Recherche opérationnelle et optimisation pour la conception testable de circuits intégrés complexes

Lilia Zaourar

UNIVERSITÉ GRENOBLE 1 - JOSEPH FOURIER

Thèse
pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ JOSEPH FOURIER

Discipline : Mathématiques & Informatique

présentée et soutenue publiquement
par

Lilia ZAOURAR

Recherche opérationnelle et optimisation pour la conception testable de circuits intégrés complexes

Thèse dirigée par : Nadia BRAUNER et Yann KIEFFER

Soutenance prévue le 24 septembre 2010

JURY

Nadia BRAUNER,	Directeur	Université Joseph Fourier, Grenoble
Yann KIEFFER,	Co-Directeur	Institut polytechnique de Grenoble
Alix MUNIER KORDON,	Rapporteur	Université Pierre et Marie Curie, Paris 6
Matteo SONZA REORDA,	Rapporteur	Politecnico di Torino
Bruno ROUZEYRE,	Examinateur	Université de Montpellier
Stéphane DAUZERE-PERES,	Examinateur	Ecole des Mines de Saint-Etienne
Chouki AKTOUF,	Examinateur	DeFacTo Technologies
Arnaud WENZEL,	Invité	ST Microelectronics

Préface

"L'esprit de l'être humain est comme un parachute, c'est quand il est ouvert qu'il fonctionne le mieux..." ***Albert Einstein****,* ***(1879-1955)****.*

Pour ma part, j'ai ouvert mon parachute et sauté dans les montagnes de Grenoble avec pour seuls bagages la recherche opérationnelle et une soif de découvertes.

De là-haut, j'ai été entrainée dans une quête ambitieuse qui avait pour but de détecter le plus tôt possible les erreurs inévitables, il fallait pouvoir tout *observer* et *contrôler*. Mon objectif était maximiser la *couverture*. Pour cela, il me fallait construire des *modèles* théoriques. En réalité, j'ai surtout connu des personnes qui m'ont personnellement servi de modèle.

Au cours de ce voyage, j'ai d'abord rencontré quelqu'un qui avait une idée pour voler plus haut que tous. Il voulait faire du *high level scan*. J'ai décidé de le suivre.

Un parapentiste m'a ensuite mis le nez dans les *masques*. J'ai alors découvert que l'infiniment petit est infiniment grand. Avec des centaines de milliers de portes à traverser, seule l'*optimisation* pouvait m'aider. J'ai donc cherché les plus courts chemins, en évitant les chemins critiques pour arriver à temps. Le voyageur de commerce m'a beaucoup soutenu.

En haut de la Bastille, une personne m'a donnée quelques conseils très utiles pour m'éclaircir la vue et m'as permis d'aller en avant.

A un moment, j'ai eu la chance de m'associer à d'autres aventuriers. Ils possédaient des talents qui complémentaient les miens. Nous avons alors proposé une *solution* pour gérer toutes sortes de perturbations pouvant apparaître durant un vol. Cela valait un prix !

Arrivée à la dent de Crolles, des personnes m'attendaient avec des problèmes de *mémoire* à partager. Il s'est avéré que la solution était dans les gênes...

Et c'est ainsi que s'achève ce voyage qui n'a été possible que grâce à ceux qui m'ont permis de voler de mes propres ailes tout en orientant les vents de temps en temps.

Remerciements

Je tiens à remercier ici l'ensemble des personnes qui ont contribué à remplir ces trois années de thèse de moments agréables et dont la bonne humeur et la gentillesse ont joué un grand rôle dans le plaisir que j'ai eu à réaliser cette thèse.

En arrivant à Grenoble pour le master, j'ai été chaleureusement accueillie par les chercheurs et doctorants de l'époque et plus particulièrement par Nadia Brauner. Elle nous avait tous « embauchés » pour l'organisation du congrès *ROADEF* 2007. Cela m'a permis de m'intégrer rapidement dans le laboratoire et plus largement de connaître la communauté francophone de RO. Cela m'a également donnée l'envie de continuer dans le domaine passionnant de l'optimisation combinatoire. C'est aussi à Nadia que je dois de m'avoir convaincu de rester faire une thèse à Grenoble (malgré la proposition que j'avais au nord-est de la France). Elle m'a présenté Yann Kieffer, et c'est là que l'aventure de la thèse a commencé. Je tiens ici à la remercier pour tout cela et d'avoir également accepté d'être co-directrice de ma thèse et pas seulement « sur le papier ». J'ai été principalement encadrée par Yann que je tiens à le remercier tout d'abord pour sa gentillesse et la confiance qu'il m'a accordée dès le début alors que j'étais encore en master. Je lui suis reconnaissante d'avoir œuvré à me transmettre ses connaissances, son expérience et sa passion pour le métier d'enseignant chercheur. J'espère que notre collaboration continuera au-delà de la thèse.

Je souhaite ensuite remercier l'ensemble des membres de mon jury qui ont pris le temps de lire ma thèse et d'assister à la soutenance, en ce début d'année toujours chargé pour les enseignants chercheurs. En particulier, je tiens à remercier mes deux rapporteurs, Matteo Reorda Sonza, Professeur à l'école Polytechnique de Turin, et Alix Munier, Professeur à l'université Pierre et Marie Curie, qui m'ont fait l'honneur d'accepter de relire ce manuscrit de thèse et dont les rapports me rendent aujourd'hui fière du travail accompli. Je remercie également Alix Munier de m'accueillir dans son département après la thèse et de me donner l'opportunité de continuer à travailler à la frontière de la RO et de la micro-électronique. Je remercie chaleureusement Bruno Rouzyere, Professeur à l'université de Montpellier 2, pour avoir accepté d'assister en qualité d'examinateur à la soutenance de cette thèse. Je le remercie aussi pour ses conseils et remarques constructives au sujet du test en général et du scan en particulier, lors de nos différentes rencontres. Merci à Stéphane Dauzere Peres, Professeur à l'école des Mines de Saint-Etienne, d'avoir accepté de présider ce jury. Je le remercie pour l'attention qu'il a porté à mes travaux ainsi que ses remarques pour améliorer le manuscrit.

Mes premiers pas dans l'univers des circuits intégrés et du test se sont déroulés lors de mon stage de master 2 à *DeFacTo Technologies*. Je tiens à adresser mes profonds remerciements à Chouki Aktouf, Directeur Général de *DeFacTo Technologies*, de m'avoir donné cette opportunité et de m'avoir ouvert tant de portes. Il a été mon « troisième » directeur, avec ses encouragements et son soutien permanent. Je le remercie pour sa disponibilité infaillible, même lorsque souvent, il était à l'autre bout du monde. Enfin, merci de m'avoir fait l'honneur de faire partie de mon jury de thèse. J'en profite aussi pour remercier l'ensemble des ingénieurs de *DeFacTo* ; en particulier Vincent Juliard pour le travail accompli ensemble sur le scan et l'algorithme de stitching, Oussama et Alban pour leur soutien, Philippe Duchene pour les outils de placement et routage et enfin Nicolas et Sorin pour les scripts qui m'ont facilité la vie.

Je tiens à remercier l'ensemble des partenaires du projet ASTER, qui a financé ma thèse. En particulier, les membres du sous-projet 2 et *STMicroelectronics*. Un grand merci à Arnaud Wenzel et Fredric Grandvaux pour leur aide à optimiser le BIST. Malgré un démarrage tardif de ce sujet, nous sommes parvenus à livrer à temps notre prototype *MBO*. Je remercie en particulier Arnaud pour sa disponibilité et sa bonne humeur tout au long de notre collaboration et d'avoir accepté de faire partie du jury de ma thèse.

Je tiens à remercier vivement l'ensemble du personnel du laboratoire *G-SCOP*. En particulier l'étage du $3H$, les quelques uns qui se trouvent au $2F$ et les membres des domaines de compétence *ROSP* et *OC*. Malgré un emploi du temps souvent surchargé, ils ont toujours été disponibles pour répondre à mes questions avec beaucoup de sympathie. Merci pour vos nombreux conseils, votre aide et votre disponibilité tout au long de ces trois années de thèse. J'adresse également mes profonds remerciements à l'équipe administrative du laboratoire pour son efficacité face aux différents problèmes administratifs que j'ai rencontré. Merci à Yannick Frein, directeur du laboratoire, qui fait tout pour que notre lieu de travail soit un endroit convivial et où il fait bon de travailler. Merci à Roxane qui m'a fait un ordre de mission pour chaque déplacement à Moirans, à Souad également pour avoir toujours traité mes ordres de missions au plus vite, Fadila et Myriam pour les papiers de la préfecture, et bien sûre Kheira, Claudia et la superbe Amandine pour pleins pleins de choses que je ne peux citer ici...

Cette thèse est à la frontière de deux disciplines : la recherche opérationnelle, ma formation d'origine et la microélectronique, en particulier le Test qui été un terrain vierge pour moi où j'ai dû tout apprendre. Je tiens à remercier tous ceux qui ont éclairé ma lanterne lorsque les choses n'étaient pas évidentes à comprendre. En particulier, je voudrais remercier les membres d'un autre laboratoire, le *TIMA*. D'abord, un merci sincère et chaleureux à Nacer et Nic pour leur amitié et aide, bien au delà du Test. Merci aux membres de l'équipe *ARIS* de m'avoir aidé par de petits conseils et des orientations. N'étant pas du domaine, leur expertise m'a été précieuse. Merci à Régis pour ses conseils et enfin Lorena, Mounir et Haralampos pour la compagnie fort sympathique durant les conférences de Test. Un grand merci également à Laurent Fesquet et à Alexandre Chagoya du *CIME* pour les nombreuses discussions et échanges constructifs.

Un grand merci à tous les thésards avec qui j'ai partagé ce fameux bureau du $H320$ comme occupants permanents : Vassilissa, Louis Philippe et Julien ainsi que les occupants

« presque » permanents : Gregory (mon compagnon de promo depuis le master) et Valentin. Plus que des co-bureau, nous avons partagé la passion du travail de recherche, les différents soucis que l'on rencontre en tant que thésard et avec Susann Schrenk nous avons partagé l'aventure du *challenge ROADEF 2009*. En particulier, je tiens à remercier Vassilissa pour ses nombreuses relectures méticuleuses de ce manuscrit, des preuves de complexité et la chasse aux fautes d'orthographe et de style qu'elle a accomplie. Plus largement, un grand merci a tous les thésards de l'*ADOC* (la liste est trop longue) pour les midis jeux, les midis gym, les pauses café et les nombreux fous rires.

A ma petite famille grenobloise : Vassilisa, Thierry, Marion, Emilie, Loubna et les deux Pierres qui ont été présents à chaque instant, pour les nombreuses invitations qui ont souvent égayé mes weekends. Ainsi que pour m'avoir fait découvrir les plaisirs de la montagne, des raquettes et du ski. Merci également à Louis Philippe et Charlotte pour leur amitié et leur soutien pendant ces trois années à Grenoble et maintenant à Paris.

Enfin, je témoigne ma plus profonde gratitude à ma famille. À ma mère, ma « directrice de thèse de lombre », pour son soutien infaillible et inconditionnel, ses encouragements quelque soit les circonstances et son dévouement quotidien durant toutes ces années. J'espère qu'un jour j'aurais une carrière scientifique aussi brillante que la sienne. Je remercie aussi du fond du cœur mon père pour son optimisme et son soutien permanent ainsi que mes deux adorables petites sœurs qui m'ont entourée et soutenue durant toutes ces années et qui nont jamais cessé de me montrer leur affection malgré les longs mois que j'ai passé loin d'elles. Je les remercie tous de m'aimer, de me redonner confiance en moi et surtout de me supporter au quotidien hier pendant la thèse et demain pour la suite. Je finis mais ne finirai jamais de remercier mon tendre époux pour sa gentillesse, sa compréhension, son soutien réconfortant, sa patience et sa présence malgré la distance.

Lilia Zaourar, Grenoble, Vendredi 24 septembre 2010.

Sommaire

Préface **i**

Remerciements **i**

Introduction **3**

Partie I Concepts de base et rappels

1 Introduction à la Recherche Opérationnelle **9**
1.1 Introduction . . . 10
1.1.1 Apparition de la Recherche Opérationnelle . . . 10
1.1.2 Les domaines d'application de la Recherche Opérationnelle . . . 10
1.1.3 La démarche de modélisation et résolution en Recherche Opérationnelle 11
1.2 Les outils de la Recherche Opérationnelle . . . 12
1.2.1 La programmation linéaire . . . 12
1.2.2 La théorie des graphes . . . 14
1.2.2.1 Un bref historique . . . 14
1.2.2.2 Qu'est-ce qu'un graphe ? . . . 16
1.2.3 La théorie de la complexité . . . 17
1.2.3.1 Les classes $\mathcal{P}$ et $\mathcal{NP}$ . . . 17
1.2.3.2 Prouver la $\mathcal{NP}-$complétude d'un problème . . . 18
1.2.3.3 Quelques problèmes $\mathcal{NP}$-Complets . . . 18
1.3 Algorithmes et outils de résolution . . . 20
1.3.1 Résolution exacte . . . 20
1.3.1.1 Programmation dynamique . . . 20
1.3.1.2 Modélisation analytique et résolution . . . 20
1.3.1.3 Méthode de séparation et évaluation . . . 21
1.3.2 Résolution approchée . . . 21
1.3.2.1 Les algorithmes d'approximation . . . 22
1.3.2.2 Quelques principes classiques des heuristiques . . . 22

1.3.2.3 Les méta-heuristiques . . . 24
1.4 Conclusion . . . 26

2 Conception et test des circuits intégrés 27
2.1 Introduction . . . 27
2.2 Vue d'ensemble et définitions . . . 28
2.2.1 Eléments de base d'un circuit intégré . . . 28
2.2.1.1 Transistor . . . 28
2.2.1.2 Portes logiques et algèbre de Boole . . . 29
2.2.1.3 Les circuits combinatoires . . . 31
2.2.1.4 Circuits logiques à mémoire : les bascules . . . 31
2.2.2 Circuits intégrés, un peu d'histoire . . . 31
2.2.3 La loi de Moore . . . 32
2.3 Flot de conception et de fabrication des circuits intégrés . . . 34
2.3.1 Conception . . . 34
2.3.1.1 Enoncé des besoins : le cahier des charges . . . 34
2.3.1.2 Ecriture des spécifications . . . 34
2.3.1.3 Description RTL . . . 34
2.3.1.4 Synthèse logique . . . 36
2.3.1.5 Placement et routage . . . 36
2.3.1.6 Vérification et validation . . . 36
2.3.2 Réalisation . . . 37
2.3.2.1 Fabrication (Silicium) . . . 37
2.3.2.2 Test . . . 37
2.3.3 Zoom sur la CAO . . . 38
2.4 Zoom sur le test . . . 39
2.4.1 Le test des CI et son importance . . . 39
2.4.2 Les différents types de test . . . 40
2.4.3 Test et conception en vue du test . . . 41
2.4.4 Scan Path pour le test des CI . . . 41
2.5 Les mémoires . . . 42
2.5.1 Généralités sur les mémoires . . . 42
2.5.2 Le test des mémoires . . . 43
2.5.3 Le BIST mémoire . . . 43
2.6 Conclusion . . . 44

Partie II Technique du Scan Path au niveau RTL

3 Problème et Modélisation de l'insertion des chaînes de scan 49
3.1 Introduction . . . 49

3.2 Méthode du test par chaînes de scan . . . 50
3.2.1 Définition . . . 50
3.2.1.1 Principe de l'insertion des chaînes de scan à bas niveau . . . 51
3.2.2 Approche de scan intégral à haut niveau : Scan RTL . . . 54
3.2.2.1 Principe de l'insertion des chaînes de scan haut niveau . . . 55
3.2.3 Comparaison entre les deux flots d'insertion . . . 56
3.3 Problème de l'insertion des chaînes de scan . . . 58
3.3.1 Problématique . . . 58
3.3.2 Objectif de la thèse pour le scan . . . 59
3.3.3 Etat de l'art . . . 59
3.4 Modélisation de l'insertion des chaînes de scan au niveau RTL . . . 61
3.4.1 Données . . . 61
3.4.2 Objectifs . . . 61
3.4.3 Contraintes . . . 62
3.5 Formulation mathématique du problème . . . 62
3.5.1 Design Graph, G_D . . . 63
3.5.2 Proximity Graph, G_P . . . 63
3.5.3 Construction du graphe G_P à partir de G_D . . . 64
3.5.4 Critère d'optimisation . . . 65
3.6 Conclusion . . . 66

4 Solution Algorithmique pour l'insertion des chaînes de scan 67
4.1 Introduction . . . 67
4.2 Position du problème . . . 68
4.2.1 Etude bibliographique . . . 69
4.2.2 Analyse de la complexité du problème . . . 69
4.2.2.1 Etude des différents cas en fonction de la taille des chaînes 70
4.2.3 Passage de plusieurs chaînes à une seule . . . 70
4.3 Analogie entre le problème des chaînes de scan et celui du TSP . . . 72
4.4 Résolution heuristique du problème du voyageur de commerce . . . 73
4.4.1 Quelques définitions . . . 74
4.4.2 Une 2-approximation pour le TSP . . . 75
4.4.3 L'heuristique de Christofidès . . . 75
4.5 Méthode de résolution pour l'insertion des chaînes de scan optimales . . . 76
4.6 Résultats numériques . . . 78
4.6.1 Description des instances, benchmarks et open cores . . . 78
4.6.2 Construction des graphes et optimisation du scan . . . 78
4.6.3 Protocole expérimental . . . 80
4.7 Conclusion . . . 83

Partie III Le BIST mémoire

5 Problème du partage des *BIST* mémoire **87**
5.1 Introduction . . . 87
5.2 Insertion du BIST mémoire . . . 88
5.2.1 Pourquoi et comment partager les blocs BIST . . . 88
5.2.2 La solution actuelle . . . 90
5.2.3 Objectif de la thèse pour le partage de blocs *BIST* . . . 90
5.3 Etat de l'art . . . 91
5.4 Modélisation du Problème du partage des blocs *BIST* . . . 91
5.4.1 Données . . . 92
5.4.2 Contraintes . . . 92
5.4.2.1 Régles de partage d'un collier . . . 93
5.4.2.2 Mémoires partageant le même collier . . . 93
5.4.2.3 Les paramètres utilisateurs . . . 94
5.4.3 Objectifs . . . 94
5.4.3.1 La surface . . . 94
5.4.3.2 La puissance . . . 94
5.4.3.3 Le temps de test . . . 95
5.4.4 Une solution au problème . . . 95
5.5 Conclusion . . . 95

6 Résolution du problème de partage des BIST mémoire **97**
6.1 Introduction . . . 97
6.2 Evaluation des critères d'optimisation . . . 98
6.2.1 Calcul des critères d'optimisation . . . 98
6.2.1.1 La surface . . . 98
6.2.1.2 La puissance . . . 99
6.2.1.3 Temps de test . . . 99
6.2.2 Calcul des bornes . . . 100
6.2.3 Optimisation multi-objectif et solution de compromis . . . 101
6.2.3.1 Quelques définitions . . . 101
6.2.3.2 Les algorithmes génétiques . . . 103
6.3 Méthode de résolution : Memory BIST Optimizer . . . 104
6.3.1 Module 1 : Modélisation et création des groupes de compatibilité . . . 105
6.3.2 Module 2 : Optimisation des mémoires . . . 107
6.3.3 Module 3 : Validation des colliers pour les mémoires . . . 109
6.4 Implémentation & résultats numériques . . . 110
6.4.1 Protocole expérimental et description des instances . . . 110
6.4.2 Résultats numériques & Discussion . . . 111
6.5 Conclusion . . . 115

Conclusion générale **117**

Bibliographie **121**

Résumé **127**

Liste des figures

1.1 Schéma général du processus d'aide à la décision. 11
1.2 Les sept ponts de Königsberg. 14
1.3 Représentation graphique de graphe non orienté et orienté. 16
1.4 Enchaînement des preuves de NP-Complétude. 19

2.1 Les différents types de portes. 30
2.2 Exemple d'une bascule D. 31
2.3 Loi de Moore. 33
2.4 Flot traditionnel de fabrication d'un circuit intégré 35
2.5 Principe de bases des vecteurs de test. 38
2.6 Testeur. 39

3.1 Niveau d'insertion du scan traditionnel. 51
3.2 Structure générale d'un circuit séquentiel. 52
3.3 Description d'un circuit séquentiel muni d'une chaîne de scan. 52
3.4 Architecture générale de la technique de scan path. 53
3.5 Niveau d'insertion du scan RTL. 54
3.6 Principe de la méthode du Scan RTL. 55
3.7 Comparaison des Flots d'insertion. 57
3.8 Graphe G_P. 65

4.1 Exemple d'application de la 2-approximation. 75
4.2 Algorithme de stitching des chaînes de scan RTL. 76

5.1 Architecture *BIST* dédiée. 88
5.2 Architecture *BIST* partagée. 89
5.3 Architecture *BIST* partagée avec colliers de différents types. 89

6.1 Diagramme puissance-temps de test pour chaque type de collier. 100
6.2 Front de Pareto. 102
6.3 Memory BIST Optimizer. 105
6.4 Intégration du module GRP. 106
6.5 Processus d'exécution des algorithmes génétiques. 107
6.6 La surface des trois circuits. 112
6.7 Temps d'exécution de MBO. 113

6.8 Sortie du module OPT. 113
6.9 Sortie simplifiée : Puissance vs Temps. 114
6.10 Sortie simplifiée : Surface vs Temps test. 114
6.11 Sortie simplifiée : Surface vs Puissance. 115

Liste des tables

4.1 Description des circuits. 78
4.2 Caractéristiques des graphes. 79
4.3 Longueurs des chaînes et temps d'exécution des heuristiques. 80
4.4 Table WL avec la stratégie de partition. 81
4.5 Table WL avec la stratégie de chaîne découpée. 82
4.6 Résultats de couverture de fautes. 83

6.1 Système BIST pour 5 mémoires. 108
6.2 MBO Résultats. 111

Introduction générale

De la médecine aux loisirs, du fond des océans jusqu'aux confins du système solaire, les *puces électroniques* sont présentes à chaque pas de notre vie quotidienne.

En effet, de nos jours des *circuits intégrés* complexes sont utilisés dans un nombre croissant d'applications critiques, au niveau des vies humaines, du transport (comme par exemple l'avionique ou le transport terrestre, vol spatial) ou encore au niveau financier (comme dans le domaine bancaire). Leur utilisation est aussi croissante dans des applications comme le contrôle de processus industriel ou dans la vie de tous les jours (pour ne citer que la téléphonie mobile, etc).

Avec une complexité actuelle de plus de 100 millions de *transistors*, les *circuits* posent naturellement des problèmes variés au niveau du **test** et quelques challenges dans le domaine de la **recherche opérationnelle** à résoudre...

Le test des *circuits intégrés* n'est pas un domaine nouveau, mais il est cependant en perpétuelle mutation. En effet, l'évolution vers la haute intégration (*VLSI* : "*Very Large Scale Integration*") a eu pour conséquence l'impossibilité de tester efficacement le circuit en production si le test n'a pas été prévu pendant la conception. La qualité et le coût du test sont devenus directement liés aux choix de conception et aux informations fournies par le concepteur pour la préparation du test.

Dans ce contexte, la notion de conception en vue du test (*DFT* : "*Design for Testability*") est apparue. En effet, dans un nombre croissant de cas, des dispositifs spécifiques doivent être ajoutés dans le *circuit* pour permettre d'atteindre le niveau de qualité de test requis. Ces aspects du test, relatifs aux différentes étapes de conception, seront l'objet essentiel de notre travail.

En effet, cette thèse est à l'interface entre la micro-électronique et la recherche opérationnelle. Les problèmes traités par la recherche opérationnelle et l'optimisation combinatoire couvrent des thèmes très variés liés à différentes industries comme le transport, les systèmes de productions, l'écologie, les télécommunications, la conception de circuit intégré. En particulier, de nombreux travaux d'optimisation combinatoire sont issus du problème de conception et test de circuits intégrés. Nous nous intéressons dans ce travail à la conception en vue du test des circuits intégrés. Ce travail porte sur l'utilisation de techniques d'*optimisation combinatoire* pour optimiser deux méthodes de *DFT* les plus

utilisées actuellement : le *scan path* pour le test de tous types de circuits intégrés et le *BIST* (*Built In Self Test*) pour le test intégré des mémoires.

Notons que ces travaux de thèse ont été menés dans le cadre du projet de *Pôle de compétitivité Minalogic* : **ASTER** (Architecture pour mémoires STatiques haute pERformance) portant sur la conception en vue du test et le développement d'outils logiciels de test pour les systèmes sur puce en particulier pour les mémoires.

Notons également qu'il a donné lieu à deux collaborations industrielles, l'une avec la start-up *DeFacTo Technologies* innovatrice dans le domaine de la *DFT*, et l'autre avec l'entreprise *STMicroelectronics* qui fait partie des grandes entreprises dans ce domaine.

Le présent travail se découpe en trois grandes parties, qui correspondent à une partie introductive et deux parties consacrées aux deux méthodes de DFT : le *Scan Path* et le *BIST* (*Built In Self Test*) mémoire.

Le plan de ce rapport a été pensé de telle manière que certains chapitres puissent être lus indépendamment des autres. En effet, les parties 2 et 3 peuvent être lues indépendamment.

Étant donné la diversité des sujets abordés dans cette thèse, les états de l'art relatifs à chacun des aspects traités ne sont pas regroupés au sein d'un seul chapitre, mais distribués à l'intérieur de chacun des chapitres concernés.

La première partie est introductive. Nous y présentons les notions de bases de la recherche opérationnelle ainsi que de la conception en vue du test des circuits intégrés. Le premier chapitre rassemble les définitions et techniques de résolution utilisées dans le domaine de l'optimisation combinatoire et de la recherche opérationnelle pour traiter des problèmes classiques issues de problématiques industrielles. Dans le second chapitre, nous présentons d'abord les composants de bases d'un circuit intégré suivie du flot de conception et réalisation de celui-ci. Nous nous penchons ensuite sur le test et la *DFT* son importance ainsi que son coût. En particulier nous nous intéressons à la méthode du *Scan Path* pour le test des circuits intégrés, puis à celle du *BIST* pour le test des mémoires qui seront les deux problèmes traités dans ce travail.

La seconde partie porte sur l'insertion de la méthode du *Scan Path* au niveau *RTL* (*Register Transfert Level*). Cette méthode est la plus utilisée actuellement pour le test des circuits intégrés mais son insertion est habituellement faite à un niveau plus bas que celui proposé ici.

En guise d'introduction, le troisième chapitre décrit les principes de la méthode du *Scan Path*, les différents niveaux d'insertion possibles ainsi que le problème de l'optimisation des chaînes de scan au niveau *RTL*.

Le quatrième chapitre est dédié à la modélisation ainsi qu'aux méthodes de résolution que nous proposons pour ce problème. Elles utilisent les techniques de recherche opérationnelle et d'optimisation combinatoire. Les résultats numériques obtenus sont décrits à la fin de ce chapitre.

La troisième partie concerne le problème d'optimisation du partage de blocs *BIST* pour le test intégré des mémoires. L'adoption de la méthode du *BIST* pour le test des mémoires est aujourd'hui généralisée. Mais avec l'augmentation du nombre de mémoires dans un circuit, l'insertion de *BIST* dédié n'est plus une solution envisageable à cause des coûts supplémentaires liés à cette intégration. L'idée de partager un bloc *BIST* entre plusieurs mémoires est alors une possibilité pour remédier à cela.

Dans le cinquième chapitre, nous présentons les différents types d'architectures de partage de blocs *BIST* possibles ainsi qu'une analyse mathématique du problème.

Le chapitre six présente la méthode de résolution avec le prototype *Memory BIST Optimizer* que nous avons développé afin de déterminer les meilleures solutions pour ce problème. Des résultats expérimentaux sont donnés à la fin de cette partie.

Première partie

Concepts de base et rappels

"Il est bien plus beau de savoir quelque chose de tout que de savoir tout d'une chose. " ***Blaise Pascal****, (**1623-1662**).*

Dans cette première partie, nous allons faire un bref rappel des définitions et notions de base. Notre objectif est de clarifier et de justifier d'une part certaines affirmations et d'autre part la démarche adoptée dans ce mémoire pour résoudre les problèmes étudiés.

L'objectif est de donner une vue d'ensemble des concepts importants du domaine de la *Recherche Opérationnelle* (*RO*), et de la *conception* et du *test* des *Circuits Intégrés* (*CI*), sans chercher à être exhaustif.

Les informations données permettront à chacun d'approfondir les points nécessaires à la compréhension des problèmes abordés et des techniques de résolution proposées dans le contexte des travaux de recherche de cette thèse.
Le chapitre 1, s'adresse aux électroniciens désirant une ouverture dans la *RO*. Il présente les notions de base de la *RO*, les domaines d'applications et les techniques de résolutions.

De même le chapitre 2, s'adresse à la communauté de *RO* désirant connaître l'univers de la *conception et test* des *CI*.

Des références bibliographiques ainsi que plusieurs sites *web* sont disponibles dans les deux chapitres pour le lecteur intéressé d'approfondir certaines notions.

Chapitre 1

Introduction à la Recherche Opérationnelle

Sommaire

1.1 Introduction 6
1.1.1 Apparition de la Recherche Opérationnelle 6
1.1.2 Les domaines d'application de la Recherche Opérationnelle . . . 6
1.1.3 La démarche de modélisation et résolution en Recherche Opérationnelle 7
1.2 Les outils de la Recherche Opérationnelle 8
1.2.1 La programmation linéaire 8
1.2.2 La théorie des graphes 10
1.2.2.1 Un bref historique 10
1.2.2.2 Qu'est-ce qu'un graphe ? 12
1.2.3 La théorie de la complexité 13
1.2.3.1 Les classes $\mathcal{P}$ et $\mathcal{NP}$ 13
1.2.3.2 Prouver la $\mathcal{NP}-$complétude d'un problème 14
1.2.3.3 Quelques problèmes $\mathcal{NP}$-Complets 14
1.3 Algorithmes et outils de résolution 16
1.3.1 Résolution exacte 16
1.3.1.1 Programmation dynamique 16
1.3.1.2 Modélisation analytique et résolution 16
1.3.1.3 Méthode de séparation et évaluation 17
1.3.2 Résolution approchée 17
1.3.2.1 Les algorithmes d'approximation 18
1.3.2.2 Quelques principes classiques des heuristiques 18
1.3.2.3 Les méta-heuristiques 20
1.4 Conclusion 22

1.1 Introduction

Au cours de ce chapitre, nous présentons dans un premier temps l'apparition de la *RO* et ses domaines d'application. Nous donnons ensuite un panorama des techniques et outils de la *RO* tels que la théorie des graphes et la programmation linéaire ainsi que certains éléments sur la complexité des problèmes. Enfin, nous présentons quelques méthodes classiques de résolution. Les différentes parties de ce chapitre ont été rédigées en s'inspirant de ouvrages de référence en *RO* tels que les livres de *Sakarovitch* [53], *De Werra et al.* [34], *Fournier et al.* [25] et *Alj et al.* [7].

1.1.1 Apparition de la Recherche Opérationnelle

La *Recherche Opérationnelle* est née lors de la deuxième guerre mondiale des efforts conjugués d'un groupe d'éminents mathématiciens (dont *Von Neumann*, *Metropolis*, *Wald Wiener*, *Danzing* et *Bellman*) qui ont, à leur manière, contribué à la victoire des *Alliés*. C'est dans ce contexte qu'a été créée une nouvelle méthodologie quantitative d'aide à la décision qui se caractérise par deux mots clés : *modélisation* et *optimisation*.

La *Recherche Opérationnelle* apporte donc des *méthodes de modélisation* et une *collection d'outils mathématiques* devenus indispensables pour l'ingénieur, le manager ou encore l'économiste. En effet, le pouvoir d'expression de ses modèles, c'est-à-dire leur capacité à représenter en termes mathématiques les systèmes les plus complexes, ainsi que l'efficacité de certains de ses algorithmes pour les résoudre, sont la clé de l'exploitation des moyens informatiques modernes dans l'aide à la décision.

1.1.2 Les domaines d'application de la Recherche Opérationnelle

Rapidement, la *Recherche Opérationnelle* repousse les frontières de son domaine d'application initial, à savoir la résolution et la gestion de problématiques dans le domaine militaire. Elle s'étend alors à la modélisation et à l'optimisation de systèmes dans les domaines les plus divers, allant de la conception et la configuration à l'exploitation de systèmes techniques complexes (réseaux de communication, systèmes informatiques, etc.), de la gestion stratégique d'investissements à celle de la chaîne logistique (transport, production, stocks, etc.).

La *recherche opérationnelle* fait alors son apparition dans des domaines tels que la santé et l'instruction publique, la voirie, le ramassage et la distribution de courrier, la production et le transport d'énergie, les télécommunications, ainsi que les banques et les assurances. De pair avec cette éclosion, les méthodes et les fondements théoriques de la *recherche opérationnelle* continuent à se développer dans une association féconde avec d'autres domaines des mathématiques appliquées [34].

Dans ce mémoire, nous abordons un domaine où la *recherche opérationnelle* est présente : la *micro-électronique* et plus précisément la conception et le **test** des *circuits intégrés* (*CI*). La plupart des méthodes de *recherche opérationnelle* utilisées dans ce domaine concernent les étapes de *placement et routage* (voir définition chapitre 2) avec quelques ap-

plications en *synthèse* (voir définition chapitre 2) ainsi que l'optimisation de la production spécialement à l'étape *placement et routage*. Le livre de *Korte et al.* [40] est une référence dans ce domaine.
Notre travail consiste en l'application des méthodes de la *recherche opérationnelle* et de l'**optimisation combinatoire** pour l'amélioration du **test** des *CI*.

1.1.3 La démarche de modélisation et résolution en Recherche Opérationnelle

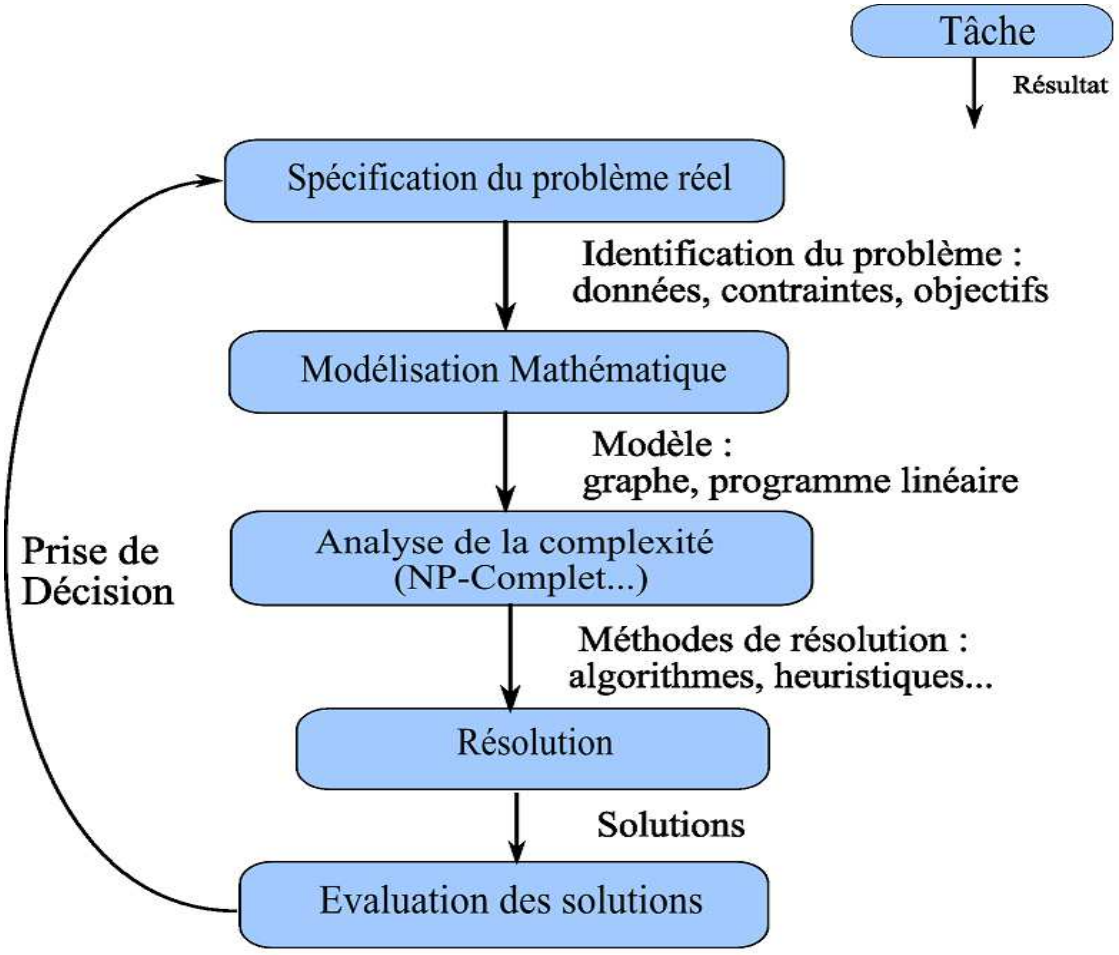

FIG. 1.1: Schéma général du processus d'aide à la décision.

La démarche scientifique de *RO* et d'aide à la décision se décompose en 5 étapes principales comme l'indique le schéma présenté en figure 1.1.

Le processus commence par une spécification précise du problème à résoudre. Cet aspect concerne la définition des données, des éventuelles contraintes liées au problème et des objectifs à atteindre. Cette phase est fondamentale dans le processus d'optimisation, dans la mesure où elle conditionne les étapes suivantes.

Après une analyse fine du problème et la compréhension du système arrive l'étape de modélisation. Elle consiste à traduire le problème physique en un problème mathématique équivalent en utilisant les outils de *RO* (qui seront décrits plus bas dans ce chapitre).

Ensuite vient l'étape d'analyse de la complexité du problème qui guidera le choix d'une méthode de résolution. La recherche de solutions est réalisée à l'aide de méthodes d'optimisation mathématiques, pouvant présenter diverses caractéristiques (mono-objectif, multi-objectifs, etc.).

Une fois le problème résolu, il est impératif d'évaluer la pertinence du résultat fourni par l'optimisation et, en cas d'échec, de s'interroger sur les choix adoptés lors des différentes phases précédentes.

Enfin, la validation est sans doute l'étape la plus délicate et la plus coûteuse du processus d'optimisation. En effet, la modélisation mathématique d'un problème n'est jamais unique, en particulier, elle dépend des paramètres et critères de performance choisis.

Nous serons dans ce travail de thèse confrontés aux problèmes de modélisation ainsi que de résolution de problèmes d'optimisation liés au domaine de la microélectronique, comme nous le verrons dans les chapitres suivants. Mais d'abord nous présentons ici les différents outils de modélisation et résolution de la *RO*.

1.2 Les outils de la Recherche Opérationnelle

1.2.1 La programmation linéaire

Généralement on appelle *programmation mathématique* la recherche de l'optimum d'une fonction de plusieurs variables liées entre elles par des contraintes sous forme d'équations ou d'inéquations. Nombreux sont les *problèmes de décision* (problèmes identifiés par des contraintes et des objectifs) qui se ramènent à un modèle de programmation mathématique. Si la fonction à optimiser et toutes les contraintes sont linéaires, on a alors affaire à un problème de *programmation linéaire*. Ces dernières années, des problèmes très divers ont été résolus grâce à l'utilisation de la programmation linéaire. En fait, le modèle est relativement général et permet de traiter une quantité de problèmes, aussi bien théoriques qu'appliqués. Cet engouement est aussi dû au fait que, dans de nombreux cas, il existe des algorithmes extrêmement efficaces pour obtenir des solutions.

La méthode du *simplexe* développée par *G. B. Dantzig* vers 1947 a conduit à plusieurs algorithmes généraux qui permettent de résoudre aisément des problèmes de taille considérable (plusieurs dizaines de milliers de variables et plusieurs centaines de milliers de contraintes). Un livre de référence sur ce sujet est celui de *Guéret et al.* [32].

Voici un petit exemple de problème pouvant être modélisé en Programme Linéaire (PL):

Une usine asiatique produit deux types de silicium rapportant respectivement 5000$ et 7000$ par tonne. Le premier (type 1) est destiné à la fabrication de puces électroniques, le second (type 2) à celles de cellules photovoltaïques. Le silicium est extrait de son oxyde par des procédés métallurgiques dans une première étape. Ensuite, il est passé dans un four à arc électrique ouvert dont la puissance peut aller jusqu'à environ 30 MW.

Pour faire une tonne de silicium de type 1, il faut 40 minutes d'extraction et 20 minutes au four. Pour fabriquer une tonne de silicium de type 2, il faut 30 minutes d'extraction et 30 minutes au four. L'atelier d'extraction est disponible 360 minutes par jour et le four 480 minutes par jour. Combien de silicium de chaque type peut-on produire par jour pour maximiser le bénéfice ? [1].

Ce problème se modélise aisément par le programme linéaire suivant :

Soient les variables de décision x_1 et x_2 représentant respectivement la quantité de silicium de type 1 et 2 à produire par jour.

$$\begin{cases} \text{Max } Z = 5000x_1 + 7000x_2 & (1) \\ \quad 40x_1 + 30x_2 \leq 360 & (2) \\ \quad 20x_1 + 30x_2 \leq 480 & (3) \\ \quad x_1,\, x_2 \geq 0 & (4) \end{cases} \tag{1.1}$$

La ligne (1) représente le profit total Z qui est le critère à optimiser. On l'appelle aussi *fonction objectif*, *fonction de coût* ou *fonction économique*. *Max* signifie que ce critère doit être maximisé; on mettrait *Min* pour minimiser. Les lignes (2) et (3) désignent les *contraintes* du problème. La contrainte (2) concerne la disponibilité de l'atelier : elle stipule que le temps total du procédé requis pour l'extraction du silicium ne doit pas dépasser 360 min. La contrainte (3) décrit de même la disponibilité du four. Les contraintes (4) précisent le domaine de définition des variables.

Remarque 1. Lorsque les variables de décision sont entières, on parle de *Programme Linéaire en Nombre Entiers* (PLNE). Ce sont des problèmes bien plus difficiles à résoudre. Il existe cependant des logiciels qui résolvent efficacement un grand nombre de *PLNE*. Le livre de *Wolsey* [50] est entièrement dédié à ce type de problèmes (qui représentent une grande partie des problèmes de *RO*).

Remarque 2. Lorsque le problème contient plusieurs objectifs à optimiser on parle de programmation *multi-objectifs*. Nous reviendrons sur cette notion dans le chapitre 6.

Dans le reste de ce travail, nous n'utilisons pas la *PL*. Elle a été présentée juste à titre d'exemple d'outils de modélisation en RO.

[1] exemple : source du procédé wikipédia

1.2.2 La théorie des graphes

Un deuxième outil complémentaire à *la programmation linéaire* et très utilisé pour la modélisation et la résolution des problèmes de *RO* est la *théorie des graphes*. Nous donnons dans cette section les définitions de base. Pour plus d'information à ce sujet, un lexique est disponible à la fin du document. Plusieurs références bibliographiques sont aussi données pour le lecteur souhaitant approfondir certaines notions.

1.2.2.1 Un bref historique

L'histoire de la *théorie des graphes* débute avec les travaux d'*Euler* $(1707 - 1783)$ au $XVIIIe$ siècle. On considère usuellement que la *théorie des graphes* est née en 1736 avec la communication d'*Euler* dans laquelle il proposait une solution au célèbre problème des *ponts de Königsberg*. Le problème posé était le suivant :

La ville de *Königsberg*, était la capitale de la *Prusse* de l'Est, aujourd'hui rebaptisée *Kaliningrad*). Elle a été fondée par le roi *Ottokar de Bavière* et abrite une des plus anciennes universités (1544). Elle possède le fleuve Pregel (aujourd'hui Pregolya) qui traverse la ville et entoure deux îles A et D. Sept ponts permettent de traverser la ville, voir figure 1.2.

La question était la suivante : est-il possible pour le roi, en partant d'un quartier quelconque de la ville, d'effectuer un circuit qui emprunte chacun des ponts une et une seule fois afin de faire le tour des deux îles et de saluer l'ensemble de son peuple ?

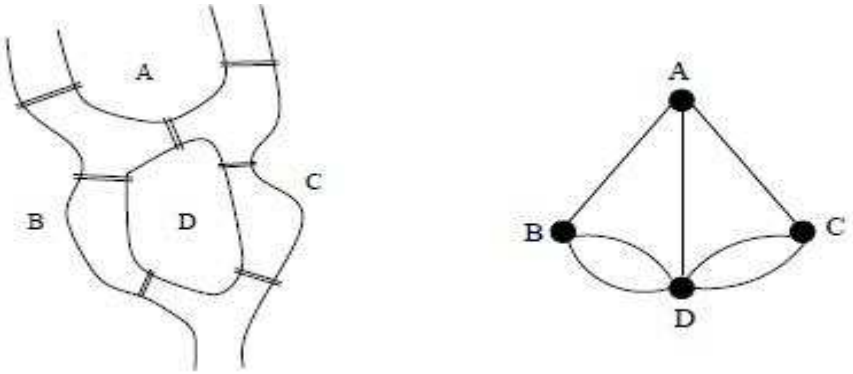

FIG. 1.2: Les sept ponts de Königsberg.

Cela revient au problème suivant : à partir d'une terre quelconque A, B, C, ou D voir figure 1.2, traverser chacun des ponts une fois et une seule et revenir à son point de départ, sans traverser la rivière à la nage !

Euler représenta cette situation à l'aide d'un dessin où les points A, B, C, D représentaient les terres et les lignes qui les reliaient, les ponts, comme le montre la figure 1.2. Le problème des ponts de Königsberg est identique à celui consistant à tracer une figure

géométrique sans lever le crayon en passant par tous les traits et sans repasser plusieurs fois sur un même trait. *Euler* démontra que le problème des *ponts de Königsberg* n'a pas de solution.

Pendant les cent années qui suivirent, rien ne fut fait dans ce domaine de recherche. Puis, en 1847, *Kirchhoff* $(1824-1887)$ développa la théorie des arbres pour l'appliquer à l'analyse des circuits électriques. Dix ans plus tard, *Cayley* $(1821-1895)$ s'intéressa aussi à la notion d'arbre alors qu'il essayait d'énumérer les isomères saturés des hydrocarbures de type C_nH_{2n+2}.

A cette époque, un autre problème d'importance pour la *théorie des graphes* fut également proposé et partiellement résolu. La *conjecture des quatre couleurs* affirmait que quatre couleurs suffisent pour colorier n'importe quelle carte plane telle que les « pays » ayant une frontière commune soient de couleurs différentes. C'est sans doute *Möbius* $(1790-1868)$ qui présenta le premier ce problème dans l'un de ses cours en 1840. Environ dix ans plus tard, *De Morgan* $(1806-1871)$ essaya de le résoudre. Les lettres de *De Morgan* à ses divers collègues mathématiciens constituent les premières références à la *conjecture des quatre couleurs*.

Le problème devint célèbre après sa publication, par *Cayley* en 1879, dans le premier volume des *Proceedings of the Royal Geographic Society*. Il est resté très longtemps sans solution. Il fallut attendre jusqu'en 1976 pour que *Appel* et *Haken* prouvent ce théorème en réduisant le problème à un nombre fini de situations particulières et en trouvant une solution pour chacune d'entre elles à l'aide d'un ordinateur.

Cette période fertile (autour de 1876) fut suivie d'un demi-siècle de relative inactivité. Les années 1920 virent la résurgence de l'intérêt pour les graphes. L'un des pionniers de cette période fut *König* à qui l'on doit le premier ouvrage consacré entièrement à la théorie des graphes (*König, 1936*). Il est sans doute à l'origine de l'utilisation du terme *graphe* pour désigner ce qui était préalablement considéré comme un ensemble de *points et de flèches*.

A partir de 1946, la théorie des graphes a connu un développement intense sous l'impulsion de chercheurs motivés par la résolution de problèmes concrets. Parmi ceux-ci, citons de manière privilégiée *Kuhn* pour ses travaux en 1955, ainsi que *Ford* et *Fulkerson* en 1956 et *Roy* en 1959.

Parallèlement, un important effort de synthèse a été opéré en particulier par *Claude Berge*. Son ouvrage *Théorie des graphes et ses applications* publié en 1958 [11] marque sans doute l'avènement de l'ère moderne de la *théorie des graphes* par l'introduction d'une *théorie des graphes* unifiée et abstraite rassemblant de nombreux résultats épars dans la littérature.

Depuis, cette théorie a pris sa place, en connaissant de très nombreux développements essentiellement dûs à l'apparition des calculateurs, au sein d'un ensemble plus vaste d'outils et de méthodes généralement regroupées sous les appellations *recherche opérationnelle* et *mathématiques discrètes*.

La *théorie des graphes* s'est aussi développée au sein de disciplines diverses telles que la

chimie (modélisation de structures), la biologie (génome), les sciences sociales (modélisation des relations dans les réseaux sociaux) ou en vue d'applications industrielles (problème du voyageur de commerce). Elle constitue l'un des instruments les plus courants et les plus efficaces pour résoudre des problèmes discrets posés en *RO*.

De manière générale, un graphe permet de représenter simplement la structure, les connexions, les cheminements possibles d'un ensemble complexe comprenant un grand nombre de situations, en exprimant les relations entre ses éléments (par exemple, réseau de communication, réseaux ferroviaire ou routier, arbre généalogique, diagramme de dépendance de tâches, etc.).

1.2.2.2 Qu'est-ce qu'un graphe ?

Définition 1. *Un **graphe** est un couple $G = (V, E)$, où V est un ensemble fini, dont les éléments sont appelés **sommets** du graphe et E un ensemble des paires de sommets, appelées **arêtes**.*

Graphiquement, les sommets peuvent être représentés par des points et les arêtes par des traits reliant ces points.

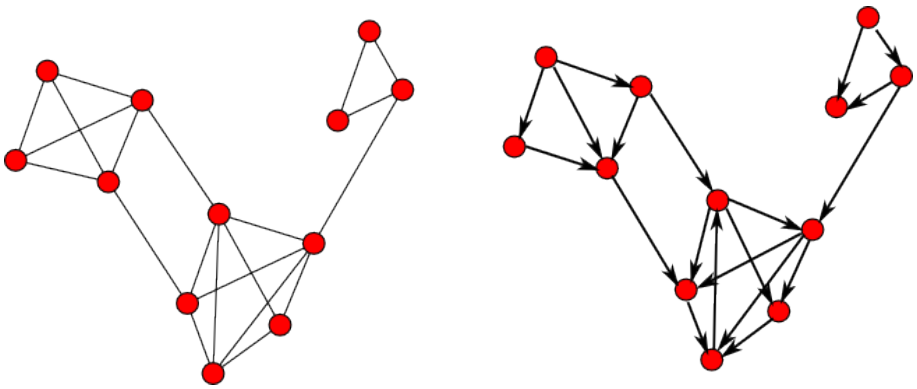

FIG. 1.3: Représentation graphique de graphe non orienté et orienté.

On distingue les graphes orientés des graphes non orientés. On peut définir un graphe orienté comme suit :

Définition 2. *Un **graphe orienté** est un couple $\overrightarrow{G} = (V, A)$, où V est un ensemble fini, dont les éléments sont appelés **sommets** et A est un ensemble de couples de sommets appelés **arcs**.*

La différence entre G et $\overrightarrow{G}$ est dans la notion de paires de sommets, et de couples de sommets. Dans un couple, noté (i, j), l'ordre des éléments est pris en compte, alors que ce n'est pas le cas dans la paire, notée $\{i, j\}$. Par exemple, $\{2, 0\} = \{0, 2\}$ mais $(2, 0) \neq (0, 2)$.

Graphiquement, dans le cas orienté, les sommets peuvent être représentés par des points et les arcs par des flèches reliant les sommets. La figure 1.3 illustre la représentation graphique d'un graphe non orienté ainsi que celle d'un graphe orienté.

Plusieurs problèmes découlent de la théorie des graphes comme les problèmes de plus courts chemins, couplage, flot maximum et coupe minimum, etc. Plus de détails sur ces problèmes sont disponibles dans les ouvrages [25] ou [53].

1.2.3 La théorie de la complexité

L'expérience montre que certains problèmes algorithmiques sont plus faciles que d'autres à résoudre, dans le sens où la meilleure solution peut être obtenue rapidement. La *théorie de la complexité* a été développée pour permettre de classer mathématiquement les problèmes en plusieurs classes de difficulté. Les principales qui nous intéressent ici, sont les classes $\mathcal{P}$ et $\mathcal{NP}$.

Nous exposons dans cette partie les grands principes de la *théorie de la complexité*. Le lecteur intéressé par de plus amples informations pourra consulter différents ouvrages dédiés à la complexité dont les livres de *Garey et Johnson* [26] ou *Papadimitriou* [52] qui sont les livres de références dans le sujet et plus récemment celui de *Goldreich* [31].

1.2.3.1 Les classes $\mathcal{P}$ et $\mathcal{NP}$

Pour pouvoir exposer la notion de classes de problèmes, il est tout d'abord nécessaire de distinguer les problèmes de décision des problèmes d'optimisation.

Un *problème de décision* est un énoncé auquel la réponse est « oui » ou « non ». Il est possible d'associer à chaque *problème d'optimisation*, un problème de décision en introduisant un seuil k correspondant à la fonction objectif f. Le problème de décision devient : Existe-t-il une solution réalisable S telle que $f(S) \leq k$? pour une fonction f à minimiser.

Il est alors possible de définir la classe $\mathcal{P}$ qui regroupe les problèmes de décision résolvables par des algorithmes polynomiaux.

Un *algorithme polynomial* est un algorithme dont le temps d'exécution est borné par $O(p(x))$ où p est un polynôme et x est la taille de l'entrée (le nombre de bits nécessaires pour stocker une instance du problème dans un ordinateur). Les algorithmes dont la complexité ne peut pas être bornée polynomialement sont couramment qualifiés d'exponentiels.

La classe $\mathcal{NP}$ regroupe les problèmes qui peuvent être résolus en temps polynomial par un algorithme non déterministe (algorithme qui comporte des instructions de choix). Pour ces algorithmes, si à chaque instruction le bon choix est effectué, le temps de calcul est polynomial. Si au contraire tous les choix sont énumérés, l'algorithme devient déterministe et son temps de calcul devient exponentiel.

Les algorithmes "ordinaires" sont évidemment des cas particuliers des algorithmes non déterministes. Aussi, tout problème de décision qui peut être résolu par un algorithme polynomial, donc appartenant à la classe $\mathcal{P}$, appartient également à la classe $\mathcal{NP}$, d'où $\mathcal{P} \subseteq \mathcal{NP}$.

Parmi les problèmes de la classe $\mathcal{NP}$, on a identifié une classe de problèmes difficiles dans la mesure où la communauté scientifique s'accorde à dire qu'il n'existe probablement pas d'algorithmes polynomiaux pour ces problèmes. Ce sont les problèmes $\mathcal{NP}$-Complets. Au sein de cette classe, tous les problèmes sont de difficulté équivalente. En particulier, s'il existe un algorithme polynomial pour résoudre un seul de ces problèmes, alors il en existe pour chaque problème de la classe $\mathcal{NP}$.

Afin de décrire cette classe d'équivalence, définissons tout d'abord la *réduction polynomiale* entre deux problèmes. Soient P_1 et P_2, deux problèmes de décision. On dit que P_1 se réduit polynomialement à P_2 (et on note $P_1 \alpha P_2$) s'il existe un algorithme de résolution de P_1, qui fait appel à un algorithme de résolution de P_2, et qui est polynomial lorsque la résolution de P_2 est comptabilisée comme une opération élémentaire. On dit alors d'un problème de décision qu'il est $\mathcal{NP}$-Complet si tout problème de la classe $\mathcal{NP}$ se réduit polynomialement à lui. Les problèmes d'optimisation combinatoire dont le problème de décision associé est $\mathcal{NP}$-Complet sont qualifiés de $\mathcal{NP}$-Difficiles. En d'autres termes, tout problème algorithmique auquel on peut réduire polynomialement tout problème de $\mathcal{NP}$ est qualifié de $\mathcal{NP}$-Difficile. Une des grandes questions ouvertes de l'informatique théorique est : est ce que $\mathcal{P}$est égal à $\mathcal{NP}$? On considère en général que $\mathcal{P} \neq \mathcal{NP}$.

1.2.3.2 Prouver la $\mathcal{NP}-$complétude d'un problème

La démonstration d'appartenance d'un problème à la classe des problèmes $\mathcal{NP}$-Complets est très importante. En effet, une fois cette démonstration faite, on peut considérer comme peu vraisemblable l'existence d'un algorithme polynomial pour résoudre ce problème de manière exacte. Prouver qu'un problème de décision P est $\mathcal{NP}$-Complet consiste en quatre étapes :

1. Montrer que P est dans $\mathcal{NP}$.
2. Choisir $P^{'}$ un problème $\mathcal{NP}$-Complet connu.
3. Construire une transformation f de $P^{'} \longrightarrow P$, polynomiale.
4. Prouver que f est bien une réduction polynomiale.

Ainsi il est nécessaire de connaître des problèmes classiques, dont on sait qu'ils appartiennent à la classe $\mathcal{NP}$-Complet. Le premier problème qui a été prouvé être $\mathcal{NP}$-Complet par *Cook* [26], est le problème de *satisfaisabilité* SAT qui peut être énoncé comme suit:

SAT : Étant donnée une expression booléenne sous forme de conjonctions de disjonctions, le problème est dit satisfaisable s'il existe une affectation en 0 et 1 de ses variables de manière à ce que l'expression booléenne prenne la valeur 1.

1.2.3.3 Quelques problèmes $\mathcal{NP}$-Complets

Nous présentons ici quelques problèmes $\mathcal{NP}$-Complets dont la complétude a pu être démontrée grâce à l'existence d'un premier problème $\mathcal{NP}$-Complet. Ce sont les six problèmes de base exposés par *Garey* et *Johnson* [26], qui détaillent ensuite la littérature concernant la complexité des problèmes. Cet ouvrage de référence [26] liste aussi d'autres problèmes connus pour être $\mathcal{NP}$-Complets. Une liste plus actualisée est disponible à l'adresse :

`http://www.csc.kth.se/~viggo/problemlist/`.

3-satisfaisabilité : Étant donné un ensemble de clauses de dimension 3, construites à partir d'un ensemble fini de ,existe-t-il une affectation de ces variables qui satisfait toutes les clauses ?

Couplage de dimension trois : Étant donné un ensemble M de triplets ($M \subseteq X \times Y \times Z$, où X, Y, Z sont disjoints et de même cardinalité q), existe-t-il $M' \subseteq M$ tel que $|M'| = q$ et les triplets de M' sont deux à deux disjoints ?

Recouvrement : Étant donné un graphe $G = (V, E)$ et un entier k, existe-t-il $X \subseteq V$ tel que $|X| \leq k$ et toute arête de G a au moins une de ses extrémités dans X ?

Clique : Étant donné un graphe $G = (V, E)$ et un entier k, existe-t-il $X \subseteq V$ tel que $|X| \geq k$ et tous les sommets de X sont deux à deux adjacents ?

Cycle hamiltonien : Étant donné un graphe $G = (V, E)$, existe-t-il un cycle passant une fois et une seule par chacun des sommets de V?

Partition : Étant donnés n entiers positifs $s_1, s_2, \ldots, s_n$, existe-t-il un sous ensemble $J \subseteq I = \{1, 2, \ldots, n\}$ tel que $\sum_{i \in J} s_i = \sum_{i \in I \setminus J} s_i$?

La figure 1.4 est un diagramme des transformations utilisées pour prouver la $\mathcal{NP}$-Complétude des six problèmes présentés.

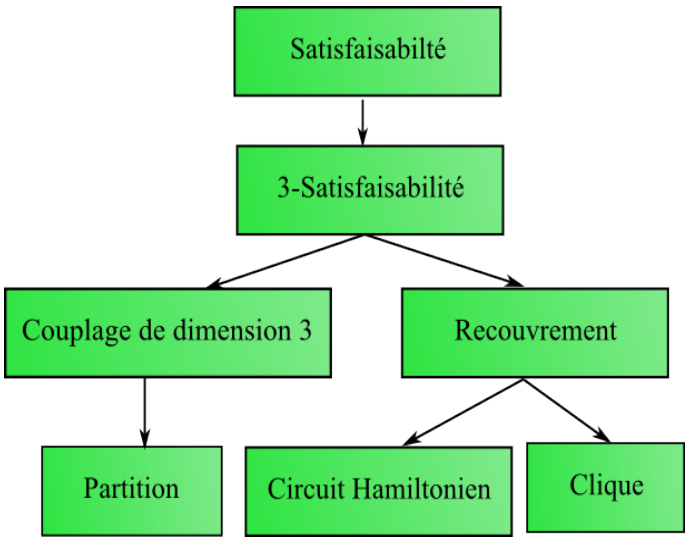

FIG. 1.4: Enchaînement des preuves de NP-Complétude.

1.3 Algorithmes et outils de résolution

Nous avons vu précédemment qu'il existe des problèmes de différentes complexités. Les problèmes appartenant à la classe $\mathcal{P}$ ont des algorithmes polynomiaux permettant de les résoudre de manière exacte. Pour les problèmes appartenant à la classe $\mathcal{NP}$-Complet, l'existence d'algorithmes polynomiaux semble peu réaliste. Il existe pour certains problèmes $\mathcal{NP}$-Difficiles des méthodes de résolution exactes si on a de petites instances par exemple ou encore beaucoup de temps calcul. Néanmoins, pour la plupart de ces problèmes il est rare de pouvoir les résoudre avec une méthode exacte, à cause du temps de calcul.

C'est pourquoi, des méthodes de résolution non exactes ou *heuristiques*, sont largement utilisées pour appréhender les problèmes $\mathcal{NP}$-Difficile. Nous exposons dans cette partie les grandes lignes des méthodes les plus connues classées en deux catégories : les méthodes exactes et les méthodes approchées ainsi que la résolution à partir d'une modélisation analytique. Le lecteur intéressé par de plus amples détails pourra se reporter aux références données dans les différents paragraphes.

1.3.1 Résolution exacte

Dans ce paragraphe, il est question de deux types de méthodes exactes : la programmation dynamique et la méthode dite de *séparation et évaluation*. Comme nous le verrons, le temps de calcul de ces méthodes est généralement exponentiel ce qui explique qu'elles ne soient utilisables que sur des problèmes de petite voire moyenne taille.

1.3.1.1 Programmation dynamique

Introduite par *Bellman* dans les années 50, *la programmation dynamique* décompose généralement un problème de taille n en un nombre fixe de problèmes de dimension $n-1$. Le système est alors constitué de n étapes que l'on résout séquentiellement. Le passage d'une étape à l'autre se faisant à partir des lois d'évolution du système et d'une décision.

Le principe d'optimalité de *Belleman* est basé sur l'existence d'une équation récursive permettant de décrire la valeur optimale du critère à une étape en fonction de sa valeur à l'étape précédente.

Pour les problèmes *NP-difficiles au sens faible*, c'est-à-dire pour lesquels il existe un algorithme de résolution *pseudo-polynomial*, il est souvent possible de construire un algorithme de programmation dynamique *pseudo-polynomial* (codage unaire des données), pouvant alors être utilisé pour des problèmes de taille raisonnable.

1.3.1.2 Modélisation analytique et résolution

La modélisation analytique d'un problème permet, non seulement de mettre en évidence l'objectif et les différentes contraintes du problème, mais également, parfois de le résoudre. L'idéal, est d'obtenir un programme linéaire dont les variables sont réelles. Dans ce cas, de nombreux solveurs peuvent le résoudre rapidement. Par contre, dès que le problème comporte des décisions nécessitant l'utilisation de variables entières, les modèles deviennent

plus difficiles à résoudre. Il en est de même lorsque le modèle n'est pas linéaire, mais quadratique par exemple.

Néanmoins, il est parfois surprenant de voir qu'un problème particulier de taille intéressante peut être appréhendé et résolu par la programmation mathématique. Il est donc justifié de commencer à étudier un problème en proposant une ou plusieurs modélisations analytiques. De plus, cette démarche a été simplifiée car il existe aujourd'hui plusieurs langages de modélisations, tels que *AMPL*, *GAMS*, *LINGO*, *MPL*, permettant d'écrire les programmes linéaires de façon formelle, proche de l'écriture mathématique. Ces langages de modélisation peuvent soit générer les fichiers lisibles par des solveurs, tels que *CPLEX*, *OSL*, *XPRESS*, soit être directement couplés à ces solveurs. Malheureusement, tous les problèmes ne peuvent être résolus par cette approche car la résolution de programmes linéaires en nombres entiers, par exemple, demande souvent beaucoup trop de temps de calcul.

1.3.1.3 Méthode de séparation et évaluation

La méthode de *séparation et évaluation* est basée sur une énumération implicite et intelligente de l'ensemble des solutions réalisables. Pour cela, la *séparation* consiste à décomposer le problème initial en plusieurs sous-problèmes qui sont à leur tour décomposables. Ce processus peut se visualiser sous forme d'un arbre d'énumération. Pour chaque sous-problème (nœud de l'arbre), la procédure d'*évaluation* calcule (dans le cas de problème de minimisation) une borne inférieure de la solution obtenue à partir des sous-problèmes. Au préalable, une borne supérieure de la solution optimale a été calculée, en relâchant certaines contraintes par exemple. Cette borne est utilisée pour éviter l'exploration de nœuds dont la valeur de la borne inférieure est supérieure à la valeur de la borne supérieure. La borne supérieure est réactualisée lorsqu'une solution réalisable d'une valeur inférieure est atteinte.
Ainsi, l'exploration de certaines branches de l'arbre n'est pas effectuée, on dit alors que ces branches sont *coupées*. Cela permet de ne pas énumérer réellement toutes les solutions. Il faut donc remarquer que l'efficacité d'un algorithme de séparation et d'évaluation est déterminée par la qualité des bornes utilisées et par de bonnes conditions de branchement. Il existe des méthodes pour les améliorer comme les *méthodes de coupes*, qui consistent à générer des contraintes pour faciliter les coupes et ainsi réduire le domaine de recherche des solutions.

1.3.2 Résolution approchée

Pour les problèmes de grande taille, les méthodes exactes ne sont pas envisageables de par le temps de calcul nécessaire. Il est dans ce cas possible d'utiliser des méthodes approximatives qui donnent des solutions *sous-optimales*, c'est-à-dire qu'elles ne fournissent pas la solution optimale, mais essayent de s'en approcher. On choisit alors ces méthodes pour leur rapidité d'exécution. Ces solutions peuvent ensuite servir de solutions initiales pour les méthodes amélioratrices (qui sont décrites plus bas).
Les algorithmes de résolution approchée sont : les *heuristiques* et *méta-heuristiques*. Une

heuristique est un algorithme qui ne donne pas toujours la solution exacte. Les *méta-heuristiques* sont des méthodes qui permettent de créer des *heuristiques*. Nous exposons les principes de chaque méthode dans les paragraphes suivant. Le lecteur intéressé par plus de détails pourra consulter les ouvrages consacrés entièrement à ces sujets. On peut citer par exemple le livre de *Dréo et al.* [21], et celui de *Talbi* [55].

1.3.2.1 Les algorithmes d'approximation

Pour résoudre les problèmes d'optimisation $\mathcal{NP}$-Difficiles on a très peu de chances de trouver des algorithmes efficaces. Par exemple, pour les problèmes d'optimisation $\mathcal{NP}$-Complets, il est souvent impossible de proposer des algorithmes exacts suffisamment efficaces pour résoudre les instances de grande taille. Or la plupart des problèmes le sont : en biologie, en réseau, en électronique. Une immense majorité de ces problèmes sont $\mathcal{NP}$-Difficiles. On se tourne donc vers les algorithmes d'approximation ou encore vers des heuristiques.

Dans cette thèse, la majorité des problèmes abordés sont $\mathcal{NP}$-Complets et les instances considérées de grande taille. Nous nous sommes donc intéresses à des méthodes heuristiques.

On distingue les heuristiques avec garantie de performance des heuristiques sans garantie de performance.

La performance des *heuristiques* est généralement calculée par le rapport entre la valeur de la solution calculée par l'heuristique et la valeur de la solution optimale : $R_a(I) = A(I)/OPT(I)$ dans le pire des cas pour toute instance I (étude théorique). D'autres analyses de performance peuvent être menées. Par exemple, l'analyse en moyenne regarde le comportement moyen de l'heuristique en calculant $R_a(I)$, non pas seulement pour le pire des cas, mais en moyenne. On dit alors qu'on a une *garantie de performance*.

De plus lorsque la solution optimale n'est pas calculable, il est également possible d'étudier expérimentalement le comportement de l'heuristique par exemple en comparant ses performances pour plusieurs instances, soit choisies pour des particularités ou prises au hasard, ou encore avec les performances d'autres heuristiques, soit avec des bornes inférieures de la solution optimale. Une telle étude ne permet pas d'obtenir une garantie de performance.

1.3.2.2 Quelques principes classiques des heuristiques

Les principes généraux et classiques qu'on retrouve dans beaucoup d'heuristiques sont décrits dans les paragraphes qui suivent.

Méthodes heuristiques constructives et algorithmes gloutons Les méthodes par construction progressive sont des méthodes itératives où à chaque itération, une solution partielle est complétée. La plupart de ces méthodes sont des *algorithmes gloutons* car elles considèrent les éléments (processus ou tâches, suivant les cas) dans un certain ordre sans jamais remettre en question un choix une fois qu'il a été effectué. De principe très simple parfois, ces algorithmes permettent souvent de trouver une solution rapidement. Ils sont couramment utilisés pour fournir rapidement des solutions initiales.

Méthodes heuristiques amélioratrices Il est d'abord nécessaire de définir la notion de *voisinage* et de *recherche locale*. Le voisinage défini pour une solution s de S est un ensemble $N(s) \subset S$ appelé ensemble de solutions voisines de s. Effectuer une recherche locale revient donc à chercher une solution meilleure que la solution courante dans son voisinage. Pour passer d'une solution à son voisin on parle de mouvements locaux.

Par exemple, prenons le célèbre problème du voyageur de commerce (*TSP* pour *Traveling Salesman Problem*) [2] : si on part d'une solution initiale quelconque comme l'ordre selon lequel les sommets sont numérotés. Il est bien évident qu'une telle solution sera de mauvaise qualité ; son intérêt est cependant de produire une solution initiale qu'on cherchera à améliorer ensuite par une procédure de recherche locale par exemple. On part alors du premier sommet et on cherche le sommet le plus proche de celui ci par exemple le sommet 3. On cherche ensuite le sommet de plus proche de 3 etc. Ainsi, on construit une solution avec la méthode du plus *proche voisin* basée sur la notion de *recherche locale*.
Nous exposons dans ce paragraphe, les méthodes d'améliorations locales les plus connues. Ces méthodes sont initialisées par une solution réalisable, calculée soit aléatoirement soit à l'aide d'une des heuristiques constructives exposées précédemment, et recherche à chaque itération une amélioration de la solution courante par des modifications locales. Cet examen se poursuit jusqu'à ce qu'un critère d'arrêt soit satisfait. L'utilisation de ces heuristiques itératives suppose que l'on puisse définir, pour toute solution S, un voisinage de solution, $N(S)$, contenant les solutions voisines (proches dans un certains sens). En général, le voisinage d'une solution est généré en appliquant, plusieurs fois de façon différentes, à cette solution, une petite transformation (échange, par exemple). A chaque solution S, nous associons le coût de cette solution $c(S)$ que l'on cherche à améliorer à chaque itération. Dans la famille des heuristiques amélioratrices, on peut citer par exemple la méthode de descente décrite ci-dessous.

Méthode de descente C'est l'une des heuristiques de recherche locale les plus simples. Elle consiste à rechercher dans le voisinage de la solution courante, une solution de coût plus faible. Elle procède ainsi jusqu'à arriver à un optimum local.
Cette méthode a l'avantage d'être rapide mais s'arrête dès qu'un optimum local est atteint, même si celui-ci n'est pas de bonne qualité.

Parmi ces méthodes, nous décrivons ici les *méthodes d'échanges* de type $r-opt$. Ces méthodes d'optimisation ont été initialement proposées par [44] pour résoudre le problème du voyageur de commerce (comme nous le verrons dans le chapitre 4), mais elles s'appliquent également à tout problème combinatoire dont la solution consiste en une permutation de r composantes parmi n pour construire une nouvelle solution.

La méthode consiste donc à sélectionner r composantes et à regarder si en interchangeant ces composantes, on obtient une meilleure solution. Nous remarquons donc qu'une solution n-optimale est une solution optimale (dans le cas d'un problème de taille n). Ainsi, plus r augmente, plus on se rapproche de la solution optimale, mais plus les calculs sont

[2] Problème du voyageur de commerce : étant donné un graphe complet non orienté à n sommets, chaque arête est munie d'un coût, trouver un cycle qui passe par tous les sommets une et une seule fois qui soit de longueur minimum.

difficiles. En effet, il y a C_n^r façons de choisir r composantes parmi n et $r!$ façons de les échanger, alors le temps de calcul est de (le nombre d'itérations). En pratique, on se limite à $r = 2$ ou 3.

1.3.2.3 Les méta-heuristiques

Tandis que les heuristiques classiques choisissent souvent la prochaine solution courante en se basant sur les informations locales, la plupart des *méta-heuristiques* se distinguent par la collecte d'informations au cours de l'exécution et dirigent la recherche vers l'optimisation globale. En effet, les méthodes *méta-heuristiques* ont besoin de plus de mémoire que les méthodes heuristiques pour sauvegarder les informations. De plus, de nombreuses méta-heuristiques permettent d'accepter des solutions qui dégradent la solution courante (c'est-à-dire que la prochaine solution soit moins bonne que la courante).

Les *méta-heuristiques* sont souvent inspirées par des systèmes naturels. Elles sont d'inspiration diverses comme : recuit simulé en physique, algorithmes génétiques en biologie de l'évolution ou algorithmes de colonies de fourmis en éthologie. Elles sont en général de complexité exponentielle, tout en restant plus rapide que les méthodes exactes.

Les *méta-heuristiques* se divisent en deux classes : celles qui se basent sur une unique solution courante et celles qui génèrent une population de solutions. Les *méta-heuristiques* de la première catégorie *intensifient* la recherche dans une zone donnée de l'espace des solutions. Les *méta-heuristiques* à population quand à elles permettent d'explorer des régions différentes de l'espace des solutions. On parle de *diversification* de la recherche. Dans la pratique, il n'est pas rare de coupler les deux stratégies en combinant une *méta-heuristiques* à population et une *méta-heuristiques* à solution unique.

Dans cette partie, nous donnons un exemple de chaque type de *méta-heuristique* : la recherche tabou qui n'a qu'une solution courante et les algorithmes génétiques qui utilisent une population de solutions.

Recherche Tabou
Cette méthode dont l'origine remonte à 1977, a été formalisée plus tard, en 1986, par *Glover* [28]. Elle n'a aucun caractère stochastique et utilise la notion de *mémoire de recherche* pour éviter de tomber dans un optimum local.

Le principe de l'algorithme est le suivant : à chaque itération le voisinage (complet ou un sous-ensemble du voisinage) de la solution courante est examiné et la solution minimisant l'augmentation de coût est sélectionnée si elle existe. Pour éviter les phénomènes de cyclage, on s'interdit de revisiter une solution récemment visitée. Pour cela, une liste dite *tabou* contenant les dernières solutions visitées est tenue à jour. Chaque nouvelle solution considérée chasse de cette liste la solution la plus anciennement visitée ou une liste de mouvements. La longueur de la liste, paramètre à définir, détermine donc le nombre d'itérations pendant lesquelles une solution ayant été visitée ne pourra être revisitée. Ainsi, la recherche de la solution courante suivante se fait dans le voisinage de la solution courante actuelle

sans considérer les solutions appartenant à la liste tabou. Tout au long de l'algorithme, la meilleure solution doit être conservée car l'arrêt se fait rarement sur la meilleure solution. En effet, l'arrêt de cette méthode peut se faire soit après un certain nombre d'itérations, soit lorsqu'il n'y a pas eu d'amélioration de la meilleure solution depuis un certain nombre d'itérations. Des versions plus élaborées permettant des recherches plus efficaces, ont été proposées par la suite [29].

Cette méthode donne de très bons résultats pratiques, malgré l'inexistence de résultats théoriques garantissant la convergence de l'algorithme vers la solution optimale.

Algorithmes génétiques

Cette classe de méthodes est basée sur une imitation des phénomènes d'adaptation des êtres vivants. L'application de ces méthodes aux problèmes d'optimisation a été formalisée par *Goldberg* en 1989 [30].

Les algorithmes génétiques fonctionnent sur une analogie avec la reproduction des êtres vivants. On part d'une population (ensemble de solutions) initiale sur laquelle les opérations de *reproduction*, de *croisement* ou de *mutation* vont être définis dans l'objectif d'exploiter au mieux les caractéristiques et propriétés de cette population. Ces opérations doivent mener à une amélioration (en termes de qualité de solutions) de l'ensemble de la population puisque les bonnes solutions sont encouragées à échanger par croisement leurs caractéristiques et à engendrer des solutions encore meilleures. Toutefois, des solutions de très mauvaise qualité peuvent aussi apparaître et permettent d'éviter de tomber très rapidement dans un optimum local. Parmi les paramètres de ces méta-heuristiques, on trouve : le codage des solutions, la taille de la population, les coefficients de reproduction, la probabilité de mutation (chaque génération correspond à une mutation). Les difficultés pour appliquer les algorithmes génétiques résident dans le choix et le réglage de ces paramètres.

Dans le chapitre 6, nous utilisons les algorithmes génétiques pour résoudre un problème pratique d'optimisation du partages de blocks *BIST* pour optimiser la conception en vue du test des mémoires. Les paramètres de cette implémentation sont donnés en détails de ce même chapitre.

Critéres de choix des heuristiques et méta-heuristiques

Le problème majeur quand on souhaite utiliser une *heuristique* ou une *méta-heuristique* est le choix de l'ensemble des paramètres. Le choix de l'*heuristique* utilisée est souvent guidé par la nécessité de trouver un bon compromis entre temps de calcul et qualité des solutions obtenues.

En effet, les heuristiques les plus simples comme les méthodes constructives sont en général polynomiales mais fournissent des solutions de qualité bien moindre qu'un algorithme génétique par exemple si le problème est $\mathcal{NP}$-Difficile et la taille de l'instance importante.

Cependant, le choix de l'ensemble des paramètres d'une *méta-heuristique* a une influence importante sur la qualité des solutions et sur leur durée d'obtention. Ainsi, il est nécessaire de faire un bon réglage des paramètres, par exemple bien choisir combien de générations doivent se succéder durant l'exécution d'un algorithme génétique.

Ces méthodes demandent aussi des efforts de calcul très importants, leur complexité n'étant pas polynomiale.

Le compromis temps calcul/qualité de la solution nous pousse souvent à choisir une méta-heuristique plutôt qu'un algorithme glouton. En effet, on a un temps de calcul raisonnable pour la qualité de la solution qu'elle donne.

Peu de résultats théoriques existent sur les *méta-heuristiques*. Néanmoins, sur le plan pratique de nombreuses expérimentations montrent de bon résultats empiriques sur des problèmes d'optimisation difficiles.

1.4 Conclusion

Ce premier chapitre présente une boîte à outils de méthodes de modélisation et de résolution de problèmes utilisées dans le reste de la thèse. En effet, les techniques mises en œuvre pour la résolution des problèmes posés par nos partenaires industriels sont issues de la recherche opérationnelle et de la théorie des graphes comme nous le verrons dans les chapitres suivants.

Nous avons d'abord présenté la *RO*, ses domaines d'application ainsi que ses outils et cadres pour la modélisation comme la programmation linéaire, la théorie des graphes, et celle de la complexité. Ensuite, nous avons exposé les méthodes de résolution exactes et approchées les plus connues. Plusieurs références d'ouvrages et d'articles classiques de la *RO* ont également été présentés.

Nous possédons à présent les outils nécessaires pour traiter les problèmes abordés dans ce travail. Pour en savoir davantage sur le domaine de la *RO*, le lecteur pourra aussi se référer à différents ouvrages et sites *web* sur lesquels se trouvent des informations régulièrement mises à jour. On peut citer par exemple le site de *INFORMS*(*Institute for Operations Research and the Management Sciences*): `http://www2.informs.org/Resources/`. Il regroupe les principales évolutions mondiales en *RO*.

Le site de la société savante *ROADEF* (*Recherche Opérationnelle et Aide à la Décision Française*): `http://www.roadef.org/content/index.htm`, regroupe l'ensemble des travaux de recherche dans les différentes thématiques de la communauté ainsi que les nouveaux ouvrages parus dans le domaine et les appels à conférences. Il propose aussi des problématiques industrielles posées à la communauté comme les célèbres *Challenges de la Roadef* !

Chapitre 2

Conception et test des circuits intégrés

Sommaire

2.1 Introduction 23
2.2 Vue d'ensemble et définitions 24
2.2.1 Eléments de base d'un circuit intégré 24
2.2.2 Circuits intégrés, un peu d'histoire 27
2.2.3 La loi de Moore 28
2.3 Flot de conception et de fabrication des circuits intégrés . . . 30
2.3.1 Conception 30
2.3.2 Réalisation 33
2.3.3 Zoom sur la CAO 34
2.4 Zoom sur le test 35
2.4.1 Le test des CI et son importance 35
2.4.2 Les différents types de test 36
2.4.3 Test et conception en vue du test 37
2.4.4 Scan Path pour le test des CI 37
2.5 Les mémoires 38
2.5.1 Généralités sur les mémoires 38
2.5.2 Le test des mémoires 39
2.5.3 Le BIST mémoire 39
2.6 Conclusion 40

2.1 Introduction

Le flot de conception d'un *Circuit Intégré* (*CI*), c'est-à-dire la suite d'applications logicielles qui permet au concepteur d'un *CI* de passer de sa spécification à sa réalisation concrète, met en jeu à de nombreux stades des problématiques d'*optimisation*.

En effet, la réalisation d'un *CI* est une opération très coûteuse en temps et en argent et qui nécessite de grands moyens aussi bien humains qu'industriels. En plus des progrès technologiques, qui améliorent continuellement l'efficacité des techniques de fabrication des *CI*, comme la finesse de gravures sur silicium par exemple, l'augmentation en continu de la puissance de calcul des ordinateurs offre de grandes possibilités d'amélioration du processus de conception et fabrication d'un *CI*.

A une époque où la concurrence dans l'industrie de la *micro-électronique* est particulièrement féroce, le paramètre temps de mise sur le marché et la qualité du produit deviennent des enjeux majeurs. Le circuit doit répondre sans faille aux attentes du client en termes de fonctionnalité, rapidité, qualité, fiabilité et coût. Dans ce contexte économique exigeant, et compte tenu du niveau de complexité important atteint par les *CI*, le **test** est plus que jamais une donnée importante du problème de conception. Il doit être à la fois court, efficace et le moins coûteux possible.

Aujourd'hui, beaucoup de recherches fondamentales et appliquées ayant pour objectif l'amélioration de l'étape de test sont poursuivies de manières très actives tant au niveau universitaire qu'industriel [41], [58]. Alors, pour améliorer et faciliter le test des *CI*, différentes techniques de conception en vue du test dites *DFT* (pour *Design For Test*) sont couramment utilisées et cela très tôt dans le processus de conception. C'est sur l'optimisation de certaines de ces techniques que portent les travaux de recherche de cette thèse.

Avant de les décrire, nous allons introduire dans ce chapitre les notions fondamentales de conception et test des circuits intégrés. Tout d'abord nous verrons les notions de base des *CI* suivies du flot de conception des *CI* avec les détails de chaque étape. Ensuite, nous exposerons de manière générale les techniques et problématiques du test des *CI* ainsi que de la conception en vue du test. Enfin la dernière partie de ce chapitre est consacrée à une famille particulière de *CI* : les *mémoires*. Elles y seront décrites ainsi que les méthodes de test associées.

La rédaction des différentes parties de ce chapitre est très inspirée des livres classiques d'électronique de base et de test des circuits intégrés ainsi que certains polycopiés de cours comme : le livre de *Calvez* [16] et celui de *Landraut* [41], ainsi que les ouvrages de *Abramovici et al.* [1], et *Wang et al.* [58].

2.2 Vue d'ensemble et définitions

Cette section présente les briques de base nécessaires à la constitution d'un circuit intégré. Nous présentons ici les notions de transistors, portes, bascules, etc. Pour plus de détails, le lecteur est invité à consulter les différentes références bibliographiques utilisées pour la rédaction de cette partie, comme [17], et l'encyclopédie en ligne *Wikipédia*.

2.2.1 Eléments de base d'un circuit intégré

2.2.1.1 Transistor

En 1947, trois chercheurs *John Bardeen*, *Walter Brattain* et *Robert Shockley* découvrent le *transistor*. Leur invention s'appuie sur le fait qu'il est possible de contrôler sélectivement

le flux d'électricité dans le silicium en faisant en sorte que certaines zones soient conductrices et d'autres isolantes (d'où le terme *semi-conducteur*). Comparé au tube à vide, technologie auparavant dominante, le transistor se révéla plus fiable, moins consommateur d'énergie et susceptible de miniaturisation. C'est ce dernier point qui donne naissance à la *loi de Moore*, qu'on verra plus loin de cette partie.

Les premiers transistors étaient faits à la main dans des conditions rudimentaires si on les compare aux *salles blanches* [1] actuelles. Les rendements de circuits opérationnels étaient bas (de 20 à 30%) et les performances étaient très variables. Les progrès techniques ont donc concerné surtout la maîtrise du processus de production. Pendant les années 50 une nouvelle industrie des semi-conducteurs se crée, basée sur d'importants progrès technologiques. Nous en citons ici deux :

- *Le processus de diffusion*, qui consiste à diffuser des impuretés (*dopants*) directement sur la surface du semi-conducteur, ce qui permet d'éliminer le processus fastidieux d'ajout de diverses couches de matériaux isolants et conducteurs sur le substrat.
- *Des techniques photographiques*, qui ont permis de projeter le dessin de masques complexes sur le semi-conducteur de sorte que la diffusion ne se produise que sur les surfaces souhaitées.

Ces deux techniques permirent de passer de la production manuelle à la production industrielle de série avec une bien meilleure qualité. Elles permirent également l'invention du circuit intégré en 1958 par *Jack Kilby*.
Jean Hoerni leur ajouta en 1959 une troisième innovation essentielle. Il observa que les techniques de diffusion et de photographie permettaient de se libérer des complications du transistor conventionnel à trois dimensions en dessinant des transistors plans (*planars*). Il devenait possible de faire les connections électriques non plus à la main, mais en déposant un film par condensation de vapeur métallique sur les parties appropriées du semi-conducteur. *Fair child* produisit le premier transistor plan en 1959, et le premier circuit intégré utilisant cette technique en 1961. *Moore* a fait de l'invention du transistor plan en 1959 le point de départ de sa *loi de Moore*. Les progrès des processus de production et des techniques se sont ensuite poursuivis. Les méthodes photographiques sont devenues de plus en plus précises, notamment par l'emprunt de techniques de photolithographie initialement conçues pour l'imprimerie.

2.2.1.2 Portes logiques et algèbre de Boole

La forme la plus élémentaire de circuit est la *porte logique*. Son comportement est dit *binaire* car il est caractérisé par deux états : l'état 0, qui représente la valeur logique faux et l'état 1, qui représente la valeur logique vrai.

Les constructeurs utilisent dans certains cas une logique dite positive : l'état 1 correspond à une tension comprise entre 2 et 5 volts (niveau haut) et l'état 0 à une tension comprise entre 0 et 1 volt (niveau bas). Dans d'autres cas, ils utilisent une logique dite

[1] La salle blanche est une pièce ou une série de pièces où la concentration particulaire est maîtrisée afin de minimiser l'introduction, la génération, la rétention de particules à l'intérieur, généralement dans un but spécifique industriel ou de recherche. Des paramètres tels que la température, l'humidité et la pression relative sont également maintenus à un niveau précis.

négative où l'état 1 correspond au niveau bas, tandis que l'état 0 correspond au niveau haut.

La transmission n'est pas instantanée : le délai de traversée d'une porte correspond au temps de propagation des signaux de l'entrée vers la sortie (de l'ordre de quelques nanosecondes pour la porte la plus simple). La porte logique la plus simple est la porte *NON* qui réalise une inversion logique. Il existe différents types de portes comme le montre la figure 2.1[2].

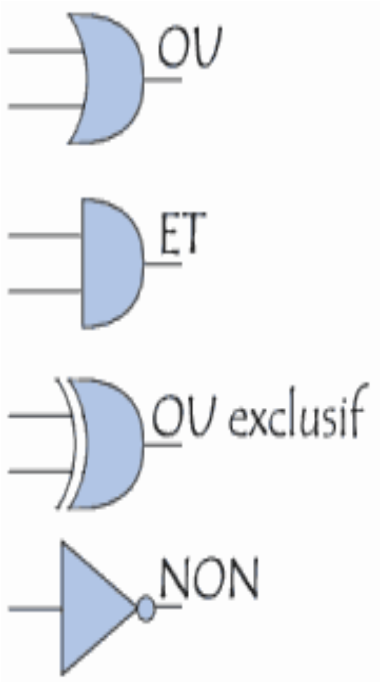

FIG. 2.1: Les différents types de portes.

La porte logique est construite à l'aide d'un *transistor*, selon deux techniques possibles : le transistor à jonctions (technologie dite bipolaire) ou le transistor à effet de champ (technologie dite unipolaire). Dans la première famille on trouve par exemple les circuits *RTL* (*Resistor Transistor Logic*), *TTL* (*Transistor Transistor Logic*), *DTL* (*Diode Transistor Logic*) ou encore *ECL* (*Emitter Coupled Logic*). La deuxième famille est celle des circuits *MOS* (*Metal Oxyde Semiconductor*) qui sont en général moins rapides, sauf les plus récents comme *HMOS* ou *XMOS3*.

On cherche souvent à réduire le nombre de portes, en utilisant les propriétés algébriques des opérations logiques. Un circuit intégré est une plaquette de silicium sur laquelle sont intégrées les portes du circuit. La plaquette est encapsulée dans un boîtier avec sur les côtés des broches permettant les connexions électriques appelées entrées et sorties primaires du circuit.

[2] *portes : source wikipédia.*

2.2.1.3 Les circuits combinatoires

On appelle circuit combinatoire ou circuit logique un ensemble de portes logiques reliées entre elles pour répondre à une expression algébrique. On peut citer par exemple les multiplexeurs ou encore les décodeurs :

- Un multiplexeur comporte 2^n entrées, 1 sortie et n lignes de sélection. La configuration des n lignes de sélection fournit une valeur parmi les 2^n entrées et connecte cette entrée à la sortie.
- Un décodeur comprend n entrées et 2^n sorties, la sortie activée correspondant à la configuration binaire du mot formé par les n entrées. Un tel circuit sert à sélectionner des adresses de la mémoire.

2.2.1.4 Circuits logiques à mémoire : les bascules

Appelés également *circuits séquentiels* ou *flip-flops*, les bascules mémorisent l'état antérieur des variables de sortie et permettent ainsi de mémoriser un bit. Il existe plusieurs types comme : Bascules D, Bascules JK, Bascules RS, etc. La figure 2.2 [3] illustre une bascule de type D.

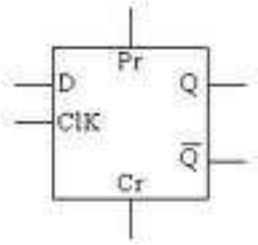

FIG. 2.2: Exemple d'une bascule D.

2.2.2 Circuits intégrés, un peu d'histoire

Jack Kilby (1923-2005) est l'inventeur du *circuit intégré*. En 1958, cet américain, alors employé par *Texas Instruments*, créait le tout premier *circuit intégré*, jetant ainsi les bases du matériel informatique moderne. Pour la petite histoire, *Jack Kilby*, qui venait de rejoindre l'entreprise, a fait cette découverte tandis que la majorité de ses collègues profitaient de vacances organisées par *Texas Instruments*. À l'époque, *Kilby* avait tout simplement relié entre eux différents transistors en les câblant à la main. Il ne faudra ensuite que quelques mois pour passer du stade de prototype à la production de masse de puces en silicium contenant plusieurs transistors. Ces ensembles de transistors interconnectés en circuits microscopiques dans un même bloc, permettaient la réalisation de mémoires, mais aussi d'unités logiques et arithmétiques.

[3] *figure : source polycopié de cours ensimag, 1ére année, conception de circuit, Catherine Bellon.*

Ce concept révolutionnaire concentrait dans un volume incroyablement réduit, un maximum de fonctions logiques, auxquelles l'extérieur accédait à travers des connexions réparties à la périphérie du circuit. Cette découverte a valu à *Kilby* un *prix Nobel* de physique en 2000, tandis qu'il siégeait toujours au directoire de Texas Instruments et détenait plus de 60 brevets à son nom.

Plus formellement, le *circuit intégré* (*CI*), aussi nommé *puce*, est un composant reproduisant une ou plusieurs fonctions plus ou moins complexes, intégrant fréquemment plusieurs types de composants électroniques de base dans un volume réduit, rendant le circuit facile à mettre en œuvre. Il existe une très grande variété de ces composants divisés en deux grandes catégories : analogique et numérique. La différence est que l'information n'est pas codée de la même façon dans les circuits analogiques et numériques.

Circuit intégré analogique Les composants les plus simples peuvent être de simples transistors encapsulés les uns à côté des autres sans liaison entre eux, les plus complexes, des assemblages réunissant l'ensemble des fonctions requises pour un appareil dont il est l'unique composant. Par exemple, les amplificateurs opérationnels sont des représentants de moyenne complexité de cette grande famille où on retrouve aussi des composants réservés à l'électronique haute fréquence et de télécommunication. Un exemple de circuit analogique : l'ampli *op LM741* et ses nombreux cousins.

Circuit intégré numérique Les circuits intégrés numériques les plus simples sont des portes logiques (et, ou, non), les plus complexes sont les microprocesseurs et les plus denses sont les mémoires. On trouve de nombreux circuits intégrés dédiés à des applications spécifiques (*ASIC* pour *Application Specific Integrated Circuit*), surtout pour le traitement du signal (traitement d'image, compression vidéo, etc.) on parle alors de *DSP* (pour *Digital Signal Processor*). Une famille importante de circuits intégrés est celle des composants de logique programmable (FPGA, CPLD). Ces composants sont amenés à remplacer les portes logiques simples à cause de leur grande densité d'intégration.

2.2.3 La loi de Moore

En 1965, *Gordon E. Moore* (un des co-fondateurs d'Intel), préparant un graphique pour un exposé sur l'évolution des performances des mémoires, constata une tendance frappante : la capacité des *puces* avait doublé à peu près chaque année de 1959 à 1965. Il en déduisit une hypothèse : la puissance des ordinateurs croîtrait de façon exponentielle. Elle était hardie, puisqu'il ne disposait pour l'étayer que de 7 observations.

Moore publia cette découverte dans un article devenu célèbre de la revue *Electronics* : *Gordon E. Moore*, « *Cramming more components into integrated circuits* », *Electronics*, 19 avril 1965. Cet article a encouragé les chercheurs et industriels à anticiper sur la croissance des performances et à concevoir des systèmes utilisant une puissance très supérieure à celle disponible lors de leurs recherches et applications. Il a ainsi suscité une forte accélération de l'innovation.

La figure 2.3[4] propose une représentation graphique de la loi de Moore.

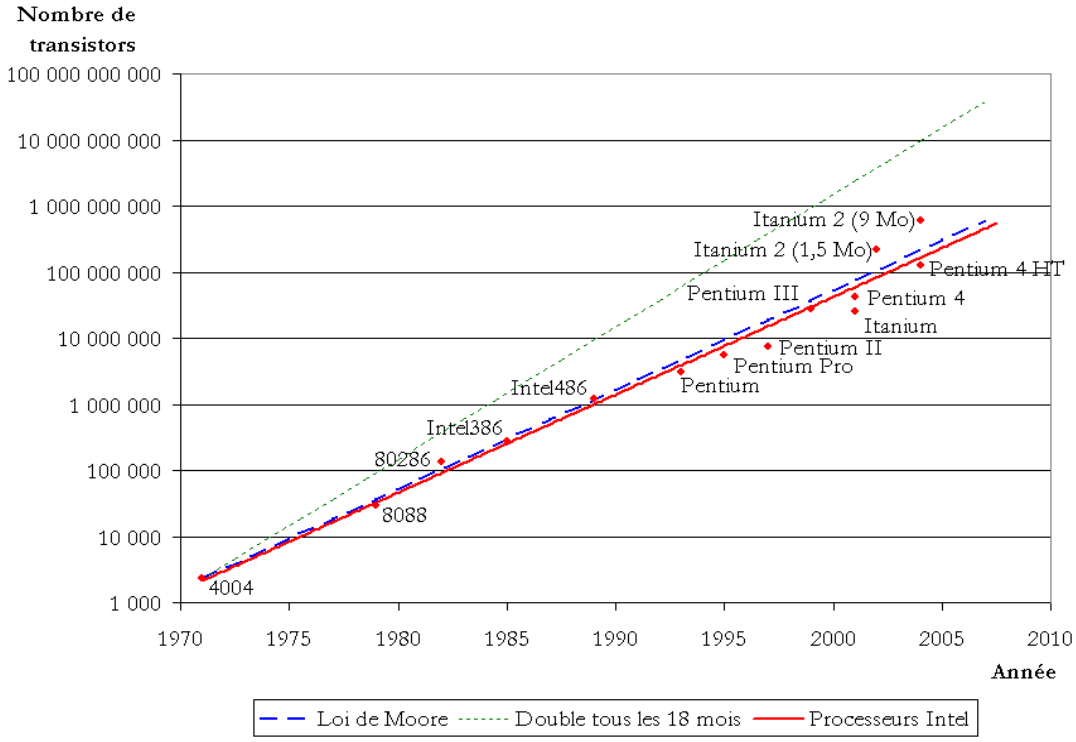

FIG. 2.3: Loi de Moore.

Le raisonnement de *Moore* est subtil. Il examine l'évolution de la fonction de coût des circuits intégrés, et considère la relation entre le coût moyen de production par composant et la complexité du circuit. Cette fonction est d'abord décroissante, puis croissante : il existe donc un niveau de complexité pour lequel le coût moyen d'un composant intégré sur le circuit est minimal. C'est ce niveau que des producteurs rationnels choisiront, car c'est celui qui permet le meilleur rapport efficacité/coût. Ensuite, *Moore* constate que ce niveau optimal de complexité est multiplié chaque année par deux. On remarque que la loi de *Moore* ne concerne pas le prix de ventes des *CI*. En 1975, *Moore* réévalua le rythme de croissance : désormais, disait-il, elle procèderait par doublement tous les 18 mois et non tous les ans ; néanmoins elle restait exponentielle. En 1995, *Moore* vérifia que la progression prévue avait bien été respectée !

Plus récemment, la crise économique a aussi freiné cette évolution dans le sens ou

[4] *Loi de Moore : source wikipédia.*

les usines de fabrication de circuit avec les nouveaux nœuds technologiques ont un coût prohibitif.

2.3 Flot de conception et de fabrication des circuits intégrés

La réalisation d'un circuit intégré est une opération très complexe. Elle passe par plusieurs étapes et nécessite de très grands investissements. Le résultat de chaque étape dépend de celle qui la précède.

On peut partager le processus de réalisation d'un circuit intégré en deux grandes phases qui sont illustrées dans la figure 2.4. Nous allons décrire les différentes étapes ainsi que leurs objectifs dans le paragraphe suivant.

2.3.1 Conception

2.3.1.1 Enoncé des besoins : le cahier des charges

Avant de commencer la conception d'un circuit intégré, l'utilisateur doit bien spécifier ses besoins. Ils proviennent en général de la fonction à laquelle le circuit intégré est destiné. On établit alors le cahier des charges c'est-à-dire la description faite par le demandeur du problème à résoudre. Il doit décrire et situer correctement le rôle que devra jouer le composant vis-à-vis de l'environnement dans lequel celui-ci va intervenir ainsi que l'utilisation qui en sera faite.

2.3.1.2 Ecriture des spécifications

Les spécifications définissent le circuit intégré selon une description interne complètement indépendante de l'environnement précisé dans le cahier des charges. Elles sont de différents types et couvrent la fonction à réaliser en terme d'entrés/sorties ainsi que des caractéristiques de diverses natures (opérationnelles ou physiques).

2.3.1.3 Description RTL

La description *RTL* (*Register Transfer Level*) est une modélisation du circuit au niveau transfert des registres. Cette modélisation revient à décrire l'implémentation du circuit sous forme d'éléments séquentiels (registres ou bascules) et de combinaisons logiques (porte ET, OU, NON, ect.) entre les différentes entrées et sorties primaires du circuit. Cette modélisation est codée à l'aide d'un langage de programmation standard comme le *VHDL*, ou le *VERILOG*.

Une description *RTL* est exploitable ou plus couramment dite synthétisable automatiquement en *portes* logiques combinatoires (portes ET, OU, multiplexeur...) et séquentielles (les bascules synchrones, etc.) issues d'une bibliothèque.

Pour cela, le codage *RTL* en *VERILOG* ou *VHDL* doit suivre certaines règles précises. Le code est alors dit « synthétisable ».

Aujourd'hui le niveau *RTL*, est le niveau d'abstraction le plus courant pour la description des spécifications.

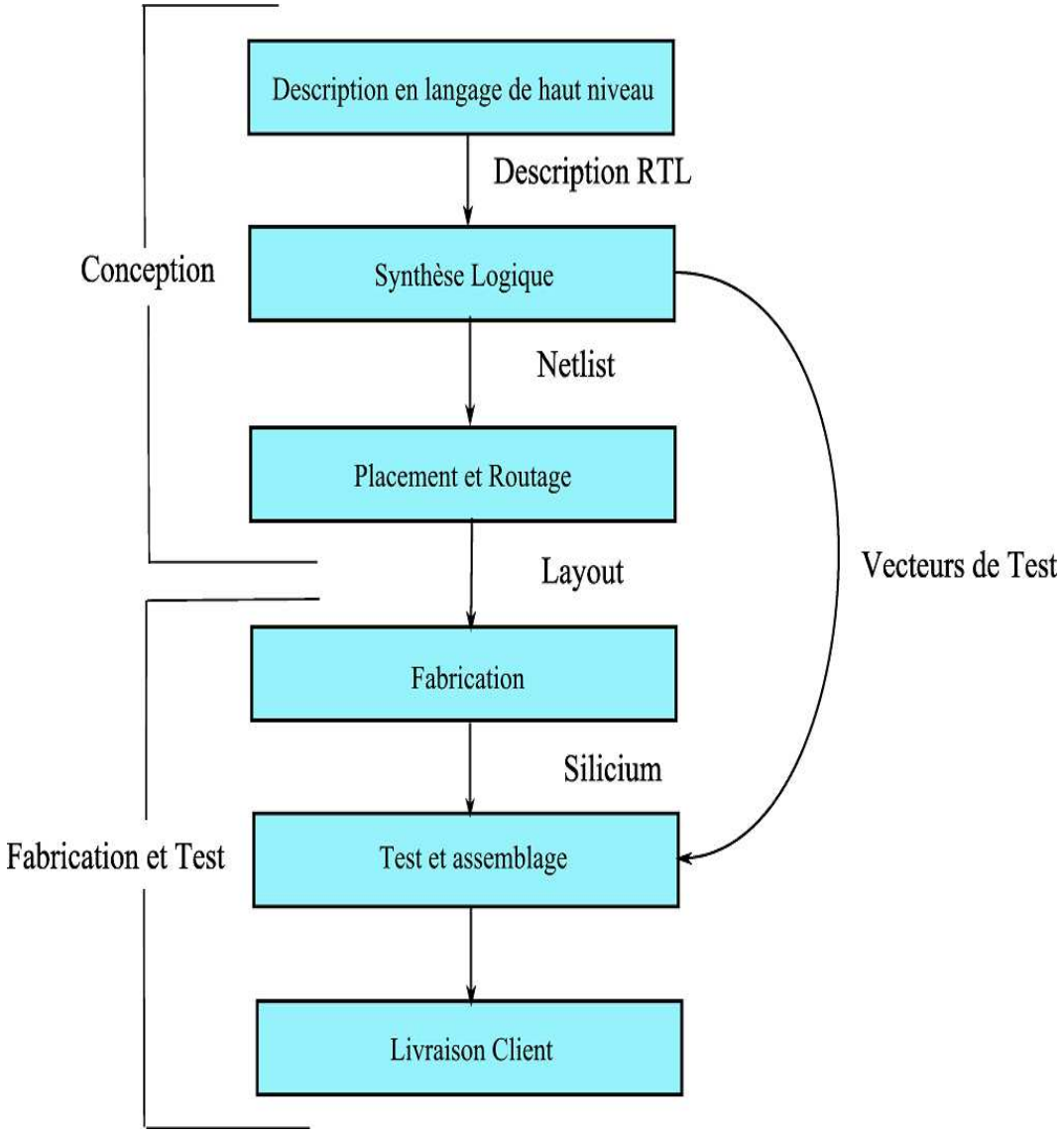

FIG. 2.4: Flot traditionnel de fabrication d'un circuit intégré

2.3.1.4 Synthèse logique

La synthèse logique permet de transformer une application logicielle en un composant électronique. En effet, la synthèse est l'étape qui permet de transcrire la description *RTL* du circuit en une description au niveau portes logiques (*Gate netlist* qui est une description de bas niveau d'un circuit basée sur des éléments de type porte logique). Au préalable, une bibliothèque cible de portes logiques doit être disponible. Celle-ci rassemble généralement plusieurs centaines d'éléments logiques (comme des portes ET, OU, BASCULES, etc). Cette bibliothèque dépend de la technologie cible, c'est-à-dire du nœud technologique (par exemple $90nm$ *cmos* ou $65nm$) et du fondeur (les règles de dessin des cellules dépendent du procédé de fabrication).

Le résultat de cette étape est un fichier informatique traduisant l'instanciation des portes de la bibliothèque cible et leurs interconnexions et représentant le circuit électronique (*gate netlist*). Des formats de fichiers spécifiques existent pour les *netlist*.

A cette étape, on génère aussi les *vecteurs de test* (2.3.2.2) avec des outils spécifiques appelés ATPG (*Automatic test Pattern Generation*) qui seront appliqués plus tard dans la phase de test sur silicium.

L'étape de synthèse, fait appel a beaucoup d'optimisation dans le sens où on cherche les meilleurs composants (qui prennent moins de place et qui ne consomment pas beaucoup) qui correspondent à la description demandée.

2.3.1.5 Placement et routage

Cette étape nous rapproche cette fois de ce que sera le circuit final. Il s'agit d'effectuer les deux opérations suivantes:

- Le placement : l'outil de placement prend la liste des cellules de base qui font la fonction du circuit et les répartit géographiquement sur l'empreinte.
- Le routage, n'a plus alors qu'à tirer les fils pour interconnecter les entrées et les sorties des cellules.

  Il obtient ainsi un masque appelé aussi *un layout*[5].

A cette étape, beaucoup de problématiques d'optimisation combinatoire apparaissent comme par exemple le placement optimal des composants afin de minimiser la surface totale ou encore la distribution d'arbre d'horloge pour les signaux afin de minimiser la longueur des interconnexions.

2.3.1.6 Vérification et validation

Avant de lancer le circuit en fabrication, il est nécessaire de valider (sur les premières tranches de silicium) l'ensemble des fonctions pour lesquelles il a été conçu ainsi que les différents paramètres électriques décris dans les spécifications. Cette étape permet d'identifier les défauts cachés. La validation fait appel en général à des techniques de test totalement différentes de celles utilisées au cours de la phase de production.

[5] Un layout est un dessin géométrique qui représente le niveau physique du circuit intégré.

2.3.2 Réalisation

2.3.2.1 Fabrication (Silicium)

Après avoir obtenu le *layout* du circuit, on passe à l'étape de fabrication sur silicium. La fabrication d'un circuit intégré à partir d'une galette de silicium est une suite d'étapes où des masques sont imprimés, des couches sont déposées et des traitements chimiques sont appliqués.

L'ensemble de ces opérations se fait dans des conditions très strictes de contrôle de l'environnement. En particulier, il est important de veiller à l'absence de poussières qui peuvent créer par la suite des défauts physiques. Ces derniers sont représentés dans le processus de conception par des modèles de fautes tels que *fautes de collages*[6].

2.3.2.2 Test

Enfin la dernière étape est le test. Une fois que les circuits sont fabriqués il faut les soumettre au test de production afin de déterminer les bons circuits qui seront transmis aux clients.

Il est bien sûr illusoire de penser que tous les circuits produits puissent être sans faille. Certains paramètres comme les impuretés des matériaux, les imperfections des procédés de fabrication, les pannes matérielles d'équipements, où encore les erreurs humaines peuvent entraîner le mauvais fonctionnement d'un circuit. En plus, des problèmes de maturité d'une technologie peuvent être aussi une cause de dysfonctionnement. En effet, plus une technologie s'affine avec le temps plus le procédé de fabrication est fiable.

La vraisemblance de l'occurrence de ce type de problèmes ainsi que leurs conséquences tant du point de vue technique, qu'économique ou humain est la principale raison de procéder à un test du circuit.

Le test d'un circuit permet donc de trier les circuits déclarés « bons » des circuits défaillants. L'échec d'un circuit intégré au test peut avoir différentes raisons :

- le processus de fabrication n'a pas été correct pour ce circuit,
- la conception n'est pas bonne,
- les spécifications présentent des problèmes,
- éventuellement le test lui-même est incorrect.

A cette étape, on effectue le test fonctionnel du circuit. Le test fonctionnel consiste à vérifier la fonctionnalité de tous les circuits d'une plaquette sous différentes conditions d'alimentation. Il est réalisé à la fin du cycle de fabrication afin de sélectionner les pièces à assembler en boîtier.

Il se fait grâce à une machine que l'on appelle *testeur* que l'on peut peut voir sur la figure 2.6 et c'est à ce niveau qu'on utilise les *vecteurs de test* générés à l'étape de synthèse (voir figure 2.4). A ce stade le rendement mesuré permet de calculer le taux de défauts de la ligne de fabrication.

[6] fautes de collages : un exemple de représentation au niveau de la porte logique, d'un défaut physique réel.

Vecteurs et séquences de test La figure 2.5 schématise le principe d'application du test d'un circuit ou d'un bloc à l'intérieur d'un circuit : des valeurs sont appliquées sur les entrées, et les résultats obtenus sur les sorties sont comparés à des valeurs pré-déterminées (par simulation par exemple). Ceci correspond respectivement à un vecteur d'entrée V_e (*stimuli*) et un vecteur de sortie V_s (*référence*). Un « *vecteur de test* » est composé de ces deux vecteurs : $V = (V_e, V_s)$. Les *vecteurs de test* sont déterminés pour permettre la mise en évidence (avec un nombre de vecteurs minimum) des différentes fautes que l'on cherche à détecter (voir [43] pour plus de détails).

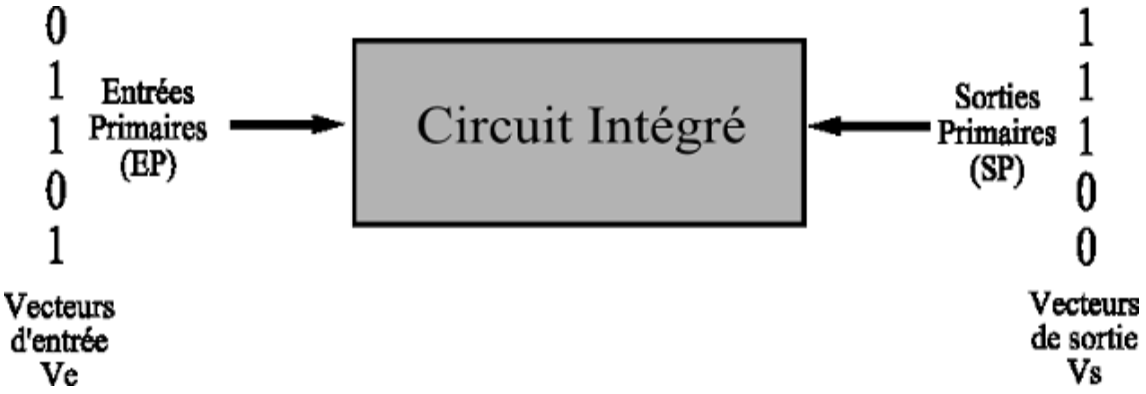

FIG. 2.5: Principe de bases des vecteurs de test.

2.3.3 Zoom sur la CAO

La décennie 80 a contribué à améliorer considérablement quelques unes des difficultés vues plus haut. Le concepteur de circuits intégrés peut de plus en plus faire abstraction des niveaux physiques, électroniques et même logiques de sa solution. Il dispose d'outils d'aide à la conception de plus en plus sophistiqués appelés *CAO* pour *Conception Assistée par Ordinateur* (en anglais : *EDA* pour *Electronic Design Automation*).

Ces outils permettent de décrire une solution sous forme d'un schéma d'architecture, puis d'entreprendre une validation par simulation. A l'aide de logiciels appropriés, la solution est ensuite transcrite en une réalisation selon les règles du fondeur (fabricant de silicium) retenues. Il n'est pas possible de dresser ici un panorama complet des outils de *CAO* disponibles sur le marché. En effet, un panorama deviendrait vite obsolète compte tenu des évolutions très rapides aussi bien des techniques que des sociétés commercialisant les outils. Néanmoins, les outils de *CAO* les plus connus et les plus répandus sur le marché sont réalisés par les sociétés *Synopsys* (numéro 1 sur le marché), *Mentor Graphics*, *Cadence*.

Les trois étapes principales de conception qui nous intéressent dans ce travail sont : la description *RTL* du circuit, la synthèse logique et le placement et routage. En effet, nous verrons par la suite que le travail réalisé dans cette thèse intervient entre la première et la

deuxième étape, c'est-à-dire avant la synthèse, puis entre la deuxième et troisième étape, c'est-à-dire avant le placement et routage.

2.4 Zoom sur le test

2.4.1 Le test des CI et son importance

Le test des circuits intégrés est un des sujets fondamentaux aujourd'hui. En effet, le marché de la micro-électronique est en perpétuelle évolution et concerne maintenant des applications diverses et grand public. Comme les volumes de production sont de plus en plus importants, il est impératif d'assurer une fiabilité satisfaisante pour un coût abordable. Cela nécessite de développer des techniques de test efficaces et le moins coûteuses possible pour les circuits en micro-électronique.

Le but des phases de test est d'éliminer les circuits défaillants, c'est-à-dire les circuits qui ne se comportent pas comme ils devraient.

Nous appellerons *circuits fonctionnels* les circuits qui fonctionnent correctement. Les défauts physiques sont causes de défaillances du circuit. Ces défauts physiques vont pouvoir être modélisés par une *faute*, modèle comportemental du défaut, qui va permettre un traitement dans le domaine du test. Ces fautes vont entraîner une erreur, c'est-à-dire une manifestation locale de la faute. Cette erreur va ensuite entraîner ou non une défaillance du circuit. Le but du test est donc de prévoir, trouver les fautes. La première phase va consister à *détecter* l'existence de fautes (comme les fautes de collages par exemple), la seconde à en *localiser* les causes.

D'une manière générale, le test consiste à envoyer un signal au circuit à tester et à recueillir sa réponse de manière à l'analyser, typiquement à la comparer à la réponse d'un circuit qui fonctionne correctement. Ceci se fait grâce à l'utilisation d'un équipement externe au circuit appelé *testeur* représenté dans la figure 2.6. Il s'agit d'une grosse machine très couteuse qui permet de tester les circuits un par un.

FIG. 2.6: Testeur.

Pour ce faire, l'idéal serait bien entendu de pouvoir accéder à tous les points du circuit pour pouvoir trouver la faute. Ceci est évidement impossible dans la pratique, tant la complexité des circuits intégrés est importante. De plus le nombre limité de plots ne permet un accès physique qu'à une petite partie des points du circuits, limitant la testabilité de celui-ci, c'est-à-dire sa capacité à être testé.

Les enjeux du test des circuits intégrés sont les suivants :
- améliorer la qualité du test. On parle de *couverture de fautes* dans le cas de circuits numériques et analogiques. Le but est alors de diminuer le nombre de circuits défaillants qui échappent au test et le nombre de circuits rejetés alors qu'ils sont fonctionnels.
- améliorer le temps de test. Cela est possible uniquement en réduisant la durée du processus de test.

Ces améliorations ont pour finalité de réduire les coûts du test. En effet, la *loi de 10* indique qu'à chaque étape de la vie d'un produit, le coût résultant de la détection d'une défaillance est multiplié par 10.

Ainsi une erreur détectée une fois la puce empaquetée coûtera dix fois plus cher qu'au niveau plaquette. Au niveau carte, le coût sera encore multiplié par 10 et à nouveau au niveau système. Si, enfin, la faute est détectée au niveau de l'utilisateur final, le coût sera encore 10 fois plus élevé, soit 100000 fois plus qu'au niveau plaquette. Il est donc particulièrement important de bénéficier d'un test efficace au plus tôt de la phase de production et conception du circuit intégré.

2.4.2 Les différents types de test

Il existe différents types de tests suivant les étapes de conception et fabrication du produit. On peut les regrouper dans les deux familles suivantes :
- au niveau prototype, une caractérisation complète du circuit est nécessaire afin de permettre un éventuel retour à la conception pour améliorer le produit. Ce type de test est très exhaustif, demande du temps, des ingénieurs de test qualifiés, mais ne concerne qu'un nombre limité de circuits. La vitesse de test n'est alors pas un critère essentiel.
- au niveau production, on peut se contenter d'un *test fonctionnel*[7] dans le cas où les processus de fabrication ne sont pas encore mûrs. Ou encore d'un *test structurel*[8] dans le cas ou la technologie est mature. Ces tests sont donc moins approfondis que la caractérisation, mais concernent un très grand nombre de circuits. Il va donc être intéressant d'automatiser et d'accélérer le test par des techniques de *DFT* (*Design For Test*), car le coût devient alors quasiment proportionnel au temps de test.

[7] test fonctionnel : test consistant seulement à vérifier si la fonctionnalité du circuit est respectée.
[8] test structurel : test consistant à vérifier l'intégrité « physique » du circuit.

Nous verrons dans les parties 2 et 3 de ce manuscrit les techniques utilisées pour l'amélioration du dernier point cité ci-dessous.

2.4.3 Test et conception en vue du test

Le test occupe une place importante dans le flot de conception d'un circuit intégré. En effet, si la validation fonctionnelle permet au concepteur de s'assurer du bon fonctionnement du circuit, et donc de l'adéquation de la solution aux objectifs, ce type de test ne permet pas de décider dans des conditions économiques acceptables si un circuit après fabrication est sans défaut. Seul le test de fabrication permet de garantir la fabrication car il a pour objectif d'activer tous les nœuds du composant.

Cependant, pour des circuits complexes, l'utilisation exclusive d'un *testeur* n'est plus possible vu l'accroissement considérable du nombre d'éléments mémoires, de portes logiques et de connexions. Des techniques particulières sont à définir et à mettre en œuvre durant la phase de conception et de définition de la réalisation pour aboutir à un circuit testable.

Ce travail est à entreprendre après l'étape de conception architecturale, car la connaissance de la solution structurelle est nécessaire pour le test d'un circuit. Pour cela, des techniques de *conceptions en vue du test* sont apparues.

La *conception pour la testabilité* (DFT) est une méthodologie de conception ayant pour objectif de faciliter les tests des composants du circuit en modifiant la manière de concevoir l'architecture des circuits et en ajoutant des composants complémentaire comme on le verra par la suite dans les parties 2 et 3 de ce manuscrit.

L'amélioration de la *testabilité* passe par la prise en compte de ce problème dès la conception de l'architecture du circuit. L'objectif d'une bonne conception pour la testabilité est d'accroître la *contrôlabilité* et l'*observabilité*.

La *contrôlabilité* indique la difficulté de positionner une ligne à 0 ou à 1 à partir des entrées primaires (EP). L'*observabilité* indique la difficulté de propager une erreur à partir d'une ligne vers les sorties primaires (SP).

Il est important d'améliorer les deux critères de *contrôlabilité* et d'*observabilité* afin de parvenir a un bon niveau de *testabilité*.

Un des moyens employés pour accroître les deux paramètres (contrôlabilité et observabilité) est l'emploi des fonctions séquentielles tel que le *scan path* que nous introduisons sommairement dans le paragraphe suivant.

Nous étudierons ensuite cette technique en détail dans la partie 2 de ce document, dans les chapitres 3 et 4.

2.4.4 Scan Path pour le test des CI

Les circuits intégrés ainsi que de nombreux produits électroniques d'aujourd'hui requièrent des temps d'apparition sur le marché de plus en plus courts. Les techniques de conception en vue du test (*DFT*) sont la meilleure garantie de concevoir des systèmes sur puce qui possèdent la qualité requise en matière de *testabilité*.

De nos jours, la technique de *scan path* (ou *chaînes de scan*) est largement adoptée dans l'industrie pour la *DFT*. Elle est employée dans le cadre le test « séquentiel » c'est-à-dire le test des circuits séquentiels.

Cette technique consiste en un ensemble de composants et de connexions électroniques qu'on rajoute au CI afin de faciliter l'étape de test post-fabrication. L'idée de base est la capacité de décaler les informations en scrutant l'ensemble des états du circuit. Toutes les bascules sont chaînées les unes aux autres afin de permettre l'introduction des vecteurs de test en entrée ainsi que la récupération des résultats de test en sortie. Ainsi, *le registre à décalage*[9] qui en résulte est totalement *contrôlable* et *observable* à travers les entrées et les sorties primaires du circuit.

Objectif de la thèse pour le *scan path*
Notre but est de développer un outil permettant de générer automatiquement des chaînes de *scan* optimales en terme de surface additionnelle et temps de test pour un circuit donné, tout en respectant un certain nombre de contraintes électroniques. Les techniques utilisées sont issues de l'optimisation combinatoire et de la recherche opérationnelle. Ces travaux ont été réalisés dans le cadre du projet ASTER, en collaboration avec la société *DeFacTo Technologies*.

2.5 Les mémoires

2.5.1 Généralités sur les mémoires

Les *mémoires* sont des circuits intégrés permettant d'enregistrer des mots binaires. Ces mots, la plupart de 8 ou 16 bits, sont généralement des instructions d'un programme ou des données à sauvegarder temporairement.

Les *mémoires* constituent des éléments importants de tout système de traitement numérique de l'information. Le tiers de parts de marché qu'elles représentent dans le secteur total des semi-conducteurs est là pour le confirmer. Leur utilisation croissante dans les circuits complexes actuels est également un point important.

On peut définir *une mémoire* comme un ensemble matériel de registres constitués de mots binaires. Par principe, *les mémoires* associent une adresse binaire à chacun de ses registres. De manière générale, les mémoires peuvent être partagées en deux familles selon les fonctionnalités de lecture et d'écriture (les read only et write only). Les types de mémoires les plus couramment utilisées dans chaque famille sont les suivantes :

- les mémoires vives dynamiques (DRAM pour : « *Dynamic Random Access Memory* ») qui sont les mémoires les plus denses mais avec un temps d'accès relativement important et la nécessité d'un rafraîchissement périodique,

[9] Registre à décalage: registre dont le contenu peut être décalé d'une ou plusieurs positions vers la droite ou vers la gauche. Les informations peuvent entrer ou sortir, en série ou en parallèle, selon le type du registre

- les mémoires vives statiques (SRAM pour « *Static Random Access Memory* ») qui sont les plus rapides,
- les mémoires mortes (ROM pour « *Read Only Memory* ») qui sont programmées à la fabrication et maintiennent leur information même sans alimentation,
- les mémoires mortes programmables qui se subdivisent en EPROM (« *Erasable Programmable Read Only Memory* ») et EEPROM (« *Electrically Erasable Programmable Read Only Memory* ») selon que les mots programmés peuvent être effacés globalement par lumière ultraviolette ou sélectivement par un moyen électrique.

2.5.2 Le test des mémoires

Les tests les plus simples et les plus efficaces pour les mémoires appartiennent à une catégorie de test appelés *marching test* [58]. Ils permettent de détecter la majorité des fautes pouvant se présenter dans une mémoire (les fautes de collage, les fautes de transition et les fautes de couplage).

Un test de type *march* parcourt toutes les cellules mémoires une ou plusieurs fois par ordre croissant ou décroissant en appliquant successivement sur chaque cellule une séquence d'opérations données appelée *élément*.

Les séquences d'opérations peuvent consister en :
- écrire un 0 dans une cellule (w0),
- écrire un 1 dans une cellule (w1),
- lire une cellule avec une valeur 0 attendue (r0),
- lire une cellule avec une valeur 1 attendue (r1).

D'un point de vue formel, chaque élément est regroupé entre parenthèses, les opérations étant séparées par des virgules. Par exemple l'élément (r0,w1) signifie lire la cellule en espérant trouver la valeur 0 puis écrire un 1.

L'utilisation massive de bancs mémoires dans les circuits actuels peut poser des problèmes de test sérieux car très souvent les lignes de données, d'adressage et de contrôle des blocs mémoires ne sont pas directement accessibles au travers de broches extérieures. Le test intégré a de ce fait été envisagé très tôt pour ce genre de blocs. La régularité des test *march* les rendent tout particulièrement adaptés à une utilisation dans le test des mémoires.

Le lecteur intéressé par le test des mémoires peut consulter l'ouvrage de *Van de Goor* [57] ou celui de *Mazumder et al.* [48].

2.5.3 Le BIST mémoire

Dans la section 2.4.4, nous avons discuté des techniques de conception visant à faciliter le test des circuits à travers l'insertion des chaînes de *scan*. Ces méthodes et techniques s'appliquent plus particulièrement dans le cadre d'un test externe faisant appel à un équipement de test indépendant du circuit comme les testeurs.

Le *Test Intégré*, ou *Built-In Self-Test*(*BIST*), est une technique de conception qui consiste à inclure dans le circuit une partie des fonctions réalisées par un testeur externe,

notamment la génération et l'application de vecteurs de test, ainsi que l'analyse des réponses du circuit à ces stimuli.
Bien que cette approche puisse apparaître comme génératrice de coûts supplémentaires lors de la conception et de la fabrication, il est maintenant admis que le coût d'intégration de fonctions additionnelles de test dans ce circuit peut être largement compensée par les bénéfices induits en termes de temps de test, coût des testeurs, voire de fiabilité et de réduction des coûts de maintenance.

Les techniques de test intégré peuvent être classées en deux catégories qui sont : le test intégré en-ligne (*on-line*) et le test intégré hors-ligne (*off-line*), incluant les approches fonctionnelles et structurelles.

Un test intégré en-ligne, est appliqué sous les conditions normales de fonctionnement du système. Le circuit sous test n'est donc pas placé dans un mode de fonctionnement particulier. Si les opérations de test sont simultanées alors on parle de test intégré en-ligne *concurrent*. Ce type de test est accompli en s'appuyant sur des techniques de codage. Pour un test intégré en-ligne non-concurrent, les opérations de test sont appliquées lors des phases de dormance du système. Il est réalisé par des routines de diagnostic qui peuvent être interrompues à tout moment pour que le système reprenne ses opérations normales.

Durant un test intégré hors-ligne, le système est placé dans un mode de fonctionnement particulier, le *mode test*, et ne réalise pas les opérations pour lequel il a été conçu. Ce type de test est appliqué à l'aide de générateurs de vecteurs de test et d'analyseurs de réponses intégrés sur le silicium. Contrairement au test en-ligne, ce type de test ne peut détecter d'erreurs en temps réel, c'est-à-dire lorsqu'elles surviennent pour la première fois. Les techniques hors ligne fonctionnelles consistent à exécuter un test basé sur la description fonctionnelle du circuit et des modèles de fautes de haut niveau. Inversement, les techniques hors lignes structurelles s'appuient sur la structure du circuit. Dans ce contexte, on peut utiliser un modèle de faute structurel comme le modèle de faute de collage par exemple.

Objectif de la thèse pour le test des mémoires
Cette partie traite du test intégré hors-ligne structurel pour le test des mémoires comme nous le verrons à la partie 3. En effet, du fait de la structure particulière des mémoires (structure régulière), une des solutions les plus utilisées actuellement pour tester les mémoires est la méthode *BIST*. L'objectif de nos travaux est de développer des solutions algorithmiques permettant d'optimiser et d'automatiser l'insertion de blocs *BIST* pour le test des mémoires. Ces travaux ont été réalisés dans le cadre du projet *ASTER*, en collaboration avec la société STMicroelectronics.

2.6 Conclusion

Si cette thèse était une bâtisse en construction, le chapitre précédent serait la boîte à outils et celui-ci les fondations. Nous avons en effet présenté dans ce chapitre d'abord les briques de base qui constituent un *CI* comme les *transistors*, *portes logiques*, *bascules*, *etc.* Nous avons ensuite présenté les différentes étapes du processus de réalisation d'un *CI*,

suivies des notions fondamentales sur la conception et le test des circuits. Enfin, nous avons introduit les mémoires ainsi que les dispositifs nécessaires à leur test.

Toutes ces informations seront utiles pour comprendre et situer la suite de nos travaux de recherche au cours de cette thèse. Pour en savoir plus sur le domaine de la conception ainsi que du test des CI, en plus des références citées dans ce chapitre, le lecteur pourra se référer à différents sites *web*. En particulier, le site : `http://www.tmworld.com`, sur lequel se trouvent des informations régulièrement mises à jour, un magazine en ligne et un guide d'achat avec la liste des principaux fournisseurs de matériel et d'outils *CAO*.

Les principales évolutions prévues pour la prochaine décennie sont détaillées dans le document édité sous l'égide de l'association des industriels du semi-conducteur (*SIA* : *Semiconductor Industry Association*, *San Jose*, *Californie*) sous l'appellation « *International Technology Roadmap of Semiconductors* » et accessible sur le site : `http://public.itrs.net`.

Deuxième partie

Technique du Scan Path au niveau RTL

A l'issue du processus de conception, on teste l'adéquation du *Circuit Intégré* à son cahier des charges par des outils logiciels de vérification et de simulation ou à l'aide de prototypes. Après l'élimination (supposée complète) de toutes les erreurs de conception, le bon fonctionnement du circuit dépend alors essentiellement de la qualité du procédé technologique de fabrication. Pour s'assurer de ce bon fonctionnement après fabrication, il est alors nécessaire de s'assurer de l'inexistence ou de la quasi-inexistance de défauts.

Cela passe par la préparation du circuit à la phase de test, en modifiant sa propre architecture pendant la conception de celui-ci. Cette modification consiste à considérer des techniques spécifiques de conception connues par les techniques de conception testables.

Compte tenue de la complexité des *Circuits Intégrés* actuels, des techniques de conception testables sont donc couramment utilisées pour améliorer la *testabilité* des *CI*. Dans cette optique, le test par *scan* est la technique de conception en vue du test (*DFT* pour le terme anglais *Design For Test*) la plus utilisée à l'heure actuelle dans l'industrie [41]. En effet, d'après le rapport de l'*ITRS*, *2000* près de 90% des circuits intégrés vendus dans le monde possèdent une ou plusieurs chaînes de scan. Cette tendance se confirme avec les circuits de nœuds technologiques avancés.

Cette deuxième partie concerne l'optimisation de l'insertion des *chaînes de scan* au niveau *RTL*. Cette problématique nous a été posée par la *start-up DeFacTo Technologies* qui propose des concepts innovateurs dans le domaine de la *DFT*.

Dans la chapitre 3, nous présentons d'abord la méthode du *scan path* ainsi que les deux niveaux d'insertion possibles dans le flot de conception d'un *CI*. Nous exposons ensuite la problématique d'optimisation de l'insertion des chaînes de scan au niveau *RTL*. Enfin, nous proposons une modélisation basée sur la théorie des *graphes*.

Dans le chapitre suivant (chapitre 4), nous exposons l'*algorithme de stitching* que nous avons établi pour déterminer un chaînage optimal des éléments mémoire dans la *chaîne de scan* au niveau *RTL*. Afin d'évaluer nos résultats nous avons effectué une série de tests sur des benchmarks classiques disponibles dans la littérature du test des *CI*. Les résultats numériques obtenus ainsi que leur interprétation sont disponibles dans la dernière partie de ce chapitre.

Chapitre 3

Problème et Modélisation de l'insertion des chaînes de scan

Sommaire

3.1 Introduction 45
3.2 Méthode du test par chaînes de scan 46
3.2.1 Définition 46
3.2.1.1 Principe de l'insertion des chaînes de scan à bas niveau 47
3.2.2 Approche de scan intégral à haut niveau : Scan RTL 50
3.2.2.1 Principe de l'insertion des chaînes de scan haut niveau 51
3.2.3 Comparaison entre les deux flots d'insertion 52
3.3 Problème de l'insertion des chaînes de scan 54
3.3.1 Problématique 54
3.3.2 Objectif de la thèse pour le scan 55
3.3.3 Etat de l'art 55
3.4 Modélisation de l'insertion des chaînes de scan au niveau RTL 57
3.4.1 Données 57
3.4.2 Objectifs 57
3.4.3 Contraintes 58
3.5 Formulation mathématique du problème 58
3.5.1 Design Graph, G_D 59
3.5.2 Proximity Graph, G_P 59
3.5.3 Construction du graphe G_P à partir de G_D 60
3.5.4 Critère d'optimisation 61
3.6 Conclusion 62

3.1 Introduction

Nous décrivons dans ce chapitre la méthode du *scan path*. Nous exposons d'abord le *scan* traditionnel appelé *scan gate* tel qu'il est utilisé actuellement. Nous présenterons ensuite

la problématique d'insertion de chaînes de scan au niveau *RTL*. Enfin, nous proposons dans la dernière partie un modèle mathématique correspondant à notre problème micro-électronique. En effet, plusieurs problèmes combinatoires se posent au niveau des choix et des décisions à prendre sur certains paramètres électroniques pour optimiser l'insertion des chaînes de scan au niveau *RTL*. La construction d'un modèle mathématique a pour but d'optimiser par la suite le phase d'insertion du scan.

C'est dans ce contexte que nous avons collaboré avec la *start-up DeFacTo Technologies* qui est une des entreprises les plus avancée dans le domaine de la recherche et développement du scan au niveau *RTL*. Ce partenariat nous a permis de profiter d'une bonne expertise dans les travaux de recherche concernant la *testabilité* au niveau *RTL* et ainsi expérimenter nos modèles et analyser nos résultats comme nous le verrons par la suite.

3.2 Méthode du test par chaînes de scan

3.2.1 Définition

La méthode du *scan path*, en Français *chaîne de scan*, consiste à modifier les bascules du circuit afin de les rendre directement *contrôlables* et *observables* durant la phase de test. Cette technique, bien qu'offrant de nombreux avantages, notamment en terme d'efficacité avec un bon taux de couverture de fautes, présente quelques inconvénients tels que son coût en surface (dû à l'insertion de composants et connexions supplémentaires nécessaires au scan) ou le temps de test important qu'elle induit malgré les bénéfices qu'elle apporte de ce côté.

Dans la technique du *scan path*, le circuit est conçu de manière à présenter deux modes de fonctionnement : un *mode de fonctionnement normal* et un *mode de test* où tous les nœuds internes de mémorisation (bascules élémentaires) sont inter-connectés sous la forme d'un ou plusieurs *registres à décalage* [1]. En mode de test, un vecteur de test peut être chargé en série à l'intérieur des bascules. Après retour au mode normal, ce vecteur peut donc être transmis à l'intérieur du circuit et le résultat obtenu agit alors pour modifier le contenu des différentes bascules du circuit. Le résultat complet peut alors être extrait du circuit en mode série puis être comparé à la réponse correcte attendue.

En utilisant cette technique, le test d'un circuit séquentiel, qui est un problème difficile se ramène au test de sa partie combinatoire qui est plus facile, ce qui réduit considérablement l'effort de génération des vecteurs de test.

Nous allons voir ici deux niveaux d'insertion possibles des *chaînes de scan* : le scan bas niveau que nous appelons *scan traditionnel* (ou encore *scan gate*) qui se réalise après l'étape de synthèse et le scan haut niveau que nous appelons *scan RTL* qui lui se réalise avant l'étape de synthèse. Les sections suivantes expliquent les détails de l'insertion du *scan gate* et du *scan RTL* dans un circuit ainsi que les flots d'insertion.

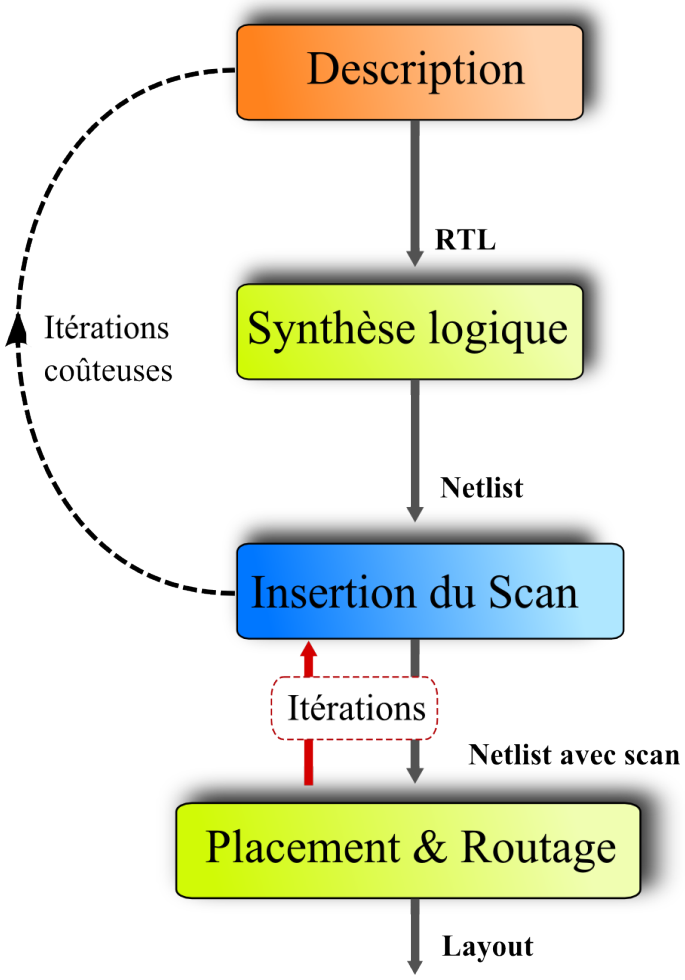

FIG. 3.1: Niveau d'insertion du scan traditionnel.

3.2.1.1 Principe de l'insertion des chaînes de scan à bas niveau

La figure 3.1 montre le niveau de conception dans lequel le scan gate est intégré dans le flot de conception d'un *CI* après l'étape de synthèse. Le processus, décrit dans la figure 3.1 est le *flot traditionnel* de conception de circuits intégrés, où le scan est inséré une fois le processus de synthèse effectué. Plusieurs outils sur le marché de l'*EDA* (*Electronic Design Automation*) insèrent le scan à ce niveau, on peut citer par exemple : *DFT Compiler* de la société *Synopsys*, *DFT Advisor* de la société *Mentor Graphics*, *EN Counter Test* de la société Cadence et *Talus* de la société *Magma*.

Dans le but de simplifier l'exposé du principe de cette approche, un exemple de circuit

[1] *registre à décalage*: est un registre de taille fixe dans lequel les bits sont décalés à chaque top d'horloge.

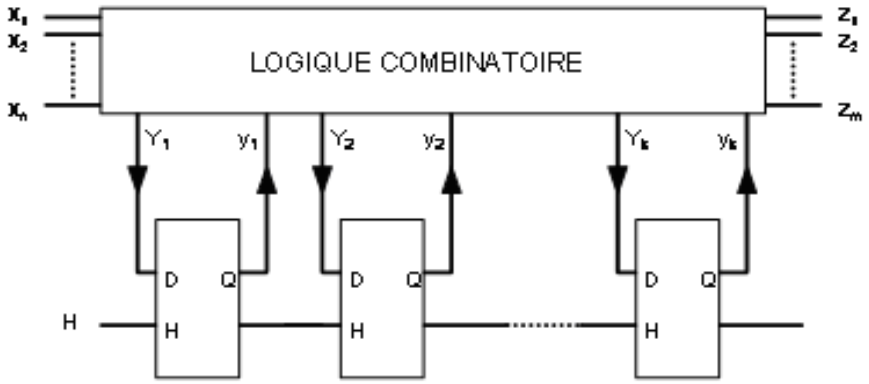

FIG. 3.2: Structure générale d'un circuit séquentiel.

est représenté par la figure 3.2[2] où la partie combinatoire est isolée de la partie séquentielle.

A partir de la structure initiale donnée sur la figure 3.2[3], le principe de base de la technique de *scan path* consiste à remplacer toutes les bascules D par des bascules D précédées d'un multiplexeur permettant de sélectionner l'entrée de la bascule, c'est ce qu'on appelle une bascule scannée. La figure 3.3 montre la structure d'une cellule scan avec le circuit contenant la chaîne de scan ; les deux figures 3.3 et 3.4 illustrent la même chose, c'est juste la présentation qui diffère. Le fait de remplacer les bascules par des bascules scannées entraîne un coût additionnel en surface.

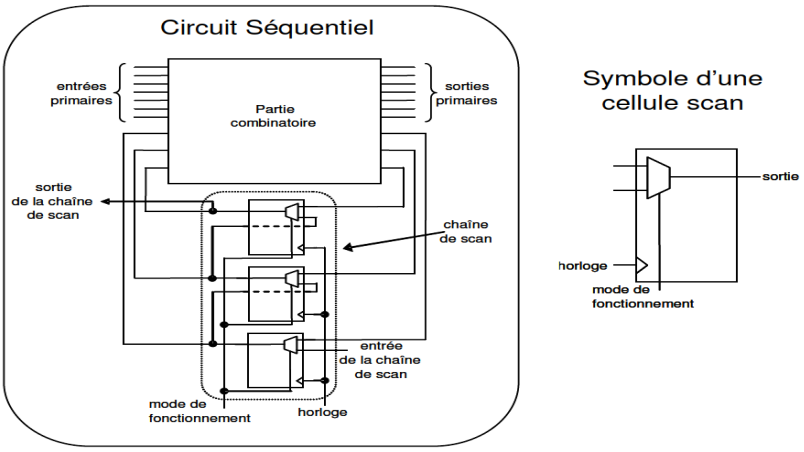

FIG. 3.3: Description d'un circuit séquentiel muni d'une chaîne de scan.

[2] *circuit séquentiel : source livre [41].*
[3] *circuit : source livre [41].*

Par rapport à la structure de départ figure 3.2, on rajoute une entrée de contrôle T, comme le montre la figure 3.4. Lorsque l'entrée de contrôle T est à la valeur logique 0, l'entrée D de la bascule est l'entrée d_0 (mode de fonctionnement normal), lorsque T est à la valeur logique 1 l'entrée D de la bascule est l'entrée d_1 (mode de test). Cette bascule modifiée est utilisée dans la technique Scan path de base comme le montre la figure 3.4. Avec T à la valeur logique 0, le circuit est en mode normal d'opération.

Pour tester le circuit, on positionne la broche T à la valeur logique 1. Les k bascules D sont alors interconnectées sous la forme d'un registre à décalage qui a pour entrée l'entrée X_n du circuit initial et dont la sortie Q apparaît sur la sortie Z_m par l'intermédiaire du multiplexeur ajouté en sortie.

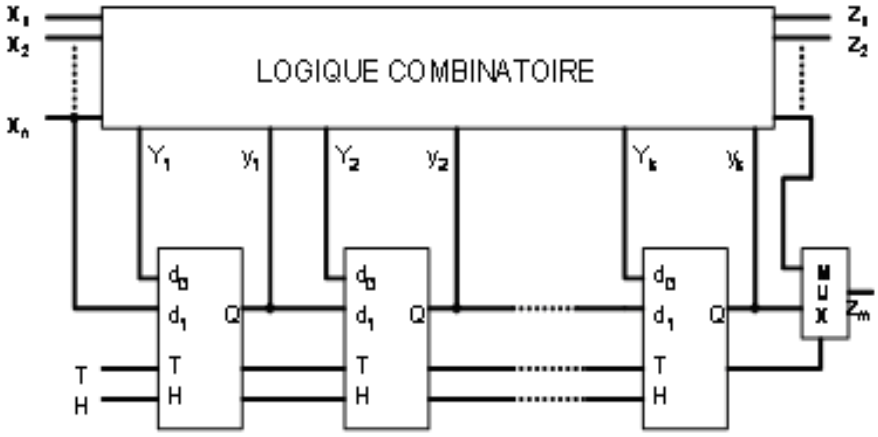

FIG. 3.4: Architecture générale de la technique de scan path.

Le test du circuit procède de la façon suivante :

1. positionner le circuit en mode test ($T = 1$),
2. entrer par décalage le vecteur de test $y_1, ..., y_k$ à l'intérieur des bascules (k : taille du vecteur de test),
3. positionner les valeurs de test correspondantes sur les entrées primaires $X_1, ..., X_n$ (n : taille de la chaîne de scan),
4. positionner l'entrée T à la valeur logique zéro et après un temps nécessaire à la stabilisation des sorties de la partie combinatoire vérifier les différentes sorties primaires Z_i,
5. appliquer une impulsion d'horloge sur l'entrée H,
6. positionner l'entrée T à la valeur logique 1 et sortir en série le contenu du registre à décalage (des bascules) par l'intermédiaire de la sortie Z_m,
7. comparer le résultat avec les résultats attendus.

On peut remarquer que simultanément à la sortie du vecteur d'observation (contenu des bascules), on peut entrer le nouveau vecteur de test par l'intermédiaire de l'entrée X_n.

Le test des bascules elles-mêmes est assuré en positionnant le circuit en mode test (entrée T à la valeur logique 1) et en décalant une chaîne de 1 et de 0 pour vérifier la potentialité de chaque bascule de décaler la valeur logique 0 et la valeur logique 1. Le temps d'exécution de la procédure de test est directement lié au temps de chargement et déchargement des vecteurs de test et de leur nombre.

3.2.2 Approche de scan intégral à haut niveau : Scan RTL

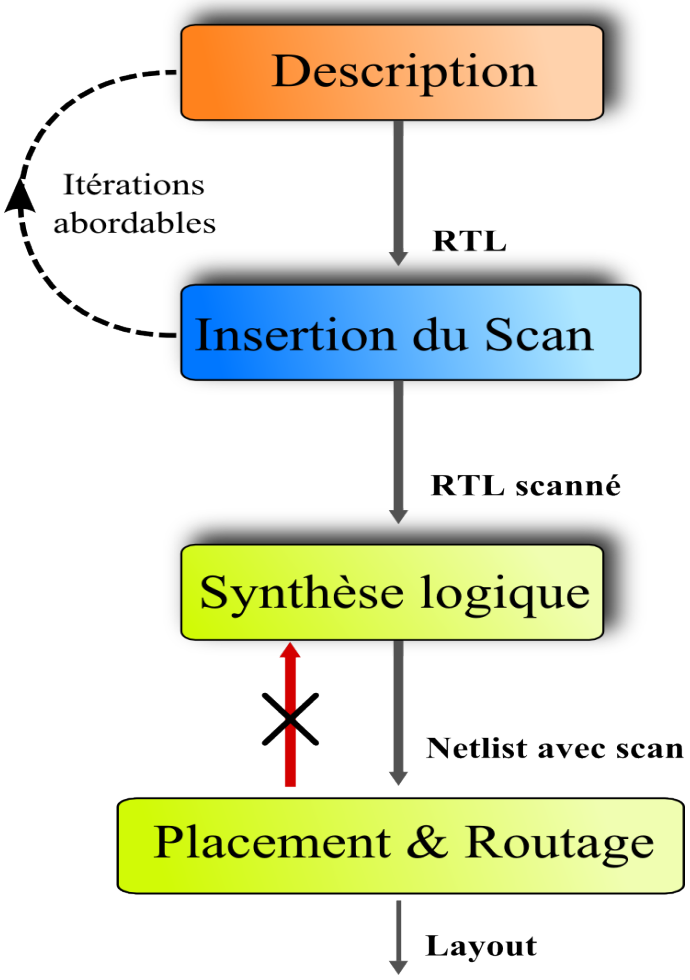

FIG. 3.5: Niveau d'insertion du scan RTL.

Le concept de *Scan RTL* est identique à celui du *scan gate* mais appliqué au niveau *RTL*, donc l'insertion des *chaînes de scan* se fait avant l'étape de synthèse comme le montre la figure suivante 3.5 [3].

L'implantation de la technique de scan ne s'est répandue que sur des architectures de

bas niveau du circuit. L'approche qui consiste à remonter le niveau d'implantation des techniques de *DFT* telle que le scan est un concept nouveau proposé par notre partenaire industriel *DeFacTo Technologies*. Ce travail a fait l'objet de la création de la *start-up DeFacTo Technologies*.

3.2.2.1 Principe de l'insertion des chaînes de scan haut niveau

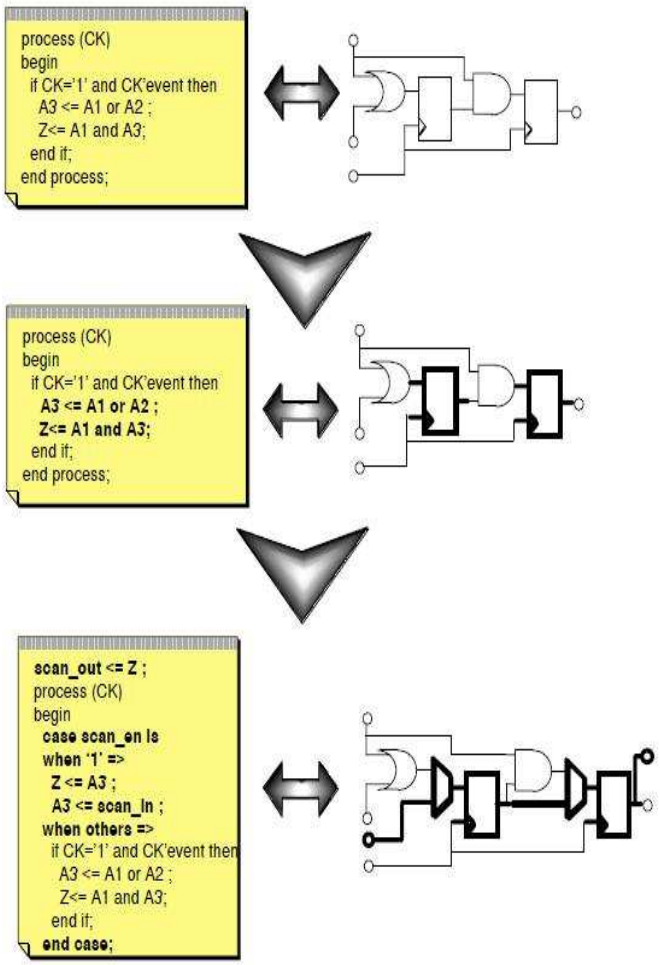

FIG. 3.6: Principe de la méthode du Scan RTL.

L'idée est de prendre en entrée la description *RTL* du circuit, et produire en sortie du *RTL* contenant des chaînes de scan (en *VHDL* ou *Verilog*).

Comme le montre la figure 3.6, la méthode proposée se base sur deux étapes permettant l'insertion du scan à partir d'une description *VHDL* comportementale : la localisation des éléments de mémorisation et l'insertion de la logique de scan.

Localisation des éléments de mémorisation :
Avant de pouvoir construire une chaîne de scan dans un circuit, les éléments de mémorisation de celui-ci doivent être identifiés. Ceci est fait à partir de deux propriétés principales liées aux descriptions comportementales en VHDL 3.6.

- Propriété 1 (signaux) : dans un processus synchronisé sur l'horloge, toute affectation de signal génère un élément de mémorisation.
- Propriété 2 (variables) : dans un processus, lorsqu'une instruction de synchronisation est exécutée, une variable affectée mais non utilisée génère un élément de mémorisation.

Modification de la description : Une fois les éléments mémoires détectés, il s'agit dans une seconde phase d'insérer la logique de scan au niveau de la description comportementale du circuit. Cette phase est décomposée en deux étapes :

- le scan local, dont le rôle est de chaîner chaque élément de mémorisation relatif à un objet *VHDL*
- le scan global qui relie les différents éléments de mémorisation liés à différents objets.

L'approche est également généralisée à d'autres langages de description matérielle tel que le *Verilog*. L'exemple de la figure 3.6 illustre l'insertion d'une chaîne de scan au niveau *RTL* en *Verilog*.

L'entreprise *DeFacTo Technologies* propose le *flot HIDFT*, où l'insertion du scan s'effectue au niveau *RTL* avec l'outil *HIDFT Scan*. Par rapport à l'approche traditionnelle, il s'agit de traiter le problème de test et de conception testable des circuits en amont dans le flot de conception d'un circuit intégré. Des informations complémentaires sur le scan *RTL* sont disponibles dans les articles de *Aktouf* [5], ainsi que dans le livre [2].

3.2.3 Comparaison entre les deux flots d'insertion

Contrairement aux techniques traditionnelles de *DFT* qui s'appuient sur une description du circuit de bas niveau (niveau portes logiques, etc.), la méthode proposée est plus générique et permet ainsi de prendre en compte les aspects de testabilité plus tôt dans le processus de conception, comme le montre la figure 3.7.

Un des avantages du scan *RTL* est de considérer une description du circuit qui est uniquement fonctionnelle et donc plus facile à traiter qu'une description au niveau portes logiques. Plus précisément, pour un circuit d'aujourd'hui, il s'agit de traiter des milliers de lignes de code *RTL* au lieu du traitement de quelques centaines de milliers de portes de base [4].

En effet, au niveau *RTL* (*Register Transfer Level*) ou comportemental, les descriptions des circuits sont plus concises et donc plus facilement appréhendées. De plus, la logique de test est globalement optimisée avec la logique fonctionnelle du circuit au moment de la synthèse et ainsi son insertion est moins coûteuse en termes de ressources.

Le fait de déplacer l'insertion du scan au niveau *RTL* permet aussi de réduire de manière considérable le temps d'insertion des chaînes de scan nécessaires au test. Classiquement, la logique associée à une technique de *DFT* telle que le scan est insérée une fois le processus

de synthèse effectué, avec plusieurs itérations entre l'étape de synthèse et le placement et routage. Alors qu'au niveau *RTL* cela ce fait en une seule passe [5].

Un autre intérêt de l'insertion des *chaînes de scan* au niveau *RTL* est la possibilité de faire tout un ensemble de simulations, vérifications et estimations de différents paramètres tant fonctionnels que de test comme le *timing*[4] et le *test power*[5] avant de passer à la synthèse.

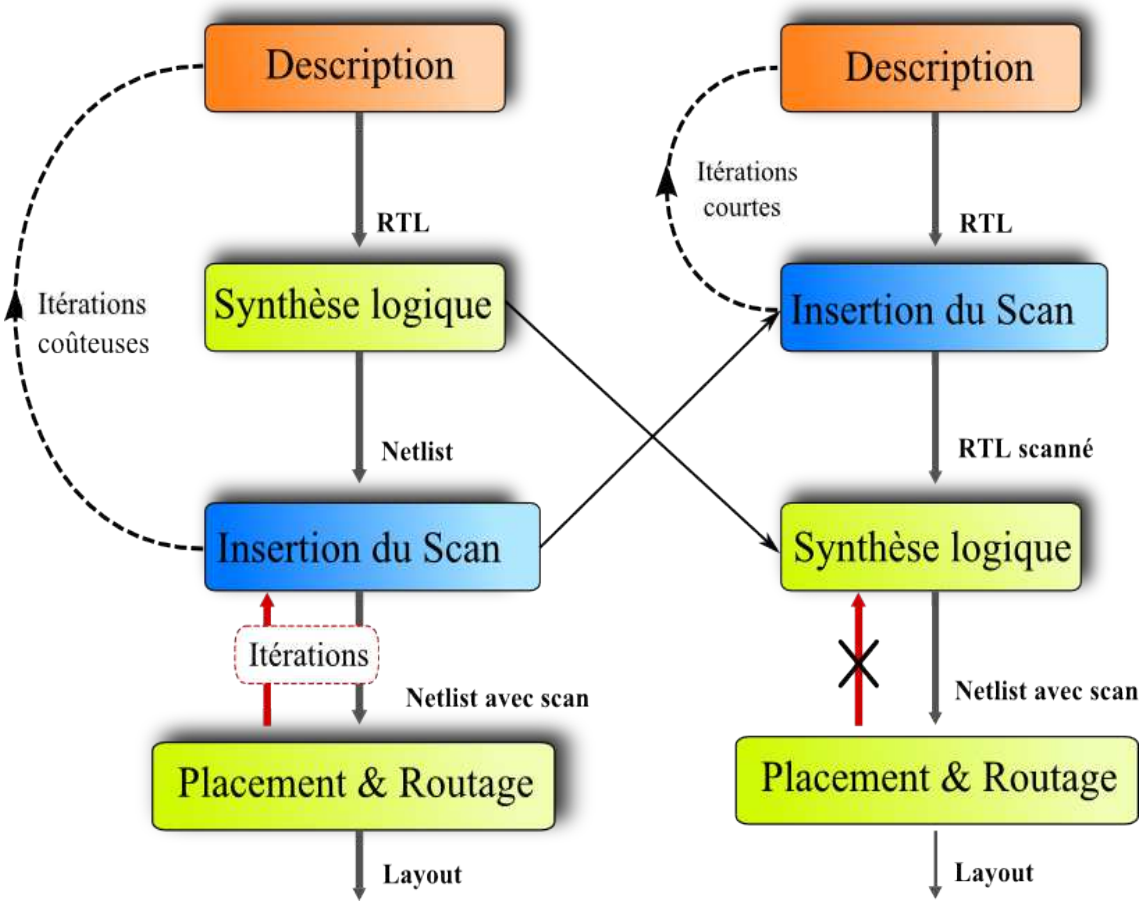

FIG. 3.7: Comparaison des Flots d'insertion.

[4]timing : chemin critique qui correspond au plus long chemin de stabilisation des signaux
[5]power : consommation en puissance du circuit en mode test

3.3 Problème de l'insertion des chaînes de scan

3.3.1 Problématique

Le test par scan permet de réduire l'effort de test en rendant directement accessible la partie combinatoire du circuit. Cependant, il présente les trois inconvénients majeurs suivants :

La surface additionnelle : elle est due à l'insertion des *bascules scan*, ainsi qu'aux connexions supplémentaires nécessaires au chaînage des éléments mémoire (routage des chaînes). En particulier, les connexions entre des bascules trop éloignées dans le *CI* entraîne : un coût en silicium, des dégradations des performances ainsi que des contraintes supplémentaires aux outils de placement et routage.

Le temps de test : un autre inconvénient de l'utilisation de cette technique est l'augmentation du temps de test qui est proportionnel à la taille de la *chaîne de scan*, car les *vecteurs de test* sont chargés en série. En effet, si le circuit à tester est de taille moyenne à importante (quelque milliers ou dizaines de milliers d'éléments mémoires), l'implémentation d'une seule chaîne de scan nécessiterait alors le décalage de toutes les données du *vecteur de test* à travers le circuit. Ceci peut être très couteux en temps vu le volume de production des circuits intégrés généralement conséquent.

Une solution consiste à diviser la *chaîne de scan* en plusieurs chaînes et d'introduire les chaînes en parallèle dans le *CI*. Mais il ne faut pas négliger le fait que chaque chaîne possède ses propres ports d'entrées/sorties dans le *CI*.

La consommation de puissance : un autre problème qui peut apparaître lors du test par scan est lié à la consommation de puissance du circuit, qui peut être supérieure à celle mise en jeu lors du fonctionnement normal du circuit [27]. Cette consommation excessive lors du test provient du nombre important de commutations qui sont générées lors du chargement et du déchargement des vecteurs de test dans la (ou les) chaîne(s) de scan. Des expérimentations réalisées sur des circuits industriels ont montrées que la consommation lors du test pouvait être de 2 à 3 fois la consommation du *CI* en fonctionnement normal.

Pour réduire certains de ces inconvénients majeurs qui sont la surface supplémentaire, des techniques dérivées dites *scan partiel* ou incomplet peuvent être utilisées. Ces techniques consistent à sélectionner les bascules qui vont être transformées en cellules scan, de façon à optimiser les différents critères en établissant un compromis.

Quoiqu'il en soit et malgré tous ces inconvénients, la méthode du scan reste la technique de conception en vue du test la plus utilisée, car permettant d'obtenir des taux de couverture proches de 100%. C'est aussi une technique fiable, éprouvée, standardisée et qui s'insère facilement dans un flot de conception [41].

Quel que soit le niveau d'insertion des *chaînes de scan* (niveau *gate* ou niveau *RTL*) on retrouve les inconvénients cités ci-dessus.

Au niveau *RTL*, la démarche est encore plus difficile par manque d'informations précises sur le placement des éléments mémoires dans le *CI* ainsi que la difficulté liée à l'impossibilité de faire du *restitching*[6].

[6] *restitching* : terme très utilisé dans la littérature pour le chaînage des élément mémoire pour former

3.3.2 Objectif de la thèse pour le scan

La méthodologie de détection de signaux et d'éléments de mémorisation pour le scan *RTL* est basée sur les principes présentés plus haut et dans les travaux de *Aktouf et al.* [6].

L'objectif de notre travail concernant les chaînes de scan au niveau *RTL* est de développer un outil permettant de générer, à partir de la description *RTL* du circuit, des chaînes de scan optimisées en termes de surface, temps de test et consommation pour un circuit donné, tout en respectant un certain nombre de contraintes électroniques.

En effet, dans le cadre de cette thèse nous nous intéressons uniquement à l'optimisation de l'insertion des *chaînes de scan*. C'est-à-dire à déterminer un *stitching* (ordonnancement) optimal des éléments mémoires dans la chaîne de scan à partir de la description RTL du circuit.

Les techniques utilisées sont issues de l'optimisation combinatoire et de la recherche opérationnelle. Pour cela, nous proposons un modèle mathématique traduisant le problème électronique ainsi qu'une méthodologie de résolution qui a été implémentée et testée sur des circuits intégrés.

3.3.3 Etat de l'art

Plusieurs études ont été menées dans la littérature concernant des techniques d'optimisation de l'insertion des chaînes de scan. La plupart de ces techniques proposent l'optimisation de l'insertion des chaînes de scan après la synthèse logique du circuit au niveau *gate*. Cette section présente un rapide aperçu de ces techniques.

Nous présentons d'abord une vue d'ensemble de la littérature sur l'optimisation de l'insertion des chaînes de scan bas niveau. Ensuite, nous faisons un exposé plus détaillé des articles les plus significatifs pour le travail effectué dans cette thèse.

Les travaux existants et publiés sur l'optimisation de l'insertion des chaînes cherchent à résoudre le problème de l'insertion des chaînes de scan à partir d'informations extraites du *layout* après l'étape de synthèse. Il s'agit en réalité de faire d'abord une passe de placement et routage avec des chaînes aléatoires, par ordre alphabétique par exemple. Ensuite, à partir des informations de placement des cellules de scan on refait l'ordonnancement des éléments mémoires dans la chaîne de scan (*restitching*). Ceci revient à résoudre le problème suivant : étant donnée une certaine localisation des éléments mémoires scannés dans le *layout*, on cherche un ordre optimal pour chaîner ensemble ces éléments. Ce nouveau chaînage doit minimiser la longueur des interconnexions due à l'insertion des chaînes de scan afin d'éviter les erreurs signalées par les outils de placement et routage. La génération des chaînes de scan est répétée jusqu'à ce que les outils de placement et routage arrivent à insérer l'ensemble des chaînes de scan.

Les métriques utilisées sont : la distance « Manhattan » d'une cellule scannée à la suivante dans les travaux de *Hirech et al.* [35], ou la distance Manhattan d'une *pin* à la suivante dans les travaux de *Gupta et al.* [33], ou encore le plus court chemin au sens Manhattan de *output net* à *input pin* dans les recherches de *Berthelot et al.* [12].

une chaîne de scan. Il signifie ré-ordonnancer l'ensemble des éléments mémoires d'une chaîne de scan afin d'optimiser un ou plusieurs critères.

Ces travaux abordent le problème de façon à minimiser la surface additionnelle due à l'insertion des chaînes de scan. Pour cela, les auteurs cherchent à minimiser la longueur totale des interconnexions afin de minimiser l'impact de l'insertion des chaînes de scan nécessaires uniquement au test, sur la surface totale du *CI*.

Dans le rapport de recherche de *Boese et al.* [14], les auteurs présentent une vue d'ensemble bien détaillée des techniques citées plus haut ainsi qu'une bonne analyse critique. Malheureusement, les auteurs n'ayant pas fait d'expérimentations pratiques, cette étude n'a jamais été publiée dans un journal.

Dans les travaux de *Bonhomme et al.* [15], les auteurs cherchent à optimiser l'insertion des chaînes de scan toujours à bas niveau avec des contraintes de routage mais avec un nouvel objectif qui est de minimiser la consommation en puissance du circuit durant l'exécution des chaînes de scan, c'est-à-dire pendant les phases de chargement et déchargement des *vecteurs de test*.

Un autre type d'insertion des chaînes de scan est proposé par *Norwood et al.* dans [51]: le scan orthogonal. Il s'agit d'exploiter les informations à haut niveau afin de construire des chaînes de scan qui minimisent la surface additionnelle due à l'insertion de ces dernières. Pour cela les registres sont connectés comme suit : le premier bit du registre 1 est connecté au premier bit du registre 2. Le deuxième bit du registre 1 est connecté au deuxième bit du registre 2, etc. Cette approche diffère du scan traditionnel où le premier bit du registre 1 est connecté au deuxième bit du registre 1 qui est lui même connecté au troisième bit du même registre, etc. Cependant, ce travail ne présente aucune optimisation dans l'insertion du scan, il montre juste la faisabilité du scan orthogonal ainsi que le gain potentiel en surface par rapport au scan traditionnel.

Aucune de ces techniques ne cible l'insertion des chaînes de scan au niveau *RTL*. Les premiers travaux sur le scan RTL sont présentés par *Aktouf et al.* [6]. Cependant, la méthode proposée est uniquement basée sur la détection d'éléments mémoire et signaux au niveau *RTL*. Dans cette article, La sélection de bascules à scanné est basée sur des critères a-doc comme la notion de hiérarchie par exemple. Le but était de réduire le nombre final de cellules de scan ce qui permet de réduire la surface.

Notre travail vient en complément de cette étude. En effet, notre but est de déterminer un *stitching* optimal des éléments mémoire dans la chaîne de scan en se basant sur les idées et principes présentés par *Aktouf et al.* [6].

Dans ce sens, les travaux présentés par *Huang et al.* [36] montrent une tentative d'insertion des chaînes de scan au niveau *RTL*. La méthode proposée est basée sur un graphe de dépendance entre les éléments mémoire afin de construire la chaîne de scan. L'algorithme donné est d'une complexité exponentielle et peut être difficile à utiliser. En effet, l'exécution est faite à la main, aucun résultat de temps de calcul n'est donné. De plus, les expérimentations sont uniquement faites sur de petits circuits ne contenant pas plus de 100 éléments mémoires.

A ce jour, aucune technique, à notre connaissance, ne permet de gérer l'optimisation de l'insertion des *chaînes de scan* automatiquement au niveau *RTL*. Pour cela, notre partenaire industriel, la start-up *DeFacTo Technologies* a contacté le laboratoire *G-SCOP* pour une collaboration. Le but de cette collaboration est de développer un algorithme efficace

pour l'optimisation globale de l'insertion automatique des *chaînes de scan* au niveau *RTL*.

Pour cela nous avons développé un modèle qui prend en compte le problème des *chaînes de scan* au niveau *RTL*, et proposé un algorithme d'optimisation qui est implémenté aujourd'hui dans l'outil industriel *HiDFT-Scan*.

La section suivante présente la modélisation. Les méthodes de résolution ainsi que les résultats expérimentaux sont présentés dans le chapitre suivant.

3.4 Modélisation de l'insertion des chaînes de scan au niveau RTL

Dans cette partie nous allons présenter un modèle mathématique détaillé associé à notre problème, c'est-à-dire les données, les contraintes ainsi que la fonction objectif. Ce travail de modélisation était complexe dans le sens où beaucoup d'échanges ont été nécessaires avec notre partenaire industriel afin de comprendre le problème correctement et de fixer les objectifs à atteindre ainsi que les contraintes à respecter.

3.4.1 Données

Les données d'entrée du problème peuvent être résumées par les points suivants :

- La description *RTL* du circuit ; cette description est dans un des deux langages de haut niveau *VHDL* ou *Verilog*. De cette description on extrait la taille du circuit qui correspond au nombre d'éléments mémoire que nous notons n.
- Les chaînes existantes : en effet, le circuit peut contenir déjà un certain nombre de chaînes de scan. Cela est dû au fait que de nos jours on reprend souvent des blocs/bouts de circuits déjà existants qu'on assemble pour en créer de nouveaux. On parle par exemple des blocs d'*IP*: *Intellectual Property*.
- Le nombre ou la taille des chaînes finales exigées par l'utilisateur. Nous notons T la taille des chaînes exigées (c'est-à-dire le nombre d'éléments mémoire par chaîne).
- Les horloges qui pilotent le circuit.

3.4.2 Objectifs

Une grande partie du travail a consisté à déterminer des contraintes et plus exactement à distinguer entre les contraintes et les objectifs. Nous allons présenter dans cette partie les objectifs tels qu'ils ont été décrits par l'industriel. Dans le paragraphe suivant, nous décrirons les contraintes extraites de ces objectifs. Nous reviendrons par la suite sur les objectifs définitifs que nous avons retenus.

L'objectif global est de minimiser l'impact de l'insertion des chaînes de scan sur le circuit fonctionnel. Ceci se traduit par les objectifs suivants:

- Minimiser la surface additionnelle due à l'insertion des chaînes de scan dans le circuit.
- Minimiser la surconsommation engendrée par l'exécution des chaînes de scan.
- Minimiser la durée de la procédure de test.

- Minimiser l'impact des chaînes de scan sur la performance du circuit, c'est-à-dire minimiser l'impact du scan sur le *timing*[7]
- Maximiser le taux de couverture de fautes du circuit.
- Minimiser les changements d'horloges.

Certains de ces objectifs sont liés et d'autres sont en réalité des contraintes comme nous le verrons dans la suite.

3.4.3 Contraintes

En analysant les objectifs décrits dans la section 3.4.2, nous avons réalisé que certains sont plus pertinents en tant que contraintes. Nous décrivons dans cette partie cette analyse.

La durée de la procédure le test Minimiser le temps de la procédure de test revient à faire plusieurs chaînes et à équilibrer la taille des chaînes de scan en termes de nombre d'éléments mémoire contenus dans chaque chaîne. En effet, comme le circuit contient plusieurs chaînes de scan, et qu'elles sont chargées en parallèle avec les vecteurs de test, le temps d'exécution est proportionnel à la longueur de la plus longue chaîne. Donc plus on équilibre la taille les chaînes, plus on minimise cette longueur.

Exemple:

Supposons que le circuit contient trois chaînes de scan de taille $70, 50, 20$ le temps d'exécution est alors égal à 70 tops d'horloges. Si on équilibre des chaînes comme suit : $50, 50, 40$ le temps d'exécution est alors égal à 50 : on gagne 20 tops d'horloge. Ainsi on diminue la durée de la procédure de test.

Donc l'objectif de minimiser le temps de la procédure de test revient à la contrainte : il faut équilibrer la taille des chaîne de scan en terme de nombre d'éléments mémoires par chaîne.

Minimiser les changements d'horloges Pour les changements de fronts d'horloges : minimiser les changements de front revient à limiter l'insertion des *latchs*. C'est un type de bascules nécessaire pour connecter deux éléments mémoire qui sont sur différents fronts d'horloge.

Comme ces bascules sont très coûteuses en surface, les deux objectifs surface et changements de fronts sont liés d'une part. D'autre part, l'objectif minimiser le nombre de *latchs* peut donc s'exprimer en contrainte comme suit : ne pas chaîner des éléments mémoires sur différents front.

3.5 Formulation mathématique du problème

Le but de notre démarche est d'extraire le maximum d'informations à partir de la description *RTL* du circuit qui nous renseignent sur un éventuel rapprochement géométrique des éléments mémoires dans le futur *layout*. Ces données sont nécessaires afin de créer des algorithmes d'insertion des *chaînes de scan* optimales dans le sens où on limite l'impact

[7] *timing* : le chemin critique du circuit.

de l'insertion des chaînes de scan lors de l'étape de placement et routage. Pour cela on propose le modèle qui suit.

Intuitivement, un *CI* peut être représenté par un graphe. Cependant, plusieurs questions se posent : quels sont les ensembles de sommets et d'arêtes? Ce graphe est-il orienté ou non? Existe-il une valuation sur les arêtes? etc. Ce travail de thèse apporte des réponses à ces questions. Il est présenté dans la partie qui suit.

Après plusieurs formulations et tentatives de résolution [59]nous avons retenu la formulation présentée dans la suite de cette section. La modélisation proposée ici a été présentée aux *JGA 2008* [62] ainsi qu'a *Roadef 2009* [63].

3.5.1 Design Graph, G_D

On note le graphe $G_D = (V_D, E_D)$ représentant la description du circuit (que nous appelons *Design Graph*), tel que :

- l'ensemble des sommets V_D est l'ensemble des éléments mémoires, portes logiques et fils du circuit.
- E_D est l'ensemble des arêtes du graphe G_D. Une arête $e_{i,j}$ entre deux sommets V_{Di} et V_{Dj} représente une connexion électronique directe entre ces deux éléments.

Le graphe G_D est non orienté car le but est uniquement d'avoir des indications sur les liens existant entre les composants du circuit. Ce graphe vise à transcrire la description *RTL* du circuit en graphe.

On peut noter que G_D est *biparti*[8] : l'ensemble des sommets peut être divisé en deux sous-ensembles :

- V_{D^1} : les éléments mémoires, les portes logiques.
- V_{D^2} : les fils.

Remarque : La détection des éléments mémoire, portes logiques et fils est possible au niveau *RTL* grâce à une étape qu'on appelle *élaboration*. Les détails de cette étape sont expliqués au chapitre 4.

3.5.2 Proximity Graph, G_P

Comme la construction des *chaînes de scan* ne concerne que les éléments mémoires (bascules) du circuit, on a alors choisi de construire un deuxième graphe qui reflète uniquement le lien entre l'ensemble des éléments mémoire du circuit.

Pour cela, on construit le graphe G_P à partir du graphe G_D. Le graphe G_P sert à donner des informations sur *la proximité* des éléments mémoires du circuit. Cette information est utile pour le *stitching* des éléments mémoires.

Le graphe $G_P = (V_P, E_P)$ est composé de :

- V_P : l'ensemble des sommets du graphe G_P. Il correspond uniquement à l'ensemble des éléments mémoires du graphe G_D qui sont aussi les éléments mémoire du circuit.

[8] *biparti* : un graphe est dit biparti s'il existe une partition de son ensemble de sommets en deux sous-ensembles U et V tels que chaque arête ait une extrémité dans U et l'autre dans V. Un graphe est biparti si et seulement s'il ne contient pas de cycle impair.

- E_P : l'ensemble des arêtes du graphe G_P. Il existe une arête $e_{i,j}$ entre deux sommets V_{Pi} et V_{Pj} si et seulement s'il existe une connexion électronique entre ces deux sommets. Cette connexion n'est pas forcément directe mais elle ne doit pas traverser d'éléments mémoires.

A chaque sommet V_{Pi}, on associe les poids suivant :
- b_i : le nombre de bits que représente l'élément mémoire V_{Pi}.
- h_i : l'horloge à laquelle appartient l'élément mémoire V_{Pi}.

Remarques :
- Les chaînes existantes sont contractées et représentées par un sommet dans G_P avec un poids représentant le nombre de bits de la chaîne.
- De même les registres sont représentés par un seul sommet dans G_P et on garde l'information du nombre de bits.

A chaque arête $e_{i,j}$ on associe un poids $w_{(i,j)}$ qui reflète l'éventuelle proximité géométrique des éléments mémoires dans le futur circuit que nous appelons **proximité fonctionnelle**. Elle est définie comme suit :

Définition 3. *La **proximité fonctionnelle** est la distance logique entre deux éléments mémoires du circuit.*

La **proximité fonctionnelle** est la longueur d'une plus courte chaîne entre deux sommets correspondants a des éléments mémoires dans le graphe G_D. Elle correspond au nombre de portes logiques traversées par une plus courte chaîne entre deux sommets qui sont des éléments mémoires dans le graphe G_D.

Si deux éléments mémoires sont trop éloignés, cela signifie probablement qu'ils ne vont pas forcément être chaînés ensemble. Alors, on fixe un paramètre t qui correspond à la limite acceptable pour la longueur d'une chaîne entre deux éléments mémoires.

En d'autre termes, si toutes les chaînes entre les deux éléments mémoires ont une longueur supérieure à t alors les sommets qui leur correspondent dans le graphe G_P ne seront pas connectés.

En résumé, le graphe G_P est construit à partir du graphe G_D en prenant les éléments mémoires de G_D comme sommets de G_P et en insérant une arête entre deux sommets de G_P si et seulement si les éléments mémoire sont connectés et/ou s'il existe entre eux une chaîne de longueur inférieure à t dans G_D. La chaîne considérée ne doit pas traverser d'éléments mémoire.

Le poids $w_{(i,j)}$ des arrêtes de E_P sera utilisé pour la construction des *chaînes de scan*.

La figure 3.8 illustre un morceau de circuit avec le graphe G_P qui lui correspond. Le graphe contient quatre sommets qui correspondent aux quatre bascules (4, 6, 7 et 9). Le poids des différentes arêtes représente le nombre de portes traversées par un plus court chemin.

3.5.3 Construction du graphe G_P à partir de G_D

Le calcul des poids des arêtes $w_{(i,j)}$ du graphe G_P se fait en appliquant l'algorithme de parcours de graphe BFS (*Breadth First Search*) [53] sur le graphe G_D.

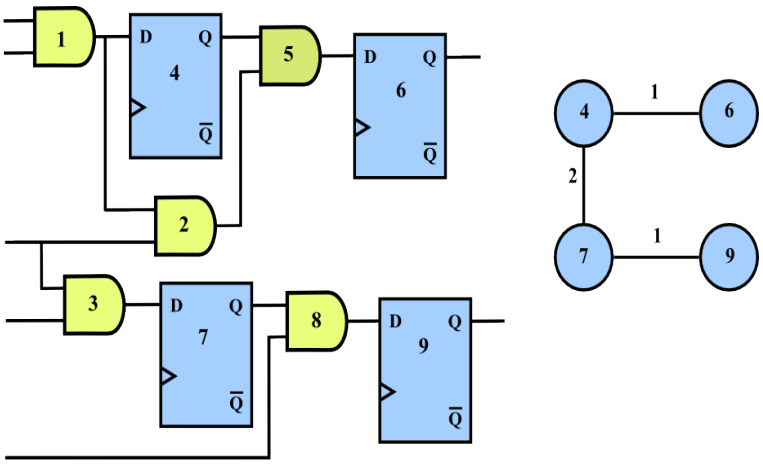

FIG. 3.8: Graphe G_P.

L'algorithme *BFS* effectue un parcours en largeur d'abord des sommets. On part d'un sommet (source), on visite les sommets qui sont plus proches de la source avant de visiter les sommets qui sont plus éloignés. Pour cela, nous appliquons l'algorithme BFS sur tous les sommets correspondant aux éléments de mémoires du graphe G_D et nous calculons la distance entre eux. Dans ce contexte, la "distance" est définie comme le nombre de portes logiques sur la chaîne du sommet source à n'importe quel autre élément mémoire.

Nous adaptons l'algorithme *BFS* afin de limiter la profondeur du parcours à une valeur t. Pour cela, nous ne considérons que les sommets à une profondeur moindre que la valeur limite t. On va donc considérer que tous les nœuds situés à une distance plus grande que la valeur limite t ne sont pas voisins dans G_P.

Le temps de calcul de cet algorithme est borné par le produit du nombre d'arêtes de G_D par le nombre d'éléments mémoires qui correspond au cardinal de V_P.

Dans la méthode globale, cette phase peut être considérée comme une sorte de pré-traitement. On peut noter que certaines optimisations se produisent déjà à cette étape, grâce à la recherche de plus courtes chaînes entre les éléments mémoires. Ces calculs aident à concevoir un ensemble précis de données pour la prochaine étape d'optimisation.

3.5.4 Critère d'optimisation

Afin de minimiser l'impact de l'insertion des chaînes de scan dans le circuit final et ne pas dégrader les performances de celui-ci, nous avons opté pour minimiser la longueur des interconnexions due à l'insertion des chaînes de scan dans le circuit. En effet, cette metrique permet de minimiser d'une part la surface car plus la connexion est longue plus cela prend de la place. D'autre part, elle minimise aussi la consommation qui est proportionnelle a la longueur de la connexion. Ainsi, le problème devient mono-objectif.

3.6 Conclusion

A l'heure actuelle, le *scan path* est une des méthodes les plus utilisées dans l'industrie de la micro-électronique. Il s'agit d'un ensemble de composants et de connexions électroniques qu'on rajoute au *CI* afin de faciliter l'étape de test post fabrication.

Traditionnellement, l'insertion du scan se fait après l'étape de synthèse. L'entreprise *DeFacTo Technologies* propose un nouveau flot, *le flot HiDFT* (cf. Figure 3.5), où l'insertion du scan s'effectue au niveau *RTL*. Par rapport aux approches traditionnelles, il s'agit de traiter le problème de test et de conception en vue du test des *CI* en amont dans le flot de conception d'un circuit intégré. Un tel processus profite de toutes les étapes d'optimisation durant la synthèse.

Dans ce travail, le problème auquel nous nous intéressons est le suivant : étant donnée la description *RTL* du circuit, il faut trouver un chaînage optimal de l'ensemble des éléments mémoires pour former les chaînes de scan. Ces dernières doivent couvrir tous les éléments mémoires du circuit afin de vérifier leur bon fonctionnement.

La première étape du travail a consisté en l'acquisition des différentes notions de micro-électronique liées au problème. Ensuite, nous avons abordé l'étape de formulation mathématique. Au départ le problème semblait multicritères vus les nombreux objectifs cités par l'industriel (optimiser le temps de test, la surface, l'équilibre des chaînes, la consommation, etc.). Mais après analyse, il s'avère que certains paramètres se présentent plus comme des contraintes à respecter et sont liés entre eux (comme l'équilibre des chaînes par exemple, qui influence le temps de test).

L'objectif retenu est de minimiser la longueur des interconnexions dues à l'insertion des chaînes de scan dans le circuit. En effet, il permet de minimiser à la fois la surface et la consommation.

Pour cela, l'idée est de chaîner ensemble les éléments mémoires qui ne sont pas trop distants dans le circuit pour ne pas créer de connexions entre éléments mémoires trop éloignés. Le problème du scan *RTL* est qu'à cette étape de conception, on ne connaît pas encore l'emplacement exact des éléments mémoires dans le circuit, contrairement au cas du scan traditionnel. Ceci rend le problème plus complexe. On essaye alors d'extraire le maximum d'informations à partir de la description RTL du circuit pouvant indiquer une proximité géométrique sur laquelle on se basera pour la construction des chaînes. Ainsi, on crée le moins possible de contraintes supplémentaires aux outils suivants dans le processus de réalisation d'un *CI*.

Pour prendre en charge tous les aspects électroniques (données, contraintes et objectifs industriels), nous avons opté pour une modélisation du problème sous forme de graphe avec différents attributs sur les sommets et arcs traduisant les contraintes électroniques. Cette modélisation vise à optimiser le choix de paramètres influençant la surface, le temps de chargement et déchargement des chaînes de scan, les horloges et la consommation en puissance du circuit. Le but est d'obtenir un chaînage optimal des éléments mémoire qui constitueront les futures chaînes de scan.

La méthode de résolution ainsi que les résultats numériques qui s'y rapportent sont décrits en détail dans le chapitre 4.

Chapitre 4

Solution Algorithmique pour l'insertion des chaînes de scan

Sommaire

4.1 Introduction 63
4.2 Position du problème 64
4.2.1 Etude bibliographique 65
4.2.2 Analyse de la complexité du problème 65
4.2.2.1 Etude des différents cas en fonction de la taille des chaînes 66
4.2.3 Passage de plusieurs chaînes à une seule 66
4.3 Analogie entre le problème des chaînes de scan et celui du TSP 68
4.4 Résolution heuristique du problème du voyageur de commerce 69
4.4.1 Quelques définitions 70
4.4.2 Une 2-approximation pour le TSP 71
4.4.3 L'heuristique de Christofidès 71
4.5 Méthode de résolution pour l'insertion des chaînes de scan optimales 72
4.6 Résultats numériques 74
4.6.1 Description des instances, benchmarks et open cores 74
4.6.2 Construction des graphes et optimisation du scan 74
4.6.3 Protocole expérimental 76
4.7 Conclusion 79

4.1 Introduction

On a vu dans le chapitre précédent que l'insertion de *chaînes de scan* est une technique qui facilite le test des circuits intégrés. Elle modifie le schéma du circuit, en y ajoutant des composants électroniques supplémentaires. Dans la méthode du *scan path*, on chaîne les éléments mémoire, ou bascules, du circuit les uns aux autres.

Pour cela, traditionnellement, on remplace les bascules du circuit par des *bascules scan* et on les relie entre elles de manière à établir un accès série (registre à décalage à l'échelle du circuit) appelé chaîne de scan. On demande aux chaînes de couvrir tout le circuit pour vérifier le bon fonctionnement de chaque composant.

Nous rappelons que le principal inconvénient de cette technique est la surface additionnelle utilisée, due à l'insertion des bascules *scan* ainsi qu'aux connexions supplémentaires nécessaires au chaînage des éléments mémoires. En particulier, si on réalise des connexions entre éléments mémoires trop éloignés dans le circuit, on crée beaucoup de contraintes supplémentaires pour les outils qui interviennent plus tard dans le processus de conception. Ce qui peut entraîner un coût en silicium, des dégradations des performances, etc.

Le temps total du processus de test et en particulier de l'exécution des *chaînes de scan* (chargements et déchargements des vecteurs de test) engendre aussi d'importants coûts qu'on doit prendre en compte lors de la phase de conception du circuit. En effet, si le circuit à tester est de taille moyenne à importante (quelques milliers ou dizaines de milliers d'éléments mémoires), l'implémentation d'une chaîne de scan unique nécessiterait le décalage de toutes les données à travers cette chaîne. Ceci peut être très coûteux en temps vu le volume de production des circuits intégrés généralement conséquent. Une solution consiste à diviser la chaîne de scan en plusieurs chaînes et à les introduire en parallèle dans le circuit. Mais il ne faut pas négliger le fait que chaque chaîne possède ses propres ports d'entrées/sorties dans le circuit, ce qui affecte beaucoup la surface.

L'objectif de ce travail est de développer un outil permettant de générer automatiquement des chaînes de scan optimales en termes de surface et temps de test pour un circuit donné, tout en respectant un certain nombre de contraintes électroniques. Les techniques utilisées sont issues de l'optimisation combinatoire et de la recherche opérationnelle.

Dans ce chapitre nous présentons un état de l'art sur les problématiques de recouvrement des sommets d'un graphe par des chaînes suivi d'une analyse de complexité en fixant la taille des chaînes en termes de nombre de sommets par chaînes. Nous présentons ensuite l'algorithme de *stiching* pour le chaînage des éléments mémoires qui est basé sur les heuristiques classiques du problème du voyageur de commerce. Enfin, nous présentons des résultats numériques obtenus avec cet algorithme ainsi qu'une comparaison entre nos résultats et la méthode traditionnelle d'insertion du scan au niveau *gate*.

4.2 Position du problème

Le chaînage d'éléments mémoires dans le circuit revient à trouver une couverture des sommets V_p du graphe de proximité $G_p = (V_p, E_p)$ par des chaînes de longueur minimale (au sens de la longueur des arêtes). De plus, les chaînes doivent couvrir tous les sommets du graphe et être deux à deux disjointes (sommets disjoints). Le nombre de sommets par chaînes doit être le même (les chaînes doivent être équilibrées, afin de charger en parallèle les vecteurs de test). Dans la suite du travail nous appelons ce problème : problème de couverture des sommets par des chaînes de taille T.

4.2.1 Etude bibliographique

L'objectif est de trouver une couverture des sommets du graphe $G_p = (V_p, E_p)$ par des chaînes, tel que chaque sommet appartient à une et une seule chaîne.

Le nombre total de chaînes doit être égal au nombre de chaînes exigées par l'utilisateur. La taille de chaque chaîne (en nombre de sommets) doit être inférieure ou égale à T, nous rappelons que T est égal aux nombre d'éléments mémoires du circuit divisé par le nombre de chaînes de scan exigées par l'utilisateur.

Plusieurs personnes s'intéressent à ce type de problème. L'article de *Masuyama et al.* [47] rappelle les principaux résultats. Il traite de la complexité du problème de recherche de chaînes disjointes dans un graphe. Les auteurs appellent ce problème *chains packing problem*. Il s'agit de la recherche de chaînes disjointes de tailles k dans un graphe, où les k-chaînes sont des chaînes simples de longueur k (c'est-à-dire des chaînes contenant k arêtes) dans un graphe non orienté. Les auteurs abordent les deux cas : celui des sommets disjoints et celui des arêtes disjointes.

Des algorithmes polynomiaux sont donnés pour le problème d'arêtes disjointes lorsque le graphe est un arbre, et pour un graphe quelconque pour $k = 2$. Les auteurs montrent que le problème de sommets disjoints pour des graphes quelconque est $\mathcal{NP}$-Complet, même si le degré maximum du graphe est trois et que $k = 2$. De la même façon, ils montrent que le problème arêtes disjointes est $\mathcal{NP}$-Complet, même si le degré maximum est quatre et que $k = 3$.

D'autres auteurs se sont intéressés à un problème similaire. Dans *Teypaz et al.* [56], les auteurs s'intéressent au problème suivant : étant donnés deux entiers a et b, ils définissent une (a, b)-décomposition d'un graphe $G = (V, E)$ par une partition de E en chaînes dont la longueur de chaque chaîne se trouve entre a et b. Dans ce cas les chaînes doivent être simples, mais pas nécessairement élémentaires. En d'autres termes une (a, b)-décomposition correspond à une partition $E_1, ..., E_k$ de E avec $a \leq |E_i| \leq b$ pour tout i, et telle que chaque sous-graphe induit (V_i, E_i) admet un chemin eulérien. Dans cet article, les auteurs étudient la complexité du problème de la (a, b)-décomposition. Ils donnent une caractérisation des graphes $(2, b)$-décomposables, à partir de laquelle il est possible de décider en temps linéaire de la $(2, b)$-décomposition. Quand $a \geq 3$, ils identifient deux classes de graphes ; les arbres et les graphes eulériens, pour lesquels le problème de (a, b)-décomposition reste polynomial. Cependant, ils montrent que le problème de $(3, 3)$-décomposition est $\mathcal{NP}$-Complet, même sur les graphes bipartis.

4.2.2 Analyse de la complexité du problème

On s'intéresse ici au problème de couverture des sommets du graphe $G_p = (V_p, E_p)$ par des chaînes sommets disjoints.

Le paramètre T qui représente la taille des chaînes en termes de nombre de sommets dans chaque chaîne, est une constante dans cette section. On considère qu'il n'intervient donc pas dans la taille de l'instance du problème pour le calcul de la complexité.

4.2.2.1 Etude des différents cas en fonction de la taille des chaînes

Si $T = 1$ le problème est trivial puisqu'on cherche une chaîne par sommet.

Si $T = 2$ on cherche à couvrir tous les sommets du graphe $G_p = (V_p, E_p)$ par des chaînes contenant chacune 2 sommets. On veut donc trouver un couplage parfait. Comme notre graphe est pondéré et qu'on souhaite obtenir des chaînes de longueurs minimales, ce problème revient à chercher un couplage parfait de poids minimum.

Ce problème est connu et très étudié dans la littérature. Un algorithme trouvant un couplage parfait de poids minimum est proposé dans [25]. Ainsi, le problème est polynomial pour $T = 2$. Si le graphe n'admet pas de couplage parfait alors, on cherche un couplage maximum de poids minimum.

Si $T = 3$ on cherche à couvrir tous les sommets du graphe, $G_p = (V_p, E_p)$ par des chaînes contenant chacune 3 sommets. Ceci revient à trouver une couverture des sommets par des chaînes de longueurs minimales contenants 3 sommets. Ce problème est $\mathcal{NP}$-Difficilecomme démontré dans [47]. Donc à partir de $T = 3$ le problème est $\mathcal{NP}$-Difficile.

Si $3 \leq T \leq n$ On cherche à couvrir les sommets du graphe $G_p = (V_p, E_p)$ par une chaîne de longueur minimum (au sens du poids des arêtes) qui passe par tous les sommets une et une seule fois. Ceci revient au célèbre problème du voyageur de commerce (*TSP* pour *Traveling Salesman Problem*), problème $\mathcal{NP}$-Complet [25].

Dans la suite de ce travail nous allons ramener le problème de recherche de *chaîne de scan* à la recherche de chaînes hamiltoniennes dans le graphe $G_p = (V_p, E_p)$.

En effet, il existe de nombreuses heuristiques très efficaces dans la littérature pour résoudre le problème du *TSP* qui peuvent être adaptées à notre problème.

Dans la section suivante, nous proposons deux méthodes pour passer d'une seule chaîne de scan à plusieurs. Nous ramenons ensuite le problème de recherche d'une seule chaîne de scan au problème du *TSP*, suivi d'heuristiques de résolution du problème du TSP. Enfin, nous présentons l'algorithme de chaînage implémenté dans l'outil industriel pour la recherche de *chaînes de scan* optimales.

4.2.3 Passage de plusieurs chaînes à une seule

Afin de satisfaire les demandes industrielles en termes de nombre de chaînes et tailles des chaînes, tout en respectant l'ensemble des contraintes liées au problème, on propose les deux stratégies décrites dans les paragraphes suivants.

Partition du graphe

La première stratégie consiste à partitionner le graphe $G_p = (V_p, E_p)$ de sorte que le nombre de partitions soit égal au nombre de chaînes de scan exigées par l'utilisateur. Il faut que le nombre de sommets dans chaque partition soit égal. Ensuite, dans chaque partie, on cherche une chaîne hamiltonienne de longueur minimum.

Le problème de partition de graphe est un problème bien étudié dans la littérature. Plusieurs ouvrages et articles traitent du problème ; on peut citer le rapport technique [22] qui résume les principales idées proposées pour résoudre le problème.

Pour plus de détails, la thèse de *Bichot* [13] est entièrement consacrée au problème de partition de graphe. Cette thèse présente un état de l'art, ainsi qu'une bonne analyse critique des méthodes proposées dans la littérature.

Elle comporte aussi une présentation détaillée de trois méta-heuristiques utilisées pour résoudre le problème : le *recuit simulé*, les algorithmes de *colonies de fourmis* et les algorithmes *génétiques*.

Dans ce travail, nous avons utilisé la bibliothèque *Metis* [37] pour partitionner les graphes. Elle a été créé par *George Karypis* et *Vipin Kumar* de l'Université du Minnesota aux États-Unis d'Amérique. Toutes les informations sur *Metis* sont disponibles à l'adresse : `glaros.dtc.umn.edu/gkhome/views/metis`.

Metis est l'une des bibliothèques les plus utilisées pour le partitionnement de graphe. Le code de cette bibliothèque est entièrement disponible. Elle est écrite en *langage C* et possède une interface facile d'utilisation.

Elle intègre deux logiciels : *pMetis* et *kMetis*. Le premier logiciel, *pMetis*, utilise un algorithme multi-niveaux récursif de bissection. Le second logiciel, *kMetis*, est implémenté avec un algorithme de k-partitionnement.

C'est le deuxième algorithme que nous avons utilisé pour partitionner le graphe $G_p = (V_p, E_p)$ en k partitions. On cherche ensuite une chaîne de scan dans chaque partie pour obtenir k chaînes de scan au total. Le but est de faire des parties de même taille, au sens du nombre de sommets par partition afin d'équilibrer la taille des chaînes de scan pour le temps d'exécution de la procédure de test (chargement des vecteurs de test en parallèle dans les chaînes). Plus d'informations sur le paramétrage et l'équilibrage des partitions peuvent être trouvées dans le rapport de *Zaourar* [59].

Recherche d'une seule chaîne Il s'agit de chercher une chaîne hamiltonienne de longueur minimale qui passe par tous les sommets du graphe $G_P = (V_P, E_P)$. Ensuite, on découpe cette chaîne afin d'obtenir le nombre de chaînes exigé par l'utilisateur. Pour cela, on divise le nombre total de sommets par le nombre de chaînes, on obtient la taille des chaînes (la meilleure taille afin d'équilibrer au mieux la taille des chaînes en terme de nombre de sommets par chaîne) ; on prend la partie entière supérieure si le nombre n'est pas entier. On découpe ensuite en fonction de ce chiffre comme suit : les T premiers sommets, puis les T suivant, etc. On supprime ensuite les sommets en plus des premières chaînes.

Exemple : supposons qu'on a 16 sommets dans le graphe $G_P = (V_P, E_P)$ et qu'on veut 3 chaînes. Le nombre de sommets qu'on veut avoir est de 6 sommets dans chaque chaînes. On construit alors 3 chaînes contenant chacune 6 sommets puis on supprime un sommet des deux premières. On obtient les chaînes de tailles : 5, 5, 6.

En résumé dans les deux stratégies le problème se ramène à la recherche d'une chaîne

hamiltonienne de longueur minimale. Dans la suite de ce travail, nous allons montrer que le problème de chaînes de scan revient au célèbre problème du voyageur de commerce (TSP). Nous allons ensuite présenter les deux heuristiques les plus connues pour résoudre ce problème. Nous présentons dans la dernière partie les implémentations et tests qui ont été réalisés, ainsi que les comparaisons des différents résultats obtenus.

4.3 Analogie entre le problème des chaînes de scan et celui du TSP

Nous montrons dans cette section, l'équivalence entre le problème du TSP qui est $\mathcal{NP}$-Complet [26] et le problème de chaîne de scan.

Modèle à une seule chaîne de scan: P_1

Instance : $G_P = (V_P, E_P)$ non orienté, $w_{(i,j)}$ poids sur les arêtes, avec :

V_P = les éléments mémoires.

$e_{i,j}$ entre deux sommets V_{Pi} et V_{Pj} si et seulement s'il existe une connexion électronique entre ces deux sommets.

$w_{(i,j)}$ = le poids de l'arête $e_{i,j}$, qui correspond à la distance entre les deux sommets V_{Pi} et V_{Pj}.

Une solution réalisable : une chaîne de scan, c'est-à-dire une chaîne qui passe par tous les sommets de G_P.

Objectif : une chaîne de scan de longueur minimale (longueur au sens des poids des arêtes).

Problème du voyageur de commerce (TSP): P_2

Instance : un graphe complet, $G = (V, E)$ avec des poids sur les arêtes

Une solution réalisable **:** un cycle hamiltonien dans G.

Objectif **:** Trouver un cycle hamiltonien dans G de longueur minimum.

Le problème de recherche d'une chaîne de scan dans le graphe G_P se réduit au problème du voyageur de commerce comme décrit ci-dessous. Le problème du voyageur de commerce étant $\mathcal{NP}$-Difficile, le problème de recherche d'une chaîne de scan est lui aussi $\mathcal{NP}$-Difficile.

Soit I_1 une instance pour le problème P_1, c'est-à-dire un graphe $G_P = (V_P, E_P)$. On ajoute le sommet V_{P0} comme sommet fictif. On le relie à l'ensemble des sommets V_p avec des arêtes de poids 0. On rajoute ensuite des arêtes de grand poids au graphe pour que celui ci soit complet.

Soit C_H une solution au problème P_2 du voyageur de commerce, c'est-à-dire un cycle hamiltonien de longueur minimale qui passe par tous les sommets de G_P. Soit V_{P0} le premier sommet du cycle et V_n le dernier sommet du cycle C_H. Si on supprime les arêtes e_{V_{P0},V_i} pour $i = 1, ..., n$ du cycle C_H on obtient alors une chaîne de scan de longueur min donc une solution pour le problème P_1.

Remarque:
Si le graphe $G_p = (V_p, E_p)$ n'est pas complet, on peut le compléter en rajoutant des raccourcis entre les sommets V_p comme suit : si l'arête $e_{V_{p_i},V_{p_j}}$ entre deux sommets V_{p_i} et V_{p_j} n'existe pas, on la rajoute avec un grand poids $w_{V_{p_i},V_{p_j}}$.

4.4 Résolution heuristique du problème du voyageur de commerce

Le problème du voyageur de commerce a suscité un grand intérêt du fait de son aspect très pratique. En 1995, *G.Reinelt* de l'université d'Heidelberg a édité une bibliothèque d'exemples de problèmes de *TSP* appelée *TSPLIB* que l'on peut trouver à l'adresse internet suivante :
`http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`.

La plupart des instances proposées correspondent à des lieux géographiques (capitales européennes, grandes villes allemandes, etc).

Le fait que le *TSP* soit un problème $\mathcal{NP}$-Difficile ne signifie pas que l'on ne puisse pas chercher à le résoudre de manière exacte. La méthode d'énumération totale fournit une solution optimale pour des problèmes de petite taille mais elle est totalement inefficace pour les problème de taille réelle puisque le nombre de cycles hamiltoniens dans un graphe complet est égal à $(n-1)!/2$.

Il existe des méthodes qui permettent d'éviter l'énumération complète de l'ensemble des cycles hamiltoniens. Ces approches sont fondées sur la *programmation linéaire* (voir chapitre 1) et permettent de résoudre de manière exacte le problème du *TSP* sur un grand nombre d'instances de la *TSPLIB*.

Une équipe de l'université de Rice au Texas (*D. Applegate*, *B. Bixby*, *V. Chvatal*, *W. Cook*) a développé un logiciel nommé *Concorde* qui permet de résoudre à l'optimum des instance du *TSP*. En 2004, le record du monde de la plus grande instance résolue à l'optimal a été battu grâce à ce logiciel puisque le tour de coût minimum reliant 24978 agglomérations de la *Suède* a pu être trouvé. Plus récemment, *Cook et al.* [8] ont réussit à atteindre le nombre de 85900 sommets. C'est le problème de plus grande taille jamais résolu.

Trouver la solution de tels problèmes est un véritable challenge. D'une part les méthodes s'appuient sur des résultats mathématiques profonds et difficiles de la théorie des graphes et de programmation mathématique. D'autre part, la mise en œuvre de ces logiciels scientifiques est également très complexe et nécessite beaucoup de temps. Dans ce travail nous n'abordons pas les approches de résolution exactes du problème du TSP en raison de la taille du problème et du temps d'exécution.

Il existe de nombreux algorithmes non exacts pour résoudre le problème du TSP. Dans ce paragraphe, nous décrivons les deux heuristiques les plus classiques pour résoudre le *TSP*. Nous les avons choisies car elles reposent sur des idées simples et sont relativement faciles à mettre en œuvre.

De plus, elles ont été testées sur plusieurs instances (comme celles de la *TPSLIB*) et produisent généralement de bons résultats en termes de qualité de solution et de temps

d'exécution. Il s'agit d'une 2-approximation et de l'heuristique Christofidès. Dans le paragraphe suivant, nous introduisons d'abord les notions d'arbre, arbre couvrant et couplage. Ensuite, nous décrivons sommairement les heuristiques avec les principales idées et étapes.

4.4.1 Quelques définitions

Définition 4. *Un graphe G est **connexe** s'il existe au moins une chaîne entre une paire quelconque de sommets de G.*

Définition 5. *Soit $G = (X, A)$ un graphe non orienté. Une **chaîne eulérienne** est une chaîne empruntant une fois et une fois seulement chaque arête de G.*

Un **cycle eulérien** est une chaîne eulérienne dont les extrémités coïncident.
Un graphe possédant un cycle eulérien est appelé *graphe eulérien.*

Arbres et Arbres couvrants de poids minimum

Définition 6. *Un **arbre** est un graphe connexe sans cycle.*

Définition 7. *Soit un graphe non orienté $G = (V, E)$, connexe, avec des arêtes de poids positifs P. Soit un **arbre couvrant** $A = (V, B)$ défini comme graphe partiel de G avec un ensemble d'arêtes B. Son poids (ou coût) total est : $P(A) = \sum_{a \in B} P(a)$. On dit que A est un **arbre couvrant de poids minimal** de G si $P(A)$ est minimal parmi les poids de tous les arbres couvrants possibles de G.*

Si toutes les arêtes sont de poids différents, l'arbre couvrant de poids minimal est unique. Plusieurs algorithmes ont été proposés pour résoudre ce problème. Les plus simples sont les algorithmes de *Prim* et de *Kruskal* [53].

Couplages et couplage maximal

Définition 8. *Étant donné un graphe simple non orienté $G = (V, E)$, **un couplage**, appelé aussi (matching) est un sous ensemble d'arêtes $M \subset E$ tel que deux arêtes quelconques de M ne sont pas adjacentes.*

Définition 9. *Soit un graphe G, un **couplage parfait** dans G est un ensemble M d'arêtes tel que chaque sommet du graphe G est incident à une et une seule arête de l'ensemble M.*

Un sommet $i \in V$ est dit *saturé* par le couplage $M \subset E$ s'il existe une arête de M incidente à i.
Un **couplage maximal** est un couplage de cardinalité maximale.

4.4.2 Une 2-approximation pour le TSP

L'idée principale de cette méthode repose sur la transformation d'un arbre couvrant de poids minimum du graphe G en un graphe eulérien.

On construit d'abord un arbre couvrant de coût minimum A_{min} de G, en appliquant l'algorithme de *Prim* ou *Kruskal*. Ensuite, on duplique toutes les arêtes de A_{min} afin d'obtenir un graphe eulérien. Puis, on cherche un cycle eulérien C de ce graphe. Enfin, on construit un cycle hamiltonien H qui passe par tous les sommets de G dans l'ordre de leur première occurrence dans C.

La complexité de cette heuristique est $O(m + nlogn)$ où n est le nombre de sommets et m le nombre d'arêtes du graphe. La performance de cette heuristique est de 2 fois l'optimum dans le pire des cas [19].

La figure 4.1 présente un exemple sur un petit graphe. Elle montre d'abord l'arbre de poids minimum, le graphe eulérien et enfin le tour obtenu en partant du sommet 1.

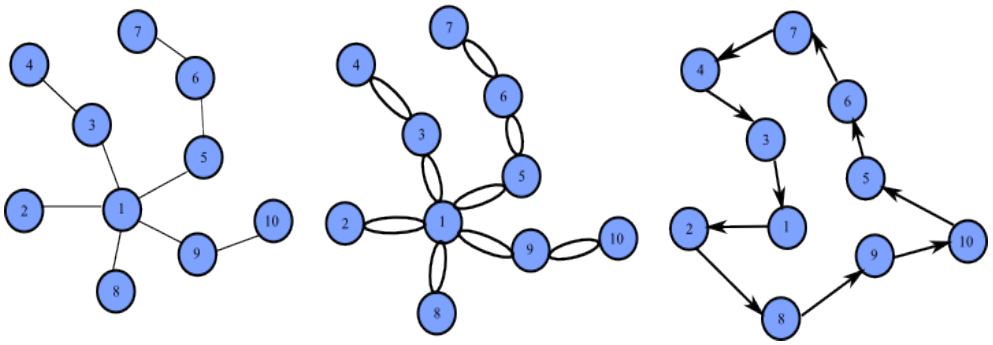

FIG. 4.1: Exemple d'application de la 2-approximation.

4.4.3 L'heuristique de Christofidès

L'heuristique de *Christofidès* a été proposée en 1976 [19]. D'abord, nous construisons l'arbre de coût minimum A_{min}. Ensuite, considérons l'ensemble des sommets de cet arbre ayant un *degré* [1] impair. Sur cet ensemble qui est de cardinalité paire, nous construisons le *couplage parfait* M_{min} de poids minimum. En ajoutant M_{min} à A_{min}, en remplaçant l'arête (i, j) par deux arêtes multiples d'extrémités i et j dans le cas ou (i, j) appartient simultanément à A_{min} et à M_{min}, on obtient un graphe où tous les sommets sont de degré pair. On peut alors construire un parcours eulérien dans ce nouveau graphe.

Nous construisons alors à partir de ce parcours eulérien, un cycle hamiltonien H de la même manière que dans le cas du parcours double de l'arbre de coût minimum décrit précédemment. La complexité est $O(n^3)$ où n est le nombre de sommets du graphe. La

[1] *degré:* le degré d'un sommet est le nombre d'arêtes incidentes à ce sommet. Le nombre de sommets de degré impair est toujours pair.

performance de cette heuristique est de une fois et demie l'optimum dans le pire des cas [19].

4.5 Méthode de résolution pour l'insertion des chaînes de scan optimales

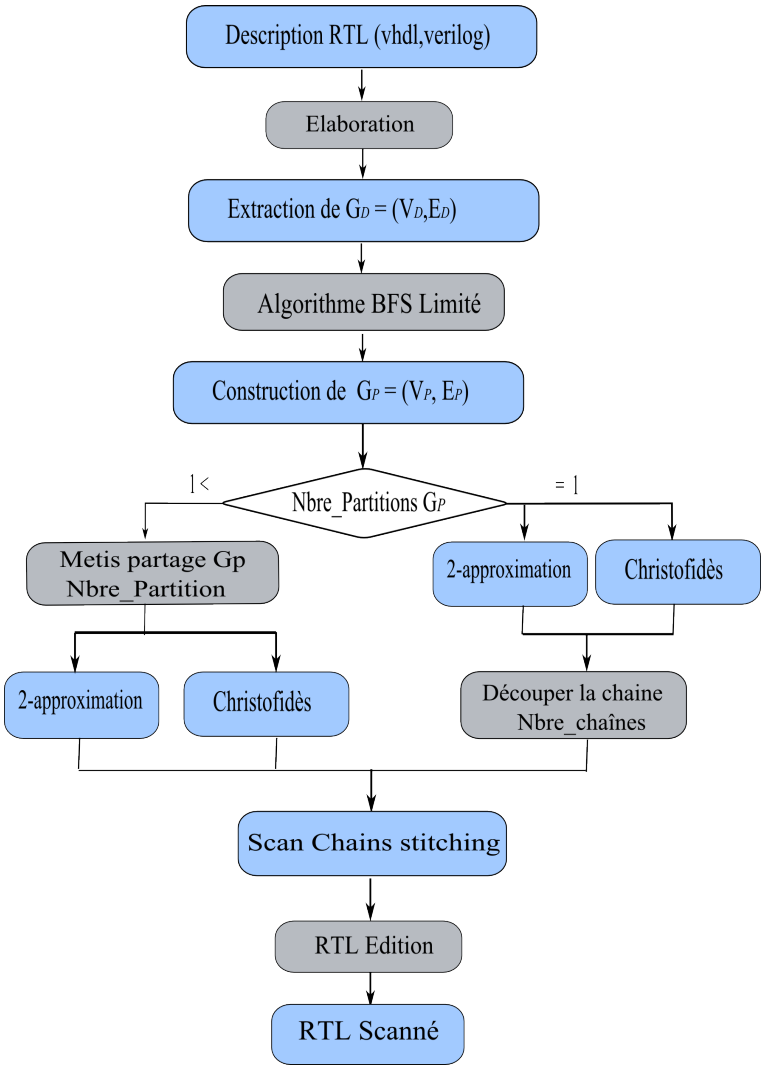

Fig. 4.2: Algorithme de stitching des chaînes de scan RTL.

Afin de déterminer les chaînes de scan qui répondent au mieux aux besoins des différents concepteurs de circuits intégrés, tout en respectant au maximum les contraintes et restrictions électroniques liées au problème, *DeFacTo* a mis au point l'outil **HiDFT**. Ce dernier contient un algorithme nommé : *Algorithme de stitching* pour l'insertion automatiques de chaînes de scan optimales au niveau *RTL* que nous avons développé en collaboration. Cet algorithme est présenté dans la section suivante.

L'algorithme de stitching

L'organigramme de la figure 4.2 présente les étapes de l' *algorithme de stitching*. Il a été implémenté en *C++*. Plusieurs librairies extraites de *Boost*[2] ont été utilisées pour la représentation des graphes G_D et G_p ainsi que pour les algorithmes classiques d'optimisation combinatoire comme la recherche d'arbres couvrants de poids minimum ou de couplages par exemple. Plus de détails sur cette bibliothèque sont disponibles sur : `http://www.boost.org/`.

Pour des raisons de confidentialité, nous ne décrivons que sommairement chacune des étapes dans le paragraphe suivant. Nous parlerons ensuite des évaluations faites pour une solution trouvée à partir de cet algorithme dans la section suivante.

D'abord, on effectue un pré-traitement appelé *Elaboration* comme suit : à partir de la description *RTL* du circuit (*VERILOG* ou *VHDL*), on élabore« *une pré-netlist* » (voir chapitre 2) générique. Il s'agit de repérer les différents liens entre les portes logiques, les éléments mémoires et les connexions électroniques du futur circuit. On ne traite pas les notions de temps critique. A partir de cette « *pré-netlist* » là on extrait le graphe $G_D = (V_D, E_D)$ représentant le circuit.

On applique ensuite l'algorithme de *BFS* avec profondeur limitée pour la construction du deuxième graphe $G_P = (V_P, E_P)$ représentant la proximité fonctionnelle. On calcule les poids $w_{(i,j)}$ qui permettent de pondérer les différentes connexions électroniques du circuit. Ce graphe permet d'estimer une probable proximité géométrique des éléments mémoires dans le *Layout*.

L'utilisateur peut ensuite choisir une des deux stratégies de résolution proposées pour la construction des chaînes de scan à partir du graphe $G_P = (V_P, E_P)$ en fonction du nombre de chaînes exigées par l'utilisateur : soit on partitionne le graphe G_P et on cherche une chaîne hamiltonienne de longueur minimum (longueur des arêtes) par partition, soit on cherche une seule chaîne qu'on découpe par la suite.

Les chaînes hamiltoniennes de longueur minimum sont trouvées en appliquant une des deux heuristiques pour la résolution du problème du TSP, la 2-approximation ou Cristofidès. On obtient ainsi *un stitching* (ordonnancement) optimal des éléments mémoires dans les chaînes de scan.

On passe ensuite à la dernière étape : l'édition dans le code *RTL* du circuit (*RTL edition*). Il s'agit d'insérer dans le code *RTL* d'origine du circuit les constructions *RTL* supplémentaires nécessaires à l'implémentation du scan. Une fois édité, le circuit est scanné et est toujours en *RTL*.

[2]Boost : est un ensemble de bibliothèques logicielles écrits en *C++*, et délivré sous licence libre.

4.6 Résultats numériques

Pour permettre aux différentes équipes de chercheurs de travailler sur les même données à travers la communauté scientifique, des jeux d'essais (*benchmarks*) existent pour certaines étapes du processus de conception. Il existe par exemples les *ISCAS* pour l'étape de synthèse, et *ITC 99* qui propose des circuits de référence pour le test des circuits intégrés.

Dans cette section, nous décrivons d'abord les instances testées et ensuite les résultats obtenus avec notre *algorithme de stitching*.

4.6.1 Description des instances, benchmarks et open cores

Les jeux de données utilisés dans cette section sont extraits de la base de donnée des opencores disponible à l'adresse :
`http://cvs.opencores.org/forum,Cores,0,1097`) et des benchmarks *ITC 99* disponibles sur :
`http://www.cerc.utexas.edu/itc99-benchmarks/bench.html`.

Le tableau 4.1 présente les circuits qui ont été sélectionnés selon un certain nombre de caractéristiques. Ces données ont un contexte de test des circuits intégrés. La première ligne donne le nom du circuit (b01,...,b19) pour les *ITC99* et les trois opencore (simple-spi, Biquad, Ac-97). Les lignes suivantes donnent le nombre de bascules (nb-ff) et le langage *RTL* (vhdl ou verilog) pour chaque circuit.

Ces circuits ont été très utilisés dans la littérature du test des circuits intégrés ; pour chaque jeu de données, un ensemble de publications (un descriptif complet du circuit avec son utilité) est présenté sur le site web. Ces jeux de données ont été utilisés également de nombreuses fois dans le cadre d'articles sur le test et la conception en vue du test des *CI*.

Design	b01	b03	b04	b05	b06	b07	
nb-ff	5	30	66	34	9	49	
Langage	vhdl	vhdl	vhdl	vhdl	vhdl	vhdl	
Design	b09	b10	b11	b12	b13	b14	
nb-ff	28	17	31	121	53	245	
Langage	vhdl	vhdl	vhdl	vhdl	vhdl	vhdl	
Design	b15	b17	b18	b19	simple-spi	Biquad	Ac-97
nb-ff	449	1415	3320	6642	132	204	2289
Langage	vhdl	vhdl	vhdl	vhdl	verilog	verilog	verilog

TAB. 4.1: Description des circuits.

4.6.2 Construction des graphes et optimisation du scan

Les résultats de la première phase (extraction du graphe $G_P = (V_P, E_P)$) sont décrits dans le tableau 4.2.

Design	nb-ff	$\mid V_D \mid$	$\mid E_D \mid$	$\mid V_P \mid$	$\mid E_P \mid$	CPU_G_D_to_G_P
b01	5	118	186	5	10	$<$ 1 ms
b03	30	450	806	30	435	$<$ 1 ms
b04	66	1111	1857	66	2145	$<$ 1 ms
b05	34	3795	6455	34	561	$<$ 1 ms
b06	9	158	263	9	36	$<$ 1 ms
b07	49	1125	1997	49	1176	$<$ 1 ms
b09	28	401	719	28	378	$<$ 1 ms
b10	17	525	947	17	136	10 ms
b11	31	1094	1845	31	465	20 ms
b12	121	3879	6993	121	7260	200 ms
b13	53	633	1042	53	1378	20 ms
b14	245	27269	46778	245	29890	3 s
b15	449	33179	56798	449	100576	6 s
b17	1415	100358	171759	1415	697990	30 s
b18	3320	271320	462326	3320	2330318	3 min
b19	6642	531161	906201	6642	7061713	5 min
simple-spi	132	1427	8646	132	8778	100 ms
biquad	204	357	20706	204	20910	350 ms
Ac97	2289	18727	2381495	2289	707911	30 s

TAB. 4.2: Caractéristiques des graphes.

Les colonnes 1 et 2 donnent le nom (*Design*) et la taille en nombre d'éléments mémoires (*nb-ff*) des circuits testés. Les colonnes suivantes donnent la taille du design graphe G_D avec respectivement le nombre de sommets $\mid V_D \mid$ (colonne 3) et le nombre d'arêtes $\mid E_D \mid$ (colonne 4). Suivent les colonnes 5 et 6 pour les paramètres du proximity graph G_P avec le nombre de sommets $\mid V_P \mid$ et le nombre d'arêtes $\mid E_P \mid$. Le nombre de sommets $\mid V_P \mid$ est bien égal au nombre d'éléments mémoire du circuit (colonne 2). Enfin, la dernière colonne donne les temps *CPU* pour l'extraction du graphe G_P à partir du graphe G_D (CPU_G_D_to_G_P).

Le tableau 4.3 permet de comparer les résultats des longueurs de chaînes obtenues avec les deux heuristiques utilisées : la 2-approximation et Christofidès. Les longueurs sont calculées avec le poids des arêtes du graphe G_P. La colonne 1 donne le nom du circuit. Les colonnes 2 et 3 donnent la taille du graphe de proximité G_P avec respectivement le nombre de sommets et le nombre d'arêtes. Les colonnes 4 et 5 donnent la longueur de la 2-approximation ainsi que le temps d'exécution. Les colonnes 6 et 7 représentent les longueurs pour Christofidès ainsi que les temps d'exécution.

Les résultats des deux heuristiques sont équivalents en termes de qualité de la solution et temps d'exécution. En effet, par exemple pour le circuit *b18* l'heuristique Christofidès donne un meilleur résultat que la 2-approximation avec une longueur de chaîne égale à 5763

Design	$\mid V_P \mid$	$\mid E_P \mid$	2-approx	CPU	Chritf	CPU
b01	5	118	9	$<$ 1 ms	12	$<$ 1 ms
b03	30	450	73	$<$ 1 ms	66	10 ms
b04	66	1111	186	$<$ 1 ms	186	10 ms
b05	34	3795	91	$<$ 1 ms	91	$<$ 1 ms
b06	9	158	21	1 ms	124	10 ms
b07	49	1125	123	1 ms	149	1 ms
b09	28	401	68	1 ms	158	1 ms
b10	17	525	39	17	38	$<$ 1 ms
b11	31	1094	70	10 ms	71	10 ms
b12	121	3879	303	20 ms	294	20 ms
b13	53	633	124	$<$ 1 ms	134	$<$ 1 ms
b14	245	27269	660	110 ms	676	80 ms
b15	449	33179	1177	320 ms	1115	270 ms
b17	1415	100358	4003	3 s	3436	3 s
b18	3320	271320	9561	10 s	5763	3 min
b19	6642	531161	9555	20 s	10638	5 min
simple-spi	132	8778	432	20 ms	411	100 ms
biquad	204	357	576	80 ms	508	350 ms
Ac97	2289	18727	7956	8 s	7822	30 s

TAB. 4.3: Longueurs des chaînes et temps d'exécution des heuristiques.

alors que la 2-approximation trouve 9561 mais par contre elle ne prend que 10 secondes contre 3 minutes pour Cristofidès. Cela fait une réduction de longueur de 40% mais avec un temps de calcul beaucoup plus élevé.

De plus, les tests montrent que la qualité des solutions est sensiblement identique pour les différentes valeurs du paramètre t de l'algorithme BFS limité. En effet, nous avons expérimenté différentes profondeurs t de l'algorithme BFS limité, (de 1 à 6) pour le passage de $G_D = (V_D, E_D)$ à $G_P = (V_P, E_P)$. Pour des raisons de temps calcul et d'espace mémoire, nous avons fixé ce paramètre à $t = 4$. En effet, à partir de $t = 5$ les plus gros circuits prennent plus de $4h$ à l'exécution.

4.6.3 Protocole expérimental

Dans cette section, le potentiel de l'approche présenté précédemment va être évalué dans une série d'expériences.

Afin de comparer nos résultats d'insertion des chaînes de scan au niveau *RTL* avec la méthode d'insertion du scan niveau *gate* nous avons déroulé l'ensemble du flot de conception, partant de la description *RTL* jusqu'au placement et routage.

L'insertion des chaînes de scan au niveau *RTL* est faite avec l'outil *HiDFT_Scan* de *DeFacTo Technologies* et au niveau gate avec l'outil *Talus* de *Magma*.

Nous avons ensuite mesuré la longueur totale des interconnexions (*wirelength : WL*) obtenue après placement et routage avec les deux méthodes. Nous avons utilisé le flot complet de *Magma* pour les étapes de synthèse et de placement routage pour des raisons d'homogénéité et de disponibilité de licence.

Enfin, pour vérifier que les chaînes de scan insérées apportent un bon *taux de couverture de faute*, nous avons exécuté un *ATPG (Automatic Test Pattern Generation)*. Pour cela nous avons utilisé l'outil *Tetramax* de la société *Synopsis* car c'est un des outils les plus utilisés dans le marché de l'EDA.

Les résultats de *Wirelength* avec la première stratégie (partition puis recherche d'une chaîne par partie) sont donnés dans le tableau 4.4.

Design	NB-Chaînes-Scan	HiDFT_WL	Magma_WL	Ratio(%)
b01	1	0,392	0,388	0,974
b03	2	1,811	1,604	11,41
b04	3	6,313	6,303	0,159
b05	2	5,334	5,193	2,647
b06	1	0,525	0,478	8,936
b07	5	4,745	4,234	10,78
b09	2	1,475	2,055	-39,32
b10	2	1,932	1,942	0,503
b11	3	5,399	4,773	11,59
b12	10	12,97	12,59	2,930
b13	5	3,318	3,394	-2,297
b14	24	70,51	63,63	9,751
b15	24	132,1	126,3	4,386
b17	70	394,2	395,6	0,355
b18	300	988,9	1187,1	-18,84
b19	600	1858,1	2329,6	-25,37
simple-spi	10	10,43	11,45	-9,83
biquad	20	36,07	34,10	5,45
Ac97	200	291,7	247,7	15,09

TAB. 4.4: Table WL avec la stratégie de partition.

La colonne 1 donne le nom du circuit, la colonne 2 correspond au nombre de chaînes de scan qui est égal au nombre de partitions. Les colonnes 4 et 5 donnent le *wirelength* obtenu après placement et routage des deux méthodes scan haut niveau (Hidfdt_WL) et scan bas niveau (Magma_WL). La dernière colonne donne le pourcentage de réduction (resp. augmentation dans certains cas) du *wirelength* avec **HiDFT**.

En moyenne, la réduction du *wirelength* est de 6% entre le scan haut niveau et le scan bas niveau. En particulier, pour les gros circuit ($b17$, $b18$, $b19$) *Hidft_Scan* donne de meilleur résultats que *Magma*.

Les résultats de wirelength avec la deuxième stratégie (recherche d'une seule chaîne puis découpage) sont donnés dans le tableau 4.5.

Design	NB-Chaînes-Scan	HiDFT_WL	Magma_WL	Ratio(%)
b01	1	0,393	0,388	1,325
b03	2	1,808	1,604	11,31
b04	3	6,555	6,303	3,847
b05	2	5,427	5,193	4,309
b06	1	0,537	0,478	10,87
b07	5	4,524	4,234	6,426
b09	2	1,632	2,055	-25,92
b10	2	2,058	1,942	5,636
b11	3	5,030	4,773	5,105
b12	10	12,92	12,59	2,522
b13	5	3,318	3,394	-2,297
b14	24	64,24	63,63	0,946
b15	24	132,1	126,3	4,386
b17	70	394,2	395,6	-0,355
b18	300	998,9	1187,1	-18,84
b19	600	1858,1	2329,6	-25,37
simple-spi	10	10,43	11,45	-9,837
biquad	20	36,07	34,10	5,453
Ac97	200	291,7	247,7	15,09

TAB. 4.5: Table WL avec la stratégie de chaîne découpée.

La colonne 1 donne le nom du circuit, la colonne 2 correspond au nombre de chaînes de scan qui est égal au nombre de partitions. Les colonnes 4 et 5 donnent le *wirelength* obtenu après placement et routage des deux méthodes scan haut niveau (Hidfdt_WL) et scan bas niveau (Magma_WL). La dernière colonne donne le pourcentage de réduction (resp. augmentation dans certains cas) du *wirelength* avec *HiDFT*.

Les résultats sont équivalents à la première stratégie. Pour les circuits de petite taille, la deuxième stratégie donne de meilleurs résultats. Par contre, pour les circuits de grande taille (en terme de nombre d'éléments mémoires) la première stratégie de partition est meilleure.

En effet, pour les circuits de grosse taille, il est difficile de calculer une seule chaîne. Par exemple, pour les besoins spécifiques de *DeFacTo Technologies*, nous avons testé les deux stratégies sur un gros circuit de plus de 60000 éléments mémoires. Comme il s'agit de circuits confidentiels les résultats numériques ne peuvent être donnés dans cette thèse. La stratégie de chaînage pour ce circuit est la deuxième vu sa taille. La réduction de *wirelength* par rapport au flot magma est de 8% pour ce circuit.

Le tableau 4.6, présente les résultats de couverture de fautes (*Fault Coverage*) avec le noms des circuits et la couverture de faute obtenue.

Les chiffres de couverture de faute (*fault coverage*) obtenus montrent que le taux de couverture de fautes du scan *RTL* est largement acceptable (très proche de 100%).

Design	b01	b03	b04	b05	b07	b08	
Fault Coverage	100.00%	99.95%	99.96%	99.51%	99.96%	99.51%	
Design	b09	b10	b11	b12	b13	b14	
Fault coverage	99.86%	99.85%	99.93%	99.97%	99.92%	99.99%	
Design	b15	b17	b18	b19	simple-spi	Biquad	Ac-97
Fault Coverage	99.97%	99,4.7%	99,81%	99,81%	98,35%	99,96%	99,80%

TAB. 4.6: Résultats de couverture de fautes.

4.7 Conclusion

Un algorithme d'insertion de c*haînes de scan* au niveau *RTL* a été proposé et implémenté pour répondre au problème industriel de *DeFacTo Technologies*.

Les tests effectués montrent que les objectifs sont atteints : l'insertion des *chaînes de scan* au niveau *RTL* est faite en une passe au lieu de plusieurs itérations au niveau *gate* ; la longueur totale des interconnexion après l'insertion des chaînes de scan est réduite de 6% en moyenne par rapport au scan bas niveau, le taux de couverture de fautes est équivalent.

Les résultats présentés sont indicatifs et montrent que la méthode du scan *RTL* peut être étendue à de plus gros circuits industriels.

Notre algorithme de stitching a été validé par notre partenaire industriel *DeFacTo Technologies*. Pour des raisons de confidentialité, il n'est pas possible de donner ici les résultats obtenus pour de plus grands circuits. Néanmoins on peut noter des résultats assez satisfaisants.

Des problèmes théoriques en théorie des graphe et couverture des sommets ont également été dérivés de cette collaboration.

Les travaux et résultats de cette partie ont été présentés à plusieurs conférences dans le domaine de la *RO* à *EURO 2009* [64] ainsi que dans la communauté du Test à *DATE 2009* [65] et *WRTLT 2009* [66].

Troisième partie

Le BIST mémoire

Le second problème traité durant cette thèse est l'optimisation du partage des blocs *BIST* (*Built In Self Test*) pour le test intégré des mémoires. Dans les *systèmes sur puce* (*SoC*) actuels, les éléments de mémorisation représentent la grande majorité des dispositifs embarqués. En effet, les mémoires occupent très couramment plus de la moitié de la surface des *SoC* et la feuille de route (*roadmap*) établie par l'*ITRS*, édition 2003, indique que ce ratio devrait dépasser 90% d'ici dix ans.

Dans le processus de conception des systèmes sur puces, l'optimisation des ressources de mémorisation en termes de densité, de puissance consommée et de temps d'accès est donc un élément prépondérant. Cette situation conduit à concevoir des mémoires en exploitant les limites de la technologie. En conséquence, ces dispositifs deviennent de plus en plus sensibles aux pannes. Ainsi, les mémoires deviennent donc très complexes à tester.

La solution utilisée actuellement pour tester les mémoires est l'intégration d'une partie des tâches réalisées par le testeur dans la mémoire elle-même à travers l'insertion des blocs *BIST*. Ils font appel à des algorithmes bien connus tels que les algorithmes de *March* [57] pour les procédures de test.

Le problème abordé dans cette dernière partie concerne l'optimisation du partage de blocs *BIST* pour le test intégré des mémoires présentes dans un circuit quelque soit leur type et leur taille. Les travaux de cette partie ont été réalisés en collaboration avec la société STMicroelectronics.

Nous allons tout d'abord présenter la problématique de partage de blocs *BIST* pour le test intégré des mémoires ainsi que la modélisation proposée dans le chapitre 5. Enfin, pour clore cette partie nous présentons dans le chapitre 6 le prototype *Memory Bist Optimizer* (*MBO*) développé afin de résoudre ce problème suivi des premiers résultats expérimentaux obtenus sur des instances réelles fournies par notre partenaire.

Chapitre 5

Problème du partage des *BIST* mémoire

Sommaire

5.1 Introduction 83
5.2 Insertion du BIST mémoire 84
5.2.1 Pourquoi et comment partager les blocs BIST 84
5.2.2 La solution actuelle 86
5.2.3 Objectif de la thèse pour le partage de blocs *BIST* 86
5.3 Etat de l'art 87
5.4 Modélisation du Problème du partage des blocs *BIST* 87
5.4.1 Données 88
5.4.2 Contraintes 88
5.4.2.1 Régles de partage d'un collier 89
5.4.2.2 Mémoires partageant le même collier 89
5.4.2.3 Les paramètres utilisateurs 90
5.4.3 Objectifs 90
5.4.3.1 La surface 90
5.4.3.2 La puissance 90
5.4.3.3 Le temps de test 91
5.4.4 Une solution au problème 91
5.5 Conclusion 91

5.1 Introduction

Les systèmes sur puce actuels contiennent un grand nombre de mémoires. Elles représentent la grande majorité des dispositifs embarqués. En effet, les mémoires occupent très couramment plus de la moitié de la surface des *SoC* actuels.

Du fait de la structure particulière des mémoires, une des solutions les plus utilisées actuellement pour tester les mémoires est la méthode *BIST* [46].

Il s'agit d'une méthode de test dans laquelle une partie du circuit est utilisée pour tester le circuit lui-même. Pour cela, il faut rajouter autour des mémoires un ensemble de composants nécessaires au test, c'est ce que nous appelons un *BIST*. Il est constitué de *Contrôleurs* et de *Colliers*.

Le *contrôleur* sert à contrôler les instructions pour l'ensemble des mémoires du circuit à tester. Chaque mémoire du circuit est pilotée par un *collier* qui a pour rôle d'exécuter les instructions données par le *Contrôleur*.

La figure 5.1 représente une architecture classique d'un système *BIST* avec 9 mémoires, 9 colliers et un contrôleur.

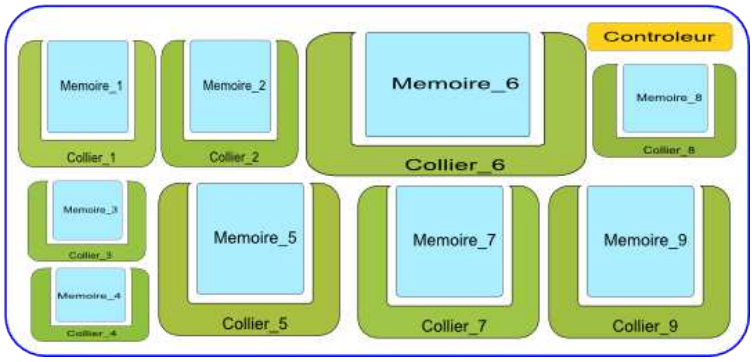

FIG. 5.1: Architecture *BIST* dédiée.

5.2 Insertion du BIST mémoire

5.2.1 Pourquoi et comment partager les blocs BIST

L'insertion de blocs *BIST* pour le test des mémoires est facile lorsque le circuit contient un nombre réduit de mémoires. On peut alors opter pour une architecture dédiée, c'est-à-dire chaque mémoire est pilotée par un *collier* qui lui est propre.

Néanmoins, de nos jours, la taille des circuits entraîne, pour cette stratégie, une augmentation trop importante de la *surface* requise, d'où l'idée de partager des colliers au sein d'un groupe de mémoires compatibles.

Une solution au problème de la surface additionnelle due à l'insertion du *BIST* consiste donc à définir un contrôleur pour l'ensemble du système et de partager chaque collier, si possible, entre plusieurs mémoires.

La figure 5.2 illustre la même architecture que celle de l'exemple de la figure 5.1 avec 9 mémoires mais avec des colliers partagés. Ainsi le nombre total de colliers passe de 9 en architecture dédiée à 5 partagés, ce qui réduit presque de moitié le nombre total de colliers.

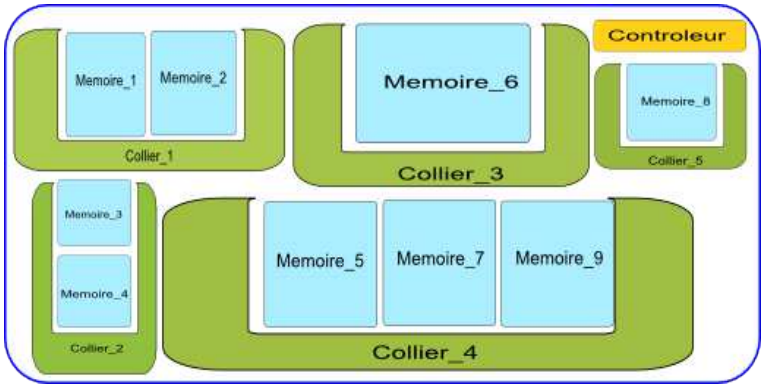

FIG. 5.2: Architecture *BIST* partagée.

Pour les architectures des circuits considérées, on distingue deux types de partage de colliers : *colliers parallèles*, pour un test simultané de mémoires et *colliers séquentiels*, pour le test en séquence de mémoires.

Le partage *parallèle* de colliers permet de réduire la surface ainsi que le temps de test. Ce type de partage est néanmoins très coûteux en terme de consommation de puissance. En effet, la procédure de test est lancée en parallèle pour toutes les mémoires du circuit, ce qui crée un pic de consommation du circuit en mode test.

Le partage *séquentiel* quant à lui réduit la surface et la puissance nécessaires mais est très coûteux en temps de test. En effet, la procédure de test est effectuée en série mémoire par mémoire pour l'ensemble du circuit.

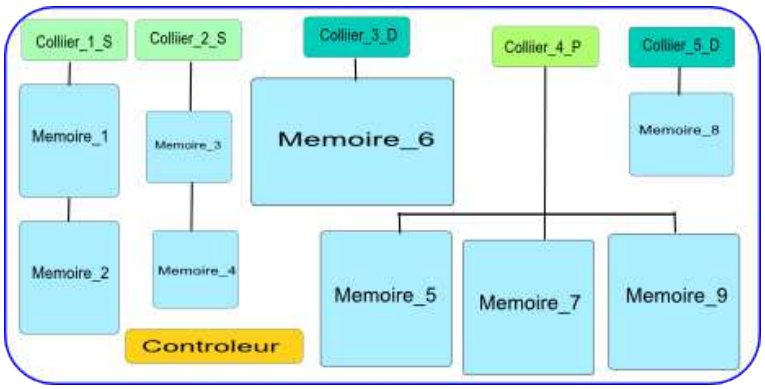

FIG. 5.3: Architecture *BIST* partagée avec colliers de différents types.

La figure 5.3 montre les différents types de partages possibles pour l'exemple précédent. C'est le même circuit que celui de la figure 5.2 avec une représentation différente des colliers

afin de voir leur type. Le circuit comporte deux colliers dédiés (colliers 3 et 5), deux colliers séquentiels (colliers 1 et 2) et un collier parallèle (collier 4).

5.2.2 La solution actuelle

Typiquement, la définition d'une architecture *BIST* pour le test intégré des mémoires d'un circuit, revient à décider d'une configuration du partage ou non des colliers *BIST* autour des mémoires en considérant les besoins et contraintes utilisateurs.

Actuellement, l'intégration du *BIST* se fait par les clients de STMicroelectronics à travers une interface appelée *webgen* développée par des ingénieurs *DFT* de l'entreprise. Elle contient des tables de définition des mémoires avec des règles de partage. A partir de ces règles, le concepteur définit à la main l'ensemble des mémoires qui seront partagées sous un même collier. Il fait ensuite une analyse du rapport performance/coût relatif à la configuration choisie. Si ce dernier est mauvais, il réalise plusieurs itérations à partir de la première étape afin de l'améliorer. Ce processus itératif est coûteux et peut ainsi impacter l'ensemble du processus de conception.

5.2.3 Objectif de la thèse pour le partage de blocs *BIST*

Le problème d'optimisation du partage des blocs *BIST* pour le test des mémoires peut être formulé de la façon suivante.

Étant donné un groupe de mémoires définies par leur type, un ensemble de paramètres et des règles de partage des colliers en séquentiel et en parallèle, on cherche à trouver une solution consistant à associer à chaque mémoire un collier. La solution obtenue doit minimiser à la fois la surface, la puissance dynamique et le temps de test.

Pour cela, nous cherchons tout d'abord à concevoir un système automatique qui permet de déterminer un ensemble de solutions valides. Ensuite, nous nous intéressons à déterminer une solution optimale.

La possibilité de partager des blocs *BIST* pour le test des mémoires peut être prise en compte de deux façons : « à la main » par l'utilisateur, ou de manière automatique par un outil. La première option n'est guère satisfaisante vue la complexité des circuits actuels. Grâce à la deuxième option, le temps de l'utilisateur est mieux exploité à orienter l'outil à travers un jeu de paramètres permettant de spécifier finement quelles contraintes respecter, quel objectif viser et quel degré d'automatisation ciblé.

Ceci nécessite une étude poussée de manière à dégager les paramètres et notions pertinentes. Ces derniers nous permettront de construire le modèle mathématique qui servira ensuite pour le développement d'algorithmes d'optimisation.

La modélisation ainsi que la solution obtenue doivent être indépendantes de la technologie. En effet, les circuits sont conçus par *génération de nœud* technologique (65nm, 45nm, 32nm, etc) qui correspondent à la finesse de gravure du circuit. Les générations technologiques changent en moyenne tous les 18 mois, il est alors essentiel de concevoir des modèles indépendants de la technologie.

5.3 Etat de l'art

Plusieurs études ont été menées dans la littérature concernant les techniques d'optimisation de partage de blocs *BIST*. Elles ont pour but de déterminer comment partager les composants nécessaires à l'insertion des contrôleurs et colliers pour plusieurs mémoires.

Dans [71], l'auteur présente la structure de partage la plus utilisée actuellement pour l'insertion des blocs *BIST* partagés entre plusieurs mémoires. Il propose aussi un ordonnancement pour l'exécution du processus *BIST*. Il introduit une nouvelle méthodologie pour l'implémenter avec une architecture très modulaire. L'objectif est l'optimisation de la dissipation de puissance. Cette architecture de *BIST* peut fournir aussi des blocs d'informations sur le diagnostic. L'architecture proposée n'est valide que pour des mémoires de même type.

Dans [9], [45], [10] les auteurs proposent différentes architectures pour partager les colliers entre des mémoires de même type (des *SRAM*) et de tailles différentes. L'objectif est de réduire la surface additionnelle due à l'insertion des colliers pour le test des mémoires. Il proposent aussi l'exécution d'algorithmes de *march* optimisés afin de réduire les temps d'exécution de la procédure de test. Cette stratégie laisse le concepteur choisir l'ensemble des mémoires qu'il veut partager dans un même collier. Ce choix se fait à la main.

A notre connaissance le seul travail qui considère le problème sous le même angle que le notre, c'est-à-dire trouver un partage optimal pour les blocs *BIST* entre plusieurs ensembles de mémoires (de différents types et tailles) est celui présenté dans [49]. Dans ce travail, les auteurs cherchent à minimiser la surface additionnelle due à l'insertion des blocs *BIST*. Pour cela, ils proposent le partage des colliers entre des ensembles de mémoires compatibles.

Ils construisent un graphe de compatibilité dont les sommets sont les mémoires et les arêtes représentent les possibilités de partager un collier entre deux mémoires. Ensuite, pour trouver les ensembles de mémoires compatibles, ils cherchent des sous-ensembles de *cliques* [1] de tailles maximales. L'avantage de cette méthode est de considérer l'ensemble des mémoires, l'inconvénient en est le temps d'exécution. Ce dernier limite l'utilisation de cette méthode à des circuits contenant un nombre restreint de mémoires. En effet, les résultats numériques présentés dans cet article se limitent à 50 mémoires avec un temps d'exécution qui augmente exponentiellement par rapport au nombre de mémoires.

5.4 Modélisation du Problème du partage des blocs *BIST*

Le problème de l'optimisation du partage des blocs *BIST* pour le test des mémoires consiste à identifier les meilleures configurations (dédiées, séquentielles et/ou parallèles) en termes de surface, puissance et temps de test. L'objectif est donc de concevoir un système automatique « intelligent » afin de :

[1] clique : une clique est un ensemble de sommets deux-à-deux adjacents. La recherche de la plus grande *clique* d'un graphe (au sens du nombre de sommets) est un problème $\mathcal{NP}$-Complet. Le terme *clique* est aussi souvent utilisé pour parler du graphe induit par une clique

- guider le concepteur du circuit, qui n'est pas forcément un expert en système *BIST*, à définir le système de test adéqua ;
- rechercher automatiquement un ensemble de solutions valides ;
- tenir compte de la mutation vers un nouveau de nœud technologique, c'est-à-dire rester le plus possible indépendant de la technologie utilisée.

Pour cela, et après plusieurs échanges avec notre partenaire industriel STMicroelectronics, nous avons identifié les données d'entrées, les contraintes à respecter et les objectifs du problème. Nous les décrivons dans les sections suivantes.

5.4.1 Données

Nous résumons les données d'entrées par la liste suivante :

- *Les cuts* : qui correspondent aux ensembles de type de mémoires du circuit. Chaque *cut* est défini par les caractéristiques suivantes : Id_Cut, Cut_name, Word, bits, mux, redundancy, Generator_Name, *self_time_control* et *bit_write*.
- Une mémoire qui représente une instance d'un *cut*. Donc en plus des instances définissant son cut approprié, une mémoire possède un paramètre de fréquence et/ou un paramètre *Id_User_Defined_Group* qui désigne un groupe de compatibilité défini par l'utilisateur.
- La liste des mémoires d'un domaine de test d'un circuit.
- Pour chaque mémoire du circuit, un certain nombre de caractéristiques importantes lui sont associées comme :
  son type (SP, DP, ROM, RF, etc), le compilateur (SPHS, SPREG, SPHD, DPREG, ROM, DPHD, etc), *word* (le nombre de mots), *bits* (la largeur des mots), *mux* (la cardinalité du multiplexeur), le masque, la *bank*, la redondance.
- La fréquence de test.
- Une estimation de la puissance de test.
- Une estimation du temps de test en fonction des périodes de l'horloge commune et de l'horloge de la mémoire testée.

Remarque : Nous citons ici uniquement les caractéristiques nécessaires pour la création des groupes de compatibilités des mémoires. Il existe d'autres paramètres qui n'interviennent pas dans ce travail, comme :

- *debug pin* : la possibilité de déboguer.
- *power down* : la veille.

5.4.2 Contraintes

Les contraintes liées au problème peuvent être partagées en trois catégories :

- les règles de partage d'un collier (séquentiel ou parallèle) ;
- les caractéristiques des mémoires ;
- des paramètres utilisateurs.

Les détails de chaque catégorie de contraintes sont données dans les sections suivantes. Notons que les contraintes de partage dépendent du nœud technologique utilisé et peuvent

changer d'une génération à une autre.

5.4.2.1 Régles de partage d'un collier

Les règles générales pour le partage d'un collier sont les suivantes :
- Sous un même collier, seulement des mémoires de même type et de même paramètre *generator_ Name* peuvent être partagées.
- Même domaine d'horloge [2] pour les mémoires de type *Single port* (*SP*).
- Les mémoires de type *Multi Ports* ont des ports d'entrées dédiés aux horloges. Il est préférable de grouper les mémoires de tel sorte que pour toutes les mémoires d'un même groupe la fréquence d'horloge la plus rapide soit identique.
- Les mémoires doivent être proches : basées sur une proximité logique qui est un paramètre défini par STMicroelectronics. Il s'agit d'une mesure de proximité basée sur les intéractions entre les différents types de mémoires, établie à partir de données physiques. On impose que seule les mémoires proches selon cette mesure peuvent faire partie d'un même collier.
- Un groupe peut être imposé par l'utilisateur avec le paramètre *Id_ User_ Defined_ Group*.

De plus certaines règles de partage sont spécifiques à un collier séquentiel ou à un collier parallèle. Elle sont données dans le paragraphe suivant.

Régles de partage d'un collier séquentiel Pour pouvoir partager un collier séquentiel entre plusieurs mémoires, toutes les règles suivantes doivent être respectées :
- Les paramètres *redundancy*, *generator_ Name*, *bit_ Write* et *self_ time_ control* doivent être (strictement) identiques.
- Les paramètres *Words*, *bits* et *mux* peuvent être différents.
- Dans le cas d'une *SPREG* avec $mux = 2$, le collier ne peut partager que d'autres *SPREG* avec $mux = 2$.
- Dans le cas d'une *SPHS* avec *redundancy*, le collier ne peut partager que des *SPHS* avec le même paramètre *bank*.

Régles de partage d'un collier parallèle Pour pouvoir partager un collier parallèle entre plusieurs mémoires, toutes les règles suivantes doivent être respectées :
- Les paramètres *Words*, *bits*, *mux*, *self_ time_ control*, *redundancy*, *generator_ Name* et *bit_ Write* doivent être (strictement) identiques.
- Les mémoires doivent être de type *DP*.

5.4.2.2 Mémoires partageant le même collier

Le nombre de mémoires pouvant partager un même collier (parallèle ou séquentiel) dépend du nombre de bits des mémoires considérées. De plus on a les contraintes suivantes :

[2] domaine d'horloge : plusieurs mémoires qui sont pilotées par une même horloge sont dites regroupées dans un domaine d'horloge.

- Pour une Single Port ou une *DPREG*, la somme des bits de données de chaque mémoire partagée doit être inférieur à un paramètre t qui dépend du nœud technologique.
- Pour une Dual Port *(DPHD)*, la somme des bits de données de chaque mémoire partagée doit être inférieur à un paramètre t' qui dépend du nœud technologique.

5.4.2.3 Les paramètres utilisateurs

Les paramètres utilisateurs, permettant de spécifier :
- Le ou les paramètres à optimiser prioritairement si besoin pour certaines applications particulières du circuit.
- Des indications de groupage limitant les possibilités de partage de collier afin de limiter plus tard les problèmes de routage dans le flot de conception.

5.4.3 Objectifs

Afin de déterminer un partage optimal des blocs *BIST* entre plusieurs mémoires les trois objectifs à minimiser sont la surface additionnelle, la puissance et le temps de test.

Nous détaillons chaque critère dans la suite.

5.4.3.1 La surface

La surface additionnelle découle de deux facteurs : la surface propre des colliers et la congestion due au routage. Seule la surface des colliers est totalement prise en compte dans ce travail. En effet, la congestion est contrôlée par les deux mécanismes suivants :
- les règles de partage, qui interdisent le partage d'un collier lorsque ce partage induit un trop grand nombre d'interconnexions, comme la limite du nombre de mémoires à partager par collier ;
- la possibilité pour l'utilisateur d'interdire certains partages, via la spécification de groupes de partages plus restreints que ceux considérés ici, qui découlent des caractéristiques des mémoires.

Remarque : Ce deuxième mécanisme permet à l'utilisateur de réitérer, si nécessaire, l'optimisation du BIST après une étape de placement-routage insatisfaisante par exemple.

5.4.3.2 La puissance

Par objectif de puissance, on entend puissance maximale instantanée dans le circuit pendant le test. Dans ce contexte, la puissance maximum est la somme des puissances dissipées par les différents colliers actifs simultanément, sachant que la puissance dissipée :
- par un collier dédié est la puissance de test de la mémoire associée,
- par un collier séquentiel est le maximum des puissances de test des mémoires associées,
- par un collier parallèle, la somme des puissances de test des mémoires associées.

5.4.3.3 Le temps de test

Le temps de test d'un collier dépend de son type.
- Pour un collier dédié, la durée nécessaire au test de la mémoire associée,
- Pour un collier séquentiel, la somme des durées nécessaires au temps de test des mémoires associées,
- Pour un collier parallèle, le maximum des durées nécessaires au temps de test des mémoires associées.

Analyse des critères d'optimisation
Une solution au problème du *BIST* consiste à définir un contrôleur pour l'ensemble du système et des colliers, pour une mémoire ou pour un ensemble de mémoires, minimisant simultanément les trois objectifs : surface, puissance de test et temps de test. Par conséquent, on est confronté à un problème d'optimisation multi-objectif. Dans ce contexte, il est quasiment impossible de parler de solution optimale. On cherche alors à identifier un ensemble de solutions de compromis entre les trois objectifs.

5.4.4 Une solution au problème

Le résultat escompté dans le cas multi-objectif est un ensemble de solutions non dominées (voir définition chapitre 6). L'utilisateur choisit ensuite celle qui le satisfait par rapport à son utilisation. Nous verrons en détail dans le chapitre 6, la méthode de résolution que nous proposons pour cela.

Pour chaque solution les informations suivantes doivent être données : types de colliers à réaliser (dédié, séquentiel et parallèle) ainsi que les valeurs des trois critères d'optimisation (surface, puissance et temps de test).

5.5 Conclusion

Dans ce chapitre, nous avons présenté le problème de partage des blocs *BIST* pour le test des mémoires. Cette problématique nous a été posée par notre partenaire industriel STMicroelectronics. Comme nous l'avons souligné tout au long du chapitre, le partage des blocs *BIST* pour le test intégré des mémoires joue un rôle important dans le domaine du test et de la conception en vue du test des mémoires. De ce fait, cette problématique est très présente aussi bien dans le monde de la recherche académique que dans l'industrie.

Ensuite, nous avons proposé une modélisation du problème décomposée en données d'entrées, contraintes à respecter et objectifs à minimiser. Cette phase est fondamentale dans le processus d'optimisation, dans la mesure où elle conditionne les étapes suivantes. Les travaux de modélisation de cette partie ont été présentés à *EURO 2009* [38] ainsi qu'a *ROADEF 2010* [61] et *ISCO 2010* [39].

Le chapitre suivant est consacré à notre contribution pour la méthode de résolution ainsi que les résultats numériques qui s'y rapportent.

Chapitre 6

Résolution du problème de partage des BIST mémoire

Sommaire

6.1 Introduction 93
6.2 Evaluation des critères d'optimisation 94
6.2.1 Calcul des critères d'optimisation 94
6.2.1.1 La surface 94
6.2.1.2 La puissance 95
6.2.1.3 Temps de test 95
6.2.2 Calcul des bornes 96
6.2.3 Optimisation multi-objectif et solution de compromis 97
6.2.3.1 Quelques définitions 97
6.2.3.2 Les algorithmes génétiques 99
6.3 Méthode de résolution : Memory BIST Optimizer 100
6.3.1 Module 1 : Modélisation et création des groupes de compatibilité 101
6.3.2 Module 2 : Optimisation des mémoires 103
6.3.3 Module 3 : Validation des colliers pour les mémoires 105
6.4 Implémentation & résultats numériques 106
6.4.1 Protocole expérimental et description des instances 106
6.4.2 Résultats numériques & Discussion 107
6.5 Conclusion 111

6.1 Introduction

Dans le cadre du projet *ASTER* qui a permis la collaboration avec STMicroelectronics, nous nous sommes intéressés au problème du partage des blocs *BIST* pour le test des mémoires, comme décrit dans le chapitre 5.

Le partenaire industriel a définit les différents types de partages possibles : dédié, séquentiel et parallèle avec les architectures correspondant à chaque type.

Nous avons travaillé ensemble afin d'évaluer les coûts relatifs à chaque architecture en termes de surface additionnelle, consommation en puissance et temps de test. Le but de cette démarche est la détermination d'une solution de partage optimale étant donnés un ensemble de mémoires de différents types et tailles ainsi que la réalisation d'un prototype pour automatiser la génération de ces solutions.

Ce prototype est issu d'une collaboration étroite entre les ingénieurs *DFT* à STMicroelectronicset plusieurs personnes de notre équipe au sein du laboratoire, notamment un stagiaire et un post-doctorant qui ont travaillé sur ce projet.

Nous exposons dans ce chapitre la méthode de résolution. Elle est implémentée dans le prototype *Memory BIST Optimizer* que nous présentons dans la deuxième partie du chapitre. Dans la dernière partie, nous donnons les résultats numériques obtenus sur des circuits industriels fournis par notre partenaire industriel.

6.2 Evaluation des critères d'optimisation

6.2.1 Calcul des critères d'optimisation

Comme nous l'avons signalé dans la section 5.4.3, nous avons identifié trois critères d'optimisation. Nous donnons ici les formules qui permettent d'estimer ces trois critères en fonction des différents types de collier *BIST* : dédiés, séquentiels ou parallèles.

Ces formules ont été établies pour les besoins spécifiques de STMicroelectronics. Elles dépendent des architectures choisies pour les différents types de collier. Si d'autres architectures s'avèrent meilleures, il faudra donc mettre à jour les formules d'estimation. Ces critères sont à minimiser.

6.2.1.1 La surface

La surface d'un collier dépend des paramètres suivants :
- type de partage (dédié, séquentiel ou parallèle),
- type de mémoire,
- nombre de mémoires attachées à ce collier,
- nombre de bits de la (ou des) mémoires pilotées par le collier.

Pour calculer la surface on utilise les formules présentées ci-dessous.

Cas des colliers dédiés :
Surface $= A \times N_{Bit} + C$;

où N_{Bit} est le nombre total de bits.
A et C sont des constantes qui dépendent du type de la mémoire (Single Port, Dual Port, ROM...) ainsi que du nœud technologique.

Pour des raisons de confidentialité, les paramètres ne sont pas donnés ici.

Cas des colliers partagés :
Surface = $\sum[A \times N_{Bit} + \alpha \times C]$
Où :
- N_{Bit}, A et C ont les mêmes valeurs que dans le cas dédié;
- $\sum$: somme sur tous les ensembles de mémoires partagées par le collier;
- α : un facteur de réduction dépendant du nombre total de cuts partagés et du type de partage (séquentiel ou parallèle). Ce facteur est commun à tous les types de mémoire.

La surface totale d'une architecture *BIST* est la somme des surfaces des différents colliers.

Pour des raisons de confidentialité, les valeurs des paramètres A, C et α ne sont pas données ici.

6.2.1.2 La puissance

Pour le calcul de la puissance, l'unité est la puissance nécessaire pour le test d'une mémoire. On obtient donc les puissances suivantes :
- **Collier dédié** : la puissance consommée par la mémoire associée à ce collier;
- **Collier séquentiel** : le maximum des puissances consommées par les N mémoires associées à ce collier;
- **Collier parallèl** : la somme des puissances consommées par les N mémoires associées à ce collier.

La puissance totale, pendant le test d'une architecture *BIST* donnée est la somme des puissances dissipées par les différents colliers.

6.2.1.3 Temps de test

Le temps de test d'un collier est calculé comme suit :
- **Collier dédié** : temps de test de la mémoire associée;
- **Collier séquentiel** : somme des temps de test des mémoires;
- **Collier parallèle**: maximum des temps de test des mémoires associées;

Tous les tests d'une architecture *BIST* sont lancés simultanément (par le contrôleur), ainsi le temps de test global du système est le maximum sur l'ensemble des colliers du temps de test associé à un collier.

Evaluation du temps de test des mémoires : Le temps de test de mémoires dépend du nombre de *words* M dans la mémoire. Le calcul du temps de test peut être décomposé en deux parties :

1. le temps de test lié à l'horloge lente du système, qui est commune à tous les colliers d'un système *BIST*,
2. le temps de test lié à l'horloge fonctionnelle de la mémoire, qui est spécifique à chaque collier (la valeur est fournie en paramètre d'entrée).

6.2.2 Calcul des bornes

La résolution du problème s'avère facile lorsque le système *BIST* contient un nombre réduit de mémoires pour lequel on peut opter pour une architecture dédiée, c'est-à-dire chaque mémoire est pilotée par un collier. Cette solution nous donne la surface maximum.

Si on opte pour une solution complètement parallèle alors on arrive à réduire la surface, minimiser le temps d'exécution mais la puissance est au maximum.

A l'inverse, si on décide de ne partager qu'avec des colliers séquentiels alors la puissance est minimum et le temps de test est maximum.

Les formules suivantes donnent une évaluation des bornes inférieures et supérieures des 3 critères :
Surface Max = tous les colliers sont dédiés.
Puissance Max = tous les colliers du circuit sont partagés en parallèle.
Puissance Min = tous les colliers du circuit sont partagés en séquentiel.
Temps test Max = tous les colliers du circuit sont partagés en séquentiel.
Temps test Min = tous les colliers du circuit sont partagés en parallèle.

La figure 6.1 suivante illustre la puissance consommée en fonction du temps de test pour les différents types de colliers dédié, séquentiel et parallèle. Les différents rectangles (rouges, verts, bleus, etc) représentent en longueur le temps de test des différents colliers et en hauteur leur puissance.

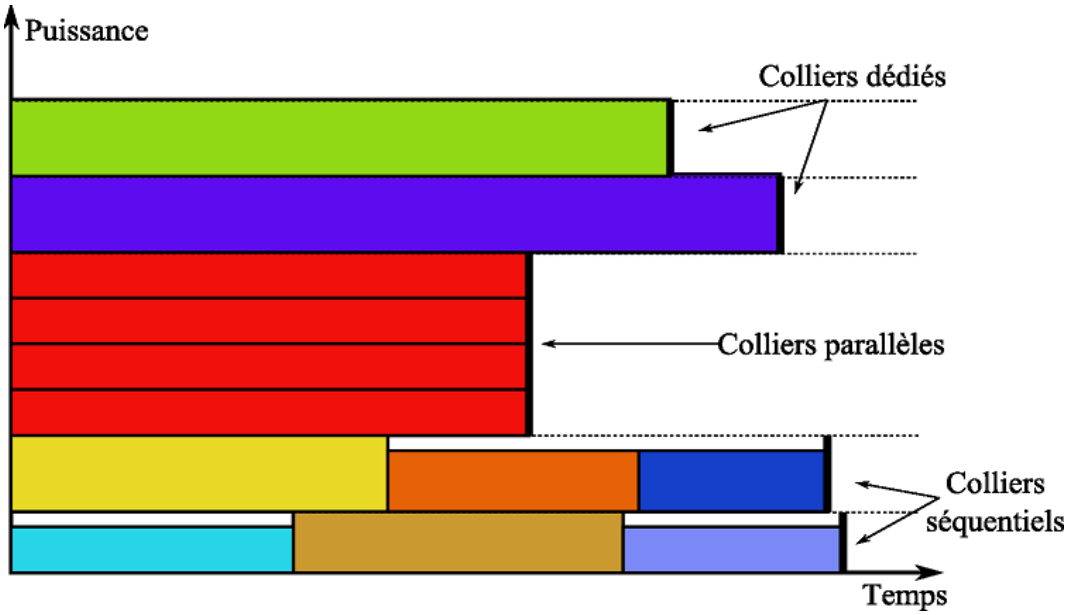

FIG. 6.1: Diagramme puissance-temps de test pour chaque type de collier.

6.2.3 Optimisation multi-objectif et solution de compromis

L'optimisation multi-objectif est née du besoin en industrie de satisfaire plusieurs critères contradictoires, simultanément. Les bases de cette optimisation ont été posées par *Pareto* et *Edgeworth* au 19ème siècle (*Pareto,* 1896). La théorie trouve ses premières applications en économie et ces dernières années dans les sciences pour l'ingénieur. Dans ce domaine, nous pouvons citer par exemple les travaux de : *Talbi*, en 1999; *Fonseca, et al.*, en 1995; *Gandibleux, et al.*,en 2004; *Hao, et al.*, 1999; *Reeves*, en 1995; *Collette*, en 2002; et plus récemment *Reyes-Sierra, et al.*, en 2006.

Les approches de résolution des problèmes multi-objectif peuvent être réparties en trois classes : approches basées sur la transformation du problème en un problème mono-objectif (simple objectif), approches non-Pareto et approches Pareto.

Dans la suite, nous faisons appel à l'optimisation multi-objectif basée sur l'approche Pareto, afin de résoudre le problème de partage de bloc *BIST* pour le test des mémoires. Cette approche consiste à utiliser la notion de *dominance* pour converger vers un ensemble de solutions efficaces, et sélectionner, en fin d'analyse, la solution optimale, au sens d'un compromis entre les différentes fonctions objectif.

Ce concept permet donc d'avoir un ensemble de solutions, dans lequel nous ne retiendrons que les solutions non dominées, afin de ne pas favoriser un critère par rapport à un autre. Avant de présenter les détails de notre algorithme nous précisons les concepts suivants.

6.2.3.1 Quelques définitions

Optimum au sens de Pareto

Vilfredo Pareto est un mathématicien italien du XIXème siècle (*Pareto, 1896*). Il a posé les bases de la solution d'un problème économique multi-objectif : « Dans un problème multi-objectif, il existe un équilibre tel que l'on ne peut améliorer un critère sans détériorer au moins un des autres ». Cet équilibre est appelé *optimum Pareto*. Plus formellement on a les définitions suivantes.

Définition 10. *Une solution x est dite Pareto-optimale si elle n'est dominée par aucune autre solution appartenant à A (domaine des solutions). Ces solutions sont appelées solutions non dominées.*

La notion de dominance Soit x une solution admissible au problème multi-objectif, $x \in A$ *domine* $x^{'} \in A$ si et seulement si $\forall i, f_i(x) \leq f_i(x^{'})$ avec au moins un i tel que $f_i(x) < f_i(x^{'})$
où $i = 1, ..., n$ et n est le nombre d'objectifs.

Une solution x est dite *faiblement non dominée*, s'il n'existe pas de solution $x^{'} \in A$ telle que $f_i(x) < f_i(x^{'})$

Une solution x est dite *fortement non dominée*, s'il n'existe pas de solution $x^{'} \in A$ telle que $f_i(x) \leq f_i(x^{'})$ avec au moins un i tel que $f_i(x) < f_i(x^{'})$ où $i = 1, ..., n$ et n est le nombre d'objectifs.

La figure 6.2 illustre la définition de la notion de dominance. Dans cet exemple, le problème multi-objectif consiste à minimiser à la fois f_1 et f_2. Les solutions représentées par les points A et B ne sont dominées par aucune autre solution.

La frontière de Pareto La frontière, appelée aussi le *front de Pareto*, est l'ensemble des points Pareto-optimaux. La figure 6.2 présente un exemple où le front de Pareto est la ligne rouge (Pareto).

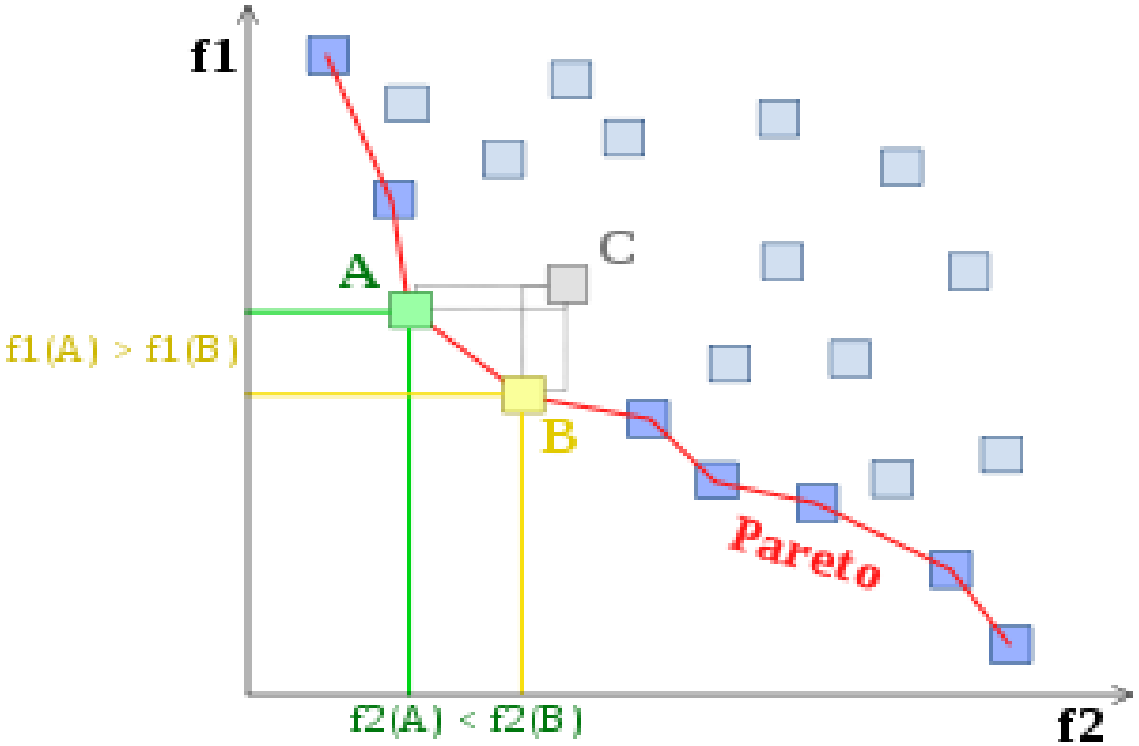

FIG. 6.2: Front de Pareto.

6.2.3.2 Les algorithmes génétiques

La taille du problème ne nous permet pas de calculer l'ensemble du front de pareto pour le problème de partage des blocs *BIST*. Nous optons pour une approche non déterministe nous permettant de générer un ensemble de solutions de qualité et diversité satisfaisantes. L'utilisateur peut ensuite choisir celle qui lui convient le mieux.

L'objectif de ce travail était de produire une méthode automatique pour optimiser le partage des blocs *BIST* pour les tests intégrés des mémoires. Nous avons cherché un algorithme suffisamment rapide et efficace pour résoudre ce problème multi-objectif. Notre choix s'est porté sur les Algorithmes Génétiques pour l'optimisation multi-objectif.

Les Algorithmes Génétiques (AG) appartiennent à la classe des algorithmes évolutionnaires, comme présenté au chapitre 1. Ce sont des méta-heuristiques inspirées de l'analogie avec l'évolution des êtres vivants. La simulation des mécanismes de variation et de sélection œuvrant dans les processus évolutifs naturels est exploitée pour résoudre des problèmes artificiels d'optimisation [30].

Dans les AG, l'analogie avec l'optimisation consiste à considérer les solutions potentielles au problème comme des *chromosomes*. Ceux-ci sont manipulés par des opérateurs de *sélection*, *mutation* et *croisement*. La qualité de la solution correspondant à chaque chromosome est quantifiée par sa propre *fitness*. Les procédures de croisement et de mutation ont pour but de créer de manière permanente de nouveaux chromosomes.

Les AG ont été très utilisés pour résoudre les problèmes multi-objectives [54]. Une étude comparative de quelques algorithmes pour ces problèmes se trouve dans le livre de *Zitzler* [70]. De plus, de nombreuses applications réelles les utilisent en particulier en microelectronique comme les travaux de *Melik-Adamyan* [23] par exemple.

L'idée principale est de chercher une collection de solutions non dominées. Dans ce type d'approche, deux familles de méthodes se dégagent : les méthodes *non-élitistes* et les *méthodes élitistes*.

Parmi les méthodes *non-élitistes*, nous pouvons citer la méthode *MOGA* (« *Multiple Objective Genetic Algorithm* ») proposée par *Fonseca et al.* [24], où chaque individu est rangé suivant le nombre d'individus qu'il domine, et la méthode *NSGA* (« *Non dominated Sorting Genetic Algorithm* ») proposée par *Fonseca et al.* [24] où le calcul de la fitness s'effectue en séparant la population en plusieurs groupes, en fonction du degré de domination au sens de Pareto de chaque individu.

Les techniques *élitistes* ne sauvegardent pas les individus dominés trouvés au cours des itérations. Elles se distinguent de deux manières : la difficulté de maintenir la diversité des individus (solution) et leur lenteur de convergence vers la frontière de Pareto. Parmi les nombreux algorithmes adoptant une stratégie élitiste, nous citons l'algorithme *SPEA* (« *Strength Pareto Evolutionary Algorithm* ») proposé par *Zitzler* [70], où le passage d'une génération à l'autre commence par la mise à jour des sauvegardes. Tous les individus non-

dominés sont sauvegardés et les individus dominés, déjà présents, sont éliminés. Après avoir amélioré cet algorithme, de meilleures performances ont été enregistrées, mais au prix de l'accentuation de sa complexité comme présenté par *Zitzler et al.* dans [42].

L'algorithme le plus connu et le plus privilégié est *NSGAII*, une deuxième version de *NSGA* proposée par *Deb* dans [20]. Dans cette version, l'auteur tente de résoudre toutes les critiques faites sur *NSGA* en termes de complexité, non-élitisme et utilisation du partage. Dans le cadre de notre travail, nous proposons d'utiliser cet algorithme.

Nous invitons le lecteur à consulter l'ouvrage de *Collette et al.* [18] et celui de *Zbigniew* [69] pour une vue plus complète des différents algorithmes génétiques pour l'optimisation multi-objectif. Nous exposons les détails de notre implémentation des *GA* dans la section suivante.

6.3 Méthode de résolution : Memory BIST Optimizer

En raison de la difficulté de prendre en charge l'ensemble des paramètres, nous avons décidé d'opter pour un schéma de résolution en trois étapes. La première étape est la traduction du problème réel en un modèle abstrait. La deuxième est la résolution du modèle mathématique. La troisième est la validation des solutions.

Cette décomposition permet également de s'affranchir partiellement des paramètres liés aux changements de nœuds technologiques pour les données en entrée de l'étape d'optimisation, comme nous le verrons par la suite.

Pour cela nous avons conçu le prototype *Memory BIST Optimizer* (*MBO*) que nous décrivons ci-dessous.

Memory BIST Optimizer Le prototype *Memory BIST Optimizer* (*MBO*) est principalement composé de trois modules.

Le premier module effectue le *Groupement des mémoires GRP*. Il prend toutes les données électroniques du problème donné (les paramètres de la mémoire, les fréquences de fonctionnement, les règles de partage, les groupes définis par l'utilisateur, etc) pour les traduire dans un format indépendant du nœud technologique approprié. Ceci est nécessaire pour que l'entrée du module suivant soit générique.

Le deuxième module est le module d'*Optimisation OPT*. Il est en charge de trouver plusieurs solutions admissibles au problème du partage des blocs *BIST* pour le test intégré des mémoires.

Enfin, le dernier module est le *Validateur VAL*. Il valide les solutions fournies par le module *OPT* et aide l'utilisateur à choisir sa solution préférée parmi celles calculées à travers une interface graphique ou textuelle. Il sert aussi à traduire la solution obtenue en une représentation électronique, en remplissant tous les paramètres de conception requis pour les colliers.

Cette méthode permet une séparation nette entre le problème électronique et la formulation du modèle mathématique. Cela permet à l'algorithme d'optimisation d'être indépendant des changements de technologies. Une autre caractéristique de ce partage est

la facilité avec laquelle la méthode peut être validée. En effet, il suffit d'adapter le module *GRP*, à travers un script en entrée et de veiller à ce que les solutions générées soient toujours valides à travers le module *VAL* pour s'adapter à un nouveau nœud technologique.

La figure 6.3 illustre les liens entre les modules du prototype *Memory BIST Optimizer*. Nous détaillons dans les sections suivantes chaque module.

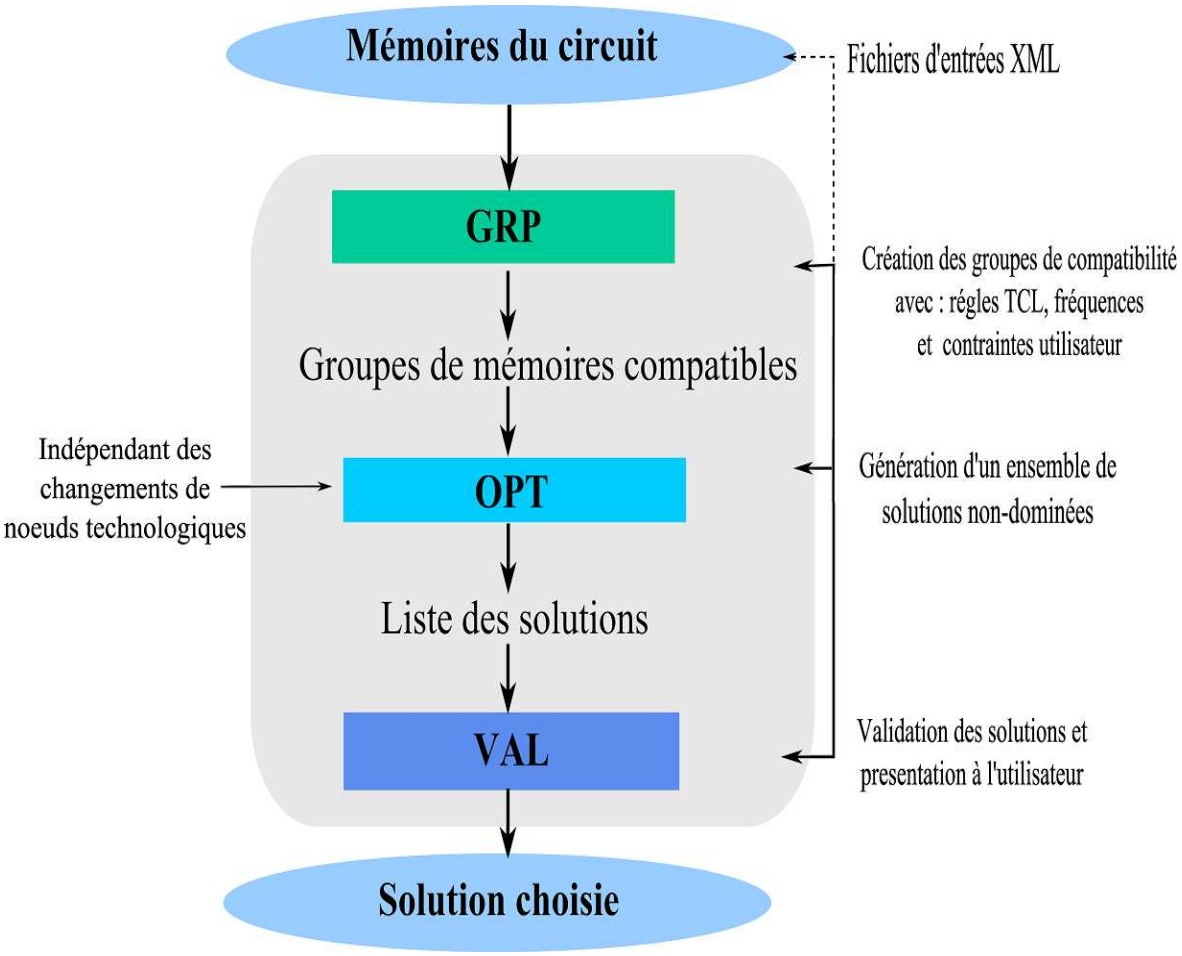

FIG. 6.3: Memory BIST Optimizer.

6.3.1 Module 1 : Modélisation et création des groupes de compatibilité

Comme les règles de partage de colliers parallèles et séquentiels changent à chaque nouvelle génération de nœuds technologiques, le rôle du module *GRP* est de générer des groupes de mémoires pouvant partager le même collier (parallèle ou séquentiel). Ceci afin

de fournir en entrée du module d'optimisation des données génériques.

Ce module revient donc à construire des classes d'équivalences pour les relations de compatibilités séquentielles et parallèles pour l'ensemble des mémoires. Ces classes correspondent à une partition totale du système *BIST* en groupe de compatibilité. Le groupage se fait en deux étapes comme suit.

Nous classons d'abord les types de mémoire (*memory cut*), en fonction de leur caractéristiques, pour les parallèles et les séquentielles. On parcourt tous les ensembles de types de mémoires pour lesquels un collier doit être défini pour créer des groupes de types mémoires compatibles. On analyse l'ensemble des types mémoire par paire afin de créer des groupes de compatibilités ou de mettre les mémoires dans les groupes déjà existants. Cette étape s'effectue en $O(n)$ opérations où n est le nombre de types de mémoire dans la conception.

Ensuite, à partir des ensembles de types mémoires compatibles, cette classification est affinée afin d'en déduire des groupes d'instances mémoires compatibles. Ces groupes sont obtenus en divisant les groupes de compatibilités des ensembles de types de mémoire (*memory cut*) en tenant compte de la période fonctionnelle (*functional period*) de l'instance mémoire et du paramètre *user_ define_ group* défini par l'utilisateur. Cette étape s'effectue en $O(m)$ opérations où m est le nombre de mémoires du circuit (nombre total d'instances).

La fiqure 6.4 illustre ce module et son intégration dans le processus global. A gauche, nous avons les données d'entrées : les mémoires 1, 2 et 3 avec leurs propriétés. A partir de ces entrées, on construit les groupes de compatibilités séquentiels ou parallèles représentés par la partie droite de la figure.

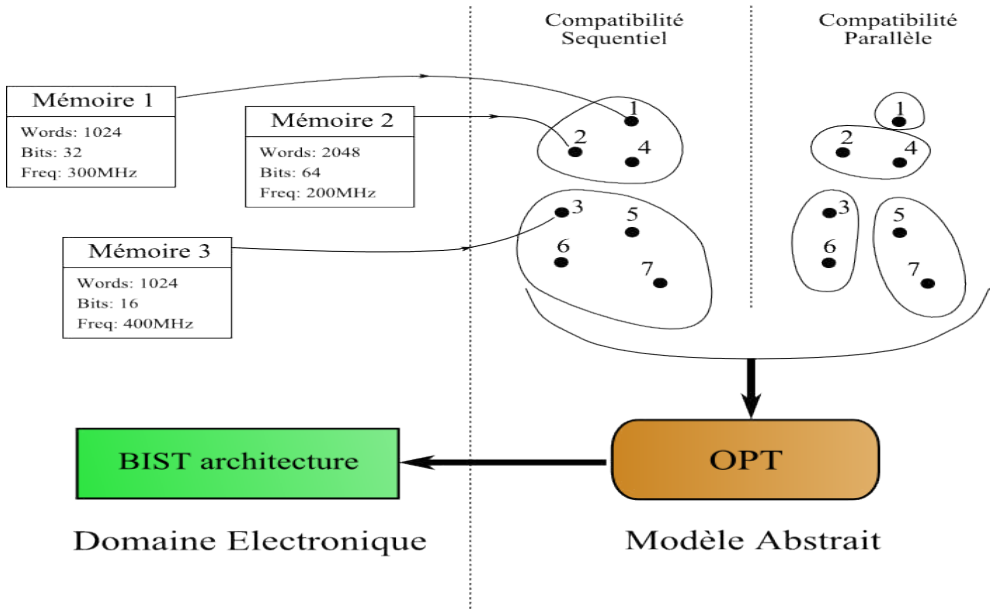

Fig. 6.4: Intégration du module GRP.

A la fin de cette phase, pour chaque instance de mémoire dans le système de *BIST* les informations suivantes sont fournies en entrée du module *OPT* :

- quelles mémoires peuvent être partagées sous un même collier en parallèle ;
- quelles mémoires peuvent être partagées sous un même collier en séquentiel ;
- le temps nécessaire pour tester la mémoire ;
- la consommation de puissance pour tester cette mémoire ;
- le nombre de bits de la mémoire, utilisé pour évaluer la surface qui lui est associée.

Ces informations sont les entrées du module OPT pour la deuxième étape de résolution.

6.3.2 Module 2 : Optimisation des mémoires

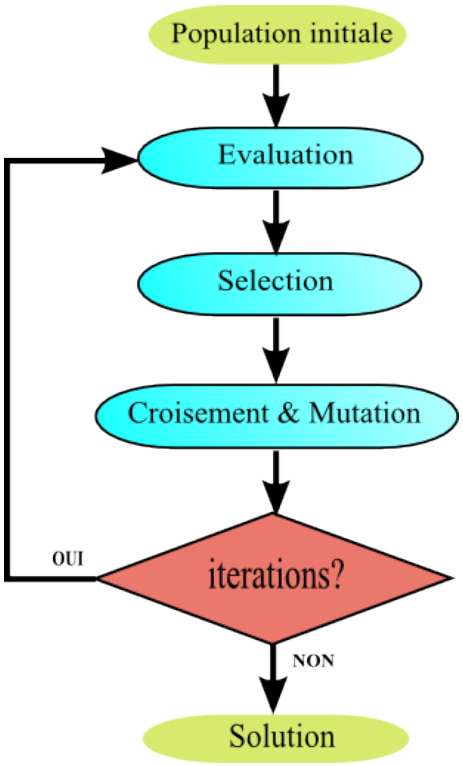

FIG. 6.5: Processus d'exécution des algorithmes génétiques.

Cette étape utilise un algorithme génétique adapté pour l'optimisation multi-objectif afin de résoudre le problème d'optimisation du partage des blocs *BIST* pour les tests des mémoires.

La figure 6.5 représente les différentes étapes du processus de l'algorithme génétique.

Pour appliquer les algorithmes génétiques à notre problème, un codage approprié doit être choisi. Ce codage détermine la mise en œuvre des opérateurs de sélection, de croise-

ment et de mutation. La représentation de la solution et des opérateurs utilisés dans *OPT* sont décrits ci-dessous.

Les solutions *BIST* sont représentées par un tableau de taille m, où m est le nombre d'instances mémoires dans le circuit. Chaque élément de ce tableau fournit des informations sur le type de collier associé à la mémoire. Un collier est défini par son identifiant (*ID_collar*) et sa configuration (séquentielle ou parallèle). Le tableau 6.1 montre un exemple d'un système avec 5 mémoires.

Mem1	Mem2	Mem3	Mem4	Mem5
collier1	collier2	collier1	collier3	collier3
Parallèle	Dédié	Parallèle	Séquentiel	Séquentiel

TAB. 6.1: Système BIST pour 5 mémoires.

Population initiale

Afin de pouvoir proposer à l'utilisateur des solutions finales diversifiées, les individus de la population initiale sont générés aléatoirement.

Pour créer les solutions aléatoires, nous prenons un collier au hasard pour chaque mémoire. La création aléatoire d'une solution est décrite comme suit.

Pour chaque mémoire, on calcule le nombre de configurations possibles. On choisit ensuite une configuration aléatoire parmi les configurations possibles. Si la configuration est dédiée alors un nouveau collier est créé pour la mémoire. Sinon, on créé une liste de mémoires qui sont compatibles (en parallèle ou séquentiel) avec cette mémoire à partir des groupes de compatibilités crés dans le module *GRP*. On choisit aléatoirement de cette liste des mémoires qui respectent les contraintes de validité d'une solution, c'est-à-dire le nombre maximum de mémoires par collier et le nombre maximum de bits par collier. On crée alors un nouveau collier partagé pour cette mémoire et les mémoires qui lui sont compatibles. Si la taille de cette liste est réduite à 1 alors un nouveau collier dédié est créé avec la mémoire qui reste.

Sélection

L'opérateur de sélection est basé sur la qualité des solutions. Celle ci est évaluée selon les trois critères : la surface, la puissance de test et le temps de test.

La sélection se fait par tournoi binaire. Les individus sont choisis au hasard par paire ; seuls les meilleurs de chaque paire survivent. On utilise la dominance de Pareto pour déterminer le vainqueur du tournoi. La meilleure solution au sens de Pareto est insérée dans la nouvelle population.

On rappelle, qu'une solution est Pareto-dominé si elle est pire qu'une autre solution pour chaque fonction objectif : surface, puissance et temps de test. Le front Pareto est l'ensemble des solutions non-dominées.

Le nombre de solutions gardées après sélection est égal à la taille de la population initiale choisie.

Croisement
Plusieurs opérateurs de croisement ont été proposés [70]. Dans cette étude, l'opération de croisement utilisé s'effectue en un point du chromosome de la façon suivante.

On parcourt la population et pour chaque individu un nombre aléatoire entre 0 et 1 est généré. Si ce nombre aléatoire est inférieur à la probabilité de croisement, alors cet individu est choisi pour le croisement. L'opérateur de croisement est appliqué a chaque paire d'individus parents.

Pour croiser deux individus parents, le point de croisement est généré aléatoirement entre 1 et $m-2$, où m est le nombre de mémoires (taille du tableau représentant un individu) présentes dans le système *BIST*. Après avoir choisi un point de croisement, les informations de gauche du point de croisement des deux individus parents sont copiées dans les individus fils, la partie droite des individus fils est prise en échangeant la partie droite des deux individus parents.

Dans le contexte du problème de partage des blocs *BIST*, l'opérateur de croisement consiste à échanger les configurations des colliers associés à chaque mémoire. Avant d'effectuer cet échange, il faut vérifier si, dans le cas d'une configuration parallèle ou séquentielle, la mémoire est partageable en parallèle ou en séquentielle. Si ce n'est pas le cas, la configuration initiale est choisie pour cette mémoire. On choisit arbitrairement celle du père.

Mutation
L'opérateur de mutation consiste à échanger de façon aléatoire, avec une probabilité P_m, deux colliers attribués à deux mémoires différentes. L'opération de mutation est appliquée avec une probabilité généralement faible par rapport au croisement.

Le choix d'un collier est effectué en générant un nombre aléatoire entre 0 et 1, si ce nombre est inférieur à la probabilité de mutation alors le collier est choisi. Si la configuration est initialement dédiée alors une configuration parallèle ou séquentielle est choisie aléatoirement et si la mémoire correspondante est partageable en parallèle ou en séquentielle.

Après la mutation, les individus ainsi creés remplacent leurs parents. Dans certains cas, il se peut que ces individus ne soient pas valides. Alors, une fonction de validation de solutions est appliquée pour vérifier et valider les solutions.

Une itération est composée des processus de sélection, croisement et mutation. La population fille obtenue en fin d'itération remplace la génération précédente. On arrête la reproduction au bout d'un certain nombre de générations. Ce critère d'arrêt est fixé de manière expérimentale en fonction du temps de calcul désiré.

6.3.3 Module 3 : Validation des colliers pour les mémoires

Ce module de validation *VAL* permet de réaliser deux tâches.

La première est de vérifier qu'une solution obtenue à partir du module *OPT* satisfait toutes les contraintes extraites par *GRP*, et donc de la valider.

Cela se fait comme suit :
i) Une solution est valide si :
– tous les colliers sont valides ;
– toutes les instances de mémoire présentes dans le système sont exactement dans un collier.
ii) Un collier est valide si :
– toutes les mémoires qu'il contient sont compatibles les unes avec les autres ;
– le nombre de mémoires dans le collier n'excède pas le nombre maximum autorisé pour cette configuration ;
– la somme des bits mémoires attachées au collier ne dépasse pas la limite autorisée ;

La deuxième tâche de ce module consiste à calculer pour chaque solution les valeurs des trois fonctions objectifs : surface, consommation et temps de test.

Le module présente ensuite les solutions à l'utilisateur sous forme textuelle et graphique. L'utilisateur peut alors choisir sa solution.

6.4 Implémentation & résultats numériques

Le prototype *Memory BIST Optimizer* (*MBO*) a été implémenté en *Java eclipse*. Il compte environ 18000 lignes de code, et fait un grand usage de formats de fichiers *XML*. Pour les besoins particuliers de STMicroelectronics, les règles de groupement sont données sous la forme de scripts *TCL*.

MBO comporte une interface purement textuelle, une interface graphique intéractive, et des sorties graphiques pour une utilisation non-intéractive.

Memory BIST Optimizer a déjà été intégré dans le flot de conception et test des *BIST* mémoire de notre partenaire STMicroelectronics et sera dans un futur très proche accessible à ses clients. Nous ne présentons dans cette section que quelques résultats numériques les plus significatifs obtenus sur des circuits réels fournis par STMicroelectronics.

6.4.1 Protocole expérimental et description des instances

L'évaluation numérique de la méthode a été effectuée en utilisant des circuits industriels fournis par STMicroelectronics. Les expérimentations sont faites sur un ordinateur de bureau doté d'un processeur *Intel Pentium 4* à $2,4$ *GHz* avec $3,45$ *Go* de *RAM*. Nous présentons ici les trois plus grands ensembles de mémoires pris en considération. Chaque ensemble de données correspond à un sous-bloc IP (*Intelectual Property*) pour lequel un système *BIST* doit être défini.

Le premier circuit contient 65 instances mémoires et le second 78. Ce sont des sous-blocs d'un circuit pour applications mobile/téléphone portable dernière génération. Le troisième circuit est le plus grand. Il dispose de 94 mémoires. C'est un sous-bloc d'un circuit destiné a être utilisé dans des imprimantes.

Pour les trois circuits, l'algorithme génétique a été utilisé avec une population initiale de 200 individus, évoluant en 5 générations. La probabilité de croisement est fixée à $0,995$

et la probabilité de mutation à $0,05$. Ces paramètres ont été choisis après plusieurs essais de différentes valeurs possibles. Nous présentons ici uniquement les résultats significatifs.

En raison des problèmes de routage et de congestion, le nombre maximum de mémoires autorisées pour chaque collier est de 4. Ce chiffre nous a été fourni par les ingénieurs *DFT* de STMicroelectronics. Il a été fixé d'après des données et paramètres physiques issus des usines de fabrication de circuits sur silicium.

6.4.2 Résultats numériques & Discussion

Le tableau 6.2, montre pour chaque circuit les valeurs des trois objectifs : la surface (A pour Area), la Puissance (P) et le Temps de test (T) pour la solution dédiée (D), et pour trois solutions proposées par *Memory BIST Optimizer* (MBO), (S_0, S_1, S_2). La colonne 1 donne le nom du circuit, la colonne 2 les critères d'optimisation. La colonne 3 représente la solution dédiée qui est notre référence. Les colonnes 4, 5, et 6 donnent les trois solutions S_0, S_1 et S_2. Les résultats que nous donnons présentent les écarts calculés par rapport à la solution dédiée.

Pour des raisons de confidentialité, les valeurs réelles ont été cachées par l'application d'un facteur multiplicatif. Par conséquent, aucune unité n'est spécifiée.

Type solution	**MBO**	**Dédié**	**MBO**	**MBO**	**MBO**
Circuits	critères	D	S_0	S_1	S_2
Circuit 1 (65 mémoires)	T	38	40	38	38
	A	2 173	1 538	1 692	1 625
	P	973	774	734	786
Circuit 2 (78 mémoires)	T	122	125	122	130
	A	2 658	2 133	2 120	2 132
	P	1 192	975	1 052	972
Circuit 3 (94 mémoires)	T	40	40	50	43
	A	3 538	2 895	2 862	2 890
	P	1 543	1 317	1 219	1 252

Tab. 6.2: MBO Résultats.

Pour le *circuit 1* avec 65 mémoires, l'utilisateur peut choisir une solution correspondant à la durée du test d'une solution dédiée, qui donne une réduction de 22% de la surface et de 24% en puissance avec la solution S_1, ou la solution S_2 qui donne une réduction de 25% de la surface et une réduction de 19% en puissance. Ou encore, la solution S_0 qui réduit la surface de 29%, avec un temps de test 5% plus long.

Pour le *circuit 2* avec 78 mémoires, les trois solutions permettent de réduire la superficie de 20% ; S_1 avec un temps de test correspondant à celui de la solution dédiée, et une réduction de la puissance de seulement 12% ; S_0, prend 2% de temps de test en plus, avec une réduction de puissance de 18%.

Pour le *circuit 3* avec 94 mémoires, la solution S_0 correspond au plus petit temps de test, tout en permettant une surface de 18% de moins et 15% en moins de puissance que l'utilisation de colliers dédiés uniquement. Pour une augmentation de 9% du temps de test, la solution S_2 a une surface un peu plus petite, et une réduction de puissance de 19%. Si l'utilisateur est prêt à avoir plus de temps de test, il peut choisir S_1 qui donne 21% de réduction de puissance et 19% de réduction de surface, avec 25% de temps de test en plus.

La figure 6.6 donne la surface des trois circuits obtenus pour les différentes configurations ; dédiée et les trois solutions de *MBO*.

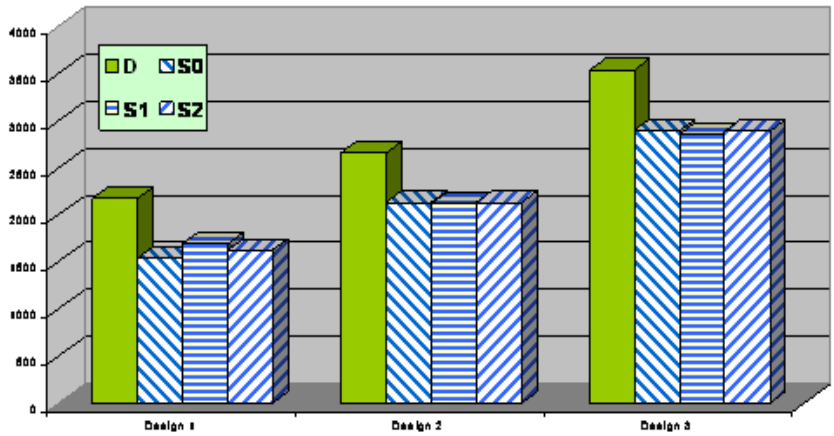

FIG. 6.6: La surface des trois circuits.

Pour tous les circuits, le temps d'exécution du module OPT pour l'optimisation est de moins de 20 secondes, tandis que le temps passé par le module GRP pour générer les groupes de compatibilité est inférieur à 3 secondes. La figure 6.7 donne le temps d'exécution en fonction de la taille de la population pour les deux modules *OPT* et *GRP* pour le *circuit 3* (la plus grande instance).

La figure 6.8 montre une sortie réelle de l'*Optimizer*, telle que présentée par le module *VAL*. Pour plus de clarté, nous présentons dans la suite un sous-ensemble de trois solutions pour le *circuits 3*. En outre, nous comparons la valeur des trois critères pour les solutions avec la solution ayant seulement des colliers dédiés.

Les figures 6.9, 6.10, 6.11 considèrent le circuit 3. Elles présentent, pour chaque paire de critères les différentes solutions, donnant trois points de vue différents sur l'ensemble des solutions. La figure 6.9 est la même que la figure 6.8 avec juste trois solutions.

L'outil *Memory BIST Optimizer* donne toujours ces trois points de vue, afin de permettre à l'utilisateur de prendre la décision qui lui convient le mieux.

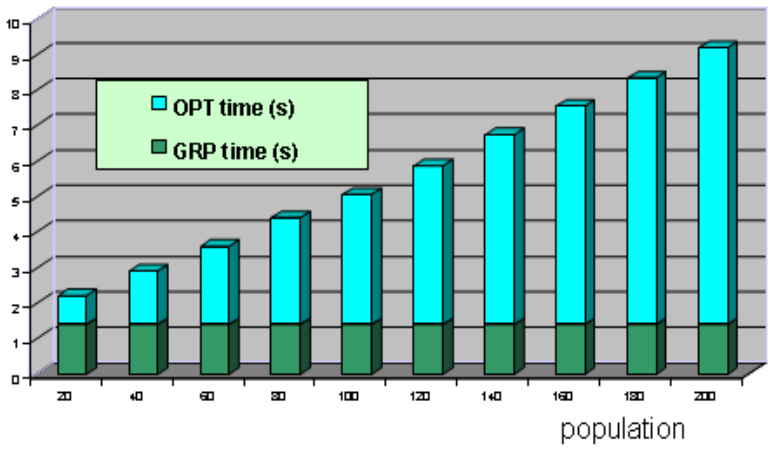

FIG. 6.7: Temps d'exécution de MBO.

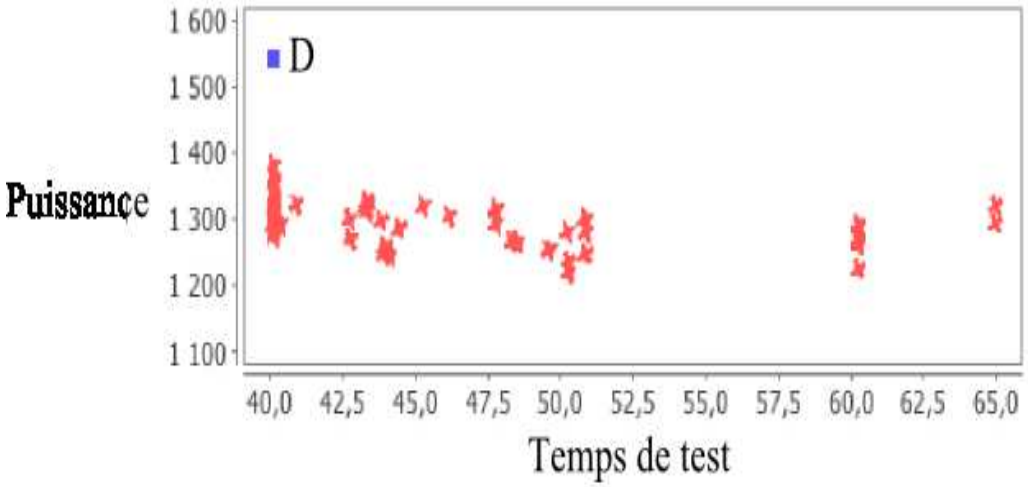

FIG. 6.8: Sortie du module OPT.

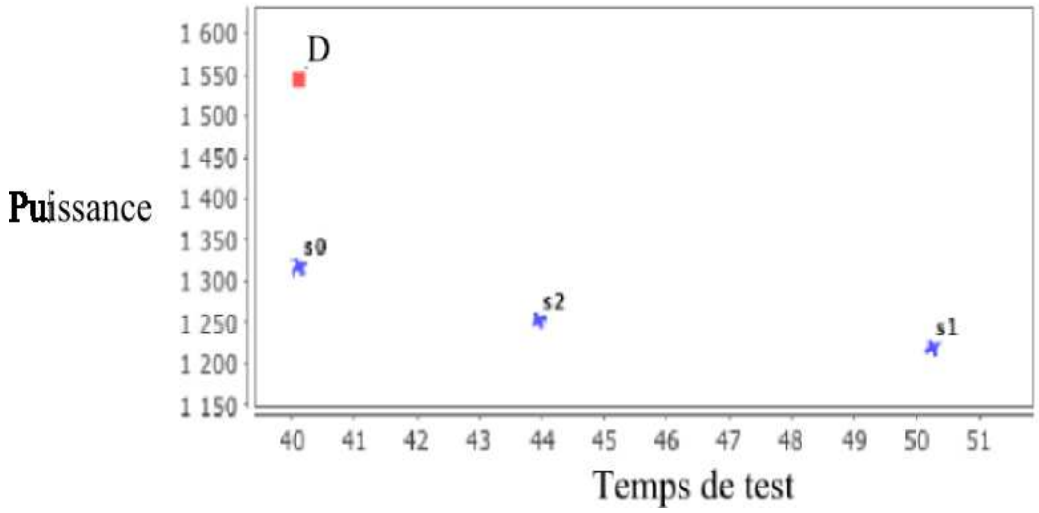

FIG. 6.9: Sortie simplifiée : Puissance vs Temps.

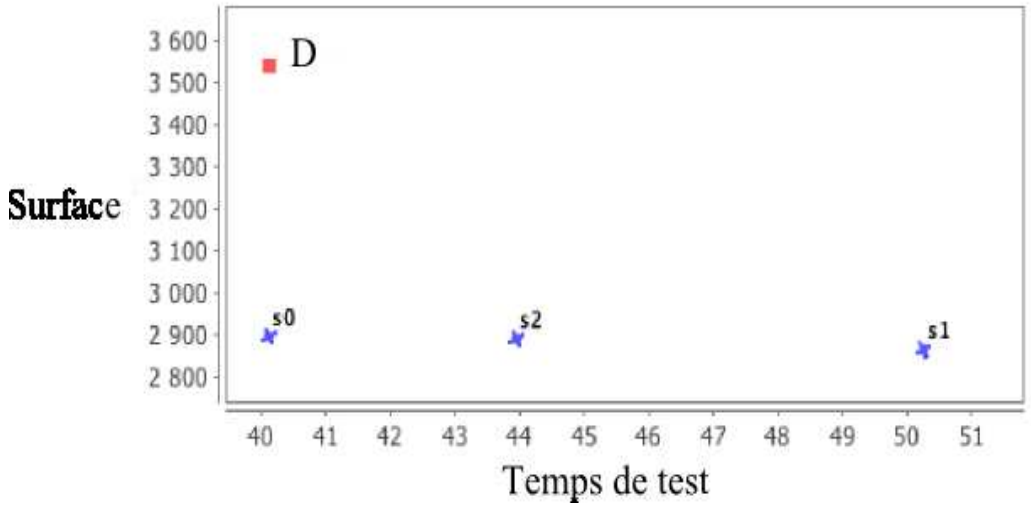

FIG. 6.10: Sortie simplifiée : Surface vs Temps test.

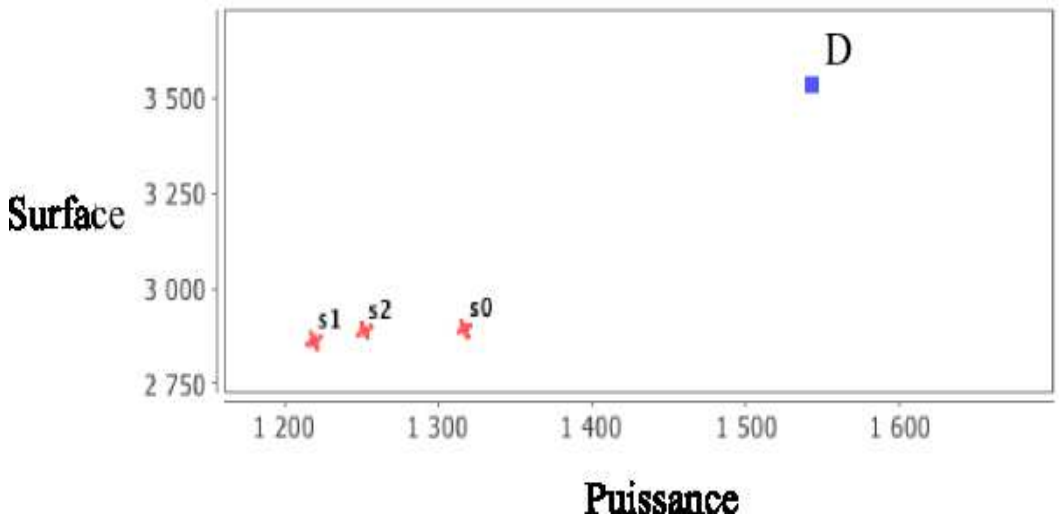

FIG. 6.11: Sortie simplifiée : Surface vs Puissance.

6.5 Conclusion

Comme nous l'avons vu tout au long de ce chapitre, le problème du partage des blocs *BIST* pour le test intégré des mémoires est un problème multi-objectif. En effet, la solution doit optimiser à la fois les trois critères : surface, puissance et temps de test.

Nous avons d'abord défini les formules utilisées pour mesurer les trois critères en fonction des types de collier : dédiés, partagés séquentiels et partagés parallèles.

Ensuite, un prototype nommé *Memory BIST Optimizer* a été conçu pour la résolution de ce problème.

Ce prototype écrit en *Java*, est entièrement intégré dans l'environnement de conception de notre partenaire industriel STMicroelectronics. Il prend en entrée l'ensemble des mémoires, les règles de partage et les paramètres utilisateur. Il fournit en sortie une architecture *BIST* partagée sous forme d'une interface (graphique ou textuelle) en fonction des besoins de l'utilisateur.

Ce prototype permet non seulement d'automatiser la génération de solutions de partage de bloc *BIST* pour le test des mémoires de différentes tailles et types, mais aussi d'optimiser le partage selon les 3 critères : surface, consommation en puissance, temps d'exécution.

Un atout majeur de cette méthode est que l'outil *MBO* est indépendant des changements de nœuds technologiques. En effet, seuls les paramètres d'entrée sont à mettre à jour pour un nouveau nœuds technologique. Il permet aussi à l'utilisateur de choisir sa solution. En effet, comme il s'agit d'un problème d'optimisation multi-objectif, est il alors important que l'utilisateur puisse ainsi décider du meilleur compromis pour la conception du système de partage de blocs *BIST*. Ceci est possible grâce à l'interface graphique des

solutions qui permet de visualiser l'ensemble des solutions non dominées et ainsi faciliter le prise de décision.

Des résultats prometteurs ont été obtenus pour différentes instances de tailles et de complexités variables fournies par STMicroelectronics. Les échanges avec l'industriel montrent que les résultats obtenus sont satisfaisants en termes de qualité des solutions et de temps d'exécution.

La méthode de résolution proposée ici avec le prototype *Memory BIST Optimizer* et les résultats numériques ont été présentés à *DATE 2010* [68], *ETS 2010* [60] et *EURO 2010* [67].

Conclusion générale

Le travail de thèse présenté dans ce mémoire se trouve à l'interface de la recherche opérationnelle et la microélectronique. L'ensemble des travaux présentés a pour thème l'utilisation des techniques de *recherche opérationnelle* pour la *conception testable* (*DFT pour Design For Test*) de circuits intégrés complexes. Il s'agit d'utiliser les méthodes d'optimisation combinatoire pour l'insertion des techniques de *DFT* à haut niveau.

Dans ce contexte, nous nous sommes focalisés plus particulièrement sur l'insertion des *chaînes de scan* au niveau *RTL* pour le test des circuits intégrés ainsi que du partage de blocs *BIST*(*Built In Self Test*) pour le test intégré des mémoires. Ces deux techniques sont aujourd'hui massivement utilisées pour le test et la *DFT* des circuits numériques.

Cela représente le passage d'un niveau d'abstraction (niveau porte) à un niveau plus haut (niveau *RTL*) pour l'insertion des techniques de *DFT*.

Une motivation que l'on retrouve dans l'ensemble des recherches menées est d'essayer de minimiser l'impact de l'insertion des techniques de *DFT* comme le *scan path* et le *BIST* sur les performances du circuit en termes de surface additionnelle, consommation en puissance et temps d'exécution de la procédure de test.

Cette motivation est l'essence de ce travail de recherche qui a aussi été guidé entre autre par des besoins qui sont apparus au cours de nos collaborations avec les industriels du domaine.

Le but de cette démarche est de montrer que la recherche opérationnelle peut apporter une aide pour la prise de décision afin d'optimiser l'insertion de la *DFT* au niveau *RTL* à travers le développement d'algorithmes efficaces et leurs validations sur des circuits intégrés réels.

Nous avons regroupé dans un premier temps les notions essentielles communes à tous les chapitres concernant le domaine de la recherche opérationnelle et celui de la conception et test des circuits intégrés.

Nous nous sommes intéressés dans un deuxième temps au problème de l'optimisation de l'insertion des *chaînes de scan* au niveau *RTL* dans la deuxième partie. Afin de mieux cerner les objectifs et contraintes de l'insertion du *scan* haut niveau, nous avons d'abord décrit les techniques utilisées au niveau porte pour l'insertion du *scan* dans le chapitre 3. Nous avons ensuite abordé la problématique de l'insertion de *chaînes de scan* au niveau *RTL*. On peut la décomposer en deux étapes : la détection d'éléments mémoires et signaux

au niveau *RTL* puis le *stitching* des éléments mémoires dans la chaîne.

Les travaux de recherche de *DeFacTo Technologies* concernent le premier problème. Notre contribution porte sur la modélisation et la résolution efficace du problème de *stitching* optimal des éléments mémoires dans la *chaîne de scan*.

Pour cela, nous avons construit deux graphes l'un reflétant la description du circuit l'autre les liens entre les éléments mémoires qui constitueront les futures chaînes de scan.

Nous avons ensuite, proposé dans le chapitre 4 deux stratégies pour ramener le problème de recherche de *chaînes de scan* dans un circuit au célèbre problème du voyageur de commerce (*TSP*) dans un graphe. Ceci nous a permis, d'adapter les heuristiques classiques disponibles dans la littérature pour résoudre le problème du *TSP* à notre problème à travers l'algorithme de *stitching*.

Les expériences menées sur des *benchmarks* disponibles pour l'étape de test ont montré que cette méthode fournit de bons résultats par rapport au *scan* au niveau porte.

L'apport principal de cette partie est de fournir un algorithme efficace capable d'automatiser l'insertion des *chaînes de scan* optimales au niveau *RTL* en une seule passe au lieu de plusieurs itérations au niveau porte.

Cette méthode a été validée par notre partenaire industriel *DeFacTo Technologies* et constitue un premier pas vers l'insertion du scan au niveau *RTL*.

Nos pas nous ont ensuite menés au problème d'optimisation de partage de bloc *BIST* pour le test intégré des mémoires. Cette problématique peut se décrire comme suit : étant donné un ensemble de mémoires à tester, l'objectif est de partager un collier entre plusieurs mémoires compatibles.

Cependant, il existe différents types de partage des colliers : le partage séquentiel qui permet de réduire la consommation et le partage parallèle qui permet de réduire le temps de la procédure de test. Les deux types de partages réduisent la surface. Il est aussi possible d'opter pour un collier dédié si la mémoire à tester possède certaines propriétés (comme une grande taille par exemple).

Notre objectif est d'identifier de bonnes solutions de compromis pour des architectures de partage *BIST* entre les trois critères : surface additionnelle, puissance de test et temps de test.

Pour cela, nous avons conçu un prototype nommé *Memory Bist Optimiser* (MBO). Ce dernier est composé de trois modules. Le premier est pour le *Groupement* des mémoires. Il permet de créer des groupes de mémoires compatibles pour le partage séquentiel ou parallèle d'un collier. Il sert aussi de module d'abstraction entre les données électroniques du problème et les entrées du deuxième module qui est celui de l'*Optimisation*. Il utilise les algorithmes génétiques adaptés pour l'optimisation multi-objectif. Notre idée est de fournir un ensemble de solutions non dominées. L'utilisateur peut ainsi choisir celle qui le satisfait le plus. Enfin le dernier module est celui de la *Validation*. Il a un double rôle. D'une part, il permet de valider les solutions proposées par le module précédent. D'autre part, il affiche à travers une interface graphique ou textuelle les solutions possibles afin d'aider l'étape de prise de décision à l'utilisateur.

Ce découpage permet de faciliter l'intégration de l'ensemble des données, contraintes

et objectifs ainsi que leurs mise à jour. Il facilite également les échanges avec l'industriel. En effet, pour valider une solution par exemple, le client doit juste mettre à jour le module de *Validation* et n'a pas besoin de se préoccuper des modules précédents.

Nous obtenons des résultats très satisfaisants avec ce prototype sur des instances fournies par STMicroelectronics.

Rapide, simple et efficace *MBO* a déjà été intégré dans le flot interne de conception et test des mémoires à *STMicroelectronique*.

Un atout majeur des modèles, algorithmes proposés ainsi que le prototype développé dans ce travail est l'indépendance par rapport aux changements de nœuds technologiques.

Nous souhaitons insister ici, plus particulièrement sur la démarche de modélisation et contact « direct » avec le monde de la microélectronique et du test des circuits. Ceci nous a permis de mettre à l'épreuve les connaissances de recherche opérationnelle en expérimentant les algorithmes sur des données réelles et en évaluant leurs bénéfices en industrie. Le second point fort réside dans la souplesse d'utilisation qui permet de prendre en compte de nouvelles contraintes, comme l'insertion de bloc d'*IP* additionnels dans le code *RTL* pour le *scan* ou encore l'ajout de nouvelles règles de partage pour le *BIST*. Il suffit de mettre à jour le modèle sans refaire de nouveaux algorithmes d'optimisation.

Ces avantages ont été confirmés par nos collaborations industrielles tant avec un vendeur d'outils de *CAO* comme *DeFacTo Technologies* qu'avec un utilisateur de ces outils comme *STMicroelectronics*.

Perspectives et futurs travaux :

Suite à cette thèse, de nombreuses pistes de travail découlent ; d'une part, il reste du terrain à défricher autour de nos modèles mathématiques et de nos résultats numériques afin de les analyser et de les étendre, d'autre part de nouveaux problèmes sont apparus aussi bien du coté pratique que du coté théorique.

En effet, au cours de cette thèse nous avons croisé plusieurs problèmes d'optimisation combinatoire et de théorie des graphes comme celui de la couverture des sommets d'un graphe par des chaînes.

Dans la continuité directe de ce travail de thèse, il me semble intéressant d'approfondir cette partie théorique concernant la couverture des sommets d'un graphe par des chaînes de taille fixée par rapport au nombre de sommets.

Il serait aussi intéressant de mesurer la qualité du modèle de graphe proposé pour la génération des chaînes de *scan*.

Dans un deuxième temps nous pourrions envisager de tester nos algorithmes sur de plus gros circuits disponibles comme l'open *sparc* par exemple. Nous pourrions aussi examiner et proposer des algorithmes afin d'optimiser l'insertion du *scan* partiel.

En ce qui concerne la partie *BIST*, il me semble fondamental de résoudre le problème de partage de blocs *BIST* avec des méthodes exactes, en relâchant l'objectif de la surface. En effet, si on part du principe que le fait de partager des colliers permet de toute façon

de gagner de la surface (quelque soit le type de partage), on se ramène à un problème ou il s'agit de trouver un compromis entre le temps et la puissance d'exécution.

Il est aussi envisagé de tester ce prototype sur d'autres circuits à travers une future collaboration avec d'autres équipes de recherches outre atlantique.

Par ailleurs, une fois que l'architecture de partage de blocs *BIST* est définie, il reste le problème de l'ordonnancement de l'exécution de ces tâches par le contrôleur. Des premières pistes sont en cours d'exploration avec des équipes de recherche spécialisées dans le domaine de l'insertion et partage du *BIST* mémoire [9].

Dans une perspective plus globale, nous aimerions continuer à utiliser la recherche opérationnelle pour l'optimisation en microélectronique. Et ainsi profiter de l'enrichissement mutuel entre les deux disciplines...

Bibliographie

[1] M. Abramovici, M. Breuer, and A. Friedman. *Digital Systems Testing & Testable Design*. Wiley-IEEE Press, 1994.

[2] C. Aktouf. *The Computer Engineering Handbook*, pages 461–469. Ed. CRC Press, January 2002.

[3] C. Aktouf. Testing on-chip multiprocessor architectures. *IEEE Design and Test of Computers*, pages 18–28, January 2002.

[4] C. Aktouf. Closing the Gap between RTL and Design for Test. *Chip Design Magazine*, October 2005.

[5] C. Aktouf. Removing bottlenecks from your SoC Design-For-Test flows. *EETimes Magazine*, December 2009.

[6] C. Aktouf, H. Fleury, and C. Robach. Inserting scan at the behavioral level. *IEEE Design & Test of Computers*, 17(3):34–42, 2000.

[7] A. Alj, R. Faure, C. Brahimi, and P. Lignelet. *Guide de la recherche opérationnelle*. Masson, 1986.

[8] D. Applegate, R. Bixby, V. Chvatal, D. Espinoza, M. Goycoolea, and K. Helsgaun. Certification of an optimal TSP tour through 85,900 cities. *Operations Research Letters*, 37:11–15, 2009.

[9] A. Benso, S. Di Carlo, G. Di Natale, and P. Prinetto. A Programmable BIST Architecture for Clusters of Multiple-Port SRAMs. In *Proc. IEEE Int. Test Conf*, page 557, Washington, DC, USA, 2000. IEEE Computer Society.

[10] A. Benso, S. Di Carlo, G. Di Natale, and P. Prinetto. Programmable Built-In-Self-Testing of Embedded RAM Clusters in System-on-Chip Architectures. *Communications Magazine, IEEE* , pages 90 – 97, Sept 2003.

[11] C. Berge. *Théorie des graphes et ses applications*. Dunod Paris, 1958.

[12] D. Berthelot, S. Chaudhuri, and H. Savoj. An efficient linear time algorithm for scan chain optimization and repartitioning. page 781. Proc. IEEE Int. Test Conf, 2002.

[13] C. Bichot. *Elaboration d'une nouvelle métaheuristique pour le partitionnement de graphe*. PhD thesis, Institut National Polytechnique de Toulouse, Novembre 2007.

[14] K.D. Boese, A.B. Kahng, and R.S. Tsay. Scan chain optimization: Heuristic and optimal solutions. *Research Report UCLA*, 1994.

[15] Y. Bonhomme, P. Girard, L. Guiller, C. Landrault, and S. Pravossoudovitch. Efficient scan chain design for power minimization during scan testing under routing constraint. In *Proc. IEEE Int. Test Conf*, pages 488–493, 2003.

[16] J.P. Calvez. *Spécification Et Conception Des ASICs*. Masson, 1993.

[17] A. Cazes and J. Delacroix. Architecture des machines et des systèmes informatiques: cours et exercices corrigés (Coll. Sciences sup,). 2005.

[18] Y. Collette and P. Siarry. *Optimisation multiobjectif*. Eyrolles Paris, 2002.

[19] W.J. Cook, W.H. Cunningham, W. R. Pulleybank, and A. Schrijver. *Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics, 1998.

[20] K. Deb. A fast elitist non-dominated sorting genetic algorithm for multiobjective optimization: NSGA II. *IEEE Trans. on Evolutionary computation*, 5(3):115–148, 2002.

[21] J. Dréo, A. Pétrowski, P. Siarry, and E. Taillard. *Métaheuristiques pour l'optimisation difficile*. Eyrolles Paris, 2003.

[22] U. Elsner. Graph partitioning–a survey. *Technische Universitat Chemnitz*, 1997.

[23] Melik-Adamyan A. F. Application of genetic algorithms in problems of optimization of the physical design in microelectronics. *Automatic Documentation and Mathematical Linguistics*, pages 244–250, septembre 2009.

[24] C.M. Fonseca and P.J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary computation*, 3(1):1–16, 1995.

[25] J.C. Fournier. *Théorie des graphes et applications, avec exercices et problèmes*. Traité IC2, série informatique et systèmes d'information, collection Informatique, 2007.

[26] M.R. Garey and D.S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness. A Series of Books in the Mathematical Sciences*. WH Freeman and Company, San Francisco, Calif, 1979.

[27] P. Girard. Survey of low-power testing of VLSI circuits. *IEEE Design & Test of Computers*, 19(3):80–90, 2002.

[28] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

[29] F. Glover and R. Marti. Tabu search. *Metaheuristic Procedures for Training Neutral Networks*, pages 53–69.

[30] D.E. Goldberg. *Genetic Algorithms in Search and Optimization*. Addison-Wesley, 1989.

[31] O. Goldreich. *Computational complexity: a conceptual perspective*. Cambridge University Press, 2008.

[32] C. Guéret, C. Prins, and M. Sevaux. *Programmation linéaire*. 2000.

[33] P. Gupta, A.B. Kahng, and S. Mantik. Routing-aware scan chain ordering. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 10(3):560, 2005.

[34] J.F. Hêche, T.M. Liebling, and D. De Werra. *Recherche opérationnelle pour ingénieurs.* PPUR presses polytechniques, 2003.

[35] M. Hirech, J. Beausang, and X. Gu. A new approach to scan chain reordering using physical design information. In *Proceedings of the 1998 IEEE International Test Conference*, page 348. IEEE Computer Society, 1998.

[36] Y. Huang, C.C. Tsai, N. Mukherjee, O. Samman, W.T. Cheng, and S.M. Reddy. Synthesis of scan chains for netlist descriptions at RT-level. *Journal of Electronic Testing*, 18(2):189–201, 2002.

[37] G. Karypis and V. Kumar. METIS: Unstructured graph partitioning and sparse matrix ordering system. *The University of Minnesota*, 2, 1995.

[38] Y. Kieffer, J. Alami-Chentoufi, and L. Zaourar. Automatizing the sharing of bist blocks for low power testing of embedded-memories. *23rd European conference on operational Research. Bonn, Allemagne*, Juillet 2009.

[39] Y. Kieffer and L. Zaourar. Optimization for the test of on-chip memories. *International Symposium on Combinatorial Optimization. Hammamet, Tunisia*, Mars 2010.

[40] B. Korte, L. Lovasz, H.J. Promel, and A. Schrijver. *Paths, flows, and VLSI-layout.* Springer-Verlag, 1990.

[41] C. Landrault. *Test de circuits et de systèmes intégrés.* HERMES Lavoisier série EGEM, 2004.

[42] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler. Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary computation*, 10(3):263–282, 2002.

[43] R. Leveugle. Test des circuits intégrés numériques : Notions de base. Génération de vecteurs. *Techniques de l'ingénieur. Electronique*, 2(E2460):E2460, 2002.

[44] S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269, 1965.

[45] M. Lobetti Bodoni, A. Benso, S. Chiusano, S. Di Carlo, G. Di Natale, and P. Prinetto. An Effective Distributed BIST Architecture for RAMs. *European Test Workshop, IEEE*, page 119, 2000.

[46] J. Lu and C. Wu. Cost and benefit models for logic and memory bist. In *DATE '00: Proceedings of the conference on Design, automation and test in Europe*, pages 710–714. DATE '00: Proceedings of the conference on Design, automation and test in Europe, 2000.

[47] S. Masuyama and T. Ibaraki. Chain packing in graphs. *Algorithmica*, 6(1):826–839, 1991.

[48] P. Mazumder and K. Chakraborty. *Testing and testable design of high-density random-access memories.* Kluwer Academic Publishers, 1996.

[49] M. Miyazaki, T. Yoneda, and H. Fujiwara. A memory grouping method for sharing memory BIST logic. In *ASP-DAC '06: Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, pages 671–676, Piscataway, NJ, USA, 2006. IEEE Press.

[50] G.L. Nemhauser and L.A. Wolsey. *Integer and combinatorial optimization*. Wiley New York, 1999.

[51] R.B. Norwood and E.J. McCluskey. High-level synthesis for orthogonal scan. In *Proc. IEEE VLSI Test Symposium*, page 370, 1997.

[52] C.H. Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

[53] M. Sakarovitch. *Optimisation combinatoire: méthodes mathématiques et algorithmiques. Programmation discrète*. Hermann, 1984.

[54] E.G. Talbi. Métaheuristiques pour l'optimisation combinatoire multi-objectif: Etat de l'art. *Rapport CNET (France Telecom) Octobre*, 1999.

[55] E.G. Talbi. *Metaheuristics: from design to implementation*. Wiley, 2009.

[56] N. Teypaz and C. Rapine. Graph decomposition into paths under length constraints. *Cahiers Leibniz*, 2008.

[57] A.J. Van de Goor. *Testing semiconductor memories: theory and practice*. John Wiley & Sons, Inc. New York, NY, USA, 1991.

[58] L.T. Wang, C.W. Wu, and X. Wen. *VLSI test principles and architectures: design for testability*. Morgan Kaufmann Pub, 2006.

[59] L. Zaourar. Conception d'algorithmes d'optimisation pour la DFT. Master's thesis, Institut National Polytechnique De Grenoble, ENSIMAG, 2007.

[60] L. Zaourar, C.H Alami, Y. Kieffer, A. Wenzel, and F. Grandvaux. A solution to optimize shared memory bist system. *15th IEEE European Test Symposium. Prague, Czech Republic.*, Mai 2010.

[61] L. Zaourar, A.J. Chentoufi, Y. Kieffer, and A. Waserhole. Optimisation du partage de blocs bist pour le test des mémoires d'un circuit intégré. *11e Congrès annuel de la société française de Recherche Opérationnelle et d'Aide à la Décision. Toulouse, France*, Février 2010.

[62] L. Zaourar and Y. Kieffer. Modélisation et optimisation d'une méthode de test d'un circuit intégré. *Journées Graphes et Algorithmes, Nice, France*, November 2008.

[63] L. Zaourar and Y. Kieffer. Chaînage des sommets d'un graphe pour le test des circuits intégrés. *10ème congrès de la société Française de Recherche Opérationnelle et d'Aide à la Désicion. Nancy, France*, Février 2009.

[64] L. Zaourar and Y. Kieffer. Scan chain optimization for integrated circuits testing: an operations research perspective. *23rd European conference on operational Research. Bonn, Allemagne*, Juillet 2009.

[65] L. Zaourar, Y. Kieffer, and C. Aktouf. A practical scan optimization algorithm at the register transfer level. *IEEE Design Automation and Test in Europe. Nice, France*, Mars 2009.

[66] L. Zaourar, Y. Kieffer, C. Aktouf, and V. Julliard. A model for scan insertion at the register transfert level. *The Tenth IEEE Workshop on RTL and High Level Testing. Hong-Kong, Chine*, Novembre 2009.

[67] L. Zaourar, Y. Kieffer, and N. Brauner. Multi-objective optimization of memory built-in-self-test sharing. *24rd EURO European Conference On Operational Resarch. Lisbon, Portugal*, Juillet 2010.

[68] L. Zaourar, Y. Kieffer, N. Brauner, A. Wenzel, and F. Grandvaux. A solution to optimize shared memory bist system. *IEEE Design Automation and Test in Europe Conference. Dresden, Allemagne*, Mars 2010.

[69] M. Zbigniew. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag New York, LLC, March 1996.

[70] E. Zitzler. *Evolutionary algorithms for multiobjective optimization: Methods and applications.* Citeseer, 1999.

[71] Y. Zorian. A distributed BIST control scheme for complex VLSI devices. In *Proc. 11th IEEE VLSI Test Symposium*, pages 4–9, 1993.

Résumé :

Le travail de cette thèse est à l'interface des domaines de la recherche opérationnelle et de la micro-électronique. Il traite de l'utilisation des techniques d'optimisation combinatoire pour la *DFT* (*Design For Test*) des Circuits Intégrés (*CI*).
Avec la croissance rapide et la complexité des *CI* actuels, la qualité ainsi que le coût du test sont devenus des paramètres importants dans l'industrie des semi-conducteurs. Afin de s'assurer du bon fonctionnement du *CI*, l'étape de test est plus que jamais une étape essentielle et délicate dans le processus de fabrication d'un *CI*. Pour répondre aux exigences du marché, le test doit être rapide et efficace dans la révélation d'éventuels défauts. Pour cela, il devient incontournable d'appréhender la phase de test dès les étapes de conception du CI. Dans ce contexte, la conception testable plus connue sous l'appellation *DFT* vise à améliorer la testabilité des *CI*.
Plusieurs problèmes d'optimisation et d'aide à la décision découlent de la micro-électronique. La plupart de ces travaux traitent des problèmes d'optimisation combinatoire pour le placement et routage des circuits. Nos travaux de recherche sont à un niveau de conception plus amont, la *DFT en présynthèse au niveau transfert de registres ou RTL (Register Transfer Level).* Cette thèse se découpe en trois parties.
Dans la première partie nous introduisons les notions de bases de recherche opérationnelle, de conception et de test des *CI.* La démarche suivie ainsi que les outils de résolution utilisés dans le reste du document sont présentés dans cette partie.
Dans la deuxième partie, nous nous intéressons au problème de l'optimisation de l'insertion des chaînes de scan. A l'heure actuelle, le « *scan interne* » est une des techniques d'amélioration de testabilité ou de DFT les plus largement adoptées pour les circuits intégrés numériques. Il s'agit de chaîner les *éléments mémoires* ou bascules du circuit de sorte à former des *chaînes de scan* qui seront considérées pendant la phase de test comme points de contrôle et d'observation de la logique interne du circuit. L'objectif de notre travail est de développer des algorithmes permettant de générer pour un *CI* donné et dès le niveau RTL des *chaînes de scan* optimales en termes de surface, de temps de test et de consommation en puissance, tout en respectant des critères de performance purement fonctionnels. Ce problème a été modélisé comme la recherche de plus courtes chaînes dans un *graphe* pondéré. Les méthodes de résolution utilisées sont basées sur la recherche de *chaînes hamiltoniennes* de longueur minimale. Ces travaux ont été réalisés en collaboration avec la start-up *DeFacTo Technologies*.
La troisième partie s'intéresse au problème de partage de blocs *BIST* (*Built In Self Test*) pour le test des *mémoires*. Le problème peut être formulé de la façon suivante : étant données des mémoires de différents types et tailles, ainsi que des règles de partage des *colliers* en série et en parallèle, il s'agit d'identifier des solutions au problème en associant à chaque *mémoire* un *collier*. La solution obtenue doit minimiser à la fois la surface, la consommation en puissance et le temps de test du *CI*. Pour résoudre ce problème, nous avons conçu un prototype nommé *Memory BIST Optimizer* (*MBO*). Il est constitué de deux phases de résolution et d'une phase de validation. La première phase consiste à créer des groupes de compatibilité de mémoires en tenant compte des règles de partage et d'abstraction des technologies utilisées. La deuxième phase utilise les algorithmes génétiques pour l'optimisation multi-objectifs afin d'obtenir un ensemble de *solutions non dominées*. Enfin, la validation permet de vérifier que la solution fournie est valide. De plus, elle affiche l'ensemble des solutions à travers une interface graphique ou textuelle. Cela permet à l'utilisateur de choisir la solution qui lui correspond le mieux. Actuellement, l'outil *MBO* est intégré dans un flot d'outils a *ST-microelectronics* pour une utilisation par ses clients.