Synthèse d'architectures de circuits FPGA tolérants aux défauts

Adrien Blanchardon

UNIVERSITE PIERRE ET MARIE CURIE

N° attribué par la bibliothèque

THÈSE

pour obtenir le grade de

Docteur de l'Université de thèse

Spécialité : **Informatique**

préparée au laboratoire **d'Informatique de Paris 6**

dans le cadre de l'École Doctorale **d'Informatique,Télécommunications et d'Electronique**

présentée et soutenue publiquement
par

Adrien BLANCHARDON

le 15/09/2015

Titre:

Synthèse d'architectures de circuits FPGA tolérants aux défauts

Directeur de thèse: **Habib MEHREZ**
Co-directeur de thèse: **Roselyne CHOTIN-AVOT**

Jury

Mme. ANGHEL Lorena (Professeur TIMA),	Rapporteur
M. GIRARD Patrick (Professeur LIRMM),	Rapporteur
M. CHILLET Daniel (Professeur Université Rennes 1-IRISA),	Examinateur
Mme. ENCRENAZ Emmanuelle (MCF HDR LIP6),	Examinateur
M. MEHREZ Habib (Professeur LIP6),	Directeur de thèse
Mme. CHOTIN-AVOT Roselyne (MCF LIP6),	Co-directrice de thèse

Résumé

L'essor considérable de la technologie CMOS a permis l'accroissement de la densité d'intégration selon la loi de Moore. Cependant, la poursuite de cette évolution est en voie de ralentissement dû aux contraintes physiques et économiques. En particulier, une réduction importante des rendements de fabrication des systèmes sur puce (SoC) est observée. Le défi devient alors de pouvoir utiliser un maximum de circuits tout en tolérant des défauts physiques présents en leur sein. Les circuits reconfigurables de type FPGA (Field Programmable Gate Array) connaissent un succès croissant car leurs performances et leurs capacités d'intégrer des applications très complexes ont directement bénéficié de l'évolution technologique. Le but de cette thèse est de proposer une architecture de FPGA contenant des mécanismes permettant de tolérer plus de 20% d'éléments défectueux après fabrication. La première partie du manuscrit étudie les différentes architectures de FPGA (matricielles et arborescentes), leurs avantages et leurs inconvénients ainsi que les différentes techniques de contournement des défauts. Ces techniques peuvent être classées en deux catégories : les techniques de redondance logicielles et matérielles. Au vue de leurs performances, nous avons retenu les techniques suggérant l'ajout de redondance matérielles. De plus, ces techniques sont adaptables à notre architecture de FPGA. Dans la seconde partie de cette thèse, nous présentons l'architecture cible matricielle (matrice de grappes ou groupes). Cette architecture combine les avantages des architectures matricielles (sa généricité) et arborescentes (réduction du taux d'utilisation de l'interconnexion). Les résultats démontrent qu'avec cette architecture le taux d'utilisation de l'interconnexion est diminué d'environ 40% par rapport à une architecture matricielle classique. La troisième partie de cette thèse présente le développement d'une méthode d'identification des blocs les plus critiques contenus dans le FPGA ainsi que l'impact des différentes techniques de contournement retenues et proposées sur l'architecture et sur la criticité des blocs de base du FPGA. Pour finir, nous définissons les performances des différentes techniques de contournements en termes de tolérance aux défauts, de performances temporelles et de surface. Ces techniques permettent de contourner en moyenne 25% de défauts pour une augmentation de la surface de 30% et du délai du chemin critique de 10%. Cependant, ces techniques ont un impact significatif sur la surface du FPGA, c'est pourquoi nous proposons des techniques de redondance locale appliquant les techniques de redondance uniquement sur les blocs les plus critiques du FPGA. Ces techniques permettent de tolérer un nombre important de défauts en minimisant l'impact sur la surface et sur le délai du chemin critique (Dans le meilleur cas, 40% de défauts sont contournés pour une augmentation de la surface d'environ 20% et du délai de 10%). En conclusion, grâce à ces techniques de redondance locale, nous sommes capable de tolérer plus de 20% de défauts dans l'architecture du FPGA. De

plus, le concepteur peut fixer sa propre métrique en termes de surface, de délai du chemin critique et de tolérance aux défauts.

Abstract

The increasing integration density according to Moore's law is being slowed due to economic and physical limits. However, this technological evolution involves an higher number of physical defects after manufacturing circuit. As yield goes down, one of the future challenges is to find a way to use a maximum of fabricated circuits while tolerating physical defects spread all over the chip. Fiel Programmable Gate Array (FPGA) are integrated circuits that contain logic blocks and reconfigurable interconnect. Their ability to integrate more complex applications, their flexibility and good performance make FPGAs the perfect target architecture. The aim of this thesis is to propose an FPGA architecture containing mechanisms to tolerate more than 20% of defective resources after manufacture. The first part of the manuscript studies the different FPGA architectures (mesh and tree), their advantages and disadvantages and different defects bypass techniques. These techniques can be classified into two categories : software and hardware redundancy techniques. In view of their performance, we used hardware redundancy techniques. In addition, these techniques can be easily adapted to our FPGA architecture. In the second part of this thesis, we present the target architecture called Mesh of Clusters (MoC). This architecture combines the advantages of mesh architectures (genericity) and tree (reduction of the interconnect). The results shows that with MoC architecture the utilization of the interconnect is reduced by 40% compared to mesh architecture. The third contribution of this thesis is the development of a method to identify the most critical blocks in the FPGA and the impact of all bypass techniques on the architecture and on the criticality. Finally, we define the performance of all bypass techniques in terms of defect tolerance, timing and area overhead. The best results shows that around 25% defects can be bypassed bypassed for an area overhead of 30% and timing overhead of 10%. However, these techniques have a significant impact on the FPGA area, that's why we propose local redundancy techniques applying redundancy techniques only on the most critical blocks of the FPGA. These techniques can tolerate a large number of defects with a minimal impact on the surface and on the timing (For the best case, 40% defects can be bypassed for an area overhead of 20% and timing overhead of 10%). Finally, thanks to these local redundancy techniques, we are able to tolerate more than 20% of defect on the FPGA architecture. In addition, the designer can fix his own metric in terms of area, timing and defect tolerance.

Remerciements

En premier lieu, je tiens à remercier mon directeur de thèse, M.Habib Mehrez, pour la confiance qu'il m'a accordée en acceptant d'encadrer ce travail doctoral, pour ses multiples conseils et pour toutes les heures qu'il a consacrées à diriger cette recherche. Sa compétence, sa rigueur scientifique et sa clairvoyance m'ont beaucoup appris. Ils ont été et resteront des moteurs de mon travail de chercheur.

J'adresse de chaleureux remerciements à ma co-encadrante de thèse, Mme. Roselyne Chotin-Avot, Maitre de conférences du LIP6, pour son attention de tout instant sur mes travaux, pour ses conseils avisés et son écoute qui ont été prépondérants pour la bonne réussite de cette thèse. J'aimerais également lui dire à quel point j'ai apprécié sa grande disponibilité et son respect sans faille des délais serrés de relecture des documents que je lui ai adressés. J'ai pris un grand plaisir à travailler avec elle.

Je remercie tous ceux sans qui cette thèse ne serait pas ce qu'elle est : aussi bien par les discussions que j'ai eu la chance d'avoir avec eux, leurs suggestions ou contributions. Je pense ici en particulier à M.Vinod Pangracious, M.Billal Benamrouche, Mme Emna Amouri et M Zied Marrakchi avec qui j'ai apprécié partager et échanger. Bien sûr, atteindre ces objectifs n'aurait pas été possible sans l'aide des membres du projet Robust FPGA. Je souhaite notamment remercier Mme Lirida Alves de Barros, reponsable du projet ; Mme Arwa Ben Dhia, sa doctorante ; M Mounir Benabdendi, Maître de conférences au TIMA et M. Saif-Ur Rehman, son doctorant.

L'aboutissement de cette thèse a aussi été encouragé par de nombreuses discussions avec des collègues de disciplines variées. Je ne citerai pas de noms ici, pour ne pas en oublier certains.

Je souhaite remercier spécialement ma compagne Anne-Laure Schmitt pour son soutien quotidien indéfectible et sa patience tout au long de la thèse. Très humblement, je voudrais te dire merci pour ton soutien pendant mes périodes de doutes et pour tes multiples encouragements répétés. Notre couple a grandi en même temps que mon projet scientifique, le premier servant de socle solide à l'épanouissement du second.

Pour leurs encouragements et leur assistance aussi bien matérielle que morale qui m'ont permis de faire cette thèse dans de bonnes conditions, je remercie ma famille, et en particulier ma maman. Malgré mon éloignement depuis de (trop) nombreuses années, leur intelligence, leur confiance, leur tendresse, leur amour me portent et me guident tous les jours.

Une pensée pour terminer ces remerciements pour toi qui n'a pas vu l'aboutissement de mon travail mais je sais que tu en aurais été très fier de ton fils.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1 Contexte de recherche

Depuis les années 70, la capacité de l'industrie du semi-conducteur à suivre la loi de Moore a permis l'essor considérable de la technologie CMOS. Cet essor conduit à une amélioration des performances ainsi qu'à un accroissement de la densité d'intégration. En effet, Gordon E. Moore, cofondateur de la société Intel, démontre dès 1965 dans son article [54], que le nombre de transistors par circuit de même taille double, à prix constant, tous les ans. Il porte par la suite ce délai à 18 mois et en déduit que la puissance des ordinateurs augmente de manière significative (voir figure 1.1). Cette loi, fondée sur un constat empirique, a été vérifiée jusqu'à aujourd'hui. Cependant, il déclare que cette croissance des performances et de la densité d'intégration se heurterait à une limite physique (due à la taille des atomes) aux alentours de 2017.

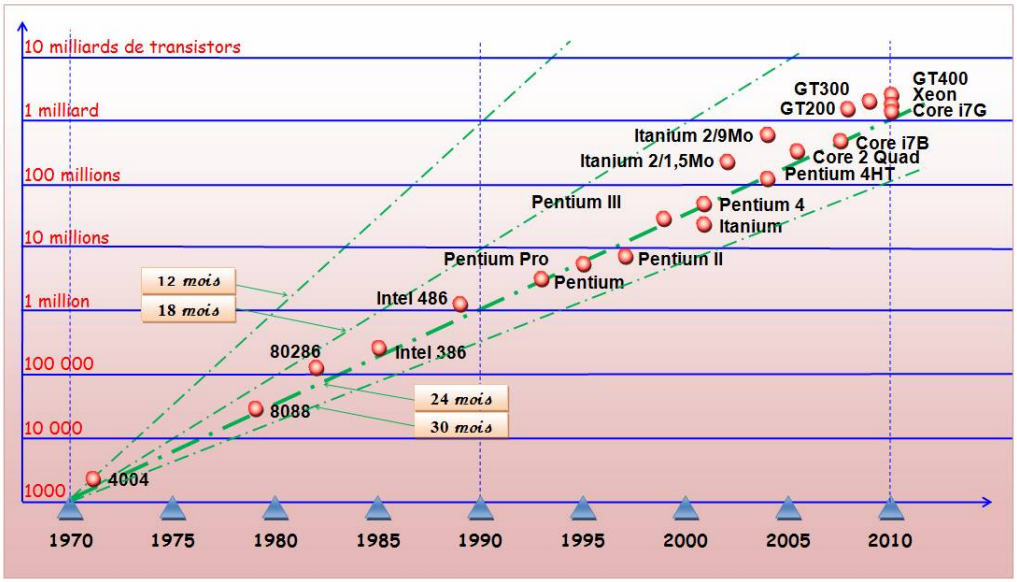


FIGURE 1.1 – loi de moore

Pour anticiper cette baisse de performances, l'International Technology Roadmap for Semiconductors (ITRS) a été créée. La première version de cette feuille de route du semi-conducteur a été publiée en 1999 par la Semiconductor Industries Associas-

tion (SIA). L'objectif de l'ITRS est de proposer la meilleure orientation possible en termes de recherche et de développement des semi-conducteurs pour les 15 années à venir. En 2013, la dernière version de l'ITRS suggère qu'à l'horizon 2025 la taille des transistors des circuits CMOS descendrait en dessous de 10nm. Cependant, la poursuite de cette évolution est en voie de ralentissement dû aux contraintes physiques et économiques. En particulier, une réduction importante des rendements de fabrication des systèmes sur puce (SoC) est observée. Elle s'accompagne de coûts de fabrication très importants. Ce changement induit un bouleversement des pratiques de conception. Les concepteurs ne doivent plus raisonner en termes de circuits seulement bons ou mauvais après test de production mais le défi devient alors de pouvoir utiliser un maximum de circuits tout en tolérant des défauts physiques présents en leur sein. La réponse à ce défi aura des répercussions sur les modèles des dispositifs, l'architecture, la sûreté de fonctionnement, la sécurité et les outils de CAO. De plus, avec l'augmentation de la densité d'intégration, le nombre de défauts de fabrication tend à augmenter [17] [64] [67] [68]. On estime à près de 20% le nombre d'éléments défectueux dans un circuit après fabrication [17] [56]. Ces défauts peuvent survenir sur les masques de silicium et engendrent une variation dans le comportement électrique des transistors et des connexions [54]. Dans les cas les plus extrêmes, ces défauts peuvent aussi engendrer un dysfonctionnement dans le comportement du circuit intégré. Un défaut représente une imperfection physique comme une connexion coupée, un défaut dans la structure d'un transistor entraînant un dysfonctionnement (collage à 0 ou à 1, *crosstalk*, etc...) permanent. Dans cette thèse, nous allons donc nous intéresser aux techniques permettant de corriger les défauts comme les collages ou les courts-circuits. Les collages conduisent au blocage d'un signal dans un état logique (0 ou 1) et les courts-circuits sont modélisés par un pont créé entre les deux broches d'un composant (par exemple un transistor) modifiant donc son comportement. Les circuits reconfigurables de type FPGA (Field Programmable Gate Array) sont des circuits intégrés qui contiennent des blocs logiques ainsi que des blocs d'interconnexions reconfigurables. Ils connaissent un succès croissant car leurs performances et leur capacité d'intégrer des applications très complexes ont directement bénéficié de l'évolution technologique. Ces circuits accroissent en permanence leur part de marché relativement aux ASIC (Application Specific Integrated Circuit). En effet, comparé aux ASICs, un FPGA offre la possibilité d'être reconfiguré à l'infini et son temps de développement ainsi que son temps de mise sur le marché sont beaucoup plus courts. L'architecture cible de cette thèse sera donc une architecture de FPGA dans laquelle des mécanismes capables de maintenir le fonctionnement du circuit malgré la présence de défauts seront inclus.

2 Cadre de la thèse

Cette thèse se déroule dans le cadre du projet Robust FPGA financé par l'ANR dont les partenaires sont : le LIP6 dont le travail est de fournir l'architecture du FPGA en y incorporant les solutions pour le contournement des défauts, le TIMA qui doit fournir des outils de Tests et diagnostic ainsi qu'une cartographie des défauts du FPGA, Telecom ParisTech qui doit fournir un outil d'analyse de la criticité des blocs composant le FPGA ainsi que des blocs robustes, et pour finir l'entreprise FlexRas qui développe les outils de configurations. Une nouvelle architecture de

FPGA nommée *Mesh of Clusters* est proposée. Cette thèse a été réalisée au sein du département "Systèmes Embarqués sur Puce" (SOC) du Laboratoire d'Informatique de Paris 6 (LIP6) et fait suite aux travaux réalisés au sein du laboratoire sur les différentes architectures des FPGAs ainsi que sur les outils de génération de circuits intégrés.

3 Objectif

Dans le contexte de la tolérance aux défauts, notre objectif est de proposer dans un premier temps une architecture de FPGA permettant l'ajout de mécanismes de contournement des défauts. Pour cela, nous nous appuierons sur les divers travaux étudiant les performances des différentes architectures de FPGAs existantes. Dans un deuxième temps, notre objectif est de proposer des mécanismes de contournement des défauts via l'utilisation de techniques de redondance logicielles et matérielles au sein du FPGA. Cette étude a été étoffée par la mise en place d'une redondance à différents niveaux de granularité. Ce qui signifie que la redondance peut aussi bien être appliquée aux blocs de base du FPGA mais aussi à l'intérieur de ceux-ci. Cependant, l'ajout de matériel dans l'ensemble du FPGA va augmenter sa surface de façon conséquente. Notre problématique se focalise donc sur la mise en place d'une redondance dans le FPGA en trois étapes :

Sélection du niveau de granularité

Notre but est de définir à quel niveau il est plus intéressant d'ajouter cette redondance. L'objectif étant de ne pas trop augmenter la surface du FPGA par rapport au gain en robustesse. Nous voulons évaluer l'impact d'une redondance à gros grain (connexions supplémentaires ajoutées entre les blocs de base du FPGA) ainsi que d'une redondance à grain fin (redondance à l'intérieur des blocs de base du FPGA) en termes de délai et de surface.

Identification des blocs critiques

Notre but est de proposer une méthode permettant d'identifier les blocs les plus critiques dans l'architecture du FPGA. Le FPGA est représenté par un graphe composé de sous-graphes et de nœuds. La criticité des blocs composant le FPGA est définit en fonction du nombre de connexions par nœud et des paramètres architecturaux. Nous pourrons établir à partir de ce calcul une hiérarchie des blocs les plus critiques présents dans le FPGA.

Redondance ciblée

Notre but est d'appliquer une redondance ciblée sur les blocs les plus critiques dans le but de diminuer l'impact des techniques de redondances sur la surface et le délai. Le choix de cette redondance ciblée sera proposé au concepteur lors de la création de l'architecture. En effet, celui-ci aura la possibilité de choisir d'optimiser la tolérance aux défauts, le délai ou la surface de son FPGA.

4 Contenu du mémoire

L'objectif principal de cette thèse est de développer une architecture de FPGA tolérante aux défauts de fabrication, nous commençons donc par présenter les différentes architectures de FPGAs existantes ainsi qu'un état de l'art des différentes techniques de contournement des défauts dans le chapitre 2. Ces techniques peuvent être classées en deux catégories :

- Des techniques de contournement dîtes logicielles lorsque le contournement est réalisé par les outils de configurations. Le principal avantage de ces techniques est qu'aucun matériel supplémentaire n'est ajouté à l'architecture de base du FPGA pour réaliser le contournement des défauts.
- Des techniques de contournement dîtes matérielles lorsque des ressources de secours sont ajoutées à l'architecture du FPGA pour réaliser le contournement des défauts.

Nous présentons ensuite une synthèse des performances de ces différentes techniques dans le but de choisir celles qui seront utilisées sur l'architecture de notre FPGA.

Nous présentons dans le chapitre 3 l'architecture de FPGA cible de cette thèse : l'architecture *Mesh of Clusters*. Les travaux de cette thèse consistent à pouvoir générer n'importe quelle architecture de FPGA *Mesh of Clusters* capable de tolérer des défauts présents en son sein. Nous détaillerons donc le flot de conception du FPGA partant du générateur d'architecture *Mesh of Clusters* jusqu'au placement/routage de diverses applications. Nous concluons ce chapitre par une exploration des performances de l'architecture en fonction du nombre d'éléments logiques, du nombre d'entrées/sorties des différents blocs composants notre FPGA par rapport aux architectures industrielles.

Nous déterminons, dans le chapitre 4, pour commencer le niveau de criticité des blocs présents dans le FPGA *Mesh of Clusters*. Ce calcul va nous permettre d'identifier les blocs les plus critiques de notre architecture. Puis, nous décrivons les modifications architecturales nécessaires pour adapter les différentes techniques de contournement des défauts retenues dans le chapitre 2 ainsi que pour les trois nouvelles techniques de contournement présentées dans cette thèse. Nous calculons aussi l'impact de ces solutions sur la criticité des différents blocs composant notre FPGA.

Nous présentons dans le chapitre 5 les performances du contournement logiciel grâce aux outils de placement/routage, puis les performances des diverses solutions de contournement retenues. Notre but étant de fournir un FPGA tolérant aux défauts de fabrication, grâce à l'étude de criticité effectuée au chapitre 4, nous proposons et évaluons les performances des techniques de redondance ciblées sur les éléments les plus critiques de notre FPGA.

Nous terminons par une conclusion et des perspectives.

Chapitre 2

Etat de l'art

1 Introduction

Ce chapitre est consacré à la présentation des diverses architectures de FPGA ainsi qu'aux différentes techniques de contournement des défauts. Nous commençons par détailler les spécificités des FPGA puis nous définirons les avantages et les inconvénients de chacune des architectures présentées. Enfin, nous présenterons les différentes techniques de contournement des défauts pouvant être classées en deux catégories : les techniques de redondances logicielles et matérielles.

2 Architectures des FPGAs

Les Field Programmable Gate Arrays (FPGAs) ont été développés depuis 1985. Grâce à l'évolution technologique ces circuits n'ont cessé de se développer et de nombreux fabricants se sont depuis lancés dans la fabrication de FPGAs (Xilinx, Altera, Actel...). Profitant de l'évolution technologique, un FPGA peut maintenant contenir plusieurs millions de portes logiques [51]. Ces circuits sont composés d'éléments logiques interconnectés entre eux par des ressources de routage. La particularité d'un FPGA vient de sa reconfigurabilité. Les éléments logiques ainsi que le réseau d'interconnexion peuvent être reconfigurés dans le but de supporter une autre application. En effet, les FPGAs sont composés de blocs mémoire qui permettent de configurer et de figer la fonctionnalité du circuit. Il existe plusieurs types de FPGA (les *SRAM-based FPGA*, les *Flash-based FPGA* et les *Antifuse-based FPGAs*), classés en fonction de la manière dont ils sont programmés. La majorité des FPGAs utilisent des mémoires statiques (Fig 2.1) et sont appelés *SRAM-based* FPGA.

La configuration du FPGA est chargée dans une mémoire statique. La cellule a besoin d'être activée pour pouvoir charger la configuration. Ce type de mémoire peut être configuré via une source externe comme une mémoire flash ou via un processeur. Les FPGAs du commerce fabriqués par Xilinx et Altera utilisent ce type de configuration. Les Flash-based FPGA quant à eux ne sont pas équipés de mémoire statique et gardent leur configuration même quand ils ne sont pas alimentés. Le principal avantage de cette technologie est qu'elle consomme moins et est aussi tolérante aux radiations, c'est pourquoi elle est principalement utilisée dans le domaine médical. Enfin, les Antifuse-based FPGAs sont composés d'une série de fusibles et sont principalement utilisés dans le domaine spatial. Ces fusibles sont brûlés lors de

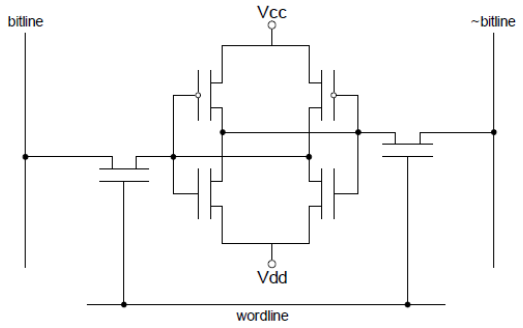


FIGURE 2.1 – Cellule SRAM

la configuration du FPGA. Il n'est donc pas possible de le reprogrammer. Cependant, tout comme les Flash-based FPGA ils consomment moins que ceux équipés d'une mémoire statique (SRAM-based). Pour finir, les SRAM-based FPGA sont les FPGA les plus répandus, c'est pourquoi nous avons choisi de développer un FPGA composé de ce type de mémoire.

2.1 Les différentes architectures des FPGAs

Un FPGA est composé d'une matrice d'éléments logiques appelés Configurable Logic Block (CLB) connectés entre eux par des ressources d'interconnexions (Fig 2.2).

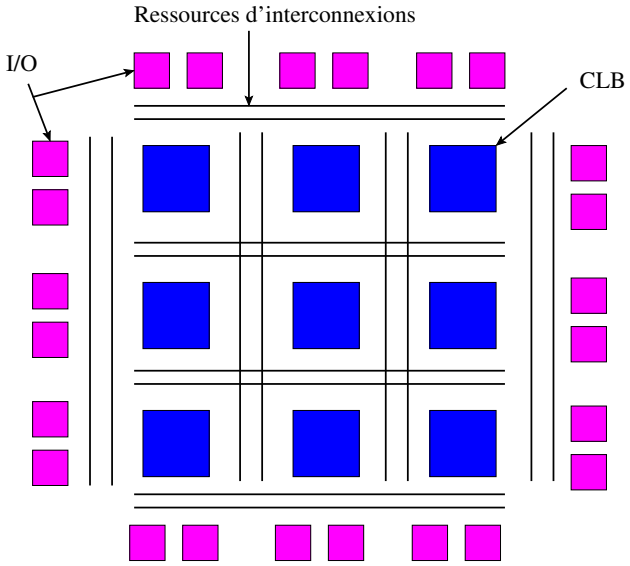


FIGURE 2.2 – Architecture d'un FPGA

Chaque CLB est composé de BLEs (Basic Logic Element) utilisés pour implémenter la partie logique du circuit. Un CLB est donc caractérisé par le nombre de ses entrées I et par le nombre N de BLE qu'il contient. Dans les FPGAs modernes ce nombre de BLE peut varier de 3 à 12 et chaque BLE peut être connecté à n'importe quelle entrée I du CLB ou à un autre BLE comme indiqué par la figure 2.3.a). Un BLE est composé d'un ensemble de tables de transcodage (*Look Up Table* : (LUT) en anglais) ainsi que d'une bascule D pour implémenter les fonctions de bases grâce

au bloc mémoire (SRAM), suivie d'un multiplexeur. Un BLE est composé de k Look Up Table ce qui permet d'implémenter une fonction logique à k entrées et une sortie et nécessite 2^k blocs mémoire pour la configuration. La bascule D quant à elle permet de choisir entre un fonctionnement logique ou séquentiel. Ensemble les LUT et la bascule D forme un Basic Logic Element (BLE illustré figure 2.3.b)). Des études ont montré que la meilleure architecture de LUT en termes de performances est une LUT ayant 4 entrées [60] [2].

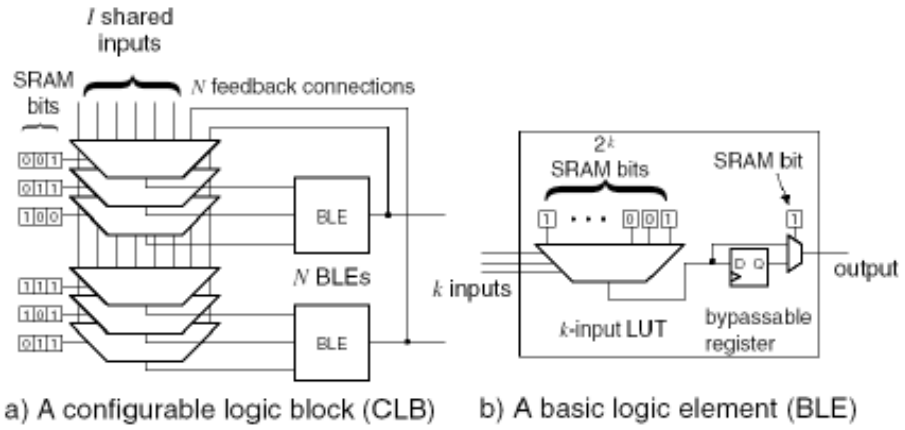


FIGURE 2.3 – Architecture d'un CLB/BLE

Les éléments logiques sont connectés entre eux et aux entrées/sorties grâce au réseau d'interconnexion. Ce réseau est programmé de la même manière que les éléments logiques avec des RAMs statiques. L'ensemble permet de configurer complètement le FPGA. En théorie, il est donc possible de charger n'importe quelle fonction logique sur un FPGA. Enfin, la reprogrammation de l'ensemble des SRAM permet de reconfigurer entièrement le circuit et donc de changer l'application ou la fonction du FPGA. Pour finir, lors de la configuration du FPGA, la *netlist* contenant l'ensemble des connexions est envoyée au FPGA. Le placement des blocs logiques et la manière dont ils sont interconnectés est alors défini par un algorithme de placement et de routage.

FPGA matriciel (Mesh)

L'architecture la plus répandue et utilisée dans le commerce est l'architecture matricielle (Mesh) [10] [11] [18] [69] [34] [59] appelée aussi architecture de type îlot (Island style). Elle est composée d'une matrice de tuiles en deux dimensions, chaque tuile étant composée d'éléments logiques (CLB) et de ressources d'interconnexions avec des blocs de Connexions (C) et des Switch boxes (S comme indiqué sur la figure 2.4). Les blocs de connexions C sont utilisés pour connecter les éléments logiques au réseau d'interconnexion tandis que les Switchs boxes permettent de connecter l'ensemble des blocs de connexions C.

Ce réseau d'interconnexion en 2 dimensions organisé en lignes et en colonnes est distribué tout autour des éléments logiques et permet de connecter n'importe quels éléments logiques entre eux ou aux entrées/sorties du FPGA. Le nombre de connexions (CW) entre les Switch boxes et les blocs de connexions représente le canal de routage (CR) du FPGA. Ces blocs d'interconnexions diffèrent par leurs topologies

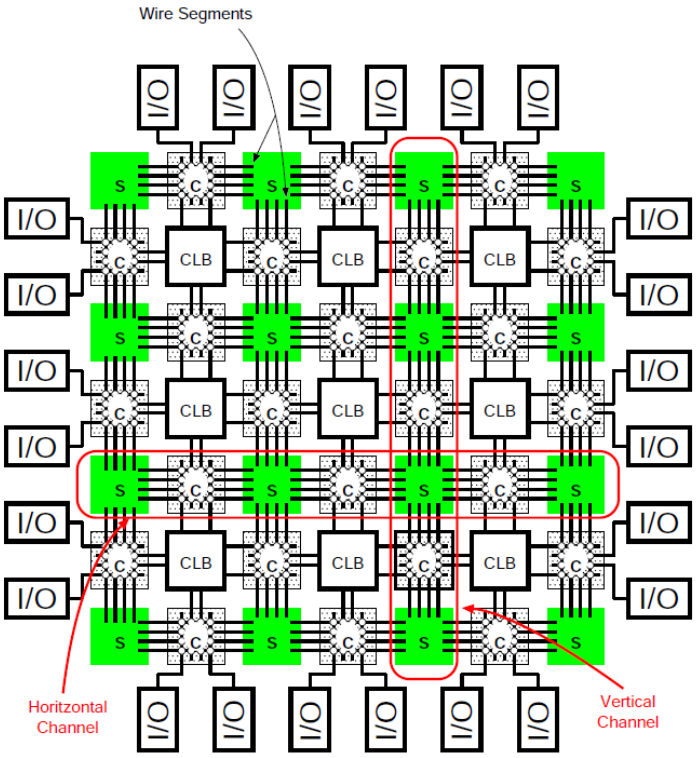


FIGURE 2.4 – Mesh-based FPGA

et leur flexibilité. Pour les blocs de connexions (C), la flexibilité est définie par le nombre de fils de routage du bloc d'interconnexions auxquels les entrées/sorties d'un bloc logique peuvent être connectées. Elle peut donc être définie par deux paramètres F_{cin} et F_{cout} (F_{cin} représente le nombre de connexions entre le bloc d'interconnexions et les entrées du bloc logique et F_{cout} entre le bloc d'interconnexions et les sorties du bloc logique). Par exemple, dans la figure 2.5.a), chaque CLB peut être connecté à deux fils du bloc de Connexions (C), la flexibilité sera donc $F_{cin} = F_{cout} = 2$. Les Switchs boxes utilisées pour interconnecter les blocs de Connexions au canal de routage (CR) ont plusieurs topologies. En effet, il existe des Switchs boxes avec une topologie appelée disjoint [32], ou wilton [70] et pour finir universel [20]. La topologie va définir le nombre de connexions entre le canal de routage et chaque entrée d'un même bloc de Connexions (C). Ce nombre permet de définir la flexibilité des Switch boxes F_s. Par exemple, dans la figure 2.5.b), chaque segment en pointillés représente une connexion possible. Il y a 3 connexions possible, La flexibilité est donc $F_s = 3$. Le canal de routage est formé de fils de routage permettant de connecter les blocs Connexions et les Switch boxes. Le nombre de fils du canal de routage CW est appelé largeur (ou taille) du canal. Cette taille est la même pour toute l'architecture du FPGA. La taille du canal de routage doit être fixée avant la fabrication du FPGA par le concepteur car c'est l'un des paramètres architecturaux les plus importants dans l'architecture d'un FPGA. En effet, le canal de routage doit permettre de router l'ensemble des signaux de l'application utilisée par le FPGA sinon on dit que l'application est non routable. Un canal de routage contient des fils de différentes longueurs. Cette longueur dépend du nombre de blocs logiques longeant le fil en question. La figure 2.6 montre un canal de routage avec des longueurs de fils allant

de 1 à 4. Pour une longueur de fil supérieure à 1, le nombre de commutateurs programmables (switchs) traversés est réduit cela permet donc de réduire la surface ainsi que le délai de routage. Cependant, cette solution diminue la flexibilité du FPGA donc augmente la difficulté de routage de l'application. Dans l'industrie, les FPGAs sont souvent composés de segments de routage de différentes longueurs.

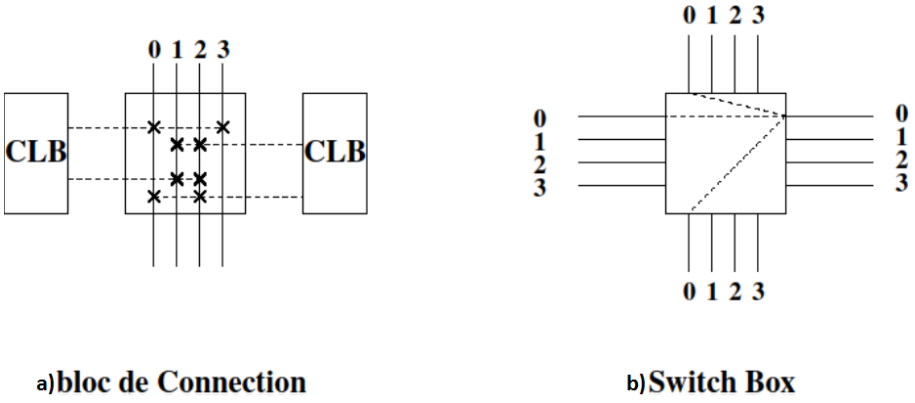


FIGURE 2.5 – Exemple d'architecture d'un bloc de Connexions et d'une Switch box

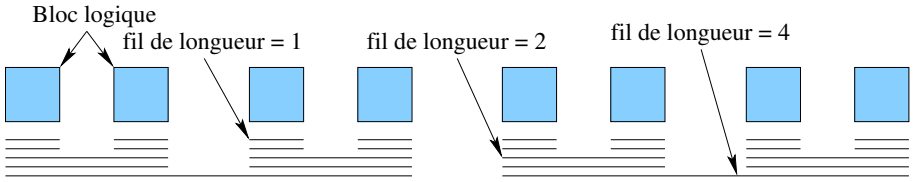


FIGURE 2.6 – Exemple d'un canal de routage

En résumé les avantages et les inconvénient de cette architecture sont :
- **Avantage :** La régularité et la flexibilité de cette architecture.
- **Inconvénient :** Le réseau d'interconnexion utilise 90% de la surface totale, est responsable de 80% du délai de fonctionnement total et de plus de 85% de la consommation globale du FPGA [2] [47] [25].

FPGA arborescent (Tree)

Pour réduire l'influence du réseau d'interconnexion et se rapprocher des performances en terme de surface et de vitesse des ASICs, une deuxième architecture de FPGA a été développée appelée Tree-based FPGA [1] [19] [43] [26] [50] [49]. En effet dans les mesh, 90% de la surface est occupée par le réseau d'interconnexions et le nombre important de commutateurs traversés augmente considérablement le délai de propagation. Une réduction du nombre de commutateurs dans l'architecture devrait donc permettre d'améliorer les performances du FPGA. Dans l'architecture arborescente (Tree), les blocs logiques sont regroupés sous forme de clusters et chaque cluster est composé d'un réseau local d'interconnexion sous forme d'une Switch box (illustré figure 2.7). Une Switch box est composé d'un réseau descendant pour connecter les entrées du cluster aux blocs logiques appelé Downward Mini Switch box (DMSB) ainsi que d'un réseau montant pour connecter les éléments logiques aux sorties du cluster appelé Upward Mini Switch box (UMSB). Pour connecter les UMSB/DMSB

aux blocs logiques une interconnexion de type Butterfly-Fat-Tree (BFT) [46] est utilisée. La particularité de cette architecture est que chaque UMSB/DMSB de niveau N est connecté à 4 fois plus d'UMSB/DMSB de niveau $N+1$. Par exemple, dans la figure 2.7, un bloc logique configurable (ici LB) est connecté à un UMSB de même niveau (ici niveau 1). Cet UMSB sera donc connecté à quatre fois plus de UMSB ($4 \times 1 = 4UMSB$) de niveau $N+1 = 2$, 16 UMSB de niveau 3... Enfin les entrées/sorties sont regroupées dans un cluster spécifique et sont directement connectées aux UMSB/DMSB comme indiqué par la figure 2.7. Cette configuration permet donc de connecter ces plots à n'importe quel élément logique de l'architecture. Grâce à ce type d'architecture, une connexion entre deux blocs logiques nécessite moins de commutateurs comparée à l'architecture matricielle (mesh) car le nombre de commutateurs traversés augmente de façon logarithmique [44](contrairement à l'architecture matricielle où le nombre de commutateurs augmente linéairement en fonction de la distance de Manhattan entre deux blocs logiques). Xilinx et Altera commercialisent des FPGA avec des architecture hiérarchiques tel que Xilinx Virtex-II [71] et Altera Apex [39]. Au LIP6, une architecture arborescente multi-niveaux appelée Multi-level FPGA (MFPGA) [73] a été étudiée et montre un gain en surface de 40% comparé à une architecture matricielle. En résumé les avantages et les inconvénients de ce type d'architecture sont :

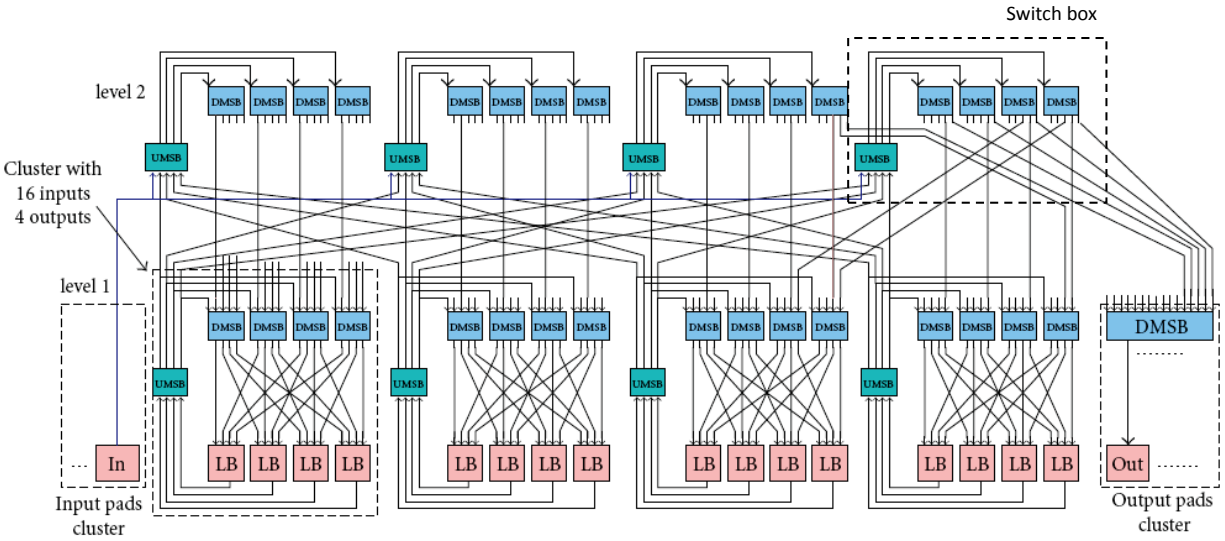


FIGURE 2.7 – Tree-based FPGA

- **Avantage :** Réduction de l'impact du réseau d'interconnexion par rapport aux éléments logique (56% de switchs utilisés en moins comparé à une architecture de type mesh).
- **Inconvénient :** La taille du chemin critique est augmentée. De plus, plus le nombre de niveaux dans l'architecture est important, plus il est difficile de générer le layout correspondant car l'architecture souffre de problèmes de scalabilité.

2.2 Conclusion

En conclusion, nous avons présenté dans cette section les spécificités des architectures FPGA, puis différentes architectures de FPGA. On constate que ces architectures diffèrent dans la manière dont sont interconnectés les blocs logiques au réseau d'interconnexion. Chacune de ces architectures présentent des avantages et des inconvénients et ont servit de base pour l'architecture utilisée dans cette thèse : l'architecture de FPGA appelé *Mesh of Clusters*.

3 Contournement des défauts

Nous nous intéressons dans cette section aux différentes techniques de contournement des défauts dans les FPGAs aussi bien appliquées sur les éléments logiques que sur les blocs d'interconnexions.Ces techniques peuvent être classifiées en 2 catégories :

- Approches logicielles : elles utilisent les outils de configuration pour éviter les ressources défectueuses à partir de leur localisation dans le FPGA (cartographie des défauts). Ces méthodes peuvent prendre du temps pour placer et router une application car elles sont dépendantes des outils de configuration mais ne nécessitent pas de matériel supplémentaire.
- Approches matérielles : elles requièrent l'ajout de matériel supplémentaire comme ressources de contournements. Ces ressources de contournements facilitent la correction des défauts puisqu'elles vont tout simplement remplacer l'élément défectueux. Ces méthodes sont donc rapides à mettre en place mais nécessitent une augmentation de la taille du FPGA.

Chaque approche a ses avantages en termes de délai (délai du chemin critique, temps de traitement...) et de ressources (surface de silicium, stockage externe...). Chaque approche sera présentée avec ses avantages et ses inconvénients.

3.1 Approches logicielles

Introduction

Dans le cas des approches logicielles, les outils de placement/routage sont utilisés pour configurer le FPGA sans utiliser les ressources défectueuses grâce à une cartographie qui permet de localiser les défauts. Ces solutions logicielles sont très utilisées car elles permettent de ne pas ajouter de matériel. Cependant, le placement/routage d'une application en évitant les défauts peut prendre beaucoup de temps car il nécessite souvent plusieurs phases de routage pour trouver une solution viable. De plus, l'efficacité de ces méthodes est dépendante de la vitesse du placement/routage des outils de configuration. En général, il faut effectuer plusieurs phases de routage jusqu'à ce que tous les défauts soient évités mais cette solution est impraticable dans l'industrie pour deux raisons. Premièrement, générer une solution de placement/routage évitant les défauts pour chaque FPGA revient trop cher en

termes de temps de configuration car cela peut nécessiter un grand nombre d'itérations. Ensuite, il est impossible de vérifier que chaque solution respecte le cahier des charges (temps caractéristiques, délai du chemin critique) car ils sont différents pour chaque nouveau placement/routage. Pour les circuits les plus complexes, il est parfois impossible de trouver une solution de placement/routage viable. Malgré ces inconvénients, l'approche logicielle est une solution intéressante car elle permet de ne pas ajouter de matériel supplémentaire.

Translation de l'application

Cette solution proposée par Vijay Lakamraju et Russell Tessier [45] propose d'éviter les défauts de deux manières différentes en fonction du matériel visé. Pour un défaut dans les éléments logiques, les ressources défectueuses sont échangées avec des ressources non utilisées.

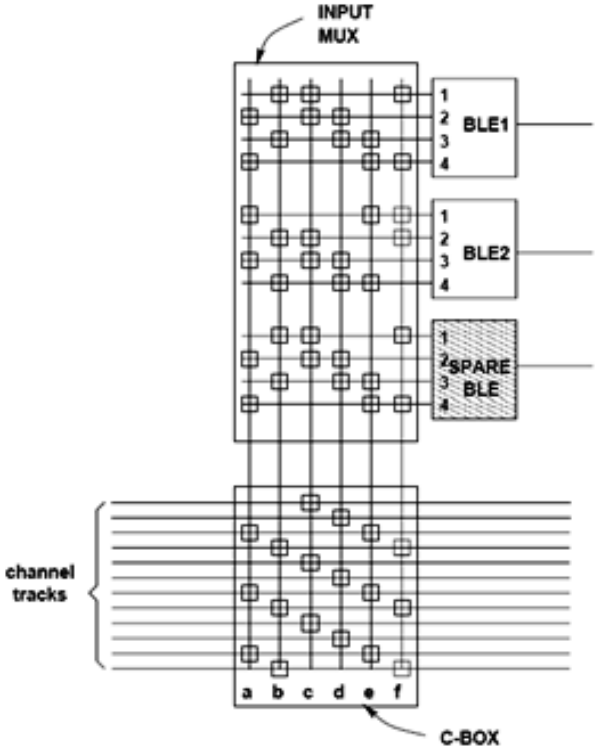


FIGURE 2.8 – Schéma de principe

Cette méthode se base sur le fait que tous les BLEs ne sont pas utilisés dans un FPGA, et que ces BLEs peuvent être permutés avec d'autres défectueux. Dans la figure 2.8 ci-dessus, on constate que le BLE 1 et le BLE de secours (*spare BLE* qui n'est pas utilisé donc pas configuré) ont les mêmes connexions en entrée. On peut donc facilement utiliser le BLE non utilisé (*spare BLE*) dans le cas où le BLE 1 présenterait des défauts. Dans le cas où le défaut interviendrait sur une connexion, un algorithme pour re-router le signal affecté par le défaut est utilisé. Pour les applications les plus complexes, trouver une nouvelle solution de routage peut prendre beaucoup de temps et même être impossible. On constate d'après la figure 2.9 que

l'utilisation de l'algorithme de routage (incremental) est avantageux pour un nombre de défauts compris en moyenne entre 1 et 500 dans les interconnexions avec un FPGA contenant en moyenne 6000 BLEs, car le temps pour re-router l'application est moins important qu'avec des outils utilisant un placement/routage classique (from scratch). Cependant au-delà d'un certain nombre de défauts, cette méthode devient très contraignante en termes de temps de routage. C'est aussi le cas pour les défauts dans les blocs logiques. Pour les circuits les plus denses, il est parfois impossible de trouver les ressources non utilisées nécessaires pour le contournement des défauts. Malgré ces désavantages, cette solution reste attractive car elle permet de ne pas ajouter de matériel et qu'elle permet de tolérer un nombre important de défauts (voir figure 2.9).

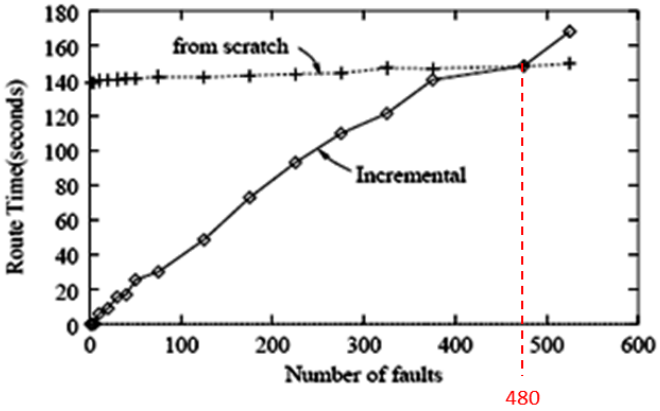


FIGURE 2.9 – Résultats de simulation [45]

Différents *Bitstream*

Une autre méthode pour éviter les défauts consiste à réserver des ressources dites de secours (à utiliser pour remplacer les ressources défectueuses) lors de la configuration. Le nombre de ressources de secours dépendra de la taille de l'application ainsi que de la taille du FPGA. Il est possible d'éviter l'utilisation de certaines ressources en effectuant une translation de l'application d'une ligne ou d'une colonne [29]. La translation d'un schéma structurel peut être effectuée en un minimum de temps en modifiant directement le *bitstream*. La figure 2.10 montre un exemple de translation de *bitstream* dans le cas où l'on rencontrerait un CLB défectueux. En effet, si le CLB central est défectueux et son plus proche voisin non-utilisé, il suffit alors de faire une translation de l'application pour que celle-ci utilise le CLB voisin non-utilisé. Cette solution permet d'effectuer une translation sur un bloc logique (figure 2.10) ou sur une partie complète du circuit (figure 2.11). En effet, la modification du bitstream permet, dans le cas où un défaut serait présent sur une ligne ou une colonne du FPGA, de faire une translation de l'application complète pour utiliser une ligne ou une colonne de secours. Cependant, pour pouvoir corriger plusieurs défauts avec une seule ligne de secours, il faudrait que ceux-ci soient parfaitement alignés(voir figure 2.11).

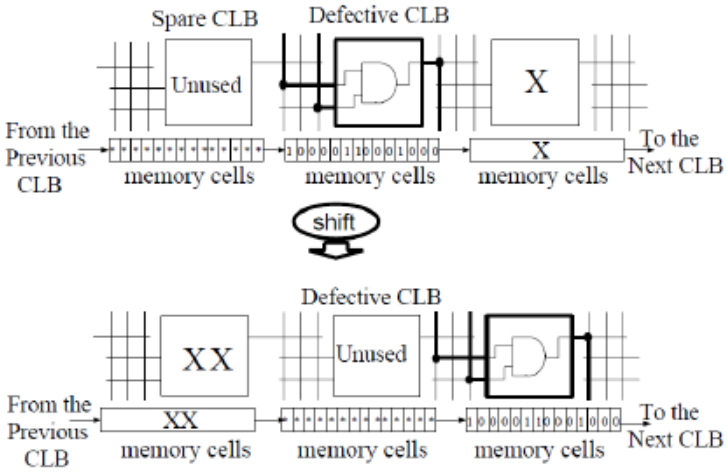


FIGURE 2.10 – Translation d'un CLB [29]

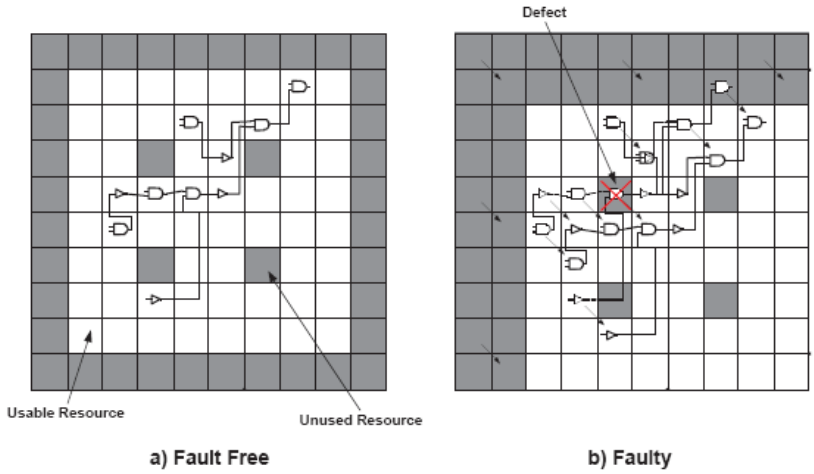


FIGURE 2.11 – Translation d'une partie de l'architecture

EasyPath de XILINX

Xilinx, un des principaux vendeurs de FPGA, propose sa solution pour éviter les ressources défectueuses *EasyPath* [72]. Les clients fournissent leur *bitstream*, en retour Xilinx fournit un lot de FPGAs fonctionnels pour ce *bitstream*. Xilinx garantit le fonctionnement de l'application en contournant eux-même les ressources défectueuses du FPGA lors du routage à partir de leur localisation dans le FPGA. Cette solution permet aux clients d'obtenir un FPGA 30 à 70% moins cher qu'un FPGA classique (série Virtex de Xilinx), et de réduire le temps de développement du circuit puisqu'il suffit de fournir le *bitstream* de l'application. En effet Xilinx se charge de tester ce *bitstream* sur un lot de FPGA contenant des défauts pour trouver les FPGAs fonctionnels. Le principal désavantage de cette solution, c'est qu'elle nécessite que l'application soit mature. En effet, Xilinx ne garantit pas qu'en cas de changement dans le *bitstream*, donc dans le placement/routage, le circuit soit toujours opérationnel.

Conclusion

Le tableau ci-dessous présente les avantages et les inconvénients de chacune des méthodes logicielles pour éviter les défauts. On constate que d'une manière générale, les méthodes logicielles permettent de gagner en temps de développement et en coût en améliorant le rendement du FPGA. Cependant, le nombre de défauts que l'on peut corriger reste très limité surtout pour les applications les plus complexes, si l'on veut tirer parti de ces avantages.

	Translation de l'application	Différents bitstream	EasyPath
Avantages	Augmente le rendement du FPGA	Translation simple et rapide	Réduction du coût et du temps de développement du FPGA
Inconvénients	Peut s'avérer inefficace pour les applications les plus complexes	Avec une seule ligne de secours tous les défauts doivent être alignés pour être corrigés	Application mature

TABLE 2.1 – Comparaison des solutions logicielles

3.2 Approches matérielles

Introduction

La redondance matérielle consiste en l'ajout de matériel supplémentaire (BLE et blocs d'interconnexions) en tant que ressources de secours. Ces ressources de secours facilitent la correction des défauts en remplaçant les éléments défectueux par ceux non utilisés. Ces modifications réduisent le temps de correction par rapport au temps que prendrait un nouveau placement et un nouveau routage de l'approche logicielle. Le désavantage de ces approches est l'incorporation d'une redondance à l'intérieur des éléments logiques et des blocs d'interconnexions du FPGA. La redondance matérielle inclut une augmentation de la surface du FPGA et peut au final corriger moins de défauts que les approches logicielles. Cependant, la redondance est déjà utilisée dans l'industrie [21, 22, 40]. Cela suggère que les désavantages en termes de coût et de surface sont négligeables par rapport au gain en termes de fonctionnement.

Redondance à gros grain [4]

Dans le schéma traditionnel Coarse Grain Redundancy (CGR), une ligne et une colonne sont utilisées pour corriger les défauts. Cette méthode peut tolérer plusieurs éléments logiques défectueux dans la même ligne/colonne. Cependant, la distribution de ces ressources additionnelles influe sur la capacité du circuit à tolérer un nombre important de défauts. Nous allons étudier deux méthodes ajoutant des ressources de secours.

Architecture

Le schéma traditionnel du CGR ajoute une ligne/colonne au FPGA qui sera utilisée en remplacement de la ligne/colonne défectueuse. Cette architecture ne peut corriger que les défauts présent dans une ligne ou dans une colonne comme présenté sur la figure 2.12. Pour pouvoir tolérer plusieurs défauts, ils doivent se trouver dans la même ligne/colonne. De plus, même en augmentant la taille du FPGA, on ne pourra corriger que des défauts se trouvant dans la même ligne/colonne. Pour augmenter le rendement de cette solution, on peut envisager d'augmenter le nombre de lignes/colonnes de secours. La CGR nécessite une modification des blocs d'interconnexions du FPGA. Pour contourner les défauts, les connexions entre les différents blocs doivent être allongées (voir figure 2.13). Dans le cas d'un FPGA sans défauts ces extensions ne seront pas utilisées. Si un défaut est constaté dans le FPGA, ces extensions permettent de ramener les signaux présents dans la ressource défectueuse vers les ressources de secours. En plus des connexions de base, de nouvelles connexions sont ajoutées à l'entrée et à la sortie des éléments logiques. L'ajout de ces connexions restera invisible pour l'utilisateur dans les logiciels de placement/routage.

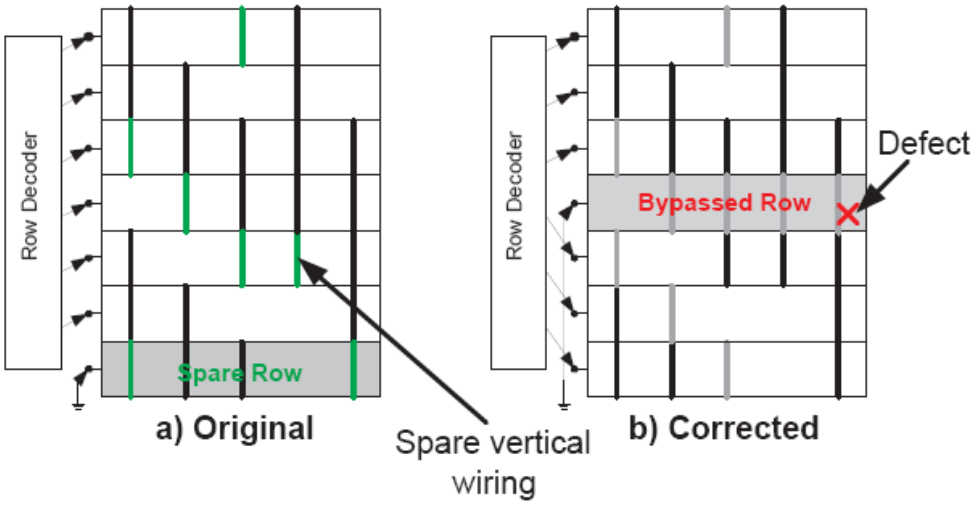


FIGURE 2.12 – Principe du CGR

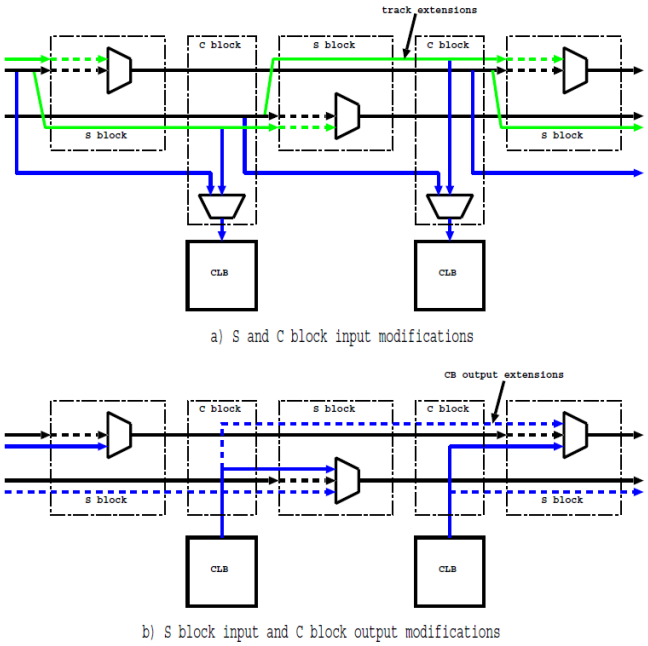


FIGURE 2.13 – Modification des blocs de connexions

Ressources de secours multiples

Pour pouvoir corriger un nombre plus important de défauts, de multiples ressources de secours peuvent être ajoutées. La 1ère méthode consiste à ajouter des lignes ou des colonnes supplémentaires au FPGA comme la CGR traditionnel (voir figure 2.14).

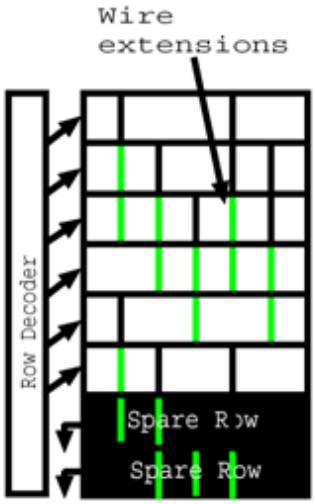


FIGURE 2.14 – FPGA avec plusieurs ressources de secours

Ces éléments de secours peuvent être utilisés pour corriger des défauts n'importe où dans le FPGA. Plus il y aura de ressources de secours et plus le FPGA pourra tolérer de défauts comme le montre le graphique figure 2.15. La courbe *baseline* représente la tolérance aux défauts d'un FPGA classique sans redondance. Les courbes *2 Global* et *4 Global* indiquent la tolérance aux défauts du FPGA lorsqu'on lui ajoute respectivement 2 puis 4 lignes ou colonnes de secours. On constate qu'on augmente donc le rendement du FPGA. L'extension des connexions sera dépendante du nombre d'éléments de secours. Pour 2 éléments, on devra augmenter les longueurs des extensions d'une longueur 2, pour 3 éléments, on utilisera des extensions d'une longueur 3 etc... Avec cette méthode on va donc augmenter la taille des extensions de connexion et donc augmenter la taille des multiplexeurs présents dans les blocs de connexions.

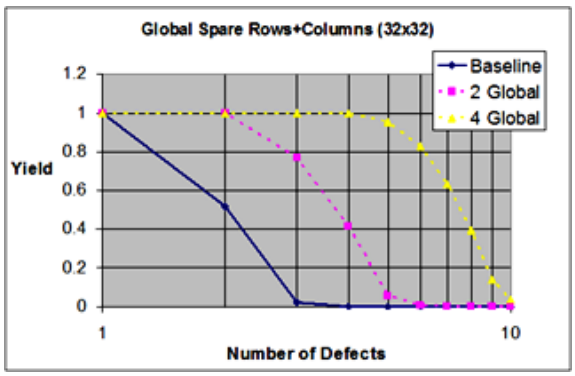


FIGURE 2.15 – Rendement de la CGR

La seconde méthode consiste à diviser le FPGA en sous-ensembles. Chaque sous-ensemble va contenir sa propre ligne ou colonne de secours(voir figure 2.16). La correction des défauts sera donc locale dans chaque sous-division. Le fait de diviser le FPGA de cette façon va permettre de réduire la taille des extensions de connexion et de répartir les éléments de secours de manière uniforme.

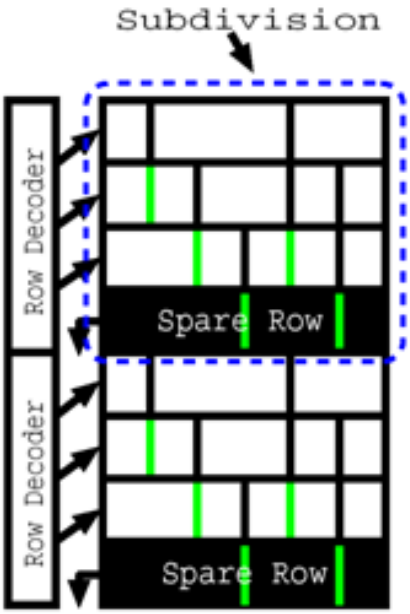


FIGURE 2.16 – Division du FPGA en sous-ensembles

Le principal inconvénient de cette méthode réside dans le fait que chaque élément de secours ne peut corriger que des défauts contenus dans sa sous-division.

La CGR traite tous les défauts de la même manière aussi bien dans les éléments logiques que dans les blocs d'interconnexion. Une ligne ou une colonne contenant un ou plusieurs éléments défectueux est toujours remplacée par une ligne ou une colonne de secours. Cela simplifie le processus de correction mais est dépendant du nombre de ressources disponibles. Le principal inconvénient de cette méthode est l'augmentation de la surface due à l'ajout de matériel. Chacune des versions du CGR conduit à la modification des blocs de connexions donc à l'augmentation de la surface du FPGA. Pour finir, certains défauts ne peuvent être corrigés. En effet, les défauts sur les extensions de connexions reliants deux lignes ou deux colonnes ne peuvent être traités avec la CGR.

Apex d'ALTERA

Altera, concurrent de Xilinx et un des principaux vendeurs de FPGA, utilise la CGR dans une gamme de ses FPGA : Apex redundancy. Des lignes et colonnes supplémentaires ont été ajoutées à l'architecture du FPGA. Si un défaut est trouvé sur une ligne ou une colonne du FPGA, une ressource de secours sera configurée et utilisée pour la remplacer sans affecter la fonctionnalité ou le temps de propagation du FPGA (voir figure 2.17). Altera est à notre connaissance la seule compagnie à proposer ce genre de redondance matérielle dans ses circuits.

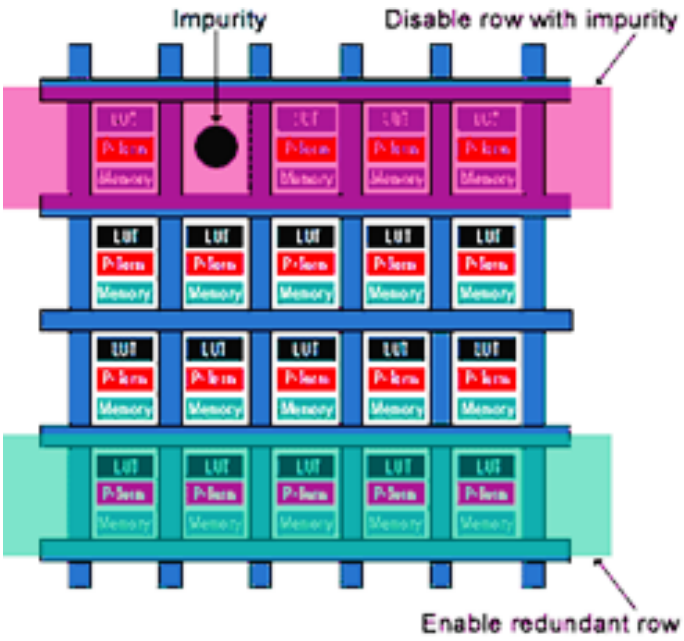


FIGURE 2.17 – FPGA avec redondance [23]

Redondance à grain fin [4]

Conservant le même principe que la méthode Coarse Graine Redundancy, la méthode Fine Graine Redundancy (FGR) consiste à ajouter des ressources matérielles au FPGA. Cependant, cette méthode utilise une approche différente puisqu'au lieu d'ajouter des ressources sous forme de lignes et de colonnes supplémentaires dans le FPGA, elle va descendre dans l'architecture des blocs de bases, notamment dans les blocs d'interconnexions (Swith Boxes). C'est pourquoi une modification de la structure des Switch boxes a été proposée.

Architecture des blocs d'interconnexions

Pour rendre les blocs d'interconnexions tolérants aux défauts, le principe consiste à ajouter des connexions de secours aux entrées et aux sorties des Switch boxes(voir figure 2.18).

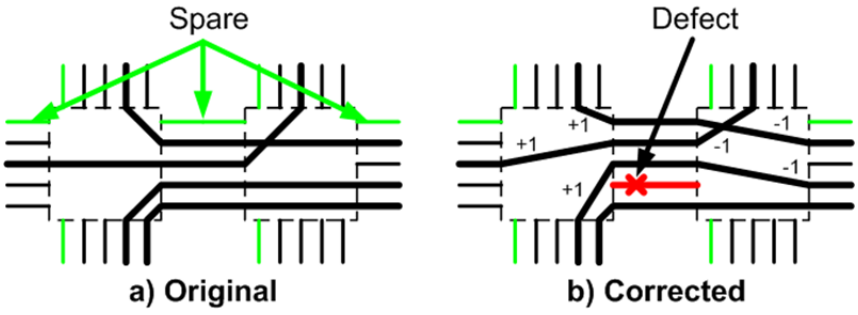


FIGURE 2.18 – Switch boxes avec des connexions de secours

Dans le cas où un défaut serait présent sur une entrée ou une sortie du bloc

d'interconnexion, les entrées/sorties de secours vont permettre de contourner la ressource défectueuse. Si on s'intéresse de plus près à l'architecture interne d'une Switch box, on constate que l'ajout de deux étages de multiplexeurs en entrée et en sortie est nécessaire (voir figure 2.19).

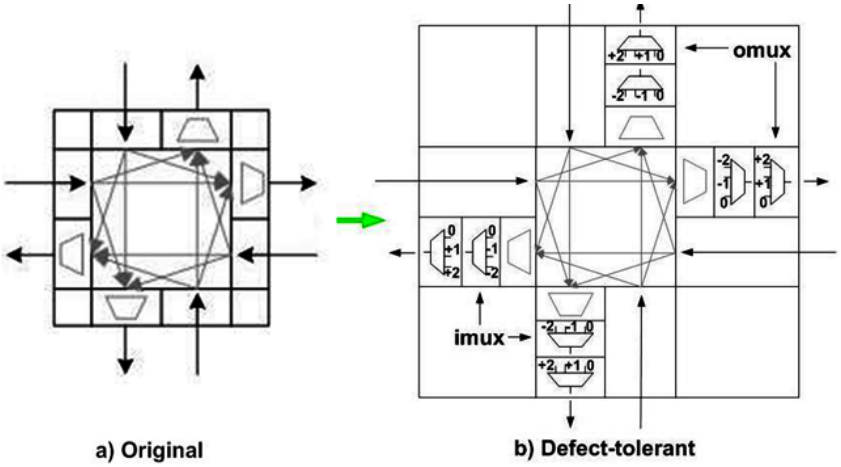


FIGURE 2.19 – Architecture d'une Switch box avec la Fine Grain Redundancy

Si un défaut est constaté (après le test et diagnostic du FPGA) à l'intérieur de l'élément, les 2 étages de multiplexeurs en entrée vont permettre de contourner ce défaut et les 2 étages en sortie serviront à restaurer le signal sur la sortie qui lui est attribuée (voir exemple figure 2.20). Dans cet exemple, on constate que les 2 étages de multiplexeurs en entrée vont servir à contourner l'élément défectueux pour utiliser son plus proche voisin. Puis les 2 étages en sortie vont aiguiller le signal sur la bonne sortie. Au final, l'aspect externe de la Switch box est identique puisque l'on garde le même nombre d'entrées et de sorties. Cependant, l'ajout de multiplexeurs va permettre de tolérer un certain nombre de défauts dans chacune des Switch boxes.

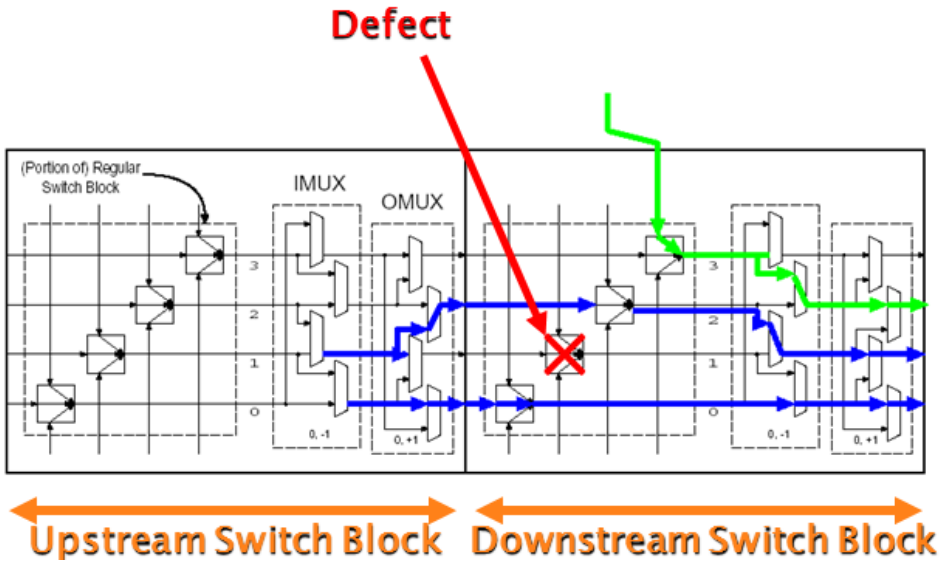


FIGURE 2.20 – Contournement d'un élément défectueux

Comparaison entre la CGR et la FGR

Dans le cas de la FGR, plus on va augmenter la taille du FPGA, plus on va ajouter de blocs robustes donc on va augmenter la tolérance aux défauts du FPGA. Contrairement à la CGR, où la tolérance aux défauts va directement dépendre du nombre de lignes et colonnes de secours présentes à l'intérieur du FPGA quel que soit sa taille (voir figure 2.21).

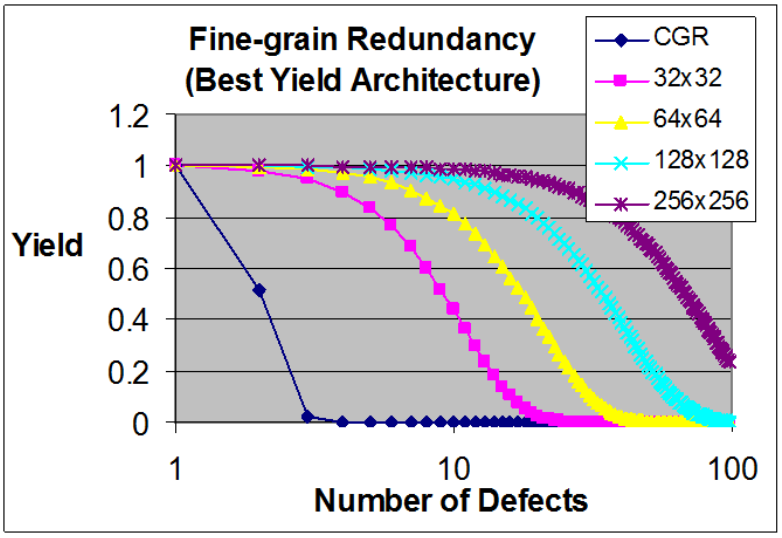


FIGURE 2.21 – Rendement de la CGR et de la FGR

La courbe CGR représente le rendement de la CGR en fonction du nombre de défauts présents dans le FPGA. On constate qu'avec la CGR classique (une seule ressource de secours), un seul défaut est toléré. Si plusieurs défauts sont présents, la fonctionnalité du circuit n'est plus assurée à 100%. Les autres courbes représentent le rendement de la FGR pour diverses tailles de FPGA. On constate que plus la taille du FPGA augmente, plus le nombre de défauts tolérés est important dû au fait que le nombre d'éléments robustes augmente de la même manière que la taille du circuit.

Amélioration de la FGR

Le principe de la FGR est applicable uniquement si toutes les ressources présentes dans les blocs d'interconnexions ne sont pas toutes utilisées. En effet, si lors du contournement d'un élément défectueux on essaye d'utiliser un bloc lui-même utilisé par une autre entrée cela va engendrer un conflit (voir figure 2.22). Pour éviter les conflits une modification de l'architecture a été proposée. Le premier étage de multiplexeurs va être embarqué à l'intérieur des Switch boxes. Cela va conduire à une modification des blocs composants les Switch boxes. En effet, les blocs d'interconnexions sont composés de mini blocs d'interconnexions appelés MSB. Ces MSB vont être dupliqués pour avoir deux sorties indépendantes. Cette modification va permettre d'utiliser chaque MSB pour piloter deux entrées et deux sorties différentes et donc éviter les conflits.

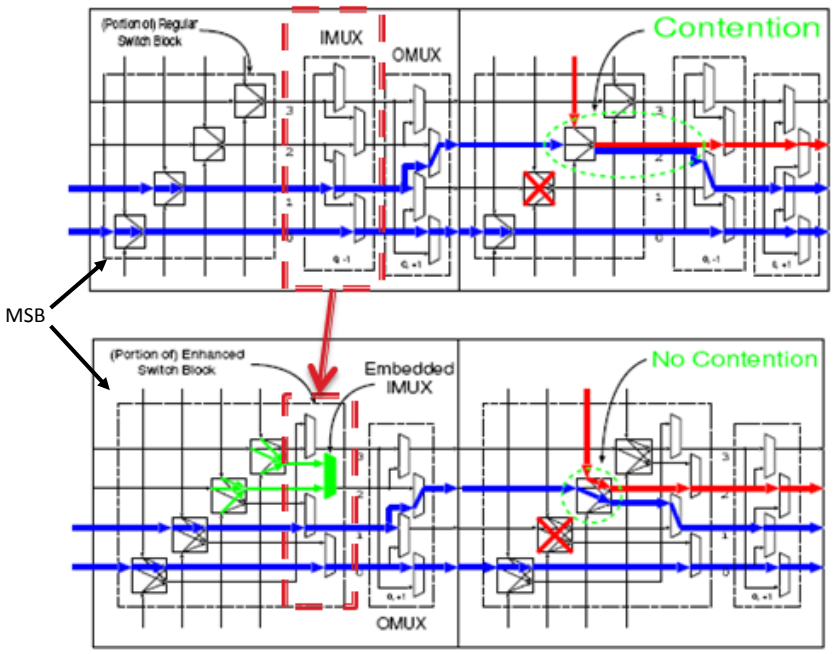


FIGURE 2.22 – Résolution des conflits

Conclusion

En conclusion, on constate dans le tableau 2.2 que contrairement à la CGR, la FGR permet de tolérer un plus grand nombre de défauts quel que soit la taille du FPGA. Cependant, une modification des blocs de base constituant l'architecture du FPGA est nécessaire. L'ajout d'étages de multiplexeurs dans chaque bloc d'interconnexion va augmenter la taille du FPGA de 35 à 50% et va aussi augmenter les délais de 5 à 20%. De plus, le contournement des ressources défectueuses n'est possible qu'avec une cartographie des défauts présents dans le FPGA en représentant l'élément défectueux par un circuit ouvert lors de la configuration du FPGA. Pour finir, cette méthode reste inefficace pour les défauts présents dans les blocs logiques.

	CGR	Apex ALTERA	FGR
Avantages	Contournement des défauts rapide	Contournement des défauts rapide	Tolère un nombre important de défauts
Inconvénients	Suppose que tous les défauts peuvent être corrigés avec un nombre limité de ressources de secours	Suppose que tous les défauts peuvent être corrigés avec un nombre limité de ressources de secours	Nécessite une cartographie des défauts

TABLE 2.2 – Comparaison des solutions matérielles

Tolérance aux défauts des blocs logiques [28]

Les différentes techniques présentées précédemment se concentrent sur la tolérance aux défauts au niveau des blocs d'interconnexions du FPGA. Nous allons maintenant nous intéresser aux techniques de tolérance aux défauts pour les blocs logiques.

La Triple Modular Redundancy(TMR)

Une des techniques les plus populaires quand on parle de redondance est la Triple Modular Redundancy. Le principe est le suivant : trois modules identiques calculent la même fonction et un voteur est ajouté à la sortie qui va transmettre le résultat majoritaire. La TMR est principalement utilisée au niveau système pour les systèmes critiques avec une probabilité très forte de défauts. Cependant, la fiabilité d'un système augmente si on applique cette redondance au niveau le plus bas. Un système peut être divisé en plusieurs sous-systèmes chacun utilisant la redondance de la TMR et mis en cascade pour former le système complet. Dans ce genre de système, une importante partie est consacrée aux voteurs qui seront triplés (voir figure 2.23).

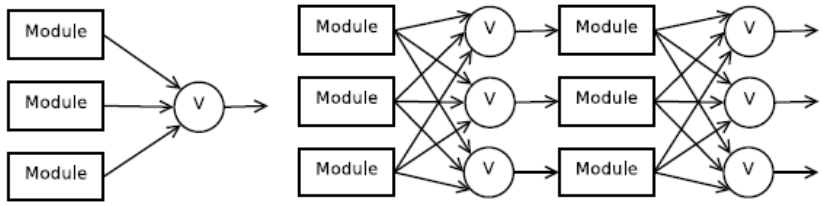


FIGURE 2.23 – Principe de la TMR

On pourrait appliquer la TMR aux portes logiques afin de les rendre plus fiables mais ce n'est pas utilisable en tant que tel (dû à l'importance du voteur par rapport au reste des éléments) c'est pourquoi une solution alternative a été proposée. *L'Interwoven logic* [58] applique de la redondance au niveau des portes logiques. Pour un circuit donné, le nombre de portes logiques va être multiplié par quatre. Un algorithme va ensuite les connecter entre elles d'une manière spécifique qui va permettre de supprimer le voteur. Cette technique peut être appliquée au niveau transistors et est utilisée pour créer des composants plus robustes. Une technique de redondance au niveau transistors connue s'inspirant de la TMR et de *l'Interwoven Logic* consiste à ajouter un transistor en série puis à dupliquer l'ensemble en parallèle (voir figure 2.24).

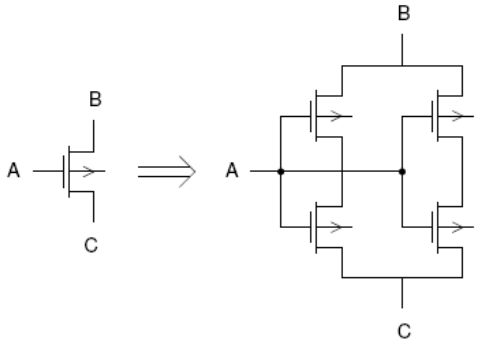


FIGURE 2.24 – Interwoven logic

Décrit à l'origine par Shannon et Moore pour les relais [12], cette solution permet de tolérer les défauts de fabrication tel que les circuits fermés (transistor toujours saturé) grâce aux transistors en série et les circuits ouverts (transistor toujours bloqué) grâce aux transistors en parallèles. Cependant, elle ne permet pas de corriger les autres défauts comme les courts-circuits entre la grille et la source par exemple pour lesquels une autre technique présentée dans la section suivante a été étudiée.

Multiple Short Open Technique(MSO)

Pour pallier le problème de court-circuit entre le drain et la source, une nouvelle technique utilisant la redondance a été développée : la MSO [27]. Pour tolérer les courts-circuits entre la grille et la source, et entre le drain et la grille, les deux réseaux de transistors appelés *pull-up* (charge à la source) et *pull-down* (charge à la masse) seront isolés de l'entrée en utilisant une résistance de tolérance aux défauts. Cette résistance est composée de deux transistors comme montré sur la figure ci-dessous. On combine donc le schéma de la TMR avec cette résistance pour obtenir le schéma de la MSO (voir figure 2.25).

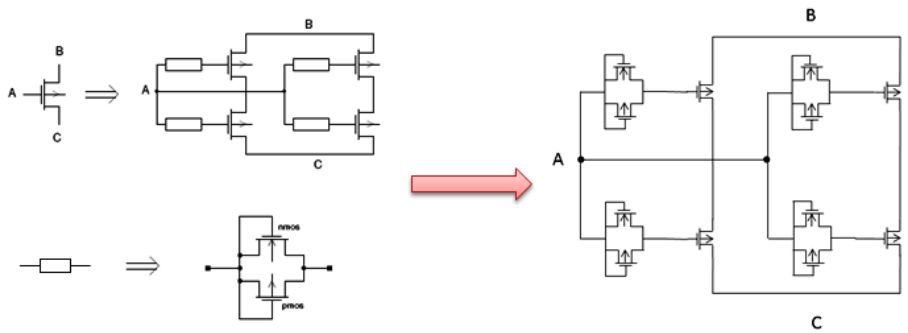


FIGURE 2.25 – Technique de la MSO

La technique de la MSO peut maintenant servir à implémenter une porte logique comme un inverseur ou un élément logique tel que la LUT. Si on considère le schéma en transistor d'un inverseur classique (voir figure 2.26), le principe de la TMR nous dit dans un premier temps qu'il est nécessaire de dupliquer l'élément en série puis le tout en parallèle. Ensuite avec la MSO, un réseau de résistances doit être ajouté pour isoler l'entrée de l'inverseur (voir figure 2.27).

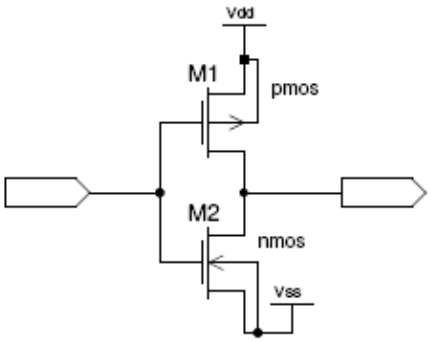


FIGURE 2.26 – Inverseur classique

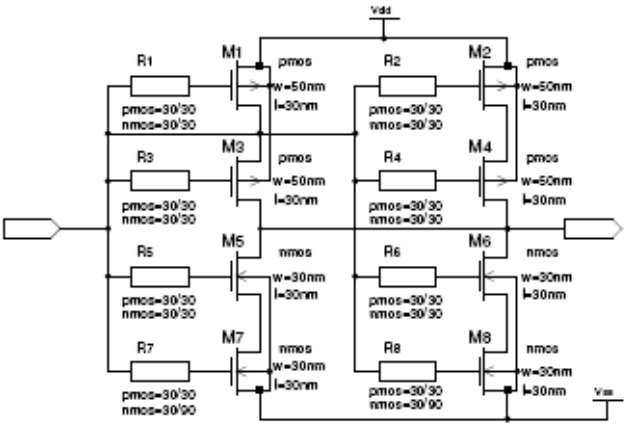


FIGURE 2.27 – Schéma de l'inverseur avec la MSO

On constate que pour un simple inverseur avec cette méthode on multiplie le nombre de transistors par 12 (soit au total 24 transistors au lieu de 2) pour pouvoir le rendre tolérant à n'importe quel défaut de fabrication.

LUT tolérante aux défauts

Le principe de la MSO permet de créer des éléments logiques tolérants aux défauts de fabrication mais augmente de manière significative leur taille. Pour réduire le nombre de transistors utilisés dans le cadre de l'exemple de l'inverseur, puis dans le cadre de l'implémentation d'une LUT à une seule entrée, un algorithme génétique a été développé (figure 2.28). Cette expérimentation tient compte de trois paramètres : la sélection, les croisements et la mutation. La sélection permet de choisir une architecture de départ pour un bloc logique comme la TMR ou la MSO. Le but

est d'optimiser ou d'améliorer le circuit. Les croisements vont permettre d'associer la TMR et la MSO dans le but de rendre le schéma plus robuste. Puis, l'algorithme garantit que l'élément le plus robuste sera utilisé en priorité dans le circuit. La mutation sert à éviter une convergence prématurée de l'algorithme. Des erreurs sont générées afin de créer de nouvelles architectures de base qui n'existaient pas auparavant. La mutation va permettre de créer des blocs originaux qui seront éliminés s'ils apparaissent moins robustes que les blocs de base. Pour finir, une évaluation est effectuée à chaque itération de la simulation pour classifier les solutions proposées en fonction de leur degré d'adaptation au problème de base (dans notre cas la tolérance aux défauts). Pour évaluer la tolérance aux défauts, une faute est injectée aléatoirement sur une entrée et une fonction calcule la probabilité d'avoir une sortie correcte. Le but de ces expérimentations est d'optimiser le circuit redondant en diminuant le nombre de transistors.

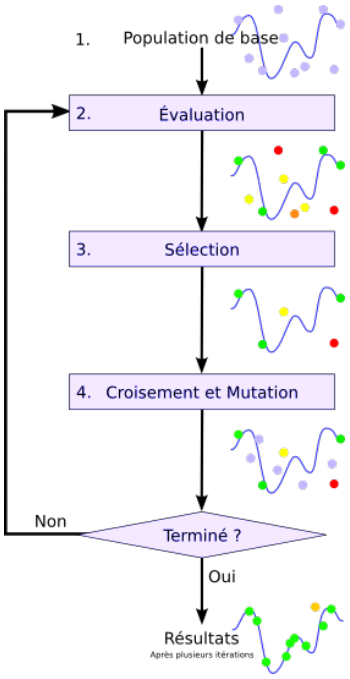


FIGURE 2.28 – Algorithme génétique

Inverseur évolué

On constate que le nombre de transistors composant l'inverseur grâce à l'algorithme génétique est passé de 24 (pour le MSO) à 15 tout en gardant la même fonctionnalité (voir figure 2.29). Les transistors M1, M2, M6 et M7 représentent le réseau *pull-up* tandis que les transistors M12 et M15 représentent le *pull-down*. Ce schéma évolué garde la redondance série de la TMR entre les transistors M1-M6, M2-M7 et M12-M15 pour les problèmes de circuits-fermés. De la même manière, on retrouve la redondance parallèle entre M1-M6 et M2-M7 pour les problèmes de circuits-ouverts. Pour finir, le réseau d'entrée (*Input network*) va assurer la séparation entre l'entrée d'inversion et les réseaux *pull-up* et *pull-down* avec un circuit résistif. Ce circuit résistif va permettre de tolérer les courts-circuits qui pourraient survenir sur l'un des transistors.

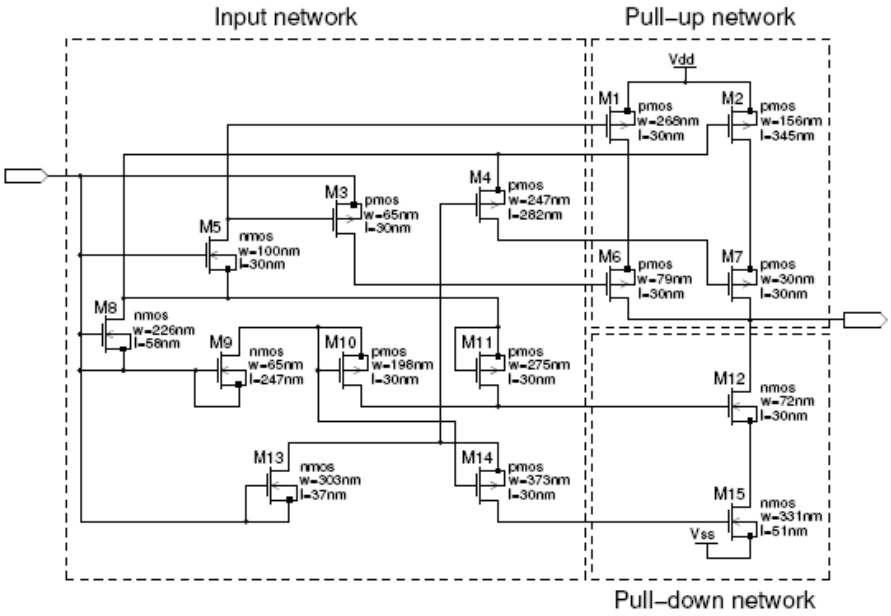


FIGURE 2.29 – Inverseur évolué tolérant aux défauts

De la même manière, l'algorithme a été utilisé pour créer une LUT à une entrée tolérante aux défauts.

Schéma de la LUT évoluée

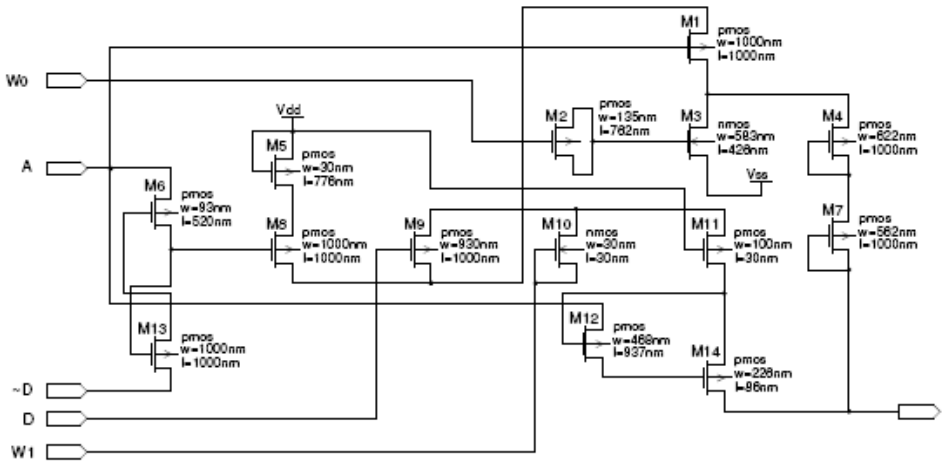


FIGURE 2.30 – LUT évoluée

On constate que la LUT évoluée (figure 2.30) est composée de seulement 14 transistors alors que 22 transistors sont nécessaire pour une LUT à 1 entrée classique (12 pour les rams et 8 pour le multiplexeur). En effet, l'algorithme génétique a modifié la structure des transistors (longueur et largeur) ce qui permet d'avoir la même fonctionnalité avec moins de transistors. Pour évaluer le rendement de cette LUT évoluée d'autres architectures ont été implémentées. La première architecture testée concerne la LUT classique sans redondance (Non-red). La seconde représente la LUT dans laquelle de la redondance a été insérée avec la TMR. Le troisième utilise

le principe de la MSO et pour finir Evolved représente la LUT évoluée. Le tableau ci-dessous (figure 2.31) présente les résultats de simulations Monte-Carlo effectuées sur l'ensemble de ces architectures pour pouvoir comparer le comportement global de chacune en fonctionnement normal puis dans le cas où il y aurait la présence d'un ou plusieurs défauts.

Property	Non-red.	TMR	MSO	Evolved
size	22 trans.	76 trans.	264 trans.	14 trans.
f_{rms}	0.998717	0.999346	0.997665	0.733561
$\hat{f}_{rms}$	0.779882	0.961079	0.994218	0.643702
R_{trad_single}	0.102273	0.858553	1.000000	0.446429
R_{trad_trans}	0.820000	0.882200	0.986700	0.921900
delay	< 1ns	< 1ns	< 1ns	≈15ns

FIGURE 2.31 – Résultats de simulations

Les premiers résultats nous donnent l'erreur quadratique moyenne f_{rms} qui mesure l'écart de prédiction par rapport à la valeur réelle en sortie de la LUT. Puis $\hat{f}_{rms}$ indique l'écart maximal entre la prédiction et la valeur réelle. On constate que la MSO et la TMR sont les architectures les plus performantes puisque les écarts par rapport aux prédictions sont très faibles contrairement à la LUT évoluée. Le délai représente le temps qu'il faut à la sortie pour se stabiliser après un changement d'état sur l'entrée. On constate que dans le cas de la LUT évoluée, la sortie met beaucoup de temps à se stabiliser. On multiplie ce temps par 15 par rapport à toutes les autres solutions testées. Ce délai est dû aux modifications effectuées sur la structure des transistors.R_{trad_single} représente la probabilité pour que le fonctionnement du circuit soit assuré dans le cas où un transistor serait défectueux. Dans ce cas, la MSO est la méthode la plus performante car le fonctionnement est toujours assuré (R_{trad_single} =1). La TMR arrive en seconde position avec 86% car cette méthode ne permet pas de corriger tous les types de défauts. Les faibles résultats pour la LUT évoluée sont justifiés par le fait qu'elle ne contient que 14 transistors, avoir un transistor défectueux est donc très contraignant. Pour finir R_{trad_trans} représente la probabilité pour que le circuit fonctionne. Une nouvelle fois, la MSO permet un fonctionnement du circuit quel que soit les fautes rencontrées. La LUT évoluée arrive en seconde position, étant composée de moins de transistors la probabilité d'avoir un transistor ou plus défectueux est moins importante que pour les autres solutions. En conclusion, la méthode évoluée présente des avantages comme le faible nombre de transistors et la probabilité élevée pour que le circuit fonctionne. Cependant, contrairement aux autres solutions traitées le délai de basculement de la sortie et la faible probabilité de fonctionnement lorsqu'un transistor est défectueux sont des contraintes importantes à prendre en compte lors du choix de l'architecture.

4 Conclusion

En conclusion, les différentes architectures de FPGA présentées dans l'état de l'art ont chacune leurs avantages et leurs inconvénients. La généricité de l'architecture matricielle (*Mesh*) permet l'utilisation de générateurs de FPGA mais 90% de

la surface est utilisée par les ressources d'interconnexions. L'architecture en arbre permet de réduire cet impact de 56% mais augmente la taille du chemin critique et souffre de problèmes de scalabilité. L'architecture utilisée dans cette thèse, devra combiner les avantages des deux précédentes architectures (à savoir la généricité d'une architecture matricielle en réduisant le taux d'utilisation de l'interconnexion par rapport à la logique grâce à une structure en arbre). Puis, nous avons pu constater qu'il existe de nombreuses solutions tant logicielles que matérielles pour la tolérance aux défauts de fabrication. Les bons résultats obtenus lors de l'ajout de ressources de secours à divers niveaux de granularité et le fait que ces techniques sont facilement intégrables dans un générateur d'architecture, nous confortent dans l'idée que l'utilisation de la redondance avec un grain fin est une solution qui peut se révéler avantageuse et qu'il convient d'explorer. Cependant, les techniques les plus performantes suggèrent l'ajout d'une redondance sur toute l'architecture du FPGA ce qui va conduire à une augmentation considérable de sa surface. Notre objectif sera donc dans un premier temps de trouver une méthode permettant de définir la criticité des blocs composants le FPGA pour dans un deuxième temps appliquer une redondance ciblée sur les blocs les plus critiques.

Chapitre 3

FPGA Mesh of Clusters

1 Introduction

Ce chapitre est consacré à la présentation de l'architecture utilisée dans cette thèse, l'architecture *Mesh of Clusters*. Nous commencerons par détailler les spécificités de l'architecture *Mesh of Clusters* qui combine les avantages de l'architecture mesh (sa généricité pour l'ajout de solutions de contournement des défauts) et de l'architecture arborescente (réduction du nombre de switchs utilisés). Nous présenterons ensuite le flot de conception ainsi que le flot de configuration de notre FPGA. Pour finir, nous comparerons les performances de cette architecture par rapport à une architecture matricielle qui est la plus utilisée dans l'industrie.

2 Architecture Mesh of Clusters

L'architecture Mesh of Clusters est composée d'une matrice en 2D de tuiles, chaque tuile étant composée d'un *cluster* qui contient les éléments logiques et des blocs d'interconnexion (Switch boxes)(illustrée figure 3.1). Cette architecture permet d'une part de profiter de la régularité du *mesh* pour connecter les *clusters* et blocs d'interconnexions entre eux (chaque cluster est connecté uniformément à 4 Switch boxes, elles-mêmes reliées entre-elles par le canal de routage (CR) sous forme de lignes et de colonnes) et d'autre part, de gagner en surface au niveau de l'interconnexion des éléments logiques des *clusters* et dans l'interconnexion des Switch boxes en utilisant une architecture en arbre. Contrairement à l'architecture Mesh, nous n'utilisons pas de blocs de connexions, chaque Switch boxes est directement connectée aux clusters et aux Switch boxes voisines dans un ordre bien précis. Comme montré dans la figure 3.2, les entrées/sorties d'une Switch box proviennent des 4 Switch boxes adjacentes ainsi que des 4 clusters adjacents. Grâce à cette configuration, chaque cluster peut se connecter aux 8 clusters adjacents en utilisant ses Switchs boxes adjacentes.

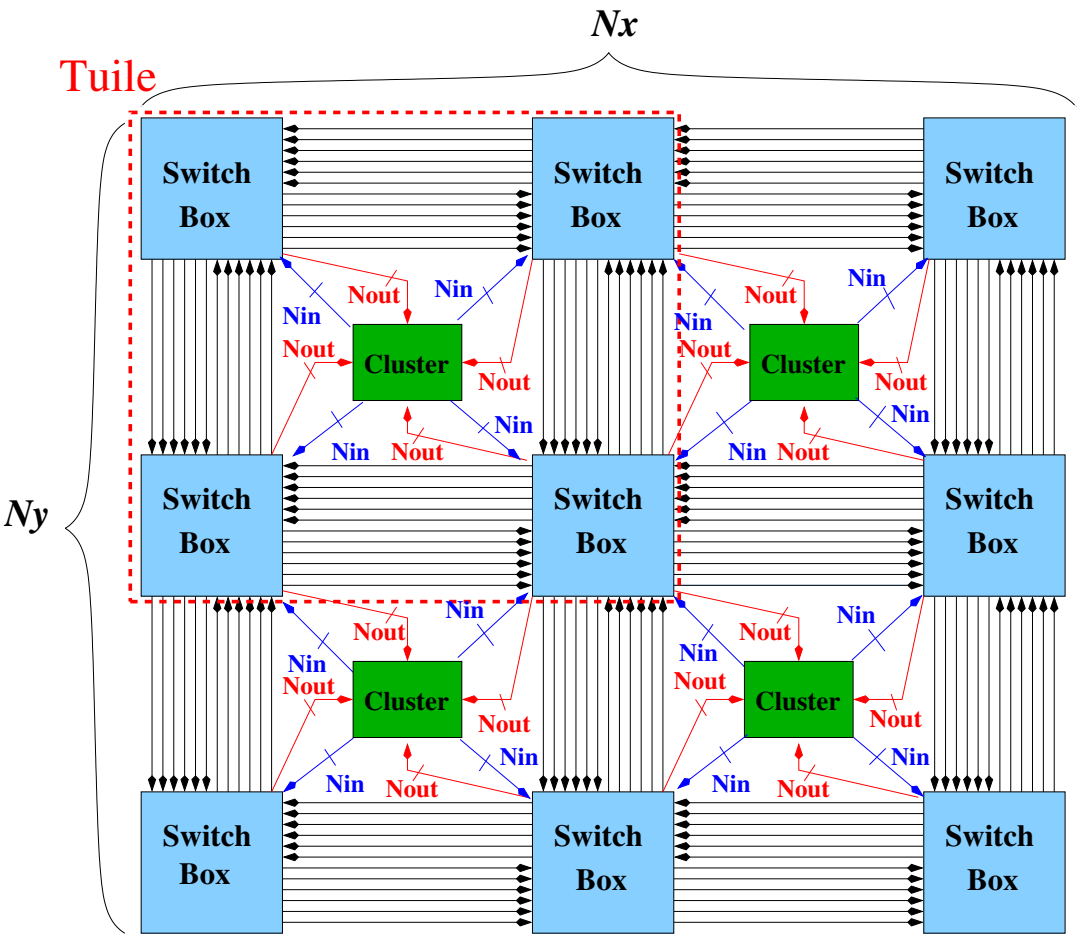


FIGURE 3.1 – Mesh of Clusters FPGA [3]

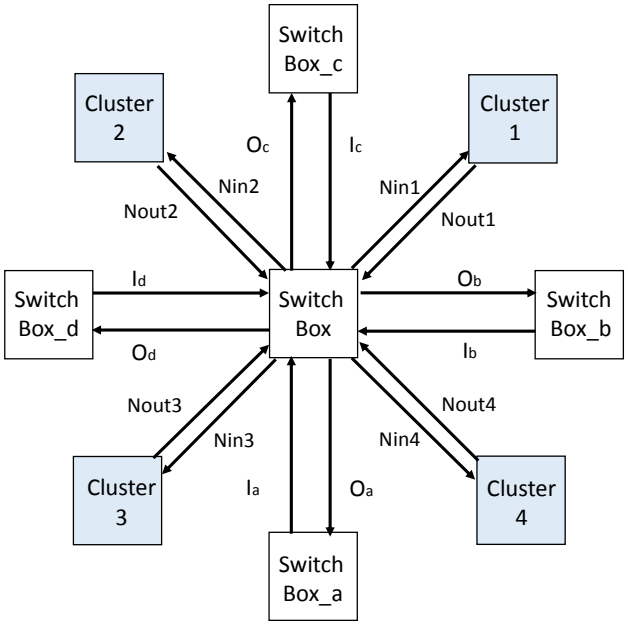


FIGURE 3.2 – Réseau d'interconnexions

2.1 Architecture des Switchs Boxes

Le premier élément essentiel de notre FPGA est la Switch box (figure 3.3). Une Switch box est composée de 2 blocs spécifiques : un Up Mini Switch Box (UMSB) et 2 niveaux de Down Mini Switch Box (DMSB). Ce sont des blocs d'interconnexion composés de multiplexeurs. Les DMSBs sont utilisés pour connecter une Switch box à ses quatre voisines ainsi qu'à un DMSB compris dans la même structure. Le nombre de DMSB de chaque niveau est indépendant mais dépend des paramètres de l'architecture. Pour permettre la connexion entre 2 *clusters* avec la même Switch box, un UMSB est utilisé. Les blocs d'interconnexions (DMSB) sont reliés entre eux dans un ordre bien précis. Le 1er bloc (le plus à gauche) du 1er niveau est connecté à l'entrée du premier DMSB du 2nd niveau, le bloc suivant à l'entrée du second DMSB du 2nd niveau, etc... Pour finir, le réseau d'interconnexion est de type *Butterfly*, ce qui permet de connecter l'ensemble des entrées à l'ensemble des sorties de la Switch box. Le nombre de chaque DMSB/UMSB ou d'entrées/sorties est calculé à partir des paramètres de l'architecture :

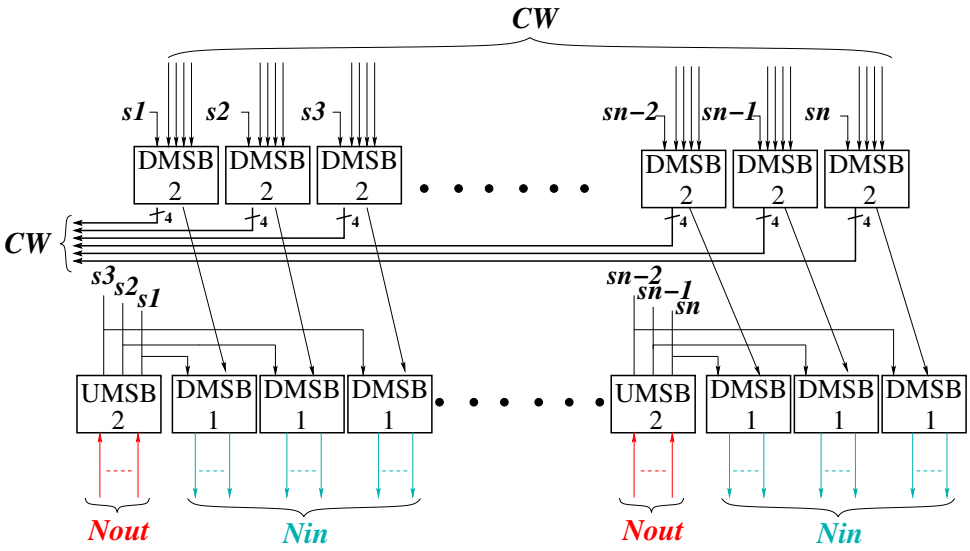


FIGURE 3.3 – Architecture d'une Switch box [3]

$$Nb\_DMSB2 = \frac{CWin}{2} \tag{3.1}$$

$$CWin = CWout = \frac{CW}{2} \tag{3.2}$$

$$Nb\_DMSB1 = \frac{Nin}{4} \tag{3.3}$$

$$Nb\_DMSB2\_inputs = Nb\_adj\_SBs \tag{3.4}$$

$$Nb\_DMSB1\_inputs = \frac{Nb\_DMSB2}{Nb\_DMSB1} \tag{3.5}$$

$$Nb\_UMSB = \frac{Nout}{4} \tag{3.6}$$

$$Nb\_UMSB\_inputs = Nb\_adj\_clusters \tag{3.7}$$

$$Nb\_UMSB\_outputs = \frac{Nb\_adj\_clusters \times \frac{Nout}{4}}{Nb\_UMSB} \tag{3.8}$$

Où Nb_DMSBi est le nombre de DMSB au niveau i, Nb_DMSBi_inputs est le nombre d'entrées pour un DMSB de niveau i, CW_in et CW_out sont le nombre d'entrées et de sorties de la Switch box connectées au canal de routage, CW est le taille du canal de routage, Nb_adj_SBs est le nombre de Switch boxes adjacentes, Nin le nombre d'entrées par cluster, $Nout$ le nombre de sortie par cluster, $Nb\_adj\_clusters$ le nombre de cluster adjacents et Nb_UMSB le nombre d'UMSB par Switch box.

2.2 Architecture des *Clusters*

Le second bloc essentiel de notre FPGA est le *cluster*. Il est composé d'éléments logiques sous forme de CLBs et d'un réseau local d'interconnexion composé de DMSB et d'un UMSB (figure 3.4). Basé sur une architecture de type « *butterfly* », chaque *MSB* (bloc d'interconnexion composé uniquement de multiplexeurs figure 3.5) est relié à tous les éléments logiques (CLBs) dans un ordre bien précis. Le bloc le plus à gauche est connecté à toutes les premières entrées de chaque CLB, le bloc suivant à toutes les secondes entrées,... Pour permettre la connexion entre 2 CLBs d'un même *cluster*, une entité appelée *UMSB* est ajoutée. Elle fonctionne de la même manière que les autres *MSB*, on peut ainsi connecter n'importe quelle sortie de CLBs à n'importe quel *MSB*. En fait, les sorties du bloc UMSB sont reliées à chaque DMSB ce qui permet d'avoir accès à chaque entrées de nos CLBs. Pour finir, la loi de Rent issue des travaux de E.F Rent, nous permet de définir le nombre d'entrées et de sorties [3].

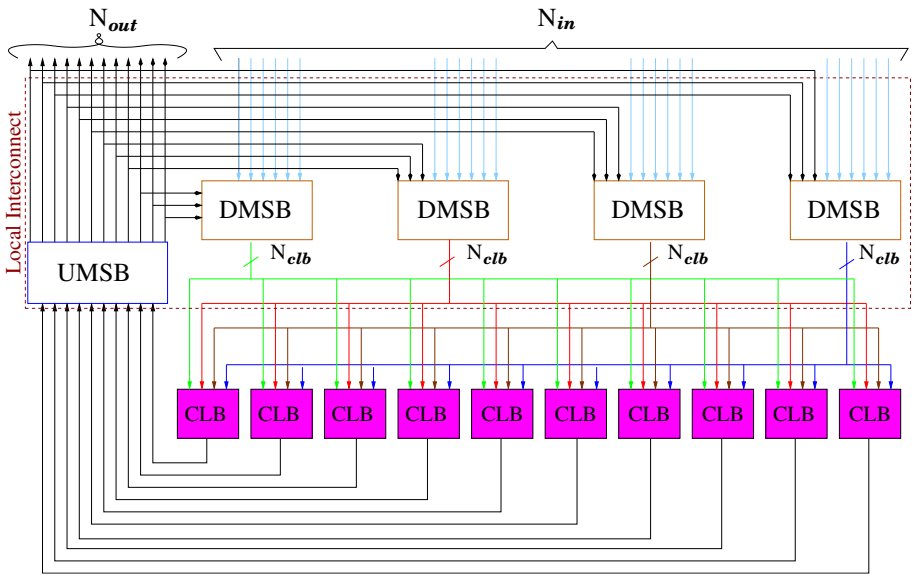


FIGURE 3.4 – Architecture d'un Cluster

En effet, ces travaux ont donnés lieu à une relation remarquable entre le nombre d'I/O et le nombre d'éléments logiques : $Nin + Nout = K.N_{clb}^{p}$ où :

- N_{clb} est le nombre de CLBs par *cluster*
- K est le nombre d'entrées par CLB
- p est le paramètre de Rent de l'architecture $0 \leq p \leq 1$
- N_{in} et N_{out} sont le nombre d'entrées et de sorties par cluster

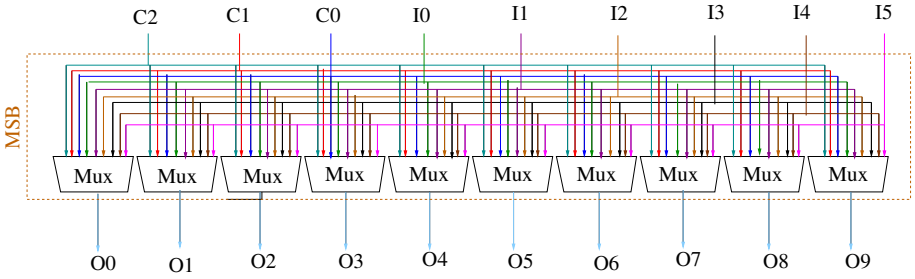


FIGURE 3.5 – Architecture d'un MSB

Comme dans une architecture Mesh classique, les CLBs sont composés d'une Look Up Table à K entrées, d'une bascule D ainsi que d'un multiplexeur pour basculer entre la logique séquentielle et combinatoire (Figure 3.6 ci-dessous). Les signaux Data et Strobe représentent les bits de configurations envoyés aux blocs mémoire.

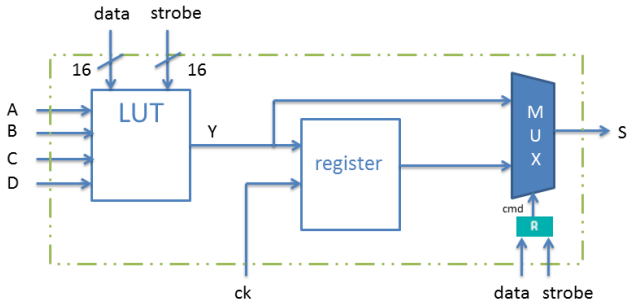


FIGURE 3.6 – Architecture d'un CLB

Une LUT à K entrées nécessite 2^k bits de configuration. L'architecture d'une LUT est représentée figure 3.7, on constate qu'elle est uniquement composée de multiplexeurs 2 :1 et de blocs mémoires.

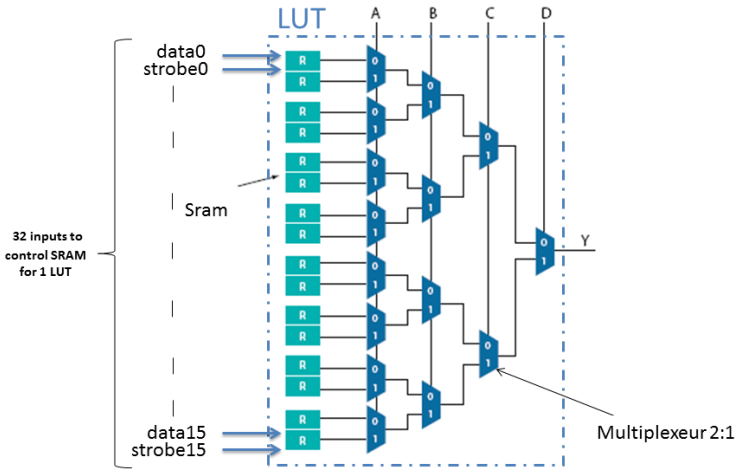


FIGURE 3.7 – Architecture d'une LUT

3 Flot de conception

Un environnement de description de circuit a été développé au LIP6 : STRATUS. Cet outil permet de représenter des circuits et les différents éléments structurels : instances, signaux, connectique... STRATUS utilise le langage python qui nous permet de créer des générateurs pour chaque élément composant notre FPGA. A partir des paramètres de l'architecture définis par l'utilisateur, le code VHDL de l'architecture est créé sous STRATUS à partir des générateurs. Ce code VHDL est ensuite utilisé par le générateur de layout. En effet, le layout du circuit correspondant est créé à partir du code VHDL de l'architecture, des paramètres de l'architecture, de la technologie choisie et du script de placement/routage. Ce script permet de définir de manière automatique la taille du circuit, le placement générique des blocs de base du FPGA et le routage du circuit. On extrait de ce layout les informations de surface, de timing et de consommation nous permettant de comparer les performances des différentes architectures de tolérance aux défauts. Pour finir, le layout généré est validé avec la vérification des règles de dessins (DRC) et une comparaison du layout

avec le schéma structurel de base (LVS). Nous pouvons donc générer n'importe quel type d'architecture pour notre FPGA avec son layout correspondant (figure 3.8).

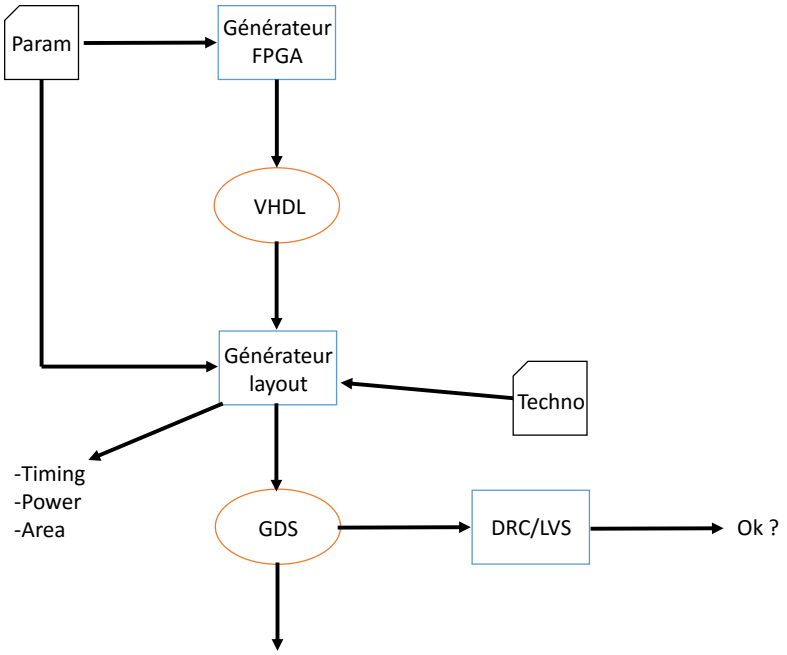
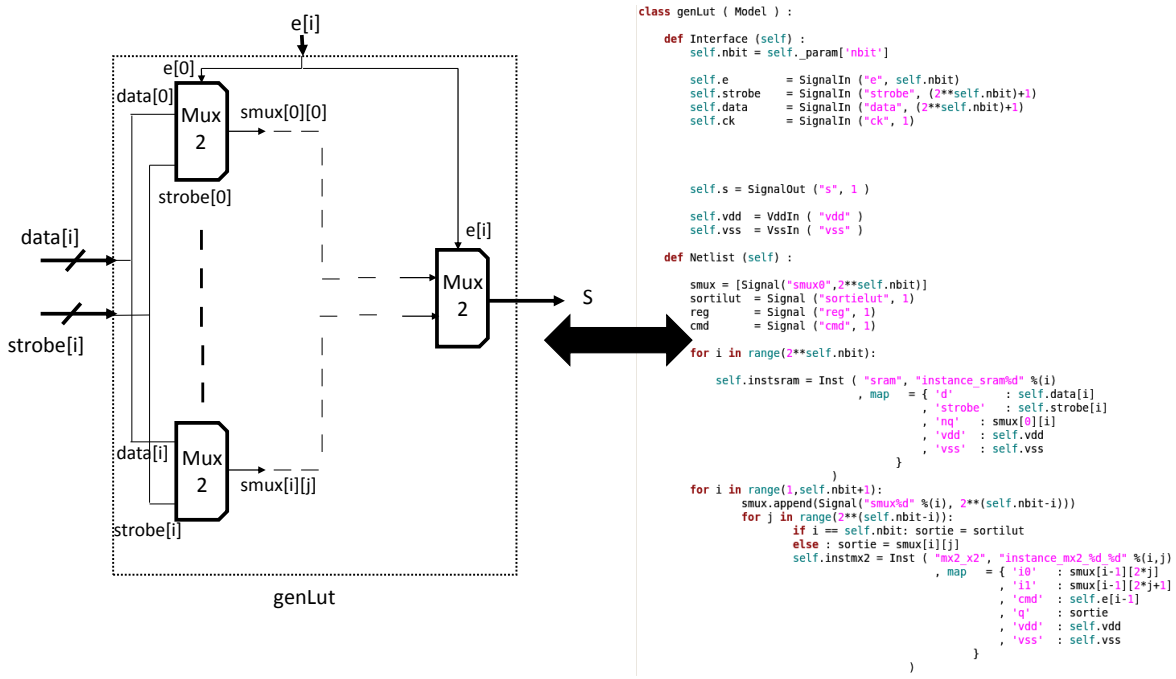


FIGURE 3.8 – Flot de conception

3.1 Environnement Stratus [6]

Stratus est un outil qui permet de représenter des circuits à l'aide d'une description structurelle : instances, signaux, connectique... Il existe 3 classes différentes qui permettent de décrire l'ensemble du circuit. La classe Model sert à définir le générateur. A ce niveau, nous avons la possibilité de décrire des générateurs capables de générer n'importe quel type d'architecture. Ces générateurs permettent de définir des modèles de références utilisés par la deuxième classe Inst (pour Instance). Cette classe instancie le modèle de référence pour créer une cellule avec son nom et ses connexions. Enfin la classe Net définit les signaux attribués à cette cellule, leurs noms, leur taille, un booléen permettant de définir si c'est un signal interne ou externe et la cellule à laquelle il appartient.

Par exemple sur la figure 3.9, le modèle donné est un générateur de LUT . Le nombre d'entrées de la LUT sera un paramètre tandis que le nombre de sortie est fixé à 1 tout comme le signal d'horloge. On constate qu'en fonction du nombre d'entrées, on générera un nombre de RAMs et un nombre multiplexeur 2 :1 adéquate. Le nom des instances est de la forme "instance_.." suivit du composant correspondant (ici une ram ou un multiplexeur). Chaque instance est ensuite définie par des ports (map) et des signaux auxquels ils sont connectés.




```
class genLut ( Model ) :

    def Interface (self) :
        self.nbit = self._param['nbit']

        self.e         = SignalIn ("e", self.nbit)
        self.strobe    = SignalIn ("strobe", (2**self.nbit)+1)
        self.data      = SignalIn ("data", (2**self.nbit)+1)
        self.ck        = SignalIn ("ck", 1)



        self.s = SignalOut ("s", 1 )

        self.vdd  = VddIn ( "vdd" )
        self.vss  = VssIn ( "vss" )

    def Netlist (self) :

        smux = [Signal("smux0",2**self.nbit)]
        sortilut  = Signal ("sortielut", 1)
        reg       = Signal ("reg", 1)
        cmd       = Signal ("cmd", 1)

        for i in range(2**self.nbit):

            self.instsram = Inst ( "sram", "instance_sram%d" %(i)
                                   , map   = { 'd'         : self.data[i]
                                             , 'strobe'    : self.strobe[i]
                                             , 'nq'    : smux[0][i]
                                             , 'vdd'   : self.vdd
                                             , 'vss'   : self.vss
                                             }
                                   )
        for i in range(1,self.nbit+1):
                smux.append(Signal("smux%d" %(i), 2**(self.nbit-i)))
                for j in range(2**(self.nbit-i)):
                        if i == self.nbit: sortie = sortilut
                        else : sortie = smux[i][j]
                        self.instmx2 = Inst ( "mx2_x2", "instance_mx2_%d_%d" %(i,j)
                                              , map   = { 'i0'    : smux[i-1][2*j]
                                                        , 'i1'    : smux[i-1][2*j+1]
                                                        , 'cmd'   : self.e[i-1]
                                                        , 'q'     : sortie
                                                        , 'vdd'   : self.vdd
                                                        , 'vss'   : self.vss
                                                        }
                                              )
```

FIGURE 3.9 – Générateur de LUT sous Stratus

3.2 Le langage python

Le python est un langage de programmation objets développé depuis 1989 par Guido van Rossum [66]. Il permet une approche modulaire et est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion des exceptions. La syntaxe simple et aérée permet au programmeur d'écrire un code lisible ce qui facilite notre travail pour la conception d'un générateur de FPGA.

3.3 Générateur de FPGA

Dans le cadre de notre projet, nous avons décidé de réaliser un générateur de FPGA ce qui nous permettra de créer n'importe quelle architecture. Ce générateur permet de créer n'importe quelle architecture de FPGA de type *mesh of cluster*. Il permettra par la suite d'implémenter plus facilement les solutions de contournement sélectionnées.

– Générateur de cluster :

Le générateur de cluster prend comme paramètres le nombre d'entrées par LUT, le nombre de CLB par *cluster* ($Nclb$) et le nombre d'entrées/sorties par *cluster* ($Nin, Nout$). Le nombre d'entrées par LUT va définir le nombre de DMSB de notre *cluster* :

$$Nb\_In\_LUT = Nb\_DMSB \tag{3.9}$$

Le nombre de CLB par *cluster* va définir le nombre d'entrées de l'UMSB et le nombre de sorties par DMSB :

$$Nclb = Nb\_In\_UMSB = Nb\_Out\_DMSB. \tag{3.10}$$

Pour finir, le nombre d'entrées/sorties par *cluster* va nous permettre de fixer le nombre d'entrées par DMSB ainsi que le nombre de sorties de l'UMSB :

$$Nb\_In\_DMSB = (Nin/Nb\_In\_LUT) \tag{3.11}$$

Avec l'ensemble de ces paramètres, nous pouvons créer n'importe quelle architecture pour les *clusters*.

– Générateur de Switch box :

Le deuxième élément essentiel dans notre architecture est le bloc d'interconnexion. Nous avons donc réalisé un générateur qui prend comme paramètres le nombre d'entrées/sorties de nos Switch boxes et la taille du canal de routage. Le nombre d'entrées par Switch box $CWin$ est définit par le nombre de sorties par *cluster* et par la taille du canal de routage :

$$CWin = (CW * 2) + Nout \tag{3.12}$$

Le nombre de sorties par *cluster* va définir plus précisément le nombre de d'entrées par UMSB :

$$Nout = Nb\_In\_UMSB2 \tag{3.13}$$

La taille du canal de routage va définir le nombre de DMSB de 1er niveau (DMSB1) (le nombre d'entrées par DMSB étant fixé par l'architecture) :

$$CW = Nb\_DMSB1 \tag{3.14}$$

Le nombre de sortie est quant à lui lié au nombre d'entrées par *clusters* ainsi qu'à la taille du canal de routage :

$$Nb\_In\_SB = (CW * 2) + Nin \tag{3.15}$$

Avec l'ensemble de ces paramètres, nous pouvons créer n'importe quelle architecture pour les Switch box. Le diagramme ci-dessous (figure 3.10) décrit les étapes de la création du FPGA. On définit, dans un premier temps, le nombre d'entrées par LUTs, le nombre de CLBs par *cluster*, le nombre d'IO par *cluster*, la taille du canal de routage, la taille du FPGA ainsi que le nombre d'I/O du FPGA. Ces paramètres

sont pris en compte par le générateur de *FPGA* qui va créer la matrice de tuiles souhaitée. Dans un deuxième temps, les paramètres vont servir au générateur de *clusters* et au générateur de Switch box. Ces générateurs vont définir l'architecture des Switch boxes en fonction de la taille du canal de routage et l'architecture des clusters en fonction du nombre de CLBs par cluster, du nombre d'entrées par CLB et du nombre d'I/O par cluster. Pour finir, l'ensemble des générateurs fournissent les codes VHDL de l'ensemble des cellules contenu dans le circuit ainsi que le code VHDL du FPGA.

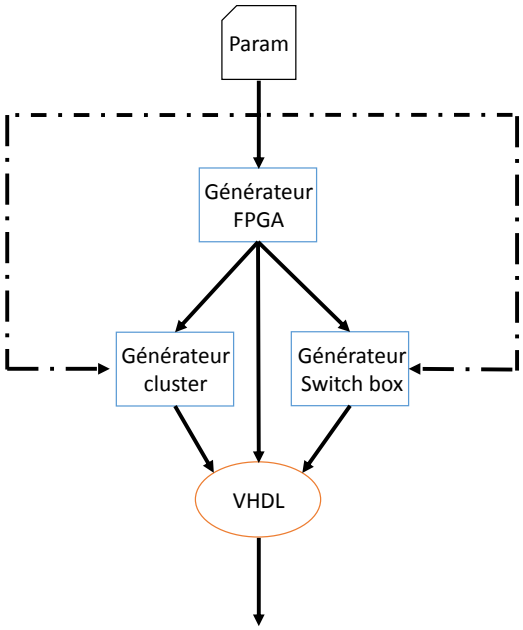


FIGURE 3.10 – Générateur de FPGA

3.4 Génération du layout

Notre flot de conception se décompose en 2 parties. Suite au générateur de FPGA décrit précédemment, le code VHDL de l'architecture est envoyé au générateur de layout. Ce générateur de layout prend en entrée le code VHDL de l'architecture, les paramètres de l'architecture défini lors du flot de conception, la bibliothèque de la technologie choisie et le script d'automatisation. L'ensemble de ces informations permettent de définir la taille optimale du circuit. Ensuite les différentes cellules de l'architecture sont placées automatiquement sur la surface minimale. Puis, l'ensemble des cellules est routé de nouveau de manière automatique. La définition du floorplan, le placement et le routage des architectures sont réalisés à l'aide des outils de placement/routage de CADENCE : SoC-Encounter. Pour finir, le fichier GDS représentant le layout du FPGA est créé et les informations de timing, surface et consommation sont extraites. Pour valider notre générateur de layout, une vérification des règles de dessin est effectuée (via soc-encounter) suivie d'une vérification du layout (via IC de Synopsys) par rapport à la vue schématique du circuit (figure 3.11).

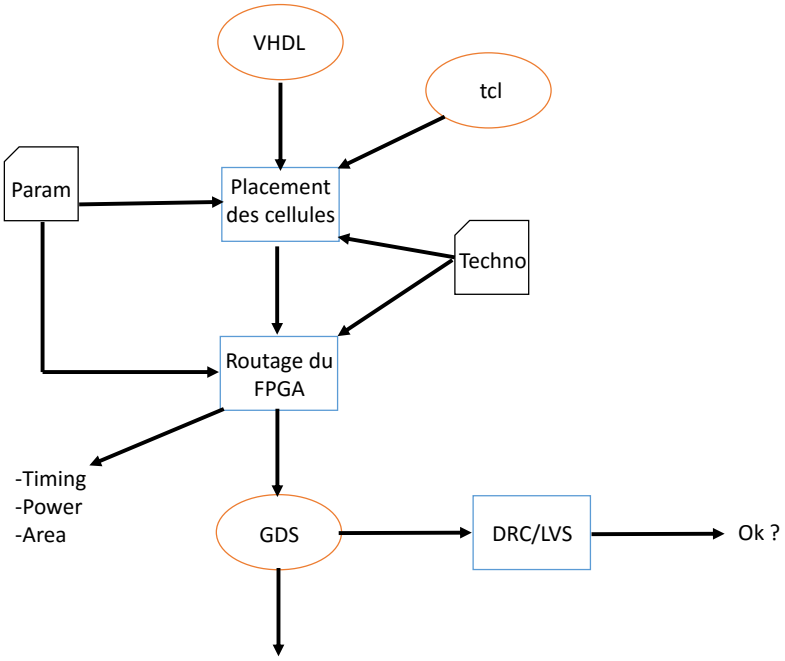


FIGURE 3.11 – Générateur de layout

Paramètres d'entrées

Comme vu précédemment, notre générateur de layout a besoin en entrées de la netlist (VHDL) correspondant à l'architecture du FPGA choisit par l'utilisateur. Puis, une technologie cible doit être défini (65nm par exemple). Le fait de choisir une technologie cible permet de charger une bibliothèque de cellules standard comme des portes logiques de bases avec leurs fonctions et leurs caractéristiques (timing, taille de la cellule...) mais aussi des informations sur les propriétés des différents niveaux de métaux, les VIAs... Lors de cette étape, on a la possibilité d'imposer des contraintes en termes de timing (temps de transmissions des signaux) qui influeront sur le placement des cellules. La figure 3.12montre l'ensemble des paramètres d'entrées nécessaires avant le placement de l'architecture.

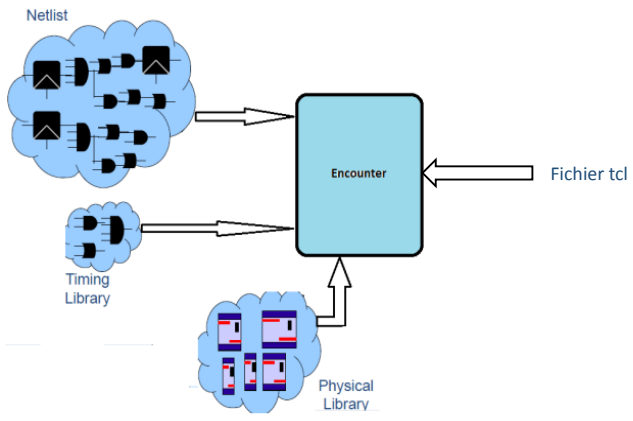


FIGURE 3.12 – Paramètres d'entrées

Définition du floorplan

La dernière étape avant le placement des cellules dans notre générateur est la définition du floorplan. Ce floorplan est défini en fonction des paramètres de l'architecture, sa taille sera donc adaptée à la taille du FPGA. Une fois la surface disponible pour le placement définie, nous avons la possibilité de figer la distribution des entrées/sorties tout autour du circuit comme dans l'architecture d'un FPGA ainsi que de définir une localisation et une surface précise pour les placements de chacune de nos cellules et ainsi respecter la structure générique sous forme de matrice de tuiles d'un FPGA (figure 3.13). Pour finir, la définition du floorplan est une étape très importante car les contraintes figées influeront sur le placement et le routage du circuit. En cas de contraintes trop strictes, notre circuit peut s'avérer non routable.

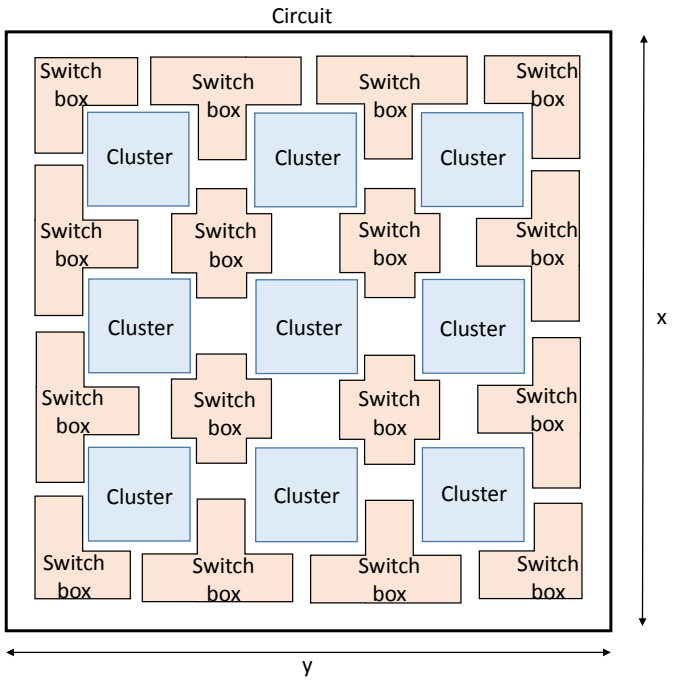


FIGURE 3.13 – Exemple d'un floorplan pour FPGA

Placement des cellules

Après la définition du floorplan, le placement des cellules consiste à organiser les cellules standard de la bibliothèque de la technologie cible sur le circuit. Ce placement peut être automatique ou contraint par le floorplan (figure 3.13 ci-dessus). La figure 3.14 montre le placement d'un cluster et des blocs le composant.

Routage du FPGA

La dernière étape avant la réalisation du layout finale est le routage. Cette étape consiste à connecter l'ensemble des cellules standard précédemment placées sur différents niveaux de métaux. Le routeur va chercher les connexions les plus courtes pour connecter deux instances et répéter cette opération jusqu'au routage complet du circuit. La figure 3.15 représente le routage d'un cluster.

FIGURE 3.14 – Placement d'un cluster

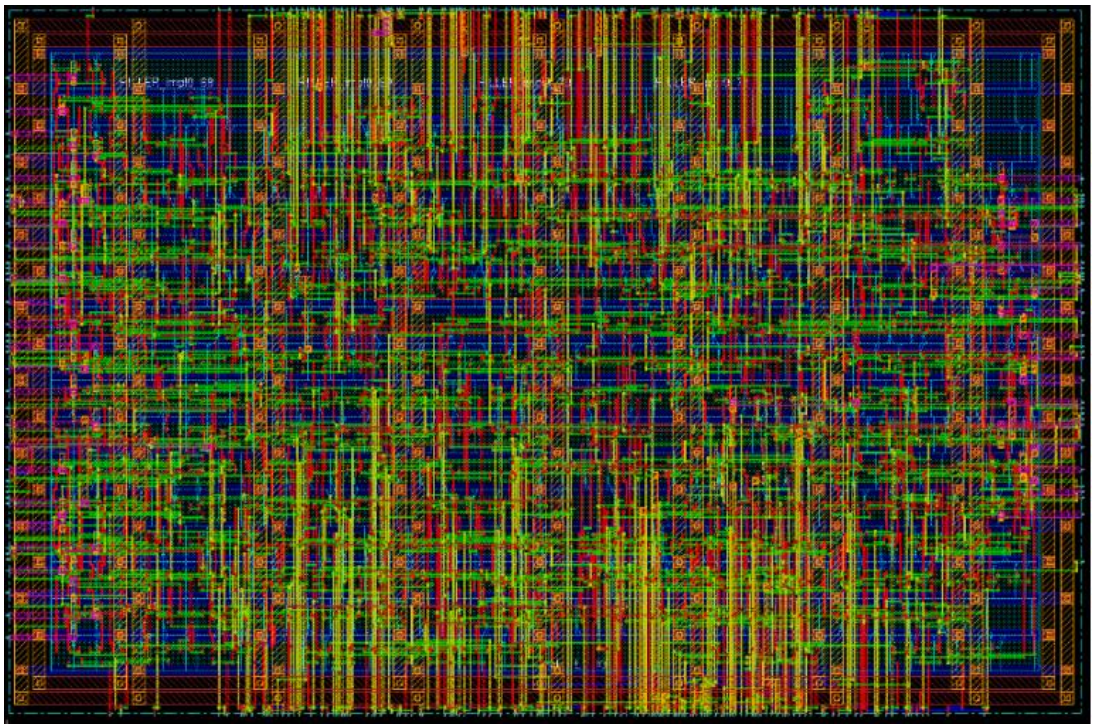


FIGURE 3.15 – Routage d'un cluster

DRC/LVS

Plusieurs vérifications sont nécessaires avant la fabrication d'un circuit. La première étape est appelée : vérification des règles de dessin (Design Rule Check DRC). Cette étape exécutée via SoC-Encounter de CADENCE vérifie à la fois la logique et la conception physique du circuit. Ces règles de dessins sont défini par la technologie de conception choisie. Le DRC vérifie par exemple, l'espacement des cellules, la longueur minimale du canal des transistors, la taille des fils de routage et leurs espacements, l'effet d'antenne... La seconde étape est appelée layout versus schématique (LVS). Le LVS compare la netlist créée à partir du layout à la netlist de base qui provient du générateur de FPGA. Cette étape est réalisée avec l'outil de Synopsis : IC. On peut décomposer le LVS en 3 étapes :

- l'extraction qui consiste à identifier l'ensemble des éléments composants notre circuit.
- la réduction qui consiste à définir la fonctionnalité du circuit à partir des éléments extraits.
- la comparaison : la netlist extraite à l'étape précédente est comparée à la netlist de base. Si celles-ci sont identiques, on considère que le circuit est opérationnel.

Les erreurs les plus souvent rencontrées lors de la vérification du layout sont : des pistes trop proches les unes des autres, des connexions supplémentaires n'apparaissant pas dans la netlist, des connexions manquantes, une cellule manquante...

4 Flot de configuration

Dans cette section, nous allons présenter le flot de configuration permettant d'implémenter une application sur un FPGA. Il est divisé en 5 étapes (illustrées figure 3.16) qui seront détaillées par la suite. Ces différentes étapes sont : la synthèse de l'application en portes logiques de base, le *mapping*, le *clustering* de la netlist créée, le placement de cette netlist sur l'architecture du FPGA et le routage des interconnexions. Une fois ces différentes étapes exécutées, un bitstream du circuit est généré ainsi que des informations portant sur la vitesse et la surface utilisée. Ce fichier de bitstream est un fichier de configuration, il définit la configuration de l'ensemble des blocs logiques et du réseau d'interconnexions du FPGA pour implémenter la fonction désirée.

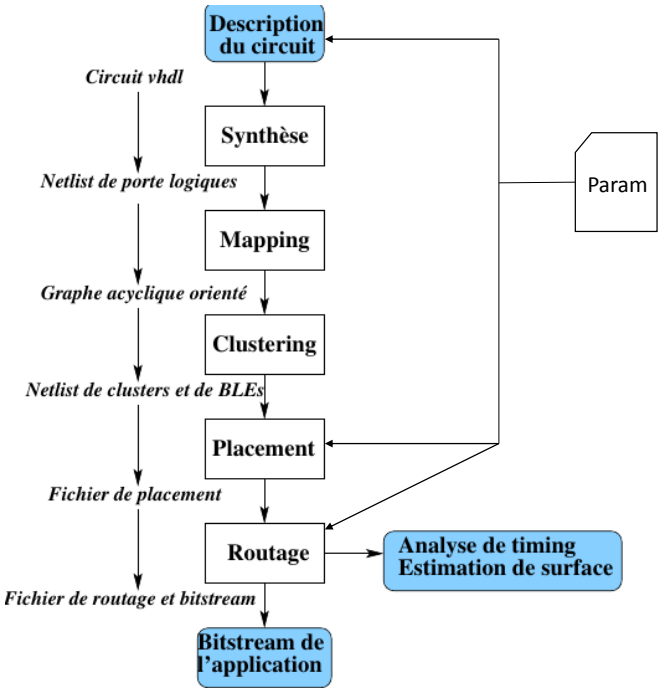


FIGURE 3.16 – Flot de configuration

4.1 Synthèse

La synthèse est la transformation d'un code comportemental donnant les spécifications d'un circuit écrit en langage de description comme le VHDL ou le Verilog en portes logiques de base. La description comportementale peut être constituée de processus séquentiels ou concurrents qui définissent l'ensemble des interconnexions et d'éléments constituants le circuit comme des opérateurs, des blocs combinatoires, des registres... Cette description n'est pas utilisable en tant que telle par le FPGA. La synthèse va donc transformer cette description en une interconnexion de portes logiques de base et de bascules [16] [15].

4.2 Mapping

En sortie de l'outil de synthèse, on obtient un réseau booléen décrivant les interconnexions entre l'ensemble des bascules et des portes logiques. Le circuit peut aussi être représenté par un graphe acyclique orienté (Direct Acyclic Graph). Dans ce graphe, une bascule, une porte logique, une entrée ou une sortie principale est représentée par un nœud et les connexions entre chaque élément sont représentées par des arcs (Fig 3.17). Le mapping consiste à remplacer cette représentation par les éléments de base d'une bibliothèque choisie. Par exemple, dans le cas d'un FPGA, on veut uniquement une netlist composée de LUTs et de bascules (illustrée figure 3.18).

4.3 Clustering

Comme vu précédemment dans ce chapitre, les FPGA sont organisés en plusieurs niveaux de hiérarchie. Le premier niveau est composé des éléments logiques (LUT à k

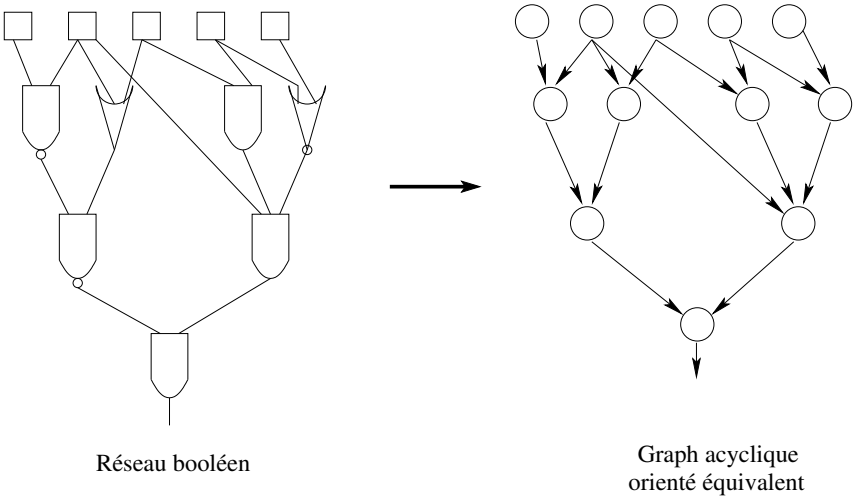


FIGURE 3.17 – Représentation en graphe acyclique orienté

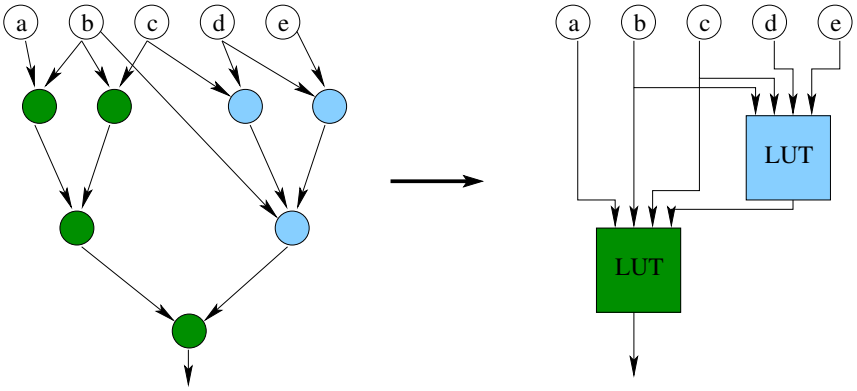


FIGURE 3.18 – Exemple de mapping

entrées) et de la bascule. Puis ces éléments logiques sont regroupés au sein d'un même bloc pour former un cluster. L'étape de clustering va donc consister à transformer notre netlist composée de blocs logiques et de bascules en une netlist de clusters. Cette étape permettra de n'avoir plus qu'à placer des clusters sur l'architecture du FPGA. Il existe trois approches différentes pour réaliser le clustering : l'approche top-down [38] [36] [33], depth optimal [55] [24] et bottom-up [48] [13] [63]. L'approche bottom-up consiste à définir une taille de clusters en termes d'entrées/sorties et de nombre d'éléments logiques par cluster. Tous les clusters de la netlist sont ensuite générés de manière séquentielle. Cette méthode a été développée dans le but de regrouper au sein d'un même cluster les éléments logiques connectés entre eux. De cette manière, on optimise l'interconnexion au sein des clusters. L'un des premiers algorithmes dont le but était de minimiser le nombre de cluster créé par circuit a été développé par Vpack [11] [9]. Les clusters sont créés un par un en utilisant comme premier bloc logique un bloc logique qui n'a pas encore été utilisé et qui possède le plus de signaux. D'autres blocs logiques sont ensuite ajoutés à ce cluster en fonction de leurs attractions jusqu'à ce qu'il ne reste plus de place dans ce cluster. La même opération est répétée jusqu'à ce qu'il ne reste plus d'éléments logiques disponibles. L'objectif est de minimiser le nombre de connexions entre clusters pour améliorer la vitesse du circuit. La façon dont seront sélectionnés les blocs logiques à l'intérieur de chaque cluster sera donc complètement différente. Le clustering à un impact significatif sur la routabilité du circuit, c'est pourquoi beaucoup d'autres algorithmes ont vu le jour comme RPack [14] dont le but est d'améliorer la routabilité ou un dérivé appelé T-RPack [14] qui cherche aussi à minimiser le délai du circuit. Dans le cadre de l'utilisation de l'architecture Mesh of Clusters, nous utilisons l'algorithme T-Vpack [48] qui comme T-RPack cherche à minimiser le délai du circuit. De plus, les clusters formés sont quasiment similaires aux blocs logiques présents dans certaines gammes de FPGA de chez Xilinx et Altera.

4.4 Placement

Le placement consiste à organiser de manière spécifique sur l'architecture du FPGA les clusters formés précédemment. L'objectif est de placer les clusters et les blocs logiques les plus interconnectés le plus près possible pour minimiser la longueur des fils de routage. On peut aussi utiliser un placement permettant d'avoir une répartition homogène du nombre d'interconnexions ou dans le but de maximiser la fréquence d'utilisation du FPGA. La figure 3.19 montre un exemple de placement des blocs logiques (gris)sur l'architecture d'un FPGA.

La façon dont est placé le circuit a un impact important sur le résultat du routage, c'est pourquoi il existe dans la littérature différentes méthodes de placement comme le placement basé sur le partitionnement [41] (min-cut partionning) notamment utilisé avec les architectures hiérarchiques ou le placement *recuit simulé* [62]. L' objectif étant bien évidemment d'améliorer les performances du FPGA soit en minimisant la longueur des fils d'interconnexions, soit en minimisant le nombre de fils d'interconnexions.

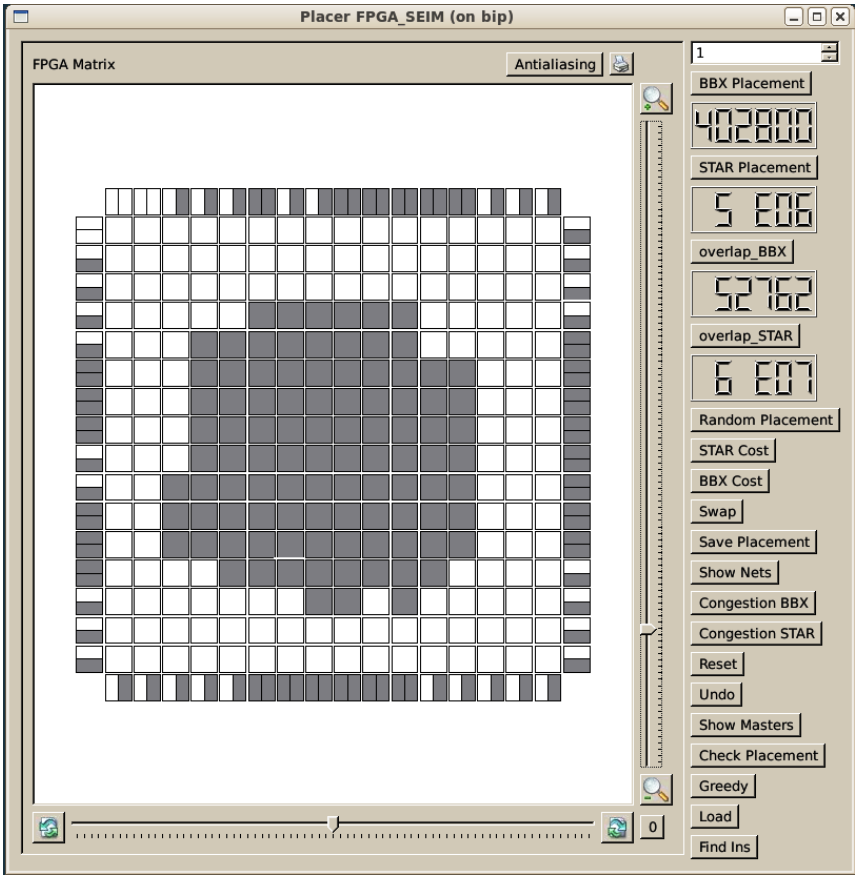


FIGURE 3.19 – Exemple de placement

4.5 Routage

Le routage d'un FPGA est la dernière et la plus importante étape dans le flot de configuration du FPGA. Elle consiste à créer des connexions physiques entre les éléments logiques du FPGA en respectant les paramètres de l'architecture (nombre de fils dans le canal de routage etc...). Chaque signal doit donc être connecté aux ressources de routage du FPGA. Pour cela, le FPGA est généralement représenté comme un graphe de routage orienté $G(V, E)$ [9]. Les entrées/sorties des éléments logiques et les segments de routage sont représentés par un ensemble de sommets $V = v_1, ..., v_i$. Puis, les connexions électriques entre deux sommets v_i sont représentées par un ensemble d'arêtes $E = e_1, ..., e_m$. La figure 3.20 représente une partie du graphe de routage pour un FPGA matriciel.

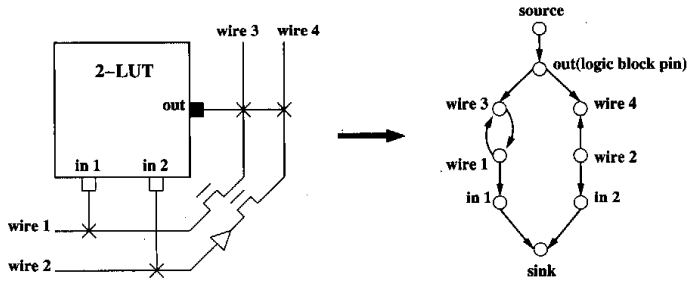


FIGURE 3.20 – Modélisation d'un graph de routage d'un FPGA [9]

Router les signaux dans un FPGA va donc consister à trouver un chemin entre la source et la destination d'un signal via plusieurs segments du graphe de routage. On appelle cet ensemble de segments entre la source et la destination un arbre de routage. Dans un FPGA, un segment ne peut être utilisé pour router deux signaux différents, on va donc créer autant d'arbres de routage (disjoints) que de signaux à router. L'algorithme utilisé pour le routage est appelé Pathfinder [52]. Cet algorithme itératif est basé sur une approche de négociation pour router tous les signaux d'une netlist. Lors de la première itération, le routage est libre et certaines ressources de routage peuvent être utilisées pour router plusieurs signaux, elles sont alors congestionnées. Dans les itérations suivantes, le coût d'utilisation d'une ressource est incrémenté à partir du nombre de signaux partagés par cette ressource et de son historique de congestion. Avec cette méthode, les signaux sont forcés à négocier pour les ressources partagées. Dans le cas où une ressource a un coût très élevé (donc fortement congestionnée), les signaux partagés par cette ressource vont essayer d'utiliser des ressources avec un coût moins important (donc moins congestionnées) dans le cas où il existe des ressources moins coûteuses. Le routeur implémente un certain nombre d'itérations jusqu'à ce qu'il n'y ait plus de conflits (ressources de routage utilisées par plusieurs signaux). A chaque itération, l'ensemble des signaux est routé et leurs coûts sont mis à jour de manière dynamique. Cette méthode qui consiste à router puis dérouter un signal est appelée "*Rise-up and retry*". Une donnée importante permet de vérifier la qualité du routage : le délai du chemin critique (CPD). Le délai du chemin critique représente le délai maximum des chemins combinatoires contenus dans la netlist. Ce délai permet d'obtenir la fréquence de fonctionnement de notre circuit.

Analyse de timing et estimation de la surface

L'outil de placement/routage nous permet d'extraire des informations importantes sur les performances de notre architecture. Dans un premier temps, l'analyse de timing, permet d'évaluer les performances de notre architecture en terme de vitesse de fonctionnement. Cette analyse est faite après placement et routage d'une application et nécessite deux graphes différents :

- Le graphe de routage : il décrit la manière dont est interconnecté l'ensemble des instances de notre netlist. Il permet d'évaluer le délai de routage en ajoutant les délais de chaque élément utilisé dans le chemin critique (connexions et ressources). Ce graphe est dépendant de l'architecture utilisée.
- Le graphe de timing : c'est un graphe acyclique créé à partir de l'hypergraphe de la netlist. Un délai est attribué à chaque arc du graphe grâce aux résultats issus du graphe de routage. A partir de ces informations, la fréquence minimale de l'horloge est alors déterminée.

Le délai du chemin critique est dépendant de l'application placée et routée sur le FPGA et donc de sa netlist. Dans un premier temps, le circuit complet est modélisé par un graphe où chaque LUT, registre, entrée/sortie est modélisée par un nœud. Toutes les ressources de routage sont quant à elles modélisées par des arcs connectant l'ensemble des nœuds. Pour définir le délai d'un nœud, un arc est ajouté entre les entrées/sorties du nœud. Un délai est alors attribué à ce nœud. On calcule ensuite le délai du circuit en appliquant un stimuli aux points d'origines (entrées du FPGA, sorties des registres) et on mesure le délai de chaque nœud. Le délai de ce nœud est alors défini par l'équation 3.16 :

$$T_{arrival} = max_{j \in fanin(i)} \{T_{arrival}(j) + delay(i,j)\} \tag{3.16}$$

Où i représente le nœud utilisé, $delay(i,j)$ le délai attribué à l'arc connectant le nœud i au nœud j. Le délai maximum du circuit sera donc le $T_{arrival}$ max une fois tous les nœuds du circuit testés.

Méthodologie

Pour chacun des 20 benchmarks MCNC, la *netlist* de l'application (.bliff) est transformée par le logiciel *tv-pack* en une *netlist* de *cluster* en fonction de divers paramètres (nombre d'entrées par LUTs, nombre d'entrées par *clusters* et nombre de LUTs dans un *cluster*) et fournit une nouvelle *netlist* (.net). Cette nouvelle *netlist* est ensuite utilisée dans le logiciel de placement pour placer l'application sur l'architecture de notre FPGA *Mesh of Cluster*. Ce placement tient compte des mêmes paramètres que *tv-pack* (nombre d'entrées par LUTs, nombre d'entrées par *clusters* de la *netlist* et nombre de LUTs dans un *cluster*) ainsi que de la taille du FPGA. Le placeur fournit un fichier de placement (.place) image du placement de l'application. Pour finir, le routeur prend en entrée le fichier de placement et la *netlist*. Les mêmes paramètres que précédemment vont lui être attribué (nombre d'entrées par LUTs, nombre d'entrées par *clusters* de la *netlist*, nombre de LUTs dans un cluster, taille du FPGA) ainsi que de nouveaux qui vont permettre de modifier l'architecture (nombre d'entrées et de sorties par cluster de l'architecture et taille du canal de routage) dans le but de trouver la configuration optimale. Le routeur fournit le fichier de routage(.route), le bitstream(.bit) ainsi que divers paramètres relatifs à

l'architecture (nombre de RAM utilisées, nombre de multiplexeurs utilisés, nombre de Switch Boxes dans le chemin critique...). La figure 3.21 montre le résultat en terme de délai après routage pour le bench MCNC *alu*.

```
critical path delay : 22.366
************* Iter number 72 *************
conflicts number = 2
critical path delay : 22.366
************* Iter number 73 *************
conflicts number = 1
critical path delay : 22.366
************* Iter number 74 *************
conflicts number = 0
critical path delay : 22.366
Successfully routed after 74 routing iterations
  ++ Routing file generationalu.route
  ++ Verify Routing Result
  ++ Routing is verified !!!
All graph wires = 43438
Total switches number = 250272
Number of used switches is 25351
critical path delay : 22.366
 Critical path crosses 117 switches
  ++ Bitstream file generation
  ++ Bitstream file generated successfully
-bash-4.1$
```

FIGURE 3.21 – Exemple de résultats en terme de délai

L'estimation de la surface est elle aussi effectuée après placement/routage. Le routeur nous fournit les informations sur le nombre de switchs, le nombre de multiplexeurs et le nombre de SRAMs utilisés à chaque niveau de l'architecture. On peut voir par exemple sur la figure 3.22, toujours pour le benchs *alu* que cette architecture utilise 242 954 multiplexeurs, 159 206 mémoires SRAMs, 41068 buffers et 250 272 switchs. A partir des informations utilisées pour le routage (taille du FPGA, taille du canal de routage, architecture des clusters), nous pouvons déterminer la surface totale utilisée (par exemple $707581.25 \times lambda2$ pour le bench *alu* illustré figure 3.22).

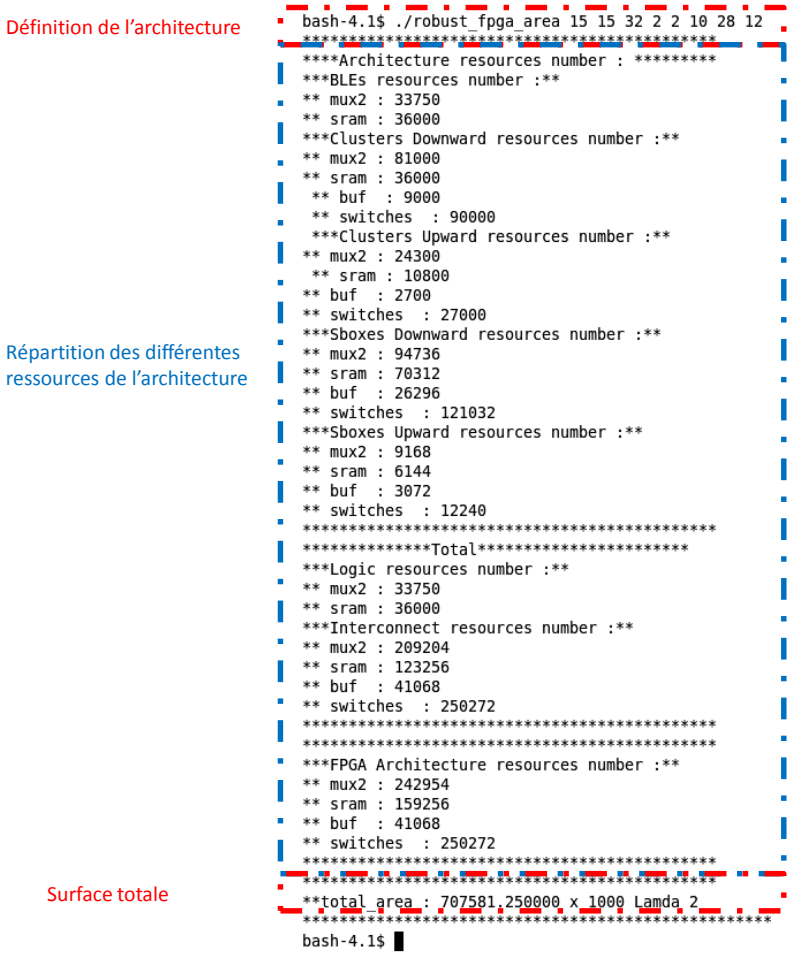


FIGURE 3.22 – Exemple sur le nombre de switch utilisés et la surface totale

5 Mesh vs Mesh of Clusters

Dans cette section, notre objectif est de définir la meilleure architecture Mesh of Clusters (MoC) en terme de surface en faisant varier le nombre d'entrées/sorties par clusters lors de la création de la netlist avec l'outil de clustering T-VPack mais aussi lors de la définition de l'architecture pour le routage. Plus le nombre d'entrées par clusters de la netlist va diminuer et plus nous aurons besoin de clusters. Pour l'architecture, le fait d'augmenter le nombre d'entrées/sorties par cluster augmentera la routabilité du cluster et permettra de diminuer la taille du canal de routage. L'objectif est de trouver le meilleur compromis entre la netlist et l'architecture. Le flot de configuration du FPGA pour l'ensemble des tests est détaillé ci-dessous figure 3.23.

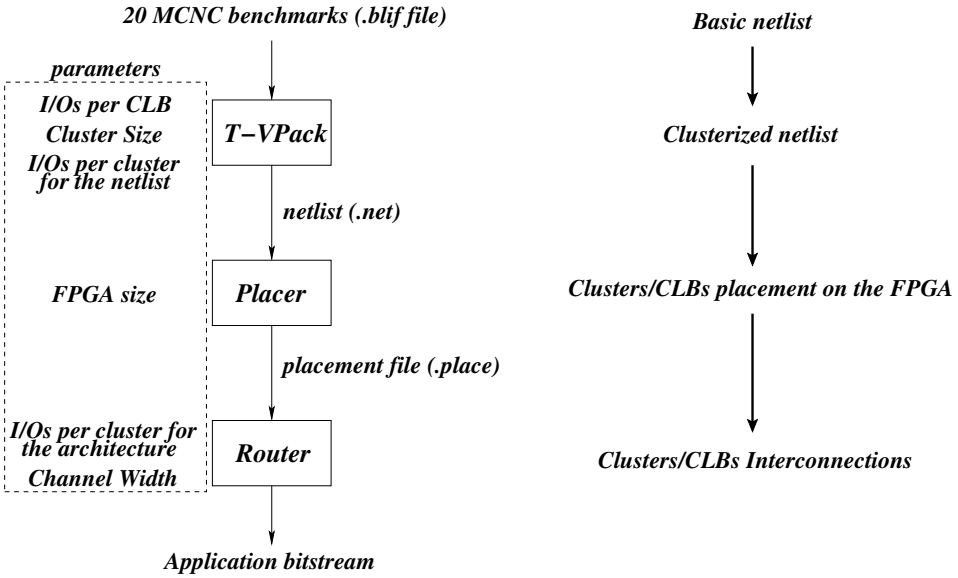


FIGURE 3.23 – Flot de configuration du FPGA

Nous avons choisi de travailler dans cette étude avec des clusters contenant 8 blocs logiques car elle correspond à la meilleure architecture pour un cluster de type Mesh of Custers [7]. Le paramètre de Rent de la netlist défini le nombre d'entrées/sorties par clusters lors du clustering de la netlist tandis que le paramètre de Rent de l'architecture défini le nombre d'entrées/sorties par clusters utilisés pour le routage de l'application. Le tableau 3.1 montre les résultats obtenus pour les 20 benchs MCNC en terme de surface moyenne et de taille de canal de routage (CW) pour différentes architectures de clusters.

Pour certaines configurations de netlist et d'architecture, on constate que l'ensemble des benchs sont Non Routable (NR). La surface est calculée à partir des tailles des éléments logiques de bases (Multiplexeurs, buffers, SRAMs) fournie par la bibliothèque de composants. On constate que dans tous les cas le paramètre de Rent de l'architecture est plus grand que celui de la netlist. En effet, les benchs ne seraient pas routables avec cette architecture si nous choisissions des Rent égaux pour la netlist et pour l'architecture. Ceci est dû à l'architecture du réseau d'interconnexion présent dans le cluster. La figure 3.24 montre l'évolution de la surface en fonction du paramètre de Rent de la netlist et de l'architecture.

Rent netlist	taille du FPGA (NxN)	Surface moyenne vs Rent de l'architecture									CW moyen vs Rent de l'architecture								
		0.42	0.56	0.67	0.83	0.89	0.95	1	1.08	1.16	0.42	0.56	0.67	0.83	0.89	0.95	1	1.08	1.16
0.42	43x43	NR	2477	2955	3327	3349	3587	3680	4017	4476	NR	14	12	12	12	12	12	12	12
0.56	22x22	NR	NR	1542	1507	1449	1529	1650	1704	1830	NR	NR	34	24	22	22	22	22	22
0.67	21x21	NR	NR	NR	1250	1208	1241	1254	1359	1441	NR	NR	NR	34	32	28	28	26	26
0.83	18x18	NR	NR	NR	NR	1245	1234	1292	1335	1346	NR	NR	NR	NR	44	38	34	34	32
0.89	18x18	NR	NR	NR	NR	1349	1300	1240	1220	1282	NR	NR	NR	NR	48	44	38	38	34
0.95	18x18	NR	NR	NR	NR	NR	1243	1241	1239	1277	NR	NR	NR	NR	NR	46	42	42	36

TABLE 3.1 – Surface moyenne et canal de routage moyen pour les 20 benchs MCNC

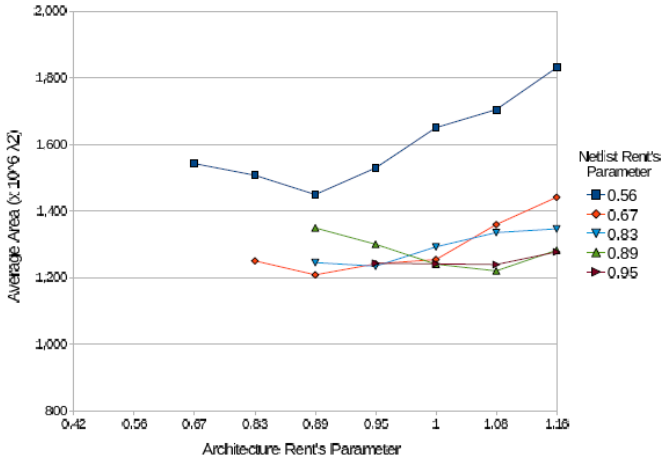


FIGURE 3.24 – Variations de la surface en fonction du paramètre de rent de l'architecture et de la netlist

On constate Fig.3.24 que le canal de routage ne change que très peu lorsque l'on augmente le paramètre de rent de l'architecture, cependant la surface elle augmente. Ces résultats s'expliquent par le fait qu'en augmentant le paramètre de Rent de l'architecture, on augmente le nombre d'entrées/sorties par cluster donc on augmente la surface occupée par les clusters. On constate aussi que le fait d'augmenter le paramètre de Rent de la netlist nous oblige à augmenter la taille du canal de routage pour que l'application reste routable. On peut noter aussi qu'après un certain point le fait d'augmenter le paramètre de rent de l'architecture et de la netlist va augmenter la surface du FPGA car on augmente considérablement le nombre de multiplexeurs à l'intérieur des Switch boxes. Il faut donc trouver le bon compromis entre le Rent de la netlist et celui de l'architecture. Les meilleurs résultats sont obtenus pour un Rent d'architecture de 0.89 et pour un Rent de netlist de 0.67. Pour comparer les performances de notre architecture par rapport à une architecture matricielle, nous avons décidé de placer et de router les 20 benchs MCNC sur une architecture à 8 blocs logiques par cluster avec des LUTs à 4 entrées. Le tableau 3.2 nous permet de définir la plus petite architecture (matricielle et MoC) et la taille de canal de routage minimale qui permet de router chaque application.

MCNC benchmarks				VPR architecture Mesh			Mesh of Clusters		
Noms	Nb LUTs	IN pads	OUT pads	taille FPGA	taux d'occupation(%)	CW	taille FPGA	taux d'occupation(%)	CW
alu4	1522	14	8	15x15	84	50	16x16	74	30
apex2	1878	39	3	16x16	91	50	18x18	72	34
apex4	1262	9	19	13x13	93	46	15x15	70	34
bigkey	1707	263	197	15x15	94	34	15x15	94	24
clma	8383	61	82	33x33	96	72	36x36	80	46
des	1591	256	245	15x15	88	32	16x16	77	24
diffeq	1497	64	39	14x14	95	32	15x15	83	28
dsip	1370	229	197	15x15	76	38	15x15	76	20
elliptic	3604	131	114	22x22	93	58	23x23	85	36
ex1010	4589	10	10	25x25	91	62	28x28	73	36
ex5p	1064	8	63	12x12	92	44	14x14	67	34
frisc	3556	20	116	22x22	91	58	23x23	84	38
misex3	1397	14	14	14x14	89	46	15x15	77	30
pdc	4575	16	40	25x25	91	86	28x28	72	48
s298	1931	4	6	16x16	94	48	17x17	83	26
s38417	6406	29	106	29x29	95	44	30x30	88	30
S38584	6447	39	304	29x29	95	40	29x29	95	34
seq	1750	41	35	15x15	97	50	17x17	75	34
spla	3690	16	46	22x22	95	78	25x25	73	42
tseng	1047	52	122	12x12	90	30	12x12	90	24
moyenne	2963	66	88	19x19	91	46	21x21	79	32

TABLE 3.2 – Caractéristiques des netlists et des architectures

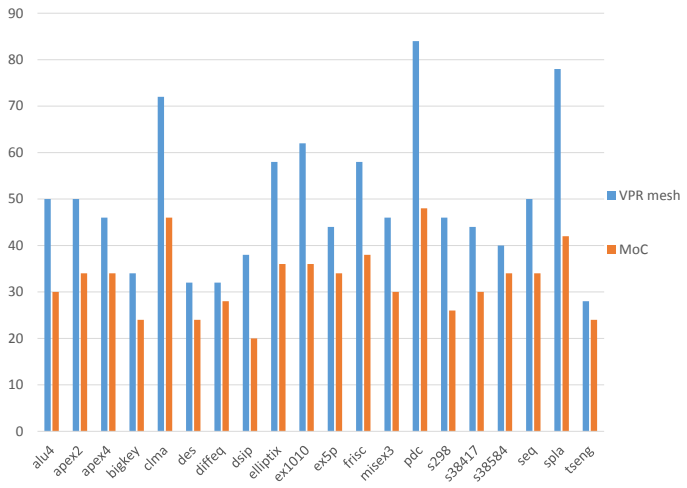


FIGURE 3.25 – Variation du CW

Dans le tableau 3.3, on peut constater qu'avec notre architecture MoC, chaque application peut être routée en utilisant moins de switch (SW) comparé à l'architecture matricielle. En moyenne, notre architecture permet de réduire de 41% le nombre de switchs utilisés. Dans le meilleur cas, avec l'application la moins complexe *Tseng* le nombre de switchs utilisés est même réduit de 44% contre 40% avec l'application le plus complexe *CLMA*. Notre architecture est donc plus intéressante quel que soit la complexité du bench utilisé. En moyenne, on constate une réduction de la surface utilisée de 42% pour implémenter l'ensemble des benchs comparé à l'architecture matricielle.

MCNC	VPR architecture Mesh		Mesh of Clusters		Gain	
circuits	SW$\times 10^3$	Area(λ^2)$\times 10^6$	SW$\times 10^3$	Area(λ^2)$\times 10^6$	SW(%)	Area(%)
alu4	390	1246	216	613	44	50
apex2	444	1278	292	828	34	35
apex4	281	812	204	557	27	31
bigkey	332	979	171	486	48	50
clma	2328	6576	1386	3964	40	39
des	325	953	196	556	39	41
diffeq	277	815	183	520	33	36
dsip	349	1023	158	449	54	56
elliptic	915	2606	489	1393	46	46
ex1010	1226	3477	723	2060	41	40
ex5p	235	683	178	505	24	26
frisc	912	2600	505	1436	44	44
misex3	326	943	190	541	41	42
pdc	1491	4166	862	2468	42	40
s298	435	1256	226	642	48	48
s38417	1371	3978	749	2139	45	46
S38584	1312	3830	783	2148	40	43
seq	392	1277	261	742	33	41
spla	1085	3049	633	1803	41	40
tseng	200	588	111	313	44	46
moyenne	726	2106	425	1208	41	42

TABLE 3.3 – Mesh vs Mesh of Clusters

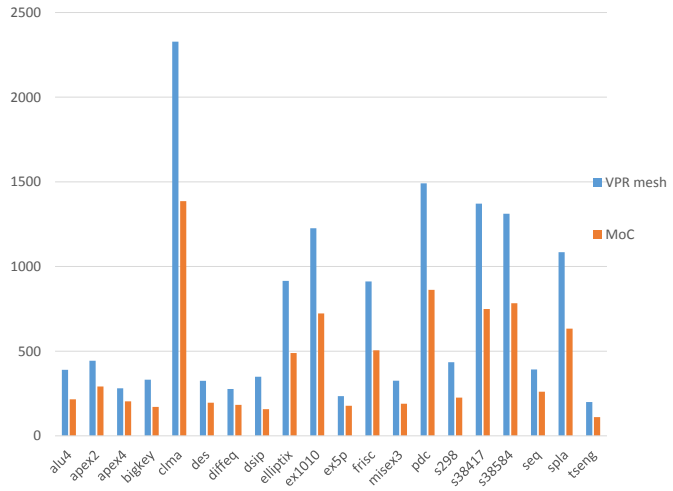


FIGURE 3.26 – Variation du nombre de switchs utilisés

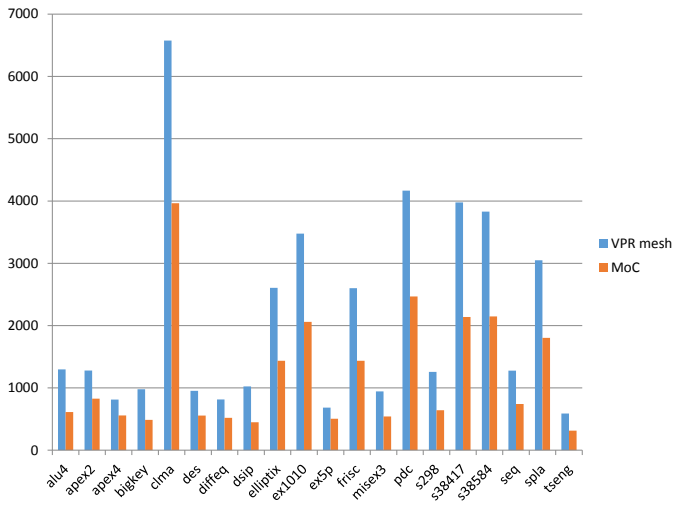


FIGURE 3.27 – Variation de la surface

Nous avons ensuite cherché les architectures mesh et MOC capablent d'implémenter l'ensemble des benchs. Cela nécessite un FPGA de taille 33x33 pour une architecture matricielle et 36x36 pour notre architecture Mesh of Clusters. A partir de là, nous avons cherché la taille de canal de routage minimale nous permettant de router l'ensemble des benchs. Dans le tableau 3.4, on peut voir que le canal de routage minimal pour implémenter l'ensemble des benchs est de respectivement 84 pour une architecture matricielle et 46 pour le Mesh of Clusters. Enfin, on constate une réduction de 45% de la surface lorsqu'on utilise une architecture Mesh of Clusters. En conclusion, l'architecture Mesh of Clusters nous permet de diminuer de 42% la surface du circuit pour l'utilisation des 20 benchs MCNC. Si on compare ces résultats à ceux de l'architecture hiérarchique, cela correspond à une augmentation de 14% de la surface. Nous avons donc proposé une architecture capable de combiner les avantages d'une architecture matricielle de par sa généricité avec les avantages d'une architecture hiérarchique à savoir une réduction de la surface et du taux d'utilisation des interconnexions.

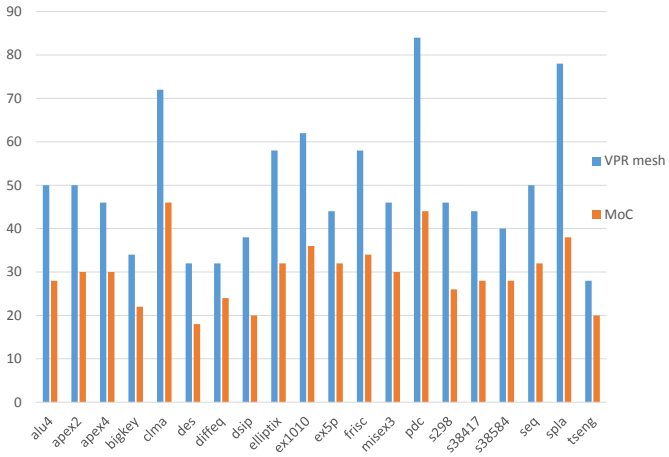


FIGURE 3.28 – Variation du CW permettant de router l'ensemble des benchs

Benchmarks MCNC	VPR architecture Mesh 33x33	Mesh of Clusters 36x36
alu4	50	28
apex2	50	30
apex4	46	30
bigkey	34	22
clma	72	46
des	32	18
diffeq	32	24
dsip	38	20
elliptic	58	32
ex1010	62	36
ex5p	44	32
frisc	58	34
misex3	46	30
pdc	84	44
s298	46	26
s38417	44	28
S38584	40	28
seq	50	32
spla	78	38
tseng	28	20
CW max	84	46
Nb switchs max	2608	1382
Surface max	7281	3954

TABLE 3.4 – CW minimal permettant de router l'ensemble des benchs

6 Conclusion

Ce chapitre a présenté dans un premier temps les particularités de l'architecture de FPGA *Mesh of Custers* utilisée dans cette thèse. Elle combine les avantages des architecture matricielle (généricité) et hiérarchique (gain de surface de 40%). Dans un deuxième temps, nous avons présenté les différentes étapes du flot de conception et de la mise en place d'un générateur de FPGA totalement paramétrable utilisant l'environnement du LIP6 Stratus et le langage Python. Le flot de configuration est ensuite détaillé ainsi que les algorithmes utilisés pour la configuration du FPGA Mesh of Clusters comme le placement recuit-simulé ou l'algorithme de routage Pathfinder. Enfin, une comparaison des performances de notre architecture comparée à l'architecture matricielle est présentée et l'on peut constater que l'architecture Mesh of Clusters permet de réduire de 42% la surface du circuit. Dans les chapitres suivants, nous nous intéresserons à la criticité des blocs composant notre FPGA, à l'ajout de mécanismes de contournement des défauts, aux changements que cela induit dans l'architecture ainsi qu'à leurs performances.

Chapitre 4

Criticité et tolérance aux défauts

1 Introduction

Avec l'accroissement de la densité d'intégration, on constate une réduction importante du rendement de fabrication des circuits. Notre défi est de proposer une architecture de FPGA capable de tolérer des défauts présents en son sein. Nous avons retenu dans l'état de l'art les bons résultats des techniques suggérant l'ajout de redondance matérielle à l'architecture du FPGA. Ces techniques peuvent se décomposer en 2 sous parties en fonction de leur niveau de granularité. Les techniques à gros grain ajoutant de la redondance sous forme de lignes et de colonnes supplémentaires à l'architecture du FPGA et les techniques à grain fin qui appliquent de la redondance sur les éléments de base du FPGA comme les multiplexeurs. Dans ce chapitre, nous allons d'abord proposer une méthode permettant de calculer la criticité des blocs composant notre FPGA Mesh of Clusters. Cette méthode va nous permettre d'identifier les éléments les plus critiques de notre architecture. Ensuite, nous détaillerons les diverses techniques de redondance retenues et proposées, leurs impacts sur l'architecture des blocs composant notre FPGA et leurs impacts sur la criticité de ces blocs.

2 Evaluation de la criticité

L'objectif dans cette section est de proposer une méthode permettant d'évaluer la criticité des éléments composant notre FPGA quel que soit l'architecture choisie.

2.1 Définition

Le FPGA est représenté par un graphe $G = (N, A)$ tel que :

1. l'ensemble des multiplexeurs à l'intérieur des clusters et des Switch boxes représente l'ensemble des nœuds N du graphe G.
2. L'ensemble des signaux connectant les multiplexeurs représente l'ensemble des arcs A tel que $(x, y) \in A$.

Chaque cluster et Switch boxe peut être représenté par un sous-graphe (illustré figure 4.1 pour le cluster). La criticité $\mathcal{C}$ de chaque nœud composant le cluster dépend

du nombre de CLBs connectés à ce nœud comparé au nombre de chemins disponibles pour atteindre ce nœud.

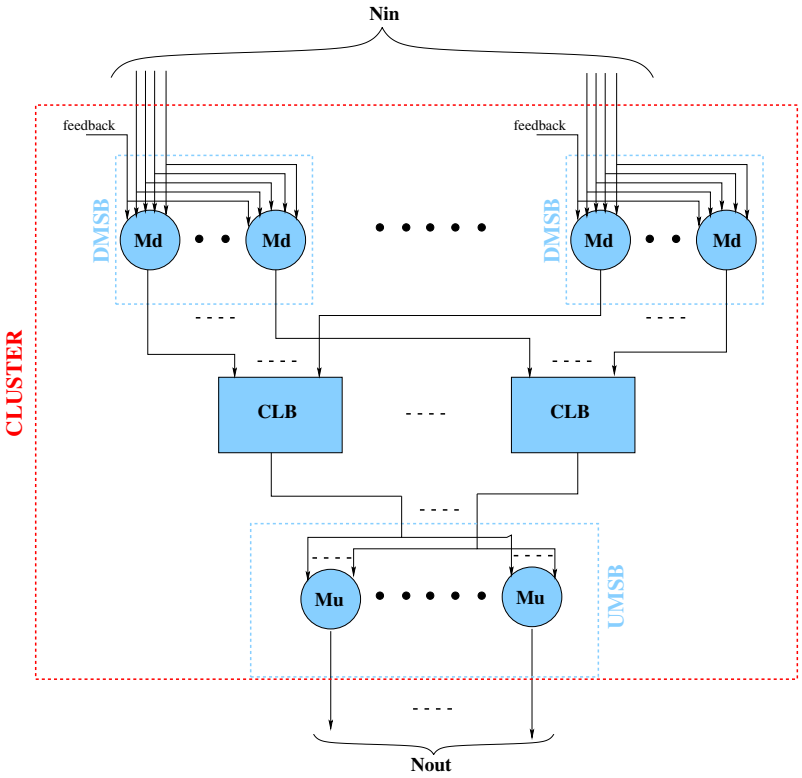


FIGURE 4.1 – Ensemble des noeuds du cluster

Soit $C \subset N$ l'ensemble des multiplexeurs composant le cluster et $D_c \subset C$ (resp $U_c \subset C$) l'ensemble des multiplexeurs Md (respectivement Mu) composant chaque MBSs (respectivement UMSB et DMSB). La criticité d'un MSB (DMSB et UMSB) dépendra alors du nombre de CLBs que connecte ce MSB et du nombre d'entrées de celui-ci. Alors la formule permettant de calculer la criticité $\mathcal{C}$ est :

$$\forall\, x \in D_c, \quad \mathcal{C}(x) = \frac{1}{d_+(x) * d_-(x)} \tag{4.1}$$

$$\forall\, x \in U_c, \quad \mathcal{C}(x) = \frac{N_{clb}}{d_+(x) * d_-(x)} \tag{4.2}$$

Où N_{clb} représente le nombre de CLBs par cluster, d_+ le nombre de sorties du nœud et d_- son nombre d'entrées. Le nombre d'entrées par cluster est toujours supérieur au nombre de CLBs. De plus, le nombre de CLB est égal au nombre de sorties par cluster. Alors dans un cluster d'une architecture Mesh of Clusters, nous avons :
$\forall\, x \in D_c$ et $\forall\, y \in U_c$, $d_-(x) > d_-(y)$, $N_{in} > N_{clb}$, alors $\mathcal{C}(x) < \mathcal{C}(y)$.
Par conséquent, l'UMSB est plus critique que le DMSB pour un cluster. De la même manière, la criticité de chaque nœud composant une Switch box dépend du nombre d'entrées/sorties de la Switch Box comparé au nombre de chemins disponibles sur le nœud (voir figure 4.2). Soit $S \subset N$ (resp. $D1_s \subset S$, $D2_s \subset S$ et $U_s \subset S$) l'ensemble des multiplexeurs composants les MSBs (resp. DMSB1, DMSB2 et UMSB2). Alors la formule permettant de calculer la criticité $\mathcal{C}$ est :

$$\forall\, x \in D2_s, \quad \mathcal{C}(x) = \frac{N_{out}/CW_{in}}{d_+(x) * d_-(x)} \tag{4.3}$$

$$\forall\, x \in D1_s, \quad \mathcal{C}(x) = \frac{N_{out}/N_{in}}{d_+(x) * d_-(x)} \tag{4.4}$$

$$\forall\, x \in U_s, \quad \mathcal{C}(x) = \frac{N_{in} - N_{out}}{d_+(x) * d_-(x)} \tag{4.5}$$

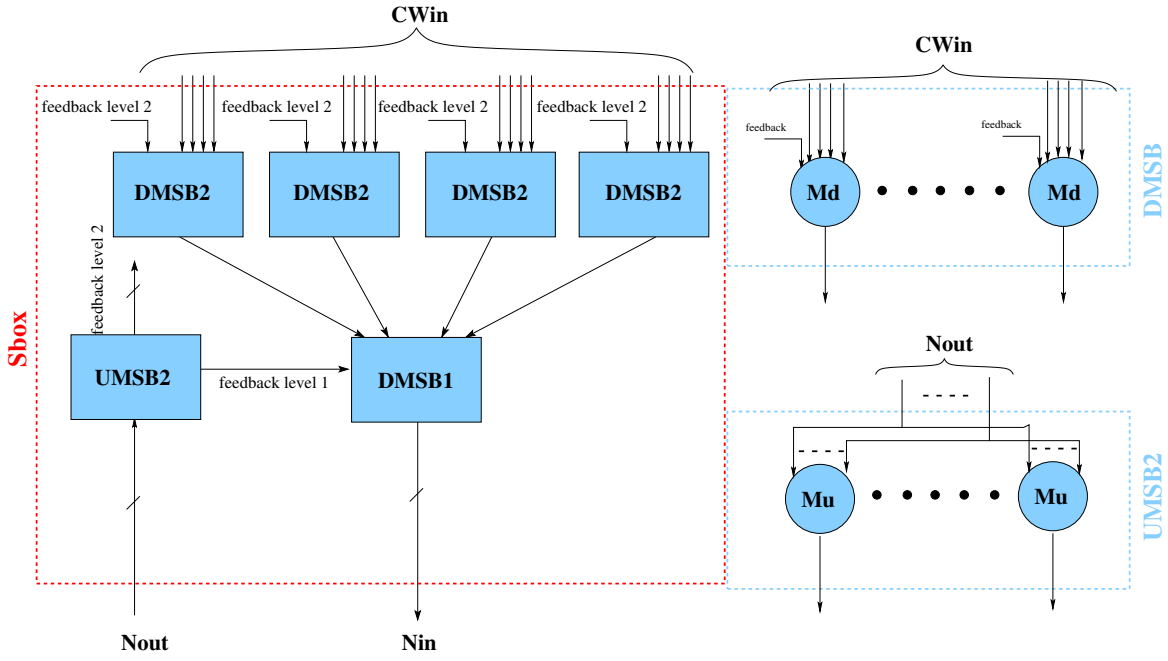


FIGURE 4.2 – Noeuds de la Switch box

Où N_{in} et N_{out} représentent le nombre d'entrées/sorties par cluster et CW_{in} la taille du canal de routage. Dans une Switch box, la différence $N_{in} - N_{out}$ est supérieure au rapport N_{out}/N_{in} et N_{out}/N_{CWin}. De plus, le nombre de connexions sur un DMSB est plus important que sur un UMSB. Donc dans une Switch Box d'une architecture Mesh of Clusters, nous avons :
$\forall\, x \in D_s$ et $\forall\, y \in U_s$, $N_{in} - N_{out} > N_{out}/N_{in}$, $N_{in} - N_{out} > N_{out}/N_{CWin}$, $d_+(x) * d_-(x) > d_+(y) * d_-(y)$, alors $\mathcal{C}(x) < \mathcal{C}(y)$.
C'est pourquoi, les UMSBs sont les éléments les plus critiques d'une Switch box. En conclusion, nous proposons une méthode de calcul de la criticité des blocs composants notre FPGA Mesh of Clusters. Cette méthode est valable quelle que soit l'architecture du FPGA donc quel que soit l'architecture des clusters et des Switch boxes. En effet, nous avons développé un générateur de FPGA Mesh of Clusters, nous devons donc être capable de définir la criticité quel que soit les paramètres architecturaux choisis (nombre de CLBs, nombre d'entrées par CLB etc...). On peut en conclure que les éléments les plus critiques de notre FPGA sont les UMBs à l'intérieur des Sbox et des clusters. Il sera donc intéressant de mesurer l'impact des différentes techniques de contournement sur la criticité de ces blocs et d'orienter notre redondance sur les blocs les plus critiques.

3 Tolérance aux défauts

Dans cette section, nous allons présenter les différentes techniques de redondance permettant de contourner les défauts de fabrication qui ont été retenus. Ces techniques peuvent être classées en 2 catégories, les techniques de redondance à gros grain et à grain fin.

3.1 Redondance à gros grain

Les techniques de redondance à gros grain suggèrent l'ajout de matériel de "secours" sous formes de ressources supplémentaires à l'architecture du FPGA pour remplacer les éléments défectueux. Cela signifie la plupart du temps l'ajout de lignes et colonnes supplémentaires sur l'architecture du circuit. Ces techniques ayant un impact trop peu significatif sur la tolérance aux défauts (1 ligne ou 1 colonne supplémentaire ne permet de corriger qu'un seul défaut) comparé au surcoût en termes de surface, nous proposons des techniques de redondance à gros grain se basant sur l'ajout de connexions entre les différents éléments qui composent notre FPGA (cluster et Switch box).

Connexions diagonales

– Impact sur l'architecture :

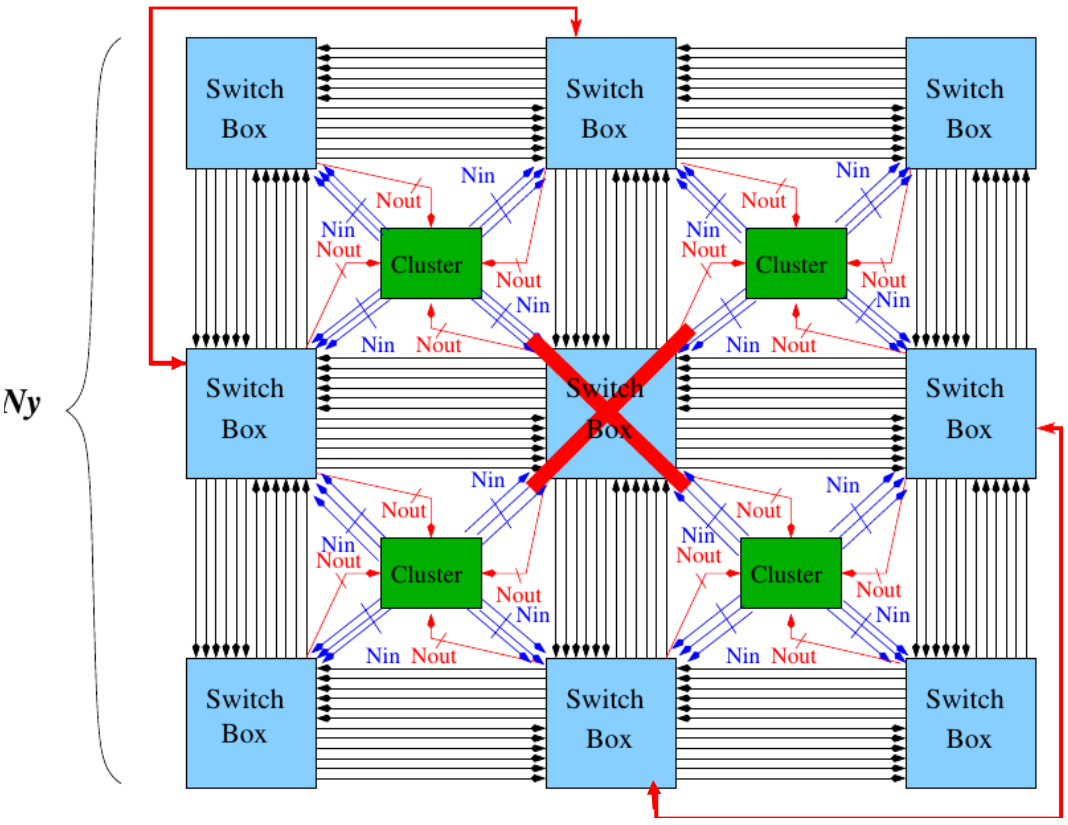


FIGURE 4.3 – Technique d'ajouts de connexions diagonales

La première technique consiste à ajouter des connexions supplémentaires au canal de routage (illustrée figure 4.3). Dans cet exemple si la Switch box centrale est défectueuse, les chemins de routage disponibles nous obligent à la contourner et deviennent ainsi plus long. Cela aura un impact sur la routabilité et sur le délai. En conséquence pour pallier ce problème et augmenter la flexibilité de l'architecture lors du routage, chaque Switch box sera connectée à une Switch box en diagonale. Chaque Switch Box(i,j) de la matrice sera alors connectée à la Switch box(i+1, j+1) ainsi qu'à la Switch box(i-1, j-1). Le nombre de connexions ajoutées entre chaque Switch box est paramétrable ce qui permet de les intégrer directement dans le générateur de FPGA. L'ajout de connexions sur le canal de routage, entraîne une augmentation du nombre d'entrées par Switch box et donc une augmentation en termes de surface de celles-ci. C'est pourquoi le nombre de connexions à ajouter est paramétrable, ce qui permet à l'utilisateur de fixer ses propres limites en termes de tolérance aux défauts et d'augmentation de surface. La figure 4.4 montre que ces entrées supplémentaires nécessitent des DMSB de niveau 2 supplémentaires dans les Switch boxes pour être routé. Par exemple, si on choisit d'ajouter à notre architecture n connexions diagonales représentant 10% de la taille du canal de routage, alors le nombre de DMSB de niveau 2 à l'intérieur des Switch boxes augmentera de 10%.

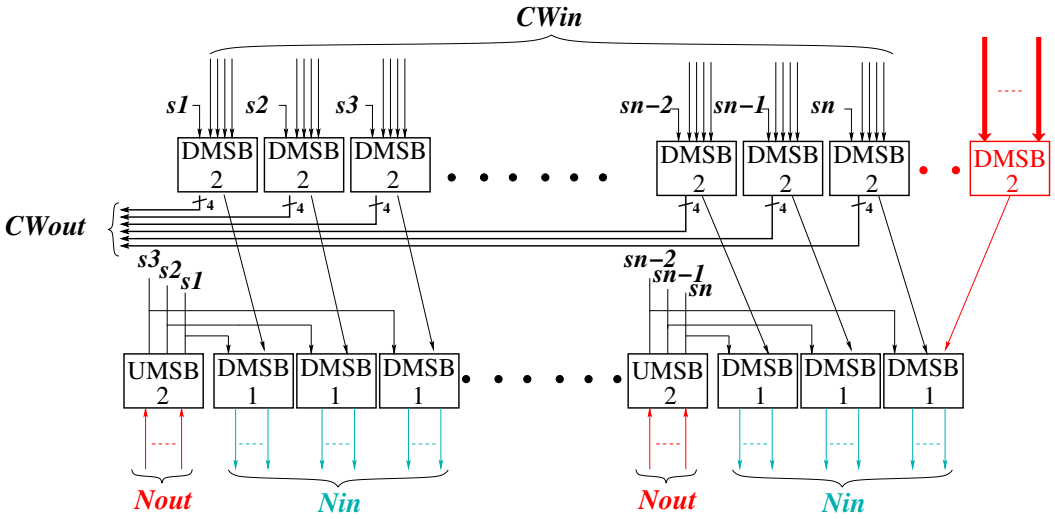


FIGURE 4.4 – Technique d'ajout de connexions diagonales

– Impact sur la criticité :

Le fait d'augmenter le nombre d'entrées par Switch box (Fig 4.5) va avoir un impact non négligeable sur la criticité des blocs composant la Switch Box. Avec les connexions diagonales, la taille du canal de routage et le nombre de DMSB de niveau 2 dans les Switch boxes augmentent. On a donc plus de nœuds au niveau 2 et des entrées supplémentaires au niveau 1. La criticité de ces 2 blocs dépend donc du nombre de connexions diagonales ajoutées N_{diag} et du nombre de connexions supplémentaires sur les DMSB niveau 1 $Nc_{diag} = \frac{N_{diag}}{4}$ car on a 4 entrées du CR par DMSB de niveau 2. La criticité devient alors :

$$\forall\, x \in D2_s, \mathcal{C}_{Diag}(x) = \frac{N_{out}/(CW_{in} + N_{diag})}{d_+(x) * d_-(x)} = \frac{\mathcal{C}(x)}{1 + \frac{N_{diag}}{CW_{in}}} \tag{4.6}$$

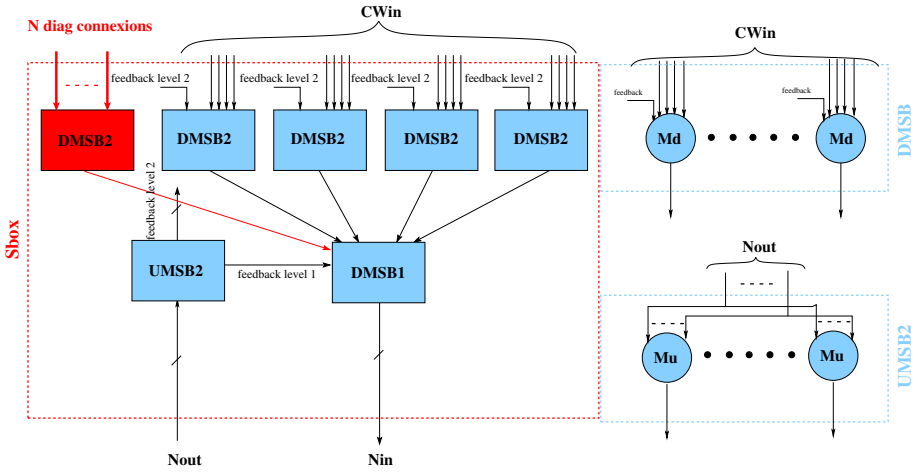


FIGURE 4.5 – Impact des connexions diagonales sur la criticité

$$\forall\, x \in D1_s, \mathcal{C}_{Diag}(x) = \frac{N_{out}/N_{in}}{d_+(x) * (d_-(x) + Nc_{diag})} = \frac{\mathcal{C}(x)}{1 + \frac{Nc_{diag}}{CW_{in}}} \tag{4.7}$$

Longues connexions

– Impact sur l'architecture :

Dans l'architecture de base de notre FPGA, les fils de connexions entre les Switch boxes sont de longueur 1. Si les Switch boxes centrales sont défectueuses, on perd la possibilité de se connecter à certaines Switch boxes ou alors on augmente aussi fortement le nombre de switchs traversés et le délai. C'est pourquoi, la seconde technique consiste à augmenter de nouveau la taille du canal de routage pour cette fois-ci ajouter des fils de connexions de longueurs 2 entres les Switch boxes (illustrée figure 4.6). Chaque Switch Box(i,j) aura donc n connexions de longueurs 2 permettant de se connecter aux Switch boxes(i+2, j) et (i, j+2). Le nombre de connexions ajoutées entre chaque Switch box est de nouveau paramétrable pour pouvoir les intégrer directement dans le générateur de FPGA. L'ajout de connexions sur le canal de routage, entraîne comme pour l'ajout de connexions diagonales une augmentation du nombre d'entrées par Switch box et donc une augmentation en termes de surface de celles-ci. Il est donc nécessaire de fixer une limite entre le nombre de défauts que l'on veut pouvoir tolérer et l'augmentation de surface qui en découlera. L'impact de cette technique sur l'architecture des Switch boxes est la même que la technique précédente (figure 4.4), avec n longues connexions supplémentaires représentant 10% de la taille du canal de routage, alors le nombre de DMSB de niveau 2 à l'intérieur des Switch boxes augmentera de 10%.

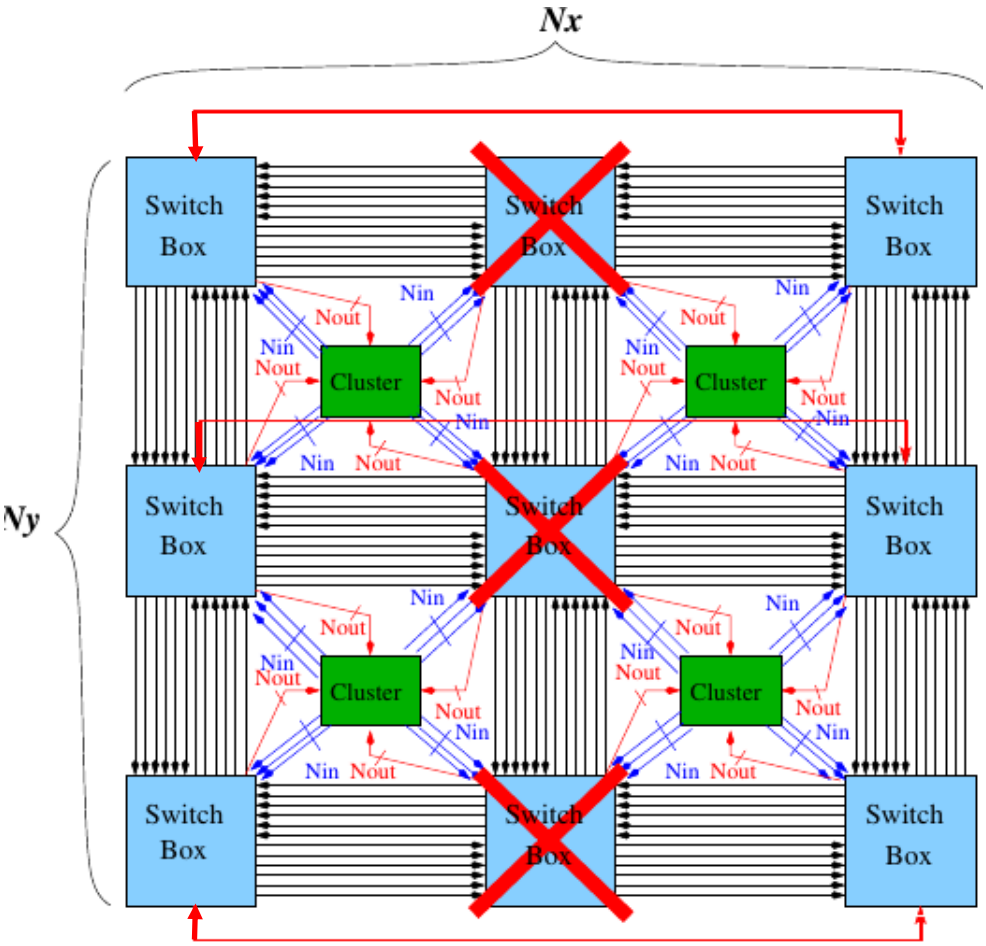


FIGURE 4.6 – Technique d'ajouts de longues connexions

– Impact sur la criticité :

L'impact de cette technique sur la criticité sera en théorie le même qu'avec la technique précédente (Fig 4.5). Avec de longues connexions, la taille du canal de routage et le nombre de DMSB de niveau 2 dans les Switch boxes augmentent. On a donc plus de nœuds au niveau 2 et des entrées supplémentaires au niveau 1. La criticité de ces 2 blocs dépend donc du nombre de longues connexions ajoutées N_{Long} et du nombre de connexions supplémentaires sur les DMSB niveau 1 $Nc_{Long} = \frac{N_{Long}}{4}$ car on a 4 entrées du CR par DMSB de niveau 2 La criticité devient alors :

$$\forall\, x \in D2_s, \mathcal{C}_{Long}(x) = \frac{N_{out}/(CW_{in} + N_{Long})}{d_+(x) * d_-(x)} = \frac{\mathcal{C}(x)}{1 + \frac{N_{Long}}{CW_{in}}} \tag{4.8}$$

$$\forall\, x \in D1_s, \mathcal{C}_{Long}(x) = \frac{N_{out}/N_{in}}{d_+(x) * (d_-(x) + Nc_{Long})} = \frac{\mathcal{C}(x)}{1 + \frac{Nc_{Long}}{CW_{in}}} \tag{4.9}$$

Connexions directes entre clusters

– Impact sur l'architecture :

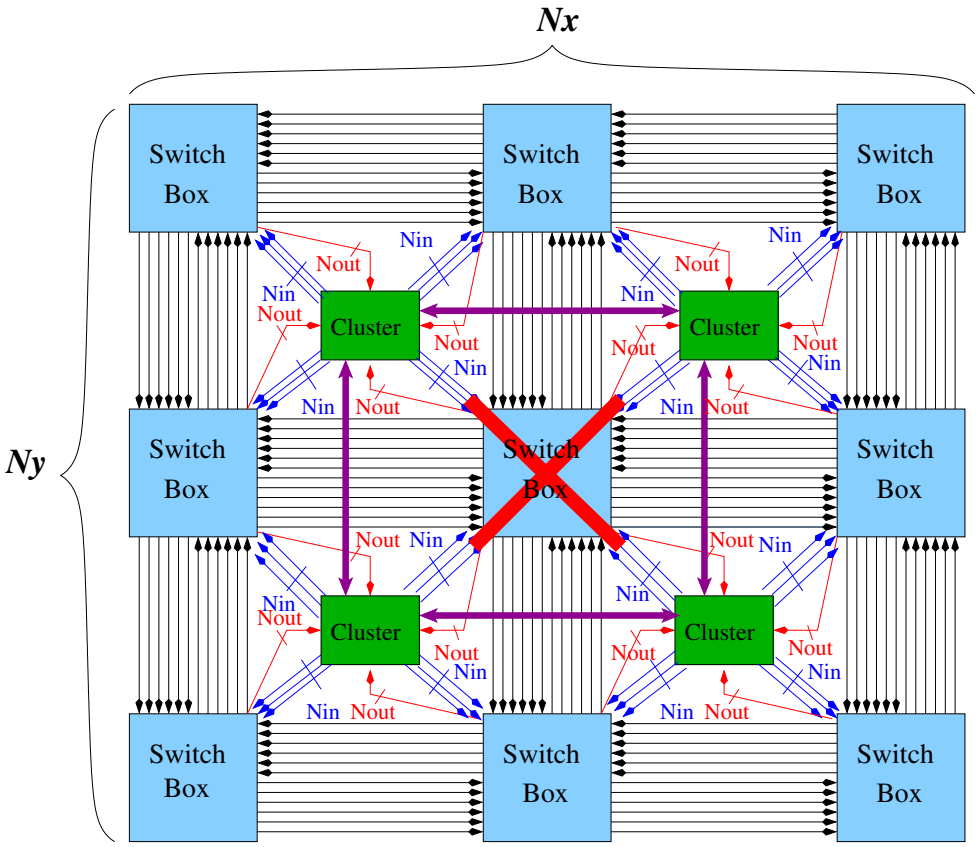


FIGURE 4.7 – Technique d'ajouts de connexion entre clusters

La dernière technique à gros grain consiste à ajouter des connexions directes entre tous les clusters du FPGA(voir figure 4.7). Chaque cluster (i,j) possède des entrées/sorties supplémentaires avec ses plus proches voisins (i+1, j)(i, j+1)(i-1, j)(i, j-1). Ces connexions permettent d'éviter une Switch Boxe au cas où celle-ci serait défectueuse et de réduire aussi le nombre de switchs que va traverser le signal. Le nombre de connexions ajoutées entre chaque cluster est paramétrable dans l'optique d'intégrer cette technique dans le générateur de FPGA. L'ajout de connexions entre clusters, entraîne une augmentation du nombre d'entrées par cluster et donc une augmentation en termes de surface. La figure 4.8 montre que ces entrées supplémentaires augmentent le nombre d'entrées par DMSB des clusters. Ces DMSBs étant composés de multiplexeurs, on va augmenter le nombre de multiplexeurs par cluster. Par exemple, si on choisit d'ajouter à notre architecture m entrées et n sorties entre clusters, alors le nombre d'entrée par DMSB augmentera de $\frac{m}{Nb\_dmsb}$ et le nombre de sortie de n.

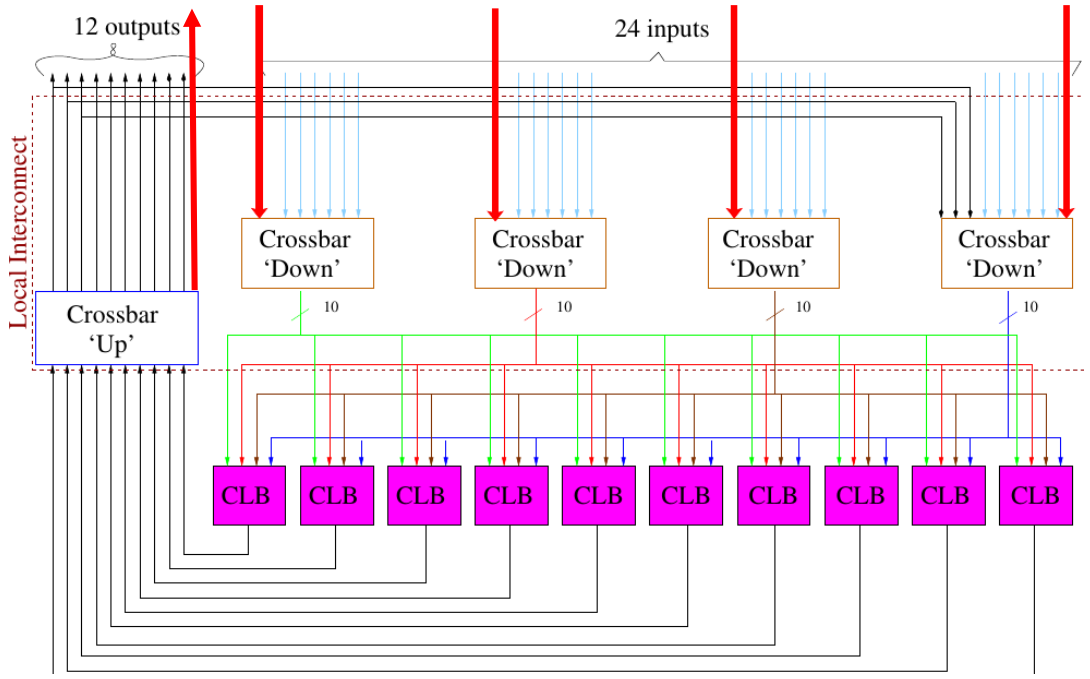


FIGURE 4.8 – Impact sur l'architecture du cluster

– Impact sur la criticité :

Le fait d'augmenter le nombre d'entrées/sorties par cluster (figure 4.8) va avoir un impact sur la criticité des DMSB et de l'UMSB. La criticité des DMSB dépend donc du nombre d'entrées supplémentaires ajoutées m au cluster (figure 4.9) et la criticité de l'UMSB du nombre de sorties ajoutées n. La criticité devient alors :

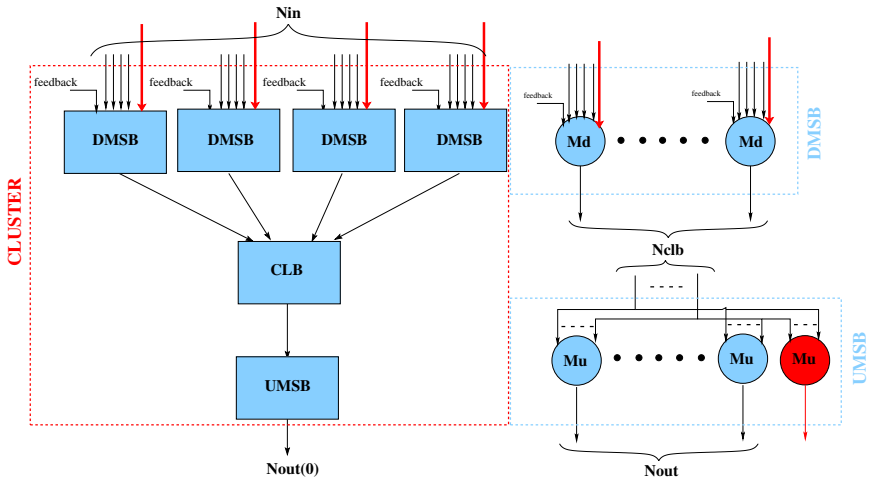


FIGURE 4.9 – Impact des connexions directes entre clusters sur la criticité

$$\forall x \in D_c, \mathcal{C}_{Dir}(x) = \frac{1}{d_+(x) * (d_-(x) + m)} = \frac{\mathcal{C}(x)}{1 + \frac{m}{N_{in}}} \tag{4.10}$$

$$\forall x \in U_c, \mathcal{C}_{Dir}(x) = \frac{N_{clb}}{d_+(x) * (d_-(x) + n)} = \frac{\mathcal{C}(x)}{1 + \frac{n}{N_{out}}} \tag{4.11}$$

3.2 Redondance à grain fin

Les techniques de redondance à grain fin ajoutent de la redondance matérielle au niveau des blocs de base du FPGA. Cela permet de tolérer plus de défauts mais l'impact sur la surface et le délai du FPGA est aussi plus important. Nous allons donc proposer différentes techniques permettant de minimiser ce surcoût en termes de surface et de délai tout en tolérant un maximum de défauts.

Fine Grain redundancy (FGR)

– Impact sur l'architecture :

Dans l'architecture de base d'un cluster ou d'une Switch box, si un signal est routé par un multiplexeur défectueux, la connexion entre les entrées et la sortie du MSB ainsi que la connexion vers l'élément suivant (un CLB, un autre MSB...) n'est plus possible. On peut donc perdre la possibilité de se connecter à certains éléments logiques. C'est pourquoi une technique appelée Fine Grain redundancy (FGR) a été proposée [4] pour contourner les défauts à l'intérieur des boîtes de connexions et des Switchs boxes d'une architecture Mesh. Cette technique peut être appliquée à l'architecture de nos MSBs, cela consiste à ajouter 2 étages de multiplexeurs en entrée de nos MSB pour pouvoir contourner le multiplexeur défectueux en utilisant son plus proche voisin et 2 étages en sortie pour restaurer la sortie du MSB qui était utilisée par l'élément défectueux (illustrée figure 4.10).

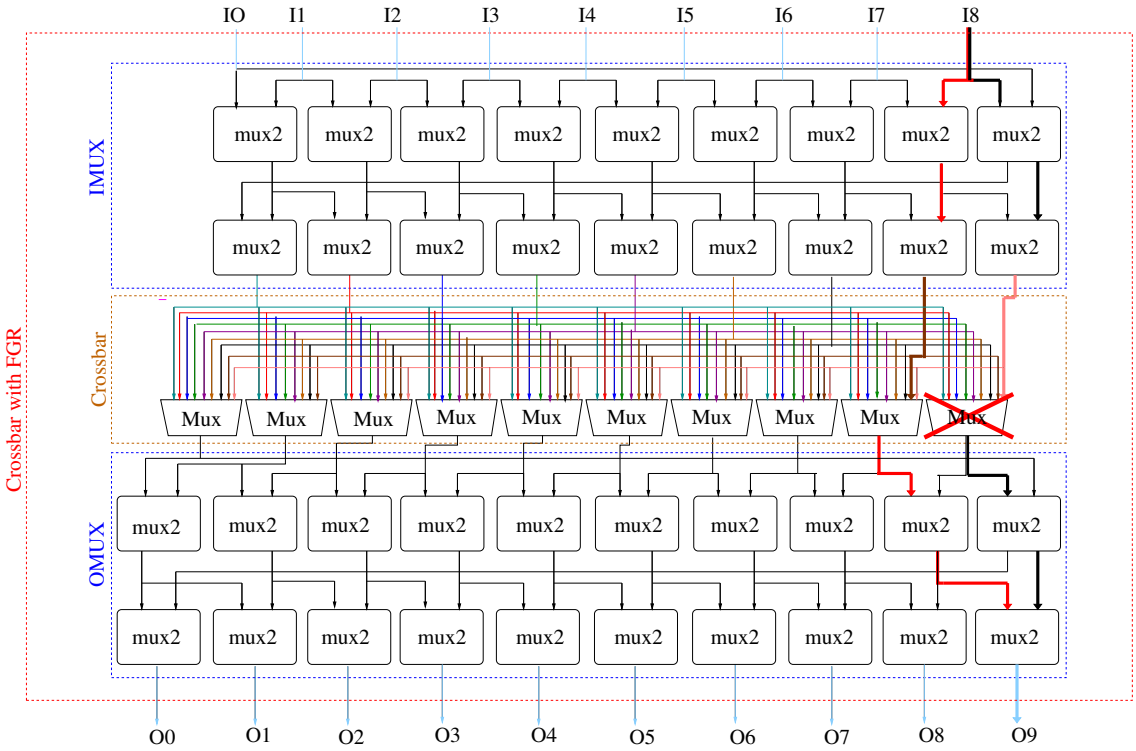


FIGURE 4.10 – Technique FGR

Par exemple, dans l'architecture initiale, le dernier multiplexeur est utilisé pour connecter l'entrée $I8$ à la sortie $O9$. Si le multiplexeur est défectueux, la sortie $O9$ ne pourra pas être connectée au CLB. Grâce à la FGR, les deux premiers étages de

multiplexeurs (IMUX) re-routent le signal provenant de I8 sur le multiplexeur voisin et les deux étages en sortie (OMUX) re-routent le signal vers la sortie du MSB $O9$. Donc, avec cette technique, le nombre de chemins possibles pour router un même signal est doublé. Cependant, le signal provenant de I8 ne peut être re-routé qu'à la seule condition que son multiplexeur voisin ne soit pas déjà utilisé par un autre signal. Sinon, il y aura un conflit.

– Impact sur la criticité :

Avec la FGR, chaque multiplexeur à l'intérieur des MSB voit son nombre de sorties augmenté. En effet, chaque nœud peut désormais être connecté à trois sorties différentes (figure 4.11). Cela aura un impact sur la criticité des MSB à l'intérieur des clusters et des Switch boxes :

$$\forall\, x \in N, \quad \mathcal{C}_{FGR}(x) = \frac{\mathcal{C}(x)}{3} \tag{4.12}$$

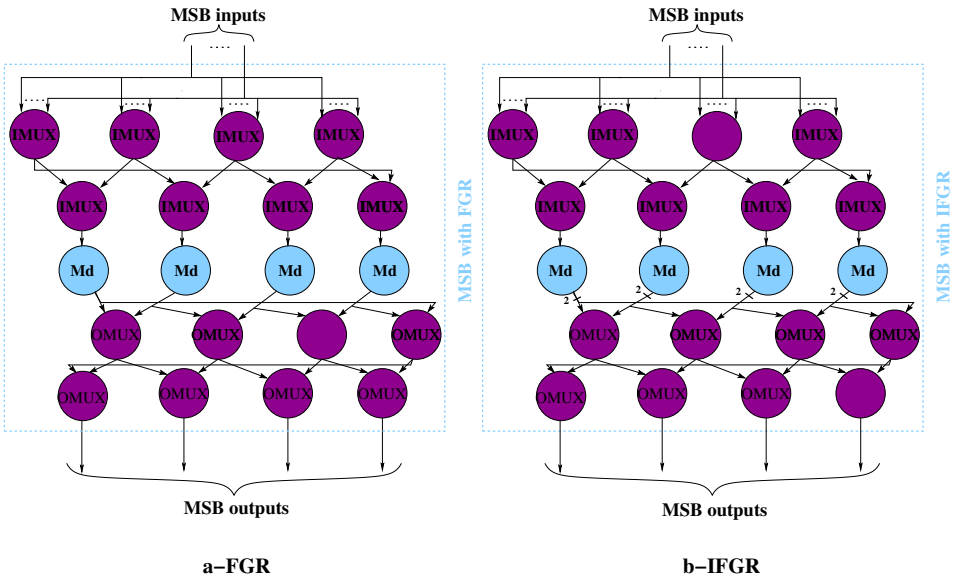


FIGURE 4.11 – Impact sur la criticité de la FGR

Improved FGR (IFGR)

– Impact sur l'architecture :

Pour éviter les conflits qui peuvent survenir avec la FGR, une seconde technique a été étudiée appelée Improved FGR (IFGR) [4]. Cette solution double le nombre de sorties du réseau d'interconnexion. Pour adapter cette technique à notre architecture, il faut doubler le nombre de sorties sur chaque multiplexeur composant nos MSBs. On peut voir dans la figure 4.12, que si l'une des deux sorties d'un multiplexeur est défectueuse, alors nous avons la possibilité d'utiliser l'autre à disposition. De plus, si le multiplexeur voisin est complètement défectueux, le contournement peut être fait même si le plus proche voisin est déjà utilisé pour router un autre signal. En effet, la deuxième sortie permettra d'éviter les conflits. En conclusion, cette méthode permet de tolérer 2 fois plus de défauts que la FGR mais cela implique de doubler le nombre de sorties par multiplexeurs et donc de fortement augmenter la surface du FPGA.

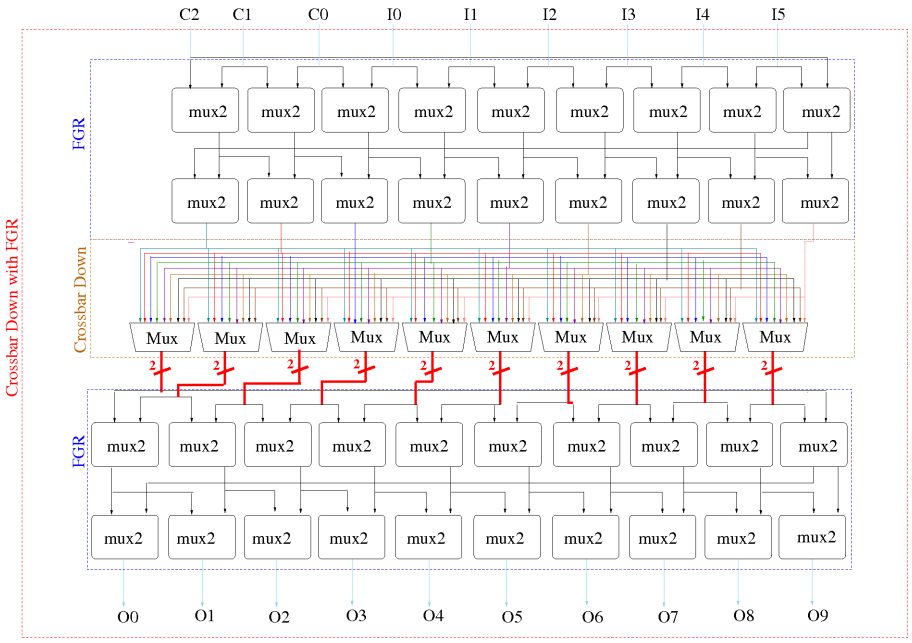


FIGURE 4.12 – Technique IFGR

– Impact sur la criticité :

Pour adapter cette technique à notre architecture, le nombre de sorties de chaque MSB est doublé. Donc le nombre de sorties par nœud est doublé (voir figure 4.13) et la criticité est quant à elle divisée par 2 comparé à la FGR.

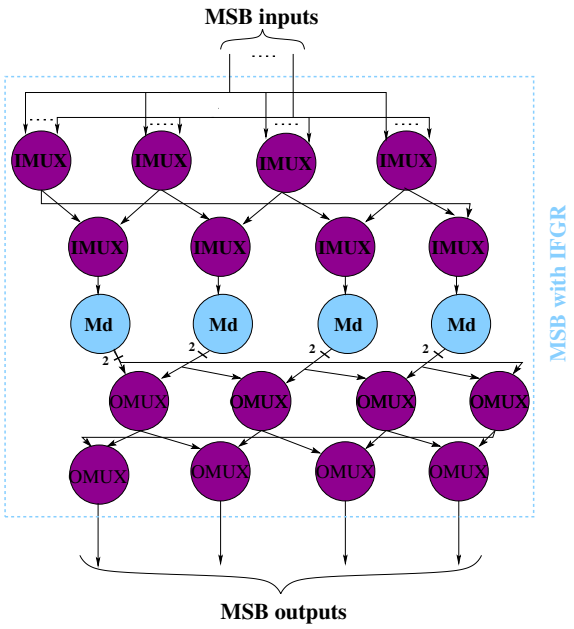


FIGURE 4.13 – Impact de l'IFGR sur la criticité

$$\forall x \in N, \quad \mathcal{C}_{IFGR}(x) = \frac{\mathcal{C}_{FGR}(x)}{2} = \frac{\mathcal{C}(x)}{6} \tag{4.13}$$

Adapted FGR (AFGR)

– Impact sur l'architecture :

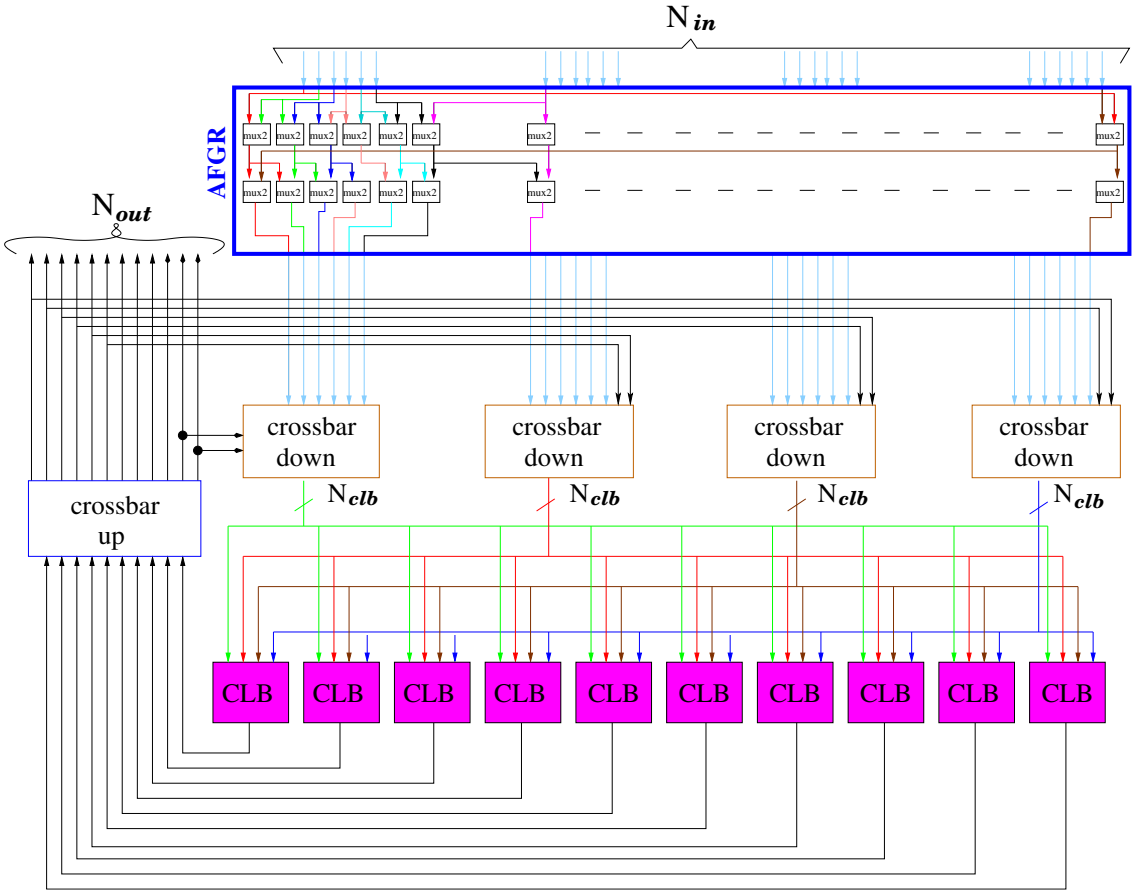


FIGURE 4.14 – Technique AFGR sur le cluster

Le principal désavantage des techniques comme la FGR et l'IFGR est leur impact sur la surface du FPGA. Pour réduire l'augmentation de surface, nous proposons une nouvelle technique appelée Adapted Fine Grain Redundancy (AFGR). Dans l'architecture de base de notre FPGA, tous les DMSB du cluster sont connectés à l'ensemble des CLBs. Pour réduire l'augmentation de surface, on peut donc envisager de supprimer les étages de restauration de la FGR. Puis, pour augmenter la routabilité de notre architecture, les deux premiers étages qui servent au contournement des défauts seront commun à tous les DMSB. Avec cette technique, nous avons la possibilité de contourner un défaut présent dans un DMSB avec un multiplexeur se trouvant dans le DMSB voisin (voir figure 4.14). Cependant, comme la FGR, si le multiplexeur voisin est déjà utilisé pour router un autre signal, le contournement ne sera pas possible.

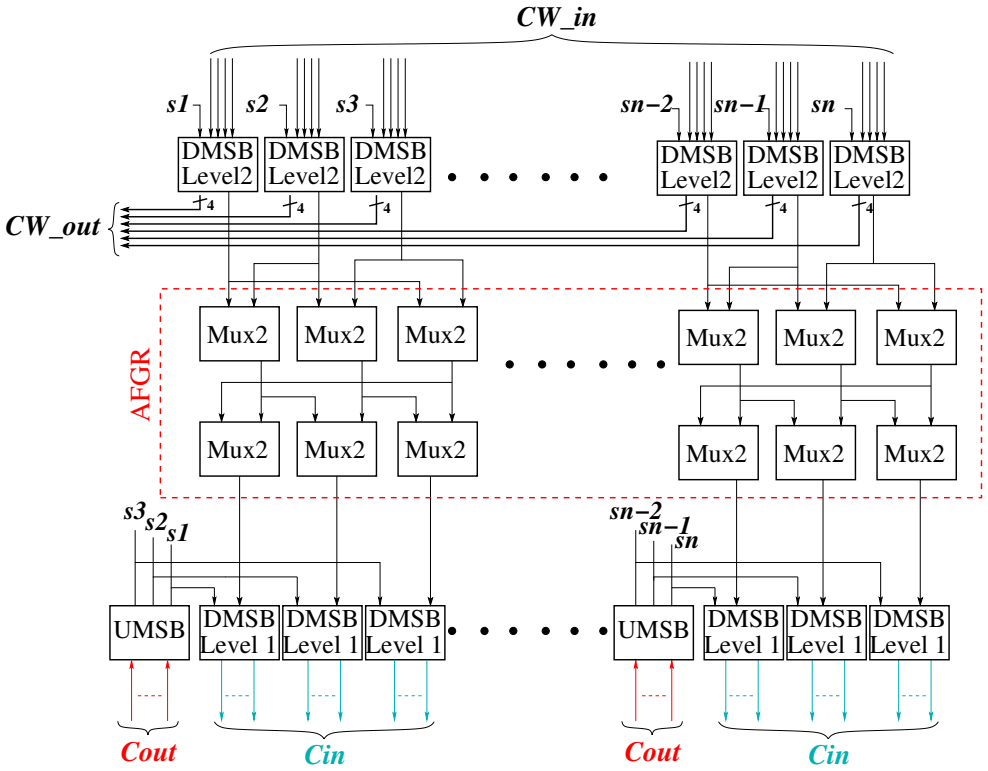


FIGURE 4.15 – Technique AFGR sur la Switch box

– Impact sur la criticité :

Avec la technique AFGR, le nombre d'entrées par nœud augmente pour chaque MSBs(Fig.4.16). En effet, chaque nœud peut désormais utiliser 2 entrées supplémentaires appartenant à ses voisins. La criticité de chaque nœud à l'intérieur des MSBs est donc :

$$\forall\, x \in N, \quad \mathcal{C}_{AFGR}(x) = \frac{\mathcal{C}(x)}{(d_{-}(x)+2)/d_{-}(x)} \tag{4.14}$$

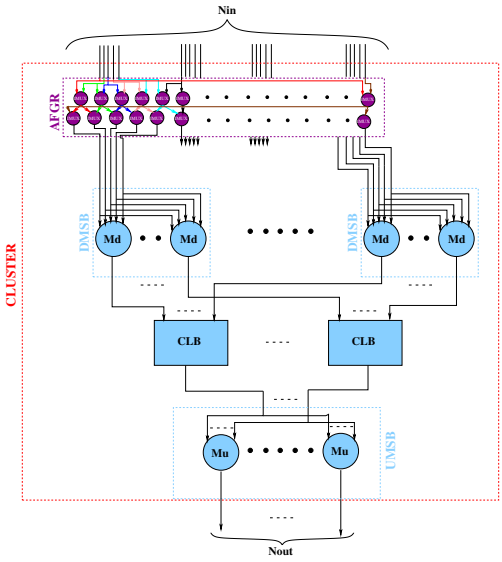


FIGURE 4.16 – Cluster with AFGR on dmsb

Distribution des Feedbacks (DF)

– Impact sur l'architecture :

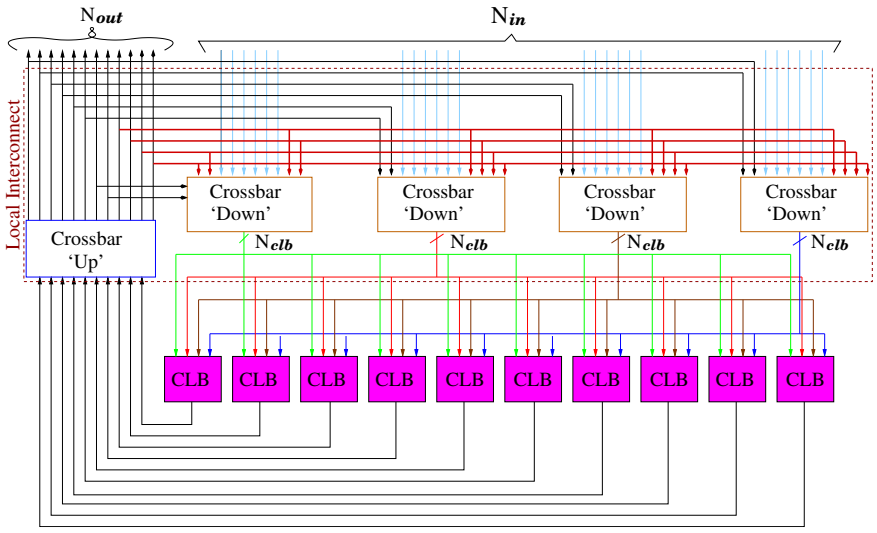


FIGURE 4.17 – Technique DF sur le cluster

Dans l'architecture MoC, les UMSBs sont connectés aux DMSBs à l'aide de feedbacks. Le nombre de feedbacks par DMSB est calculé en fonction du nombre de feedbacks et du nombre de DMSB :

$$Nb_{feedbacks\_par\_dmsb} = \frac{Nb_{DMSB}}{Nb_{feedbacks}} \tag{4.15}$$

Pour réduire l'augmentation de surface, nous proposons une nouvelle technique ciblant uniquement les DMSBs appelée Distributed Feedbacks (DF). Dans l'architecture de base de notre FPGA, les feedbacks sont répartis de manière homogène sur l'ensemble des DMSBs (cluster et Switch box). Pour augmenter la routabilité et

la tolérance aux défauts de l'architecture, nous proposons de distribuer un nombre n de feedbacks (voir figure 4.17 et 4.18). Si un défaut est présent dans un multiplexeur d'un DMSB utilisé par un feedback, alors grâce à la DF, le signal peut être re-routé vers un autre DMSB.

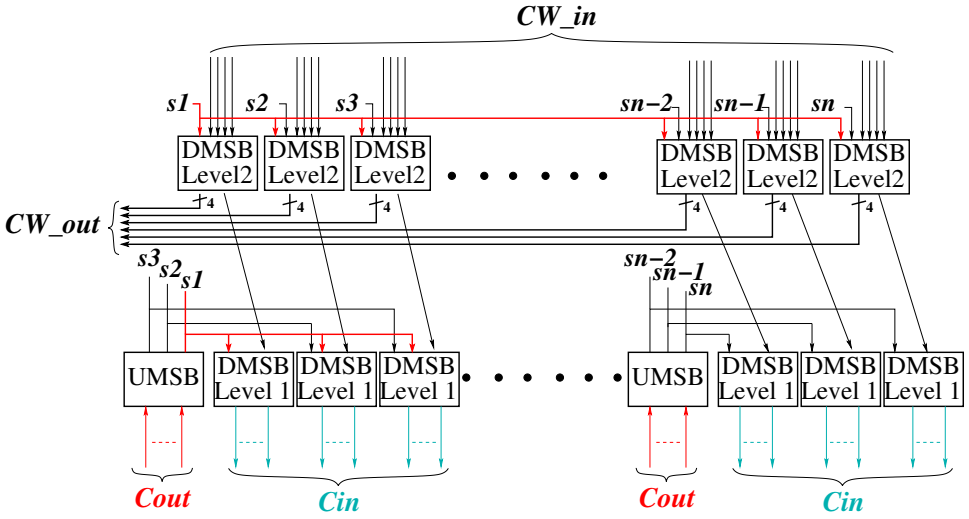


FIGURE 4.18 – Technique DF sur la Switch box

– Impact sur la criticité :

Avec la distribution des feedbacks, le nombre d'entrées par DMSB et par noeud compris dans les DMSBs augmente (figure 4.19). La criticité de chaque noeud va donc dépendre du nombre de feedbacks distribués sur chaque DMSB (N_{df}).

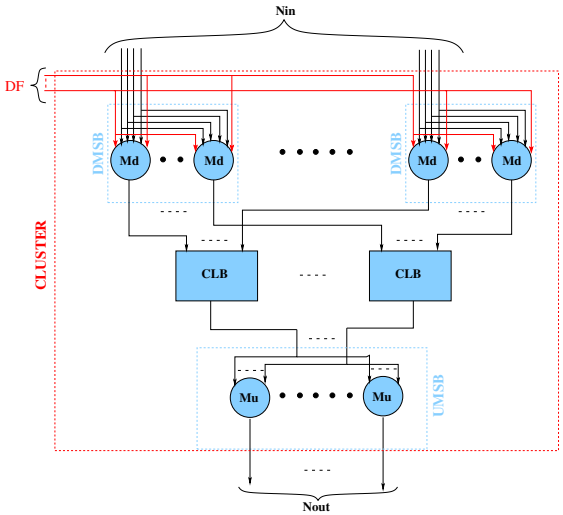


FIGURE 4.19 – Cluster with DF

$$\forall\, x \in D_c \cup D1_s \cup D2_s, \mathcal{C}_{DF}(x) = \frac{\mathcal{C}(x)}{(d_-(x) + N_{df})/d_-(x)} \tag{4.16}$$

Upward Redundant Multiplexers (URM)

– Impact sur l'architecture :

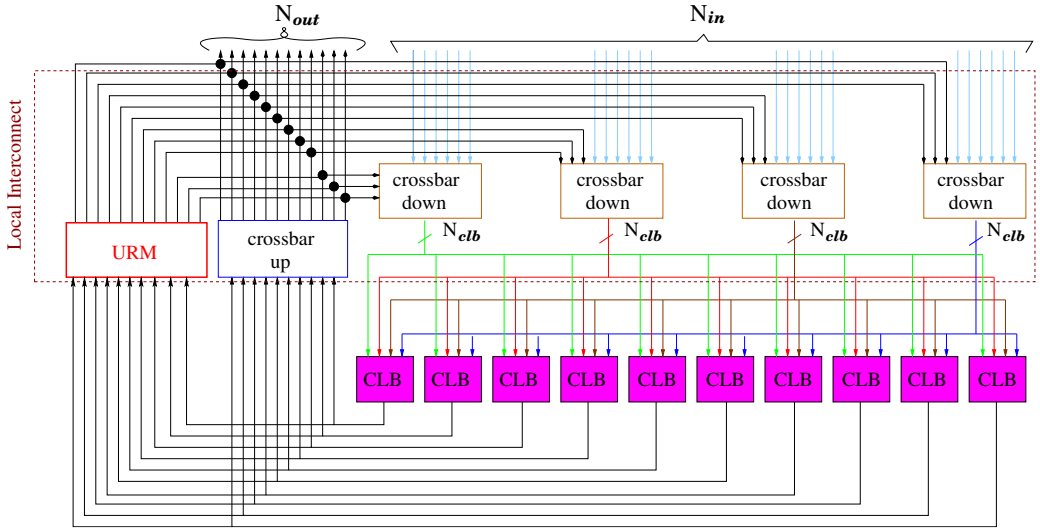


FIGURE 4.20 – Technique URM sur le cluster

Les UMSBs sont des éléments essentiels de notre architecture. En effet, ils sont utilisés pour connecter les sorties des blocs logiques aux sorties des clusters, aux Switch boxes et au canal de routage du FPGA. Dans le but de réduire l'impact des techniques de contournement sur la surface et sur le délai du FPGA, nous proposons une nouvelle technique ciblant uniquement les UMSBs appelée Upward Redundant Multiplexers (URM). Des multiplexeurs sont ajoutés en parallèles aux UMSBs (voir figure 4.20 et 4.21). Dans le cas où un ou plusieurs défauts surviendraient dans un UMSB, les multiplexeurs de l'URM seraient utilisés pour remplacer les défectueux et maintenir la fonctionnalité du circuit.

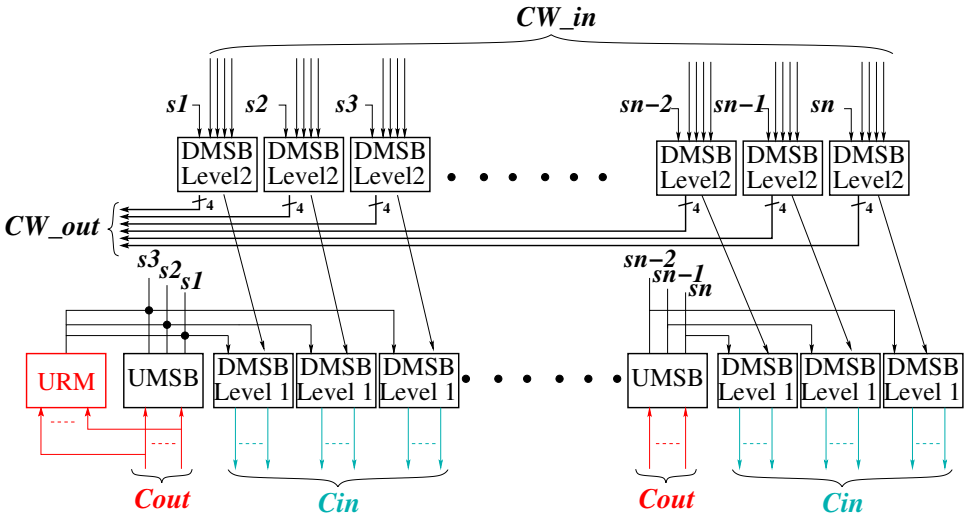


FIGURE 4.21 – Technique URM sur la Switch box

– Impact sur la criticité :

La technique URM est appliquée uniquement sur les UMSB des clusters et des Switch boxes donc elle aura un impact sur la criticité des nœuds compris dans les UMSBs seulement (voir figure 4.22). Comme pour la distribution des feedbacks la valeur de la criticité va dépendre du nombre d'URM (N_{urm}) ajouté à l'architecture des UMSBs.

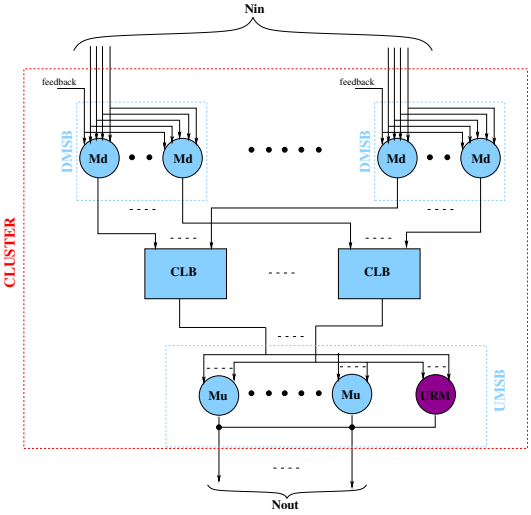


FIGURE 4.22 – Cluster with URM

$$\forall\, x \in U_c \cup U_s, \quad \mathcal{C}_{URM}(x) = \frac{\mathcal{C}(x)}{1 + N_{URM}} \tag{4.17}$$

4 Conclusion

Dans ce chapitre, nous avons proposé une méthode permettant de calculer la criticité des blocs composant un FPGA Mesh of Clusters. Le FPGA est modélisé par un graphe, chaque cluster et chaque Switch box par un sous-graphe dans lequel chaque multiplexeur est représenté par un nœud. La criticité de chaque bloc est défini par le nombre d'éléments logiques et par les paramètres architecturaux. Le tableau ci-dessous montre que les éléments les plus critiques de notre architecture sont les UMSBs présents dans les clusters et dans les Switch boxes. Ensuite, chaque technique de redondance à gros grain puis à grain fin est adaptée à l'architecture Mesh of Clusters. L'impact de chacune de ces techniques sur l'architecture et sur la criticité des blocs est détaillé. Les calculs de criticité ci-dessous ont été faits pour un FPGA de 36x36 avec un canal de routage de 36 (CW), 10 CLBs à 4 entrées par cluster, 28 entrées et 12 sorties par cluster. Dans ce tableau, un 1 représente une ressource très critique et un 0 une ressource non critique. Le tableau 4.1 montre que sans redondance, les UMSBs sont les éléments les plus critiques suivis par les DMSBs du cluster. Pour finir, l'IFGR est la technique qui a l'impact le plus important sur la criticité des DMSBs (-84%) et l'URM sur les UMSBs (-90%).

Techniques de redondance matérielles	**Criticité dans la Switch box**			**Criticité dans le cluster**	
	Mu	Md1	Md2	Mu	Md
sans redondance	1(ref)	0.086(ref)	0.033(ref)	1(ref)	0.1(ref)
Connexions diagonales	0.85(-15%)	0.067(-22%)	0.029(-11%)	1(-0%)	0.1(-0%)
Longues connexions	0.85(-15%)	0.067(-22%)	0.029(-11%)	1(-0%)	0.1(-0%)
Connexions directes	1(-0%)	0.086(-0%)	0.033(-0%)	0.88(-12%)	0.085(-15%)
FGR	0.33(-67%)	0.028(-67%)	0.011(-67%)	0.33(-67%)	0.033(-67%)
IFGR	0.16(-84%)	0.014(-84%)	0.005(-84%)	0.16(-84%)	0.016(-84%)
AFGR	0.83 (-17%)	0.071(-17%)	0.027(-17%)	0.83(-17%)	0.083(-17%)
DF	1 (-0%)	0.046(-47%)	0.017(-47%)	1(-0%)	0.053(-47%)
URM	0.01 (-90%)	0.086(-0%)	0.033(-0%)	0.01 (-90%)	0.1 (-0%)

TABLE 4.1 – Impact on criticity

Chapitre 5

Exploration

1 Introduction

Ce chapitre est consacré à l'impact des solutions de contournement logicielles et matérielles retenues et proposées dans cette thèse sur la tolérance aux défauts, sur la surface et sur le timing. L'objectif et de déterminer quelles sont les techniques les plus performantes en fonction des trois critères définis précédemment. Enfin, des stratégies de redondance locale combinant deux techniques de redondance sur les blocs les plus critiques du FPGA sont proposées et comparées en termes de performances aux techniques de redondances logicielles et matérielles (FGR, IFGR, AFGR, DF, URM).

2 Méthodologie

Pour mesurer l'impact des solutions de contournement des défauts présentées au chapitre 4, des défauts sont injectés aléatoirement dans l'architecture du FPGA pendant sa configuration (figure 5.1). A partir des netlits des 20 benchs MCNC (.bliff), on réalise le clustering de cette netlist avec l'outil T-VPACK. Cette nouvelle netlist (.net) doit ensuite être placée puis routée. C'est lors de cette phase de placement/routage que des défauts sont injectés. Un défaut est modélisé par un état "Non défini" (Undifined) de sa sortie et représente un transistor toujours bloqué ("Stuck-open"). Les entrées du ou des éléments défectueux ne sont alors plus connectées à leurs sorties. Lors du placement, on définit des CLBs ou des clusters défectueux qui ne pourront pas être utilisés pour placer l'ensemble des applications. Le placeur va donc devoir placer les applications sur les CLBs/clusters fonctionnels si possible. Le fichier de placement est ensuite transmit au routeur. Il est là aussi possible d'injecter des défauts dans les réseaux d'interconnexions du FPGA. De nouveau, en cas de défauts sur un multiplexeur, l'ensemble des entrées/sorties ne sera plus connecté. Pour déterminer les performances en termes de tolérances aux défauts des outils de configuration et des techniques de contournement des défauts, des défauts sont alors injectés sur des multiplexeurs jusqu'à rendre les applications Non Routable (NR). Ce nombre de multiplexeurs défectueux pouvant être contournés correspond à la tolérance aux défauts de chaque technique. Le nombre de multiplexeurs ajoutés par chaque technique est comparé au nombre de multiplexeurs composant l'architecture initiale pour déterminer l'impact de chaque technique sur la surface du FPGA.

Pour finir, le délai du chemin critique (CPD) est lui aussi déterminé pour chaque technique et comparé à celui de l'architecture initiale pour déterminer l'impact sur le délai. Pour l'ensemble de nos simulations, on considère que les blocs mémoire sont protégés contre les défauts en utilisant des mécanismes de détection/correction d'erreur [65] [53].

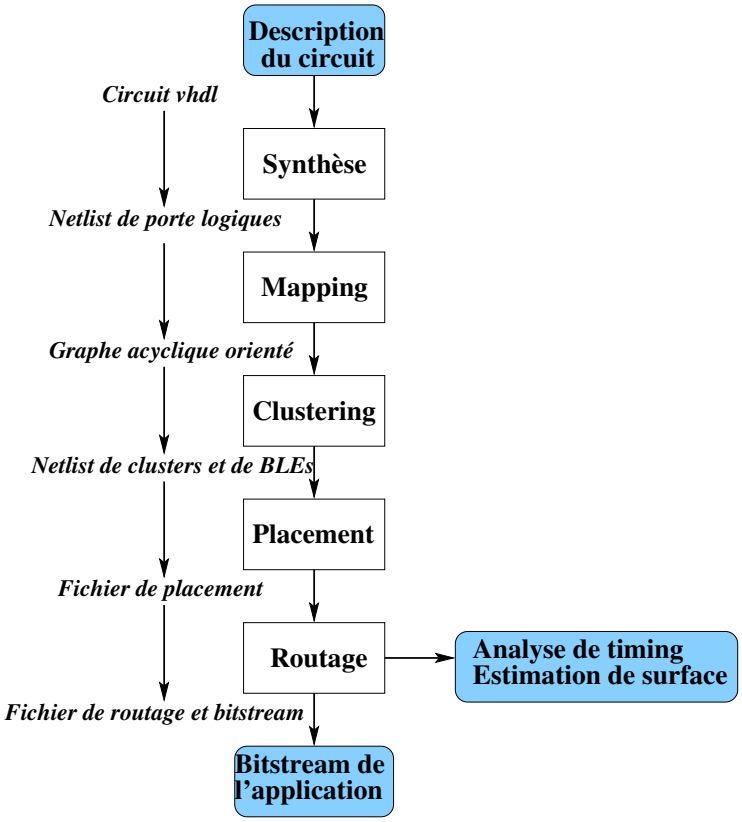


FIGURE 5.1 – Flot de configuration

3 Contournement logiciel

Dans cette section, nous allons établir les performances en termes de contournement des défauts et de timing du logiciel de placement/routage développé par l'entreprise Flexras pour l'architecture Mesh of Clusters. Le contournement logiciel consiste à laisser les outils de configuration trouver des solutions pour placer et router les applications en cas de défauts présents dans l'architecture. Pour établir ces performances, nous avons injecté des défauts lors du placement et du routage de chacune des applications MCNC jusqu'à rendre les applications Non Routable. Le placeur et le routeur utilisent chacun en entrée un fichier (.txt) contenant une liste d'éléments défectueux ainsi que leurs coordonnées dans le FPGA. Le nombre de défauts tolérés correspond à la tolérance aux défauts logiciels. Nous avons détaillé les performances du contournement logiciel pour chaque MSB composant notre architecture :

Ressource du FPGA	% d'éléments défectueux contournés	Impact sur le timing
DMSB level 2 Sbox	75.5%	+7.7%
DMSB level 1 Sbox	63%	+12.1%
UMSB Sbox	83%	+0.16%
DMSB cluster	25%	+8.9%
UMSB cluster	10%	+0%
moyenne	51.3%	+2.7%

TABLE 5.1 – Performances du contournement logiciel

On constate que le contournement logiciel est plus performant pour les ressources comprises dans les Switch box. En effet, jusqu'à 75.5% des ressources défectueuses peuvent être contournées dans les Switch box tout en maintenant la fonctionnalité du FPGA. Au contraire, le contournement logiciel s'avère moins performant pour les ressources comprises dans les clusters avec moins de 25% des ressources défectueuses contournables. Ces résultats vont nous permettre de définir une stratégie dans le but d'augmenter la tolérance aux défauts du FPGA tout en minimisant la surface ajoutée et l'impact sur le timing. En effet, les stratégies de redondance matérielle devront plutôt être utilisées sur les ressources comprises dans le cluster là où le contournement logiciel est le moins performant.

4 Contournement matériel

Dans cette section, nous allons déterminer l'impact de chacune des solutions ajoutant de la redondance décrites dans le chapitre 4 sur la tolérance aux défauts, sur le délai du chemin critique et sur la surface du FPGA. Pour chacune des techniques, nous détaillerons les résultats en termes de timing, de surface et de tolérance aux défauts pour chacun des 20 benchs MCNC.

4.1 Redondance à gros grain

Connexions diagonales

Dans un premier temps, nous allons déterminer les performances des connexions diagonales. Cette technique consiste à ajouter des connexions diagonales entres les Switch box. Ces nouvelles connexions augmentent le nombre de chemins disponibles entre les Switch box et devraient nous permettre de tolérer plus de défauts dans les Switch box. Le tableau 5.2 présente les performances de cette technique :

MCNC	Nombre d'éléments défectueux contournés		Pourcentage de connexions diagonales ajoutées	Impact sur la surface du FPGA	Impact sur le timing du FPGA
Circuits	Min	Max			
alu4	4 140 (1.84%)	16 500 (7.33%)	15%	+12.02%	+7.57%
apex2	17 180 (6%)	24 000 (8.53%)	25%	+17.28%	-3.23%
apex4	20 122 (9.13%)	26 600 (12.07%)	32%	+20.4%	-9.85%
bigkey	0 (0%)	2 590 (1.51%)	28%	+4.5%	+40%
clma	26 720 (2%)	121 900 (9.26%)	17%	+12.3%	-15.2%
des	0 (0%)	4 120 (2%)	20%	+8.9%	+3.6%
diffeq	0 (0%)	2 680 (1.52%)	10%	+4.45%	+32.5%
dsip	0 (0%)	4 500 (2.62%)	25%	+11.39%	-58.1%
elliptic	35 560 (7%)	46 080 (9.16%)	28%	+17.3%	+41.13%
ex1010	64 160 (7.88%)	80 120 (9.84%)	27%	+17.3%	+95.5%
ex5p	12 000 (5.28%)	16 900 (7.42%)	28%	+15.5%	+67%
frisc	40 560 (7.53%)	52 440 (9.73%)	20%	+17.7%	+72.2%
misex3	7 120 (2.73%)	16 000 (6.13%)	21%	+10.4%	+4.9%
pdc	29 720 (2.96%)	75 120 (7.45%)	20%	+14.08%	+9.45%
s298	8 540 (3.55%)	21 900 (9.11%)	32%	+17.05%	-6.88%
s38417	15 120 (1.93%)	45 590 (5.94%)	27%	+12.26%	-13%
S38584	15 720 (2.33%)	57 680 (8.71%)	20%	+13%	-5%
seq	9 560 (3.24%)	20 000 (6.8%)	23%	+11.8%	-1.2%
spla	28 320 (4.17%)	55 500 (8.23%)	15%	+11.1%	-6.6%
tseng	34 500 (21.04%)	37 020 (22.64%)	29%	+5.27%	+14.2%
moyenne	18 452 (4.43%)	36 167 (7.8%)	23%	+12.7%	+13.45%

TABLE 5.2 – Performances des connexions diagonales

On constate que cette technique peut être inefficace pour certains benchmarks (des, diffeq...) avec aucun ou très peu de défauts contournés. Au contraire, elle est très efficace et permet de contourner jusqu'à 22.64% de défauts avec des benchs comme *alu*, *apex*2 en ajoutant respectivement 15% et 25% de connexions diagonales. Cette technique est en effet plus efficace sur les benchs comme *alu* et *apex*2 qui utilisent beaucoup d'interconnexions entre clusters. En moyenne, cette technique permet de contourner entre 4.43% et 7.8% d'éléments défectueux dans l'architecture du FPGA avec une augmentation de surface de 12.7% (23% de connexions diagonales ajoutées en moyenne) et une augmentation du délai du chemin critique de 13.45%.

Longues connexions

Pour déterminer les performances de cette technique, nous avons utilisé uniquement des longues connexions de longueur $L = 2$. Ces nouvelles connexions augmentent le nombre de chemins disponibles entre les Switch boxes et devraient nous permettre de tolérer plus de défauts dans les Switch box. Le tableau 5.3 présente les performances de cette technique :

MCNC	Nombre d'éléments défectueux contournés		Pourcentage de longues connexions ajoutées	Impact sur la surface	Impact sur le timing
Circuits	min	max			
alu4	4 500 (2%)	18 000 (8%)	15%	+12.02%	+5.23%
apex2	19 440 (6.9%)	25 920 (9.21%)	25%	+17.28%	-5%
apex4	22 500 (10.22%)	27 000 (12.26%)	32%	+20.4%	-13%
bigkey	0 (0%)	3 920 (2.28%)	28%	+4.5%	+32.6%
clma	23 120 (1.75%)	115 600 (8.78%)	17%	+12.3%	-12.9%
des	0 (0%)	4 500 (2.23%)	20%	+8.9%	+1.72%
diffeq	0 (0%)	3 920 (2.23%)	10%	+4.45%	+29.2%
dsip	0 (0%)	3 920 (2.28%)	25%	+11.39%	-52.5%
elliptic	38 720 (7.7%)	48 400 (9.63%)	28%	+17.3%	+48.5%
ex1010	62 720 (7.7%)	78 400 (9.63%)	27%	+17.3%	+102%
ex5p	15 680 (6.9%)	19 600 (8.6%)	28%	+15.5%	+65.8%
frisc	42 320 (7.86%)	52 900 (9.82%)	20%	+17.7%	+69.9%
misex3	9 000 (3.45%)	18 000 (6.90%)	21%	+10.4%	+4.9%
pdc	31 360 (3.12%)	78 400 (7.82%)	20%	+14.08%	+6.1%
s298	5 120 (2.13%)	15 360 (6.39%)	32%	+17.05%	-4.2%
s38417	15 680 (2%)	47 040 (6.13%)	27%	+12.26%	-16.6%
S38584	13 520 (2%)	54 080 (8.17%)	20%	+13%	-6.3%
seq	11 560 (3.92%)	23 120 (7.86%)	23%	+11.8%	-2.45%
spla	25 000 (3.71%)	50 000 (7.42%)	15%	+11.1%	-3.5%
tseng	34 560 (21.08%)	37 440 (22.84%)	29%	+5.27%	+13.2%
moyenne	18 540 (4.62%)	36 276 (8%)	23%	+12.7%	+13.13%

TABLE 5.3 – Performances des longues connexions

On constate que cette technique est inefficace pour 4 benchs (des, diffeq, dsip, bigkey) avec aucun défaut contourné. Au contraire, elle est très efficace et permet de contourner jusqu'à 22.84% de défauts avec des benchs comme *alu*, *apex*2 avec respectivement 15% et 25% de longues connexions ajoutées. En moyenne, cette technique permet de contourner entre 4.62% et 8% d'éléments défectueux dans l'architecture du FPGA avec une augmentation de surface de 12.7% (23% de longues connexions ajoutées en moyenne) et une augmentation du délai du chemin critique de 13.13%. En conclusion, cette approche est similaire en termes de performances par rapport à l'ajout de connexions diagonales. Ces résultats s'expliquent par le fait que le canal de routage est pour la plupart des benchs de petites tailles. Le nombre de connexions distribuées sera donc similaire avec ces deux techniques. L'impact en termes de tolérance aux défauts sera donc très proche.

Connexions Directes entre Cluster

Pour finir les techniques à gros grain, nous allons maintenant établir les performances de la technique qui consiste à ajouter des connexions directes entre les clusters du FPGA. Le nombre de connexions entre clusters ajoutées est paramétrable et dépend de la tolérance aux défauts pour chaque application. Pour certains benchs, 2 connexions supplémentaires seulement seront nécessaires alors qu'un maximum sera nécessaire pour d'autres. Ce nombre de connexions dépend du nombre d'interconnexions par clusters pour chaque bench. Ces nouvelles connexions permettent de créer des connexions entre les clusters sans passer par les Switch boxes, le nombre de chemins disponibles est donc augmenté et devrait nous permettre de tolérer plus de défauts dans les clusters. Le tableau 5.4 présente les performances de cette technique pour chacun des 20 benchmarks MCNC :

MCNC	Nombre d'éléments défectueux contournés		Impact sur la surface	Impact sur le timing
Circuits	Min	Max		
alu4	3 718 (1.94%)	14 534 (7.58%)	+4.71%	+7.244%
apex2	14 400 (5.45%)	38 700 (14.65%)	+9.43%	+10.76%
apex4	16 224 (8.31%)	27 378 (14.04%)	+19.67%	-3.73%
bigkey	1 859 (1.13%)	18 083 (11%)	+29%	+10.1%
clma	19 339 (1.75%)	97 025 (8.78%)	+14%	-18.4%
des	3 718 (2.12%)	19 942 (11.44%)	+19.67%	+9.61%
diffeq	1 152 (0.77%)	7 488 (5%)	+29%	+18.65%
dsip	0 (0%)	7 744 (6.44%)	+29%	-7.84%
elliptic	10 648 (1.84%)	41 624 (7.20%)	+19.67%	+29.17%
ex1010	11 638 (1.87%)	62 422 (10.05%)	+24.33%	+12.8%
ex5p	0 (0%)	9 216 (5.64%)	+29%	+5.8%
frisc	15 884 (3.69%)	23 660 (11.73%)	+24.33%	+8.2%
misex3	3 718 (1.94%)	181 440 (12.34%)	+24.33%	+13.87%
pdc	25 344 (3.3%)	62 208 (8.09%)	+29%	+13%
s298	1 568 (0.76%)	10 584 (7.6%)	+24.33%	-5.2%
s38417	12 672 (2%)	31 104 (4.93%)	+24.33%	-2.71%
S38584	12 672 (1.9%)	31 104(4.68%)	+24.33%	-28.6%
seq	4 312 (1.87%)	10 584 (4.6%)	+29%	+0.96%
spla	9 702 (1.78%)	37 926 (7%)	+9.43%	-12.74%
tseng	0 (0%)	6 400 (6.08%)	+4.71%	+20.16%
moyenne	8 428 (2.12%)	36 958 (8.44%)	+21.06%	+4.06%

TABLE 5.4 – Performances des connexions directes entre clusters

On constate que cette technique peut être inefficace pour certains benchmarks (ex5p, tseng...) avec aucun ou très peu de défauts contournés car ces benchs utilisent la localité à l'intérieur de chaque cluster. Les connexions entre clusters n'ont donc que peu d'effets sur la tolérance aux défauts. Au contraire, elle est très efficace et permet de contourner jusqu'à 14.65% de défauts avec des benchs comme *apex*2, *apex*4 utilisant beaucoup plus d'interconnexions entre clusters. En moyenne, cette technique permet de contourner entre 2.12% et 8.4% d'éléments défectueux dans l'architecture du FPGA avec une augmentation de surface de 21.06% et une augmentation du délai du chemin critique de 4.06%. En conclusion, cette approche est plus performante avec certaines applications. En effet, elle sera plus efficace avec des applications ayant de nombreuses connexions entre clusters.

Conclusion

On constate qu'en termes de tolérance aux défauts, la technique la plus performante est celle qui consiste à ajouter des connexions directes entre les clusters (figure 5.2). En effet, elle permet de contourner jusqu'à 8.44% de multiplexeurs défectueux. Cependant, elle augmente la surface du FPGA de 21.06% contrairement aux deux autres techniques qui ont un impact sur la surface de seulement 12.7%. Pour finir, en termes de timing, les meilleures performances sont là aussi obtenues avec l'ajout de connexions directes entre clusters avec une augmentation du timing de 4.06%.

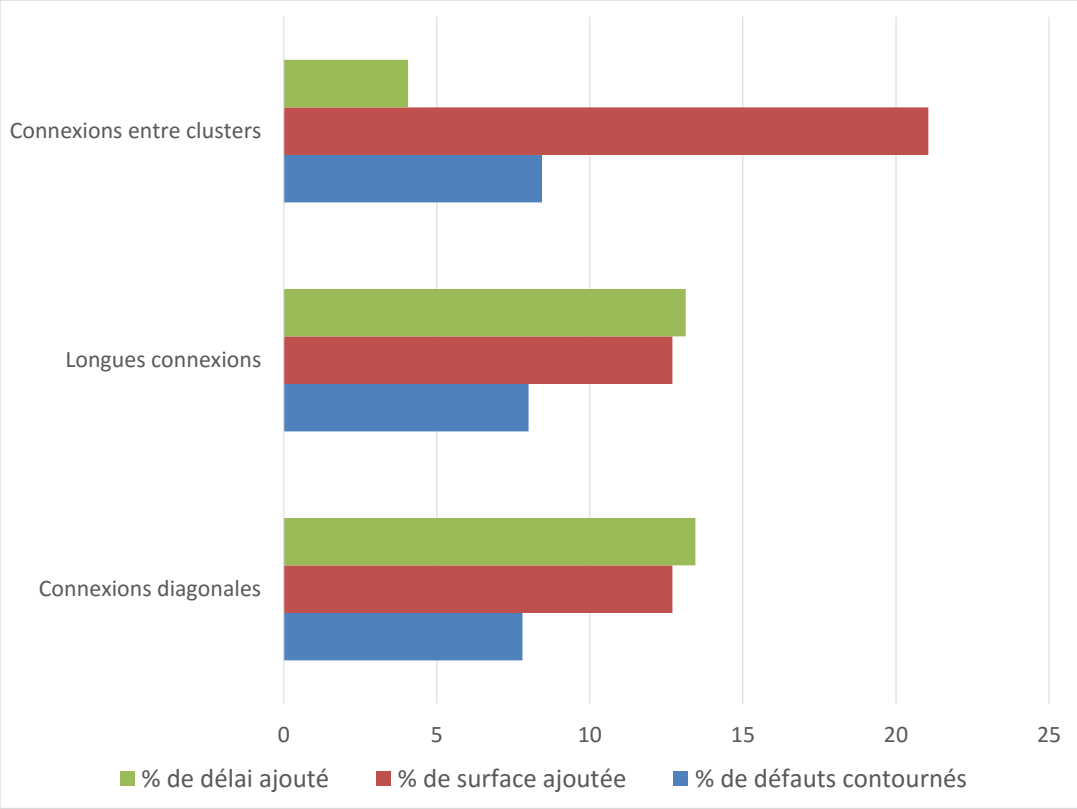


FIGURE 5.2 – Comparaison des performances des techniques à gros grain

4.2 Technique de redondance à grain fin

Redondance à grain fin (FGR)

Dans un premier temps, nous allons établir les performances de la FGR. Cette technique consiste à ajouter des multiplexeurs sur les entrées/sorties des MSBs du FPGA. Ces multiplexeurs permettent de contourner les éléments défectueux à l'intérieur des MSBs et devraient nous permettre de tolérer plus de défauts dans les Switch boxes et les clusters. Le tableau 5.5 présente les performances de cette technique :

MCNC	Moyenne de défauts contournés dans le FPGA	Impact sur la surface	Impact sur le timing
alu4	37.5%	+64.6%	+38%
apex2	29.3%	+64.6%	+27.7%
apex4	34.7%	+64.6%	+22.15%
bigkey	4.24%	+64.6%	+54%
clma	11.6%	+64.6%	+17.8%
des	12%	+64.6%	+32.1%
diffeq	17.2%	+64.6%	+34.5%
dsip	27.2%	+64.6%	+27.2%
elliptic	24.6%	+64.6%	+43%
ex1010	26.4%	+64.6%	+35.5%
ex5p	18%	+64.6%	+27%
frisc	23.1%	+64.6%	+32.4%
misex3	14.3%	+64.6%	+24.6%
pdc	19.5%	+64.6%	+29.5%
s298	24.1%	+64.6%	+23.6%
s38417	17.4%	+64.6%	+17.3%
S38584	19.6%	+64.6%	+24.7%
seq	16.8%	+64.6%	+26.6%
spla	23.3%	+64.6%	+22%
tseng	48.36%	+64.6%	+34.3%
moyenne	20.1%	+64.6%	+29.7%

TABLE 5.5 – Performances de la FGR

On constate qu'en moyenne, cette technique permet de contourner 20.1% d'éléments défectueux dans l'architecture du FPGA avec une augmentation de surface de 64.6% et une augmentation du délai du chemin critique de 29.7%.

Redondance à grain fin améliorée (IFGR)

L'IFGR est une amélioration de la FGR développé pour éviter les conflits entre signaux lors du contournement des défauts. Cette technique consiste à doubler les sorties par multiplexeurs à l'intérieur des MSB en plus des multiplexeurs ajoutés sur les entrées/sorties des MSBs issues de la FGR. Le fait de doubler les sorties permet de multiplier par 2 le nombre de chemins permettant de contourner un défaut. Le tableau 5.6 présente les performances de cette technique pour chacun des 20 benchmarks MCNC :

MCNC	Moyenne de défauts contournés dans le FPGA	Impact sur la surface	Impact sur le timing
alu4	+84.58%	+150%	+38%
apex2	+84.58%	+150%	+27.7%
apex4	+84.58%	+150%	+22.15%
bigkey	+84.58%	+150%	+54%
clma	+84.58%	+150%	+17.8%
des	+84.58%	+150%	+32.1%
diffeq	+84.58%	+150%	+34.5%
dsip	+84.58%	+150%	+27.2%
elliptic	+84.58%	+150%	+43%
ex1010	+84.58%	+150%	+35.5%
ex5p	+84.58%	+150%	+27%
frisc	+84.58%	+150%	+32.4%
misex3	+84.58%	+150%	+24.6%
pdc	+84.58%	+150%	+29.5%
s298	+84.58%	+150%	+23.6%
s38417	+84.58%	+150%	+17.3%
S38584	+84.58%	+150%	+24.7%
seq	+84.58%	+150%	+26.6%
spla	+84.58%	+150%	+22%
tseng	+84.58%	+150%	+34.3%
moyenne	+84.58%	+150%	+29.7%

TABLE 5.6 – Performances de l'IFGR

Cette technique est très performante en termes de tolérance aux défauts car elle permet de contourner 84.58% d'éléments défectueux dans l'architecture du FPGA. Cependant pour tolérer autant de défauts, elle nécessite une augmentation de surface de 150% et une augmentation du délai du chemin critique de 29.7%.

Redondance à grain fin adaptée (AFGR)

Pour réduire l'impact sur la surface du FPGA des techniques comme la FGR et l'IFGR, une nouvelle technique a été développée appelée AFGR. Pour limiter la surface ajoutée, le deuxième étage de multiplexeurs utilisé pour restaurer les sorties est supprimé. Puis, l'étage de contournement devient commun à l'ensemble des DMSBs des clusters et des Switch boxes. De nouvelles connexions sont donc créées, il est désormais possible de contourner un défaut en utilisant le DMSB voisin. Le tableau 5.7 présente les performances de cette technique :

MCNC	Moyenne de défauts contournés dans le FPGA	Impact sur la surface	Impact sur le timing
alu4	35%	+30%	+17.5%
apex2	29.3%	+30%	+8.85%
apex4	34.7%	+30%	+8.3%
bigkey	7%	+30%	+5.2%
clma	11.6%	+30%	-11.3%
des	12%	+30%	-4.7%
diffeq	17.2%	+30%	+27.3%
dsip	22%	+30%	-12.5%
elliptic	24.6%	+30%	+17.9%
ex1010	26.4%	+30%	+16.36%
ex5p	18%	+30%	+5.3%
frisc	21.1%	+30%	+6.4%
misex3	14.3%	+30%	+7.13%
pdc	19.5%	+30%	+3.95%
s298	24.1%	+30%	+8.9%
s38417	13.4%	+30%	+40%
S38584	15.6%	+30%	-11.8%
seq	18%	+30%	-3%
spla	21.1%	+30%	+3.4%
tseng	48.36%	+30%	+15.2%
moyenne	+21%	+30%	+7.46%

TABLE 5.7 – Performances de l' AFGR

Cette technique limite bien l'impact de la redondance sur la surface du FPGA avec une augmentation de 30% (contre 64.6% pour la FGR et 150% pour l'IFGR. En contrepartie la tolérance aux défauts est réduite puisque l'on peut contourner en moyenne 21% des défauts avec une augmentation du délai du chemin critique de 7.46%.

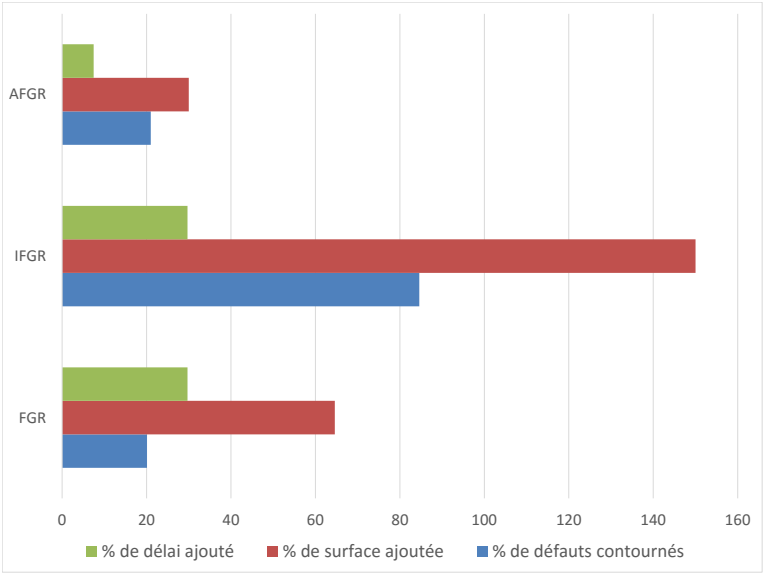


FIGURE 5.3 – Comparaison des performances des techniques FGR, IFGR et AFGR

En termes de tolérance aux défauts, l'IFGR reste la technique la plus performante avec 84.58% de défauts tolérés mais l'impact sur la surface du FPGA est trop importante. Cependant, la technique que nous proposons appelée AFGR propose le meilleur rapport entre la tolérance aux défauts de l'architecture (21%), l'impact sur la surface (30%) et sur le timing (7.46%).

Distribution des feedback (DF)

La seconde technique que nous avons proposée appelée DF utilise les feedbacks présents dans l'architecture des clusters et des Switch boxes. Nous avons décidé de distribuer les feedbacks de l'architecture sur l'ensemble des DMSBs pour augmenter la tolérance aux défauts de l'architecture. De nouvelles connexions sont donc créées, il est désormais possible de contourner un défaut en utilisant le DMSB voisin. Le tableau 5.8 présente les performances de cette technique pour les 8 benchmarks MCNC utilisant des feedbacks dans les clusters :

MCNC	Moyenne de défauts contournés dans le FPGA	Impact sur la surface	Impact sur le timing
alu4	21.4%	+56%	+11.9%
apex4	4.7%	+11.2%	+4.6%
dsip	14.6%	+44.8%	-10.6%
frisc	4%	+11.2%	-12.4%
pdc	29.5%	+67.2%	-3.2%
s298	4.1%	+11.2%	+2.23%
spla	24.1%	+56%	-3.5%
tseng	57.8%	+112%	-1.5%
moyenne	+20%	+46.2%	-1.56%

TABLE 5.8 – Performances de la DF

On constate qu'en moyenne, cette technique permet de contourner 20% d'éléments défectueux dans l'architecture du FPGA avec une augmentation de surface de 46.2% et une réduction du délai du chemin critique de 1.56%.

Multiplexeur redondant (URM)

La DF se concentre sur le contournement des défauts à l'intérieur des DMSBs. Nous avons donc proposé une technique appelée URM qui est applicable uniquement sur les UMSB de l'architecture. Cette technique qui cible une ressource de l'architecture permet de diminuer l'augmentation de surface et l'impact sur le timing mais limite aussi le nombre de défauts contournables. Le tableau 5.9 présente les performances de cette technique pour chacun des 20 benchmarks MCNC :

MCNC	Moyenne de défauts contournés dans le FPGA	Impact sur la surface	Impact sur le timing
alu4	16.9%	+17.34%	+2.36%
apex2	14%	+17.34%	+1.5%
apex4	16.6%	+17.34%	+0.8%
bigkey	1.5%	+17.34%	+1.2%
clma	2.8%	+17.34%	+1.3%
des	7.8%	+17.34%	+0.7%
diffeq	11.1%	+17.34%	+2.3%
dsip	15.1%	+17.34%	+1.5%
elliptic	8.2%	+17.34%	+1.9%
ex1010	10.1%	+17.34%	+0.5%
ex5p	15.1%	+17.34%	+0%
frisc	17%	+17.34%	+0.34%
misex3	12.3%	+17.34%	+0.13%
pdc	11%	+17.34%	+0.75%
s298	15.2%	+17.34%	+0.9%
s38417	7.5%	+17.34%	+0%
S38584	8.2%	+17.34%	+1.8%
seq	6%	+17.34%	+0.15%
spla	12.1%	+17.34%	+0.4%
tseng	17.34%	+17.34%	+0.47%
moyenne	+11.29%	+17.34%	+1%

TABLE 5.9 – Performances de l'URM

Cette technique permet de contourner peu de défauts 11.29% puisqu'elle n'est appliquée que sur les UMSBs de l'architecture. L'impact sur la surface 17.34% et sur le délai du chemin critique de 1% sera au contraire réduit grâce à cette redondance ciblée sur une partie de l'architecture.

Conclusion

En conclusion, on constate que les techniques à gros grain sont moins performantes que les techniques à grain fin si on prend l'ensemble des résultats. La figure 5.4 montre que la technique la plus performante en termes de tolérance aux défauts est l'IFGR. En terme d'impact sur la surface, les techniques les plus performantes sont des techniques à gros grain (longues connexions et connexions diagonales) puis l'URM. Ensuite, en termes de timing, l'impact sera minimum avec la distribution des feedbacks (DF). On peut expliquer ce résultat par le fait que de nouveaux chemins entre les sorties et les entrées des clusters et Switch boxes sont ajoutés donc de nouvelles possibilités de routage. Pour finir, la technique qui est la plus performante en termes de tolérance aux défauts (21%), en terme de surface (30%) et de timing (7.46%) est l'AFGR. Cependant l'ensemble de ces techniques appliquées séparément a un impact significatif sur la surface du FPGA. C'est pourquoi par la suite nous avons décidé de nous orienter vers une redondance ciblée sur les éléments les plus critiques du FPGA.

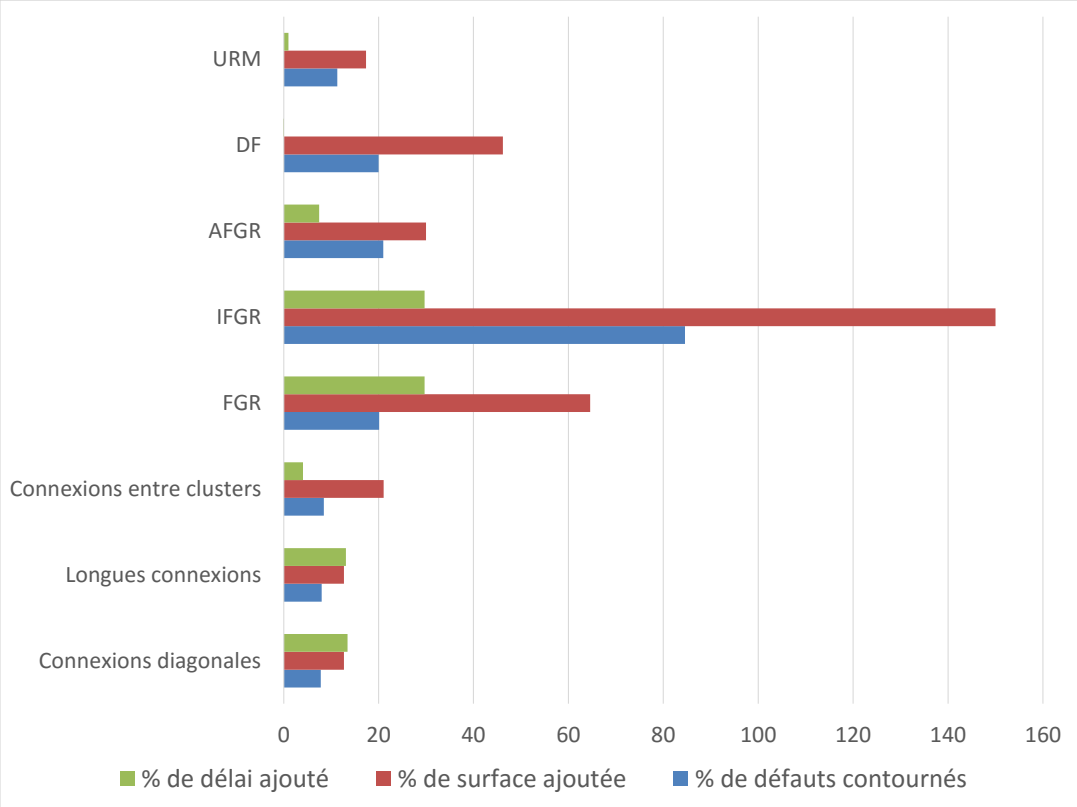


FIGURE 5.4 – Comparaison des performances des techniques FGR, IFGR et AFGR

5 Stratégies de redondance locale

5.1 Introduction

Au chapitre 4, nous avons proposé une méthode permettant de calculer la criticité de l'ensemble des blocs composant notre FPGA. Nous en avons conclu que les blocs les plus critiques sont les UMSBs des clusters ainsi que des Switch boxes puis les DMSBs des clusters. Nous allons donc appliquer dans cette section les techniques étudiées précédemment uniquement sur les éléments les plus critiques en les combinant pour trouver le meilleur ratio entre la tolérance aux défauts et l'impact sur la surface et le timing.

5.2 Stratégies de redondance locale

Notre objectif est d'évaluer l'impact de différentes techniques de redondance locale (Local Redundancy Strategies) sur la tolérance aux défauts, la surface et le timing du FPGA. Ces techniques sont appliquées sur les éléments les plus critiques du FPGA (UMSBs des clusters et des Switch Boxes et DMSBs des clusters). Dans le tableau 5.10, le *dsmb* indique que la technique est appliquée sur les DMSBs des clusters et le *umsb* indique qu'elle est appliquée sur les UMSBs des clusters et des Switch boxes. L'URM est la technique la plus performante sur les UMSB avec le moins d'impact sur la surface et sur le timing. La DF est applicable uniquement sur les DMSBs, c'est pourquoi pour la première technique de redondance locale LRS1, nous appliquons la DF sur les DMSBs des clusters et l'URM sur l'ensemble

des UMSBs. L'IFGR est la technique la plus performante en termes de tolérance aux défauts, c'est pourquoi nous l'appliquons sur l'ensemble des UMSBs et avec la DF pour la seconde technique LRS2. Le principal inconvénient de l'IFGR est son impact sur la surface du FPGA. Pour la troisième technique LRS3, nous utilisons la technique qui a le meilleur ratio en termes de tolérance aux défauts, impact sur la surface et sur le timing : l'AFRG sur les DMSBs et la technique la plus performante en terme de tolérance aux défauts l'IFGR sur les DMSBs des clusters. Afin de nous comparer aux techniques de l'état de l'art utilisées en redondance locale, la stratégie LRS4 combine la FGR sur les DMSBs des clusters et l'URM sur les UMSBs. Pour finir, nous associons l'AFGR sur les DMSBs des clusters et l'URM sur les UMSBs.

Stratégies de redondance locale	**FGR**	**IFGR**	**AFGR**	**DF**	**URM**
LRS1				dmsb	umsb
LRS2		umsb		dmsb	
LRS3		umsb	dmsb		
LRS4	dsmb				umsb
LRS5			dmsb		umsb

TABLE 5.10 – Stratégies de redondances locales

Techniques de redondance locale	**Nombre d'éléments défectueux contournés**		**Impact sur la surface du FPGA**	**Impact sur le timing**
	Min	Max		
LRS1	12(1.15%)	492(47.39%)	+52.34%	-7.81%
LRS2	180(17.34%)	492(47.39%)	+51.4%	+6.16%
LRS3	180(17.34%)	394(37.95%)	+20.9%	+22.36%
LRS4	12(1.15%)	394(37.95%)	+31.34%	+17.09%
LRS5	12(1.15%)	394(37.95%)	+21.84%	+9.65%

TABLE 5.11 – Performances des stratégies de redondance locale

5.3 Conclusion

Le tableau 5.11 présente les performances des différentes techniques en terme de tolérance aux défauts, de surface et de timing. Les stratégies de redondance lo-

cale LRS1 et LRS2 permettent de contourner un grand nombre de défauts (47.39%) avec une augmentation de la surface du FPGA d'environ 52%. Au contraire avec les techniques de redondance locale LRS3, LRS4 and LRS5, l'impact sur la surface du FPGA est réduit (+20.9% pour LRS3, +31.34% pour LRS4 et +21.84% pour LRS5) avec un nombre moins important de défauts contournés (37.95%). En conclusion, utiliser des stratégies de redondance locale sur les blocs les plus critiques du FPGA permet de tolérer un nombre important de défauts avec un impact moins important que les techniques présentées au chapitre 4 sur la surface du FPGA. Au niveau du timing, la technique la plus performante est LRS1 (-7.81%) car de nouvelles possibilités de routage sont ajoutées à l'architecture des clusters avec la DF. Pour finir le meilleur ratio est obtenu avec LRS2 avec 37.95% d'éléments défectueux contournés, une surface ajoutée de +21.84% et une augmentation du délai du chemin critique de +9.65%.

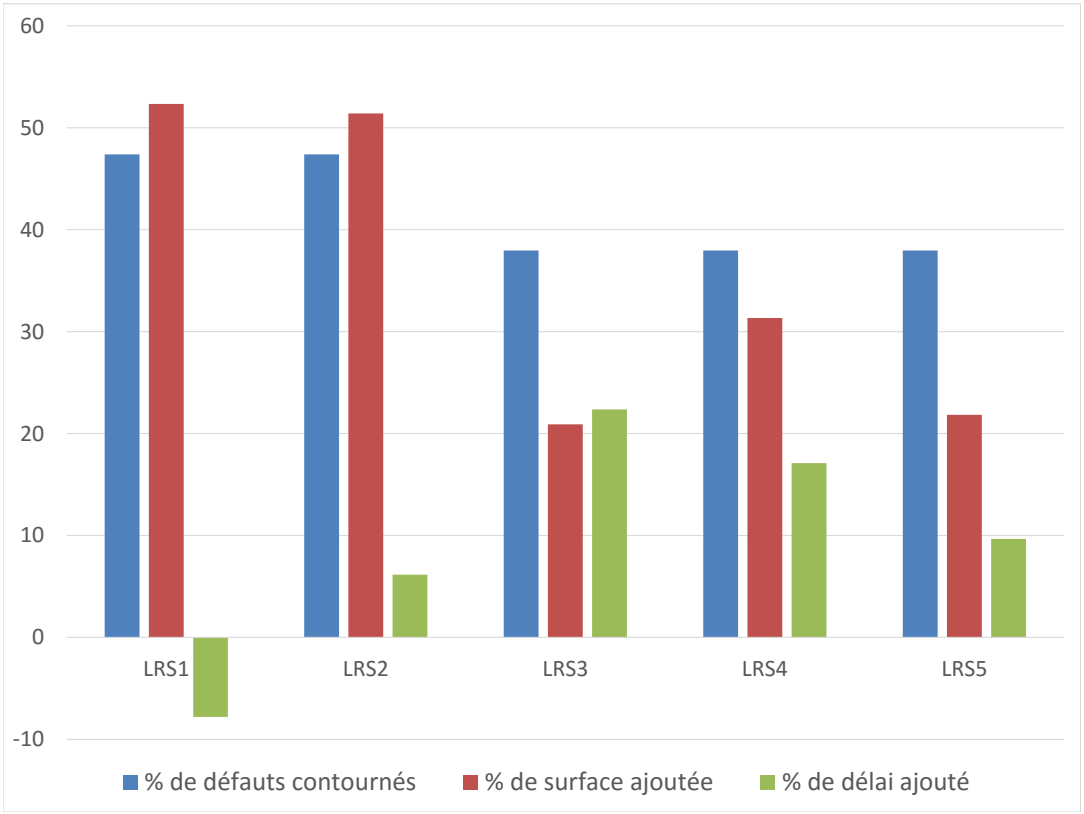


FIGURE 5.5 – Comparaison des performances des techniques de redondance locale

6 Conclusion

Nous avons étudié dans ce chapitre les performances des techniques de contournement à gros grain, à grain fin puis de stratégies de redondance locale. La technique la plus performante en termes de tolérance aux défauts est l'IFGR. Cependant, son impact sur la surface du FPGA (+150%) fait que cette technique ne peux pas être retenue. En termes d'impact sur la surface du FPGA, les techniques de redondance à gros grain (longues connexions, connexions diagonales) sont les plus performantes. Cependant, elles aussi ne sont pas retenues car elles ne contournent que trop peu

de défauts (environ 8%). Pour finir en termes de timing, la technique la plus performante est la Distribution des Feedbacks. En effet, cette technique consiste à ajouter de nouveaux chemins entre les entrées/sorties des clusters c'est pourquoi le délai du chemin critique peut être réduit. Pour finir le meilleur ratio est obtenu avec la stratégie de redondance locale LRS2 qui combine l'AFGR sur les DMSBs du cluster et l'URM sur les UMSBs. En effet, avec cette technique 37.95% d'éléments défectueux sont contournés, la surface ajoutée est de seulement +21.84% et l'augmentation du délai du chemin critique est de +9.65%.

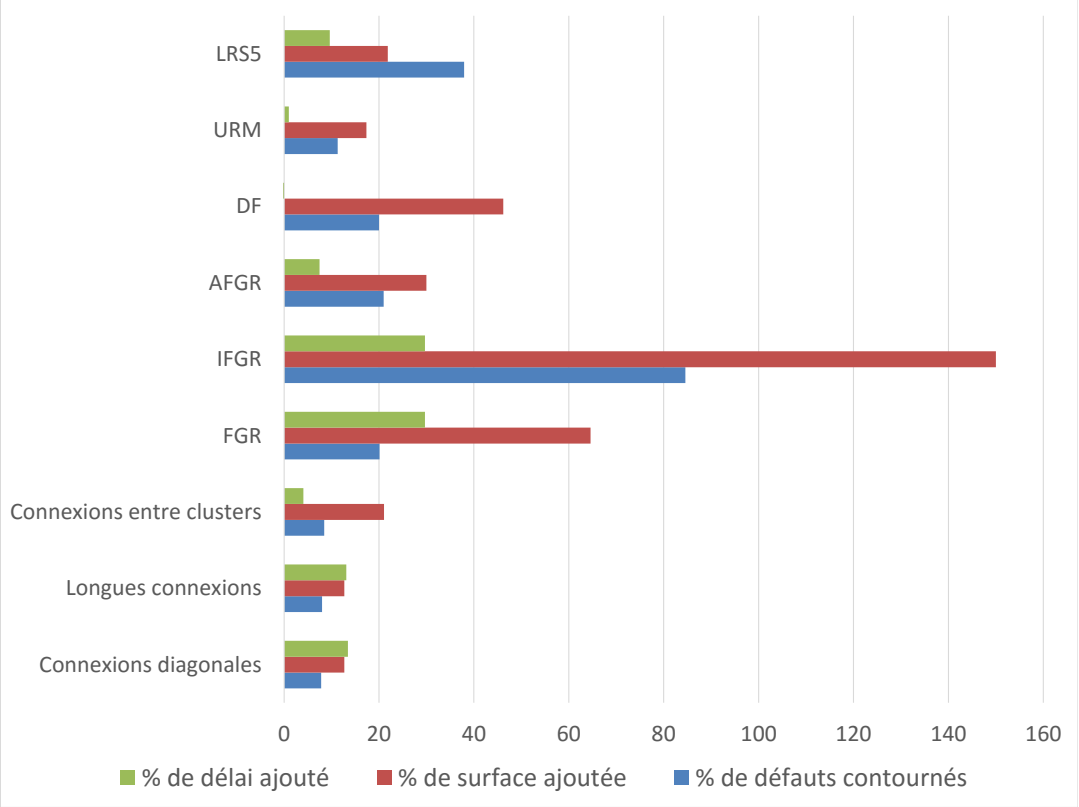


FIGURE 5.6 – Comparaison des performances des techniques à gros grains, à grain fin et de redondance locale

Chapitre 6

Conclusion

1 Contributions

Le but premier de cette thèse était de créer une architecture de FPGA Mesh of Clusters tolérante aux défauts de fabrication. Nous sommes intervenus sur l'ensemble de la chaîne de conception et de configuration d'un FPGA. De ce fait, nous avons pu apporter des contributions à chacune des étapes inhérentes à la conception d'un FPGA.

1.1 Flot de conception

La première étape a consisté à créer le flot de conception du FPGA sous forme de générateurs sous l'environnement du LIP6 : STRATUS et avec les outils de conception de CADENCE. L'architecture du FPGA Mesh of Clusters est créée à partir de générateurs écrits en langage python. Ces générateurs offrent la possibilité à l'utilisateur de choisir l'ensemble des paramètres architecturaux du FPGA à savoir la taille du FPGA, le nombre de blocs logiques par cluster, le nombre d'entrées par bloc logique, le nombre d'entrées/sorties par cluster, la taille du canal de routage et pour finir le nombre d'entrées/sorties du FPGA. En résumé, nos générateurs permettent de créer n'importe quel type d'architecture de FPGA Mesh of Clusters. Les différents générateurs permettent de créer le code VHDL (ou verilog) de notre architecture. A partir de la description de notre circuit en VHDL, notre flot de conception utilise un second générateur pour réaliser le *layout* du circuit. La définition du floorplan est faite de manière automatique en fonction de la taille du FPGA. Le placement des différentes cellules est lui aussi automatisé de manière à respecter l'organisation sous forme de matrice de tuiles. Le routage des blocs est alors fait automatiquement à partir du placement. Pour finir, une vérification des règles de dessin et une comparaison entre le schéma structurel et le *layout* permettent de vérifier que le *layout* généré est conforme à nos attentes. Avec ces deux générateurs, l'utilisateur choisit l'architecture de son FPGA Mesh of Cluster et le flot est déroulé jusqu'à fournir le *layout*. A partir des 20 benchs MCNC, nous avons démontré que l'architecture Mesh of Clusters permet de réduire l'utilisation des ressources d'interconnexions de 48% par rapport à une architecture mesh classique.

1.2 Flot de configuration

Pour permettre de démontrer les performances de notre architecture et des architectures de tolérances aux défauts, un flot de configuration nous a été fourni par FleXras. Ce flow permet à partir des netlists des applications de réaliser le clustering, le mapping, le placement, le routage et l'extraction du bitstream. De plus, grâce à nos modifications, il est désormais possible d'injecter des défauts lors du placement (sous forme de CLBs ou de clusters défectueux) ainsi que du routage (sous formes de multiplexeurs défectueux dans les réseaux d'interconnexions) des applications. Des modifications ont été aussi apportées pour permettent de tester les stratégies de contournement proposées au chapitre 4.

1.3 Tolérance aux défauts

Nos travaux consistaient aussi à améliorer la tolérance aux défauts de fabrication des FPGA. Nous avons dans un premier temps étudié les différentes techniques de contournement existantes (matérielles et logicielles). Au vue des bonnes performances, nous avons retenu les techniques de redondance matérielles à gros grain et à grain fin (CGR et FGR). Ces techniques suggèrent l'ajout de ressources de secours pour contourner les éléments défectueux du FPGA et ont été développées pour des architectures de type mesh (FGR et IFGR). Nous avons donc adapté ces techniques à l'architecture Mesh of Clusters et proposé de nouvelles (AFGR, DF, URM) pour réduire l'impact de ces techniques sur la surface et sur le délai du FPGA. Ce flot de configuration nous a aussi permis d'étudier les performances des différentes solutions de contournement proposées. Il en ressort que l'IFGR est la technique la plus performante en termes de contournement des défauts mais augmente de manière conséquente la surface et le délai du FPGA. De manière générale, les techniques de contournement appliquées sur l'architecture globale du FPGA permettent de tolérer un nombre important de défauts mais augmentent beaucoup trop la surface du FPGA. Nous proposons donc une méthode de calcul de la criticité des blocs composant notre FPGA. Le FPGA est modélisé sous forme d'un graphe de clusters et de Switch boxes. Chacun des clusters et des Switch boxes sont modélisés par un sous-graphe dans lequel chaque multiplexeur contenu dans les MSBs est représenté par un nœud. La criticité de chaque nœud est définie par le nombre d'éléments logiques connectés à ce nœud et par les paramètres architecturaux. Cette méthode démontre que les UMSBs sont les éléments les plus critiques de l'architecture Mesh of Clusters puis ensuite viennent les DMSBs du cluster. Ces informations nous ont permis de proposer des stratégies de redondance locale où les techniques de contournement comme l'AFGR, l'URM sont dorénavant appliquées sur les éléments les plus critiques de notre circuit. Grâce à ces stratégies de redondance locale, nous minimisons l'impact de ces techniques sur la surface et sur le délai du FPGA. En conclusion, nous avons mis au point un flot de conception pour FPGA Mesh of Clusters permettant à l'utilisateur de définir sa propre architecture de FPGA dans laquelle il peut fixer sa métrique en terme de tolérance aux défauts, en terme de surface et en terme du délai du FPGA.

2 Perspectives

La première amélioration possible concerne la génération du *layout*. En effet, nous utilisons un placement automatique des cellules de notre architecture. Il serait intéressant d'utiliser la généricité du FPGA lors du placement et de recréer une matrice de tuiles. Une comparaison des performances de ce type de placement par rapport au placement automatique pouvant être étudiée par la suite. L'impact de la technologie cible permettrait de vérifier que nos résultats sont viables quel que soit la technologie. Dans un second temps, nous avons limité notre étude à 2 techniques adaptées de l'état de l'art puis nous avons proposé 3 nouvelles techniques de tolérance aux défauts. D'autres techniques peuvent être proposées, notamment au niveau de l'architecture des multiplexeurs. De plus, au niveau des techniques à gros grain, nous avons choisi d'ajouter des connexions supplémentaires pour améliorer la routabilité de notre architecture. Il serait intéressant de comparer les performances de nos techniques gros grains à celles qui suggèrent l'ajout de lignes et de colonnes de secours. Ces études ont été faites à partir d'une distribution aléatoire et sous forme de paquets des défauts. Il serait intéressant d'utiliser des statistiques de distribution des défauts issue de données réelles (après fabrication). Pour finir, au niveau de l'architecture, nous avons utilisé des CLBs à 4 entrées. Une étude sur l'impact du nombre d'entrées par CLBs sur les performances du FPGA peut être effectuée. Dans la même optique, il existe plusieurs structures de Switch box (wilton, universelle...) dont l'impact sur les performances de notre architecture nécessite d'être étudié. Enfin, Une étude sur l'impact des différentes techniques de contournement sur la consommation et sur la puissance dissipé du circuit peut aussi être menée.

List of Publications

Conference Papers

2014 Impact of defect tolerance techniques on the criticality of a SRAM-based Mesh of Cluster FPGA (Adrien Blanchardon, Roselyne Chotin-Avot, Habib Mehrez, Emna Amouri), *In ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on*, IEEE, pp. 1-6, 2014.

2014 Improve defect tolerance in a cluster of a SRAM-based Mesh of Cluster FPGA using hardware redundancy (Adrien Blanchardon, Roselyne Chotin-Avot, Habib Mehrez, Emna Amouri), *In Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, pp. 1-4, 2014.

2014 Impact of Cluster Size on Routability, Testability and Robustness of a Cluster in a Mesh FPGA (Saif Ur Rehman, Adrien Blanchardon, Arwa Ben Dhia, Mounir Benabdenbi, Roselyne Chotin-Avot, Lirida Naviner, Lorena Anghel, Habib Mehrez, Emna Amouri, Zied Marrakchi), *In VLSI (ISVLSI), 2014 IEEE Computer Society Annual Symposium on*, pp. 553-558, 2014.

2013 Efficient Multilevel Interconnect Topology for Cluster-based Mesh FPGA Architecture (Emna Amouri, Adrien Blanchardon, Roselyne Chotin-Avot, Habib Mehrez, Zied Marrakchi), *In International Conference on Reconfigurable Computing and FPGAs*, pp. 1-6, 2013.

2013 A Defect-tolerant Cluster in a Mesh SRAM-based FPGA (Arwa Ben Dhia, Saif Ur Rehman, Adrien Blanchardon, Lirida Naviner, Mounir Benabdenbi, Roselyne Chotin-Avot, Habib Mehrez, Emna Amouri, Zied Marrakchi), *In IEEE International Conference on Field-Programmable Technology*, pp. 434-437, 2013.

2012 Générateur d'Architecture de FPGA (Adrien Blanchardon, Roselyne Chotin-Avot, Habib Mehrez), *In Colloque GDR SOC-SIP*, pp. 1-3, 2012.

Bibliographie

[1] Aditya A Aggarwal and David M Lewis. Routing architectures for hierarchical field programmable gate arrays. In *Computer Design : VLSI in Computers and Processors, 1994. ICCD'94. Proceedings., IEEE International Conference on*, pages 475–478. IEEE, 1994.

[2] Elias Ahmed and Jonathan Rose. The effect of lut and cluster size on deep-submicron fpga performance and density. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 12(3) :288–298, 2004.

[3] Emna Amouri. *Outils de placement et de routage pour des architectures FPGA sécurisées contre les attaques DPA*. PhD thesis, Paris 6, 2011.

[4] J Yu Anthony. *Defect Tolerance for Yield Enhancement of FPGA Interconnect Using Fine-grain and Coarse-grain Redundancy*. PhD thesis, The University of British Columbia, 2005.

[5] S Areibi, G Grewal, D Banerji, and P Du. Hierarchical fpga placement. *Electrical and Computer Engineering, Canadian Journal of*, 32(1) :53–64, 2007.

[6] Sophie Bellœil-Dupuis. *Optimisation automatique des chemins de données arithmétiques par l'utilisation des systèmes de numération redondants*. PhD thesis, Paris 6, 2009.

[7] Arwa Ben Dhia, Saif Ur Rehman, Adrien Blanchardon, Lirida Naviner, Mounir Benabdenbi, Roselyne Chotin-Avot, Habib Mehrez, Emna Amouri, and Zied Marrakchi. A Defect-tolerant Cluster in a Mesh SRAM-based FPGA. pages 434–437, December 2013.

[8] V. Betz and J. Rose. How much logic should go in an fpga logic block ? *IEEE Design and Test of Computers*, 15(1) :10–15, 1998.

[9] Vaughn Betz and Jonathan Rose. Vpr : A new packing, placement and routing tool for fpga research. In *Field-Programmable Logic and Applications*, pages 213–222. Springer, 1997.

[10] Vaughn Betz and Jonathan Rose. How much logic should go in an fpga logic block ? *IEEE Design & Test of Computers*, 15(1) :10–15, 1998.

[11] Vaughn Betz, Jonathan Rose, and Alexander Marquardt. *Architecture and CAD for deep-submicron FPGAs*. Kluwer Academic Publishers, 1999.

[12] Cristiana Bolchini, Giacomo Buonanno, Donatella Sciuto, and Renato Stefanelli. Static redundancy techniques for cmos gates. In *Circuits and Systems, 1996. ISCAS'96., Connecting the World., 1996 IEEE International Symposium on*, volume 4, pages 576–579. IEEE, 1996.

[13] Elaheh Bozorgzadeh, S Ogrenci Memik, Xiaojian Yang, and Majid Sarrafzadeh. Routability-driven packing : Metrics and algorithms for cluster-based fpgas. *Journal of Circuits, Systems, and Computers*, 13(01) :77–100, 2004.

[14] Elaheh Bozorgzadeh, S Ogrenci Memik, Xiaojian Yang, and Majid Sarrafzadeh. Routability-driven packing : Metrics and algorithms for cluster-based fpgas. *Journal of Circuits, Systems, and Computers*, 13(01) :77–100, 2004.

[15] R. Brayton, G. Hachtel, and A. Sangiovanni-Vincentelli. Multilevel logic synthesis. *Proceedings of the IEEE*, pages 264–300, 1990.

[16] Robert K Brayton. The decomposition and factorization of boolean expression. *ISCAS-82*, 1982.

[17] Nicola Campregher, Peter YK Cheung, George A Constantinides, and Milan Vasilko. Analysis of yield loss due to random photolithographic defects in the interconnect structure of fpgas. In *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, pages 138–148. ACM, 2005.

[18] W.S. Carter, K. Duong, R.H. Freeman, H.C Hsieh, J.Y. Ja, E. Mahoney, L.T. Ngo, and S.L. Sze. A user programmable reconfigurable logic array. *Proc. IEEE, Custom Integrated Circuits Conference*, pages 233–235, 1986.

[19] Vi Cuong Chan and David M Lewis. Area-speed tradeoffs for hierarchical field-programmable gate arrays. In *Field-Programmable Gate Arrays, 1996. FPGA'96. Proceedings of the 1996 ACM Fourth International Symposium on*, pages 51–57. IEEE, 1996.

[20] Yao-Wen Chang, DF Wong, and CK Wong. Universal switch modules for fpga design. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 1(1) :80–101, 1996.

[21] Altera Corp. Altera's patented redundancy technology dramatically increases yields on high-density apex 20ke devices. *In Press Release*, Novembre 2000.

[22] Altera Corp. #6,034,536, #6,166,559, #6,337,578, #6,344,755, #6,600,337 and #6,759,871,. *In United states patents*, 2000-2004.

[23] Altera Corp. Apex redundancy. *http ://www.altera.com/products/devices/apex/features/apx-redundancy*, 2005.

[24] Mehrdad Eslami Dehkordi and Stephen Dean Brown. The effect of cluster packing and node duplication control in delay driven clustering. In *Field-Programmable Technology, 2002.(FPT). Proceedings. 2002 IEEE International Conference on*, pages 227–233. IEEE, 2002.

[25] Andre DeHon. Reconfigurable architectures for general-purpose computing. 1996.

[26] André DeHon. Balancing interconnect and computation in a reconfigurable computing array (or, why you don't really want 100% lut utilization). In *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, pages 69–78. ACM, 1999.

[27] Asbjoern Djupdal and Pauline C Haddow. Defect tolerance inspired by artificial evolution. In *Symposium on VLSI, 2008. ISVLSI'08. IEEE Computer Society Annual*, pages 28–33. IEEE, 2008.

[28] Asbjørn Djupdal. Evolving static hardware redundancy for defect tolerant fpgas. 2008.

[29] Abderrahim Doumar, Satoshi Kaneko, and Hideo Ito. Defect and fault tolerance fpgas by shifting the configuration data. In *Defect and Fault Tolerance in VLSI Systems, 1999. DFT'99. International Symposium on*, pages 377–385. IEEE, 1999.

[30] Peng Du, Gary William Grewal, Shawki Areibi, and Dilip K Banerji. A fast hierarchical approach to fpga placement. In *ESA/VLSI*, pages 497–503. Citeseer, 2004.

[31] Mohamed A Elgamel and Magdy A Bayoumi. Crosstalk noise analysis in ultra deep submicrometer technologies. *Interconnect Noise Optimization in Nanometer Technologies*, pages 45–57, 2006.

[32] Hongbing Fan, Yu-Liang Wu, and Catherine L Zhou. Augmented disjoint switch boxes for fpgas. In *Proceedings of the 4th international symposium on Information and communication technologies*, pages 129–134. Trinity College Dublin, 2005.

[33] Charles M Fiduccia and Robert M Mattheyses. A linear-time heuristic for improving network partitions. In *Design Automation, 1982. 19th Conference on*, pages 175–181. IEEE, 1982.

[34] Robert J Francis, Jonathan Rose, and Zvonko G Vranesic. *Field-programmable gate arrays*, volume 180. Springer Science & Business Media, 1992.

[35] Rudy Garcia. Rethink fault models for submicron-ic test. *Test and Measurement World*, 21(12) :35–46, 2001.

[36] Lars W Hagen and Andrew B Kahng. Combining problem reduction and adaptive multistart : A new technique for superior iterative partitioning. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 16(7) :709–717, 1997.

[37] Scott Hareland, Jose Maiz, Mohsen Alavi, Kaizad Mistry, Steve Walsta, and Changhong Dai. Impact of cmos process scaling and soi on the soft error rates of logic processes. In *VLSI Technology, 2001. Digest of Technical Papers. 2001 Symposium on*, pages 73–74. IEEE, 2001.

[38] DJ-H Huang and Andrew B Kahng. When clusters meet partitions : new density-based methods for circuit decomposition. In *Proceedings of the 1995 European conference on Design and Test*, page 60. IEEE Computer Society, 1995.

[39] Michael Hutton, Khosrow Adibsamii, and Andrew Leaver. Timing-driven placement for hierarchical programmable logic devices. In *Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays*, pages 3–11. ACM, 2001.

[40] Crosspoint Solutions Inc. Fpga redundancy. *In United states patents #5,777,887*, 1998.

[41] Scott Kirkpatrick. Optimization by simulated annealing : Quantitative studies. *Journal of statistical physics*, 34(5-6) :975–986, 1984.

[42] Ian Kuon and Jonathan Rose. Measuring the gap between fpgas and asics. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(2) :203–215, 2007.

[43] Yen-Tai Lai and Ping-Tsung Wang. Hierarchical interconnection structures for field programmable gate arrays. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 5(2) :186–196, 1997.

[44] Yen-Tai Lai and Ping-Tsung Wang. Hierarchical interconnection structures for field programmable gate arrays. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 5(2) :186–196, 1997.

[45] Vijay Lakamraju and Russell Tessier. Tolerating operational faults in cluster-based fpgas. In *Proceedings of the 2000 ACM/SIGDA eighth international symposium on Field programmable gate arrays*, pages 187–194. ACM, 2000.

[46] Charles E Leiserson. Fat-trees : universal networks for hardware-efficient supercomputing. *Computers, IEEE Transactions on*, 100(10) :892–901, 1985.

[47] Fei Li, Deming Chen, Lei He, and Jason Cong. Architecture evaluation for power-efficient fpgas. In *Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays*, pages 175–184. ACM, 2003.

[48] Alexander Sandy Marquardt, Vaughn Betz, and Jonathan Rose. Using cluster-based logic blocks and timing-driven packing to improve fpga speed and density. In *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, pages 37–46. ACM, 1999.

[49] Zied Marrakchi, Hayder Mrabet, Umer Farooq, and Habib Mehrez. Fpga interconnect topologies exploration. *International Journal of Reconfigurable Computing*, 2009 :6, 2009.

[50] Zied Marrakchi, Hayder Mrabet, Christian Masson, and Habib Mehrez. Mesh of tree : unifying mesh and mfpga for better device performances. In *Proceedings of the First International Symposium on Networks-on-Chip*, pages 243–252. IEEE Computer Society, 2007.

[51] Clive Maxfield. *The design warrior's guide to FPGAs : devices, tools and flows*. Elsevier, 2004.

[52] Larry McMurchie and Carl Ebeling. Pathfinder : a negotiation-based performance-driven router for fpgas. In *Proceedings of the 1995 ACM third international symposium on Field-programmable gate arrays*, pages 111–117. ACM, 1995.

[53] Fabrice Monteiro, Stanislaw J Piestrak, Houssein Jaber, and Abbas Dandache. Fault-secure interface between fault-tolerant ram and transmission channel using systematic cyclic codes. In *On-Line Testing Symposium, 2007. IOLTS 07. 13th IEEE International*, pages 199–200. IEEE, 2007.

[54] Gordon E Moore et al. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1) :82–85, 1998.

[55] R. Murgai, R. Brayton, and A. Sangiovanni-Vincentelli. On clustering for minimum delay/area. *IEEE International Conference on Computer-Aided Design*, pages 6–9, 1991.

[56] Helia Naeimi and André DeHon. A greedy algorithm for tolerating defective crosspoints in nanopla design. In *Field-Programmable Technology, 2004. Proceedings. 2004 IEEE International Conference on*, pages 49–56. IEEE, 2004.

[57] Sani R Nassif. Within-chip variability analysis. In *Electron Devices Meeting, 1998. IEDM'98. Technical Digest., International*, pages 283–286. IEEE, 1998.

[58] William H Pierce. *Failure-tolerant computer design*. Academic Press, 2014.

[59] Jonathan Rose and Stephen Brown. Flexibility of interconnection structures for field-programmable gate arrays. *Solid-State Circuits, IEEE Journal of*, 26(3) :277–282, 1991.

[60] Jonathan Rose, Robert J Francis, David Lewis, and Paul Chow. Architecture of field-programmable gate arrays : The effect of logic block functionality on area efficiency. *Solid-State Circuits, IEEE Journal of*, 25(5) :1217–1225, 1990.

[61] Jonathan Rose, Robert J Francis, David Lewis, and Paul Chow. Architecture of field-programmable gate arrays : The effect of logic block functionality on area efficiency. *Solid-State Circuits, IEEE Journal of*, 25(5) :1217–1225, 1990.

[62] Carl Sechen and Alberto Sangiovanni-Vincentelli. The timberwolf placement and routing package. *IEEE Journal of Solid-State Circuits*, 20(2) :510–522, 1985.

[63] Amit Singh, Ganapathy Parthasarathy, and Malgorzata Marek-Sadowska. Efficient circuit clustering for area and power reduction in fpgas. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 7(4) :643–663, 2002.

[64] R Sinh, V Parthar, K.F Poole, and K Rajkanan. Semiconductor manufacturing in the 21st century. *Semiconductor Fabtech 9th Edition*, pages 223–232, 1999.

[65] Charles W Slayman. Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations. *Device and Materials Reliability, IEEE Transactions on*, 5(3) :397–404, 2005.

[66] Python software foundation. http ://www.python.org. 1992.

[67] Charles H. Stapper. Modeling of integrated circuit defect sensitivities. *IBM Journal of Research and Development*, 27(6) :549–557, 1983.

[68] James H Stathis. Physical and predictive models of ultrathin oxide reliability in cmos devices and circuits. *Device and Materials Reliability, IEEE Transactions on*, 1(1) :43–59, 2001.

[69] Stephen Trimberger. A reprogrammable gate array and applications. *Proceedings of the IEEE*, 81(7) :1030–1041, 1993.

[70] Steven JE Wilton. *Architectures and algorithms for field-programmable gate arrays with embedded memory*. PhD thesis, Citeseer, 1997.

[71] Xilinx. Technical report. *Xilinx Virtex-II Platform FPGA User Guide*, 2005.

[72] CA Xilinx, San Jose. Easypath solutions. *http ://www.xilinx.com/products/easypath/*, 2005.

[73] Marrakchi Zied, Mrabet Hayder, Amouri Emna, and Mehrez Habib. Efficient tree topology for fpga interconnect network. In *Proceedings of the 18th ACM Great Lakes symposium on VLSI*, pages 321–326. ACM, 2008.