une coupure sera significative seulement si la résistance résultante est supérieure à 10 KOhm [Auv86].
- Le second équipement est un système laser à déposition qui utilise également un faisceau laser Argon focalisé. Le circuit, placé dans un environnement gazeux adéquat, est localement chauffé par le laser, et les molécules gazeuses environnantes se décomposent sur le point chaud en laissant sur place des atomes métalliques. Ceux-ci forment un dépôt métallique conducteur (nickel dans notre cas) placé entre 2 lignes d'aluminium, réalisant ainsi un court-circuit électrique [Auv91].

I.1. Le laser de découpe

I.1.1. Principe

La figure 2.2 donne le schéma de principe du laser de découpe.

Le laser est couplé à une station de travail, sur laquelle un logiciel spécifique permet la manipulation du faisceau grâce à une surveillance vidéo. Les opérations de cadrage de la zone à découper, de choix de grossissement par zooms successifs, de correction de l'alignement du circuit et de déclenchement du faisceau laser se font directement à partir de la station. Chaque mouvement au niveau de la console est répercuté au niveau des tablettes de réglages X, Y, Z du microscope optique ; chaque mouvement du zoom déclenche un changement d'objectif réalisant ainsi une focalisation de plus en plus importante du faisceau laser. La commande de découpe déclenche l'allumage du faisceau dont la puissance doit être préalablement réglée en fonction du matériau à couper (environ 0,4 W pour le Silicium polycristallin et 2 W pour l'aluminium).

[6] *Matériel développé au CNET CNS par G. Auvert*

Figure 2.2 : schéma de la partie optique du laser à découpe.

Les figures 2.3 et 2.4 montrent le résultat de coupures d'une ligne d'aluminium réalisée avec un objectif de grossissement 50 pour la découpe de gauche (0,8 µm) et un objectif de grossissement 40 pour la découpe de droite (1 µm).
La figure 2.3 est une photographie prise au microscope optique montrant ce que l'on voit à l'aide du moniteur vidéo couplé au laser. La figure 2.4 est une vue au microscope électronique à balayage des 2 mêmes coupures.

Figure 2.3 : coupures vues au microscope optique.
(champs de 85 microns × 75 microns)

Figure 2.4 : coupures vues au microscope électronique à balayage.
(largeur 0,8 µm à gauche, objectif 50 × - largeur 1 µm à droite objectif 40 ×)

Un système de déplacement en X, Y précis au 1/10 de micron permet de placer le faisceau laser à l'endroit désiré en déplacement relatif par rapport à une origine. Une correction d'angle θ permet de contrôler l'alignement des axes X, Y avant déplacement.

I.1.2. Mise en oeuvre

Nous pouvons travailler soit sur des circuits encapsulés, soit sur une plaquette complète.
Dans le cas de circuits encapsulés, après mise en place sur la platine du microscope, chaque circuit cible doit subir les différentes phases de correction d'angle, déplacement, choix d'objectif,… avant la découpe proprement dite. Cette procédure doit être répétée entièrement à chaque fois que l'on change de pièce.
Si on travaille sur une plaquette complète, une procédure automatique permet, après calcul des déplacements de base nécessaires, de passer directement d'une puce à la suivante soit en déplacement latéral, soit en déplacement horizontal.

Dans ce cas le réglage du θ d'alignement se fait une seule fois sur l'ensemble de la plaquette, seule une correction minime de définition d'origine est nécessaire pour chaque circuit de la plaquette.

I.2. Le laser à déposition

I.2.1. Principe

La figure 2.5 ci-dessous donne le schéma de principe du laser à déposition. Comme dans le cas du laser à découpe celui-ci est relié a une station de travail qui permet de contrôler par le biais d'une vidéo, les déplacements X, Y, Z, l'angle θ d'alignement et les calibres d'objectifs. De plus, ce système nécessitant de travailler en ambiance gazeuse particulière, un système de pompage et d'injection de gaz est nécessaire.

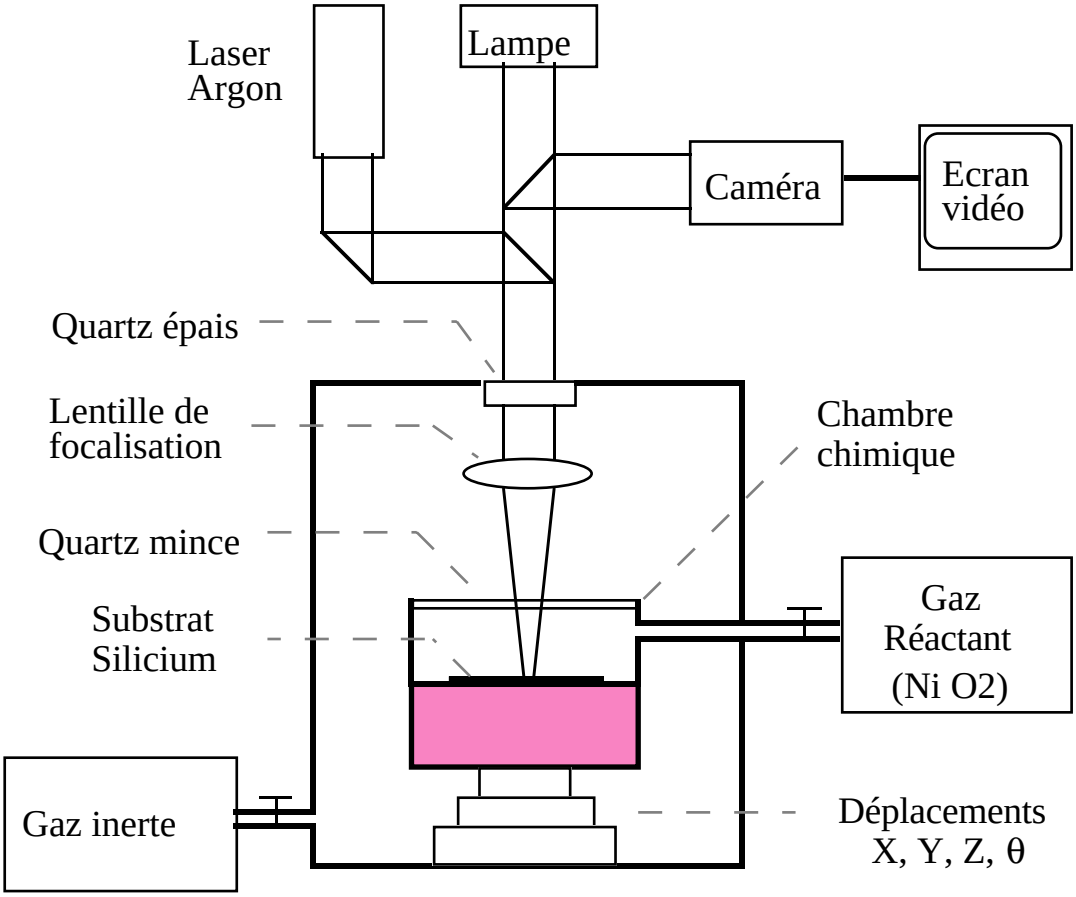


Figure 2.5 : le laser à déposition

Un vide assez poussé (10^{-3} millibars) doit être réalisé dans la chambre où a été déposé le circuit à traiter, avant d'injecter un gaz réactant (NiO2) sous une pression adequate (2 millibars). La première chambre baignant elle même dans une athmosphère spéciale composée d'un gaz inerte. C'est une procédure assez longue qui nécessite également un pompage et une évacuation des vapeurs avant de revenir à un état normal pour retirer le circuit. Chaque opération de ce type prend environ une demi-journée.

La reproduction de photographie (figure 2.6) d'une vue au microscope électronique à balayage montre la connexion entre une piste d'aluminium et un contact.

Figure 2.6 : dépôt de nickel entre une connexion et un contact.
vue au microscope électronique à balayage

I.2.2. Mise en oeuvre

Pour réaliser un court-circuit entre deux éléments d'un circuit il faut donc, en plus de la procédure d'alignement, déplacement et choix d'objectif, mettre en branle toute un ensemble de mécanismes de pompage, d'injection et de vidange des différents gaz. Sachant que l'on ne peut introduire qu'un maximum de 4 à 5 circuits encapsulés dans la chambre centrale il est difficilement envisageable de traiter ainsi un grand nombre de circuits.
Comme pour le laser à découpe, il est plus simple de travailler sur des plaquettes complètes, un système automatique de gestion de plaquette permettant de passer d'une puce à l'autre en déplacement horizontal ou vertical.

Remarques :

- Afin de déposer le nickel au niveau de l'aluminium il est nécessaire d'opérer sur des circuits dépassivés. Cette opération peut s'avérer délicate si on ne dispose pas des informations sur la composition exacte du dépôt de passivation et de son épaisseur afin de choisir le produit décapant adéquat et de calculer finement le temps de trempage dans la solution.

- L'immersion prolongée d'un circuit dans le gaz réactant peut lui être nocif voir fatal en créant des courants de fuite importants ; donc en cas de travail sur plaquette, il est nécessaire d'agir rapidement afin d'éviter au maximum cet inconvénient.

I.3. Expériences réalisées

Dans les deux premières expériences réalisées, décrites ci-après, seul le laser de découpe a été utilisé. Les circuits testés étaient encapsulés.
Dans la troisième campagne, les expériences ont pu être réalisées sur des plaquettes entières dépassivées. Les deux types de laser ont été employés.

II. INJECTION DE COUPURES DANS LE MICROPROCESSEUR 6800

II.1. Approche expérimentale

Dans cette expérience, notre but a été de comparer objectivement différentes méthodes de test en les appliquant à des circuits réels. Les résultats obtenus ont été publiés en [Mar90]
Afin de ne pas se laisser guider par certaines hypothèses liées à la nature de la faute injectée ou à son emplacement, qui auraient pu influencer notre jugement nous avons choisi d'injecter aléatoirement des défauts sur la surface du circuit. Ceux-ci ont été créés par coupure de pistes d'aluminium ou de silicium polycristallin, à l'aide du laser à découpe décrit dans le paragraphe précédent.
Un couple X, Y de coordonnées est tiré au hasard par le biais d'un générateur aléatoire de nombres entiers.
La démarche choisie à été de couper la piste la plus proche des coordonnées aléatoires, exception faite des gros rails d'alimentation dont la probabilité de coupure accidentelle n'est pas très élevée vu leur taille et dont les effets seraient sûrement importants. De plus, techniquement leur découpe n'est pas facile à réaliser de façon sûre et efficace.

Nous avons choisi également de tester ces pièces endommagées par plusieurs programmes de test d'origines différentes :

- des tests fonctionnels obtenus par le système GAPT, mais également un test ad hoc et un test aléatoire ;
- un test fabricant mêlant le paramétrique et le fonctionnel.

Afin d'obtenir un éventail plus large de mesures, nous avons essayé, chaque fois que cela était possible, de réaliser ces tests sur plusieurs équipements de test différents.

II.2. Les méthodes de test évaluées

Quatre programmes de test ont été appliquées : un test aléatoire, un test fonctionnel ad hoc, un test fonctionnel systématique et un test de fin de fabrication. Un cinquième test composé de quelques instructions, dit test minimal, a également été utilisé.

II.2.1. Le test aléatoire

Nous avons utilisé la séquence de test et le matériel développés par Fedi et David [Fed84] pour le microprocesseur 6800.
Dans ce test, des séquences d'entrées générées aléatoirement sont appliquées à un circuit sous test et à un circuit de référence, leurs sorties respectives étant comparées.
Par opposition aux méthodes de test déterministe, le problème n'est pas de trouver la séquence de test mais d'en déterminer sa longueur. Ce calcul est réalisé à partir d'une méthode probabiliste de pire cas. La longueur du test à appliquer sera fonction de la probabilité de détecter la faute la plus difficile f.

La longueur du test N est dérivée de l'équation suivante :

$$N = \log Q_D / \log(1-P(f))$$

où N représente la borne maximale de la longueur de la séquence aléatoire nécessaire pour détecter la faute f avec une probabilité donnée de $1-Q_D$ ($Q_D = 10^{-3}$ est la valeur généralement admise).
P(f) est la probabilité moyenne de détection de la faute f par chacune des instructions sachant que f est présente. P(f) sera donc déterminée par une analyse du jeu d'instruction.

Le problème majeur de cette méthode est bien entendu ce calcul. Pour le réaliser les auteurs ont utilisé le modèle de fautes fonctionnelle et le S-graph définis par Abraham et Thatte.
La séquence de test aléatoire est composée de plus de 6×10^6 instructions (pour $Q_D = 10^{-3}$) et nécessite 22 secondes pour s'exécuter sur le testeur aléatoire.

II.2.2. Test ad hoc

Nous nous sommes servis comme test ad hoc d'un test développé par l'université de Dijon [Mil87] pour tester des cartes à base de 6800 lors d'expériences d'irradiation réalisées en collaboration avec le C.E.A..
Ce programme est composé de 9 modules vérifiant chacun un sous-ensemble des instructions du 6800. Son principe est basé sur le principe dit de "self test" proposé en [Hun84]. Le même calcul (manipulation de données, transfert de données) est réalisé par au moins deux voies différentes ; les différents résultats sont comparés par une instruction adéquate. Si une différence intervient, un branchement conditionnel déroute le programme sur une procédure d'impression de l'erreur. Chaque interruption est activée et vérifiée une fois durant le test. Ce programme, d'une taille de 1400 octets, active en fait près de 4000 motifs du fait des nombreuses boucles qu'il possède.

II.2.3. Test systématique

Le test fonctionnel systématique du 6800 a été généré à l'aide du logiciel GAPT.
Chaque instruction (code-opération ? mode d'adressage) a été générée en utilisant le mode pseudo-conformité (cf. chapitre précédent), les modules élémentaires obtenus étant donc de la forme :

- observation de tous les registres source ou destination de l'instruction,
- initialisation des registres source,
- activation de l'instruction à tester.

Ce mode nous a permis de réduire la longueur du programme de test à 5 500 octets.
Les opérandes de test employés étaient aléatoires, sauf pour certains tests particuliers, comme les branchements conditionnels, qui étaient testés avec des valeur booléennes rendant la condition de test vraie ou fausse par exemple. Chaque signal asynchrone est activé une fois durant le test (activation d'un seul signal à la fois).

II.2.4. Le test fabricant

Le test employé[7] est le test standard Motorola 6800 auquel ont été ajoutés des motifs de test spécifiques dus aux modifications apportées par les changements de technologies ou les corrections d'erreurs après des retours clients dans la phase de jeunesse du microprocesseur. Ce programme représente 7120 motifs de 40 bits.

II.2.5. Le test minimal

Ce test, composé seulement de quelques instructions, a été conçu dans le but d'obtenir des informations sur le nombre de défauts créés qui pouvaient être détectés de façon triviale.

II.3. Les équipements de test

Les différents programmes de test obtenus ont été appliqués en utilisant trois types d'équipements de test. Chaque fois que les combinaisons "type de test", "type de machine" étaient possibles, elles ont été employées.

II.3.1. Un testeur "aléatoire"

Un générateur pseudo-aléatoire de motifs de test envoie des codes-opération valides et des opérandes de données (données ou adresse) au microprocesseur 6800 sous test. En fait le générateur adresse aléatoirement dans des tables une instruction valide pour le 6800. Les entrées asynchrones (interruption, halt reset, contrôle trois états ...) sont également contrôlés par le générateur. Des poids différents sont affectés aux diverses composantes d'un vecteur d'entrée (par exemple, une interruption ne sera pas aussi fréquente qu'une instruction).
Les sorties sont compactées par des registres de signature, leurs valeurs sont comparées plusieurs fois aux valeurs de référence durant l'exécution de la séquence pseudo-aléatoire. Les valeurs de référence sont obtenues par application de la même séquence pseudo-aléatoire à un circuit réputé bon.
Ce prototype de testeur développé par David et al [Fed84] nous a permis de réaliser cette partie de l'expérimentation. Les séquences de test aléatoire n'ont pu être utilisée que sur ce matériel.

II.3.2. Des testeurs "fonctionnels"

Dans les diverses expériences que nous avons menées, les testeurs fonctionnels développés au laboratoire ont été employés. Pour les expériences autour du 6800 les testeurs FUTE et TEMAC ont été utilisés.

II.3.2.a FUTE8

Ses caractéristiques sont : l'exécution d'un programme de test rangé dans la mémoire RAM du testeur, avec ses opérandes propres, et l'observation des résultats par écriture des valeurs

---

[7] *Ce test a pu être réalisé grâce à la collaboration de la société Thomson qui s'est servie de son équipement de test (Sentry 6).*

obtenues directement dans la même mémoire après l'exécution de chaque instruction. Un tel testeur ne peut supporter que des tests booléens, et seules des fautes de type booléen conduisant à une mauvaise écriture mémoire peuvent être détectées. Il est possible d'activer les signaux asynchrones par le biais de circuits périphériques sous le contrôle du microprocesseur sous test.

II.3.2.b TEMAC

Le même principe que précédemment est utilisé par le circuit sous test, c'est-à-dire l'exécution d'un programme stocké en RAM , mais les résultats sont compactés par signature à chaque cycle du processeur. Les séquences de test sont de type booléen. Les signaux asynchrones sont activés séparément les uns des autres pendant l'exécution du programme.

II.3.3. Les testeurs classiques

Le test fabricant a été exécuté sur un testeur Sentry 20 en utilisant toutes les capacités du testeur, c'est-à-dire que des tests paramétriques, dynamiques et booléens ont été réalisés. Seul le test fabricant a été exécuté sur ce testeur. Un second testeur de ce type a néanmoins été utilisé avec les autres programmes de test. Il s'agit d'un testeur Genrad 125 utilisant uniquement le test booléen. Un traducteur spécifique a été développé afin d'adapter les programmes de test au format du testeur.
Le test systématique et le test ad hoc ont été appliqués sur tous les testeurs mis a part le testeur aléatoire.

II.4. Résultats obtenus

75 circuits MC6800 bons, c'est-à-dire ayant passé avec succès le test du fabricant, ont été traités. Chaque circuit a reçu une coupure dans une connexion d'aluminium ou de silicium polycristallin.
Le tableau 1 donne le nombre de circuits acceptés, et la couverture vis à vis d'une faute simple pour chaque circuit.

Méthode de test	Type de testeur	Nombre de circuits détectés bons (sur 75)	Couverture	Numéro des circuits détectés bons
Minimal	FUTE8	30	60 %	
Ad hoc	FUTE8	12	84 %	10, 14, 15, 19, 20, 32, 41, 50, 64, 67, 75
	TEMAC	11	85,3 %	10, 14, 15, 19, 20, 32, 41, 50, 64, 67, 75
	GR 125	8	89,3 %	10, 14, 20, 32, 41, 50, 64, 67
Systématique	FUTE8	12	84 %	10, 14, 15, 19, 20, 32, 41, 50, 64, 67, 75
	TEMAC	10	86,7 %	10, 14, 15, 19, 20, 32, 41, 64, 67, 75
	GR 125	7	90,7 %	10, 14, 15, 20, 42, 64, 67
Aléatoire	Aléatoire	5	93,3 %	14, 15, 19, 64, 75
Fabricant	Sentry 6	4	94,7 %	10, 19, 64, 75

Tableau 1 **:** résultats des différents tests du 6800.

II.4.1. Commentaires

Un seul circuit a passé avec succès tous les tests : le circuit n°64

Les deux méthodes fonctionnelles, la méthode systématique et la méthode ad hoc, donnent quasiment le même résultat, avec un léger avantage au test systématique. On remarquera néanmoins qu'il n'y a pas parfaite inclusion des résultats. En effet, pour le test donnant le meilleur résultat pour chacun d'eux sur le Genrad 125, en plus de l'ensemble commun 10, 14, 20, 41, 64, 67, le test systématique laisse passer le circuit n°15 détecté fautif par le test ad hoc, alors que celui-ci laisse passer les circuits n°32 et 50.

La qualité du test dépend du testeur sur lequel il a été exécuté. Les meilleurs résultats étant obtenus sur les testeurs du commerce qui permettent l'observation de toutes les sorties, les plus mauvais au niveau du testeur FUTE8 qui ne détecte que les erreurs booléennes.

II.4.2. Analyse des résultats

Afin de pouvoir évaluer uniquement la couverture de fautes de type booléen, nous avons exclus les circuits dans lesquels le défaut créé s'est manifesté seulement par des mesures paramétriques ou dynamiques sur les testeurs classiques. Les circuits n°14, 15, 19, 32 et 75 ont donc été retirés de la liste.

Seuls ont été conservés les 69 circuits présentant une faute booléenne avec au moins une méthode de test. Les meilleurs résultats obtenus sur l'ensemble des testeurs sont retenus. Le tableau 2 résume cette analyse.

Méthode de test	Nbre de circuits acceptés (sur 69)	Couverture de faute fonctionnelle	N° des circuits acceptés
Minimal	24	65,2 %	
Ad hoc	5	92,7 %	10, 20, 41, 50, 67
Systématique	4	94,2 %	10, 20, 41, 67
Aléatoire	0	100 %	
Fabricant	1	98,6 %	10

Tableau 2 **:** résultats du test pour les défauts de type booléens.

Les résultats obtenus représentent en fait une borne supérieure, puisque rien ne prouve que le circuit 64 ne porte pas de défaut de type booléen. Aucune analyse complémentaire n'est venue alors pour confirmer ou infirmer l'hypothèse d'une mauvaise coupure ou d'une erreur de manipulation.

Les résultats des deux tests fonctionnels (systématique et ad hoc) sont proches.

Seul le test aléatoire détecte 100 % des circuits défectueux, le test fabricant laissant passer le circuit n°10.

L'analyse des séquences de test aléatoire montre que dans certains cas plus de 80 000 instructions sont nécessaires pour mettre en évidence la faute. En effet, les circuits 3, 10 et 63 nécessitent un nombre d'instructions compris entre 80 000 et 250 000 pour être détectés fautifs. Nous ne pouvons malheureusement pas être plus précis, l'observation des résultats compactés ne se faisant que périodiquement.

Le circuit 67 nécessite quant à lui plus de 250 000 instructions. Ceci va à l'encontre des résultats présentés par David et al [Fed84] ; dans cet article les résultats d'expériences, effectuées sur un lot de 3000 microprocesseurs 6800 supposés défectueux, donnaient un maximum de 80 000 vecteurs pour détecter toutes les fautes.

Lorsque l'on reporte sur une courbe les résultats obtenus on constate que nous obtenons un tracé classique d'augmentation de la couverture de test avec le nombre de motifs appliqués, au moins pour les tests de type fonctionnel. Le test fabricant est nettement en dehors de cette coupe laissant sous entendre une meilleure optimisation (figure 2.7).

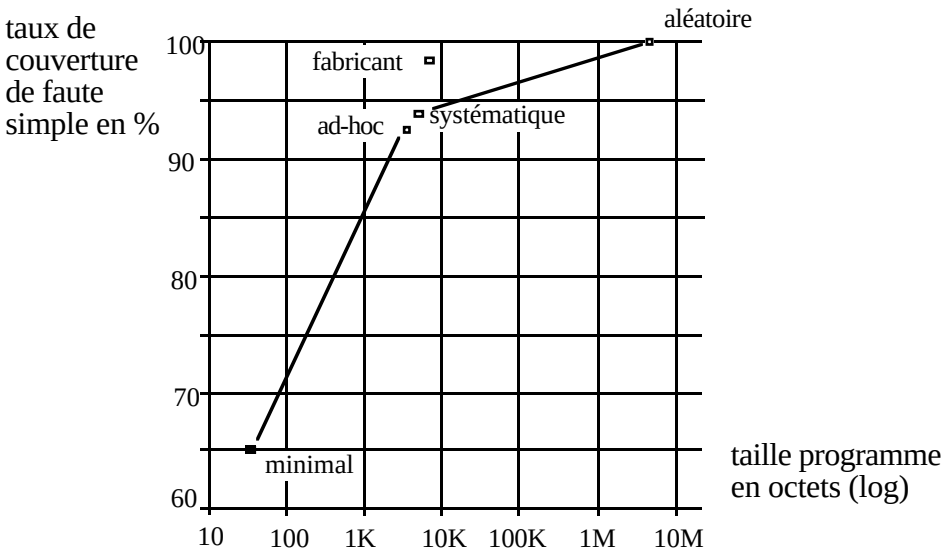


Figure 2.7 : taux de couverture en fonction de la taille des programmes.

II.5. Analyse des circuits présentants des différences pour les diverses méthodes

Les trois circuits 20, 41 et 67 détectés fautifs par le test aléatoire et par le test fabricant, et considérés bons par les tests fonctionnels systématiques et ad hoc ont été analysés.

Les vecteurs du test fabricant mettant en cause ces circuits ont été décompilés. Les résultats obtenus montrent que, pour chacun d'entre eux, une combinaison de signaux asynchrones particulière est nécessaire à la détection de la faute. Les vecteurs du test aléatoire mettant en évidence les fautes n'ont pas pu être analysés puisque les résultats sont observés compactés périodiquement. Il est par conséquent impossible de trouver le vecteur responsable.

- Pour le circuit n°20 la séquence détectant la faute nécessite l'activation simultanée de IRQ, NMI et HALT.
- Pour le circuit n°41, la détection a lieu quand IRQ et HALT sont activés simultanément pendant une instruction de type WAIT suivie de l'instruction CLI (clear interrupt mask).
- Pour le circuit n°67, l'erreur apparaît lorsque IRQ et HALT sont activés ensemble, une adresse fausse étant alors émise.

II.6. Discussion

Les chiffres obtenus doivent être considérés seulement que comme un ordre de grandeur. En effet, les pourcentages obtenus sur un nombre réduits de pièces sont faussés, car dans notre cas un circuit représente 1,4 %.
Néanmoins, nous pouvons tirer certaines conclusions.

• Les deux tests fonctionnels déterministes vérifient le jeu d'instruction. Par contre, au niveau du test des signaux, il y a une grande lacune qui peut difficilement être comblée.
De même que l'on ne peut pas envisager de tester tous les couples d'instructions afin de détecter les problèmes de séquencement, il est impensable de tester toutes les combinaisons possibles de signaux (combinaisons à 2, 3 ou plus ?) avec chacun des codes-opération. Seule une connaissance détaillée du circuit comme dans le cas du test fabricant, ou un tirage au hasard comme dans le cas du test aléatoire, peuvent aboutir à une solution.
Cette remarque peut être étendue aux fautes dites "data dépendantes", c'est à dire aux fautes qui ne sont détectables qu'en présence d'une combinaison de données particulières. Dans ce cas, seul le test aléatoire semble pouvoir donner des résultats. Pour les autres on peut espérer trouver des opérandes permettant de vérifier complètement chaque opérateur, mais qu'en sera-t-il des combinaisons d'opérandes ?

• Le test fabricant est très efficace pour un produit stable comme le 6800. Il tire avantage de toutes les connaissances acquises durant la longue vie du produit.

• Le test aléatoire donne le meilleur résultat dans cette expérience, mais il faut néanmoins apporter quelques remarques. La séquence de test est très longue ($6{,}5 \times 10^6$ instructions) pour un circuit ne présentant qu'une faible complexité et ce test nécessite un testeur particulier. Si on se reporte aux chiffres du tableau 1 on voit que ce test souffre également de ce que le testeur ne concerne que le fautes booléennes. Un des avantages majeurs qu'il possède sur les tests fonctionnels déterministes est sa capacité à tester les signaux externes.
Ses principaux inconvénients sont donc :

- la longueur du programme de test mais cela n'est pas trop pénalisant car le programme n'est pas stocké, mais généré au moment même de l'application du test. Cela peut néanmoins le devenir si la longueur du test s'accroît énormément.
- la nécessité de définir un nouveau testeur pour chaque processeur, ou du moins si l'existence d'un testeur aléatoire universel est réaliste, de se donner la possibilité de filtrer les codes invalides différents pour chaque processeur. Il est évident que s'il est possible de construire des tables d'instructions valides pour le 6800, cela risque fort d'être irréalisable pour un 68000 voir un 68020. La complexité augmentant encore avec ce dernier exemple puisque la taille même de l'instruction (sans ses extensions possibles) va varier suivant son type ; elle peut en effet, être codée sur 1 ou 2 mots de 16 bits. De même, la validité des opérandes (leur taille, leur nature dans certains cas lorsqu'il s'agit d'adressage en particulier) va dépendre de l'instruction. Il devient assez difficile donc, dans ce cas, d'imaginer un système similaire.

En [Jac90], il est montré que la longueur de la séquence de test aléatoire peut être divisée par 10 si la probabilité d'occurrence des fautes est prise en compte.

III. INJECTION DE COUPURES DANS LE MICROPROCESSEUR 68000

Les résultats encourageants de l'expérience ont été immédiatement confirmés par un premier essai de faisabilité sur des microprocesseurs 68000 [Mar90].
Une faute simple a été injectée dans un premier jeu de 60 microprocesseurs. Les résultats obtenus sont donnés dans le tableau 3.
Malheureusement pour cette expérience, et toutes celles qui vont suivre pour le 68000, nous ne disposons pas de testeur aléatoire. Les seules comparaisons possibles vont avoir lieu entre les tests systématiques et le test fabricant.

Type de test	Circuits déclaré bons (sur 60)	Couverture de faute simple
fabricant	2	97,7 %
Systématique	9	85 %

Tableau 3 **:** résultats des tests pour l'injection d'une faute simple sur des 68000.

Les chiffres obtenus montrent une baisse du taux de couverture pour le test systématique (85% au lieu de 94%) et pour le test fabricant (97% au lieu de 98%) par rapport au 6800. Le faible nombre de pièces utilisé peut en partie expliquer cet écart, une pièce représentant alors 1,5%. Néanmoins l'ordre de grandeur est intéressant puisque l'on est assez loin des prévisions pessimistes sur l'efficacité du test fonctionnel [Klu88] qui donnent des valeurs de l'ordre de 70%.
L'analyse des circuits acceptés s'est avéré très difficile. Sur l'ensemble des circuits déclarés fautifs par le test fabricant et déclarés bons par le test systématique, 6 ont été détectés par le test de la ROM de microprogramme (nanoROM). Ce type de test, inapplicable par un utilisateur consiste à tester séparément la nanoROM et son environnement. Une entrée du circuit est portée à une valeur particulière permettant alors d'effectuer ces opérations de lecture/écriture. On remarquera que le mécanisme de décodage des codes invalides est testé par ce biais.
Il est à ce niveau difficile de savoir si les fautes détectées ont réellement ou pas une influence sur le fonctionnement "normal" du circuit.

Le but de l'expérimentation suivante était de confirmer ces premières valeurs obtenues et d'obtenir un échantillon plus conséquent de circuits traités du même type afin d'affiner les résultats quant à l'efficacité du test fonctionnel.

III.1. Buts de l'expérience

Les résultats obtenus lors du premier essai de faisabilité sur le 68000, donnait un taux de couverture de faute simple pour un test fonctionnel respectivement de aux alentours de 85 % (88 % si on prend le test fabricant comme référence), alors que pour l'expérience sur le 6800 ce taux était d'environ 90 % (95 % avec le test fabricant comme référence). Il était intéressant à ce niveau de chercher à savoir si cette baisse devait être mis sur le compte de l'imprécision des calculs ou sur celui d'une réelle diminution d'efficacité du à l'accroissement en complexité du circuit cible.
Le diagnostic de certaines fautes s'étant révélé assez aisé sur ce produit et le fait que des circuits passaient le test fabricant dans l'expérience précédente, nous laissait entrevoir l'espoir de détecter non pas plus de circuits que celui-ci mais peut être quelques circuits non vus fautifs par celui-ci.

Le principal reproche que l'on fait au test systématique est sa longueur. Nous avons dans cette seconde expérience tenté de réduire la taille des programmes de test générés par GAPT sans perte d'efficacité. La vérification de cette hypothèse est le second but important de cette expérience.
Les résultats obtenus dans cette expérience ont été publiés en [Mar91a][Mar91b].

III.2. Mise en oeuvre

Pour réaliser cette deuxième expérience nous avons réuni un lot de processeurs 68000 encapsulés déclarés bons après le test fabricant.

III.2.1. L'injection de défauts

Le matériel utilisé pour l'injection de défauts est le même que pour la première expérience, à savoir le laser de découpe. Les circuits étant encapsulés il n'a pas été possible d'envisager d'injecter des courts-circuits, cette opération nécessitant de dépassiver les circuits, opération toujours délicate à réussir et qui peut entraîner la destruction de pièces. Nous avons renoncé afin de conserver un lot assez important numériquement. La seconde raison qui nous a fait renoncer est qu'on ne peut mettre que 4 à 5 circuits maximum encapsulés dans la cloche centrale du laser à déposition. Le temps d'immobilisation des équipements et du personnel spécialisé nécessaire pour obtenir un échantillon de taille acceptable (une centaine de pièces) était trop important.

III.2.2. Les programmes de test

Deux types de programmes de test ont été appliqués : le test fabricant et deux programmes de test systématique (un test pseudo-exhaustif et un test réduit.)
Comme dans le cas du 6800, le test fabricant est le test Motorola enrichi par des vecteurs spécifiques dus aux nombreux retours client dans la phase de jeunesse du circuit.

III.2.2.a Le test pseudo-exhaustif

Généré par la méthode GAPT en mode pseudo-conformité, le programme de test comporte 50 modules d'environ 60 KOctets chacun, soit près de 3 MOctets de programme. Il passe en revue chaque code-opération avec un exemplaire de chaque mode d'adressage, dans toutes les tailles d'opérandes possibles (seuls les numéros du registre d'index, lorsqu'ils interviennent dans un mode d'adressage, sont tirés au hasard). Son principal défaut est également sa principale qualité. En effet, de par sa taille importante, le grand nombre d'opérandes aléatoires utilisé permet un test minutieux des blocs de la partie opérative. En cela on tend à se rapprocher du test purement aléatoire.

III.2.2.b Le test réduit

Afin de pouvoir réaliser cette expérience et permettre la génération de programmes de test pour des circuits encore plus complexes (comme le 68020 par exemple), nous ajoutés des nouvelles primitives au logiciel GAPT. Ces fonctionnalités permettent de choisir un nombre donné d'éléments parmi les N d'un même groupe. Par exemple, pour un mode d'adressage du type "déplacement + registre de base + index" le 68000 requiert l'emploi d'un des 8 registres de base (les registres A0 à A7) et d'un des 16 registres d'index (les registres D0 à D7 et A0 à A7). Dans GAPT, pour tester ce mode d'adressage, le choix est offert de combiner de 1 à 8 registres pour la base et de 1 à 16 registres pour l'index. Le nombre de cas générés pour ce mode d'adressage variera donc, suivant les cas, de 1 à 64 cas.
On mesure toute l'importance de cette fonctionnalité si on se rappelle les chiffres donnés dans la première partie. Le développement complet du mode d'adressage "indirect par registre adresse avec index et base de déplacement" du 68020 conduit à 3487 cas. Si les registres base et index et le mécanisme d'échelle (scaling) sont considérés comme des paramètres de ce mode d'adressage, seuls 18 cas restent à tester. La vérification de toutes les instructions ADD possibles de ce même microprocesseur demande de passer en revue environ 13×10^6 instructions, alors que l'hypothèse de paramétrisation conduit à n'en vérifier que 2100.

Le choix que nous avons donc fait est de considérer les registres Données et Adresses comme paramètres des divers modes d'adressage. Nous conservons un exemplaire de chaque taille d'opérande lorsque ceci est possible. L'importance de la réduction est considérable puisque nous obtenons un seul module de 90 KOctets. Ce programme de test sera appelé test réduit.

III.2.3. Le matériel de test

Le test fabricant a été réalisé par la société Thomson sur un testeur Sentry 20.
Les tests fonctionnels systématiques ont été appliqués sur le testeur TEMAC. Bien que plus ancien dans sa conception ce testeur a donné de meilleurs résultats que le testeur FUTE dans les précédentes expériences[8]. Ceci est dû au mécanisme d'observation par signature de tous les bus (bus données, adresses et contrôle) à chaque cycle, qui est plus complet que la simple observation à travers des écritures mémoire de FUTE.

III.3. Résultats

III.3.1. Présentation

Sur 140 circuits traités, 12 ont passé tous les tests sans détection d'erreur. N'ayant pas à cette époque les moyens d'analyser finement les causes de cette non détection, nous avons dû nous résoudre à mettre cela sur le compte d'erreurs de manipulation ou de fautes non efficaces[9]. Dans cette expérience, nous avons donc choisi de prendre comme référence absolue le test du fabricant. Pour un circuit stable comme le 68000, le taux d'échappement du test fabricant (circuits défectueux déclarés bons à tort) est très bas (de l'ordre de la centaine de p.p.m). Vu la taille relativement modeste de notre échantillon, nous avons considéré le test fabricant fiable à 100 %. Ceci fera l'objet d'une discussion dans la dernière partie de ce travail.
Le tableau 4 résume les résultats obtenus après application des trois différents tests.

Type de test	Circuits défectueux déclarés bons (sur 128)	Couverture pour une faute simple
Fabricant	0	100 %
pseudo-exhaustif	11	91,4 %
réduit	11	91,4 %

Tableau 4 : résultats du test du 68000 après injection d'une faute simple.

III.3.2. Discussion

Les valeurs obtenues de 91,4 % pour les tests fonctionnels vont dans le sens d'une confirmation des expériences précédentes, néanmoins il faut garder à l'esprit qu'un seul type de faute, à savoir la coupure de pistes d'Aluminium ou de Silicium polycristallin, a été introduit. Il

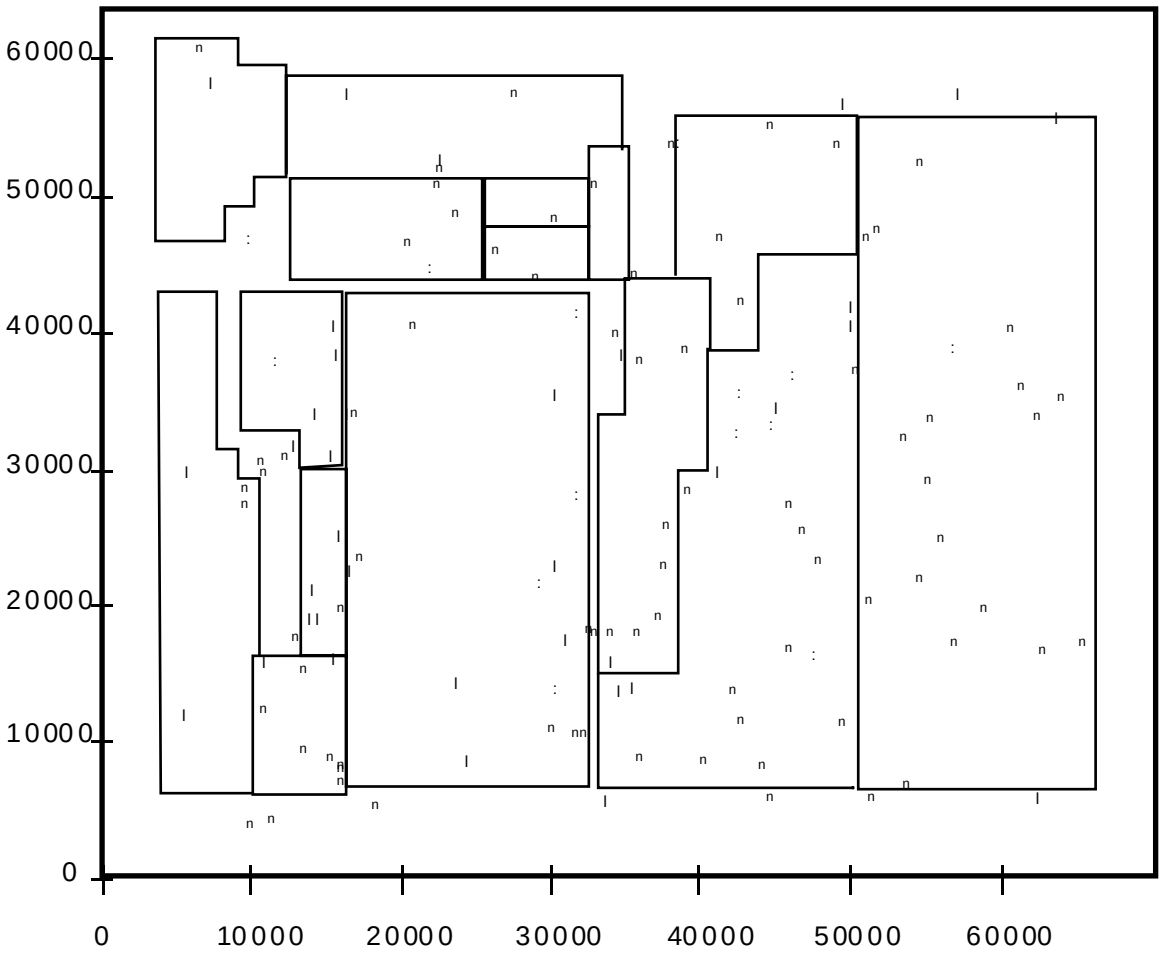
---

[8] *L'expérience a néanmoins du être vérifiée et terminée sur le testeur FUTE16, le testeur TEMAC faisant preuve d'instabilité avec une montée en température après quelques heures d'activité*

[9] *Cette hypothèse sera confirmée dans les expériences suivantes*

faut se garder de faire des conclusions hâtives mais prendre ces valeurs comme une tendance. En l'absence de modèles de fautes véritablement crédibles, ces chiffres ne sont pas à négliger. Un des objectifs que nous nous étions fixés lors de cette expérience est atteint : pour le type de fautes injectées et le type de circuit utilisé, il n'y a aucune différence entre le test pseudo-exhaustif et le test réduit. Il convient néanmoins là aussi d'être prudent : on aurait tort de rejeter trop vite l'apport que peut donner le grand nombre de valeurs aléatoires fourni par le test pseudo-exhaustif. La différence d'efficacité n'ayant pas pu être mesurée dans cette expérience, elle sera donc considérée ici comme négligeable, voire nulle.

La répartition des défauts et leur classement en fonction du type d'erreur générée, ou plutôt du mode de détection entraîné est montré dans la figure 2.8.
Cette figure fait ressortir que dans ce type d'expériences, toutes tentatives de corrélation entre la zone concernée par le défaut et la nature du comportement fautif induit semblent être vouées à l'échec si les détails de l'implantation sont inconnus. En effet les différents types d'erreurs issus des défauts implantés semblent être équiprobablement répartis. Pour pouvoir vérifier ou infirmer cette première impression, il aurait fallu réaliser une expérimentation spéciale, avec une injection ciblée dans chaque partie du microprocesseur (partie opérative, PLAs, ROM de microprogramme,…). Nous avons choisi de ne pas réaliser ce type d'expérience pour deux raisons :
- le nombre de circuit à traiter pour réaliser une telle expérience avec des échantillons significatifs aurait été trop important ;
- notre principe de base d'injection totalement aléatoire de défauts n'aurait pas été respecté.



: défauts qui ne créent pas d'erreurs

● défauts qui créent des erreurs de séquencement (boucles, problèmes de reset, exceptions,…)

- défauts qui créent d'autres erreurs (données, adresses, signaux, …)

Figure 2.8 **:** distribution des défauts sur la surface du circuit.

IV. INJECTION DE COURTS-CIRCUITS ET DE COUPURES DANS LE MICROPROCESSEUR 68000

Afin d'élargir le champs de nos recherches, dans cette expérience, des courts circuits seront créés.
Le but ici est le même que celui des deux expériences précédentes, à savoir essayer d'évaluer l'efficacité de tests fonctionnels de longueurs différentes, afin d'essayer de quantifier et/ou de qualifier cette efficacité et éventuellement de trouver quelle réduction raisonnable on peut appliquer à l'exhaustivité tout en obtenant les mêmes taux de couverture.
Les résultats seront publiés en [Mar92].

IV.1. Approche expérimentale

Dans le but de pouvoir injecter des défauts se manifestant par des courts-circuits, nous nous sommes procuré des plaquettes non passivées comportant un nombre de circuits bons suffisamment important pour réaliser ce travail. Les équipements laser décrits au paragraphe I ont été de nouveau employés pour toute la partie technique de la création de défauts.

La figure 2.9 résume l'approche utilisée [Mar91c].

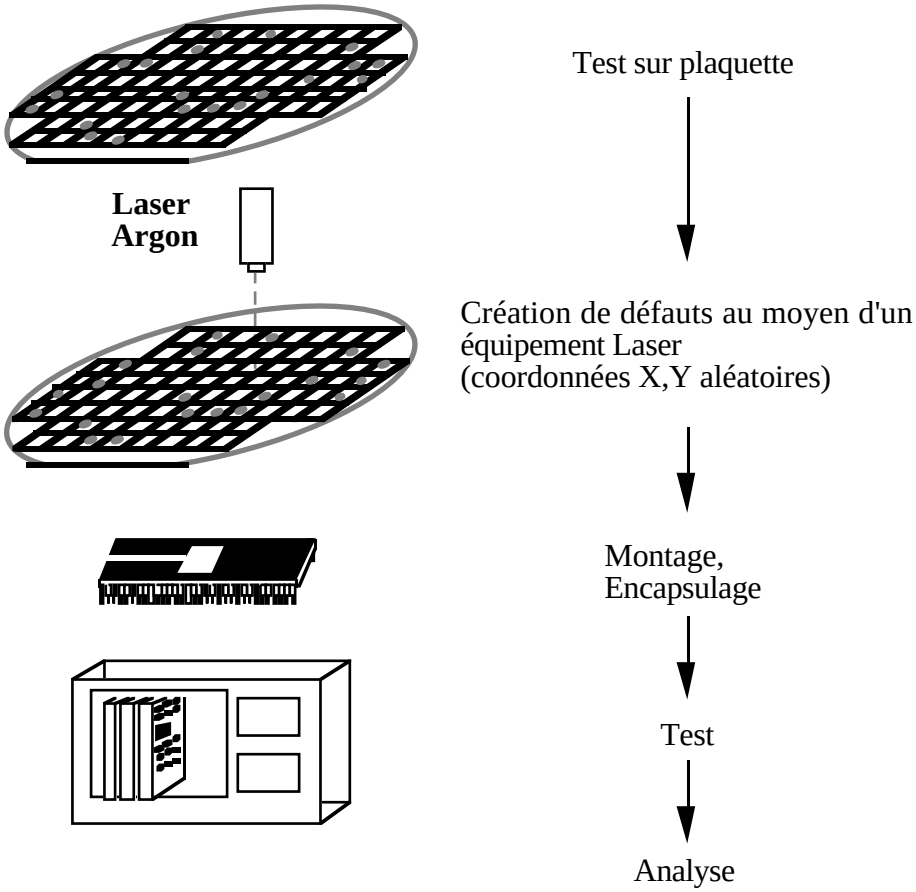


Figure 2.9 : démarche expérimentale.

Nous disposions au départ de 220 circuits bons disposés sur 5 plaquettes. Nous avons choisi de créer des courts circuits sur 2 plaquettes représentant 101 circuits bons et de réserver les 3 restants pour réaliser des coupures. Cette phase de l'expérimentation devait se faire assez rapidement afin d'éviter la détérioration des circuits par création d'importants courants de fuite qui apparaissent après un contact prolongé avec le gaz réactant.

Après injection des défauts (1 par circuit) les plaquettes ont été remises à la société Thomson qui a réalisé le montage et le test fabricant. Un problème lors de la découpe d'une plaquette à endommagé quelques pièces si bien qu'en sortie du montage nous disposions de 195 pièces présentant des défauts.
La figure 2.10 donne la méthode employée pour l'injection de défauts et le test des circuits défectueux.

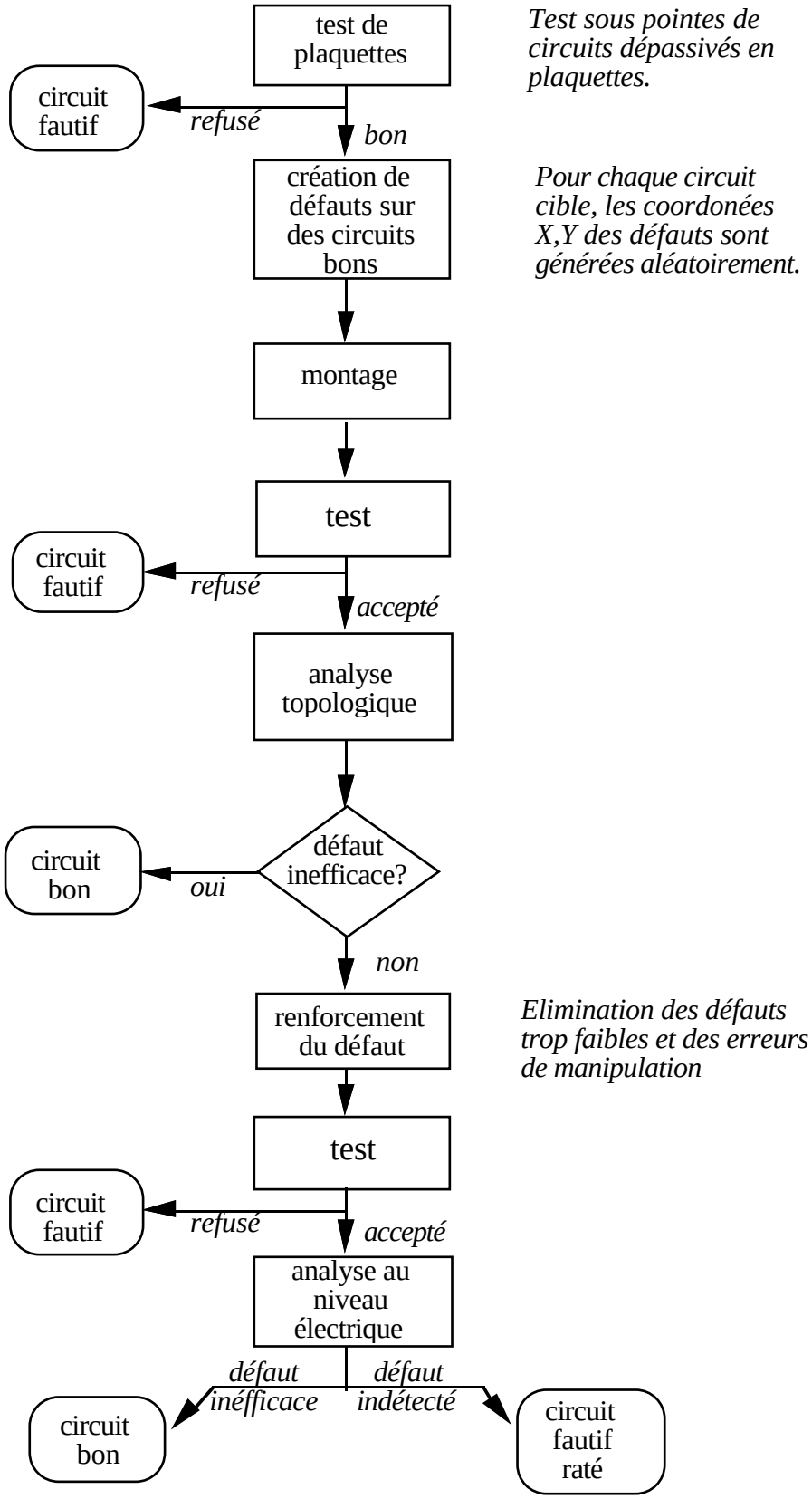


Figure 2.10 : méthode expérimentale pour l'injection laser de défauts ponctuels. Les deux types de défauts, coupures et courts-circuits, sont injectés dans des circuits bons sur plaquette dépassivées. Un défaut est placé dans chaque circuit par utilisation de coordonnées X, Y générées aléatoirement.

Après injection des défauts, les circuits sont montés et encapsulés, puis testés par différentes méthodes en utilisant les environnements de test adéquats.
Les circuits défectueux qui passent un test donné sont analysés afin de faire la distinction entre les défauts ne créant pas d'erreur et les défauts non détectés par la séquence considérée. Pour chacun des circuits passant les séquences de test utilisées, l'implantation de la région entourant le défaut est analysée afin d'éliminer les circuits portant un défaut sans conséquence électrique ou logique (par exemple : coupure de lignes bouclées ou de lignes de polarisation...). Pour les autres circuits le défaut est renforcé afin d'éviter les problèmes d'erreur de manipulation ou de défaut inefficace (nous rappelons qu'une coupure doit avoir une résistance équivalente de 10 KOhm pour être efficace [Auv86]).
Ces circuits sont ensuite testés une nouvelle fois. Pour chaque circuit qui continue de passer les tests sans détection d'erreur, une analyse électrique est nécessaire afin de déterminer si les vecteurs de test destinés à stimuler la partie du circuit concernée sont efficaces ou pas. Ceci peut permettre d'améliorer la couverture d'une séquence de test par addition de vecteurs de test adéquats.

IV.2. Programmes de test

Outre le test de fin de fabrication réalisé par Thomson avec ses propres programmes de test, issus du test originel Motorola, deux tests fonctionnels ont été utilisés : un test systématique pseudo-exhaustif et un test fonctionnel minimal. Afin de donner une borne inférieure à l'efficacité des tests, un programme utilisateur simple a été employé pour mettre en évidences les erreurs triviales à détecter.

IV.2.1. Le test fonctionnel systématique

En fait nous nous sommes servi ici de deux programmes de tests différents. Forts des résultats de l'expérience précédente, le test dit "réduit" a été appliqué. En cas d'échec dans la détection, le test pseudo-exhaustif est venu renforcer la détection. On notera tout de suite qu'aucune différence n'a été mise en évidence entre les deux test. Sauf en cas de précision explicite, l'appellation test systématique recouvrira les programmes de test fonctionnel pseudo-exhaustif et réduit.

IV.2.2. Le test fonctionnel minimal

Un programme de test de taille réduite a été spécialement développé pour cette expérience. Ce programme de test passe en revue tous les types d'instruction (chaque mnémonique différent constituant un type d'instruction différente). Chacun d'entre eux n'est utilisé qu'avec une seule taille d'opérande. Lorsque plusieurs tailles sont possibles c'est aléatoirement qu'une taille est choisie. Pour chaque instruction on utilise en tant que source et destination de l'instruction (quand c'est possible) un exemple de registre et un exemple d'adressage mémoire. Ceci conduit à envisager au maximum 4 cas.

Exemple : instructions ADD et MOVE

ADD.B	D0, D3			MOVEL	D0, D7
ADD.W	D4, (A6)			MOVEB	D4, -(A2)
ADD.L	(A3)+, D5		MOVEL	16(A0), D3	
(ADD mem,mem non valide)		MOVEW	(0324h).W, 4(PC)		
- 3 cas					- 4 cas

Cette méthode conduit à construire un test de 13 KOctets (programme + données).

IV.2.3. Le programme de tri

Le programme de tri est un programme classique de “quick-sort”. Il effectue un tri d’un tableau de quelques entiers et écrit en mémoire le résultat du tri. Ce programme est obtenu par compilation d’un programme source écrit en langage C. Sa taille est inférieure à 1 KOctet.

Le tableau 5 résume les particularités de chaque programme de test utilisé.

Nom du programme	Taille en octets	Activation
Pseudo-exhaustif	24 x 128K	Chaque combinaison valide de code-opération et de mode d’adressage (le N° de registre d’index quand il est nécessaire est tiré au hasard). Toutes les tailles d’opérandes
Réduit	90 K	Chaque code-opération avec une combinaison choisie au hasard de chaque mode d’adressage (dans chaque mode d’adressage le N° de registre est un paramètre). Toutes les tailles d’opérandes.
Minimal	13 K	Chaque code-opération avec un exemple de registre et de mode d’adressage mémoire en source et/ou destination (4 possibilités maximum). La taille des opérandes est choisie au hasard.
Tri	1 K	Sous-ensemble de registres, opérateur et instruction dépendant de l’algorithme et du compilateur utilisé

Tableau 5 : les différents programmes utilisés dans la 3ème expérience.

IV.3. Equipement de test

Pour cette troisième expérience, seul le testeur FUTE16 a pu être utilisé, le testeur TEMAC étant désormais hors service (matériel obsolète très difficile à remettre en service). Le test fabricant est quant à lui, toujours exécuté sur un testeur Sentry 20.
Le problème posé par l’utilisation unique de FUTE16 va donc se situer autour du test des signaux asynchrones, comme nous l’avons vu par le passé.

IV.4. Déroulement de l'expérience et résultats

Les 195 circuits défectueux après montage se répartissent ainsi : 87 courts circuits et 108 coupures (57 % de pistes d'aluminium, 43 % de pistes de Silicium polycristallin). La figure 2.11 montre la répartition originelle des 220 défauts sur le plan de masse du circuit.

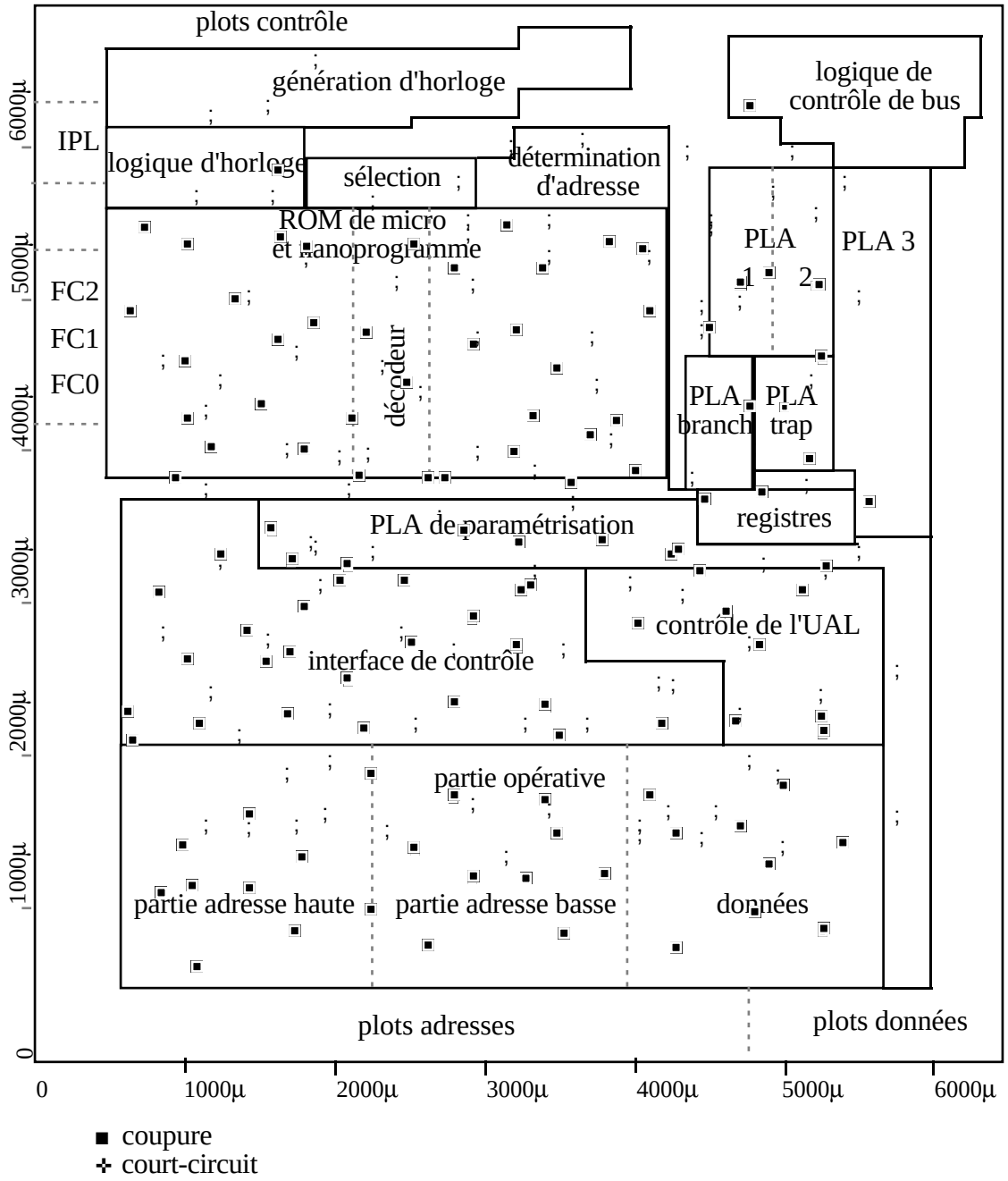


Figure 2.11 **:** répartition des coupures et courts-circuits sur le circuit.

IV.4.1. Résultats des premiers tests

Le tableau 6 donne l'ensemble des résultats obtenus par chaque test. Les résultats complets pièce par pièce sont fournis en annexe (annexe A).

Type de test	Nbre courts circuits non détectés (sur 87)	Nbre de coupures non détectées (sur 108)	Nbre total de pièces non détectées (sur 195)
Fabricant	3	20	23
Systématique	9	25	34
Minimal	11	25	36
Tri	19	37	56

Tableau 6: résultats des différents types de test.

IV.4.2. Analyse des pièces présentant un défaut non détecté

L'analyse a lieu à deux niveaux distincts :

- l'analyse des pièces acceptés par tous les tests. Dans ce cas il convient d'éliminer les erreurs de manipulation et les défauts inactifs ;
- l'analyse des pièces déclarées défectueuses par certains tests et pas par d'autres. Dans ce cas le but recherché est de fixer les limitations d'un test donné.

IV.4.2.a Analyse des pièces déclarées bonnes par tous les tests

Cette partie est difficile à réaliser sans connaissance de l'implantation physique et du schéma électrique équivalent. Ce sont donc des ingénieurs de la société Thomson qui s'en sont chargé. Les 23 pièces à analyser se répartissaient comme suit (tableau 7) :

N° Plaquette	Type de défaut	Numéro de pièces
A	coupure	1 - 2 - 3 - 4
B	coupure	1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10
C	coupure	1 - 2 - 3 - 4 - 5 - 6
D	court-circuit	1
E	court-circuit	1 - 2

Tableau 7 : répartition des pièces acceptées par tous les tests.

La phase de recherche des coordonnées des défauts créés a été longue et fastidieuse. Elle s'est faite, soit au microscope optique pour certaines pièces, soit grâce au microscope du laser de découpe pour d'autres en passant alors en revue l'ensemble des coordonnées d'une plaquette. La deuxième méthode s'est avérée la plus sûre, mais demandant un temps assez long (de l'ordre de 3 minutes pour le premier couple de coordonnées, puis de minute en minute pour les couples suivants).

Le défaut créé n'a pas pu être retrouvé sur 2 circuits (A/3 et E/2). Ceux-ci ont donc été sortis de la liste des circuits à retester et ils sont considérés comme bons suite à une erreur de manipulation (défauts créés sur un circuit voisin sur la plaquette, défaut créé sur un circuit voisin marqué comme défectueux lors du tri sur plaquette,…).
L'ensemble des résultats est donné dans le tableau 8.

Les 8 circuits portant des défauts inefficaces, ou considérés comme erreurs de manipulation ont été retirés du lot, soit les circuits : A/2, A/3, B/5, B/8, C/2, C/3, C/4, E/2. La procédure de création de courts-circuits étant lourde à mettre en oeuvre, les courts-circuits incomplets n'ont pas été renforcés ; les circuits D/1 et E/1 ont donc été introduits dans la liste précédente portant à 10 le nombre de circuits effectivement écartés des mesures.
Pour les autres circuits le défaut a été renforcé puis les circuits testés de nouveau (les résultats de ces test seront donnés et discutés dans le paragraphe c).

IV.4.2.b Cas de circuits trouvés défectueux par au moins un test

Si on analyse les résultats des premiers tests (fournis en annexe), on trouve une parfaite inclusion des résultats des différents tests. On va donc pouvoir étudier les différences trouvées entre les tests. Seuls les circuits échappant au programme de tri ne seront pas analysés, celui-ci n'étant pas un programme de test à proprement parler ; il nous donne seulement une indication sur le nombre de défauts créant des fautes facilement détectables. On notera une différence de 20 circuits avec le test minimal et de 33 circuits avec les test fabricant. 139 circuits sont refusés par le programme de tri

N° de circuit	Type de défaut	Observation
Plaquette A		
1	coupure incomplète	inefficace ; à refaire
2	ligne de Si-Poly d'un PLA coupée en bout de ligne	inefficace
3	défaut non identifié	erreur de manipulation
4	coupure dans l'additionneur d'adresse partie haute bit 17	vérifier coupure
Plaquette B		
1	bus inter PLAs ; coupure douteuse	vérifier coupure
2	coupure non faite (Si-Poly trop près d'un Alu qui a absorbé l'énergie)	inefficace
3	ligne de sortie utilisée de la nanoROM	vérifier coupure
4	bus (adresse ou donnée) de la partie donnée de la P.O.	vérifier coupure
5	coupure ligne d'alimentation (Vdd) transfert par le caisson	inefficace
6	switch de bus données et adresse basse P.O.	vérifier coupure
7	coupure douteuse	vérifier coupure
8	colonne non utilisée de la nanoROM	inefficace
9	coupure incomplète	inefficace ; à refaire
10		inefficace ; à refaire

Plaquette C		
1	coupure mal faite	inefficace ; à refaire
2	coupure provoquant la déconnexion d'une partie d'accélérateur (aucune influence fonctionnelle)	inefficace
3	alimentation bouclée	inefficace
4	coupure ligne de polarisation de substrat	inefficace
5	alimentation de grilles ; résistance équivalente de la coupure trop faible	inefficace ; à renforcer
6	alimentation de grille	vérifier coupure
Plaquette D		
1	court circuit entre 2 lignes non utilisées d'un PLA	inefficace
Plaquette E		
1	court circuit incomplet	inefficace
2	défaut non identifié	erreur de manipulation

Tableau 8 : analyse des 23 pièces acceptées par tous les tests.

• Différence entre le test minimal et le test réduit
2 circuits qui échappent au test minimal font la différence avec le test réduit. Ce sont les circuits D/11 et D/45.

- Pour le circuit D/11 : la mise à 0 ou à 1 du bit V du registre d'état, en accord avec le résultat d'une instruction ASL (Arithmetic Shift Left) ne se fait pas. Pour mettre en évidence cette faute, il faut des valeurs d'opérandes ayant les 2 premiers bits dans des configurations précises.

  exemple : 00 ou 11 → mise à zéro puis 01 ou 10 → mise à 1.

  Le programme Minimal ne balaie pas assez de valeurs d'opérandes pour cela.
- Pour le circuit D/45 : les instructions de type immédiat (ADDI, SUBI, ANDI ...) en format "long" montrent une erreur avec le mode d'adressage indirect par registre avec prédécrémentation. Dans le programme minimal, le format "long" pour ce type d'instructions n'existe que pour ADDI.L #val, depl(An) et ORI.L #val, D_n ; le mode d'adressage incriminé n'est testé qu'en format "word" avec SUBI.W #val, (-An) et EORI.W # val, (-An).

Remarques :

- La première erreur confirme l'hypothèse de perte relative d'efficacité au niveau du test lorsque l'on réduit la taille du programme donc le nombre d'opérandes aléatoires engagés dans le test de chaque instruction.
- La deuxième erreur confirme la nécessité de tester un exemplaire de chaque mode d'adressage pour chaque instruction. Une connaissance approfondie de la partie contrôle permettrait sûrement de réduire ceci à un exemplaire par famille d'adressage.

• Différence entre le programme réduit et le programme pseudo-exhaustif
Comme il a été précisé au début de cette expérience, nous n'avons pas pu mettre en évidence des différences entre le programme fonctionnel réduit et le programme fonctionnel pseudo-exhaustif. C'est pourquoi dans les tableaux les deux résultats sont regroupés sous le terme "test systématique".
• Différences entre les programmes systématiques et le test fabricant
L'analyse des circuits qui échappent au test systématique s'est fait par décompilation, compréhension et analyse de la séquence de motifs de test du programme du fabricant, concernée par l'erreur. Cette portion en moyenne de 30 vecteurs s'est portée à plus d'une cinquantaine dans certains cas. Le résultat des analyses est donné dans le tableau 9.

N° de pièce	Séquence de vecteur détectant la faute	Observation

Plaquette A pièce 24	Traitement d'exception : Lors du traitement d'une adresse impaire en mode utilisateur on note un mauvais empilement de la valeur du compteur programme	Exception non testée
Plaquette B pièce 11	Traitement d'exception : traitement de code illégal en mode utilisateur.	Exception non testée
pièce 32	Traitement des interruptions. Erreurs dans la prise en compte du masque d'interruption à un niveau donné.	Toutes les combinaisons possibles doivent être testée
Plaquette C pièce 31	Après la séquence de reset, le signal Halt (bidirectionnel) est modifié à tort.	Montée du signal sans cause fonctionnelle ?
pièce 32	Détection d'erreurs en mode TEST.	Mode inaccessible à l'utilisateur ; faute sans conséquence fonctionnelle
Plaquette D pièce 32 pièce 33 pièce 41 pièce 47	Mise en HALT Sortie E (émulation du bus 6800) erronée après reset Mise en HALT après temporisation Mise en HALT après test de l'arbitre de bus. Erreur sur les signaux de contrôle.	Pas de conséquence fonctionnelles
Plaquette E pièce 26	Faute dans test de synchronisation avec des périphériques. Faute sur signal E	Connexion au périphérique non testée
pièce 28	Erreur dans la reconnaissance du masque d'interruption pour un niveau donné.	Toutes les combinaisons possibles niveau masque doivent être traitées.

Tableau 9 : analyse des 11 circuits acceptés par le test systématique.

Les fautes pour les 11 circuits qui échappent au test fonctionnel systématique sont résumées dans le tableau 10.

Type d'erreur rencontré	Nombre de circuit
Opération mettant en jeu le signal HALT	4
Opération mettant en jeu le signal de sortie E (émulation du bus 6800)	2
Traitement d'instructions illégales et d'exceptions	2
Traitement d'interruption	2
Mode Test	1

Tableau 10 : classement des circuits acceptés par le test fonctionnel en fonction du type d'erreur engendré

Ces résultats confirment les résultats déjà obtenus dans les expériences précédentes. Les principales difficultés rencontrées lorsque l'on génère un programme de test fonctionnel à partir des instructions utilisateur concernent la vérification des signaux asynchrones.
Nous noterons au passage que vu leur manifestation, quelques unes de ces erreurs auraient pu être mises en évidence en utilisant le testeur TEMAC grâce à sa signature du bus contrôle.
Certains phénomènes testés, notamment le comportement en présence du signal HALT en entrée ou son positionnement en sortie, sont difficiles à comprendre avec la seule aide du manuel utilisateur.
De même, le test des exceptions pose des problèmes. Le premier d'entre eux est qu'il est assez compliqué de tester des exceptions en mode utilisateur, car dans ce cas un basculement a lieu vers le mode superviseur et il est donc difficile de revenir de façon simple au programme général ; ce qui implique que ces vérifications sont faites pour la plupart en mode superviseur.
Le deuxième problème est celui du test des codes invalides. En effet, sans information sur le décodage des codes invalides la seule solution serait de tester toutes les combinaisons invalides, tâche qui est assez complexe à mettre en oeuvre et qui serait sûrement assez longue à exécuter.
De plus, après le traitement des instructions invalides au niveau code-opération, faudrait-il envisager de tester toutes les combinaisons invalides avec les extensions quand elles existent ?

IV.4.3. Résultats après deuxième test

Le défaut à pu être renforcé pour 11 des 13 circuits présentant un défaut suspect ; pour les 2 autres une dégradation causée par l'opération de décapotage à rendu cela inutile, les circuits étant définitivement hors service.
La composition finale de l'échantillon est résumé dans le tableau 11.

	nombre de circuits après montage.	pièces défectueuses	pièces rejetées (erreur / inefficaces)	pièces retestées	nombre de circuits de l' échantillon final
coupures	108	88	7 (1 / 6)	11	99
courts-circuits	87	84	3 (1 / 2)	0	84
total	195	172	10 (2 / 8)	11	183

Tableau 11 : composition de l'échantillon après renforcement des défauts.

Après renforcement des défauts, les 11 circuits ont été détectés fautifs par le test fabricant. Seul le circuit B/4 n'a pas été refusé par le test systématique. Les résultats enrichis par les nouvelles informations sont donnés dans le tableau 12.

Type de test	Nbre courts circuits non détectés (sur 84)	Nbre de coupures non détectées (sur 99)	Nbre total de pièces non détectées (sur 183)
Fabricant	0	0	0
Systématique	6	6	12
Minimal	8	6	14
Tri	16	21	37

Tableau 12 : nombre de pièces acceptées par chacun des tests.

Pour chaque type de test et chaque type de défaut, un taux de couverture de fautes booléennes à pu être calculé pour l'échantillon final de 183 pièces. Les résultats sont donnés dans le tableau 13.

Type de test	taux de couverture pour les courts-circuits	taux de couverture pour les coupures	taux de couverture total
Fabricant	100%	100%	100%
Systématique	92,8%	93,9%	93,4%
Minimal	90,5%	93,9%	92,3%
Tri	80,9%	78,8%	79,8%

Tableau 13 : taux de couverture de fautes booléennes pour chacun des tests.

Les résultats ci-dessus peuvent être condensés dans des courbes montrant l'évolution du taux de couverture en fonction de la taille du programme utilisé (figure 2.12).

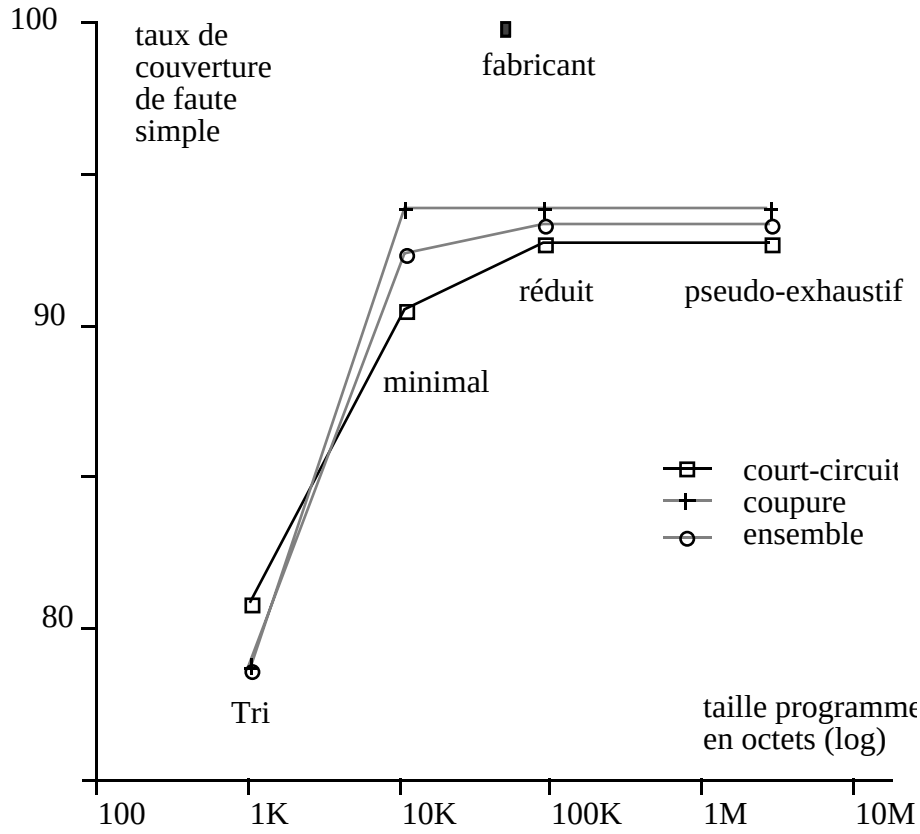


Figure 2.12 : taux de couverture en fonction de la taille des programmes de test.

Les courbes obtenues pour les différents types de fautes montrent un tracé classique. Les valeurs sont trop proches pour donner une conclusion sur l'efficacité relative par rapport aux coupures ou aux courts-circuits. On peut assimiler les deux résultats et considérer uniquement la courbe pour l'ensemble des deux types de fautes.

Il est également intéressant de noter que comme pour le 6800, le test fabricant se situe largement en dehors de ces courbes, preuve s'il en fallait que celui-ci est bien compacté, ou plutôt que les

programmes de test fonctionnel sont très redondants si on considère le test d'un chemin de données particulier.
Un dernier point intéressant à constater est que 80% des fautes sont mises en évidence par un simple programme de tri de moins de 1 KOctets, données comprises. On peut constater au vu des résultats du premier test, que très peu de circuits ont reçu un défaut créant une faute que j'appellerais "fine", c'est à dire une faute qui n'est visible qu'avec un nombre restreint d'instructions et/ou de combinaisons d'adressages. Comme fautes fines on peut citer sûrement celles affectant les 2 circuits qui font la différence entre le test Minimal et le test Systématique. Même si on inclut dans cet ensemble les fautes ratées par le test Systématique, on arrive seulement à une dizaine de pièces sur 200, c'est à dire environ 5%, ce qui est loin de nos prévisions. Nous pensions en effet que certains courts-circuits ou coupures situés dans la partie opérative ou dans la ROM de microprogramme (nanoROM) pouvaient avoir pour conséquences des fautes difficiles à mettre en évidence, ce qui ne semble pas être le cas.

IV.5. Discussion

Les résultats obtenus confirment donc ceux des deux expériences précédentes.
La borne supérieure du test fonctionnel, pour des défauts de type coupure et court-circuit, se situe aux alentours de 93 %. Les écarts avec les 100 % obtenus par le test fabricant sont essentiellement dus au traitement inefficace du test des signaux asynchrones et du traitement des exceptions.
A ce stade on peut reposer la question soulevée un peu plus tôt : comment tester les signaux ? Sans plus d'information que le manuel utilisateur, cela semble être une tâche difficile. La deuxième question posée par ces résultats est comment doit-on tester les codes invalides ? Enfin la dernière question issue de ces résultats provient de l'analyse du circuit B/4 responsable de l'écart entre le test fonctionnel et le test fabricant : comment comprendre la signification des motifs du test fabricant ?
En effet le circuit B/4 porte une coupure sur un bus transférant bit à bit les informations des différentes parties de la partie opérative. Or, cette coupure, une première fois mise en doute, a été renforcée. Le test fabricant a alors diagnostiqué cette pièce comme mauvaise alors que le test systématique continue de la trouver bonne. La détection de la défaillance par le test fabricant soulève la question précédente. En effet, c'est au niveau du test de la nanoROM que cette faute est mise en évidence, alors que d'après les informations que nous possédons, la partie opérative est normalement isolée durant ce test. Il est possible que ces chemins de données restent actifs expliquant le phénomène. Mais alors, comme se fait-il qu'aucune autre partie du test fabricant (partie fonctionnelle par exemple) ne le mette en évidence ?. Pourquoi le test fonctionnel ne voit-il pas cette erreur ? L'explication finale est peut être à chercher du côté des fréquences. La fréquence du test de la nanoROM n'est que de quelques centaines de Kilo Hertz alors que la fréquence normale de fonctionnement du microprocesseur est comprise entre 4 et 16 MégaHertz. Il peut être envisageable que l'utilisation de ce bus, fautif uniquement en transfert de données à basse fréquence, mette en évidence cette faute alors qu'à haute fréquence et en fonctionnement normal la faute soit masquée. Malheureusement nous ne disposons à l'heure actuelle d'aucune information pour confirmer ou infirmer cette dernière hypothèse. Cette dernière analyse soulève encore une nouvelle question : comment doit-on considérer un défaut qui ne manifeste une erreur qu'en dehors des plages normales ou du mode normal de fonctionnement ? Toutes ces questions issues de cette expérience et des précédentes seront analysées dans la troisième partie de ce travail.

Chapitre 3

Analyse Critique des Expériences

Nous allons faire porter cette analyse principalement sur trois points :
- le bien fondé de la campagne d'expérimentation réalisée et, à un niveau général, sur l'ensemble des résultats obtenus ;
- la méthode d'injection des défauts employée ;
- l'avenir du test fonctionnel en général, et du système GAPT en particulier.

I. SUR L'EXPERIMENTATION

Depuis bientôt dix ans que le système GAPT existe, ou du moins depuis que l'on fait du test fonctionnel au laboratoire, une question nous a été sans cesse posée par les industriels avec qui nous étions en relation : *quelle est l'efficacité de la méthode de test fonctionnelle utilisée ?* Inlassablement nous devions répondre qu'on ne peut chiffrer cette efficacité en taux de couverture puisque la méthode employée n'est pas basée sur des hypothèses d'erreur.

Le but des expériences présentées dans ce travail était donc d'essayer de quantifier et/ou de qualifier l'efficacité du test fonctionnel. On peut dire que ce but est atteint. En effet, nous avons constitué des échantillons de circuits, avec une distribution aléatoire de défauts (un défaut par pièce) qui, du point de vue statistique, peuvent s'assimiler à un sondage.

Soient :

N la taille de l'échantillon

n le nombre de "oui" (défauts couverts)

p le pourcentage (inconnu) de "oui" (taux de couverture que l'on cherche a estimer)

n suit une loi binomiale : $Prob(n = m) = C_N^m \; p^m (1-p)^{N-m}$

avec pour moyenne : $E(n) = Np$ et pour variance : $sigma^2(n) = Np(1-p)$

d'où les propriétés de l'estimateur n/N :

$E(n/N) = p$

$sigma^2(n/N) = p(1-p)/N$

Au delà de $Np > 10$, n/N suit a peu près une loi Gaussienne de moyenne : $E(n/N) = p$

et de variance : $sigma^2(n/N) = p(1-p)/N$

L'application de l'approximation Gaussienne aux valeurs obtenues pour chacune des expériences, donne les résultats présentés dans le tableau 14. Afin de conserver une cohérence aux résultats, les valeurs considérées sont celles obtenues par le test fonctionnel par rapport au test fabricant. On ne tient compte que des fautes booléennes. Les résultats des trois expériences sur le 68000 sont présentées, ainsi qu'un résultat global pour ce circuit.

type d'expérience	Nbre de pièces	Nbre de pièces refusées	taux de couverture	sigma	2 sigma	p à 95% de confiance
6800	68	3	0,956	0,025	0,050	95,6 ± 5 %

68000 (1)	58	7	0,879	0,043	0,086	87,9 ± 8,6%
68000 (2)	128	11	0,914	0,025	0,050	91,4 ± 5%
68000 (3)	183	12	0,934	0,018	0,036	93,4 ± 3,6%
total 68000	369	30	0,918	0,014	0,028	91,8 ± 2,8%

Tableau 14 : taux de couverture des différentes expériences (confiance à 95%)

On peut noter que l'efficacité du test fonctionnel GAPT est plus élevée pour le 6800 (≈96 %) que pour le 68000 (≈92 %). Bien que l'incertitude sur les résultats du 6800 soit assez grande(± 5%), ces valeurs montrent une baisse "inversement proportionnelle" à la complexité du circuit à tester. On peut étayer cette dernière hypothèse en regardant de près le traitement des signaux asynchrones qui est le point faible du test fonctionnel.
Pour le 6800 un seul niveau de masquage existe pour les interruptions de type externe (IRQ) alors qu'il en existe 7 pour le 68000. De même, le décodage des codes invalides est beaucoup plus complexe sur le 68000 puisque, pour les instructions complexes, un code-opération valide peut déboucher sur une instruction invalide à la lecture d'une mauvaise extension. Le codage des instructions du 68000 peut utiliser jusqu'à 7 mots de 16 bits pour une seule instruction. Ce phénomène s'accentue encore si on prend le 68020, élément suivant dans la famille Motorola, où le codage d'une instruction peut nécessiter jusqu'à 11 mots de 16 bits.

Néanmoins quelque soit le chiffre retenu pour l'efficacité du test fonctionnel, mettons 92 %, cette valeur est de beaucoup inférieure au résultat obtenu par le test fabricant. Alors pourquoi réaliser un test fonctionnel s'il existe un meilleur test ?
On doit ici faire un léger retour en arrière. Les tests fonctionnels sont nés quand les utilisateurs, ne possédant pas d'autres informations que celles fournies par les manuels d'utilisation, ont voulu faire des tests, parce qu'ils doutaient des tests fabricants. Si on se reporte aux travaux de Hnatek en 1987 [Hna87] et de Westover en 1989 [Wes89] qui mettent en doute les valeurs d'efficacité annoncées par les fabricants de circuits intégrés, on a envie de dire que les utilisateurs ont raison.
En effet, d'après les travaux décrits en [Hna87] qui font part de résultats de test obtenus par des utilisateurs, les différences entre les chiffres des fabricants et ceux des usagers sont considérables. En nombre de pièces défectueuses échappant aux tests, alors que les fabricants revendiquent des résultats entre 10 et 200 ppm pour les tests de sortie des circuits (certains allant même jusqu'à zéro défauts), les usagers annoncent que leurs tests d'entrée de circuits du commerce font ressortir des taux d'échec allant de 2000 à 20 000 ppm. Ces chiffres sont même poussés jusqu'à une fourchette de 140 000 à 200 000 ppm pour les circuits destinés à des applications à haute sûreté de fonctionnement militaires ou spatiales.
La polémique sur ces chiffres provient du fait qu'il n'existe pas de normalisation pour l'évaluation de l'efficacité des tests. Les travaux rapportés en [Wes89] qui argumentent pour la nécessité de tests utilisateurs, montrent que les stratégies de test doivent être bâties d'après un compromis entre le coût du test et le degré de risque accepté. Cela peut aller d'un simple test à 25° C sur quelques échantillons pris aux hasard dans certains lots (faible coût, haut risque), jusqu'au test systématique de tous les circuits de tous les lots après leur avoir fait subir toute une gamme de contraintes physiques comme des test à 70°, des cycles et des chocs de température, l'application de haut voltage pendant 168 heures, …(fort coût, faible risque). En appliquant de telles stratégies sur des lots de circuits, la différence entre la qualité annoncée par le fabricant et celle mesurée devient frappante. Par exemple, dans le même article, des tests sur des mémoires annoncées avec un taux d'échappement de 100 ppm montrent pour certains lots des taux d'échec de 16 000 ppm.
De leurs côté certains fabricants se contenteraient de compter les retours client et de réduire encore ce chiffre d'environ 50 %, partie que représente, selon eux, les retours dus à une mauvaise utilisation des circuits incriminés. Ces résultats sont bien entendu faussés. En effet, pour conserver de bons rapports avec son fournisseur, pour des questions de coût (un renvoi au fournisseur coûtant plus cher que la pièce défectueuse), et pour d'autres raisons plus ou moins

valables [Tho89], de nombreuses de pièces défectueuses finissent dans des poubelles et ne font pas l'objet de retour au fabricant.
De plus, pour les processeurs, et notamment pour les processeurs complexes, il est reconnu que seule une faible partie des potentialités est en fait utilisé lors de leur fonctionnement dans l'application finale. Ce sont d'ailleurs ces réflexions qui sont en partie à la base du retour vers une simplification dans le jeu d'instructions pour les processeurs RISC. On peut donc penser, et cela s'est d'ailleurs vérifié dans le passé[10], que des fautes peuvent exister à l'état latent dans des processeurs montés dans diverses applications et que seules quelques unes d'entre elles, les manifestent. Ceci provoque un nombre de retours client inférieur au véritable taux d'échec des tests du fabricant.
S'il est possible de construire des tests efficaces, lorsqu'il s'agit de mémoires ou de circuits logiques de faible complexité, il n'en est pas de même pour les processeurs. La question que l'on doit se poser est alors : faut-il croire les fabricants, ou doit-on se dire que même si ils annoncent des chiffres optimistes, on n'a pas les moyens de faire mieux ? Il semblerait malheureusement que la dernière remarque soit la bonne.

Après ces quelques critiques sur l'utilité du test utilisateur, nous pouvons nous pencher sur les résultats de nos expériences.
Les grandes leçons que nous pouvons tirer de ces expériences sont :

- on ne sait pas faire de tests de signaux corrects à partir du manuel utilisateur,

- il est difficile de comprendre la nature des erreurs au vu des résultats du test fabricant.

La première affirmation a été bien étayée au cours de la description de ces expériences. Au vu de certains extraits des séquences de test du fabricant que nous ne pouvons pas reproduire ici puisque strictement confidentiel, il est clair que des informations manquent à l'utilisateur pour bâtir des tests. Si on se reporte aux expériences 6800 et notamment aux fautes non détectées par le test fonctionnel, comment expliquer qu'une séquence du test du fabricant positionne en même temps IRQ, NMI et HALT, si ce n'est pas parce qu'elle est dédiée au test d'un chemin particulier ? Il en est de même pour les expériences sur le 68000, où la décompilation des séquences de test du fabricant mettant en évidence les différences entre les deux types de test (utilisateur et fabricant), laisse apparaître des enchaînements étranges. Notamment certains motifs sont répétés plusieurs fois sans modifications, attentes qui correspondent à coup sûr à un aspect particulier qui nous est totalement inconnu en tant qu'utilisateurs, sinon pourquoi existeraient-elles lorsque l'on sait dans quel esprit de compaction sont fait ces tests ?
La deuxième affirmation a déjà été étayée dans la partie précédente lors de l'essai de compréhension de la faute B/4 ratée par le test fonctionnel.

II. SUR LA METHODE D'INJECTION DE DEFAUTS

La seule possibilité qui s'offrait à nous pour évaluer l'efficacité des programmes de test était l'injection de fautes à un niveau physique car nous ne disposions pas d'outils performants de simulation, et nous n'avions pas les moyens d'en élaborer un. Même sans tenir compte des objections formulées dans la première partie quant à la validité des modèles de simulation fonctionnelle et surtout des modèles de faute associés, cette voie nous était fermée.
Il est reconnu dans la littérature que des défauts ponctuels viennent perturber les processus de fabrication de circuits intégrés. Ce sont les effets de ces défauts ponctuels qui sont généralement simulés à des niveaux logiques ou électrique. Par le biais des moyens laser mis à notre

---

[10]Le cas d'une faute de conception dans un 68000 de seconde source, créant une erreur d'addition longue, visible à 125 °C , et qui est restée cachée plusieurs années est bien connu des spécialistes.

disposition, ce sont les effets causés par certains défauts ponctuels que nous avons voulu reproduire.

II.1. La méthode d'injection utilisée.

Parmi l'ensemble de défauts possibles à créer, nous avons choisi de nous limiter au deux les plus simples à mettre en oeuvre : les coupures et les courts-circuits. Si cette notion de simplicité est réelle pour les coupures, on a pu constater qu'il n'en est pas de même pour les courts-circuits ; dans ce cas, le processus de création réclame un appareillage assez lourd et délicat à mettre en oeuvre. Nous sommes de plus restreints à un travail sur un seul niveau du circuit (le niveau métallisation) bien qu'il soit possible de travailler sur des niveaux plus profonds ou entre deux niveaux. Pour cela la panoplie d'outils doit s'enrichir d'un générateur de faisceau d'ions afin de creuser pour atteindre les régions enterrées.
Si on analyse les résultats obtenus lors de la troisième expérience sur le 68000, en fin de première passe de test on voit que sur les 23 circuits détectés bons par tous les tests, 15 circuits étaient porteurs soit d'un défaut mal créé, soit d'un défaut inefficace de par sa création (résistance équivalente trop faible par exemple). Ceci représente un taux d'échec de près de 8%. Parmi ces échecs, ce sont surtout les coupures que nous avons réalisées qui sont en cause, alors que pour les courts-circuits, qui ont été eux réalisés par un spécialiste, seul un sur la centaine créée s'est avéré défectueux. Ceci pour mettre en évidence que non seulement ces expériences ont nécessité l'emploi d'un matériel sophistiqué, mais également que pour obtenir un taux de réussite maximum, il est nécessaire de bénéficier du concours d'un personnel spécialisé.
Une conclusion similaire quant à la difficulté de créer des défauts efficaces à l'aide d'un laser, notamment au niveau des coupures de Silicium Polycristallin, a été énoncée en [Nor91]. Ces travaux sont, à notre connaissance, les seuls publiés décrivant l'utilisation d'une méthode d'injection laser de défauts.

II.2. Les défauts injectés

On va ici pouvoir critiquer le fait que seul deux types de défauts ont été injectés. Les seuls arguments que l'on pourrait opposer à cela sont ceux de temps, de coût et de complexité de création des autres défauts. Il est évident que d'autres défauts seraient intéressants à étudier, mais leurs nécessité dans le type d'expérience que nous avons mené est difficile à chiffrer. Le coût du traitement de plusieurs centaines de microprocesseurs afin d'obtenir un échantillon de taille acceptable est lui facilement chiffrable. Nous nous sommes donc limités a des fautes physiques (coupures et courts-circuits) qui représentent suivant les différents auteurs entre 80 % et 90 % des fautes réelles [Gal80][Fer88][Jaco89].
Il est un type de défaut que nous voulions étudier dans une seconde phase, c'était les défauts que l'on pourrait qualifier "d'incomplets", comme des pistes non entièrement coupées ou des courts-circuits pas totalement jointifs par exemple. Il nous semblait intéressant d'étudier le comportement du circuit en présence de tels défauts espérant voir apparaître des phénomènes dus à des fautes transitoires ou aux fautes fortement liées aux paramètres de fonctionnement du circuit. Sans le vouloir, nous avons créés quelques défauts incomplets dès cette phase d'expérimentation (1 court-circuit incomplet et 10 coupures incomplètes notamment dans la troisième expérience sur le 68000). Dans tous les cas, ces défauts n'ont eu aucun effet, même lors du test paramétrique réalisé par le fabricant. Sans vouloir conclure sur l'inefficacité de tels défauts, il est clair que c'est un domaine d'étude très délicat. Si on arrive à créer des fautes de ce type, elles seront sûrement difficiles à analyser. Si ce type d'expérience doit avoir lieu, il doit être réalisé par et pour des électroniciens et des physiciens des matériaux et non par des informaticiens.

II.3. Conclusion

Au vu des multiples expériences, on peut conclure que notre méthode d'injection de défauts à l'aide d'équipements laser est intéressante, mais peut-être un peu coûteuse tant sur la plan matériel que sur la plan investissement humain. Seul l'ensemble des fautes injectable peut être critiqué ; encore que d'après les spécialistes la plupart des défauts décrits dans la littérature peuvent être reproduits ou simulés par l'utilisation des différents moyens laser.
Cependant, un telle méthode doit être réservée à ceux qui ont à leur disposition, non seulement toutes les facilités pour obtenir les pièces, mais aussi les renseignements nécessaires à l'analyse des résultats. En effet, on ne peut pas se contenter de résultats bruts de rejet ou d'acceptation d'une pièce par un test. Nous avons vu dans la troisième expérience que 6% des défauts sont restés inefficaces (alimentation rebouclée, lignes non utilisées court-circuitées, coupure en bout de ligne, …), ce qui implique que l'on doit absolument analyser les circuits passant le test avant d'annoncer quelque chiffre que ce soit. En un mot ce sont des expériences pour les fabricants de circuits et non pour les utilisateurs.
Son gros avantage, réside dans le fait qu'on peut se permettre de ne pas avoir d'a priori sur les fautes, ni de doutes sur les modèles employés. Sa principale qualité est son impartialité. En effet, on ne va pas établir d'hypothèses sur la relation entre les défauts créés et les fautes engendrées. Seuls des défauts pouvant arriver sont injectés.
Même dans le cas où l'on va contester les pourcentages obtenus, seuls les résultats quantitatifs seront mis en cause, pas les résultats qualitatifs qui vont déclarer incomplets tous les tests ne déclarant pas défectueux un circuit portant un défaut efficace.

III. - SUR L'AVENIR POUR LE TEST FONCTIONNEL

Une conclusion rapide de ce travail pourrait être : puisque le test fonctionnel ne fait pas mieux que le test fabriquant, et même ne peut pas mieux faire au moins sur le plan des signaux asynchrones, alors il ne sert à rien.
Cette conclusion pessimiste nous est bien entendu assez rapidement venue à l'esprit. Pourtant, depuis de nombreuses années, des programmes issus de notre méthode de test fonctionnel nous ont servi pour faire du diagnostic pour des fabricants.

Alors ici on doit se poser une question : que faire lorsque l'on détecte une pièce fautive ?
Si la réponse est : on la jette et on la remplace par une bonne, alors le test fonctionnel n'a vraiment aucune place. Mais si la réponse est : on va tenter d'expliquer pourquoi elle est mauvaise, alors là, tout peut changer.
Pour pouvoir faire une analyse de défaillance on à besoin d'outils performants mais également d'utilisation facile. Une faute peut être paramétrique ou booléenne, mais même dans le premier cas, dans les conditions d'erreur, elle peut affecter certaines fonctionnalités et pas d'autres. Dans tous les cas on va avoir besoin de localiser la cause de l'erreur. Ici le test fonctionnel retrouve sa place. Pour mettre en évidence cette affirmation, nous allons décrire des expériences de diagnostic réalisées parallèlement aux expériences d'injection de défaut.

III.1. Description d'expériences de diagnostic

Les deux expériences décrites ici concernent des microprocesseurs 68000. Dans le premier cas, ce sera un diagnostic pour un microprocesseur défaillant dans une application, dans l'autre cas, ce sera une analyse de défaillance pour un rejet de fabrication. Dans les deux cas des symptômes préliminaires permettent dès le départ d'orienter le diagnostic.

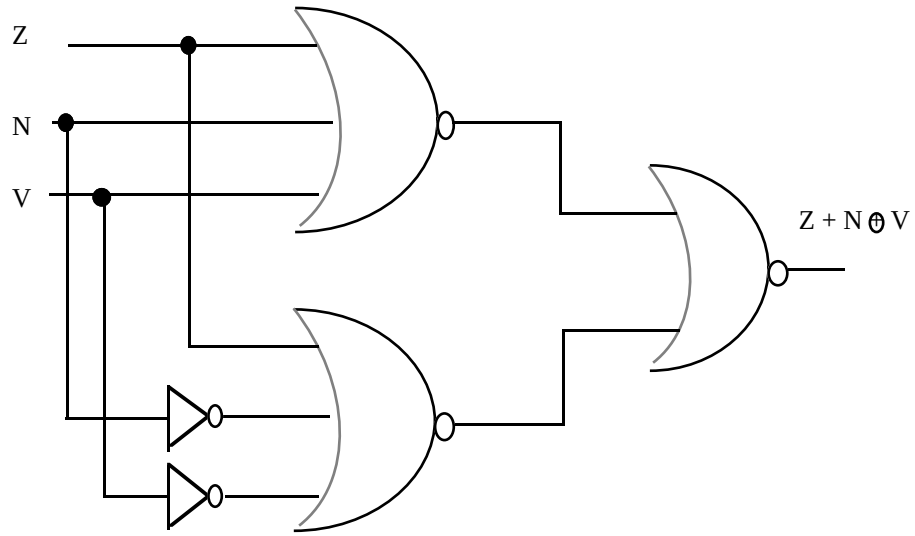
III.1.1. Diagnostic fonctionnel pour un 68000 défaillant dans une application

Un microprocesseur présentant une défaillance au coeur d'une application nous a été confié aux fins de diagnostic. Le programme source mettant en évidence le mauvais fonctionnement nous a été également fourni.
Nous avons fait subir à ce circuit le programme de test systématique complet. Celui-ci à décelé un problème lors du traitement de la condition "LE" (less or equal) mais uniquement pour certaines valeur du code condition.
Dans une deuxième phase un nouveau programme de test GAPT a été généré en mode balayage afin d'étudier le comportement du microprocesseur en présence de toutes les instructions conditionnelles (Bcc, DBcc, Scc) et ce pour toutes les conditions et toutes les valeurs possibles du registre de code condition.
Cette phase à fait ressortir une erreur pour toutes les instructions conditionnelles utilisant la condition LE, et seulement pour deux configuration du registre code condition :

N	Z	V	C
0	1	0	0

N	Z	V	C
0	1	0	1

Au vu de ces résultats, une tentative de diagnostic au niveau logique a été proposée. En supposant que l'évaluation de la condition "LE" se fasse par une logique simple, le diagramme logique en porte NOR donné en figure 3.1 a été proposé.



Valeurs des entrées manifestant l'erreur :

Z = 1 , N = V = 0

Figure 3.1 : diagramme logique de l'évaluation de la condition LE en porte NOR.

Les valeurs manifestant l'erreur correspondent à la propagation d'un collage à zéro de la ligne Z (figure 3.2)

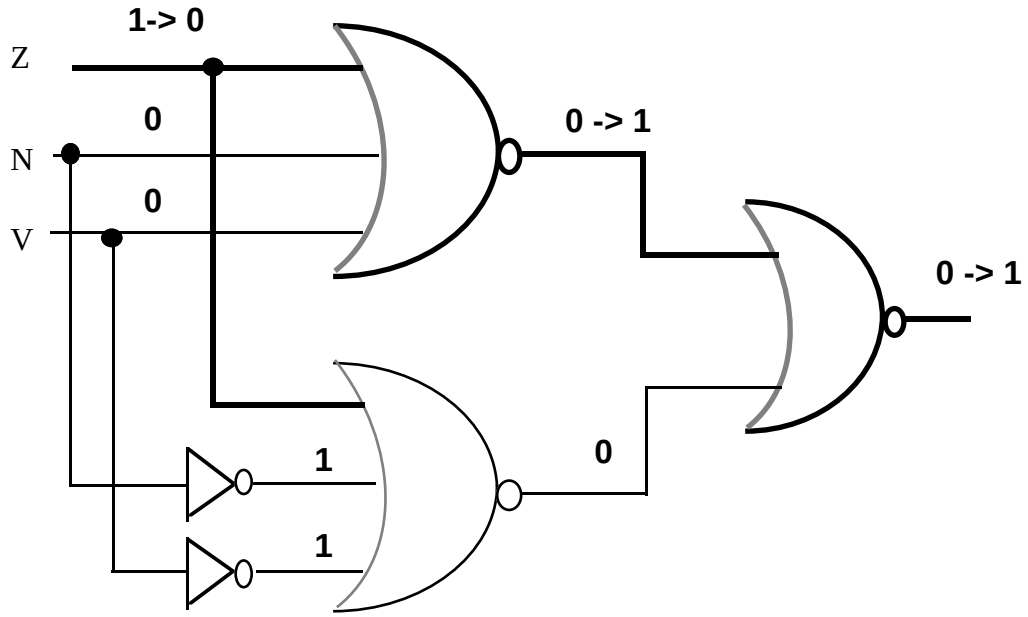


Figure 3.2 : identification de l'erreur par le collage à zéro d'une connexion

Sur le circuit, l'évaluation des conditions est en fait réalisée par un PLA. L'identification de la faute physique au sein de ce PLA nécessite l'analyse d'au plus quelques transistors (figure 3.3)

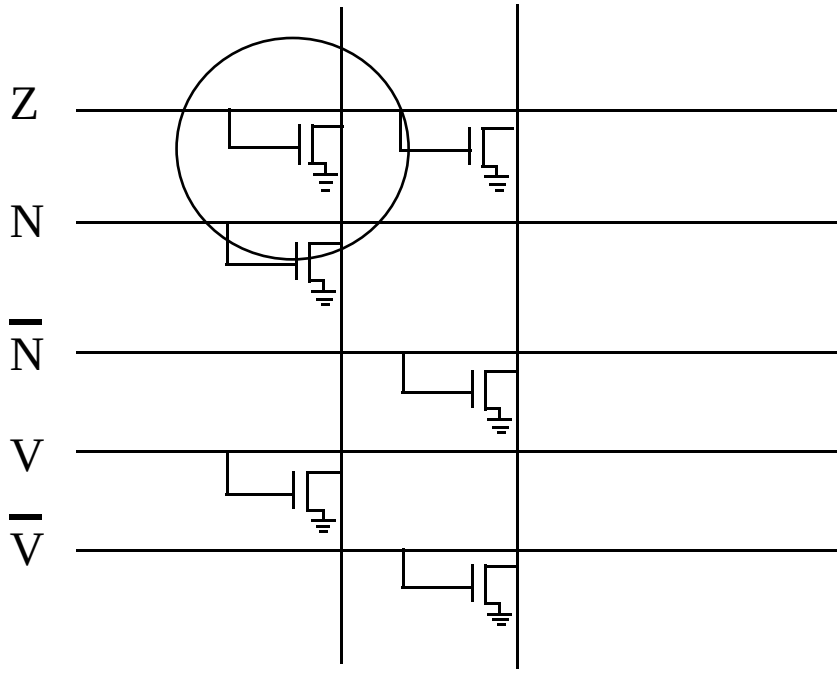


Figure 3.3 : diagramme électrique de la région suspecte du PLA de décodage.

III.1.2. Analyse de défaillance d'un rejet de fabrication.

Cette expérience rapportée dans [Vel90], est représentative de ce que permet le test fonctionnel pour les phases de diagnostic.

Après test à l'aide du programme systématique généré par GAPT, des erreurs lors de l'exécution des instructions de multiplication ont été détectées. Le tableau rapporté figure 3.4 donne certaines des valeurs erronées relevées.

Opérande 1	Opérande 2	Résultat attendu	Résultat obtenu
A379	4FE1	3302 0459	A23C 4594
CD68	9EDB	7F75 E7F8	B2B6 7F89
97B1	3114	1D14 BAD4	0A38 AD43
82B3	486F	24FB 089D	44E4 89D4

Figure 3.4 : échantillon des erreurs détectées.

L'analyse des valeurs de ce tableau fournie deux indices :

- les 4 bits de poids faible du résultat fautif sont égaux aux 4 bits de poids fort du 2ème opérande ;
- le motif constitué des 12 bits de poids faible du résultat attendu se retrouve décalé de 4 bits dans le résultat obtenu.

Exploiter ces indices nécessite la connaissance de la nature du multiplieur. Dans le cas du 68000, les multiplications sont réalisées suivant un algorithme d'additions et décalages à l'aide de l'additionneur de l'UAL et des registres généraux (pour le stockage des opérandes et des résultats partiels). L'expérience cumulée au cours de diverses séances de diagnostic à mis en évidence l'utilité d'opérandes de test ayant des propriétés connues, par rapport à la fonction testée (identité, neutre, absorption,…). Dans le cas étudié, la multiplication par 1 à fourni une première hypothèse sur la nature de l'erreur. En effet, le résultat obtenu a été égal au premier opérande décalé à gauche de 4 bits et complété à droite par des zéros (figure 3.5).

Opérande 1	Opérande 2	Résultat attendu	Résultat obtenu
A379	0001	0000 A379	000A 3790
CD68	0001	0000 CD68	000C D680
97B1	0001	0000 97B1	0009 7B10
82B3	0001	0000 82B3	0008 2B30

Figure 3.5 : résultats obtenus dans la cas de multiplication par 1.

Ces résultats consistants avec ceux de présentés figure 3.4 ont permis d'établir le diagnostic fonctionnel suivant : "l'instruction de multiplication s'arrête 4 pas avant la fin".
L'étape suivante a été l'identification de la zone suspecte du circuit. Ceci a nécessité des informations précises sur l'implantation du circuit et sur le séquencement du multiplieur
A partir de ce diagnostic et d'informations topologiques obtenues par "reverse engineering" du 68000, la faute physique à l'origine de l'erreur de multiplication a été identifiée à l'aide d'un microscope électronique à balayage. Il s'agissait d'un défaut dans un contact se traduisant par une initialisation incorrecte d'un registre interne utilisé comme compteur lors de la multiplication (initialisé à la valeur erroné 11, au lieu de 15). Ces résultats sont illustrés par les figures 3.6 à 3.8.

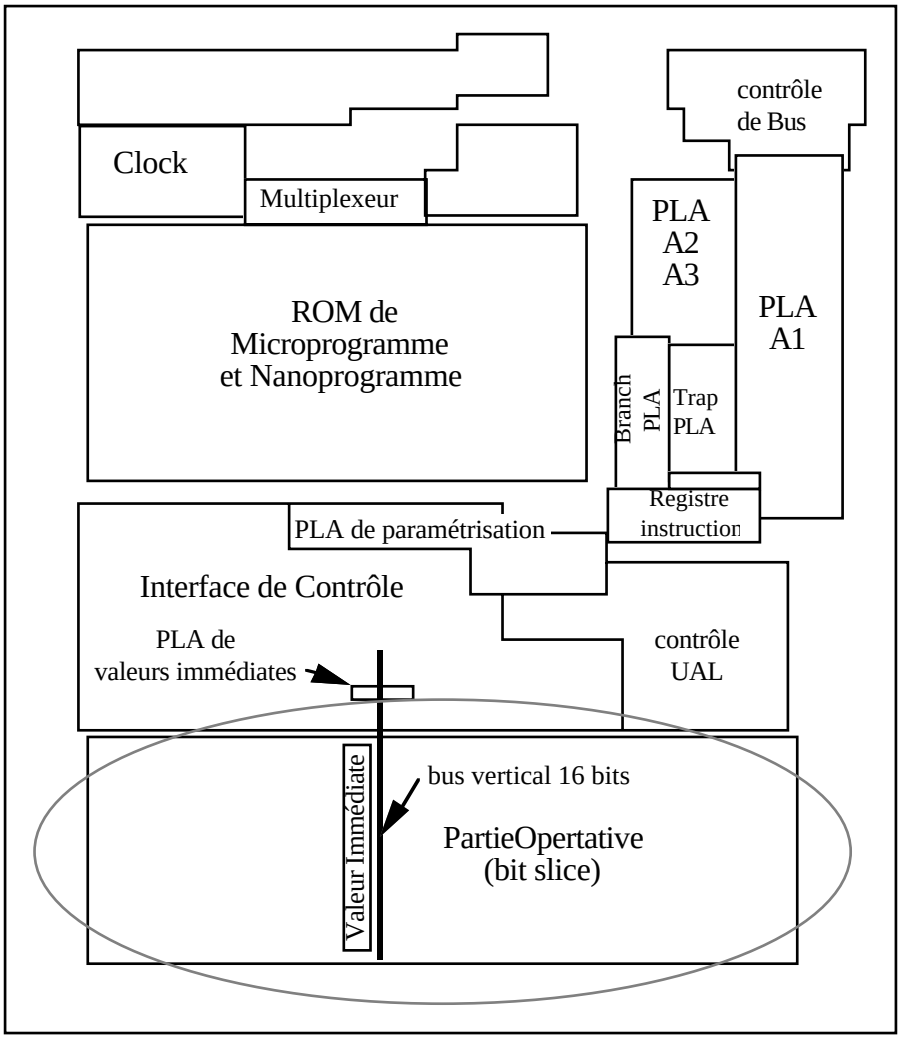


Figure 3.6: plan de masse du 68000

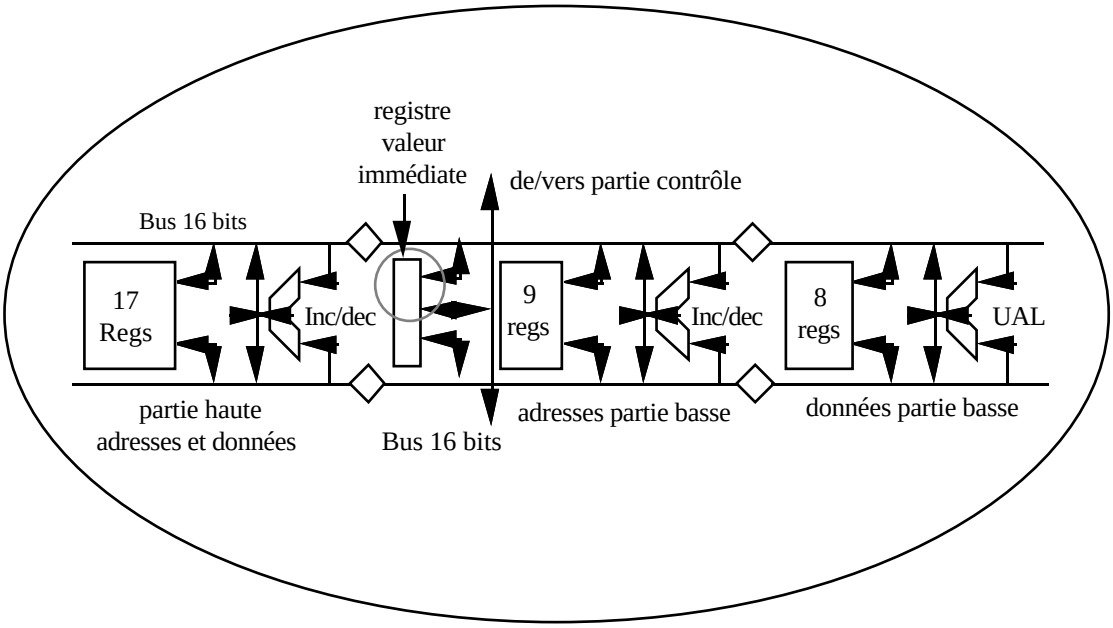


Figure 3.7 : schéma de la partie opérative

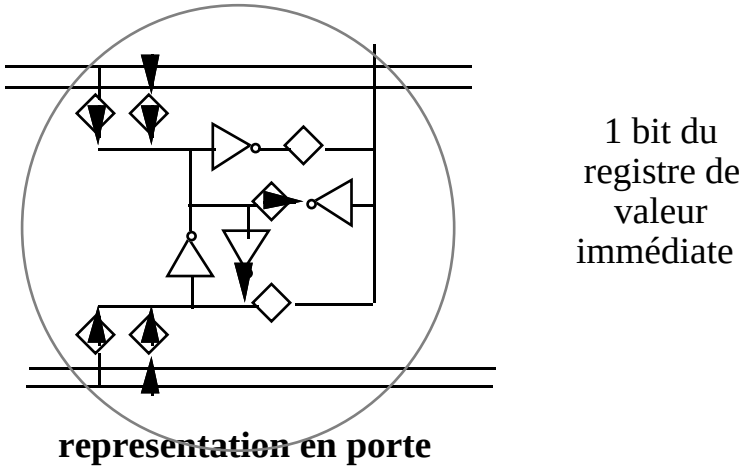


Figure 3.8 : représentation d'un bit du registre suspect.

III.1.3. Discussion

L'approche que nous utilisons pour faire du diagnostic fonctionnel à l'aide de GAPT peut se résumer par le diagramme donné figure 3.8.
A partir d'un fonctionnement erroné initial (soit identifié par un utilisateur, soit obtenu après exécution d'un programme de test systématique généré par GAPT) une séquence d'actions de diagnostic est réalisée. Chacune de ces actions est composée des étapes suivantes :

- analyse du symptôme ;
- élaboration d'une hypothèse d'erreur ;
- génération d'une séquence de test adaptée ;
- exécution par le circuit sous test ;
- analyse des résultats.

Une fois qu'un diagnostic fonctionnel plausible est obtenu (au niveau des instructions ou des blocs), une tentative de localisation de la faute à des niveaux plus fins (logique ou électrique) est

réalisée. Cette étape nécessite bien sur des informations structurelles sur les blocs suspects et doit donc être effectuée en collaboration avec un concepteur.

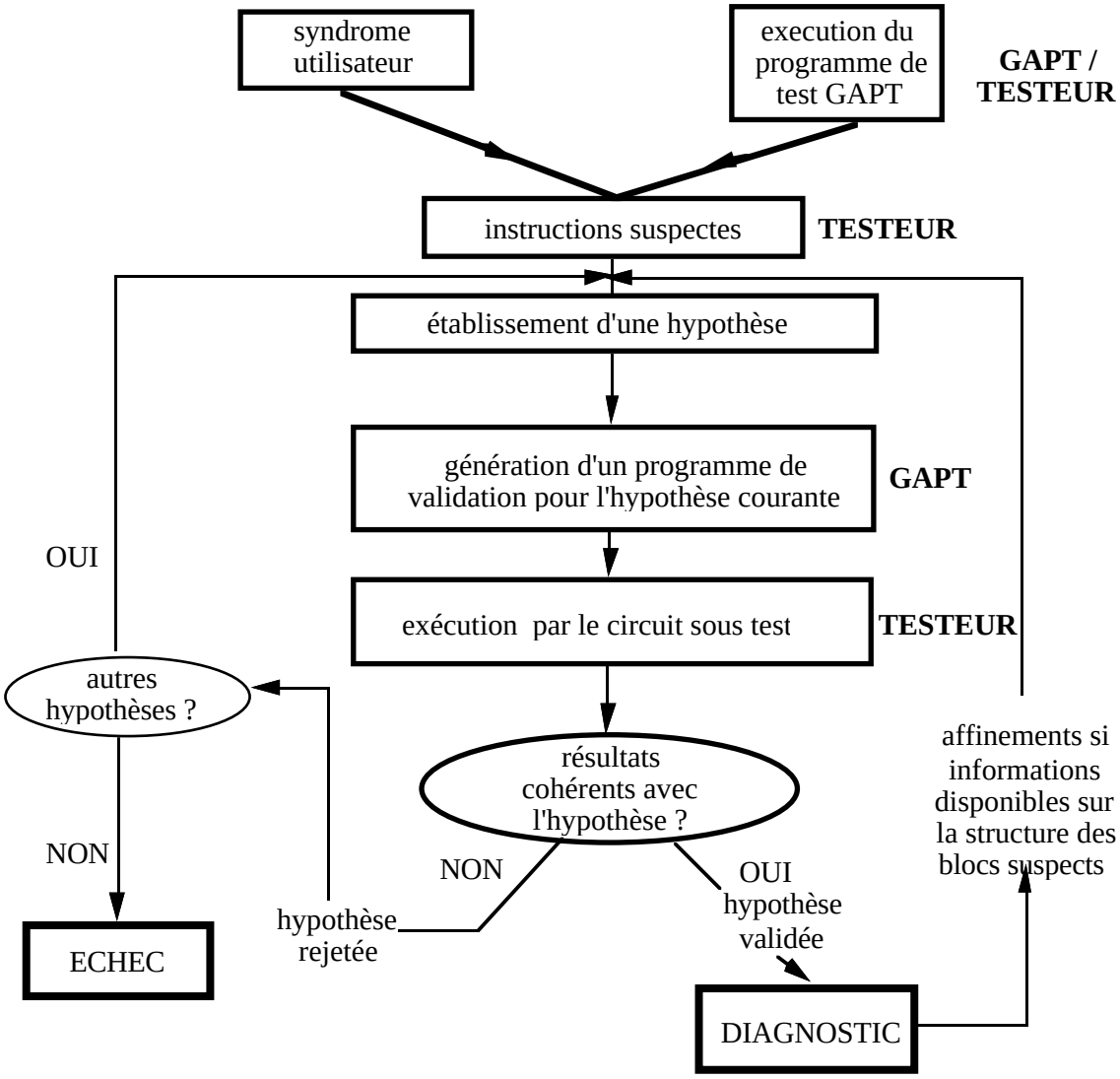


Figure 3.8 : principe de l'approche de diagnostic employée

III.2. Conclusion sur la place du test fonctionnel

Ces expériences sont à rapprocher d'autres expériences de diagnostic de circuits fautifs réalisées avec divers partenaires. Je ne peux pas malheureusement en rapporter beaucoup, toujours pour cause de confidentialité, mais ces travaux existent. Ce sont ces expériences qui nous ont confirmé dans notre opinion : le test fonctionnel en général, n'est pas le meilleur pour détecter des défauts, mais tel qu'il est utilisé dans GAPT, il est sûrement parmi les outils les plus simples et conviviaux à utiliser pour diagnostiquer. C'est là qu'est peut être la vrai place d'un tel test. Non pas chez un utilisateur mais bien chez un fabricant, et même plus précisément au sein des équipes de conception des fabricants, pour tous les besoins de diagnostic durant la phase de conception d'un circuit ou dans sa phase d'évolution vers une technologie nouvelle plus rapide.

IV. - LE TEST FONCTIONNEL "IDEAL"

Au vu des résultats d'expériences présentées dans cette thèse, on peut dire que la limitation principale du test fonctionnel en général, et en particulier du test systématique généré par GAPT en ce qui concerne l'efficacité est l'incapacité à tester correctement les signaux asynchrones. Les générations actuelles de microprocesseurs possèdent des dispositifs permettant d'accélérer les traitements (mémoires caches, pipes, …). Ces types d'architectures vont faire ressortir une autre limitation inhérente au test fonctionnel qui n'était pas aussi critique pour les génération précédentes de processeurs : le test des séquences d'instructions. Nous rappelons ici que vu sa structure en modules élémentaires organisés autour d'une instruction à tester, GAPT ne peut pas apporter de solution à ce problème.
Hormis les tests purement aléatoires, une esquisse de solution a été apporté en [Gad86]. Dans cet article, il est conclu que de courtes séquences aléatoires d'instructions (2 à 5) seraient très efficaces pour mettre en évidence des fautes liées à la sensibilité aux séquences ("pattern sensivity"). Une séquence optimale permettant de tester toutes les paires d'instructions est proposée en [Fuj85] pour des procédures de "self test".

A partir des acquis du système GAPT, nous allons ébaucher ici les évolutions logicielles et matérielles à apporter à un tel outil pour pouvoir faire face aux deux problèmes évoqués ci-dessus.
La philosophie du testeur quant à l'acquisition des données par le circuit sous test sera conservée ; c'est à dire : le microprocesseur est actif lors du test et lit ses données en mémoire.

IV.1. Logiciel

Nous allons présenter ici le résultat de réflexions quant au traitement des problèmes soulevés par le test des séquences d'instructions et des signaux asynchrones. Ces solutions logicielles seront souvent liées à des solutions matérielles qui seront développées dans le paragraphe suivant.

IV.1.1. Les séquences d'instructions.

Pour traiter les problèmes du test des séquences d'instructions, l'outil idéal doit pouvoir enchaîner aléatoirement ou pas, un nombre *n* d'instructions valides.
Pour chacune des instructions, les opérandes peuvent être déterministes ou aléatoires. Le premier cas implique le contrôle total de tous les paramètres des *n* instructions (nom de registre, mode d'adressage, valeurs, …). Ceci conduit à prendre en compte la sémantique de la séquence, et donc la résolution de multiples cas de conflit que la génération aléatoire peut entraîner.

Exemple du 68000 :
Soit la séquence d'instructions donnée ci-après, qui charge à un moment donné un registre d'adresse utilisé comme pointeur vers la mémoire dans une autre instruction.

MOVEA.L	D0,**A1**	(D0 → A1)	(1)
ADD.B	#023h,D3	(D3 + 023h → D3)	
:		:	
MOVE.L	(**A1**),D2	(mem(A1) → D2)	
(2)			

Pour amener une donnée spécifique au registre D2 , le registre d'adresse A1 devra contenir l'adresse mémoire de la donnée à charger par l'instruction (2). Par conséquent, cette donnée devra avoir été mise auparavant dans le registre A1 par l'intermédiaire de l'instruction (1).

Même dans ce cas simple, on s'aperçoit de la complexité de l'analyse sémantique que doit réaliser le générateur d'un tel programme de test. Si pour le même exemple on rajoute une troisième instruction utilisant le registre A1 avec un autre mode d'adressage, des conflits difficilement solubles peuvent survenir comme le montre la séquence ci-dessous.

```
MOVEA.L     D0,A1                    ; (D0 → A1)
                    (1)
ADD.B       #023h,D3                 ; (D3 + 023h → D3)
   :                    :
MOVE.L      (A1),D2                       ; (mem(A1) → D2)
                    (2)
   :                    :
SUB.W       12(A1,D0.W),D6   ; (mem(A1 + D0.W + 12) → D6)
(3)
```

La valeur dans D0 imposée par les instructions (1) et (2) ne pourra pas correspondre à la valeur nécessaire pour résoudre le mode d'adressage de l'instruction (3)

Une première conclusion serait donc que tous les opérandes soient aléatoires. Pour rendre ceci valide, tout mot de la mémoire susceptible d'être adressé en lecture doit être détenteur d'une donnée. Pour les écritures mémoire, le problème est différent car le contenu de toute adresse mémoire serait également susceptible d'être modifié à tout instant. Bien entendu, cette condition est inacceptable pour que la propre séquence de test ne soit pas perturbée. Seule une solution au niveau du matériel de test pourrait être alors envisagée (double plan mémoire, observation par signatures…). Ce point sera développé dans le paragraphe sur le matériel.

Il existe un problème lié au mode de test utilisé par GAPT qu'il faudra essayer de résoudre :: le test des instructions de déséquencement. En effet, nous fonctionnons avec un microprocesseur actif, qui va donc lire ses propres programmes de test en mémoire ; ceci implique que les programmes doivent être cohérents notamment au niveau des instructions de saut de type "JUMP" ou "BRANCH".
Si on admet la solution du tout aléatoire (code opération, mode d'adressage, opérande) qui semble être la plus simple, on risque de se trouver en présence de sauts en dehors des zones programmes pouvant déclencher des cascades d'exceptions pour code invalide, ou encore d'introduire des boucles infinies par retour arrière dans le programme. Un programme GAPT est bâti séquentiellement, il n'accepte donc que des sauts vers l'avant d'une longueur limitée.

Le schéma suivant montre la solution actuellement adoptée dans GAPT :

```
            INIT          initialisation de l'état interne
            JUMP étiq     instruction de saut (adresse calculée relativement à une
étiquette)
            INSTR         instruction témoin du passage en séquence
étiq        OBSERV        observation de l'état interne.
```

En cas de passage en séquence l'instruction témoin servira de marqueur pour la séquence d'observation (chargement d'un registre avec une valeur particulière par exemple).
L'introduction d'une telle séquence au coeur d'instruction aléatoire briserait la cohérence de l'ensemble du test, de même que le retrait pur et simple des instructions de sauts pour le test des séquences d'instructions. En effet, dans le cas de test des mémoires cache, ce sont l'empilement de contexte et les basculements qui vont être intéressants à vérifier, dans les instructions de sauts à sous programmes par exemple. Pour les instructions de type JSR (saut à sous programme) ou d'exception, on peut envisager des solutions simples de branchement à une

adresse ou une zone mémoire contenant des instruction de retour (retour de sous programme, ou retour d'exception). Pour les branchements relatifs et les sauts simples le problème reste entier.

Afin de conserver une cohérence au test, il conviendrait d'encadrer les séquences d'instructions à tester par des séquences d'initialisation et d'observation ; ceci dans le but de conserver ce qui donne de la puissance et de la convivialité pour le diagnostic aux programmes GAPT : à chaque instruction , l'état interne courant du microprocesseur est connu. On peut, pour des séquences d'instructions, envisager des initialisations et des observations de tous les registres comme en mode conformité, ou choisir d'appliquer le mode pseudo-conformité, c'est à dire d'observer, puis de n'initialiser que les registres mis en cause par les instructions à tester.
Suivant la taille que l'on va donner au nouveau module élémentaire, il va être intéressant de se poser le problème de l'observation. Si la séquence est trop longue, des risques de masquages d'erreur peuvent intervenir du fait que la probabilité pour qu'un même registre serve plusieurs fois augmente.

Exemple 68000 :
soit la séquence d'instruction suivante :

ADD.L	D3,D0	(D0 + D3 → D0)·
:	:	
CLR	D0	(0 → D0)

Quelque soit le résultat juste ou faux de l'instruction d'addition (contenu dans D0), il sera écrasé par l'instruction CLR (clear).
Bien sur, si seule l'addition est fautive, la probabilité que la prochaine erreur soit également masquée est faible, mais si l'erreur est une conséquence d'un enchaînement d'instructions, alors c'est la probabilité que cette séquence réapparaisse qui devient très faible. De toutes façon, il y a un risque non nul de masquage, donc d'imperfection du test. On réduira sans doute de beaucoup ces risques en choisissant une longueur de séquence faible (4 à 5 instructions ?), mais on n'éliminera pas tous les risques. Seules des observations à chaque instruction (par signature par exemple) peuvent palier le problème. Si cette solution est choisie, avec une séquence de taille plus importante, un mécanisme automatique de dichotomie (logiciel ou matériel) devra intervenir pour cerner la ou les instructions mettant en évidence le mauvais fonctionnement

IV.1.2. Le test des signaux

Si nous considérons la structure actuelle, le test des signaux pose un problème. En effet, la configuration en module élémentaire fait qu'une instruction à tester est entourée d'un nombre plus ou moins important d'instructions d'initialisation et d'observation toutes du même type "LOAD" ou "STORE" (pour le 68000 en conformité une instruction est entourée par 34 instruction de type MOVE ou MOVEA). Si on fait arriver au hasard des signaux asynchrones pendant ces phases de test, la probabilité pour qu'ils arrivent toujours sur le même type d'instruction est trop importante.
Si on choisit de considérer la structure en séquence d'instructions le problème se réduit de lui même mais reste important. Une solution viable serait de masquer l'arrivée des signaux asynchrones pendant les séquences d'initialisation et d'observation, et de la valider pendant le traitement des instructions. Donc un générateur aléatoire de signaux pourrait injecter des signaux uniquement pendant les phases actives intéressantes. Le problème du masquage peut introduire une nouvelle difficulté au niveau du matériel cette fois, le problème de la reproductibilité de la séquence aléatoire de signaux pour cerner les causes d'erreur en phase de diagnostic.

IV.2. Matériel

Au niveau du matériel, les problèmes se situent sur plusieurs plans différents :

- l'écriture en mémoire par des instructions à adressage aléatoire ;
- le test des signaux ;
- l'observation des bus internes ;
- le temps déchargement des programmes en mémoire et le temps de comparaison des résultats.

L'écriture en mémoire à des adresses aléatoires peut être résolue par deux systèmes :

- l'utilisation de deux plans mémoires distincts, l'un comportant les programmes et les données, l'autre recevant les résultats. La sélection de l'un ou l'autre plan se faisant simplement grâce au signal de lecture/écriture (r/w). L'avantage d'une telle solution est de résoudre également partiellement le dernier point, a savoir les temps de chargement et de comparaison. En effet si l'on veut tester plusieurs microprocesseurs à la suite, le même programme source pourra être exécuté sur plusieurs microprocesseurs montés chacun à leur tour sur la carte d'interface avec le testeur. Seule la mémoire d'observation devra être nettoyée avant chaque exécution. Un schéma explicatif est donné en figure 3.9.

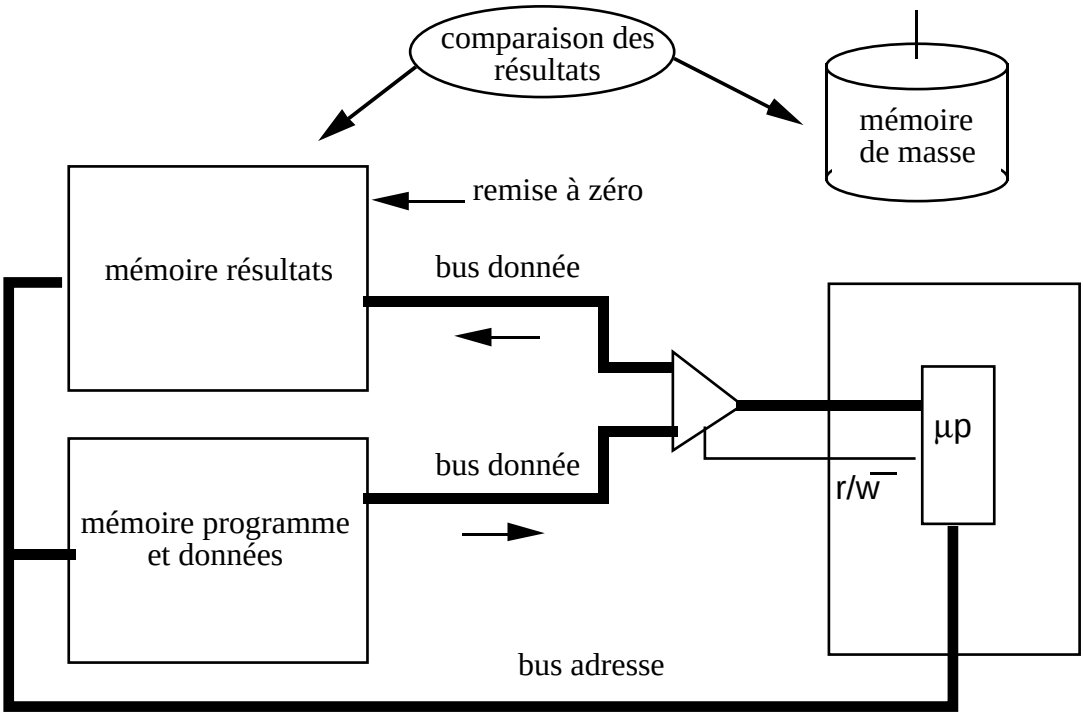


Figure 3.9 : utilisation de deux plans mémoire pour le test.

- l'utilisation de registres de signature sans écriture mémoire. Cette seconde méthode résout également les problème de temps de comparaison, puisqu'il suffit de comparer deux signatures et non plus deux plans mémoires. L'avantage certain de cette solution se situe également au niveau de l'observation des bus de contrôle. Toutes les sorties peuvent être observées, et non uniquement les données et adresses comme dans le cas d'écriture mémoire. Le problème est reporté au niveau de la phase de diagnostic, une valeur de signature étant parfaitement incompréhensible, seuls des mécanismes annexes peuvent permettre une analyse aisée des erreurs. Des mécanismes dichotomiques comme ceux implantés sur TEMAC doivent alors être mis en place.

Une bonne solution pour résoudre tous les problèmes évoqués ci-dessus est la concaténation des deux solutions précédentes. Un schéma de la solution qui pourrait être retenue est donné figure 3.10.

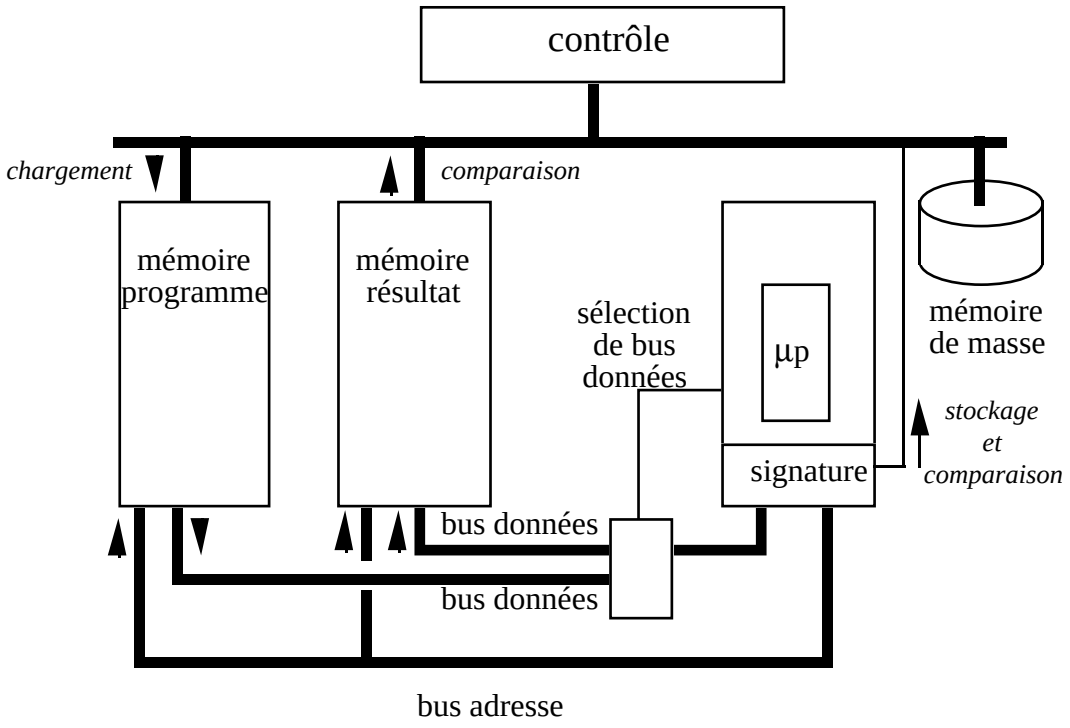


Figure 3.10 : solution mixte : registre de signature et double plan mémoire.

Le dernier problème à résoudre est celui du test des signaux. Pour cela, on peut bien entendu reprendre la solution utilisée dans TEMAC [Bel84b], mais cette méthode présente le désavantage de déclencher par programme l'émission des signaux externes, ce qui implique que le délai de réception du signal est toujours le même. On ne peut pas parler dans ce cas de

génération aléatoire de signaux. Toute solution essayant de s'affranchir de ce problème alourdit considérablement soit le matériel, soit le logiciel du moniteur pilotant le testeur. Un "cahier des charges" du problème à résoudre est donné figure 3.11.

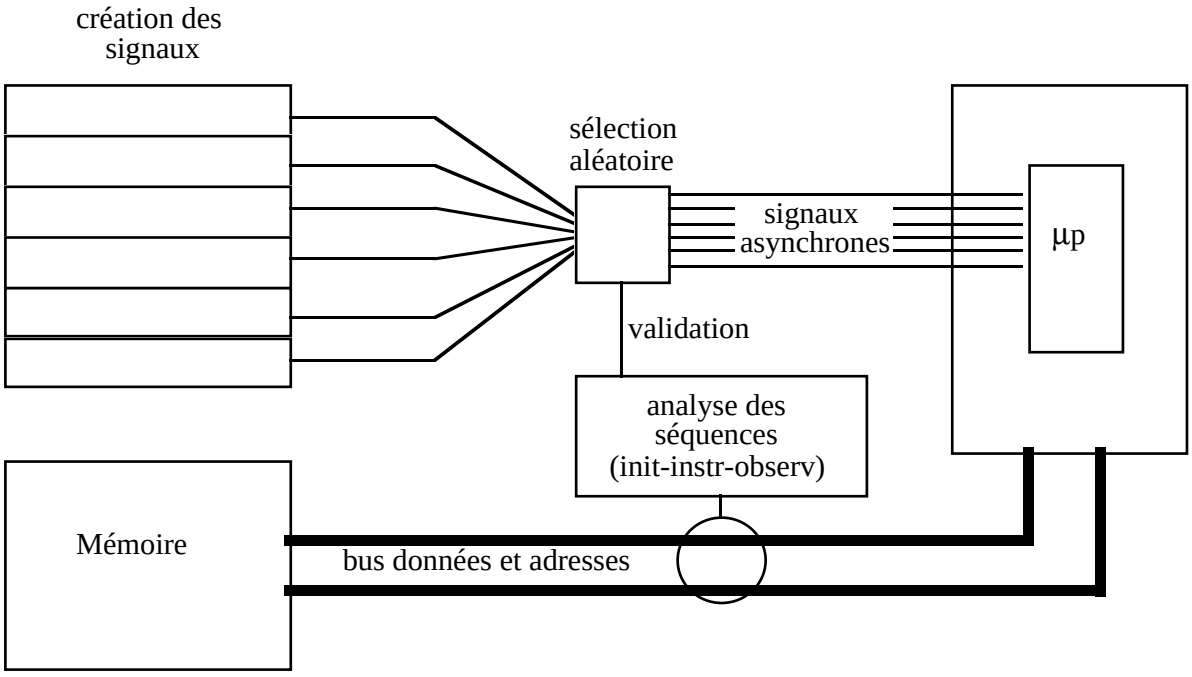


Figure 3.11 : problème à résoudre pour le test des signaux

Une batterie de LFSR génère de façon aléatoire les différents signaux. Une sélection à lieu sur le signal ou la combinaison de signal à appliquer. Pour pouvoir appliquer ce signal il faut que le processeur soit dans une phase d'exécution d'instruction ou de séquence d'instruction. Une analyse est donc nécessaire pour décoder les phases d'initialisation et d'observation. On peut ici facilement imaginer réaliser ceci par décodage d'une instruction spéciale (exemple lecture à une adresse précise facilement décodable comme FFFFFFFF) qui pourrait encadrer les instructions à tester.

En conclusion, l'outil matériel "idéal" devrait être une sorte d'hybride entre TEMAC et FUTE, afin de conserver la légèreté et la grande maniabilité du second, et retrouver la puissance de détection du premier. Des mécanismes supplémentaires très sophistiqués seraient nécessaires pour réaliser un bon test aléatoire de signaux.

Conclusion

Références bibliographiques

[Abr82] Abramovici M, Breuer M.A. : *'Fault Diagnosis in synchronous sequential circuits based on an effect-cause analysis'*, IEEE Transactions on Computers, 1982, Vol. C-31, N°12

[Ann82] Annaratone M.A., Sami M.G. : *'An approach to functional testing of microprocessors'* 12ème Fault Tolerant Computing Symposium , Santa Monica, USA, 1982, pp. 158-164.

[Arl88] Arlat J., Crouzet Y., Laprie J., Amat L. : *'L'injection de fautes pour la validation de la sûreté de fonctionnement'*, 6ème Colloque International de Fiabilité et de Maintenabilité, Strasbourg, France, 1988, pp. 30-35.

[Arl90] Arlat J. et al : *'Fault injection for dependability validation : a methodology and some application'*, IEEE Transactions on Software Engineering, Février 1990, Vol. 16, N°2, pp. 166-182.

[Auv86] Auvert G. : *'Microchirurgie de circuits intégrés'*, dans 'Le vide : les couches minces', ed. par Société Française du Vide, N° 233, Août 1986, pp. 135-138.

[Auv91] Auvert G. : *'C.W. laser induced chemical reaction with integrated circuits'*, Journal of vacuum Sc. Techn. B., Janvier 1991.

[Bel82] Bellon C. et al : *'Automatic generation of microprocessor test programs'*, 19ème Design Automation Conference, Las Vegas, USA, 1982, pp. 566-573.

[Bel84a] Bellon C. : *'Le test fonctionnel de circuits intégrés complexes'*, Thèse de Doctorat d'Etat, Institut National Polytechnique, Grenoble, France, 1984.

[Bel84b] Bellon C., Velazco R. : *'Taking into account asynchronous signal in functional test of complex circuits'*, 21ème Design Automation Conference, Albuquerque, USA, 1984, pp.490-496.

[Bel84c] Bellon C., Velazco R. : *'Hardware and software tools for microprocessor functional test'*, International Test Conference, Philadelphie, USA 1984, pp. 804-810

[Bel85] Bellon C., Kolokithas E., Velazco R. : *'Le système GAPT : une chaîne de test pour microprocesseurs'*, L'Onde Electrique, Novembre 1985, Vol. 65, N°6.

[Bra84] Brahme D., Abraham J.A. : *'Functional testing of microprocessors'* IEEE Transactions on Computers, Juin 1984, Vol C-33, N°6, pp. 475-485.

[Cas91] Caspi P., Piotrowski J., Velazco R. : *'An a priori approach to the evaluation of signature analysis efficiency'*, IEEE Transactions on Computers, 1991, Vol 40, N°9, pp. 1068-1071.

[Chi76] Chiang A.C.L., McCaskill R. : *'Two new approaches simplify microprocessor testing'*, Electronics, Janvier 1976, pp. 25-29

[Cor87] Cortner J.M. : *'Test strategy for the 1990s'* International Test Conference, Washington, USA, 1987, pp. 532-537

[Dav78] David R. : *'Feedback shift register testing'*, 8ème Fault Tolerant Computer Symposium, Toulouse, France, 1978, pp.103-107.

[Dav84] David R. : *'Signature analysis of multi-output circuits'*, 14ème Fault Tolerant Computer Symposium, Kissimee, USA, 1984.

[Fed84] Fedi X., David R : *'Experimentation results from random testing of microprocessors'* 14ème Fault Tolerant Computer Symposium, Kissimee, USA, 1984, pp. 225-230

[Fer88] Fergusson F.J., Shen J.P : *'Extraction and simulation of realistic CMOS faults using inductive fault analysis'*, International Test Conference, Philadelphie, USA, 1988, pp. 475-483.

[Fin87] Finelli G.B. : *'Characterization of fault recovery through fault injection on FTMP'*, IEEE Transactions on Reliability, Juin 1987, Vol R-36, pp. 164-170.

[Fra84] Franzel J.F., Marinos P.M. : *'Functional testing microprocessors in a user environment'*, 14ème Fault Tolerant Computer Symposium, Kissimee, USA, 1984, pp. 219-224

[Fuj85] Fuji R., Abraham J.A. : *'Self test for microprocessors'*, International Test Conference, Philadelphie, USA, 1985, pp. 356-361.

[Gad86] Gadenz R.N., Hays W.P. : *'A method for extensive verification of programmable VLSI devices'*, VLSI System Design, Juillet 1986, pp. 84-89.

[Gal80] Galiay J., Crouzet Y., Vergniault M. : *'Physical versus logical faults models in MOS LSI circuits - Impact on their testability'*, IEEE Transactions on Computers, 1980, Vol C-29, N°6, pp. 527-531

[Gun89] Gunneflo U., Karlsson J., Torin J. : *'Evaluation of error detection schemes using fault injection by using heavy ion radiation'*, 19ème Fault Tolerant Computer Symposium, Chicago, USA, 1989, pp.340-347.

[Kar91] Karlsson J., Gunneflo U., Lidén P, Torin J. : *'Two fault injection techniques for test of fault handling mechanisms'*, International Test Conference, Nashville, USA, 1991, pp. 140-149.

[Hay85] Hayes J.P. : *'Fault Modeling'*, IEEE Design and Test of Computer, Avril 1985, pp. 88-95

[Hao91] Hao H., McCluskey E.J : *'On the modeling and testing of Gate oxide shorts in CMOS logic gates'*, IEEE International Workshop on Defect and Fault Tolorance on VLSI Systems, Hidden Valley, USA, 1991, pp. 161-174.

[Hen86] Henshaw B. : *'An MC68020 users test program'*, International Test Conference, Washington, USA, 1986, pp. 386-393.

[Hna87] Hnatek E.R. : *'IC quality - Where are we ?'*, International Test Conference, Washington, USA, 1987, pp. 430-445.

[Hun84] Hunger A., Gartner A. : *'Functional charcaterization of microprocessors'*, International Test Conference, Philadelphie, USA, 1984, pp. 794-803.

[Jac89] Jacomino M., : *'Sur la théorie du test des circuits digitaux : mesure de la confiance'*, Thèse de Doctorat, Institut National Polytechnique, Grenoble, France, 1989.

[Jac90] Jacomino M., David R. : *'Influence of faults hypothesis on random test length'*, Rapport interne N°90/119, Décembre 1990, L.A.G., Institut National Polytechnique, Grenoble, France, 1989.

[Jaco89] Jacomet M. : *'Fantestic, toward a powerful fault analysis and test pattern generation for integrated circuits'*, International Test Conference, Washington, USA, 1989, pp. 631-641.

[Jai83] Jain S.K, Susskind A.K. : *'Test strategy for microprocessors'*, 20ème Design Automation Conference, Miami Beach, USA, 1983, pp. 703-708.

[Kar91] Karoui S., Velazco R., Chapuis T. : *'A new approach for S.E.U testing'*, ICM 91, Le Caire, Egypte, 1991, pp.

[Kar92] Karoui S., Martinet B., Velazco R. : *'A low cost functional test system, the FUTE16 tester'* ICM92, Monastir, Tunisie, Décembre 1992 (à paraître).

[Kil86] Kildiran G., Marinos P.M. : *'Functional testing of microprocessors-like architectures'*, International Test Conference, Washington, USA, 1983, pp. 913-920.

[Klu88] Klug H.P. : *'Microprocessor testing by instruction sequences derived from random patterns'*, International Test Conference, Washington, USA, 1988, pp. 73-80.

[Kor90] Koren I., Singh A.D. : *'Fault tolerance in VLSI circuits'*, Computer, Special issue on fault tolerant systems, Juillet 1990, Vol. 23, pp.73-83.

[Lai83] Lai K., Sieworiek D.P. : *'Functional testing of digital systems'*, International Test Conference, Philadelphie, USA, 1983, pp.207-213.

[Lap85] Laprie J.C. : *'Sûreté de fonctionnement des systèmes informatiques et tolérance aux fautes : concepts de base'*, Technique et Sciences Informatiques (TSI), 1985, Vol. 4, N°5, pp. 419-429.

[Lin80] Lin M.G., Witz M., Yuen A.K., Rose K. : *'Testing the 8086'*, International Test Conference, Philadelphie, USA, 1980, pp. 426-432.

[Lin82] Lin M.G., Rose K. : *'Applying test theory to VLSI testing'*, International Test Conference, Philadelphie, USA, 1982, pp. 580-586.

[Mal87] Maly W. : *'Realistic fault modeling for VLSI testing'*, 24ème Design Automation Conference, Miami Beach, USA, 1987, pp. 173-180.

[Mar88] Marshall M. : *'Techniques for user test of the 68882'* , International Test Conference, Washington, USA, 1988, pp. 942-947.

[Mar89] Martinet B., Bellon C., Velazco R. : Manuel d'utilisation GAPT, Version V2.1. Rapport de Recherche LGI-IMAG, 1989.

[Mar90] Martinet B., Bellon C., Velazco R. : '*Failure coverage of functional test methods : a comparative experimental evaluation*'', International Test Conference, Washington, USA, 1990, pp. 1012-1017.

[Mar91a] Martinet B., Velazco R. : *'Physical fault injection : a suitable method for the evaluation of functional test efficiency'* IEEE International Workshop on Defect and Fault Tolorance on VLSI Systems, Hidden Valley, USA, 1991, pp. 179-182.

[Mar91b] Martinet B., Velazco R. : *'New results on functional testing of complex microprocessors'* European Symposium on Reliability of Electron devices, Failure physics and analysis (ESREF), Bordeaux, France, 1991, pp. 523-528.

[Mar91c].Martinet B., Velazco R. : *'An experimental evaluation of functional test efficiency'*, ICM91, Le Caire, Egypte, 1991, pp. 164-168.

[Mar92] Martinet B., Velazco R., Auvert G. : *'Laser injection of spot defects on integrated circuits'*, 1er Asian Test Symposium, Hiroshima, Japon, Novembre 1992 (à paraître).

[Mic82] Miczo A. : *'Fault modeling for functional primitives'* , International Test Conference, Cherry Hill, USA, 1982, pp.43-49.

[Mil87] Milan M., Janin M. : *'Microprocessor board for testing irradiated 6800'*, Rapport Interne, Université de Dijon, France, Juin 1987.

[Min82] Min Y., Su S.H.Y. : *'testing functional faults in VLSI'*, 19th Design Automation Conference, Las Vegas, USA, 1982, pp. 384-392.

[Mor82] Morillon F. : *'Physical fault simulation'*, International Symposium EUROCON'82, Copenhague, Danemark, 1982, pp. 489-492.

[Nor91] Nordsieck A. : *'Two stage faults location'*, International Test Conference, Washington, USA, 1991, pp.916-925.

[Pro89] Provost-Grellier A. : *'Test aux ions lourds de VLSI programmables'*, Thèse de Doctorat en Informatique, Institut National Polytechnique, Grenoble, 1989.

[Rob80] Robach C., Saucier G. : *'Microprocesseur functional testing'*, International Test Conference, Philadelphie, USA, 1980, pp.433-443.

[Sal83] Saluja K.K., Shen L., Su S.Y.H. : *'A simplified algoritm for testing microprocessors'*, International Test Conference, Philadelphie, USA, 1983, pp. 668-675.

[Sch86] Schuette M.A., Shen J.P., Siewiorek D.P., Zhu Y.X. : *'Experimental evaluation of two concurrent error detection schemes'*, 16ème Fault Tolerant Computer Symposium, Vienne, Autriche, 1986, pp. 138-143.

[Sod86] Soden J.M., Hawkins C.F. : *'Test consideration for gate oxide shorts in CMOS ICs'*, IEEE Design & Test, Août 1986, pp. 56-64.

[Su82] Su S.Y.H., Hsieh Y. : *'Testing functional faults in digital systems described by register transfer languages'*, Journal of Digital Systems, 1982, Vol 6, N°2/3, pp.161-183.

[Su84] Su S.Y.H., Lin T. : *'Functional testing techniques for digital VLSI systems'*, 21ème Design Automation Conference, Albuquerque, USA, 1984, pp. 517-528.

[Su85] Su S.H.Y., Lin T. : *'VLSI functional test pattern generation - a design and implementation'*, International Test Conference, Philadelphie, USA, 1985, pp. 922-929.

[Tew91] Tewksbury S.K. : *'Physical boundaries of performance : the interconnection perspective'*, IEEE International Workshop on Defect and Fault Tolorance on VLSI Systems, Hidden Valley, USA, 1991, pp. 227-246.

[Tha78] Thatte S.M., Abraham J.A. : *'A methodology for functional level testing of general microprocessor architectures'*, 8ème Fault Tolerant Computer Symposium, Toulouse, France, 1978, pp. 90-95.

[Tha80] Thatte S.M., Abraham J.A. : *'Test generation for microprocessors'*, IEEE Transactions on Computers, Juin 1980, Vol C-29, N°6, pp. 429-441.

[The81] Thevenod-Fosse P., David R. : *'Random testing of the data processing section of a microprocessor'*, International Test Conference, Philadelphie, USA, 1980, pp.275-280.

[The83] Thevenod-Fosse P., David R. : *'Random testing of the control section of a microprocessor'*, 13ème Fault Tolerant Computer Symposium, Milan, Italie, 1983, pp.366-375.

[Tho87] Thomas R.W. : *'How to use military standards follow the evolution from SSI to VLSI'*, 3ème Colloque International : la Qualité des Composants Electroniques, Bordeaux, France, 1987, pp. 35-44.

[Tho89] Thomas R.W. : *'The US Department of defense procurement strategy and the semiconductor industry in the 1990s'*, 4ème Colloque International : la Qualité des Composants Electroniques, Bordeaux, France, 1989

[Vel82] Velazco R. : *'Test comportemental de microprocesseurs'*, Thèse de doctorat, Institut National Polytechnique, Grenoble, France, 1982.

[Vel85] Velazco R. : *'Fault localisation when testing complex circuits'*, IEE Proceedings, 1985, Vol. 132, N°5, pp. 241-245.

[Vel87a] Velazco R., Ziade H. : *'Towards and automated system for the verification and diagnosis of "intelligent" VLSI circuits'*, International Symposium on Electronic Devices, Circuits and Systems, Kharagpur, Inde, pp. 607-609.

[Vel87b] Velazco R., Provost-Grellier A. : *'Stratégie de validation de circuits intégrés intelligents destinés à des application spatiales'*, 3ème Colloque International : La Qualité des Composants Electroniques, Bordeaux, France, 1987, pp. 232-237.

[Vel88] Velazco R., Bellon C., Ziade H. : *'An analysis of experimental results on functional testing and diagnosis of complex circuits'*, International Test Conference, Washington, USA, 1988, pp. 64-72.

[Vel89] Velazco R. et al. : *'Comparison between californium and cyclotron S.E;U. tests'*, IEEE Transactions on Nuclear Science, 1989, Vol 36, N°6, pp. 2383-2387.

[Vel90] Velazco R., Conard D., Guyot A., Ziade H. : *'Top down IC failure analysis using an E-beam system coupled to a functional tester'*, Microelectronic Engineering 12, Publishers E.S., pp. 113-120.

[Wad78] Wadsack R.L. : *'Fault modeling and logic simulation of CMOS and MOS integrated circuits'*, Bell System Technical Journal, Juin 1978, Vol. 57, N°4.

[Wes89] Westover J.H. : *'Practical test strategies for users of 100 PPM ICs'*, International Test Conference, Washington, USA, 1989, pp. 295-303.

[Zia86] Ziade H. : *'Méthodes et outils pour le diagnostic fonctionnel de microprocesseurs'*, Thèse de Doctorat, Institut des Sciences Appliquées, Toulouse, France, 1986.

Annexes

Annexe A : résultats expérimentaux

Résultats pour chacun des 4 tests : fabricants (Fabr), systématique (Syst), minimal (Min) et programme de tri (Tri). Chaque pièce reconnue comme bonne est notée B ; Une notation différente suivant l'erreur détectée sera attribuée à chacune des pièces défectueuses, suivant la légende ci-dessous.

test fabricant

F : Fonctionnel (structurel)

CC : Court Circuit (déséquencement)

CO : Circuit Ouvert

NR : NanoROM

RP : Refus Paramétrique

tests comportementaux

M : Mauvais (plus de 50 erreurs)

TO : Time Out

nE : n erreurs

Plaquette A - Coupures - 24 pièces.

	Fabr.	Syst.	Min.	Tri
1	B	B	B	B
2	B	B	B	B
3	B	B	B	B
4	B	B	B	B
5	NR	M	TO	CO
6	NR	M	TO	M
7	NR	TO	TO	TO
8	NR	M	TO	TO
9	NR	TO	TO	TO
10	NR	TO	M	TO
11	NR	TO	M	TO
12	NR	TO	TO	TO
13	F	26E	3E	B
14	F	M	TO	M
15	F	TO	M	M
16	F	M	M	3E
17	F	M	M	M
18	F	M	46E	M
19	F	M	29E	B
20	F	TO	M	M
21	F	M	4	B
22	F	M	M	B
23	F	M	5	M
24	F	B	B	B

Pièces non détectées : Systématique : 5 Tri : 9

Fabricant : 4 Minimal : 5

Plaquette B - Coupures - 42 pièces.

42	RP	B	B	B

	Fabr.	Syst.	Min.	Tri
1	B	B	B	B
2	B	B	B	B
3	B	B	B	B
4	B	B	B	B
5	B	B	B	B
6	B	B	B	B
7	B	B	B	B
8	B	B	B	B
9	B	B	B	B
10	B	B	B	B
11	F	B	B	B
12	F	M	TO	TO
13	F	M	M	M
14	F	M	M	TO
15	F	M	TO	TO
16	F	M	M	B
17	F	TO	TO	TO
18	F	TO	M	M
19	F	TO	TO	M
20	F	M	M	TO
21	F	M	M	B
22	F	M	TO	M
23	NR	TO	TO	TO
24	NR	TO	TO	M
25	NR	TO	M	M
26	NR	M	M	M
27	NR	M	M	M
28	NR	TO	TO	TO
29	NR	TO	TO	TO
30	NR	TO	TO	TO
31	NR	TO	TO	TO
32	NR	B	B	B
33	NR	TO	TO	TO
34	NR	TO	TO	TO
35	NR	TO	TO	TO
36	NR	TO	TO	TO
37	NR	TO	TO	TO
38	NR	TO	TO	TO
39	NR	TO	TO	M
40	NR	M	M	M
41	CC	TO	TO	TO

Systématique : 12

Pièces non détectées : (+ 1 paramétrique) Tri : 15

(+ 1 paramétrique)

Fabricant : 10 Minimal : 12

(+ 1 paramétrique)

Plaquette C - Coupures -
42 pièces.

					41	NR	TO	TO	TO
	Fabr.	Syst.	Min.	Tri	42	RP	B	B	B
1	B	B	B	B					
2	B	B	B	B					
3	B	B	B	B					
4	B	B	B	B					
5	B	B	B	B					
6	B	B	B	B					
7	CC	M	M	TO					
8	CO	TO	TO	B					
9	CO	TO	TO	TO					
10	CO	TO	TO	TO					
11	F	M	M	M					
12	F	TO	TO	TO					
13	F	TO	TO	TO					
14	F	M	M	B					
15	F	TO	TO	TO					
16	F	TO	TO	TO					
17	F	TO	TO	TO					
18	F	M	M	M					
19	F	TO	TO	TO					
20	F	TO	TO	M					
21	F	M	TO	M					
22	F	TO	TO	TO					
23	F	TO	TO	TO					
24	F	M	TO	M					
25	F	M	M	TO					
26	F	M	M	M					
27	F	M	TO	M					
28	NR	TO	TO	TO					
29	NR	TO	TO	TO					
30	NR	TO	TO	TO					
31	NR	B	B	B					
32	NR	B	B	B					
33	NR	TO	TO	TO					
34	NR	TO	TO	M					
35	NR	TO	M	TO					
36	NR	M	TO	B					
37	NR	TO	TO	TO					
38	NR	TO	M	TO					
39	NR	TO	M	B					
40	NR	M	M	M					

Systématique : 8

Pièces non détectées : (+ 1 paramétrique) Tri : 12

(+ 1 paramétrique)

Fabricant : 6 Minimal : 8

(+ 1 paramétrique)

Plaquette D - Courts-circuits - 47 pièces.

	Fabr.	Syst.	Min.	Tri
1	B	B	B	B
2	CC	TO	TO	TO
3	NR	TO	TO	TO
4	NR	TO	TO	TO
5	NR	TO	TO	TO
6	NR	M	M	40E
7	NR	TO	TO	TO
8	NR	M	TO	TO
9	NR	M	M	M
10	NR	M	M	TO
11	NR	TO	TO	TO
12	NR	TO	B	B
13	NR	TO	M	TO
14	NR	TO	TO	TO
15	NR	TO	TO	B
16	NR	M	TO	TO
17	NR	TO	TO	TO
18	NR	M	M	M
19	NR	M	M	TO
20	NR	M	TO	25E
21	NR	TO	TO	TO
22	NR	TO	TO	TO
23	NR	TO	TO	TO
24	NR	TO	TO	TO
25	NR	M	M	M
26	NR	TO	TO	TO
27	NR	TO	TO	TO
28	NR	TO	TO	TO
29	NR	M	M	4E
30	F	TO	TO	M
31	F	M	5E	B
32	F	B	B	B
33	F	B	B	B
34	F	M	M	M
35	F	M	7E	B
36	F	M	12E	B
37	F	M	M	M
38	F	TO	TO	M
39	F	M	8E	B
40	F	TO	TO	TO
41	F	B	B	B
42	F	TO	TO	TO
43	F	M	TO	M
44	F	M	M	B
45	F	5E	B	B
46	F	M	M	M
47	F	B	B	B

Systématique : 5

Pièces non détectées : (+ 1 paramétrique) Tri : 13

(+ 1 paramétrique)

Fabricant : 1 Minimal : 7

(+ 1 paramétrique)

Plaquette E - Courts-circuits - 40 pièces.

	Fabr.	Syst.	Min.	Tri
1	B	B	B	B
2	B	B	B	B
3	CC	TO	TO	4E
4	CC	TO	M	M
5	CC	TO	TO	M
6	CO	TO	TO	TO
7	CO	TO	TO	TO
8	CO	M	TO	M
9	CO	TO	TO	TO
10	CO	TO	TO	TO
11	F	TO	TO	M
12	F	TO	TO	TO
13	F	M	M	4E
14	F	M	M	M
15	F	TO	TO	TO
16	F	M	M	M
17	F	M	M	M
18	F	M	M	M
19	F	TO	TO	9E
20	F	TO	TO	9E
21	F	M	M	B
22	F	TO	TO	TO
23	F	M	M	7E
24	F	TO	TO	TO
25	F	M	TO	TO
26	F	B	B	B
27	NR	TO	TO	TO
28	NR	B	B	B
29	NR	TO	TO	M
30	NR	M	M	M
31	NR	M	11E	M
32	NR	TO	TO	TO
33	NR	TO	TO	TO
34	NR	M	M	B
35	NR	TO	TO	M
36	NR	TO	TO	TO
37	NR	TO	M	TO
38	NR	TO	TO	TO
39	NR	M	M	TO
40	NR	TO	TO	M

Pièces non détectées :

Fabricant : 2

Systématique : 4

Minimal : 4

Tri : 6

Annexe B : GAPT

Pour tester un microprocesseur, la méthode GAPT distingue :

- le test de la partie contrôle, qui active les fonctions du circuit (instructions : **test de conformité**, ou pseudo-conformité),
- le test de la partie opérative, qui vérifie les modules de la partie opérative (registres, opérateurs, connexion...), ou **test de balayage**.

Le test de conformité vérifie la bonne exécution d'un ensemble de fonctionnements représentatifs du circuit. On s'intéresse aux fonctions exécutables par le microprocesseur entre la lecture de deux codes opération successifs, ou **fonctionnements élémentaires**.
En d'autres termes, le problème des séquences d'instructions n'est pas pris en compte : la description comportementale d'un microprocesseur ne permet pas de déterminer quelles sont les séquences critiques, et il est impossible de tester tous les couples d'instructions, par exemple.

Le logiciel GAPT génère automatiquement le programme de test d'un microprocesseur (en langage assembleur) à partir de sa description "haut niveau" en langage GAPT et éventuellement de la base de données des opérandes de test.

Les programmes de test générés par GAPT, après assemblage, sont des programmes exécutables, et le test peut être effectué :

- sur un système de développement : le processeur exécute normalement le programme, et les résultats du test sont observés en mémoire centrale,
- sur le testeur FUTE16/32, qui est un testeur fonctionnel défini dans le cadre du projet GAPT, pour les processeurs d'usage général 8, 16 ou 32 bits : le processeur exécute normalement le programme, mais toutes ses sorties sont observées,

- sur un testeur de composants classique : cela nécessite une interface assez sophistiquée, surtout si on veut utiliser les capacités algorithmiques propres au testeur.

I. Spécification du logiciel GAPT

A partir de la description du microprocesseur dans le langage GAPT, le système génère les programmes de test du microprocesseur, à l'aide de la base de données des opérandes de test.

Le langage GAPT est basé sur la notion de groupes, permettant la description d'un grand nombre de fonctionnements représentatifs à l'aide d'une seule primitive du langage. Des groupes de valeurs, de registres, de modes d'adressage, et de formats peuvent être définis et utilisés pour décrire de manière compacte de nombreuses instructions effectives.

A partir d'une telle description, le générateur GAPT :

- détermine l'ensemble des instructions effectives,
- produit les séquences d'initialisation et d'observation correspondantes d'après le mode de test (conformité, pseudo-conformité ou balayage),
- génère les opérandes en mémoire.

1. Modes de test

Les programmes de test comportemental d'un microprocesseur sont des programmes longs, mais très répétitifs, composés de modules élémentaires indépendants les uns des autres.

a) Modules élémentaires

Pour chaque fonctionnement à tester sera créé un module élémentaire de test, de la forme suivante :

- initialisation de l'état interne du microprocesseur,
- activation du fonctionnement à tester,
- observation de l'état interne.

Pour un microprocesseur, l'initialisation est une séquence d'instructions du type "LOAD", permettant de charger les registres internes du microprocesseur. L'observation est une séquence d'instructions du type "STORE" en mémoire, permettant au testeur d'observer les registres internes ; les instructions choisies seront les plus simples possibles.
Suivant le mode de test choisi : (conformité, pseudo-conformité, balayage), les modules élémentaires auront une forme et une fonctionnalité différentes.

b) Le test de conformité

Il consiste à vérifier que :

- les blocs fonctionnels qui doivent être activés par la transition sont activés effectivement et correctement,
- les blocs fonctionnels qui ne doivent pas être activés ne le sont pas.

Un module élémentaire de conformité a la forme suivante :

- initialisation de tous les registres,

- instruction à tester,

- observation de tous les registres.

On initialise et observe donc **tous** les registres internes du microprocesseur, de telle sorte que :

- les registres sources contiennent les valeurs de test,

- les registres “destination” et les registres non utilisés aient une valeur différente des registres source (valeur aléatoire). Ce mode permet de vérifier qu’un registre qui doit être non utilisé n’est pas pris à tort comme source ou puits de l’opération,

Ce mode de test qui mène à des programmes de test très longs, sert dans la phase de diagnostic à localiser rapidement une erreur. L’état interne du microprocesseur étant réinitialisé à chaque module élémentaire, un seul bloc du programme est à mettre en cause afin de détecter l’origine de l’erreur.

c) Le test de pseudo-conformité

L'utilisation de ce mode de test permet de réduire la longueur d'un test d'identification, sans diminuer les capacités de détection d'erreurs.
Un module élémentaire en mode pseudo-conformité est de la forme :

- observation de tous les registres internes qui doivent être modifiés (sources et destinations de l'instruction à tester),
- initialisation des sources de l'instruction,
- instruction à tester.

La concaténation de telles séquences, précédée d'une initialisation globale de tous les registres et suivie par une observation globale, constitue un programme de test aussi puissant mais plus court qu'en mode conformité. L'inconvénient est que les données ne sont pas locales à un module élémentaire, donc quand une erreur est détectée une large portion du programme de test doit être analysée pour identifier les instructions suspectes. Une autre possibilité serait de générer à nouveau le même test en mode conformité.

d) Le test de balayage

Le module de test a la forme suivante :

- initialisation des sources de l'instruction,
- instruction à tester,
- observation des puits de l'instruction.

Ce type de test vérifie que les résultats sont corrects, mais non que d'autres registres ne sont pas modifiés. Ce test est en général utilisé pour vérifier les blocs de la partie opérative (UAL, registres...).

2. Opérandes de test

Les opérandes de test peuvent être :

- déterministes,
- aléatoires.

L'utilisateur peut définir ses opérandes de test ; ils peuvent être des ensembles de valeurs ou des ensembles de chaînes de caractères. Un ensemble d'opérandes de test est appelé groupes de valeurs.
Considérons une instruction à n sources ; à chaque source sera associé un groupe de valeurs. GAPT combine les opérandes de test dans les différents groupes ; il y a donc ainsi itération sur les opérandes de test. On peut avoir une itération complète (listes de valeurs séparées par des virgules) ou itération partielle (listes de valeurs séparées par des points virgules).
Exemple :

pour une instruction à 2 sources : itération complète

GV1 = (V1,V2) GV2 = (V3,V4)

tests générés V1,V2

V1,V4

V2,V3

V2,V4

itération réduite

GV1 = (V1;V2) GV2 = (V3;V4)

tests générés V1,V3

V2,V4

L’utilisateur peut également utiliser des opérandes aléatoires, ou associer des opérandes déterministes et des opérandes aléatoires.
L’ensemble des valeurs aléatoires étant considéré comme un groupe de valeurs de longueur indéterminée.

3. Notion de groupe

A un même code mnémonique (ADD, SUB, MOVE, BR...) correspond un grand nombre d'instructions ; toutes celles qui sont considérées comme des fonctionnements représentatifs devront être activées au cours du test de conformité.

Exemple :

Instruction ADD <ea>, Dn du MC68000

Cette instruction peut se faire en 3 tailles (B,W,L) ; <ea> représente 1 parmi 14 modes d'adressage, Dn est 1 parmi les 8 registres de données. Dans le test de conformité, toutes les instructions correspondantes doivent être activées, c'est à dire toutes les combinaisons possibles de format, de mode et de registre d'adressage, et de registre données (environ 7600 combinaisons).

Il n'est pas question de décrire une à une chaque instruction effective associée à un même mnémonique. Le langage GAPT permet une description compacte d'une famille d'instructions effectives. Cette description est basée sur la notion de **groupe**.

Un groupe est un ensemble de registres, ou de modes d'adressage, ou de conditions, ou de formats, qui peuvent former un champ opérande associé à un code mnémonique.

Par exemple, pour l'instruction ADD size <ea>, Dn il faudra avoir défini :

- le groupe des formats size = (B,W,L)
- le groupe des modes d'adressage autorisés pour chacun des opérandes dans les différents formats.

Les groupes de modes d'adressage sont définis progressivement, par la définition :

- des groupes de registres, utilisés soit comme registres opérandes, soit comme registres adresses,
- des modes d'adressage élémentaires, en faisant appel éventuellement aux groupes de registres ; dans ce cas, un mode d'adressage élémentaire est en fait un groupe d'adressage,
- des groupes de modes d'adressage.

Exemples :

- groupes de registres

DN = (D0,... D7)

AN = (A0,... A7)

RN = (D0,... D7, A0, ... A7)

- modes d'adressage

l'adressage indirect indexé : d8 (AN,RN.W) correspond à 120 combinaisons.

- groupes de mode d'adressage

Pour l'instruction ADD.size <ea>, Dn, il faut utiliser 2 groupes :

- celui formé de tous les modes d'adressage : EA
- celui qui exclut l'adressage des registres adresse (interdit en format B) : EA'

Ces groupes étant définis, les 7600 instructions associées à "ADD size <ea>, Dn" peuvent être décrites par 2 pseudo-instructions :

- ADD.size EA, DN avec size = (W,L)
- ADD.size EA', DN avec size = (B)

A partir de ces deux pseudo-instructions, le système GAPT peut générer par itération les modules de test de toutes les instructions effectives.

4. Gestion de l'espace mémoire

Le programme de test est un programme en assembleur formé :

- d'une zone programme,

- et en général d'une ou plusieurs zones données.

Pour simplifier la gestion de la mémoire, chaque zone est remplie séquentiellement. Le système GAPT range séquentiellement chaque module élémentaire dans le fichier programme et réserve et initialise séquentiellement dans le fichier données adéquat les emplacements nécessaires aux opérandes et aux résultats.

Le nombre de zones données dépend étroitement des caractéristiques de l'adressage du microprocesseur, par exemple :

- formats d'adresse différents (ex MC68000)

- adressage segmenté (ex 8086)

De plus, l'utilisateur peut définir des zones résultats distinctes des zones opérandes.
La taille d'un programme de test est soumise à des limitations dues :

- soit au système hôte (capacité mémoire),

- soit aux caractéristiques de l'adressage du microprocesseur testé.

Ces limitations sont indiquées par l'utilisateur dans la description du microprocesseur ; les contraintes de taille peuvent porter sur les zones individuellement et/ou sur l'ensemble des zones.
Le logiciel GAPT gère automatiquement le partitionnement du programme de test en portions : modules de test, vérifiant ces contraintes.

5. Sources et puits d'une instruction

a) Définitions

A chaque instruction sont associés deux ensembles d'éléments de mémorisation : l'ensemble des sources et l'ensemble des puits.
L'élément de mémorisation S est une source pour l'instruction I si S est support d'une information utilisée (traitée) pendant l'exécution de I.

On distingue :

- les sources immédiates qui sont les informations contenues dans l'instruction (opérandes immédiats, adresse ou déplacement).
- les sources externes : l'information n'est pas immédiate et son support de mémorisation est externe au microprocesseur (mémoire centrale ou circuit périphérique).
- les sources internes : l'information n'est pas immédiate et le support de l'information est un élément de mémorisation interne du microprocesseur (registre ou bascule d'état).

L'élément de mémorisation P est dit puits pour l'instruction I si P est un support d'information dont la valeur est, ou peut être modifiée par l'exécution de I.

On distingue de même :

- les puits implicites, qui sont le compteur programme et le registre d'état, qui sont modifiés par l'exécution de presque toutes les instructions. Le compteur programme sera cependant mentionné explicitement comme puits dans le cas de rupture de séquence.
- les puits externes
- les puits internes

Suivant la nature de l'information qu'elle contient, une source peut être :

- une source de type adresse : son contenu sert à calculer l'adresse d'un support mémoire effectivement accédé au cours de l'exécution de l'instruction (comme source ou comme puits)
- une source de type opérande dans le cas contraire.

Une source de type opérande peut contenir :

- soit un opérande adresse
- soit un opérande donnée.

Exemples : MC6800

- instruction LDAA n,X

  X est une source interne de type adresse

  n est une source immédiate de type adresse

  le mot mémoire d'adresse (X)+n est une source externe de type opérande (donnée)

  Le registre A est puits interne.

- instruction JMP n,X (branchement à l'adresse (X)+n)

  X est une source de type opérande adresse, ainsi que n, car le mot mémoire d'adresse(X)+n n'est pas effectivement accédé.

b) Initialisation des sources et puits d'une instruction

L'initialisation de l'état interne du microprocesseur, précédant l'activation d'une instruction, consiste à :

- initialiser les sources de type opérande donnée soit avec les valeurs de test prédéterminées correspondant à l'opération référencée, soit avec des valeurs aléatoires.

On initialisera la mémoire grâce à une directive assembleur de définition de donnée et un registre grâce à une instruction de type "LOAD".

- initialiser les sources de type opérande adresse et les sources de type adresse avec des adresses (étiquettes).

On initialisera la mémoire grâce à une directive assembleur de définition d'adresse (dans le cas d'adressage indirect par mémoire), et un registre grâce à une instruction de type "LEA" (Load Effective Address).

- initialiser tous les autres registres internes (ceux qui ne sont pas sources) avec une valeur aléatoire s'il s'agit du test de conformité.

Le système GAPT désigne une adresse mémoire à l'aide d'une étiquette (appelée pointeur).
On aura un pointeur ZDi sur le début de chaque zone données i.
Une adresse mémoire dans la zone donnée i aura la forme :

ZDi + j avec $j \geq 0$.

La valeur absolue de cette adresse ne sera connue qu'après assemblage.

Le système GAPT initialise séquentiellement, au fur et à mesure de ses besoins, une zone données i. A un instant donné (K mots de la zone données i étant utilisés), l'adresse libre dans cette zone est égale à ZDi + K (valeur du pointeur courant sur cette zone).

II. Le langage GAPT

Le langage GAPT est un langage de description fonctionnelle de microprocesseurs et de circuits programmables.
La description d'un microprocesseur dans ce langage est décomposée en 3 parties :

- la description de l'architecture (registres, groupes de registres, mode d'adressage, groupes de modes d'adressage,...),

- la description du fonctionnement (jeu d'instructions),

- la description des opérandes de test spécifiques au microprocesseur testé.

Le découpage physique de la description d'un microprocesseur en 8 unités permet une indépendance relative dans l'écriture des unités et leur compilation :

- unité1 : formats, registres et groupes de registres
- unité2 : modes d'adressage et groupes de modes d'adressage
- unité3 : opérandes de test
- unité4 : utilitaires assembleur
- unité5 : signaux asynchrones
- unité6 : séquences d'initialisation et d'observation des éléments de mémorisation du microprocesseur
- unité7 : jeu d'instructions
- unité8 : description du test du comportement du microprocesseur face aux signaux asynchrones.

L'unité5 et l'unité8 concernent les signaux, et sont liées au testeur TEMAC. Elles ne sont pas développées dans ce manuel.
Il existe une unité0 : opérandes de test standard, qui permet de définir une base de données d'opérandes de test non spécifiques à un microprocesseur.

Le langage GAPT permet de décrire le jeu d'instructions d'un microprocesseur de manière compacte (similaire à celle proposée dans le manuel d'utilisation).
En conséquence, on décrira dans une unité7, des familles d'instructions paramétrées (un paramètre peut être un format, un mode d'adressage, un groupe de modes d'adressage,...). Il est possible de diviser la description du jeu d'instructions en un nombre variable d'unités de type 7, compilées séparément, afin de générer progressivement le programme de test.
La compilation d'une telle unité consiste à développer chaque famille d'instructions en un nombre fini de modules élémentaires ; il s'agit de la génération proprement dite du programme de test en assembleur.
Les unités de type 1 à 6 sont des unités "déclaratives", dont la compilation se résume, par définition, à mémoriser :
- les paramètres qui seront référencés dans la description d'une famille d'instructions,
- les informations nécessaires à la génération d'un module élémentaire.

Nous présentons ici les principales caractéristiques du langage sur un exemple.
Exemple :

Famille d'instructions du MC68000 : addition avec résultat stocké dans un registre.

ADD.size <ea>, Dn <ea> + Dn $\rightarrow$ Dn

Cette instruction a trois paramètres :

- "size" spécifie le format (B octet, W mot, L double mot),

- "ea" représente les modes d'adressages données du 68000,

- "Dn" désigne 1 parmi les 8 registres données (D0,...,D7).

Cette famille d'instructions effectives est décrite dans l'unité7 par

gen

'ADD.**'** **size** **' '** GDATAMA **','** DRD **:** $\leftarrow \alpha$

CCR, DDR PLUS (DDR, GDATAMA) $\leftarrow \beta$

size = (B **;** W **;** L**)** $\leftarrow \gamma$

fin_gen

La description d'une famille d'instruction est donc composée de trois parties :

α) la syntaxe assembleur de l'instruction,

β) la description fonctionnelle de l'instruction,

γ) des compléments d'information.

Chaque famille d'instructions est décrite dans un pseudo-assembleur : suite de parties fixes (entre quotes) et de parties variables (mode d'adressage : DRD, groupe de modes d'adressage : GDATAMA, format : size, ...)
L'utilisateur référence l'opérateur correspondant à la fonction de l'instruction (PLUS), en lui associant le ou les modes d'adressage des sources (GDATAMA, DRD) et des puits (DRD, CCR : registre code condition).
La description d'une instruction est éventuellement complétée par, entre autres, les valeurs d'itération du format (B, W ou L).

Toutes les informations nécessaires à la description et à la compilation d'une famille d'instructions (unité7) sont définies dans les unités de type déclaratif (unité0 à unité6).
L'opérateur PLUS ainsi que les opérandes qui lui sont associés dans les différents formats sont spécifiés dans l'unité3.
Les modes d'adressage (ou groupes de modes d'adressage) GDATAMA et DRD ont été définis au préalable dans l'unité2.
Le registre CCR et les registres (ou groupes de registres) impliqués dans les modes d'adressage ont été définis dans l'unité1, ainsi que les formats données B,W et L.
Les utilitaires assembleur nécessaires à la génération d'un programme sont décrits dans l'unité4.
On trouve dans l'unité6, les séquences d'initialisation et d'observation des registres.

Lors de la compilation d'une famille d'instructions, les fonctions essentielles de GAPT sont :
- l'itération sur les divers paramètres en vue d'obtenir toutes les instructions effectives correspondantes (itération sur les valeurs de format, les modes d'adressage, les registres, les valeurs d'opérandes). Cette phase met en oeuvre un processus de substitution qui consiste à remplacer toute partie variable, référencée dans la description syntaxique d'une famille d'instructions ou d'un mode d'adressage par sa "valeur assembleur".
- la gestion de l'adressage permettant de générer la séquence d'initialisation de chaque module élémentaire et les différentes zones données utiles à l'exécution des instructions testées.

Un module élémentaire du test de conformité des instructions du MC68000 contient 35 instructions :

- initialisation de 17 registres (D0, ..., D7, A0, ..., A7, CCR)

- instruction à tester

- observation des 17 registres.

La taille d'un module élémentaire de pseudo-conformité varie avec l'instruction testée. Par exemple, pour l'instruction : ADD.B (A0),D2 il y a 6 instructions :

- observation de CCR, A0 et D2,

- initialisation de A0 et D2,

- instruction ADD.B (A0),D2.

De même pour la taille d'un module élémentaire de balayage qui, pour la même instruction sera de 5 instructions :

- initialisation de A0 et D2

- instruction ADD.B (A0),D2

- observation de CCR et D2.

Le logiciel GAPT permet la génération du test comportemental d'une grande variété de circuits programmables. Il est évident que l'effort de description en langage GAPT n'est rentable que si la génération du test fait appel aux possibilités "intelligentes" de GAPT : itérations (en particulier sur les modes d'adressage), gestion de l'espace mémoire... Notons que l'utilisation du logiciel GAPT n'est pas limitée par les fonctions (au sens d'opérations) réalisées par le circuit : l'utilisateur peut définir des opérations en leur associant des opérandes de test (dans l'unité3), ou faire appel à la génération aléatoire de vecteurs.

On distingue deux types de circuits programmables :

- circuits maîtres (microprocesseurs, micro-ordinateurs...),

- circuits esclaves (coprocesseurs, circuit d'interface...).

En ce qui concerne les circuits maîtres, le test est effectué par exécution normale d'un programme par le circuit testé. Les difficultés possibles sont liées à la commandabilité et l'observabilité des éléments de mémoire internes, qui influent sur la longueur et la qualité du programme de test.
Le système GAPT peut traiter les circuits esclaves, si le test est effectué à travers le processeur maître. On décrit alors dans le langage GAPT, le sous-ensemble d'instructions du processeur maître qui active le circuit esclave. La commandabilité et l'observabilité des registres internes du circuit esclave est un problème d'autant plus critique qu'il s'agit d'un accès indirect à travers les instructions et les registres du processeur maître.

III. Les différentes unités

Forme générale de l'unité 1

unite1
nom du microprocesseur
taille_mot_memoire notation entière ! utilisé comme unité de référence sous le
! terme mot mémoire adréssable (mma)
taille_code_operation notation entière [**mma]**
format_operandes
donnees
(taille du format [**mma]** nom GAPT du format ['nom assembleur du format'],...)
adresses
(taille du format [**mma**] nom GAPT du format, ...)
compteur_programme notation entière nom GAPT du compteur ordinal
registre_code_condition nom GAPT du registre [**masque**]
registres

taille du registre nom GAPT du registre ['nom assembleur du registre'] [@]

registre simple

nom GAPT du registre ['nom assembleur du registre'] [@] (...(,), ...)

registre formé par l'association des 2 registres

description de registres simples ou associations

groupes_registres
nom GAPT de groupe de registres = (nom GAPT de registre,...)
fin

Exemple : unité 1 du MC68000

```
unite1
MC68000
taille_mot_memoire 8
taille_code_operation 16
format_operandes
donnees (8 B, 16 W, 32 L)
adresses (16 ADR16B, 16 ADR16H, 24 ADR24)
compteur_programme 32 PC
registre_code_condition CCR  masque
registres
        32 D0 32 D1 32 D2 32 D3
        32 D4 32 D5 32 D6 32 D7
        32 A0 @ 32 A1 @ 32 A2 @ 32 A3 @
        32 A4 @ 32 A5 @ 32 A6 @ 32 A7 @
         8 CCR
groupes_registres
        DN = (D0,D1,D2,D3,D4,D5,D6,D7)
        AN = (A0,A1,A2,A3,A4,A5,A6,A7)
        ANB = (A0,A1,A2,A3,A4,A5,A6)
fin
```

Forme générale de l'unité2

unite2
Nom microprocesseur
donnees
modes_adressage
nom mode d'adressage : syntaxe assembleur du mode / description GAPT du mode / taille du mode
groupes_modes
nom groupes modes = (nom mode,...) liste de modes
ou
= nom groupe modes **sauf** liste modes
ou
= nom groupes modes **et** liste modes
ou
= nom groupes modes + nom groupe modes
branchements
modes_adressage
.
.
.
groupes_modes
.
.
.
fin

Exemple : unité 2 du MC68000 (description partielle)

```
unite2
MC68000
donnees
modes_adressage
ARI : ‘(‘AN’)’ / mem * size (AN.ADR24) / 0
DRD : DN / DN.size / 0
ARD : AN / AN.size / 0
ARIWP : ‘(‘AN ‘+)’ / mem * size (AN.ADR24) / 0
ARIWPR : ‘(-’AN’)’ / mem * size (AN.ADR24-size) / 0
ARIWD : depl ‘(‘AN’)’ / mem * size (AN.ADR24+depl.16) / 2
ARIWIW : depl ‘(‘AN’,’RN’)’ / mem * size (AN.ADR24+depl.8+RN.W) / 2
ARIWIL : depl ‘(‘AN’,’RN’.L)’ / mem * size (AN.ADR24+depl.8+RN.L) / 2
ASAB : adr / mem * size (adr.ADR16B) / 2
ASAH : adr / mem * size (adr.ADR16H) / 2
ALA : adr / mem * size (adr.ADR24) / 4
PCWD : adr ‘(PC)’ / mem * size (PC+depl.16) / 2
PCWIW : adr ‘(PC, ‘RN’)’ / mem * size (PC+depl.8+RN.W) / 2
PCWIL : adr ‘(PC,’RN’.L)’ / mem * size (PC+depl.8+RN.L) / 2
IMMB ‘#’vi / vi.B / 2
IMMW ‘#’vi / vi.W / 4
IMML ‘#’vi / vi.L / 4
groupes_modes
IMM = (IMMB, IMMW, IMML)
AD = (ASAB, ALA, ASAH)
IMMAD = IMM + AD
branchements
modes_adressage
BALA : adr / adr.ADR24 / 2
BPCWDL : ‘.S ‘ adr / PC + depl.8 / 0
BPCWD : adr / PC + depl.16 / 1
groupes_modes
   .
   .
   .
fin
```

Forme générale de l'unité 3

unite3
nom du microprocesseur
groupes_valeurs
nom de groupe de valeurs [base] =
(‘V’,....;....; **alea** [**et** ‘V1’] [**ou** ‘V2’], **...**) : nom de format
nom de groupe de valeurs [base] =
nom de groupe de valeurs + nom de groupe de valeurs
groupes_chaine_car
nom de groupe de chaîne de caractères = (‘chaîne de caractère’,...;...;...)
nom de groupe de chaînes de caractères =
nom de groupe de chaîne de caractères + nom de groupe de chaîne de caractères
.
.
.
operateurs
nom d’opérateur (**alea**, **adr**, nom de groupe de valeurs, nom de groupe de chaînes de caractères)
nom d’opérateur (nom de groupe de valeurs, ..., nom de groupe de valeurs)
.
.
.
fin

La base, dans laquelle sont définies les valeurs, est par défaut la base hexadécimale.
Les mots clé définissant la base sont : **hexadecimal, decimal, binaire**.

Forme générale de l'unité 4

unite4
nom microprocesseur
zones_donnees
nom zone donnée **ram_externe** [taille limite]
. .
notation entière
definition_donnees
nom format donnée : [**align** = notation entière] **def_cte** = [‘chaîne’] **cte** [‘chaîne’]
[notation entière **mma**]
definition_adresses
nom format adresse : [**align** = notation entière] [**def_cte** = [‘chaîne’] **cte** [‘chaîne’]]
[notation entière **mma**] nom zone donnée associée

symboles_constantes
hexadecimal[‘symbole’] **valeur** [‘symbole’]
decimal [‘symbole’] **valeur** [‘symbole’]
binaire [‘symbole’] **valeur** [‘symbole’]
commentaire [‘chaîne’]
NOP ‘mnémonique’ taille instruction
BRINC ‘mnémonique’ [; ‘mnémonique’...] taille instruction
fin

Toutes les tailles indiquées ici sont en nombre de mma.

Exemple : unité 4 du MC68000

```
unite4
MC68000
zones_donnee
ZDC1 ram_externe
ZDL ram_externe
ZDC2 ram_externe
longueur_max_memoire 4096
definition_donnees
B : def_cte = ‘DC.B ‘ cte
W : align = 2  def_cte = ‘DC ‘ cte
L : align = 2  def_cte = ‘DC.L ‘ cte
definition_adresses
ADR16B : ZDC1
ADR24 : 4 mma ZDL
ADR16H : ZDC2
symboles_constantes
hexadecimal ‘0’ valeur ‘h’
decimal valeur
binaire ‘0’ valeur ‘b’
commentaire ‘;’
NOP ‘NOP’ 2
BRINC ‘BRA.S’ 2
fin
```

Forme générale de l'unité 6

unite6
nom microprocesseur
initialisation
reg nom registre : degré commandabilité / taille Si / Si
[degré commandabilité / taille Si / Si]
observation
reg nom registre : degré observabilité / taille So / So
[degré observabilité / taille So / So]
fin

Exemple : unité 6 du MC68000

```
unite6
MC68000
initialisation
reg D0 : 1 / 6 / ‘MOVE.L  #’ vi ‘, D0’
reg D1 : 1 / 6 / ‘MOVE.L  #’ vi ‘, D1’
reg D2 : 1 / 6 / ‘MOVE.L  #’ vi ‘, D2’
reg D3 : 1 / 6 / ‘MOVE.L  #’ vi ‘, D3’
reg D4 : 1 / 6 / ‘MOVE.L  #’ vi ‘, D4’
reg D5 : 1 / 6 / ‘MOVE.L  #’ vi ‘, D5’
reg D6 : 1 / 6 / ‘MOVE.L  #’ vi ‘, D6’
reg D7 : 1 / 6 / ‘MOVE.L  #’ vi ‘, D7’
reg A0 : 1 / 6 / ‘MOVEA.L  #’ vi ‘, A0’
        1 / 6 / ‘LEA  ‘ vi ‘, A0’
reg A1 : 1 / 6 / ‘MOVEA.L  #’ vi ‘, A1’
        1 / 6 / ‘LEA  ‘ vi ‘, A1’
reg A2 : 1 / 6 / ‘MOVEA.L  #’ vi ‘, A2’
        1 / 6 / ‘LEA ‘ vi ‘, A2’
reg A3 : 1 / 6 / ‘MOVEA.L  #’ vi ‘, A3’
        1 / 6 / ‘LEA ‘ vi ‘, A3’
reg A4 : 1 / 6 / ‘MOVEA.L  #’ vi ‘, A4’
        1 / 6 / ‘LEA ‘ vi ‘, A4’
reg A5 : 1 / 6 / ‘MOVEA.L  #’ vi ‘, A5’
        1 / 6 / ‘LEA ‘ vi ‘, A5’
reg A6 : 1 / 6 / ‘MOVEA.L  #’ vi ‘, A6’
        1 / 6 / ‘LEA ‘ vi ‘, A6’
reg A7 : 1 / 6 / ‘MOVEA.L  #’ vi ‘, A7’
        1 / 6 / ‘LEA ‘ vi ‘, A7’
reg CCR : 1 / 4 / ‘MOVE  #’ vi ‘, CCR’

observation
reg D0 : 1 / 6 / ‘MOVE.L D0,’ adr.ADR24
reg D1 : 1 / 6 / ‘MOVE.L D1,’ adr.ADR24
reg D2 :  1 / 6 / ‘MOVE.L D2,’ adr.ADR24
reg D3 : 1 / 6 / ‘MOVE.L D3,’ adr.ADR24
reg D4 : 1 / 6 / ‘MOVE.L D4,’ adr.ADR24
reg D5 : 1 / 6 / ‘MOVE.L D5,’ adr.ADR24
reg D6 : 1 / 6 / ‘MOVE.L D6,’ adr.ADR24
reg D7 : 1 / 6 / ‘MOVE.L D7,’ adr.ADR24
reg A0 : 1 / 6 / ‘MOVE.L A0,’ adr.ADR24
reg A1 : 1 / 6 / ‘MOVE.L A1,’ adr.ADR24
reg A2 : 1 / 6 / ‘MOVE.L A2,’ adr.ADR24
reg A3 : 1 / 6 / ‘MOVE.L A3,’ adr.ADR24
reg A4 : 1 / 6 / ‘MOVE.L A4,’ adr.ADR24
reg A5 : 1 / 6 / ‘MOVE.L A5,’ adr.ADR24
reg A6 : 1 / 6 / ‘MOVE.L A6,’ adr.ADR24
reg A7 : 1 / 6 / ‘MOVE.L A7,’ adr.ADR24
reg CCR : 1 / 6 / ‘MOVE SR,’ adr.ADR24
        1 / 10 / ‘ANDI.B #’ masque ‘, CCR’ ; ‘MOVE SR,’adr.ADR24
fin
```

Forme générale de l'unité7

unite7
nom microprocesseur
[**nb_modules_elementaires** nombre entier]
[taille 'INSTR' [;...]]
mode
gen [entier]
syntaxe de l'instruction : description fonctionnelle
[**size =** (liste de formats)]
[**masque =** 'valeur masque']
[taille supplémentaire en nombre de mma]
fin_gen
[taille 'INSTR' [;...]]
[mode]
gen
.
.
fin_gen

fin

Exemple : unité 7 du MC68000

```
unite7
M68000

pseudo_conformite

gen
‘ADD.’ size ‘ ‘ DRD ‘,’ ARI :
CCR OPAL2 (DRD, ARI)
size=(L)
fin_gen

gen
‘ADD.’ size ‘ ‘ ARIP ‘,’ DRD :
CCR OPAL2 (DRD, ARIP)
size=(B)
fin_gen

gen
‘ADD.’ size ‘ ‘ DRD.1 ‘,’ DRD.2 :
CCR OPAL2 (DRD.1, DRD.2)
size=(W)
fin_gen

gen
‘ADD.’ size ‘ ‘ DRA ‘,’ DRD :
CCR OPAL2 (DRD, DRA)
size=(W)
fin_gen

gen
‘AND.’ size ‘ ‘ DRD ‘,’ ARIPR :
CCR OPAL2 (DRD, ARIPR)
size=(L)
fin_gen

gen
‘AND.’ size ‘ ‘ ARIP ‘,’ DRD :
CCR OPAL2 (DRD, ARIP)
size=(B)
fin_gen

gen
‘AND.’ size ‘ ‘ DRD.1 ‘,’ DRD.2 :
CCR OPAL2 (DRD.1, DRD.2)
size=(W)
fin_gen

gen
‘EOR.’ size ‘ ‘ DRD ‘,’ ARID :
CCR OPAL2 (DRD, ARID)
size=(L)
fin_gen

        :
        :
        :
        :
        :
gen
‘SUB.’ size ‘ ‘ DRD.1 ‘,’ DRD.2 :
CCR OPAL2 (DRD.1, DRD.2)
size=(W)
fin_gen

gen
‘SUB.’ size ‘ ‘ DRA ‘,’ DRD :
CCR OPAL2 (DRD, DRA)
size=(W)
fin_gen

gen
‘ADDI.’ size ‘ ‘ IMM ‘,’ ARID :
```

```
CCR OPAL2 (IMM, ARID)
size=(L)
fin_gen

gen
‘ADDI.’ size ‘ ‘ IMM ‘,’ DRD :
CCR OPAL2 (IMM, DRD)
size=(B)
fin_gen

gen
‘SUBI.’ size ‘ ‘ IMM ‘,’ ARID :
CCR OPAL2 (IMM, ARID)
size=(B)
fin_gen

gen
‘SUBI.’ size ‘ ‘ IMM ‘,’ DRD :
CCR OPAL2 (IMM, DRD)
size=(W)
fin_gen

gen
‘ANDI.’ size ‘ ‘ IMM ‘,’ ARIP :
CCR OPAL2 (IMM, ARIP)
size=(W)
fin_gen

gen
‘ANDI.’ size ‘ ‘ IMM ‘,’ DRD :
CCR OPAL2 (IMM, DRD)
size=(L)
fin_gen

gen
‘ANDI.’ size ‘ ‘ IMMB ‘,CCR’ :
CCR OPAL2 (CCR, IMMB)
size=(B)
fin_gen

gen
‘ORI.’ size ‘ ‘ IMM ‘,’ ABSC :
CCR OPAL2 (IMM, ABSC)
size=(W)
fin_gen

gen
‘ORI.’ size ‘ ‘ IMM ‘,’ DRD :
CCR OPAL2 (IMM, DRD)
size=(L)
fin_gen

gen
‘ORI.’ size ‘ ‘ IMMB ‘,CCR’ :
CCR OPAL2 (CCR, IMMB)
size=(B)
fin_gen

gen
‘EORI.’ size ‘ ‘ IMM ‘,’ ABSL :
CCR OPAL2 (IMM, ABSL)
size=(W)
fin_gen

        :
        :

fin
```

Exemple : programme généré par GAPT pour le MC68000

Ce programme correspond à l'exemple d'unité7 donné précédemment

```
; zone programme
;
      MOVE   SR,ZDL+4
      MOVE.L A4,ZDL+6
      MOVE.L D1,ZDL+10
      LEA ZDL,A4
      MOVE.L #H'9D12FE15,D1
      ADD.L D1,(A4)
      MOVE   SR,ZDL+16
      MOVE.L D2,ZDL+18
      MOVE.L A1,ZDL+22
      MOVE.L #H'46B379E0,D2
      LEA ZDL+14,A1
      ADD.B (A1+),D2
      MOVE   SR,ZDL+26
      MOVE.L D0,ZDL+28
      MOVE.L D7,ZDL+32
      MOVE.L #H'A31DE4F4,D0
      MOVE.L #H'DC6FBEFE,D7
      ADD.W D7,D0
      MOVE   SR,ZDL+36
      MOVE.L D7,ZDL+38
      MOVE.L A0,ZDL+42
      MOVE.L #H'9173CBF2,D7
      MOVEA.L #H'117561EC,A0
      ADD.W A0,D7
      MOVE   SR,ZDL+50
      MOVE.L A5,ZDL+52
      MOVE.L D1,ZDL+56
      LEA ZDL+50,A5
      MOVE.L #H'EC0B442A,D1
      AND.L D1,(-A5)
      MOVE   SR,ZDL+62
      MOVE.L D3,ZDL+64
      MOVE.L A4,ZDL+68
      MOVE.L #H'CF4FDEB1,D3
      LEA ZDL+60,A4
      AND.B (A4+),D3
      MOVE   SR,ZDL+72
      MOVE.L D7,ZDL+74
      MOVE.L D0,ZDL+78
      MOVE.L #H'4AD234D5,D7
      MOVE.L #H'17CC8BE7,D0
      AND.W D0,D7
      MOVE   SR,ZDL+86
      MOVE.L A4,ZDL+88
      MOVE.L D0,ZDL+92
      LEA ZDL-5607,A4
      MOVE.L #H'170A382D,D0
      EOR.L D0,5689(A4)
      MOVE   SR,ZDL+96
      MOVE.L D1,ZDL+98
      MOVE.L D6,ZDL+102
      MOVE.L #H'DD38B42F,D1
      MOVE.L #H'BEA71118,D6
      EOR.B D6,D1
      MOVE   SR,ZDL+108
      MOVE.L D7,ZDL+110
      MOVE.L A0,ZDL+114
      MOVE.L #H'3319B235,D7
      LEA ZDL+106,A0
      MOVE   #H'D708,CCR
      CMP.B (A0+),D7
      MOVE   SR,ZDL+118
      MOVE.L D2,ZDL+120
      MOVE.L D7,ZDL+124
      MOVE.L #H'377705E6,D2
      MOVE.L #H'DF998380,D7
      MOVE   #H'2E9C,CCR
      CMP.W D7,D2
      MOVE   SR,ZDL+128
      MOVE.L D6,ZDL+130
      MOVE.L A3,ZDL+134
      MOVE.L #H'695AB13D,D6
      MOVEA.L #H'1A941B8F,A3
      MOVE   #H'A4FB,CCR
      CMP.W A3,D6
      MOVE   SR,ZDL+142
      MOVE.L A5,ZDL+144
      MOVE.L D2,ZDL+148
      LEA ZDL+2172,A5
      MOVE.L #H'E2933892,D2
      OR.L D2,-2034(A5)
      MOVE   SR,ZDL+154
      MOVE.L D0,ZDL+156
      MOVE.L A4,ZDL+160
      MOVE.L #H'7C9DD7C6,D0
      LEA ZDL+152,A4
      OR.B (A4+),D0
      MOVE   SR,ZDL+164
      MOVE.L D3,ZDL+166
      MOVE.L D4,ZDL+170
      MOVE.L #H'D15B043A,D3
      MOVE.L #H'809DCC74,D4
      OR.W D4,D3
      MOVE   SR,ZDL+178
      MOVE.L A5,ZDL+180
      MOVE.L D6,ZDL+184
      LEA ZDL+174,A5
      MOVE.L #H'72F3A772,D6
      SUB.L D6,(A5)
```

```
       MOVE   SR,ZDL+190
       MOVE.L D3,ZDL+192
       MOVE.L A1,ZDL+196
       MOVE.L #H'372FA659,D3
       LEA ZDL+188,A1
       SUB.B (A1+),D3
       MOVE   SR,ZDL+200
       MOVE.L D5,ZDL+202
       MOVE.L D7,ZDL+206
       MOVE.L #H'EF1A5FFD,D5
       MOVE.L #H'4E54C04F,D7
       SUB.W D7,D5
       MOVE   SR,ZDL+210
       MOVE.L D4,ZDL+212
       MOVE.L A1,ZDL+216
       MOVE.L #H'E6A89353,D4
       MOVEA.L #H'66526255,A1
       SUB.W A1,D4
       MOVE   SR,ZDL+224
       MOVE.L A2,ZDL+226
       LEA ZDL-6587,A2
       ADDI.L #H'AF2061EB,6807(A2)
       MOVE   SR,ZDL+230
       MOVE.L D3,ZDL+232
       MOVE.L #H'AFD43E91,D3
       ADDI.B #H'98,D3
       MOVE   SR,ZDL+238
       MOVE.L A3,ZDL+240
       LEA ZDL+16374,A3
       SUBI.B #H'10,-16138(A3)
       MOVE   SR,ZDL+244
       MOVE.L D3,ZDL+246
       MOVE.L #H'62AE6141,D3
       SUBI.W #H'3688,D3
       MOVE   SR,ZDL+252
       MOVE.L A2,ZDL+254
       LEA ZDL+250,A2
       ANDI.W #H'F64B,(A2+)
       MOVE   SR,ZDL+258
       MOVE.L D4,ZDL+260
       MOVE.L #H'4C7851E3,D4
       ANDI.L #H'5AA2F565,D4
       MOVE   SR,ZDL+264
       MOVE   #H'4EBD,CCR
       ANDI.B #H'14,CCR
       MOVE   SR,ZDL+266
       ORI.W #H'B67B,ZDC,W
;      :          :
;      :          :

;;     zone donnees courtes

ZDC    DC.W H'E2F0
       DC.L H'EC75D0EC
       DC.B H'62
       DC.B H'D7
       DC.W H'40C6
       DC.B H'6E
       DC.B H'E4
       DC.B H'2A
       DC.B H'C0
       DC.B H'26
       DC.B H'DC
       DC.B H'62
       DC.B H'38
       DC.B H'DE
       :       :
       :       :
;      zone donnees longues
;
ZDL    DC.L H'17684113
       DC.W H'CBA1
       DC.L H'453F748E
       DC.L H'F96D5204
       DC.B H'AB
       DC.B H'FC
       DC.W H'6457
       DC.L H'6A8265C5
       DC.L H'D5585743
       DC.W H'78BA
       DC.L H'7FD07FDB
       DC.L H'33B64629
       DC.W H'8748
       DC.L H'5D6EB101
       DC.L H'2BE4579F
       DC.L H'5CD9CBC0
       DC.W H'1C26
       DC.L H'22DC1AB7
       DC.L H'C1620125
       DC.B H'38
       DC.B H'7D
       DC.W H'26D4
       DC.L H'FC3B239A
       DC.L H'F7898BB0
       DC.W H'71D3
       DC.L H'A661C728
       DC.L H'F9FF964E
       DC.L H'BFA0B06B
       DC.W H'6906
       DC.L H'40BC6117
       DC.L H'6842AC85
       DC.W H'1E7A
       DC.L H'6790089B
       DC.L H'CD76B8E9
       DC.B H'AC
       DC.B H'2E
       DC.W H'ABC1
       DC.L H'4FA42C5F
       DC.L H'39EA6F8D
       DC.W H'67E5
       DC.L H'D0635F22
       DC.L H'B4714CF8
       DC.W H'1370
       DC.L H'12568A49
       DC.L H'F38C38A7
       DC.L H'C121B6E8
```

```
        DC.W H'EEBF
        DC.L H'B6ED9984
        DC.L H'E92BACCA
        DC.B H'F9
        DC.B H'02
        DC.W H'3345
        DC.L H'14D818C3
        DC.L H'E27ED1D1
        DC.W H'8F50
        DC.L H'A7366BA9
;       :           :
;       :           :
```