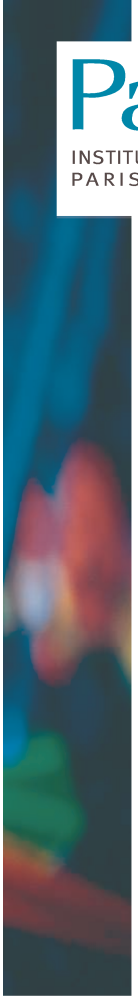
Hardening basic blocks in a mesh of clusters FPGA

Arwa Ben Dhia



ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2014-ENST-0068

EDITE - ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité : « Électronique et Communications »

Présentée et soutenue publiquement par :

Arwa BEN DHIA

le 14 novembre 2014

Durcissement de circuits logiques reconfigurables

Directrice de thèse : **Lirida ALVES DE BARROS NAVINER**

Jury

M. Walter STECHELE, Professeur, Université Technique de Munich	Rapporteur
M. Matteo SONZA REORDA, Professeur, École Polytechnique de Turin	Rapporteur
M. Habib MEHREZ, Professeur, Université Pierre et Marie Curie	Examinateur
M. Arnaud VIRAZEL, Maître de conférences, Université Montpellier 2	Examinateur
Mme Lirida NAVINER, Professeur, Télécom ParisTech	Directrice de Thèse

TELECOM ParisTech
École de l'Institut Mines-Télécom - Membre de ParisTech
46 rue Barrault 75013 Paris - (+33) 1 45 81 77 77 - www.telecom-paristech.fr

T H È S E

Hardening Basic Blocks in a Mesh of Clusters FPGA

PhD candidate: Arwa Ben Dhia

Supervisor: Prof. Lirida Naviner

"All roads that lead to success
have to pass through
Hard Work Boulevard at some point."
Eric Thomas

Acknowledgements

This thesis and its results could be achieved thanks to the support of many people from Télécom ParisTech and from Robust FPGA project. I am sincerely grateful to all of them, especially to my supervisor Lirida Naviner, not only for her valuable scientific advice but also for her extraordinary human qualities. I would also like to take the time to thank the whole jury for accepting to appraise this work. I am sure their comments and suggestions will help improve the manuscript.

None of this would have been possible without the love, encouragement and support of my family. This work is dedicated to them. Mom, dad, and little brother, thank you !

I would also like to thank all my friends, colleagues and Télécom ParisTech staff. You are all part of this accomplishment. Thank you so much !

Abstract

As feature sizes scale down to nano-design level, electronic devices have become smaller, more performant, less power-consuming, but also less reliable. Indeed, reliability has arisen as a serious challenge in nowadays' microelectronics industry and as an important design criterion, along with area, performance and power consumption. For instance, physical defects due to imperfections in the manufacturing process have been observed more frequently, impacting the yield. Besides, nanometric circuits have become more vulnerable during their lifetime to ionizing radiation which causes transient faults. Both manufacturing defects and transient faults contribute to decreasing reliability of integrated circuits.

When moving to a new technology node, Field Programmable Gate Arrays (FPGAs) are the first coming into the market, thanks to their low development and Non-Recurring Engineering (NRE) costs and their flexibility to be used for any application. FPGAs have especially attractive characteristics for space and avionic applications, where reconfigurability, high performance and low-power consumption can be fruitfully used to develop innovative systems. However, missions take place in a harsh environment, rich in radiation, which can induce soft errors within electronic devices. This shows the importance of FPGA reliability as a design criterion in safety and critical applications.

Most of commercial FPGAs have a mesh architecture and their logic blocks are gathered into clusters. Therefore, this thesis deals with the fault tolerance of basic blocks (clusters and switch boxes) in a mesh of clusters FPGA. These blocks are mainly made up of multiplexers. In order to improve their reliability, it is imperative to be able to assess it first, then select the proper hardening approach according to the available budget. So, this is the main outline in which this thesis is conceived. Its goals are twofold : (a) analyze the fault tolerance of the basic blocks in a mesh of clusters FPGA, and point out the most vulnerable components (b) propose hardening schemes at different granularity levels, depending on the hardening budget.

As far as the first goal is concerned, a methodology to evaluate the reliability of the cluster is proposed. This methodology uses an existent analytical method for reliability computation of combinational circuits. The same method is employed to identify the worthiest components to be hardened. Regarding the second goal, hardening techniques are proposed at both multiplexer and transistor levels. At multiplexer level, two hardening solutions are presented. The first solution resorts to spacial redundancy and concerns the logic block structure. A novel Configurable Logic Block (CLB) architecture baptized *Butterfly* is introduced. It is compared with other hardened CLB architectures in terms of reliability and cost penalties. The second hardening solution is a *redundanceless* scheme. It is based on a "smart" synthesis that consists in seeking the most reliable design in a given founder library, instead of directly using a redundant solution. Then, at transistor level, new single-ended and dual-rail multiplexer architectures are proposed. They are compared to different other transistor structures, according to suitable design metrics.

French Summary

Les circuits électroniques connaissent depuis plusieurs décennies une densité d'intégration de plus en plus accrue, les rendant plus compacts, plus performants, mais aussi moins fiables. En effet, d'une part, les circuits deviennent plus vulnérables aux défauts physiques, avec un procédé de fabrication qui n'a pas évolué de la même vitesse que les réductions d'échelle. Et d'autre part, ils sont plus susceptibles aux fautes transitoires qui peuvent être causées par des particules chargées attaquant le silicium. Cette baisse de la fiabilité constitue un problème majeur dans les applications critiques : spatiales, militaires ou médicales. Aussi est-il très important de pouvoir estimer la fiabilité d'un circuit dans le but d'y appliquer des techniques adéquates d'amélioration de la fiabilité, dites de *durcissement*.

Les FPGAs sont largement utilisés dans diverses applications, notamment celles requérant un haut niveau de fiabilité, grâce à leur temps de développement réduit, leur coût attractif et leur reconfigurabilité. Le projet ANR *Robust FPGA* qui a financé cette thèse s'est intéressé à la conception d'un FPGA tolérant aux défauts de fabrication. Néanmoins, cette thèse ne s'est pas limitée à ce genre de fautes permanentes, et a étendu l'étude de la fiabilité au cas de fautes transitoires. Comme la plupart des FPGAs en industrie ont une architecture matricielle et rassemblent leurs blocs logiques en *clusters*, l'architecture initiale du FPGA adoptée dans le projet est de type *matrice de clusters*. L'objectif de la thèse a été donc de proposer des architectures de blocs de base (*clusters* et boîtes d'interconnexion) plus robustes. C'est pour cela qu'il fallait, dans un premier temps, évaluer la fiabilité initiale de ces blocs, afin de pouvoir l'améliorer par des méthodes appropriées et selon le budget de durcissement disponible.

Les travaux de cette thèse s'articulent essentiellement autour de deux axes principaux : (a) l'évaluation de la fiabilité des blocs de base du FPGA, et (b) la proposition de méthodes de durcissement à différents niveaux de granularité.

Analyse de la fiabilité des blocs de base

Types de masquage

La fiabilité d'un circuit peut être définie comme la probabilité que ses sorties soient correctes, malgré la présence de fautes. En effet, si une faute a lieu dans un circuit, elle ne se propage pas forcément en sortie. Auquel cas, la faute est dite *masquée*. Donc, évaluer la fiabilité d'un circuit revient à évaluer sa capacité de masquage. Il y a trois phénomènes de masquage :

- *Le masquage logique*, a lieu pour certaines combinaisons d'entrée quand la faute apparaît dans un chemin l'empêchant d'arriver à la sortie, grâce à l'architecture inhérente du circuit. Ce type de masquage est indépendant de la technologie La figure 1 montre deux exemples de masquage logique avec deux portes OR et NAND.
- *Le masquage électrique*, a lieu quand la faute n'a suffisamment pas de durée ou d'amplitude pour parvenir jusqu'à la sortie du circuit. La figure 2 représente un exemple d'un glitch qui se masque électriquement.
- *Le masquage temporel*, a lieu quand la faute à la sortie du circuit apparaît en dehors de la fenêtre d'échantillonnage. La figure 3 montre un exemple d'une faute transitoire masquée temporellement.

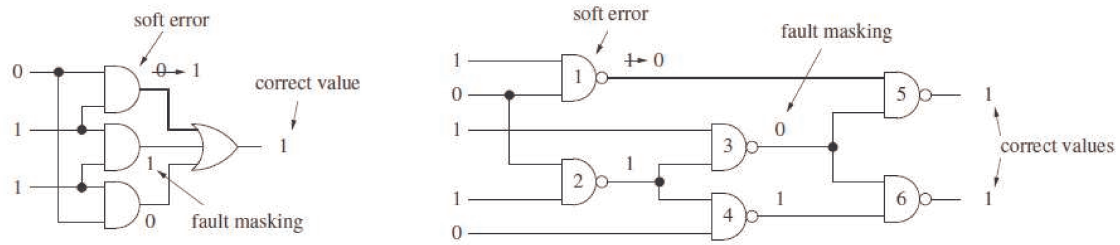


FIGURE 1 – Exemples de masquage logique.

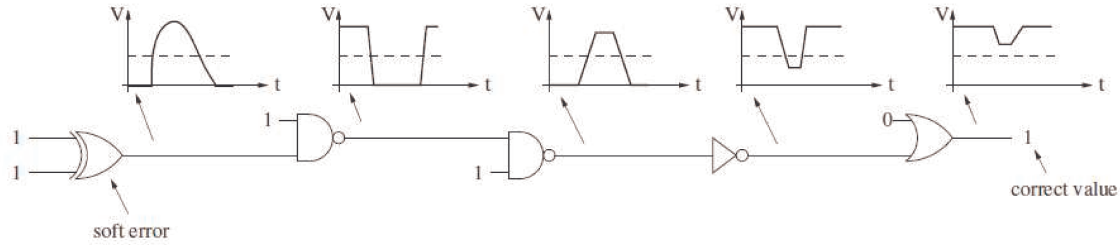


FIGURE 2 – Exemple de masquage électrique.

Comme les deux derniers phénomènes de masquage sont dépendants de la technologie, leurs effects continuent à diminuer à cause des réductions d'échelle. En effet, la baisse de la tension d'alimentation et du délai de propagation des portes logiques diminuent la probabilité de masquage électrique, et les fréquences d'opération plus élevées réduisent le masquage temporel. Par conséquent, le masquage logique est de nos jours le plus prédominant parmi les phénomènes de masquage, et il est le plus difficile à modéliser et caractériser.

La méthode SPR

Il existe dans l'état de l'art différentes méthodes permettant d'évaluer le masquage logique d'un circuit. Une des méthodes analytiques les plus utilisées est le SPR. Cette approche basée sur un algorithme de propagation des signaux traite des fautes multiples simultanées et a une complexité linéaire avec le nombre de portes dans le circuit.

En fait, la fiabilité d'un signal donné est définie comme étant la probabilité que ce signal ait une valeur correcte. Un signal binaire x peut prendre quatre valeurs différentes : un zéro/un correct ($0_c/1_c$), un zéro/un incorrect ($0_i/1_i$). Ces états sont représentés dans une matrice 2×2 comme le montre l'équation 1, où $P(.)$ est la fonction de probabilité. Une telle matrice SPM est attribuée à chaque signal dans le circuit. La fiabilité du signal x, notée R_x, est égale à $P(x = 0_c) + P(x = 1_c) = x_0 + x_3$.

$$SPM(x) = \begin{bmatrix} P(x = 0_c) & P(x = 1_i) \\ P(x = 0_i) & P(x = 1_c) \end{bmatrix} = \begin{bmatrix} x_0 & x_1 \\ x_2 & x_3 \end{bmatrix} \quad (1)$$

À part les matrices SPMs, SPR a aussi besoin des matrices ITMs et PTMs des portes pour déterminer l'effet cumulatif des fautes multiples sur la fiabilité du circuit. En effet, la matrice ITM d'une porte traduit sa table de vérité où les colonnes correspondent aux différentes valeurs de sortie et les lignes aux combinaisons d'entrée. Ensuite, la matrice

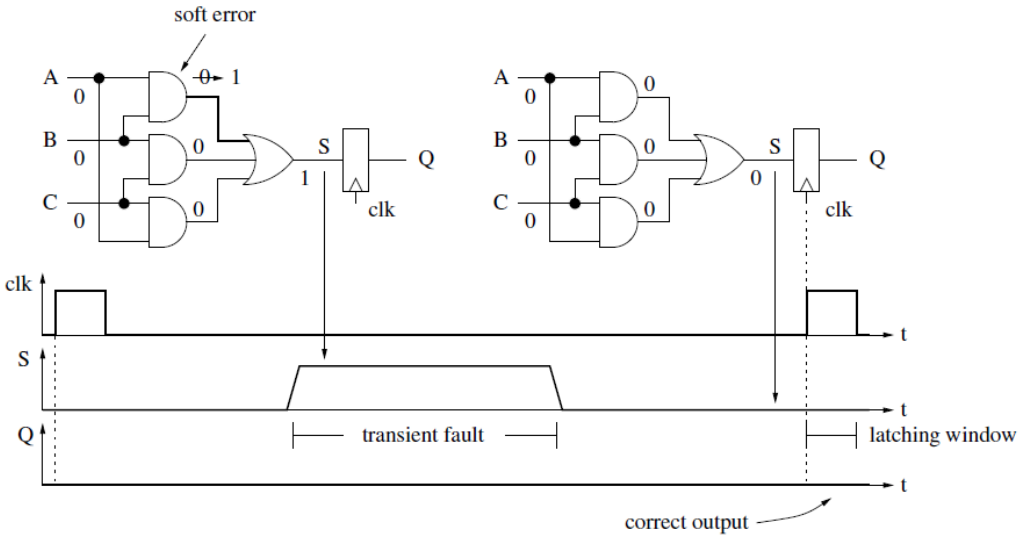


FIGURE 3 – Exemple de masquage temporel.

PTM est directement déduite de la ITM en remplaçant les '1' par q la valeur de fiabilité de la porte. La figure 4 montre un exemple pour la porte NAND dont $q = 0.92$. Donc, pour avoir la fiabilité du signal en sortie, il faut faire le produit tensoriel des SPMs en entrée puis multiplier par la matrice PTM de la porte. Ensuite, la matrice obtenue est mutipliée en amont par la matrice ITM transposée pour retomber sur une matrice SPM en sortie dont la trace correspond à la fiabilité du signal en sortie. Et dans l'exemple de la figure 4, on retrouve bien la valeur de q. Pour un circuit plus complexe, les matrices SPMs se propagent à travers les couches logiques suivant le même principe. Cette méthode SPR va être utilisée pour l'analyse de la fiabilité des blocs de base : *cluster* et boîte d'interconnexion.

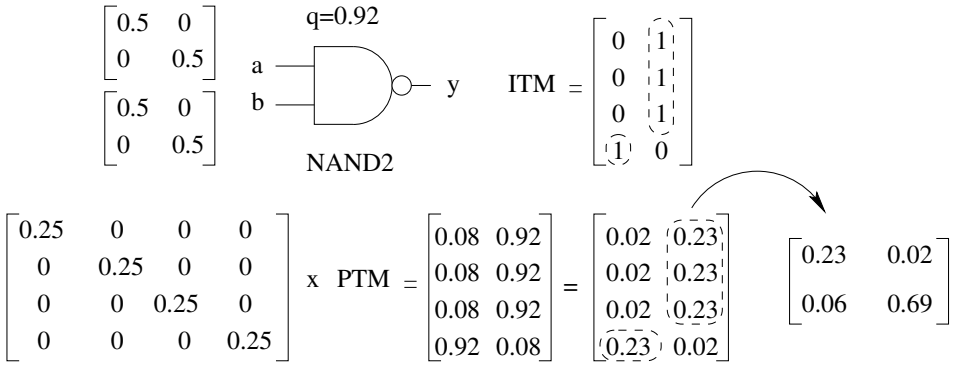


FIGURE 4 – Exemple de SPR appliqué à la porte NAND.

Analyse de la fiabilité du *cluster*

L'architecture initiale du FPGA est la matrice de *clusters* représentée sur la figure 5. Les *clusters* regroupent un nombre de blocs logiques élémentaires (BLE), ont un réseau interne d'interconnexion et sont directement connectés aux boîtes de commutation (les

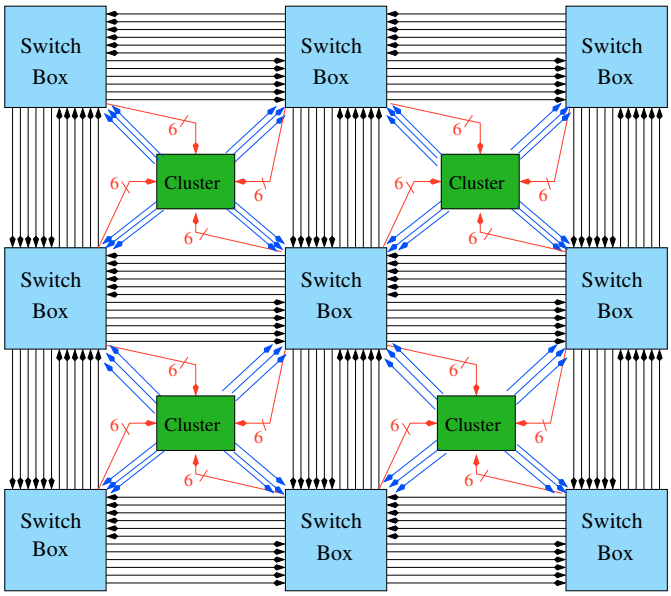


FIGURE 5 – Architecture en matrice de *clusters*.

switch boxes). Cette architecture de FPGA profite à la fois des avantages de l'architecture matricielle, par son agencement en tuiles, et de ceux de l'architecture hiérarchique, par le regroupement des BLEs en *clusters*.

Le *cluster mesh*, représenté à la figure 6, a 24 entrées primaires et 12 sorties. Il contient 10 BLEs (les CLBs sur la figure) et un réseau local d'interconnexion composé de crossbars. Un BLE, représenté à la figure 7(a) est composé d'une LUT à 4 entrées, suivie d'un registre puis d'un Mux2 permettant de choisir soit de fonctionner en combinatoire soit en séquentiel. La figure 7(b) montre le schéma de la LUT-4 ayant 16 bits SRAM où est stockée la fonction logique, puis selon les bits d'entrée, la sortie sera sélectionnée à travers le corps de Mux2s. Les crossbars sont aussi composés de Mux2s. Par exemple, le Crossbar 'Down' représenté à la figure 8 contient 10 multiplexeurs 9 :1, chacun ayant 6 entrées primaires et 3 autres venant des sorties calculées précédemment, de façon à pouvoir réutiliser des résultats et privilégier la localité de l'application. On a ainsi un total de 40 multiplexeurs 9 :1 dans tous les crossbars 'Down', plutôt que 40 multiplexeurs 36 :1, si on avait opté pour une structure classique en 'full crossbar' qui serait coûteuse en nombre de commutateurs et augmenterait le délai et la consommation de puissance.

Hypothèses

- N'importe quel Mux2 dans les BLEs et les crossbars est sujet aux erreurs : collage à zéro ou à un en sortie avec la même probabilité notée s, ou inversion binaire avec une fiabilité notée q (ceci est spécifiée dans la matrice PTM du Mux2).
- L'objectif étant de caractériser la fiabilité inhérente du *cluster*, les entrées sont supposées exemptes de fautes.
- La mémoire de configuration est censée être protégée contre les fautes.

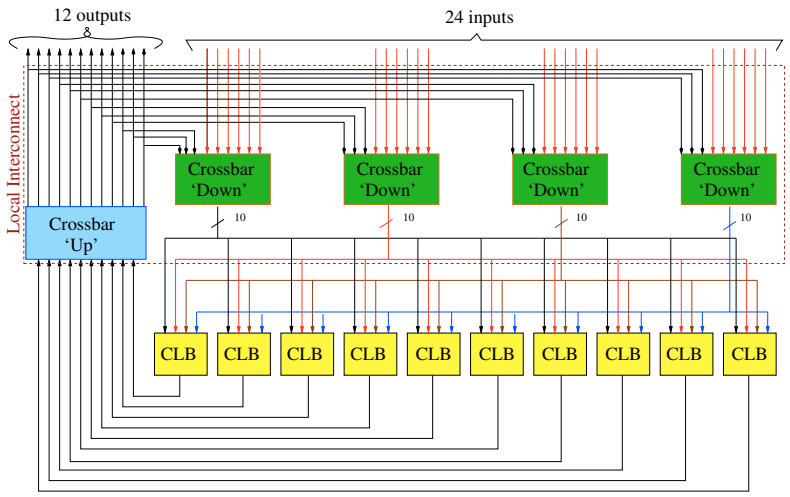


FIGURE 6 – Un *cluster mesh*.

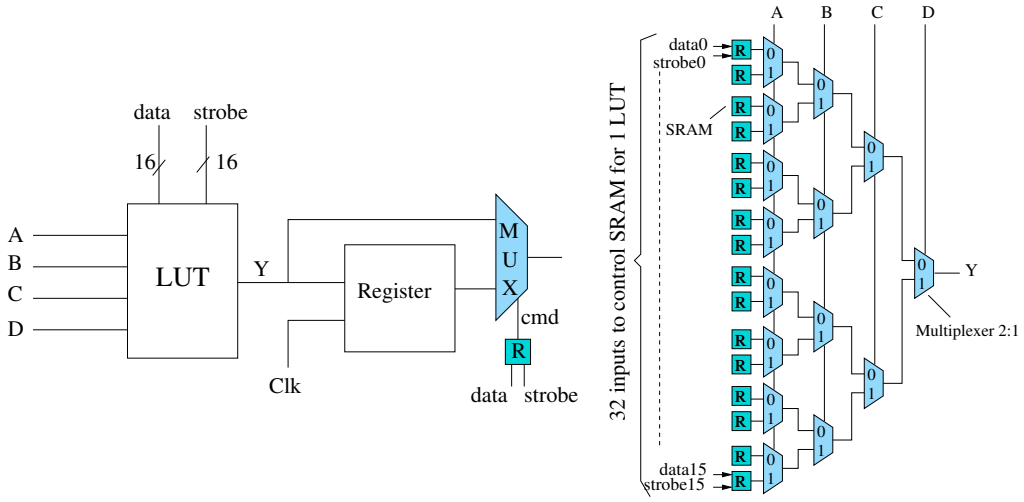


FIGURE 7 – a. Structure d'un BLE b. Structure d'une LUT

Méthodologie

Le *cluster* est divisé en 3 blocs principaux : un bloc logique composé des BLEs, un bloc de connexion 'Down' composé des 4 Crossbars 'Down' et un bloc de connexion 'Up' qui est le crossbar 'Up'. La propagation des signaux avec SPR se fait à travers les 3 blocs du *cluster* comme le montre le graphe de la figure 9. En fait, dans chaque bloc, SPR a besoin des matrices PTMs de tous les Mux2s, ainsi que des matrices SPMs de la mémoire de configuration. Sur ces dernières, les éléments de la digonale valent 0.5 et ceux de l'anti-diagonale 0. Les matrices SPMs des entrées primaires sont pareilles. Par contre, les SPMs des entrées 'rebouclées' provenant des sorties (les trois les plus à gauche à l'entrée de chaque crossbar 'Down' sur la figure 6) ne peuvent être considérées de la même manière et doivent être estimées. Pour ce, on 'coupe' le rebouclage et on procède comme suit :

1. On choisit arbitrairement une certaine configuration pour la mémoire : 160 bits de sélection dans le bloc de connexion 'Down', 160 bits SRAM dans le bloc logique, et 48 bits de sélection dans le bloc de connexion 'Up'.

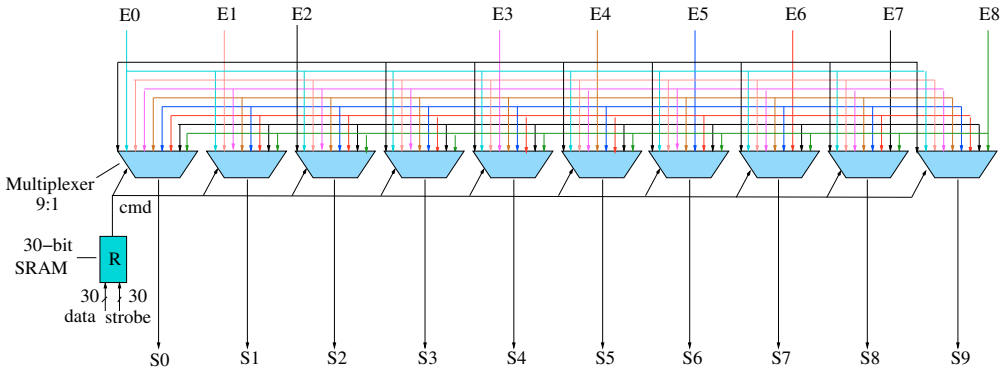


FIGURE 8 – Structure d'un crossbar 'Down'.

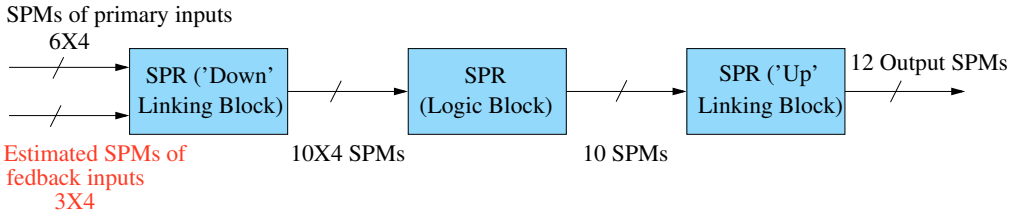


FIGURE 9 – SPR appliqué au *cluster*.

2. On génère une séquence binaire uniformément distribuée pour chaque entrée. La longueur de cette séquence est égale au nombre d'itérations avec laquelle la simulation à la prochaine étape va être faite.
3. On simule le *cluster* (en appliquant SPR) avec un grand nombre d'itérations (10^4), sans rebouclage et sans aucune erreur à la sortie des Mux2s.
4. On observe statistiquement la distribution binaire des sorties en comptant le nombre de zéros/uns dans chque sortie puis on moyenne par le nombre d'itérations, de façon à obtenir les probabilités des signaux. Ainsi, on peut déduire les matrices SPMs des entrées rebouclées qui vont servir pour démarrer l'analyse de la tolérance aux fautes du *cluster*.

Après l'application de SPR sur le *cluster*, on peut calculer la fiabilité de chaque signal de sortie. Enfin, la fiabilité du *cluster* est donnée par l'équation 2.

$$R_{cluster} = \prod_{i=1}^{12} R_{output_i} \tag{2}$$

En faisant varier la probabilité d'erreur (collage s ou inversion binaire $1-q$) à la sortie d'un Mux2 entre 0.1% et 5%, et pour différentes configurations de mémoire, la fiabilité du *cluster* est calculée et représentée sur la figure 10 comme étant une fonction convexe décroissante de la probabilité d'erreur. Les différentes courbes dépendent très peu du bitstream. Les courbes de collage sont presque identiques. On remarque que le *cluster* est plus vulnérable aux inversions binaires qu'aux fautes de collage. C'est pour cela que dans l'analyse qui suit, on prendra ce type d'erreur pour déterminer quels composants sont plus éligibles à être durcis.

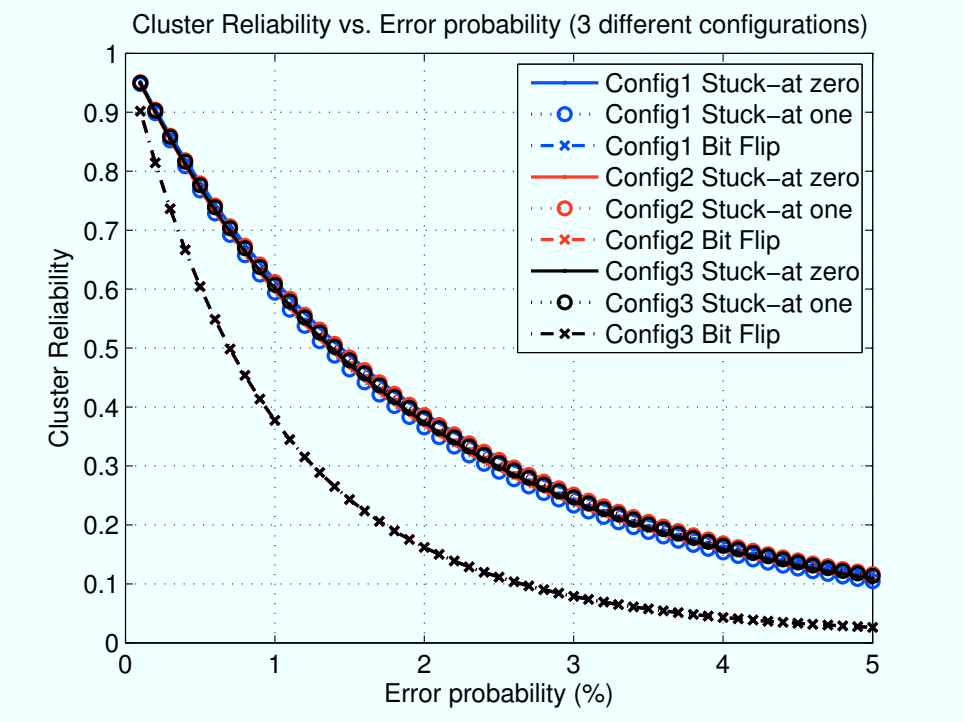


FIGURE 10 – Variation de la fiabilité du *cluster* avec la probabilité d'erreur.

Analyse des éléments critiques du *cluster*

Maintenant, nous nous intéressons à repérer les éléments critiques du *cluster* en vue d'un durcissement sélectif. Ainsi, les composants seront triés selon leur contribution à améliorer la fiabilité du *cluster*. En effet, nous gardons la même méthode d'analyse par SPR décrite précédemment, sauf que certains multiplexeurs seront désormais durcis (et donc pour lesquels $q = 1$). Nous prendrons des calculs précédents une valeur de fiabilité de référence notée R_{ref} par rapport à laquelle nous mesurons les améliorations de fiabilité. Pour ce, nous définissons le gain en fiabilité tel exprimé dans l'équation 3, où R est la fiabilité calculée après durcissement d'un composant du *cluster*.

$$G = \frac{R}{R_{ref}} \tag{3}$$

Nous commençons par durcir tous les multiplexeurs d'un BLE dans le bloc logique, ensuite tous les Mux2s d'un Mux9 dans le crossbar 'Down', et tous les Mux2s d'un Mux10 dans le crossbar 'Up'. Entre Mux10s, aucune différence de gain n'est notée. Par contre, nous remarquons que les 8 premières LUTs (celles tout à droite sur la figure 6, recevant en entrée les 8 premières sorties des Mux9s) réalisent moins de gain que les 2 dernières. Nous remarquons aussi une différence de gain entre les Mux9s dans chaque crossbar 'Down', puisque leurs sorties $S8$ et $S9$ sont connectées aux entrées des deux dernières LUTs. En réalité, cette différence de gain au niveau des deux blocs en amont est due à l'asymétrie de l'architecture du Mux10 du bloc en aval. Cette architecture est réprésentée sur la figure 6(a). Le 5^{eme} Mux2 sur la figure reçoit en entrée les sorties des 2 dernières LUTs et est contrôlé par le bit de sélection $c3$ ayant un poids plus important que $c1$ contrôlant les Mux2s du premier niveau lesquels reçoivent en entrée les sorties des 8 premières

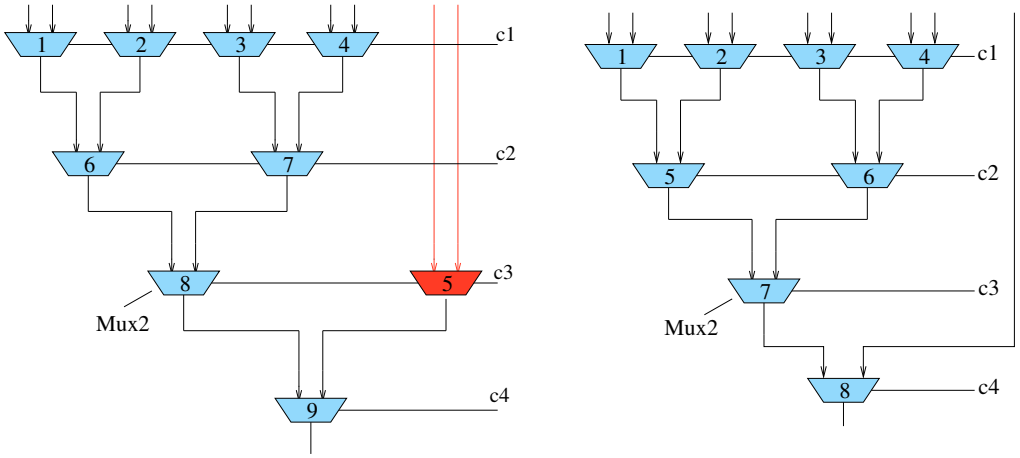


FIGURE 11 – a. Mux10 b. Mux9

LUTs. Ainsi, le chemin critique du *cluster* a pu être identifié : c'est celui passant par les 2 dernières LUTs.

Puisque les techniques de durcissement sont coûteuses en surface, nous pourrons robustifier seulement les Mux2s les plus éligibles à augmenter la tolérance aux fautes du *cluster*. Nous avons donc procédé à l'analyse de l'éligibilité au niveau Mux2. Dans le chemin critique, nous refaisons l'analyse par SPR du *cluster* en durcissant un Mux2 à la fois. Il en découle que le 15^{eme} Mux2 de la LUT, celui commandé par le bit de sélection de poids le plus fort, apporte le plus de gain en fiabilité. C'est donc le Mux2 le plus critique et le plus éligible à être robustifié.

Analyse de la fiabilité de la boîte d'interconnexion

Dans l'architecture FPGA en matrice de *clusters*, un *cluster* est entouré de quatre boîtes d'interconnexion. Les entrées/sorties du *cluster* sont directement routées par la boîte d'interconnexion dont la structure en arbre contient des réseaux 'upward' et 'downward'. L'architecture de la boîte de commutation est représentée dans la figure 12. En fait, comme le montre cette figure, les sorties du *cluster* entrent dans deux UMSBs coloriées en rouge. Chaque UMSB est constitué de trois multiplexeurs 6 :1 (Mux6). Les sorties des multiplexeurs sont contrôlées par les bits de sélection qui font partie du bitstream complet chargé en programmant le FPGA pour une application donnée. Ensuite, les sorties de la UMSB sont dirigées à deux types de DMSBs. La DMSB2 routerait les sorties de la UMSB au même *cluster* de nouveau, ou à un autre. La DMSB1 dirigerait ces signaux à une autre boîte de commutation. Chaque DMSB2 est composée de quatre multiplexeurs 4 :1 et chaque DMSB1 contient cinq multiplexeurs 5 :1. Deux fois le nombre de DMSB1s définit la largeur de canal qui est le nombre d'entrées/sorties entre deux boîtes d'interconnexion adjacentes.

La même analyse de fiabilité par SPR appliquée au *cluster* a été effectuée sur la SB, dans le même but d'identifier les éléments les plus critiques. L'analyse de criticité révèle que la UMSB est le bloc le plus éligible à être durci, notamment le Mux2 commandé par le bit de sélection de poids fort.

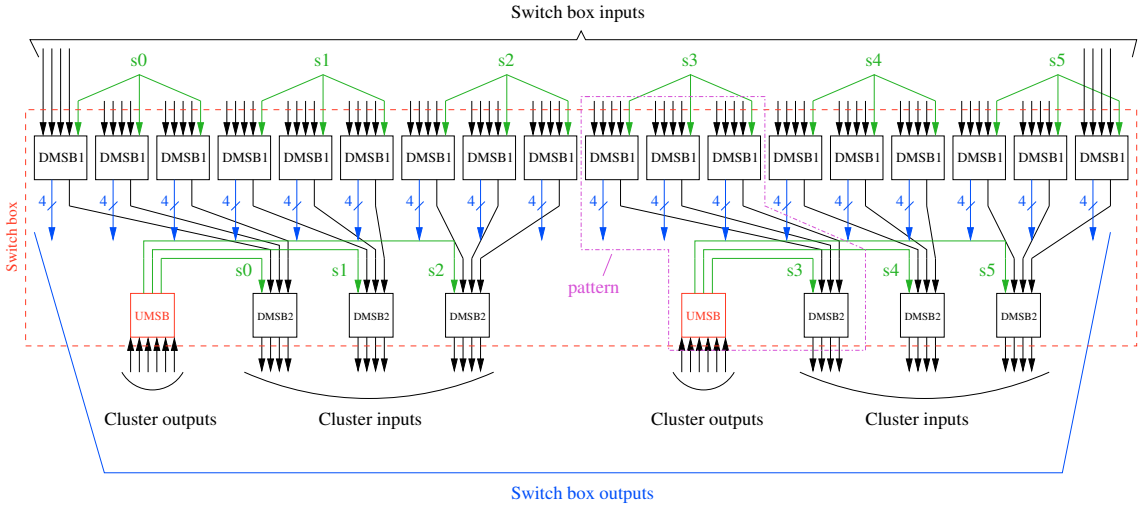


FIGURE 12 – Structure de la boîte de commutation

Durcissement des blocs de base

L'analyse de la fiabilité résumée à la section précédente a révélé que les composants les plus éligibles à être durcis sont :

- La Lut-4 du chemin critique dans le *cluster*.
- La UMSB (Mux6) dans la boîte d'interconnexion.

Nous avons adopté différentes approches pour leur durcissement. Pour le durcissement de la LUT, nous avons recouru à la redondance modulaire. Comme le réseau d'interconnexion occupe la majorité des ressources dans le FPGA, nous avons employé pour la UMSB une technique de durcissement n'engendrant pas de surcoût matériel.

Durcissement du *cluster*

Présentation de la nouvelle architecture de BLE

L'analyse de criticité du *cluster* a mis en évidence les contributions différentes des Mux2s à l'amélioration de la fiabilité, selon leur niveau hiérarchique (on appelle niveau hiérarchique l'ensemble des Mux2s commandés par un même bit de sélection). Prenons l'exemple de la LUT-4. Si le dernier Mux2 est défectueux, la sortie sera forcément fausse. Par contre, si l'un des Mux2s du niveau hiérarchique précédent est défectueux, on a quand-même 50% de chance d'avoir un résultat correct en sortie. Ainsi, comme les Mux2s ne sont pas de la même importance, l'architecture conventionnelle de la LUT est appelée *hiérarchique*.

Une des techniques de durcissement couramment utilisée dans l'état de l'art est le TMR, consistant à tripliquer un module et à mettre en aval un voteur à majorité binaire. En effet, l'idée derrière la nouvelle architecture de LUT est de 'casser' la hiérarchie en ayant le même nombre de Mux2s par niveau, et engendrant par là un moindre surcoût que le TMR. La figure 13(b) montre la nouvelle architecture pour la LUT-4, baptisée *Butterfly* à cause des interconnexions entre les niveaux de Mux2s rappelant des papillons. Et la régularité de ces motifs d'interconnexion est vérifiée si on augmente le nombre d'entrées de la LUT. Au dernier niveau, on aura autant de sorties que le nombre de Mux2s.

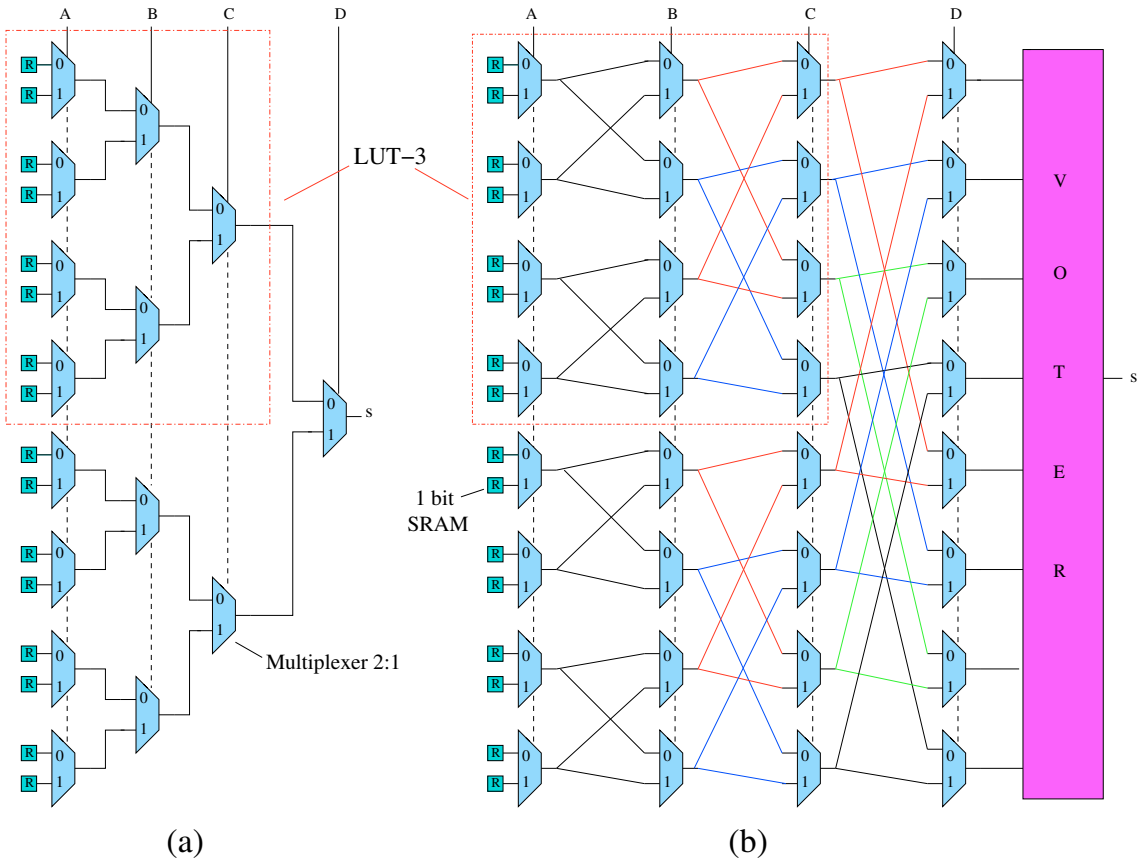


FIGURE 13 – LUT-4 (a) Hiérarchique (b) Butterfly

Par conséquent, un voteur est nécessaire pour décider de la sortie finale. Parmi les structures de voteurs étudiées dans cette thèse, nous avons retenu l'architecture optimisée en *CarryAdd*. Si η est le nombre d'entrées au voteur, le *CarryAdd* compte, d'une façon intelligente, le nombre de bits à 1 pour vérifier s'il dépasse $\frac{\eta}{2}$. Un autre voteur intéressant était le voteur *Cascade-TMR*, composé de voteurs TMR déployés en cascade. Le nombre de voteurs TMR requis dépend du nombre d'entrées. Par exemple, pour 8 entrées, 4 voteurs TMR sont nécessaires.

Comme nous voudrions trouver un bon compromis entre fiabilité et surcoût en surface, nous avions eu l'idée d'alléger l'architecture Butterfly en réduisant le nombre de Mux2s au dernier étage à seulement 3. Ainsi, nous recourrions à un simple voteur TMR. La figure 14 l'architecture de la version modifiée de la Butterfly.

Comparaison d'architectures de BLEs

Remarques préliminaires Les architectures de LUT-4 que nous avons comparées sont :

- La hiérarchique (Hr), ayant 15 Mux2s ;
- La hiérarchique avec triplication du Mux2 le plus critique au dernier étage (Hr-3m), ayant 17 Mux2s ;
- La Butterfly originale (Bf), ayant 32 Mux2s ;
- La Butterfly modifiée (Bf-M), ayant 25 Mux2s ;
- La hiérarchique tripliquée (TMR-Hr), ayant 45 Mux2s ;

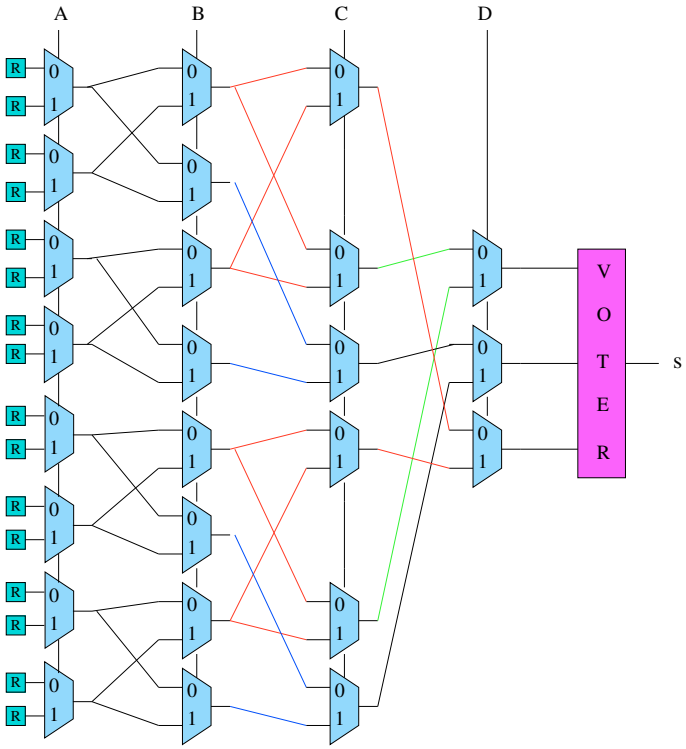


FIGURE 14 – Architecture de Butterfly modifiée pour la LUT-4.

Les architectures de voteurs que nous avons étudiées sont :

Pour la synthèse de la structure Bf, nous avons pris en compte les voteurs *CarryAdd* et *Cascade-TMR*. Quant à la synthèse des architectures Bf-M, Hr-3m et TMR-Hr nous avons considéré de différents voteurs TMR : le voteur standard à majorité binaire (*Stand*) et deux autres voteurs tolérants aux fautes. Le premier des deux est appelé *Ban* [1]. Ce voteur s'est avéré tolérant aux fautes simples. Le deuxième voteur TMR tolérant aux fautes [2], nécessitant plus de *hardware* que le premier, est appelé *Kshir*. Il a été démontré dans [2] que si toutes les entrées du voteur sont correctes, alors la sortie suit l'entrée indépendemment des fautes pouvant avoir lieu à l'intérieur de l'arbitre.

Les critères de comparaison sont :

- La surface, la fréquence maximale et la consommation de puissance utilisant la technologie CMOS STM 65nm et la librairie de cellules standard CORE65LPSVT qui est optimisée pour les applications à faible puissance. La tension d'alimentation est $1.2V$, la température de jonction nominale est de $25°C$ et une description Verilog des architectures a été utilisée comme entrées pour le synthétiseur de circuits ASICs Cadence Encounter RTL Compiler. La puissance dynamique est calculée à une fréquence d'horloge de $100MHz$;
- Le nombre de bits de configuration requis dans la mémoire SRAM qui est supposée exempte de fautes ;
- Le masquage logique de k fautes simultanées pouvant avoir lieu dans n'importe quel Mux2, considérant toutes les configurations possibles de bits de sélection. Nous avons varié k de 1 à 4. En l'occurrence, tous les Mux2s sont faillibles. Et un Mux2 est défaillant quand sa sortie est inversée. Le masquage logique est calculé à partir de la plateforme FIFA décrite dans [3] ;

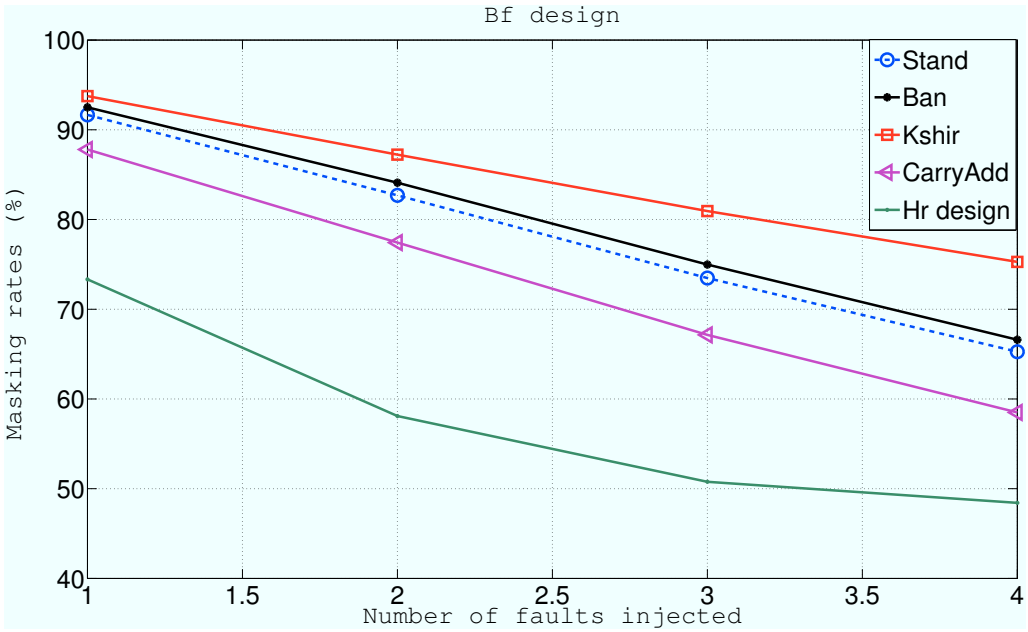
– La fiabilité des LUTs où chaque Mux2 est caractérisé par sa fiabilité q. Sans perte de généralité, nous avons pris $q = 0.96$ comme exemple pour calculer la fiabilité de la LUT-4. En utilisant les valeurs de masquage logique des fautes multiples simultanées obtenues avec la FIFA, la fiabilité est calculée selon le modèle PBR décrit dans [4].

Injection de fautes et fiabilité Avant de procéder à l'injection de fautes, nous avons synthétisé les architectures de BLEs, de sorte à injecter les fautes proportionnellement à la surface des composants logiques. En utilisant la plateforme FIFA, nous avons injecté jusqu'à 4 inversions binaires simultanées dans nos structures de LUT étudiées. Les résultats pour les architectures de BLEs employant les différents circuits d'arbitrage sont montrées aux figures 15 et 16. Les taux de masquage logique de chaque structure de BLE sont confrontés avec ceux de la Hr qui sont marqués avec un trait épais continu sur chaque sous-figure. Nous remarquons que la TMR-Hr réalise les gains les plus hauts en taux de masquage pour n'importe quel nombre de fautes injectées, et ce fait est pratiquement indépendant du choix du voteur. Les gains les plus bas en taux de masquage sont réalisés par la Hr-3m où l'utilisation du voteur *Kshir* augmente la capacité de masquage par 3% tout au mieux. Le gain en masquage peut être augmenté d'une manière plus importante en employant le voteur *Kshir* pour la Bf. En l'occurrence, les gains en masquage sont de l'ordre de 30% approchant ceux de la TMR-Hr. Le voteur *CarryAdd* est le plus vulnérable aux inversions binaires.

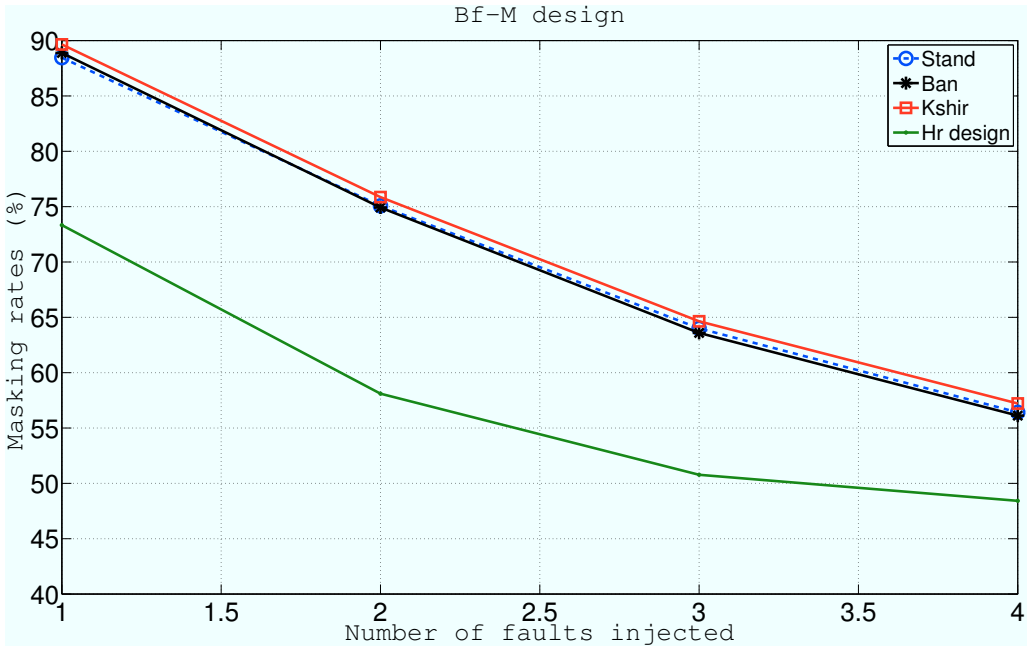
La fiabilité est toujours calculée selon le modèle PBR, et est exprimée dans l'équation 4. Les résultats de calcul de la fiabilité pour les BLEs sont montrés dans les figures 17 et 18. La fiabilité de chaque architecture est comparée à celle de la Hr qui est égale à 85.81%. Puisque notre premier objectif est d'améliorer la fiabilité de la structure de BLE conventionnelle, nous avons pu éliminer dès cette étape les architectures qui se sont avérées moins fiables que la Hr : la Hr-3m avec n'importe quel voteur et la Bf utilisant soit une cascade de *Kshir* soit le voteur *CarryAdd*. En ce qui concerne la Hr-3m, la technique TMR est appliquée uniquement sur le Mux2 le plus critique. Le reste de ses Mux2s ne sont pas protégés contre les fautes, ce qui explique les faibles gains en masquage réalisés par la Hr-3m. Donc, les termes $R_k, k \in [1, 4]$ dans l'équation 4 ne parviennent pas à compenser la faible valeur de R_0, correspondant à un scénario où aucune faute n'a lieu. En effet, pour n'importe quelle architecture redondante, le terme R_0 est toujours plus faible que pour une architecture non-redondante. D'autre part, l'arbitre d'un système TMR/NMR est un composant critique dont la fiabilité détermine les performances de tout le système. Si la probabilité d'erreur de l'arbitre dépasse un certain seuil, la technique TMR/NMR dégrade la fiabilité initiale et devient ainsi inutile. Et plus l'arbitre est grand en surface, plus sa probabilité d'erreur est grande. Ceci peut être vérifié dans les figures 17 et 18, où dans chaque structure l'utilisation du voteur le moins complexe améliore la fiabilité. Pour la Bf, les voteurs *Kshir* cascadé et *CarryAdd* sont si complexes qu'ils détériorent la fiabilité de toute l'architecture.

$$r = \frac{1}{2^N} \sum_{k=0}^{w} f_k(q) m_k = \frac{1}{2^N} \sum_{k=0}^{w} R_k \tag{4}$$

Résultats de synthèse et métrique de qualité Les architectures durcies retenues de la précéndente sous-section ont été synthétisées avec Cadence RTL compiler sous les

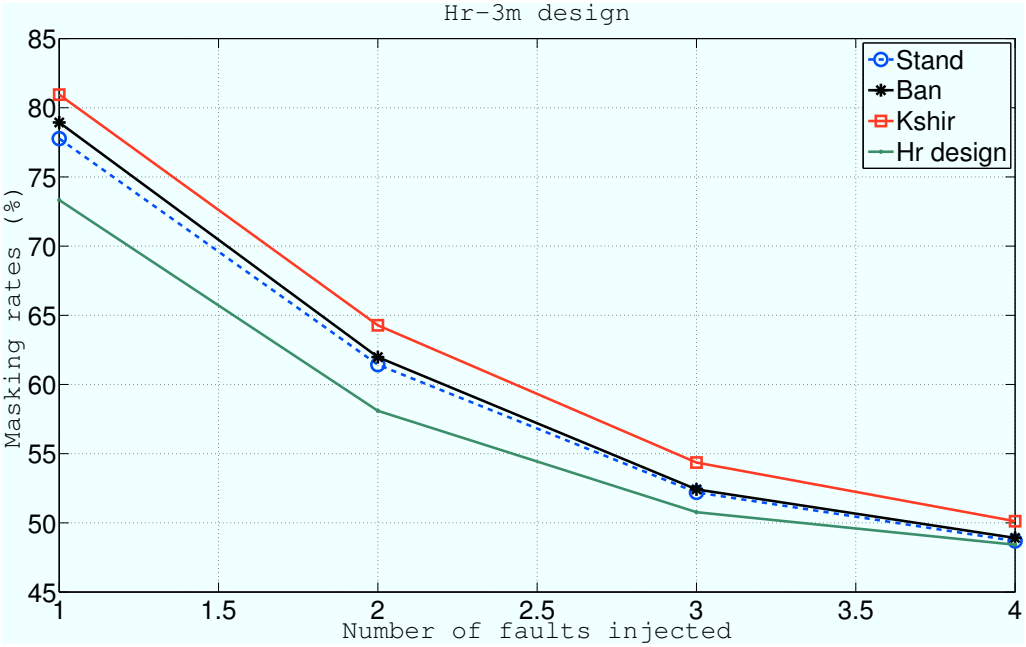


(a) Bf

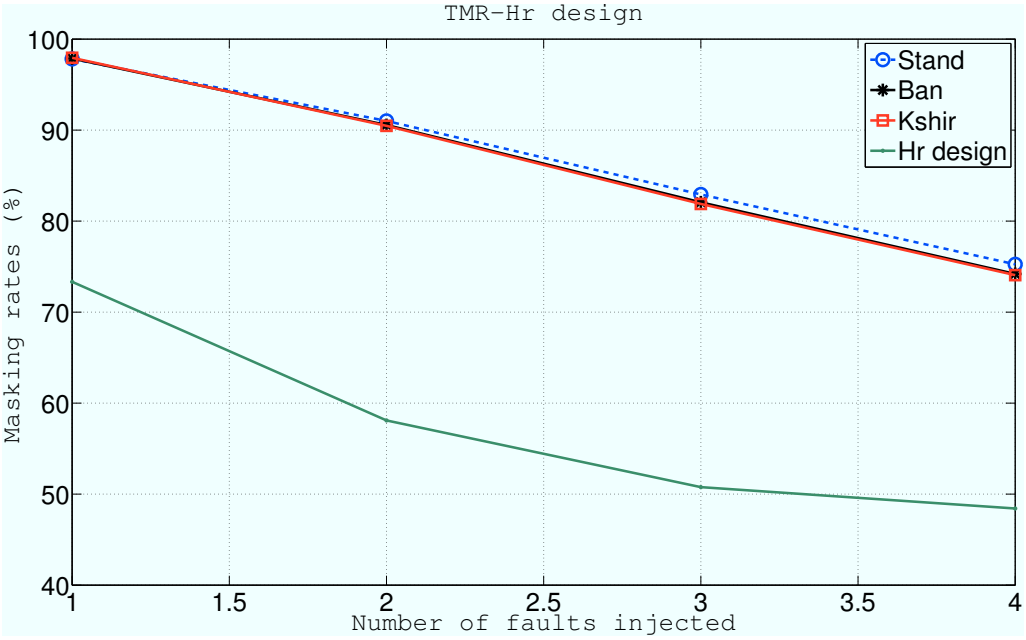


(b) Bf-M

FIGURE 15 – Taux de masquage des BLEs durcis (1)

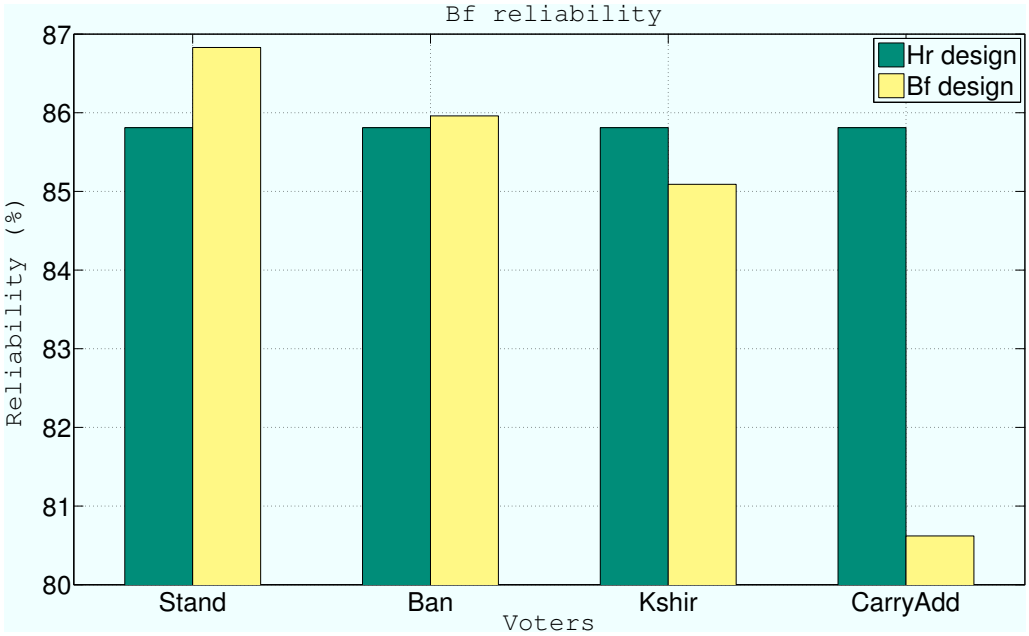


(a) Hr-3m

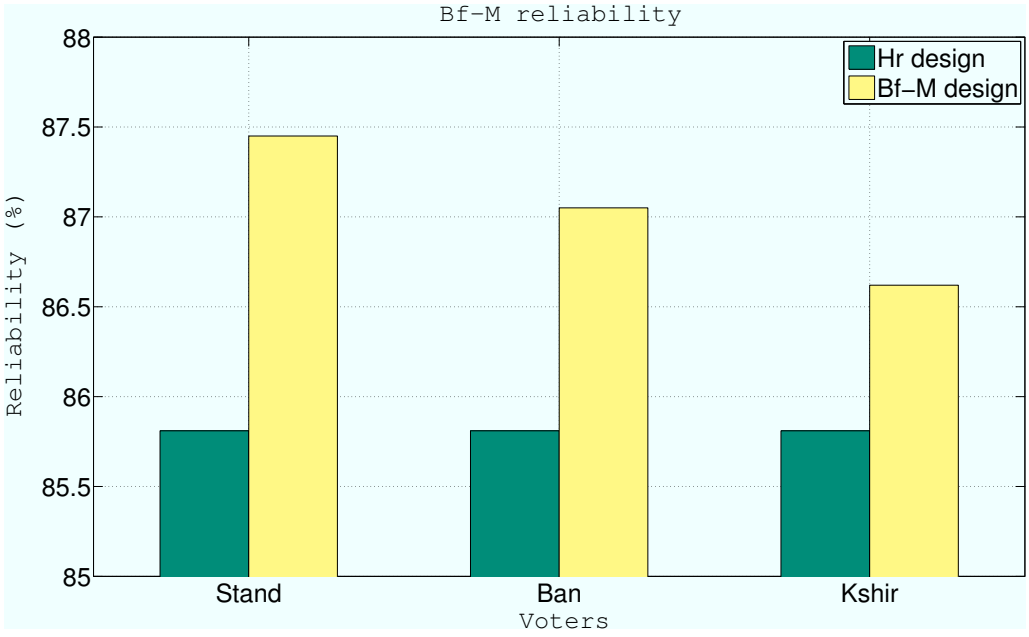


(b) TMR-Hr

FIGURE 16 – Taux de masquage des BLEs durcis (2)

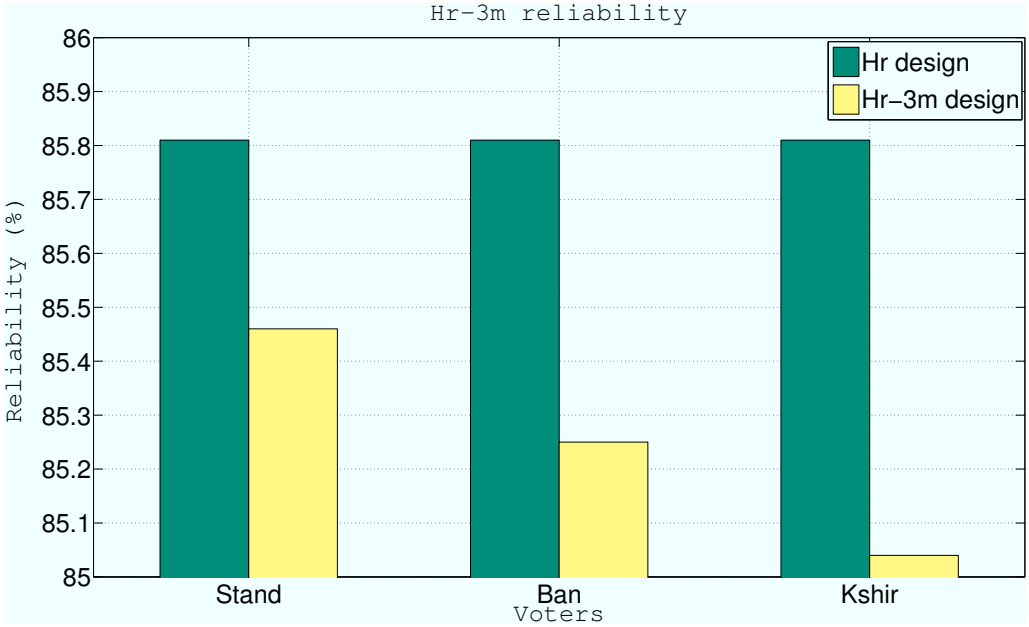


(a) Bf

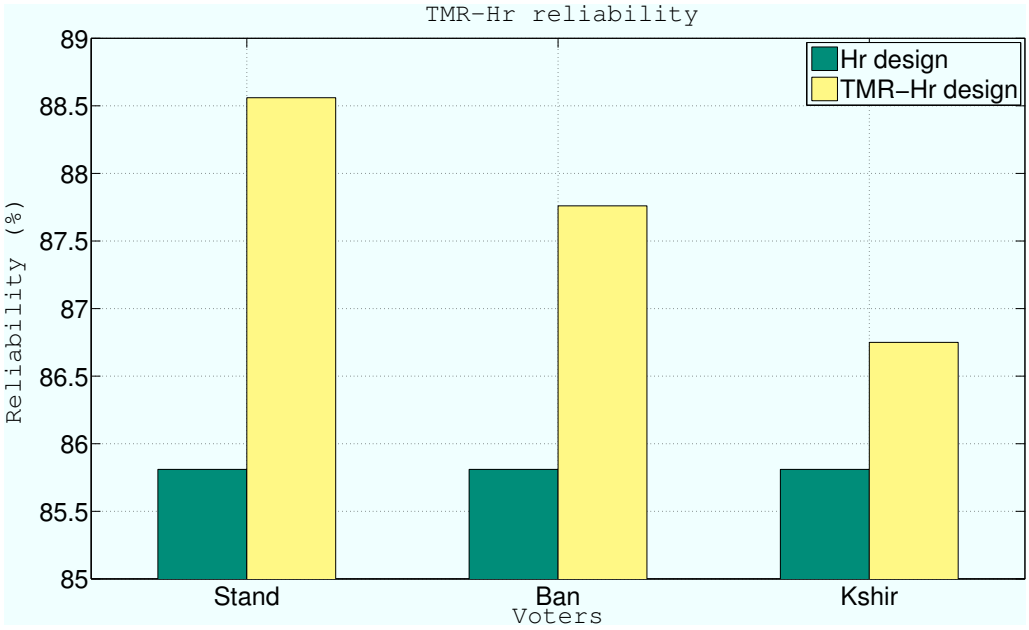


(b) Bf-M

FIGURE 17 – Fiabilité des BLEs durcis (1)



(a) Hr-3m



(b) TMR-Hr

FIGURE 18 – Fiabilité des BLEs durcis (2)

TABLE 1 – Résultats de synthèse pour les architectures de LUT-4.

	Surface totale (μm^2)	**Chemin critique (ps)**	**Puissance dynamique (μW)**	**Bits de configuration**
Hr	165.18	371.4	3.52	16
Bf Stand	360.22	642.4	7.10	16
Bf Ban	389.93	913.0	7.11	16
Bf-M Stand	266.78	514.7	5.52	16
Bf-M Ban	274.20	627.3	5.53	16
Bf-M Kshir	288.22	711.3	5.54	16
TMR-Hr Stand	492.33	480.0	9.72	48
TMR-Hr Ban	499.76	606.5	9.73	48
TMR-Hr Kshir	513.78	675.4	9.74	48

mêmes conditions susmentionnées. Ensuite, une métrique de qualité exprimant le compromis entre gain en fiabilité et différents surcoûts a été définie pour classer les architectures de LUT-4 et sélectionner la meilleure.

Résultats de synthèse :

Le tableau 1 donne les résultats de synthèse pour les architectures de LUT-4 retenues, confrontées à la structure hiérarchique. La dernière colonne ne fait pas partie des résultats de synthèse. Seule la partie combinatoire a été synthétisée.

On peut remarquer qu'indépendemment du voteur l'architecture Bf-M engendre les surcoûts les plus faibles en termes de surface, délai, consommation de puissance et mémoire. Elle est suivie de la Bf puis de la TMR-Hr. En outre, dans toutes les structures, l'utilisation d'un voteur TMR standard réduit les surcoûts, étant donné qu'il est synthétisé en une seule cellule du type HS65_LS_PAO2X4. En revanche, les voteurs TMR *Ban* et *Kshir* sont constitués respectivement de deux et de quatre cellules de la librairie standard CORE65LPSVT.

Métrique de qualité :

Afin de comparer les architectures par rapport à celle de référence qui est la structure Hr, nous avons défini quelques métriques :

- Le gain en fiabilité défini comme $G_r = \frac{r}{r_{Hr}}$ où r et r_{Hr} sont les fiabilités de l'architecture LUT-4 considérée et de la Hr respectivement ;
- La dégradation de performances définie comme $O_T = \frac{T}{T_{Hr}}$ où T et T_{Hr} sont les chemins critiques de l'architecture LUT-4 considérée et de la Hr respectivement ;
- Le surcoût en surface défini comme $O_A = \frac{A}{A_{Hr}}$ où A et A_{Hr} sont les surfaces totales de l'architecture LUT-4 considérée et de la Hr respectivement ;
- Le surcoût en puissance défini comme $O_P = \frac{P}{P_{Hr}}$ où P et P_{Hr} sont les consommations de puissance totale (de fuite et dynamique) de l'architecture LUT-4 considérée et de la Hr respectivement ;
- le surcoût en mémoire défini comme $O_M = \frac{M}{M_{Hr}}$ où M et M_{Hr} sont les tailles de la mémoire SRAM de l'architecture LUT-4 considérée et de la Hr respectivement.

La métrique de qualité globale Q est exprimée à l'équation 5. Les valeurs $w_i, i \in [1, 5]$

TABLE 2 – Calcul de la métrique globale Q.

LUT-4	Métrique Q
Bf-M Stand	0.280
Bf-M Ban	0.220
Bf-M Kshir	0.180
Bf Stand	0.130
Bf Ban	0.085
TMR-Hr Stand	0.031
TMR-Hr Ban	0.024
TMR-Hr Kshir	0.021

sont des pondérations appliquées aux métriques, choisies par le concepteur pour favoriser la fiabilité par rapport aux surcoûts ou vice versa, selon les exigences de l'application cible.

Dans l'exemple présenté ci-après, la même importance est accordée à tous les paramètres. Donc, toutes les pondérations dans Q sont égales. Les résultats de calcul pour les architectures LUT-4 sont montrés au tableau 2, où les architectures sont déjà classées dans l'ordre suivant les valeurs décroissantes de Q.

Finalement, puisque la métrique globale Q a la valeur la plus élevée pour la Bf-M utilisant le voteur TMR standard, cette architecture a été sélectionnée comme donnant le meilleur compromis entre gain en fiabilité et différents surcoûts.

$$Q = \frac{G_r^{w_1}}{O_A^{w_2} O_T^{w_3} O_P^{w_4} O_M^{w_5}} \tag{5}$$

Durcissement de la boîte d'interconnexion

Comme la UMSB est un Mux6, nous en avons exploré différentes architectures en assemblant différemment les cellules standard de la librairie STM CORE65LPSVT. Nous avons eu recours à l'outil Mentor Graphics Tessent CellModelGen [5] qui injecte des défauts de façon réaliste. En effet, cet outil utilise la netlist extraite d'un circuit pour modéliser les défauts au niveau transistor.

La librairie contient Mux2, Mux3, Mux4 comme sous-modules pouvant constituer le module du Mux6. Une netlist Verilog structurelle a été établie pour chaque architecture de Mux6 constituée à partir de ces briques de base. Ensuite, la synthèse a été effectuée sous Cadence RTL Compiler. Les simulations avec CellModelGen ont révélé que la structure Gr43, composée seulement d'un Mux4 et d'un Mux3, avait les taux de défaillance les plus faibles comparés aux autres assemblages. En outre, il est l'un des assemblages les plus compacts. Donc, le Gr43 est clairement le meilleur choix à faire selon cette approche de synthèse intelligente qui n'engendre pas de surcoût matériel.

Robustification des blocs de base au niveau transistor

Nous avons raffiné le durcissement au niveau transistor en étudiant des solutions d'architectures de Mux2 en *single-ended* et *dual-rail*.

Solutions de Mux2 étudiées

Nous avons proposé deux architectures de Mux2 basées sur la logique *dual-rail*. La première utilise la logique DCVS. La deuxième est basée sur une nouvelle logique que nous avons baptisée *Cross Logic* (CL). Quant à la solution *single-ended*, afin de proposer un Mux2 tolérant aux défauts, nous avons procédé de deux manières différentes.

Le Mux2 en tristate D'abord, nous avons exploré quatre alternatives de conception de Mux2 en changeant l'assemblage interne des transistors, utilisant la technologie CMOS STM 65nm et la librairie de cellules standard CORE65LPSVT.

Trois architectures communes de Mux2 ont été étudiées (standard en portes de transmission, portes NAND et *full custom*). Le Z-Mux proposé est la quatrième architecture de Mux2, principalement composée de 2 cellules tristate. La figure 19 montre son schéma en transistors.

La deuxième façon 'classique' de concevoir un Mux2 tolérant aux défauts est la technique TMR. Cette dernière a été appliquée à la cellule standard compacte Trans-mux composée de portes de transmission, en utilisant deux différents types d'arbitre : la cellule du voteur TMR standard, et un voteur TMR tolérant aux fautes introduit dans [1].

Les résultats de simulation ont montré que le Z-Mux était le plus tolérant aux défauts affectant les transistors.

Le Mux2 en DCVS La cellule Trans-Mux de la librairie standard STM CORE65LPSVT (MUX2X4) est composée de 2 portes de transmission CMOS et de 3 inverseurs, où les signaux $in0$ et $in1$ sont les entrées du Mux2, $s0$ est le bit de sélection et out est le signal de sortie. Nous avions montré que le Z-Mux constitué de deux cellules tristate prises de la même librairie était plus robuste que le Mux2 standard contre les défauts simples sur les transistors. Par contre, il compte 26 transistors, ce qui est plus que le double du nombre de transistors du standard. Par conséquent, nous avions pensé à la logique différentielle comme un moyen d'améliorer la robustesse et de réduire le surcoût en surface en même temps.

Dans les circuits différentiels, l'information est traitée et transmise d'une manière redondante et complémentaire, offrant une résistance intrinsèque accrue aux défaillances. La figure 20 montre le schéma du Mux2 en DCVS proposé, qui est composé de 8 transistors NMOS et de 2 transistors PMOS croisés. L'information et son complémentaire (signaux out et $\overline{out}$) sont traitées par deux chemins différents. Pour chaque combinaison d'entrée, seul un chemin propage la valeur logique 0 depuis la masse, et polarise ainsi le PMOS dont la grille lui est connectée, amenant la sortie complémentaire à une valeur logique 1. En fait, nous avons remarqué que le dimensionnement du transistor PMOS était un paramètre important pour propager les signaux corrects et donc pour améliorer la tolérance aux défauts. Nous étudierons cet aspect dans les paragraphes suivants. En réalité, on s'attend à ce que cette architecture soit plus tolérante aux défauts de type circuit-ouvert (*open*) plutôt qu'aux courts-circuits (*bridges*). Prenons l'exemple de $sel = 1$, $A = 0$ et $B = 0$, auquel cas $out = 0$ grâce à 3 transistors fermés. Si l'un des transistors contrôlés par $\overline{A}$ et sel est constamment ouvert, la sortie sera correcte tout de même. Donc, considérant cette combinaison des entrées en particulier comme exemple, il y a une probabilité de $\frac{2}{3}$ d'avoir une sortie correcte en présence d'un défaut de type *open*. Les simulations ont effectivement prouvé que l'architecture Mux2 proposée était plus tolérante aux *opens* qu'aux *bridges*, conformément aux attentes théoriques.

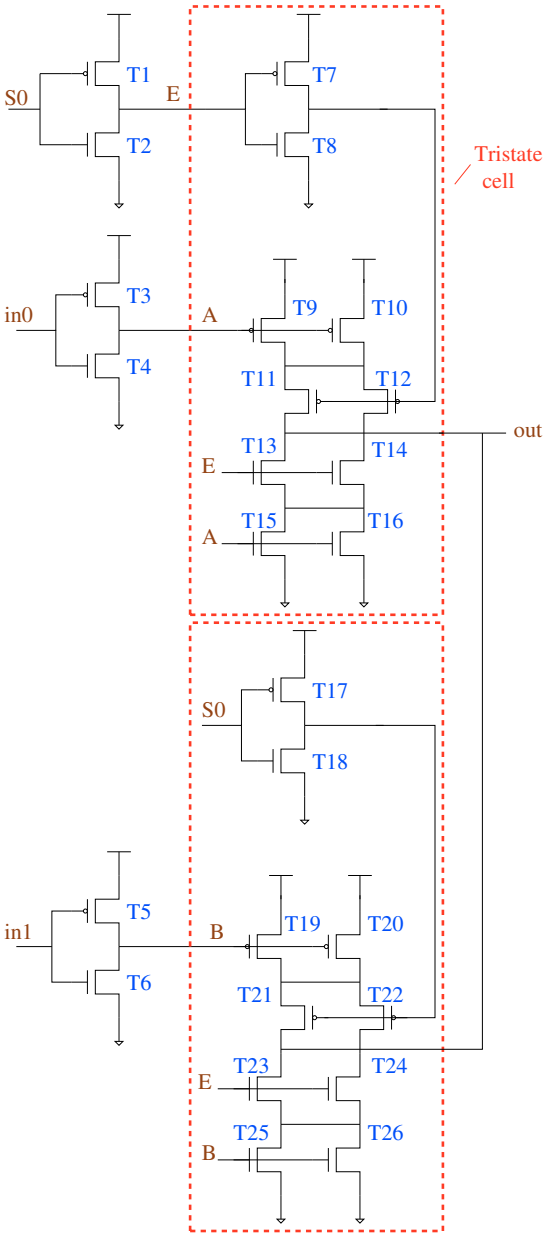


FIGURE 19 – L'architecture Z-Mux.

Le Mux2 en CL Nous avons proposé de construire des cellules tolérantes aux défauts en utilisant des cellules CMOS standard. L'idée principale est d'obtenir la sortie à partir de deux chemins différents de sorte que si un chemin est défectueux, le deuxième reste intact et corrige la valeur donnée par le chemin défaillant. Pour réaliser cet objectif, nous avons inventé une nouvelle logique appelée *Cross Logic* (CL), inspirée de la DCVS, bien que leurs implémentations soient différentes.

Une cellule CL est composée de la cellule CMOS correspondante, son dual et deux inverseurs CMOS croisés. Nous appelons cellule duale celle qui génère les sorties complémentaires à partir des entrées complémentaires. La figure 21 montre le schéma général d'une cellule CL où Q et $\overline{Q}$ correspondent au signal de sortie et son complémentaire, res-

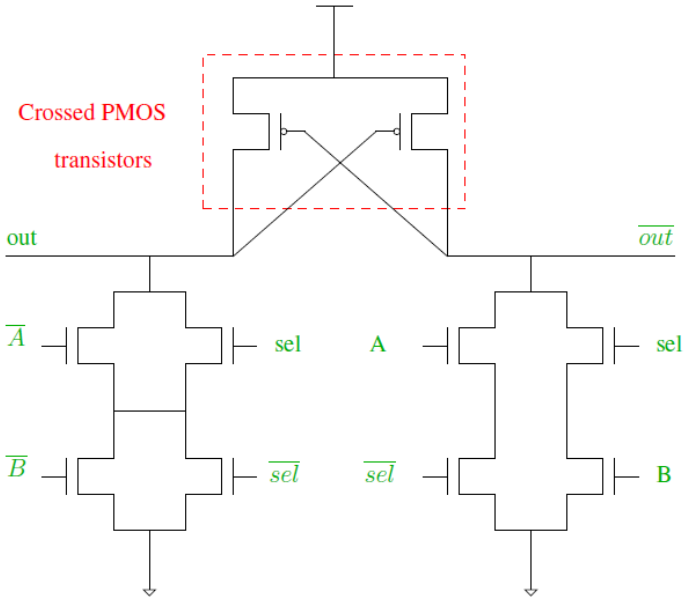


FIGURE 20 – Le Mux2 en DCVS.

pectivement. OP^* est le dual de l'opérateur OP. Par exemple, la porte NOR est la duale de la NAND. La fonction Mux2 est sa propre duale, c-à-d que si $Q = MUX2(sel, A, B)$ alors $\overline{Q} = MUX2(\overline{sel}, \overline{B}, \overline{A})$. Pour des raisons de simplicité, nous expliquons CL avec l'exemple de la NAND.

La figure 22 représente la cellule NAND en CL. A droite, la porte NAND en CMOS génère le signal de sortie Q. La porte NOR en CMOS à gauche génère le signal complémentaire de la sortie $\overline{Q}$. La paire d'inverseurs croisés permet la génération du signal de sortie depuis son complémentaire et vice versa. Maintenant, supposons qu'un défaut *stuck-open* vienne affecter le transistor T_1. Si $A = 1$ et $B = 0$, la sortie Q de la porte NAND en CMOS serait normalement à l'état haute impédance. Néanmois, grâce aux inverseurs croisés, la valeur correcte de Q est imposée par $\overline{Q}$. Bien entendu, le même raisonnement peut se faire indépendamment de l'endroit où le défaut *stuck-open* a lieu. D'autre part, si la valeur de Q ou de $\overline{Q}$ est affectée par n'importe quel type de défaut *bridge* (port bridge, T_{on}...), il peut être corrigé par son complémentaire. Par contre, on pourrait se demander pourquoi ce n'est pas plutôt la valeur incorrecte qui ne viendrait pas perturber la valeur correcte. En effet, cela dépend de différents paramètres dont le dimensionnement des inverseurs croisés ou leurs régions de fonctionnement.

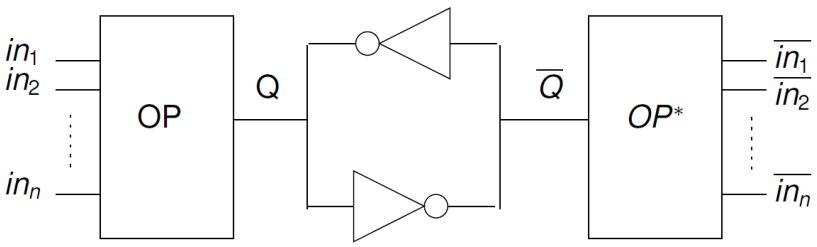


FIGURE 21 – Schéma général de CL.

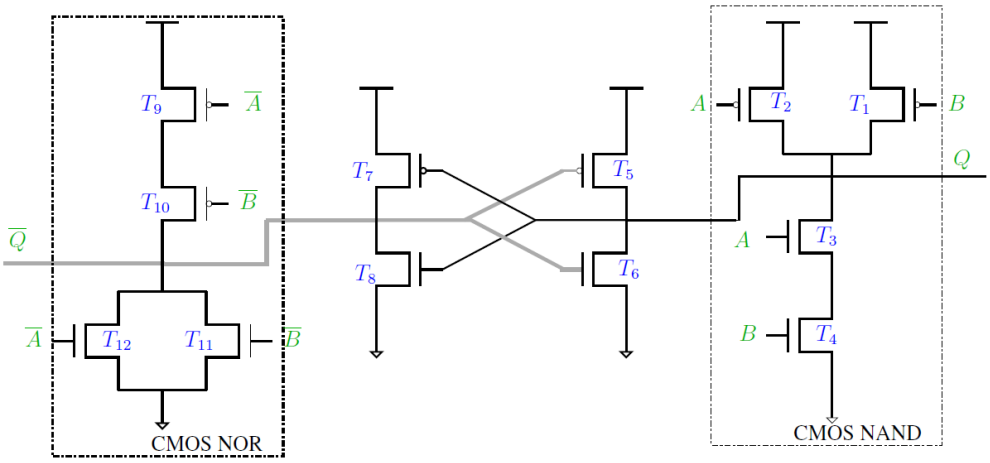


FIGURE 22 – La porte NAND en CL.

Comparaison des différentes solutions

Nous avons comparé les architectures de Mux2 les plus intéressantes, à savoir le Z-Mux, le DCVS, le CL et la solution TMR, en termes de fiabilité et de surcoût (non seulement la surface, mais aussi le délai et la consommation de puissance). Ensuite, nous avons pu sélectionner l'architecture la plus convenable selon une métrique de qualité bien définie.

Analyse de tolérance aux défauts Nous avons effectué l'analyse de tolérance aux défauts à l'aide de l'outil CellModelGen.

Le tableau 3 récapitule les résultats de cette analyse pour les architectures de Mux2 susmentionnées. La première colonne rapporte le taux de masquage global r. r_{opens} et $r_{bridges}$ sont les taux de masquage dûs aux *opens* et aux *bridges* respectivement. La solution TMR réalise le gain le plus élevé en taux de masquage. Elle est suivie du CL, de l'architecture Z-Mux et après du DCVS. Si l'on se concentre sur les gains en taux de masquage dûs aux *opens*, le gain du DCVS dépasse celui du Z-Mux mais reste toujours devancé par les architectures CL et TMR. Il est important de noter que, contrairement à toutes les autres architectures, le Mux2 en TMR est plus robuste aux *bridges* qu'aux *opens*.

TABLE 3 – Tolérance aux défauts des architectures de Mux2 étudiées.

Mux2	r **(%)**	r_{opens} **(%)**	$r_{bridges}$ **(%)**
Standard	73.34	76.57	71.56
DCVS	76.48	87.50	73.72
CL	83.43	91.50	77.08
Z-Mux	82.11	84.52	79.88
TMR	92.14	90.00	94.29

Métrique de comparaison Dans cette sous-section, nous montrons les pénalités engendrées par les architectures durcies en termes de surface, consommation de puissance et

délai. Ensuite, nous définissons une métrique de qualité exprimant le compromis entre le gain en robustesse et les différents surcoûts, et permettant de classer les architectures de Mux2.

Le tableau 4 montre que le DCVS est l'architecture la plus compacte, étant 1.5 fois plus gros que le Mux2 standard, alors que le Z-Mux, le CL et la solution TMR sont approximativement 8, 9 et 10 fois plus gros que leur homologue standard, respectivement. Néanmoins, le DCVS a les pires caractéristiques en termes de délai et d'énergie moyenne. Il est 11 fois plus lent et consomme 4 fois plus de puissance que le standard. L'architecture la plus performante et la moins consommatrice en puissance est le Z-Mux.

TABLE 4 – Surface, délai et énergie des architectures de Mux2 étudiées.

Mux2	**Surface (μm^2)**	**Délai (ps)**	**Energie moyenne ($\mu W/GHz$)**
Standard	4.16	74.32	9.5467
DCVS	6.02	808	41.036
CL	36.78	97.78	23.786
Z-Mux	34.35	78.68	19.236
TMR	42.43	108.59	20.834

Puisque nous nous intéressons aux applications critiques sous des contraintes de surface, délai et consommation de puissance, nous avons besoin d'exprimer le compromis entre le gain en robustesse et les différents surcoûts dans la comparaison des architectures de Mux2. Donc, nous utilisons une métrique Q définie à l'équation 6, où les métriques suivantes sont définies par rapport à la structure standard :

- $Gr = \frac{r}{r_{stand}}$ est le gain global en robustesse ;
- $O_A = \frac{A}{A_{stand}}$ est le surcoût en surface ;
- $O_D = \frac{D}{D_{stand}}$ est le surcoût en délai ;
- $O_E = \frac{E}{E_{stand}}$ est le surcoût en énergie.

$$Q = \frac{G_r^{w_1}}{O_A^{w_2}.O_D^{w_3}.O_E^{w_4}} \tag{6}$$

Il est à noter que $w_i, i \in [1, 4]$ sont des pondérations choisies par le concepteur pour décider de l'importance relative de la robustesse et des différentes contraintes, selon les exigences de l'application. Plus Q est élevé, plus l'architecture est convenable.

Le tableau 5 donne un exemple où tous les paramètres sont de la même importance. Donc, toutes les pondérations sont égales. Nous pouvons voir que, dans ce cas-ci, l'architecture Z-Mux a la valeur de Q la plus élevée. Par conséquent, c'est l'architecture la plus convenable pour être sélectionnée comme structure durcie tenant compte des contraintes en surface, délai et énergie. Et comme cela a été dit précédemment, en fonction des exigences de l'application, on peut penser à définir une autre métrique focalisée uniquement sur les *opens* ou les *bridges*.

Robustification de la LUT Le gain global en robustesse qui peut être atteint dans une LUT (de taille N) utilisant des Mux2s durcis au lieu des standards est mis en exergue.

TABLE 5 – Métrique de comparaison des Mux2s.

Mux2	Q
DCVS	0.015
CL	0.039
Z-Mux	0.063
TMR	0.038

Ensuite, le choix le plus approprié d'architecture de Mux2 est discuté.

Pour différentes valeurs de N, nous avons calculé le taux de défaillance global de la LUT utilisant seulement des Mux2s standards et seulement des Mux2s durcis. Comme prévu, la LUT implémentée avec seulement des Mux2s TMR a toujours le taux de défaillance le plus bas. Donc, on est bien tenté de sélectionner la solution TMR pour le Mux2. Il ne faut pas oublier toutefois qu'un seul Mux2 en TMR a l'une des pires valeurs de métrique Q, 2.5 fois inférieure à celle du Z-Mux. Pour l'exemple de la LUT-4, le gain global en robustesse employant l'architecture Z-Mux est environ 2.5. Ce gain peut être doublé en utilisant la solution TMR. Néanmoins, les pénalités conséquentes en termes de surface, délai et consommation de puissance seront certainement lourdes, rendant vain le gain en robustesse. Ainsi, l'architecture de Mux2 la plus convenable est le Z-Mux.

Conclusion

Cette thèse s'est intéressée à la conception d'une nouvelle architecture de FPGA de type matrice de *clusters*, tolérante aux fautes. Pour ce faire, il fallait tout d'abord évaluer la fiabilité initiale de l'architecture de départ afin de pouvoir l'améliorer par la suite. Dans un premier temps, une méthodologie d'analyse de la fiabilité des blocs de base (*cluster* et boîte de commutation) a été proposée. Cette même méthodologie a permis d'identifier les composants les plus éligibles au durcissement, dans le cas d'un budget limité. Ensuite, en ce qui concerne l'amélioration de la fiabilité, différentes approches de durcissement ont été proposées à différents niveaux de granularité. Au niveau multiplexeur, une nouvelle architecture redondante a été proposée pour le bloc logique de base, et une méthode utilisant une synthèse intelligente orientée fiabilité a été appliquée pour la boîte d'interconnexion. Quant au niveau transistor, des solutions *single-ended* et *dual-rail* ont été explorées et comparées selon une métrique de conception bien définie. Le plus important à retenir est que le concepteur de circuits se trouve toujours face à un compromis entre gain en fiabilité et surcoûts induits. La décision d'adopter telle ou telle technique de durcissement se fera à la base des contraintes de l'application visée.

List of Acronyms

ALM	Adaptive Logic Module
ANR	Agence Nationale de la Recherche
ASIC	Application Specific Integrated Circuit
BDD	Binary Decision Diagram
CL	Cross Logic
CLB	Configurable Logic Block
DCVS	Differential Cascode Voltage Switch
d-DNNF	deterministic Decomposable Negation Normal Form
DES	Discrete-Event Simulation
DMSB	Downward Mini Switch Box
DWC	Duplicate With Compare
EA	Evolutionary Algorithm
EHW	Evolvable Hardware
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
GMR	Generalized Modular Redundancy
IC	Integrated Circuit
IO	Inputs and Outputs
ITM	Ideal Transfer Matrix
LUT	Look-Up Table
MC	Monte Carlo
MRF	Markov Random Field
MSO	Multiple Short Open
NMR	N-Modular Redundancy
NRE	Non-Recurring Engineering
PBR	Probabilistic Binomial Reliability

PDD	Probabilistic Decision Diagram
PGM	Probabilistic Gate Model
PTM	Probabilistic Transfer Matrix
PWL	Piece Wise Linear
RALF	Reliability Analysis for Logic Faults
RMS	Root Mean Square
RTL	Register Transfer Level
SEE	Single Event Effect
SET	Single Event Transient
SEU	Single Event Upset
SPM	Signal Probability Matrix
SPR	Signal Probability Reliability
SRAM	Static Random Access Memory
TMR	Triple Modular Redundancy
UMSB	Upward Mini Switch Box

Table of Contents

List of Figures

List of Tables

Introduction

Since the 1940s, reliability has been a major concern in the electronic industry. Actually, many approaches for improving reliability, such as Duplicate With Compare (DWC), Triple Modular Redundancy (TMR), parity codes, etc. were proposed. Works by J. Von Neumann, E. F. Moore and C. E. Shannon resorted to redundancy to obtain reliable systems from unreliable components [6, 7].

Ever since, integrated circuit technology has evolved rapidly, driven by a continuously higher density (billions of transistors in a chip). As CMOS feature sizes decrease into nanometers, electronic devices have been shrinking and operating at higher speeds (multiple GHz). On the other hand, fabrication process has not evolved at the same pace, so circuits have become more prone to manufacturing defects. Besides, complexity of the interconnect systems especially with the introduction of the 3D die integration scheme increases the probability of defects in a die [8, 9]. This led to a challenging issue: a decrease in the production yield. The latter has been considered as a fundamental criterion in defining an implementation's cost, as important as area, speed and power consumption which intervene in the tradeoff that a designer has to make [10]. In addition to that, deep-submicron technologies have become more and more susceptible to environmental disruptions like radiations. They cause transient faults that randomly occur while the circuit is functioning. Transient faults used to be a concern only in the design of memories. Yet, with technology downscaling, the vulnerability of combinational parts to transient faults has augmented resulting in error rates approaching those of memories [11]. In the meanwhile, higher operating frequencies also impose strict constraints on timing, thus increasing the probability of timing errors [12]. Therefore, the overall circuit reliability is expected to degrade [13]. This is a serious threat to circuits meant to be used in critical applications such as space, military and health. Therefore, fault-tolerant design solutions are hugely required. Field Programmable Gate Arrays (FPGAs) have been widely used in several applications thanks to their high flexibility and reduced time-to-the-market. They especially have attractive characteristics for space and avionic applications, where reconfigurability, high performance and low-power consumption can be fruitfully used to develop innovative systems.

This thesis is part of the Agence Nationale de la Recherche (ANR) project *Robust FPGA* whose objective is the design of a defect-tolerant FPGA. We were assigned the task of hardening the basic blocks in the FPGA against manufacturing defects. But the present work also extended the study to transient faults. Most of FPGAs in industry have a mesh topology and collect their logic blocks into clusters. Thus, the initial FPGA architecture chosen to be hardened in the *Robust FPGA* project is the mesh of clusters. To be able to harden the basic blocks (clusters and switch boxes), we need in a first step, to evaluate their reliability. Then in a second step, depending on the available budget, the most convenient hardening scheme will be selected. Therefore, these two main steps steered

the current work.

First and foremost, we looked into the state of the art for methods to assess reliability. We picked the most appropriate one, according to our needs, and proposed a methodology adapted to the cluster and switch box. We could also point out the most vulnerable components worth hardening. Thereafter, hardening techniques were proposed at two different granularity levels: multiplexer and transistor. At multiplexer level, a new fault-tolerant architecture called *Butterfly* for the cluster logic block was introduced. It uses modular redundancy and requires a voting circuit. A part of this work also consisted in proposing different optimized architectures for the voter. Then, the Butterfly design was compared to other hardened architectures in terms of fault tolerance and cost penalties. Moreover, another hardening approach based on a "smart" synthesis was proposed. Its principle relies on searching for the most reliable design within a set of possible implementations offered by a founder library. If this design already meets the reliability requirements, it is pointless employing redundancy. Hence, the proposed method is called *redundanceless*.

When hardening at transistor level, different single-ended multiplexer architectures were investigated. From the STM CORE65LPSVT library, we came up with a new multiplexer structure made up of two tristate cells. This design called *Z-Mux* proved to be more robust to common transistor defects than all other studied assemblings from the same library. Then, a Differential Cascode Voltage Switch (DCVS) multiplexer architecture was examined and proved to be an interesting solution in terms of robustness and area. But its delay and power consumption were penalizing. So, another new dual-rail multiplexer inspired from the DCVS logic was proposed and it uses what we called the *Cross Logic*. It is more area-consuming than its DCVS counterpart but its robustness and performances are better. Eventually, all studied multiplexer designs (single-ended and dual-rail) were compared and ranked with respect to design metrics expressing the tradeoff between reliability gain and different overheads.

This dissertation is organized into five main chapters, divided as follows:

Chapter 1 is dedicated to introducing the background of reliability and fault tolerance, FPGA topologies, and the mesh of clusters architecture studied in this thesis. The scope and the terminology of the current work are herein defined.

Chapter 2 describes the state of the art regarding analysis of digital circuits' reliability. Moreover, several hardening techniques are discussed.

Chapter 3 analyzes the fault tolerance of basic blocks in the mesh of clusters FPGA under study. The most vulnerable components are identified to be hardened.

Chapter 4 describes the proposed hardening solutions at the multiplexer level.

Chapter 5 deals with hardening at transistor level.

Conclusion summarizes the most important results and suggests further research foresights.

Chapter 1

FPGA's Fault Tolerance in Nanoelectronic Technology

As process technology scales to nanometers, electronic circuits are becoming more and more prone to faults: manufacturing defects decreasing their yield, and transient faults affecting their operation during their lifetime. Both phenomena arise reliability issues.

Thanks to their reconfigurability, low development costs and reduced time-to-market, FPGAs are widely used in several applications from networking to space [14–17]. Thus, the defect tolerance (manufacturer side) and fault tolerance (user side) of an FPGA are crucial to accomplish its functions without failure [18] [19], especially in critical applications demanding a high level of reliability.

This chapter aims to define the scope of the work as well as the terminology used throughout this manuscript. First, we are reviewing basic concepts of reliability and fault tolerance. Then, we are looking into FPGA architectures, and presenting the mesh of clusters architecture studied in this thesis.

1.1 Basic Concepts

According to Avizienis [20], an electronic system can be characterized by four properties: functionality, performance, cost and dependability. Avizienis et al. define *dependability* as "the ability to avoid service failures that are more frequent and more severe than is acceptable to the user(s)". In fact, it is a global concept including attributes such as reliability, availability, safety, integrity, maintainability, etc. Figure 1.1 depicts the taxonomy related to dependability, its attributes, threats and means [20]. Concepts that are in the scope of our work are marked in red.

In this work, we focus on the attribute of *reliability*, faults threatening it and fault tolerance as a means to improving reliability. The reliability of a given system/circuit is the degree of confidence observed in its outputs, given a certain scenario in which *faults* are expected to occur with a given probability [21]. A fault may result in an *error* at circuit level and then produce a system-level *failure*. Faults can be classified, according to their duration, into two main categories, namely permanent and non-permanent. And when they occur in a circuit, they do not necessarily propagate to the output. Indeed, they can be *masked* by different phenomena.

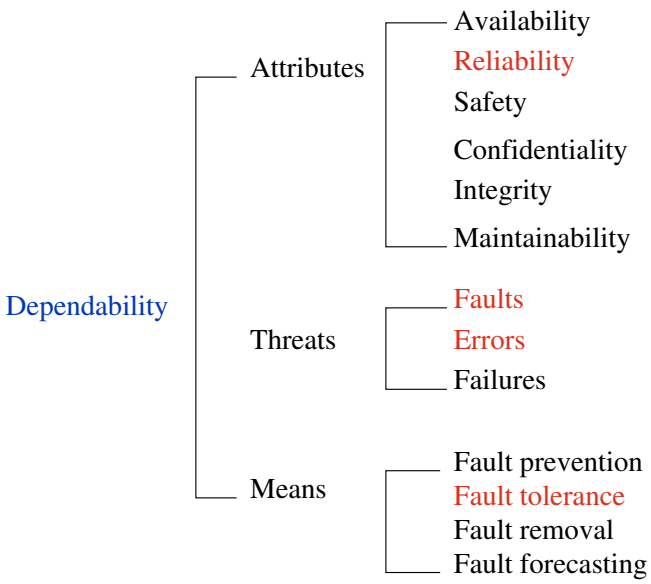


Figure 1.1: Taxonomy of dependability.

1.1.1 Types of Faults

1.1.1.1 Permanent Faults

Permanents faults generally result from imperfections in the design process. Indeed, the chip manufacturing process can not be perfectly controlled as it involves a series of complex processing steps. So, chips may be affected by manufacturing defects. The fraction of chips that, upon completion of manufacture, can meet a set of test requirements is called the *yield* [22]. On the other hand, permanent faults may also occur during the operational life of a circuit due to different reasons like aging or external factors. Permanent faults can be preceded by the occurrence of intermittent faults [23]. In this thesis, we focus on manufacturing defects as permanent faults.

Permanent faults can be classified into two main types: gross and spot defects [24, 25].

Gross defects are global defects originating from issues during the manufacture process like mask mismatching, scratching in the wafer, etc. Spot defects are local defects that happen at random spots on the wafer. They're typically caused by impurities in the materials and particles deposited on the chip.

As a matter of fact, the frequency of gross defects is virtually independent of the die size. This is not the case for spot defects whose expected number increases with the chip area [24]. Technology downscaling augments the number of spot defects. These defects may take place in any layer of the wafer or between layers.

Accurate defect modeling based on layout can be complex [26]. That is why different defect models have been proposed at different levels of abstraction.

– At gate level: a defect is often modeled by the *stuck-at* model at gate level, where a net is stuck at the logical value 0 or 1. This model is simple and widely used for test pattern generation [27–29]. Nevertheless, it is insufficient to characterize the real behavior of spot defects. In fact, the latter lead to resistive shorts and opens at random locations, which do not necessarily result in a constant 0 or 1. Instead, they may cause degraded voltage levels, increased propagation delays or transition

faults [30]. Thus, for more realistic defect modeling, one should descend at a finer level where *bridges* and *opens* are the most used defect types.

– At transistor level: an industrial study found that in a 130nm fabricated Integrated Circuit (IC), bridges and opens account for about 58% of all defects [31]. A bridge defect (or short) is an unwanted metal connection between two or more nets. Figure 1.2 shows an example of a bridge defect. An open defect is a connection break between two nets that should be connected. In realistic defect modeling at transistor level, the *stuck-open* and *stuck-closed* defects are the most used. A stuck-open (or stuck-OFF) transistor is never conducting, whereas a stuck-closed (or stuck-ON) transistor is always conducting. The short/open model can be also applied to connections between transistor terminals and any other net of the circuit.

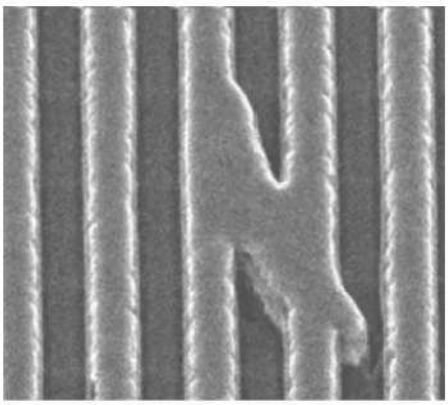


Figure 1.2: Example of a bridge defect [32].

Hence, in this thesis, permanent faults are modeled by stuck-ats at gate level. And at a finer level, they are modeled by bridges, opens, stuck-closed and stuck-open transistors.

1.1.1.2 Non-permanent Faults

There are two types of non-permanent faults: transients and intermittents. Intermittent faults are characterized by repeated and sporadic occurrence [33]. Errors induced by intermittents tend to happen in bursts, at the same location, when the fault is activated. They may affect circuits with parametric variations, and act as shorts or opens under certain conditions. They may become permanent faults. Unlike transients, intermittents can be repaired by replacement of the offending circuit. Intermittent faults are out of the scope of this thesis.

Transient faults, also known as soft errors, are a growing issue in nanoscale CMOS technology. They occur at random times and are responsible for one of the highest error rates in electronic circuits [13]. Transient faults are caused by different physical phenomena, e.g. alpha particles, cosmic rays, interconnect noise, electromagnetic interference, etc. Decreasing supply voltage along with a greater integration density made circuits more vulnerable to ionizing particles. Nowadays, circuits have become susceptible to such particles not only in dense radiation environments such as space, but also at ground level [34].

Energized particles strike PN junctions of a circuit, provoking a Single Event Effect (SEE). This temporarily charges or discharges nodes of the circuit, generating transient

pulses that can be interpreted as valid internal signals, engendering an erroneous result [35]. Figure 1.3 shows the effect of an ionizing particle after it hits a silicon junction. Two different phenomena are depicted: drift and diffusion currents. The drift current phenomenon is electric field driven and occurs very rapidly but the other one is not quick. There could be other charge transport phenomena.

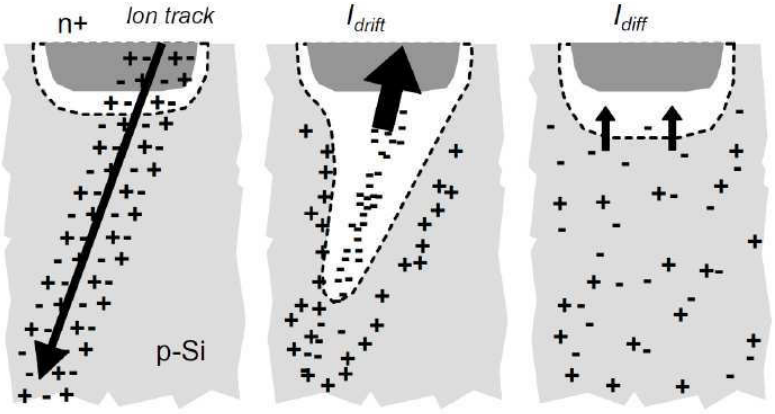


Figure 1.3: Impact of a charged particle on a silicon junction [36].

When SEEs occur in memory elements, they can alter the stored logical value and result in a *bit flip*. In this case, they are called Single Event Upsets (SEUs). Otherwise, SEEs can be Single Event Transients (SETs), manifesting themselves as transient pulses in the combinational logic. Other kinds of soft errors may still happen due to SEEs.

In this thesis, we are modeling soft errors by bit flips at gate level.

1.1.2 Types of Masking

Indeed, faults tended to be a concern only in the design of memories, which led to the commonly used error detecting and correcting codes [37, 38] and other mitigation techniques such as nodal interleaving [39, 40]. But now, with the reduced sizes of devices, faults in logic circuits result in error rates approaching those of memories [41, 42]. In earlier technologies, the effect of a fault in a logic circuit could be filtered out by three different masking phenomena:

- *Logical masking*, that happens for some input combinations when a fault appears in a non-sensitized path of the circuit and thus can not propagate to the output. It is technology-independent and occurs thanks to the circuit's inherent architecture. Figure 1.4 shows two examples of logical masking with OR and NAND gates.
- *Electrical masking*, that happens when a fault does not have enough duration or amplitude to reach the output of the circuit. Figure 1.5 depicts an example of a glitch being electrically masked.
- *Temporal masking*, that happens when the fault in the output signal does not arise at the sampling moment. Figure 1.6 represents an example of a transient fault occurring outside the latching window.

As long as electrical and temporal masking effects are both technology-dependent, their effects have been reducing because of CMOS technological downscaling [44]. Indeed, lower supply voltage and lower gate delay decrease the probability of electrical

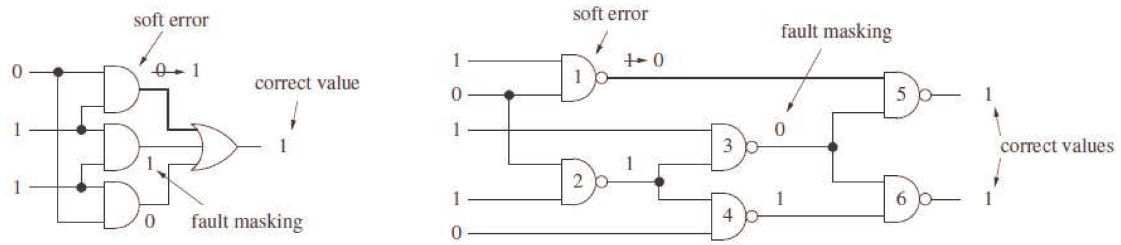


Figure 1.4: Examples of logical masking [43].

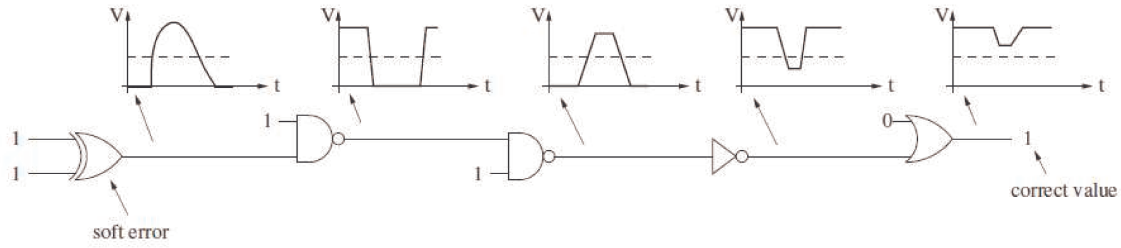


Figure 1.5: Example of electrical masking [43].

masking, and higher operating frequencies reduce temporal masking. Therefore, logical masking is nowadays more predominant than any other masking phenomenon and is the hardest to model and characterize [45].

1.1.3 Fault Tolerance

A circuit is said to be fault-tolerant if its operation is not affected by the occurrence of a fault. In this work, we focus on fault masking in combinational logic, assuming that memory is protected against faults thanks to suitable error detection and correction schemes. First of all, we need to estimate fault tolerance, in order to be able to improve it if necessary. Fault tolerance is estimated by considering only logical masking. This is basically how reliability is assessed. The next chapter gives more details about different methods to compute reliability from logical masking. Thus, we are providing an underestimate of the actual circuit reliability. Indeed, faults that are not logically masked may still be filtered out by other masking phenomena. To make the difference between tolerance to transient faults and to permanent ones, we use the term *robustness* to point out defect tolerance.

Fault tolerance improvement can be done by *hardening* that consists in the protection of logic blocks in a circuit against faults. It is a much more complicated task than protecting memory elements, because of the variety in logic functions. Therefore, protecting logic blocks is associated with important overheads in terms of area, time and power. Hardening techniques are typically based on redundancy (in information, in hardware, in time, or in any combination of them). Actually, thanks to the regularity of their structure and to their spare resources, FPGAs offer other possibilities of hardening than those commonly used for Application Specific Integrated Circuits (ASICs). In the technical literature, there are several approaches for repairing FPGAs when they are affected by faults. Some of these hardening schemes are presented in the next chapter.

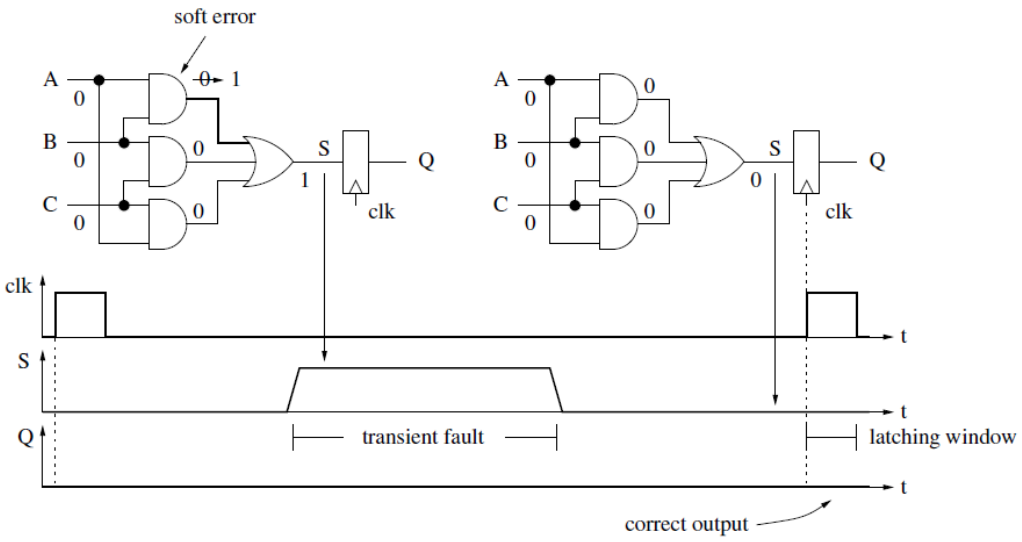


Figure 1.6: Example of temporal masking [43].

1.2 FPGA Architectures

FPGAs are flexible ICs that can be configured for any application after manufacturing. The reconfigurability property of an FPGA is commonly accomplished through reprogrammable Static Random Access Memory (SRAM) cells. FPGAs are basically made up of configurable logic blocks (CLBs) for implementing any kind of logic and interconnect blocks for routing signals. These blocks constitute the FPGA's *basic blocks* that this thesis proposes to harden. To be implemented on an FPGA, any circuit can be described by using general hardware languages such as Verilog and VHDL. Then, it is synthesized, placed and routed, so as to generate the configuration bitstream to be loaded in SRAM memory and that allows the circuit to be mapped on the FPGA.

1.2.1 Mesh Topology

The *mesh* architecture, also known as *island style*, is composed of logic blocks (CLBs) regularly aligned and placed within a routing network, as depicted in Figure 1.7. Interconnection wires organized in rows and columns form horizontal and vertical routing channels.

A CLB is made up of a Look-Up Table (LUT), a register and a multiplexer 2:1 to choose either the combinational or the sequential path. An example of a 4-input CLB is shown in Figure 1.8.a. A schematic of its corresponding LUT is represented in Figure 1.8.b. A LUT is simply a precalculated table stored in static memory containing all values of a logic function's outputs. From the values of inputs, we can extract the corresponding ouput, in order to avoid expensive runtime computation. It is typically based on a body of SRAM followed by a series of multiplexers. The SRAM allows loading the configuration bits (or simply the truth table) of the logic block, while the multiplexers implement the desired function, according to the values of inputs A, B, C and D.

The routing network is composed of connect blocks, switch boxes and routing channels. Connect blocks are switches used to connect a CLB to an adjacent routing channel, while switch boxes serve to connect intersecting routing channels. Each routing channel

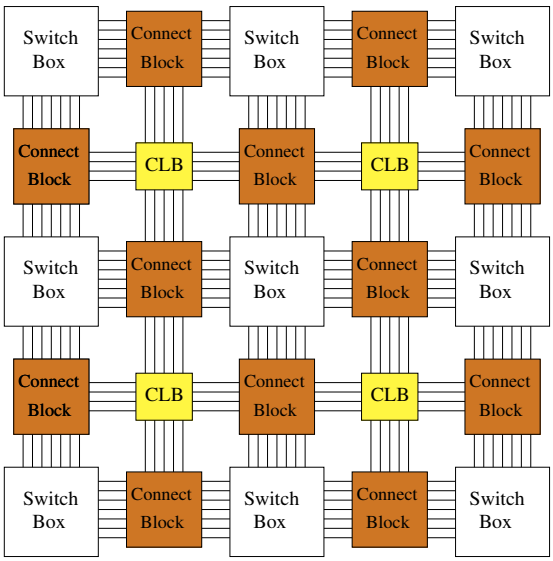


Figure 1.7: Mesh FPGA.

consists of W parallel wires, defining the channel width. All channels have the same width. The FPGA routing tool ensures that an application uses no more than this width. If it is not possible, the application is considered non-routable on that FPGA.

Because of the simplicity of its physical layout and its scalability, the mesh architecture is the most popular in industry. For instance, it is used for Altera Stratix II [46], Xilinx Virtex II [47] and is widely used in academic research especially by the developers of the VPR place and route tool [48]. However, the mesh architecture suffers from area and delay overheads. As a matter of fact, up to 90% of area is occupied by routing resources whose programmable switches increase the signal propagation delay [49]. In fact, the interconnect structure in common FPGA architectures is designed generally to maximize logic utilization.

1.2.2 Hierarchical Topology

Since most logic designs exhibit locality of connections, which implies a hierarchy in placement and routing of connections between logic blocks, gathering the latter into clusters provide smaller routing delays. Modern FPGAs use clustering to improve area and speed efficiency of the routing architecture [50–52]. The number of switches can also be reduced by depopulating the interconnect structure, e.g. by using sparse rather than full crossbars into the cluster [53]. Connection blocks, an intermediate level of multiplexers connecting the routing tracks to input multiplexers of logic blocks, which are used in the VPR architecture [48] and the Altera Stratix architecture [52], can be avoided by connecting the routing tracks directly to the input multiplexers, as in the Xilinx Virtex architectures [54]. Another possibility to avoid the use of connection blocks is to connect clusters directly to switch boxes [55].

Figure 1.9 depicts a hierarchical FPGA topology. Several commercial FPGAs have a hierarchical topology, such as Altera Apex [56] with two levels of hierarchy. The concept of hierarchical FPGA architectures has recently interested academic researchers. Many hierarchical architectures have been proposed in [57–59]. In [59], an efficient tree topol-

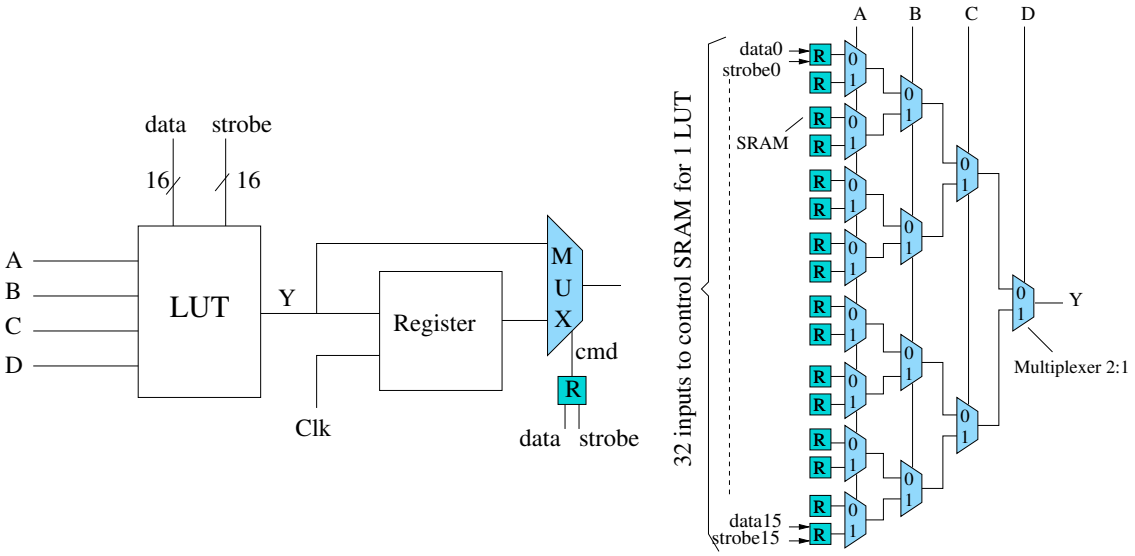


Figure 1.8: a. Structure of a CLB b. Structure of a LUT

ogy for the FPGA interconnect network is introduced and demonstrated a gain of 40% in total area compared to a mesh topology.

1.2.3 Mesh of Clusters Topology

The mesh of clusters architecture is built as a matrix of abutted tiles, as shown in Figure 1.10. Each tile is based on two different blocks: a cluster and a switch box. Therefore, we can benefit from the regularity of the mesh topology by duplicating tiles and generating a scalable layout on one hand, and from the use of depopulated clusters, on the other hand. A mesh cluster is surrounded by four routing channels from the mesh topology, and is directly connected to switch boxes.

In this work, we consider the structure of a cluster having 10 CLBs, 24 primary inputs and 12 outputs. These numbers were chosen in agreement with the *Rent parameter* p that calculates the number of inputs/outputs so as to obtain an optimized architecture in terms of locality of logic computations.

Equation (1.1) gives the relationship between the Rent parameter (p), the number of inputs and outputs of the cluster ($N_{IO(c)}$), the number of inputs and outputs per LUT ($N_{IO(l)}$) and the number of LUTs per cluster (K) [60].

$$N_{IO(c)} = N_{IO(l)}.K^p \tag{1.1}$$

The smaller the value of p, the more local the application turning on the cluster, i.e. there are less connections between the LUTs inside the cluster and those outside. It was shown that $0.5 \leq p \leq 0.65$ for most applications [61]. So, depending on the architecture's type of interconnect (whether it is full or sparse crossbar), the architecture's Rent parameter will be equal (if full crossbar) or superior (if sparse crossbar) to the application's Rent parameter. In the considered case, $p = 0.85$. Actually, a full crossbar interconnect is costly in terms of number of switches, hence increases delay and power consumption.

A detailed description of the basic blocks' structures (cluster and switch box) is given in Chapter 2 when analyzing their fault tolerance.

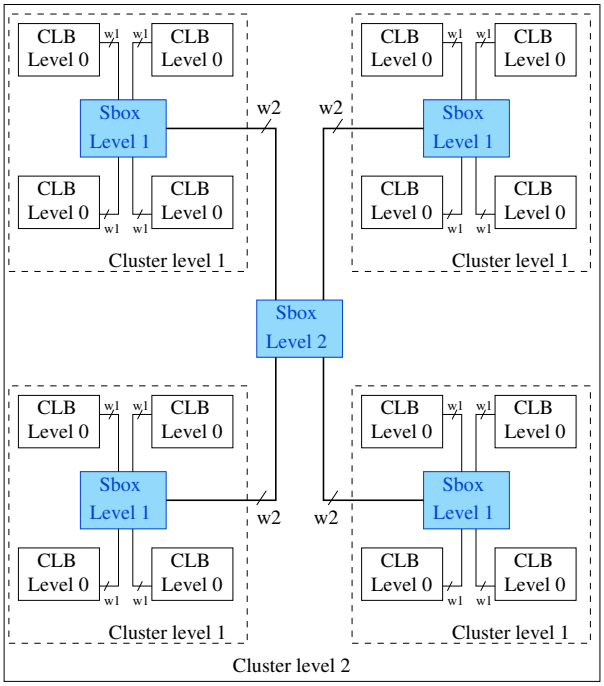


Figure 1.9: Hierarchical architecture.

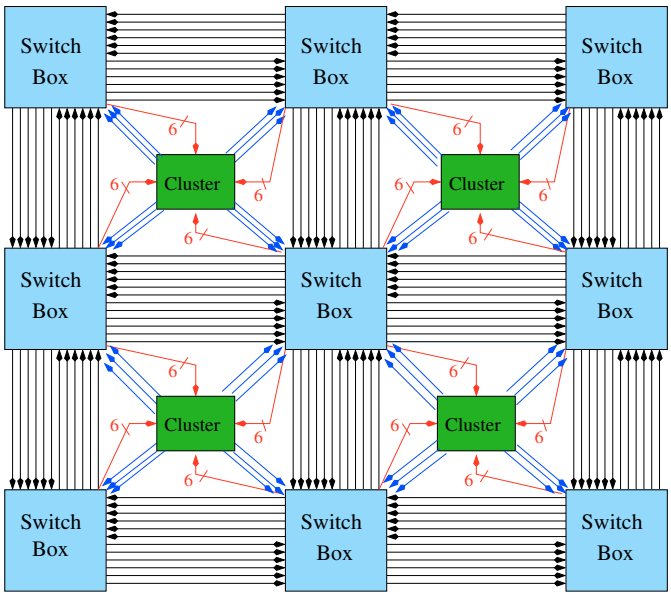


Figure 1.10: Mesh of clusters architecture.

Chapter 2

Fault Tolerance Analysis and Hardening Techniques

In order to improve the fault tolerance of a circuit, it is crucial to evaluate its reliability beforehand. This chapter is dedicated to presenting several reliability analysis methods that have been proposed in the literature. Then, general hardening schemes and others specific to FPGAs are exposed.

2.1 Fault Tolerance Analysis

Fault tolerance analysis approaches can be classified into two main categories: methods based on simulation/emulation and analytical methods.

2.1.1 Simulation/Emulation-based Methods

The principles of simulation-based and emulation-based approaches are the same, except that a hardware support platform such as an FPGA is used as an accelerator in the emulation-based techniques.

2.1.1.1 Simulation-based

Thanks to the simplicity of their concept, fault injection methods based on simulation have attracted many researchers [62]. The concept consists in comparing two versions of the circuit, a fault-free version and a fault-prone one, after simulating them and under the same conditions. Simulation checks whether the outputs of the faulty version correspond to the correct ones. If so, injected faults were masked. Thus, one may calculate the ratio between the number of times when a masking occurs and the total number of simulation runs. And this ratio is a metric for reliability assessment.

As a matter of fact, faults are injected according to a given model and depending on the granularity of the analysis (at block, gate, or transistor level). Several works in the current literature do fault injection at Register Transfer Level (RTL) where the circuit is described in Verilog or VHDL. On one hand, the drawback of such fault injection is that, at RTL level, the faults occurring inside the blocks are not taken into account. On the other hand, simulation at RTL level is much simpler and faster than at gate level.

Exhaustive simulations of all relevant fault sites with all input vectors in a circuit considerably increase simulation time. And this is all the more true so since we deal with multiple faults. Therefore, simulation-based fault injection is usually restricted to single faults. And even for complex circuits, it is still hard to evaluate all possible single faults, hence partial evaluations are carried out. However, this arises the issue of selecting the parts of the circuit to be evaluated.

A traditional and common method is the Monte Carlo (MC) [63] which imitates the real system behavior by a parametric adaptation at each iteration of the simulation. MC method is by far the most used in semi-conductors industry [64]. MC is an artificial sampling method used to solve problems that are analytically complex and probabilistic in nature. Indeed, the number of factors that intervene in the problem solving is so big that an analytical solution becomes intractable. The disadvantage of MC is that simulations are very time-consuming for large-sized circuits or circuits having a large number of inputs. This time constraint is less problematic with modern processors getting more and more performing.

The Discrete-Event Simulation (DES) [65] is an experimental method which reproduces the system behavior. To do so, the DES randomly samples the activity of a system under analysis. As soon as the system model is built, the processor executes as many samplings of the model as needed, to deduce right conclusions regarding the model behavior.

In order to circumvent the problem of time consumption in simulation-based approaches, hardware-based methods have been created.

2.1.1.2 Emulation-based

Several works used FPGAs for speeding up simulation of single stuck-at faults. For example, an emulation technique that does not require circuit reconfiguration was proposed in [66] where fault injection is performed by shifting a scan chain.

In [67], an FPGA platform for studying the impacts of SEUs on ASICs was explored. The fault-prone version of the circuit contains modified flip-flops. A computer supervises fault injection and classifies the effect of the faults into silent (totally masked), latent (the fault is stored in a memory element, though the circuit output is correct), failure (incorrect output) or detected (the fault is detected/bypassed by some mechanism).

THESIC+ is another fault-injection platform characterizing transient faults in digital circuits [68]. It focuses on the effects of SEUs occurring in the memory elements of a design. The platform also has been used to reproduce the results of radiation ground testing for microprocessors [69], and to analyze the reliability limits of TMR implemented in an SRAM-based FPGA [70].

Partial reconfiguration in FPGAs has been used in the last years [71]. The fault-injection method proposed in [72] benefits from the FPGA reconfiguration abilities. After a fault-free run of the design where the state of every flip-flop at each cycle is saved, a second run is performed, where reconfiguration is done to modify the state of one flip-flop at a time, thus emulating an SEU.

A twofold simulation/emulation approach is presented in [73]. SETs are simulated in complex digital circuits. Simulation informs about the propagation path of SETs from the affected logic gate to memory elements, and also about the occurrence of multiple errors. Concerning the emulation part, it is used to observe the errors resulting from SETs, and possibly propagating to the outputs of the circuit. FuSE is a platform proposed

by Jeitler et al. in [74] that uses both emulation and simulation-based fault injection schemes. However, data is collected by reading text files that tend to be large and tough to manipulate.

The authors of [75, 76] propose the AMUSE platform that employs a multilevel FPGA-based approach for SET analysis. The approach is multilevel in the sense that fault injection is done at the gate level, whereas fault propagation across clock cycles is performed at the RTL to speed up the process.

The Fault-Injection-Fault-Analysis (FIFA) platform dealing with multiple faults is proposed in [3]. Faults can be either stuck-ats or bit flips and are generated by means of *saboteurs*. The FIFA platform can inject as many simultaneous faults as the number of all gates in the design under study. And for all input combinations, the platform counts the number of logical maskings that occurred. This value is later used in the Probabilistic Binomial Reliability (PBR) model [4] to compute the circuit reliability. The PBR method will be further explained in the next section.

2.1.2 Analytical Methods

Analytical approaches reduce the circuit into its elementary elements and model its behaviour through formal equations. Generally speaking, these methods are less time-consuming than the simulation-based ones and they were essentially developed to cope with large circuits.

2.1.2.1 PTM

The Probabilistic Transfer Matrix (PTM) approach, introduced in [77], is based on modeling the logic gates and circuit architecture via the use of matrices. The truth table of each gate in the circuit is represented by an Ideal Transfer Matrix (ITM) whose rows correspond to the different input combinations, and columns to the different output combinations. For a gate having m inputs and n outputs, the ITM size is $2^m \times 2^n$. Figure 2.1 gives an example of an ITM for an AND gate.

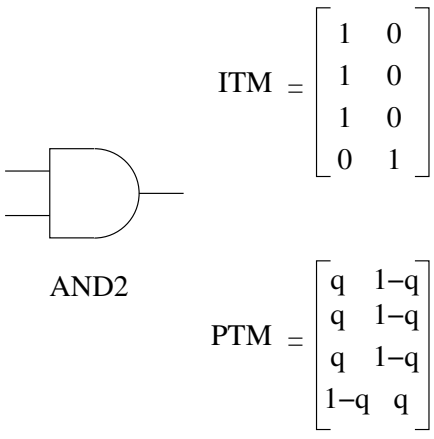


Figure 2.1: ITM and PTM matrices for an AND gate.

Then, the PTM of a gate is obtained from the ITM, according to the fault model and to the gate's inner reliability denoted q. The latter defines the ability of the gate to output

a correct response. Thus, for the AND example, in case of a bit flip model, the PTM is simply extracted by replacing in the ITM the '1's by q and the '0's by $1-q$. The PTM's AND is shown in Figure 2.1. The PTM method also allows other fault modeling such as the stuck-at.

The circuit is then organized into logical levels where gates are parallel in each level, and in series from one level to another. The PTM at the output of each level is the result of the Kronecker product of all PTMs inside that level. Thereafter, the PTM of the whole circuit is obtained by multiplying all PTMs of the different levels in a topological order.

The main advantage of the PTM approach is its accuracy in modeling logical masking which gives an exact computation for the circuit reliability. Nevertheless, PTM suffers from scalability problems. In fact, a circuit PTM size grows exponentially with the number of inputs and outputs, leading to an intractable computation time and a need for a big storage space in memory, even for medium-sized circuits.

2.1.2.2 SPR

The Signal Probability Reliability (SPR) method, introduced in [78], is also based on PTM matrices. It improves the PTM method in the sense that it overcomes intermediate level matrices increasing quickly. Based on a straightforward signal reliability computation and propagation algorithm, SPR assigns a reliability value to each signal. Indeed, the *signal reliability* of a given signal is defined as the probability that this signal carries a correct value. A binary signal x can take four different values: correct zero (0_c), correct one (1_c), incorrect zero (0_i) and incorrect one (1_i). These states are represented in a 2×2 matrix shown in Equation 2.1 where $P(.)$ stands for the probability function. This Signal Probability Matrix (SPM) is attributed to each signal in the circuit.

$$SPM(x) = \begin{bmatrix} P(x = 0_c) & P(x = 1_i) \\ P(x = 0_i) & P(x = 1_c) \end{bmatrix} = \begin{bmatrix} x_0 & x_1 \\ x_2 & x_3 \end{bmatrix} \tag{2.1}$$

The signal reliability for x, noted R_x, comes directly from Equation 2.2.

$$R_x = P(x = 0_c) + P(x = 1_c) = x_0 + x_3 \tag{2.2}$$

SPR uses the SPMs as well as the PTMs and ITMs of gates to determine the cumulative effect of simultaneous multiple faults on the circuit reliability. To do so, the joint probability of input signals of a given gate is the Kronecker product of the inputs' SPMs. Then, the product matrix is multiplied by the PTM of the gate to consider both reliability and logic function of that gate. Finally, this matrix together with the ITM serve to get the SPM of the gate output. That is how we return to an SPM size for each signal, and the algorithm is repeated through all circuit levels till the outputs.

An important advantage of the SPR algorithm is that its complexity grows linearly with the number of logic gates in the circuit. As a result, it is more performant than the PTM method. Nonetheless, the more reconvergent fanouts there are in the circuit, the more SPR loses accuracy in computing the circuit reliability. This issue is discussed in the next section.

2.1.2.3 SPR-MP

SPR-Multi Pass (MP) enhances the original SPR algorithm by tackling the problem of reconvergent fanouts. To illustrate this issue, let us take as an example a simple circuit

shown in Figure 2.2. Although this circuit does not contain an actual reconvergent fanout, its reliability $R(circuit)$ computed by SPR is the product of the reliabilities of its output signals. This multiplication has the same effect as a reconvergence, and gives inconsistent results since it assumes independent output signals, which is not the case. In the example we are giving, input signals are supposed fault-free, hence the SPMs of the inputs only contain the probabilities of correct signal values (a_0 and a_3). This assumption is not a limitation and is done solely for simplification purposes.

$$R(circuit) = S_1 \times S_2 = a_3^2q^2 + a_3a_0q^2 + a_0a_3q^2 + a_0^2q^2 \tag{2.3}$$

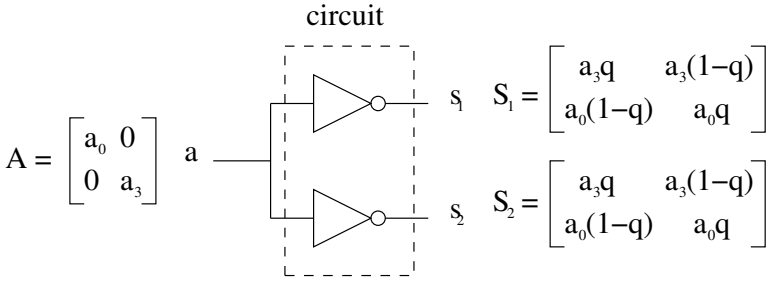


Figure 2.2: Computing the reliability of a simple circuit with a reconvergent fanout.

As long as SPR can not handle reconvergent fanouts, it gives an approximation of the accurate result. Equation 2.3 expresses the circuit reliability calculated by SPR algorithm. There are redundant terms like $a_3^2q^2$ and incompatible ones like $a_0a_3q^2$ containing both probabilities of the signal being at different logic states.

The SPR-MP approach takes into account the correlation of signals by performing the analysis in multiple passes. A single state of the fanout signal is considered in each pass. Thus, there are 4 possible passes for each fanout, corresponding to four partial reliability values that should be added in the end. Figure 2.3 corresponds to the SPR-MP computation for the same aforementioned example. Only two passes are being shown because the input signal has merely two states as previously justified. The reliability of this example circuit is the sum of the two partial reliability values, as expressed in Equation 2.4.

$$R(circuit) = R(circuit, a_0) + R(circuit, a_3) = a_0q^2 + a_3q^2 \tag{2.4}$$

As expected, the reliability computed by SPR-MP no longer contains inconsistent terms. If all fanouts are taken into account, SPR-MP is completely accurate. However, its complexity grows exponentially with the number of fanouts in the circuit.

In order to lower the execution time, it is possible to consider only a set of the most important fanouts. Of course, this yields some accuracy in the reliability result. Depending on the topology of the target circuit, even when only a small number of fanouts is considered, a large reduction in execution time is possible with an error that is smaller than 2% [79]. An interesting trade-off between accuracy and complexity can be obtained using SPR-MP.

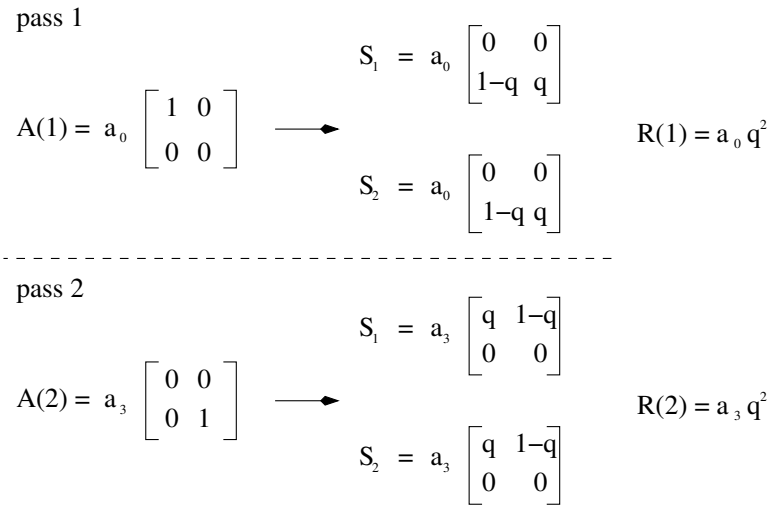


Figure 2.3: SPR-MP algorithm applied to a simple reconvergent circuit.

2.1.2.4 PBR

The PBR method [4] is categorized among analytical solutions for reliability assessment. Nonetheless, its inputs are usually extracted from a fault injection method.

In the PBR approach, like in the previously described analytical methods, every gate in the circuit is characterized by its reliability q. Let us define the probability of occurrence of k simultaneous faults in a circuit with w gates as a function $f_k(q)$, given by Equation 2.5.

$$f_k(q) = (1-q)^k q^{(w-k)} \tag{2.5}$$

Let us take an error vector e whose length is equal to the number of fault-prone gates. The number of 1's in vector e corresponds to the number of faults. We will use $e_{w:k}$ the set of vectors e with k 1's and length w. Let us note c_k the sum of the total number of error vectors e that result in correct values at the output of the circuit, considering N inputs, and k faults injected. Then, the reliability of the circuit is expressed in Equation 2.6. The expression of c_k is given in Equation 2.7 where x and y are the input and output vectors respectively. If $\overline{y(x_j, e_{w:0}) \oplus y(x_j, e_{w:k}(l))} = 1$, there is an error masking in the output. Note that the calculation of c_k takes into account not only error configurations for masked faults, but also the error configuration for fault-free operation $(k = 0)$.

$$r = \sum_{k=0}^{w} f_k(q) c_k \tag{2.6}$$

$$c_k = \sum_{j=0}^{2^N-1} p(x_j) \sum_{l=1}^{C_k^w} \overline{y(x_j, e_{w:0}) \oplus y(x_j, e_{w:k}(l))} \tag{2.7}$$

If we consider a uniform distribution for input vectors x, the coefficients c_k can be rewritten as $c_k = \frac{1}{2^N} \times m_k$, where m_k is the number of maskings computed by a fault injection method, and whose expression is given by Equation 2.8.

$$m_k = \sum_{l=1}^{C_k^w} \sum_{j=0}^{2^N-1} \overline{y(x_j, e_{w:0}) \oplus y(x_j, e_{w:k}(l))} \tag{2.8}$$

In this case, a simplified reliability expression of Equation 2.6 is obtained by Equation 2.9.

$$r = \frac{1}{2^N} \sum_{k=0}^{w} f_k(q) m_k = \frac{1}{2^N} \sum_{k=0}^{w} R_k \tag{2.9}$$

The PBR method separates the logical masking evaluation from the probabilistic analysis of the circuit. Therefore, since logical masking is technology-independent, the circuit reliability can be assessed for various technologies without having to re-perform fault injection. In fact, we only need to re-evaluate Equation 2.9 using a different value of q.

2.1.2.5 Other Methods

The work of Ogus et al. [80], targeted to stuck-at faults, focuses on the evaluation of signal reliability. A method based on the equivalence of functional classes was proposed to evaluate the probability of having a correct output in a logic circuit. However, partitioning into equivalent classes leads to an exhaustive characterization of the circuit states, which results in a high operational complexity.

Dokouzgiannis and Kontoleon [81] were also interested in stuck-at faults and used an equivalent graph for the circuit to evaluate its reliability. Vectors of dominant faults are determined for some basic logic circuits, and are associated with an equivalent graph of gates. Equivalent graphs of gates are interconnected according to the circuit topology, and a procedure for path plotting identifies the vectors related to correct functions contributing to the circuit reliability. The proposed methodology was presented for circuits having single outputs, and claimed to work for multiple-output circuits, but no results on benchmarks were provided to prove that. Bogliolo et al. [82] propose a symbolic simulation to assess circuit reliability. Binary decision diagrams are used to model the original circuit and the defective one, considering stuck-at faults. The method is limited by the number of nodes in the diagram. As an alternative, an incremental evaluation is proposed where the target circuit is partitioned into two sections. Reliability is calculated in an incremental way inside these sections. There are other approximations modeling the circuit by independent sets, representing smaller binary diagrams. This work presented some results on the reliability of certain circuits, but did not mention the error due to partitioning and to approximations.

In [83] Chen et. al proposed a Markov Random Field (MRF)-based model which works only for small circuits without re-convergent fanouts. An exact probability model, based on Bayesian networks, is proposed in [84] to capture signal dependencies in the circuit through the conditional probability specifications in the Bayesian network. Besides, this model has a minimal representation which is important in terms of scalability.

The Probabilistic Decision Diagram (PDD) method [85] can be used to calculate exact reliability values. This approach models the circuit by a graph where each gate corresponds to a PDD node. Each node implements the original logic function of the gate, with a probabilistic inverter connected to its output. Such an inverter models the gate's fault probability.

The authors in [86] describe three accurate and scalable techniques for reliability analysis of logic circuits for different gate failure models. The first algorithm, called observability-based reliability analysis, provides a closed-form expression for reliability using the observabilities of the gates, and is accurate when single gate failures are dominant. The second algorithm, named single-pass reliability analysis, is based on topological sorting, and is accurate even for multiple gate failures, provided that there is no reconvergent fanout. The third algorithm is the maximum-k gate failure reliability analysis which sets an upper limit, k, on the number of simultaneous gate failures in a circuit. The three algorithms have potential applications to reliability analysis of failures arising due to a broad range of mechanisms including SEEs, process variations, and reliability degradation due to aging.

An exact algorithm, called Reliability Analysis for Logic Faults (RALF), is introduced in [87]. The major innovation in RALF is the compilation of circuits into a representation for Boolean functions called deterministic Decomposable Negation Normal Form (d-DNNF) [88]. The fault rate at the primary output can be computed in time and space linear in the size of the d-DNNF representation, even in the presence of path reconvergence in the circuit. The d-DNNF representation is generally more concise than a Binary Decision Digram (BDD) representation, enabling RALF to scale to many real-world circuits.

Using the SPR approach, Flaquer et al. [89] propose conditional probabilities to deal with correlation in reconvergent signals. Obtained results are accurate, and execution time depends on the circuit topology and how it can be arranged into clusters. This method proved to be faster than SPR-MP, but it requires storing intermediate matrices of conditional probabilities. Moreover, its application to sequential logic circuits leads to inaccurate results because of the existence of loops. In order to circumvemt this problem, the authors in [90] convert the sequential circuit into a secondary combinational circuit, then perform reliability analysis by applying the method of [89] to the circuit iteratively.

Han et al. [91] use the Probabilistic Gate Model (PGM) to assess circuit reliability. In the PGM, there are two computational algorithms, one being approximate and the other accurate, in the same way as SPR and SPR-MP. And by the way, PGM propagates signal probabilities from inputs to outputs in a similar manner as SPR does. Since the operations of binary functions can be mapped into a set of probability values in the real interval $[0, 1]$, gates of any complexity can be modeled as PGMs. The latter can be used to model bit flips as well as stuck-at faults.

Another interesting method that was recently developed is called SNaP [92], which is a hybrid method. Indeed, SNaP is both an empirical and a simulation-based method for the assessment of circuit reliability. It can also benefit from emulation speed-up when used in FPGA device. Eventhough some of its steps rely on simulation, absolutely no fault injection takes place, since no internal signal is altered in SNaP. The method deals naturally with the occurrence of multiple faults while taking logical masking into account, and it can assess the reliability of both combinational and sequential circuits. SNaP has a linear complexity with the number of gates in the longest path for combinational circuits, and a constant calculation time for the sequential logic. The core concept behind SNaP is the idea of fault sources and fault propagation. Considering a whole circuit with multiple gates, the goal is to assess how many of those are capable of generating a fault that successfully propagates to the primary outputs. Logical masking is considered during that assessment.

To conclude, various approaches to analyze the reliability of digital circuits are avail-

able in the literature. Each one has its pros and cons. The designer selects the suitable method depending on the objective of the reliability analysis, the number and type of faults expected to occur, among others.

2.2 Hardening Techniques

This section presents a few techniques aimed to increase the reliability of digital circuits. There are general hardening approaches that could be used for any circuit and others specific to FPGAs. A different set of techniques is targeted for memories, but they are not covered by this thesis.

2.2.1 General Hardening Schemes

A circuit is usually hardened by redundancy. Spacial redundancy requires additional hardware resources, and it is employed to mitigate both transient and permanent faults. Time redundancy is efficient only against transient faults and engenders significant speed and power penalties. We are giving examples of fault-tolerant techniques based on spacial redundancy, at different granularity levels.

2.2.1.1 Modular Redundancy

Proposed by Von Neumann [93], Modular redundancy uses replicas of the same module operating in parallel, with a voter downstream deciding the final output, according to a voting strategy (majority, median or plurality) [94]. Triple Modular Redundancy (TMR) is the most popular hardening technique in this family. A simple TMR system is represented in Figure 2.4(a). With a bitwise majority voting, one out of the three copies of a TMR system can be faulty. This means TMR is able to tolerate a single erroneous module output, but multiple faults can occur within this very defective module. In an N-modular Redundancy (NMR) system, having N identical modules, up to $\lfloor \frac{N-1}{2} \rfloor$ erroneous outputs can be tolerated.

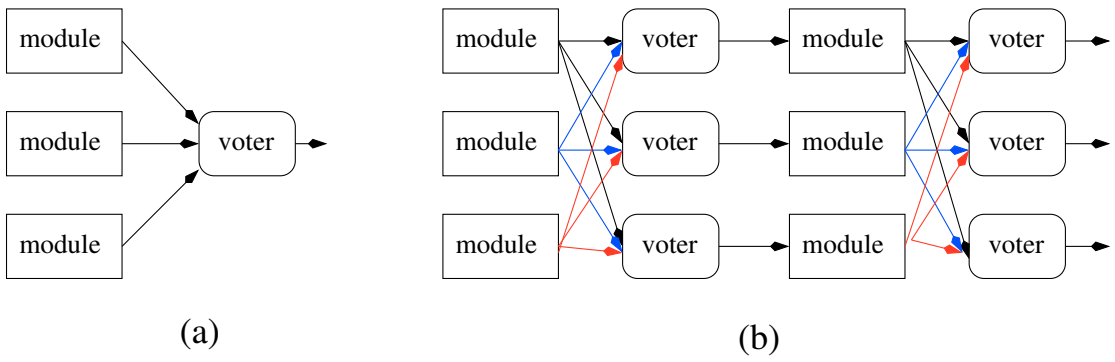


Figure 2.4: TMR technique (a) Single system (b) Cascaded system.

An advantage of TMR over NMR is the simplicity in the design of the voter. On the other hand, in a single fault scenario, if the TMR voter is faulty, the final output is

inevitably corrupted. Hence, the voter can be triplicated as shown in Figure 2.4(b). This figure depicts the concept of a cascaded TMR system.

As a matter of fact, the simple TMR system is mainly used at system level for critical systems. However, the reliability of a system increases if TMR is applied at a finer level. A system can be divided in sub-systems, each using TMR, and cascaded to make up the whole system. Then, the voters can be triplicated [95].

In [96], a Generalized Modular Redundancy (GMR) scheme is proposed. It replicates modules according to the probability of occurrence of output combinations and also to the number of faults. To recover from soft errors in protected combinations, Hamming distance is used. Reliability analysis showed that GMR could achieve reliability figures higher than that of TMR, and generally with less overheads.

2.2.1.2 Gate Redundancy

TMR could be applied to logic gates to enhance their reliability, but this is not feasible in practice, since the voter would take too much space relatively to the gates. Thus, an alternative solution, called *interwoven logic* [97], was proposed. It uses redundancy at gate level. For a given circuit, the number of logic gates is multiplied by four. Then, an algorithm connects them in a specific way, so as to suppress the voter. Interwoven logic can be also applied at transistor level to create more robust components.

2.2.1.3 Transistor Redundancy

The *Quadrupled transistor* [98] is inspired from TMR and interwoven logic. It consists in adding a transistor in series, and duplicating the whole in parallel. Figure 2.5 shows a quadrupled transistor. This structure is tolerant to the most common types of transistor defects mentioned in Section 1.1.1.1. The transistors in parallel resist to stuck-open defects. Thanks to the transistors in series, stuck-closed defects can be tolerated. Nevertheless, the quadrupled transistor technique is not resilient to other types of transistor defects like shorts between gate and source/drain.

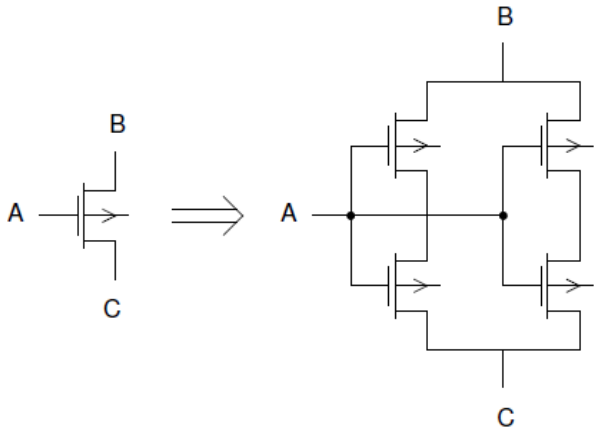


Figure 2.5: Quadrupled transistor technique [99].

The Multiple Short Open (MSO) technique, introduced in [100], comes as a solution to tolerate shorts between transistor terminals. Two transistor networks, called *pull-up* (charge to Vdd) and *pull-down* (charge to gnd), are isolated from the input by the use of

a defect-tolerant resistance. The latter is composed of two transistors as shown in Figure 2.6. Actually, the quadrupled transistor technique is combined with such resistance to obtain the MSO structure depicted in Figure 2.7. The major drawback of the MSO is multiplying the number of transistors by 12.

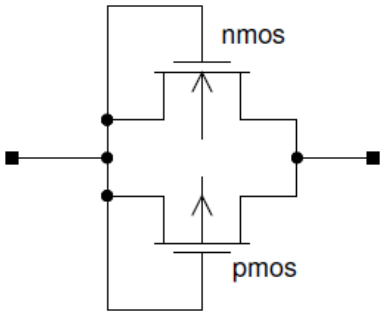


Figure 2.6: Resistance implementation in MSO technique [99].

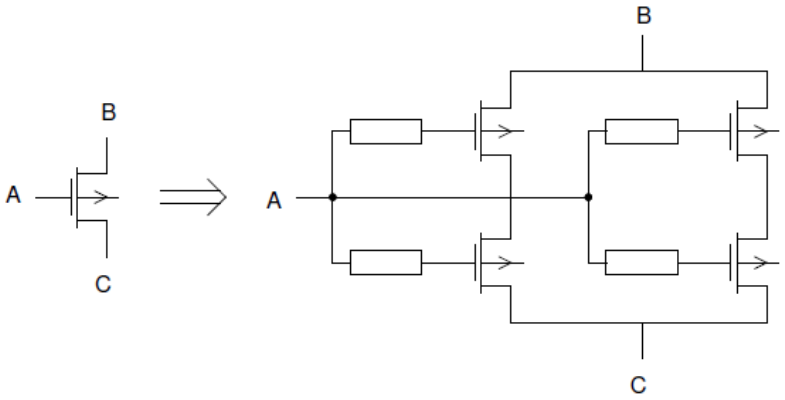


Figure 2.7: MSO technique [99].

2.2.1.4 Evolvable Hardware

Evolvable Hardware (EHW) [101] is the application of Evolutionary Algorithms (EAs) [102, 103] for hardware design. Inspired from the biological evolution of species, an EA is based on the idea of population of individuals. Each individual represents an electronic circuit. The population is iteratively changed by reproduction of selected parents and mutation of children. A fitness function indicates how well a candidate circuit meets the design specifications. In addition to defining the functionality of the circuit, the fitness function may also take into account other aspects like fault tolerance, area and power consumption, and assign them different weights. Then, the role of the EA is to guide evolution towards the goal described by the fitness function. The benefit of EHW is that the design is not restricted to traditional design techniques, and might reveal original features a human designer would not think of.

Fault tolerance in EHW was pioneered by Thompson with the evolution of fault-tolerant electronic control systems in 1995 [104]. Many other recent works have been conducted on evolved fault tolerance [105–108]. In [109], NASA evolved field-programmable

transistor arrays to be fault-tolerant. And EAs were also used for fault recovery in FPGAs [110]. The authors in [111] proposed, through artificial evolution, a defect-tolerant LUT for FPGAs that proved to be unsuitable in real FPGAs, because of its high delay and low output voltage swing.

2.2.1.5 Differential Logic

In differential circuits, information is processed and transmitted in a redundant and complementary way, intrinsically offering an increased resistance to faults. Differential Cascode Voltage Switch (DCVS) logic was introduced in [112] and used in [113] for defect tolerance purposes. In [113], DCVS logic in a circuit architecture inspired by an artificial neural network model proved to be more reliable than single ended standard CMOS logic, using very-deep submicron technologies.

2.2.2 FPGA-specific Hardening Schemes

In the technical literature, there are several techniques for repairing FPGAs when they are affected by faults. Most of these hardening schemes resort to redundancy and can be classified into software-based and hardware-based techniques. Software-based techniques avoid defective resources by means of place-and-route tools which map around defects. Hence, the efficiency of the software approaches rely on the performance of such tools. Hardware-based techniques employ modifications in the basic architecture. In some cases, extra hardware resources are added, providing redundancy at different granularity levels. While in some cases, architecture optimization is done to automate the configuration bits shift mechanism.

2.2.2.1 Software-based Techniques

Software-based hardening requires no modification in the basic FPGA architecture. Fault tolerance is provided solely through place and route mechanism. When a fault is detected and located, the application is shifted to the spare resources in the FPGA. These kinds of software-based hardening cost in terms of reconfiguration time and memory and they also reduce effective resources in FPGA. In some cases [114], this application shifting mechanism has been made time-efficient by providing pre-compiled alternate place and route solution. Memory cost has been reduced by producing differential configuration in which the application is shifted to the spare neighbouring column of the FPGA array. In [115], the authors proposed a method to identify fault resources by combining different kinds of test patterns, and they also developed a place and route tool avoiding defective resources at both tile and multiplexer levels.

2.2.2.2 Hardware-based Techniques

Hardware-based harderning involves modification of the original FPGA architecture to make it fault-tolerant. In [116], defect tolerance is achieved by automatic shifting of configuration data from defective to defect-free FPGA resources. It requires neither configuration data to be loaded from off-chip memory nor any intervention from the user. A small modification is made in the FPGA architecture where SRAM cells are altered such that they can shift the stored data to the neighbouring cells. Figure 2.8 illustrates an example of bitstream shifting in case of a defective CLB, where SRAM memory is supposed

made up of shift registers. This approach saves the reconfiguration time but costs the spare resources and alteration in SRAM cells. Besides, the approach is limited in case of multiple defects, because the latter are not necessarily aligned with spare resources. In [117], a horse-shifting algorithm provides an efficient way of distribution of spare CLBs. The spare allocation distribution is inspired from the movement of horse in chess.

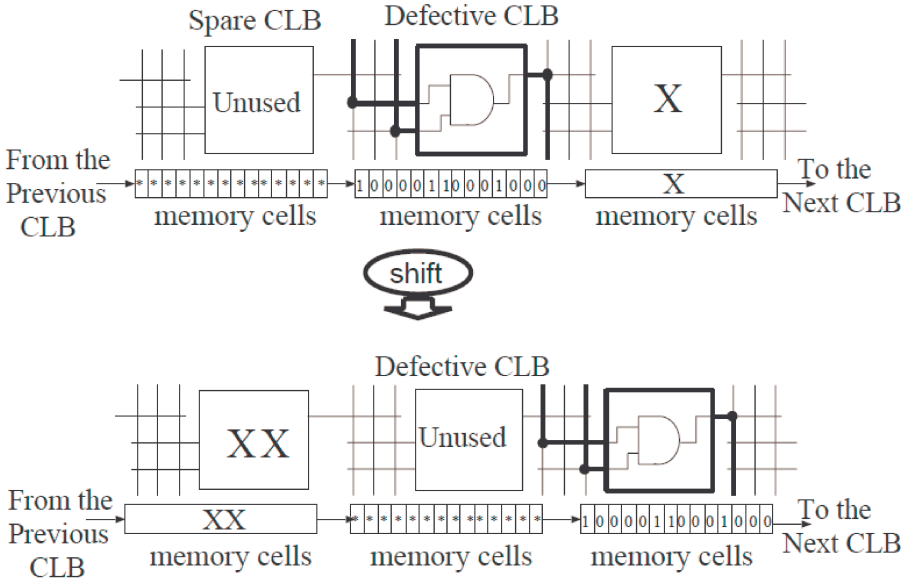


Figure 2.8: Bitstream shifting in CLBs [116].

In some cases, harware redundancy is achieved by adding spare logic/interconnection resources that are swapped with the faulty ones. The swap time is typically less than the time needed to generate a new placement and routing solution. This hardware-based redundancy can be implemented at different granularity levels. For instance, in [118], Lach et al. proposed a defect tolerance method based on the partitioning of the physical design into tiles. Each tile contains an additional CLB, so as to allow the correction of one defect per tile. Thus, only the defective tile will be reconfigured.

Coarse-grain redundancy [119] uses the redundancy of rows/columns in a mesh FPGA architecture. This technique has been used in industry [120]. Defects are avoided by substituting the supplementary row/column for the defective one, as shown in Figure 2.9. As a matter of fact, the coarse-grain redundancy is suitable for tolerating clustered defects. It can not tolerate multiple distributed defects.

Fine-grain redundancy [119] employs redundant routing resources by adding switches in the switch box. In order to successfully bypass a defect, a defect map should be either stored on-chip in non-volatile storage, or in an off-chip database indexed using a unique on-chip ID. When the FPGA is being programmed, defect avoidance is performed according to the defect map. Figure 2.10 shows how a defect is avoided by shifting individual signals. A shift-avoid and shift-restore mechanism is fully described in [121]. This mechanism eliminates the need to re-route the whole application and thus reduces the correction time. Unlike the coarse-grain redundancy, fine-grain redundancy technique tolerates multiple randomly distributed defects. In terms of area, coarse-grain redundancy engenders less overhead for low-density defects. However, fine-grain redundancy

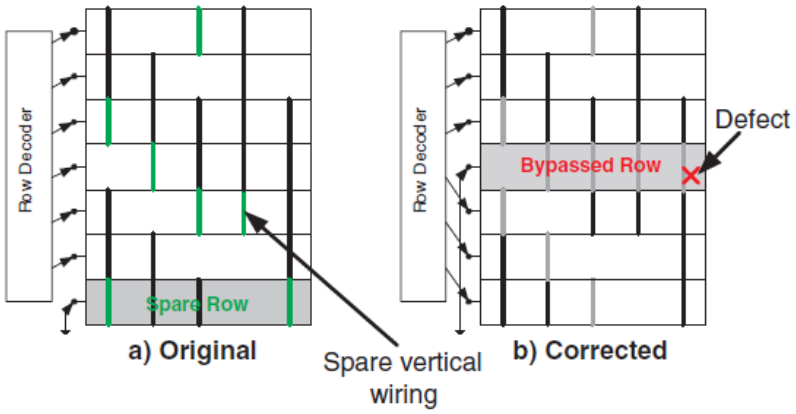


Figure 2.9: Coarse-Grain redundancy [119].

incurs a fixed area overhead of 50% for all array sizes according to [121], thus tolerating an increasing number of defects as the FPGA size grows at a constant cost. A detailed comparison of the coarse-grain and fine-grain redundancy is provided in [119]. Redundancy can be implemented at a finer level. For instance, spare connections can be added inside the switch block to tolerate one transistor defect per switch block [122]. However, this approach costs an average of 3% delay penalty and a partial modification of original data.

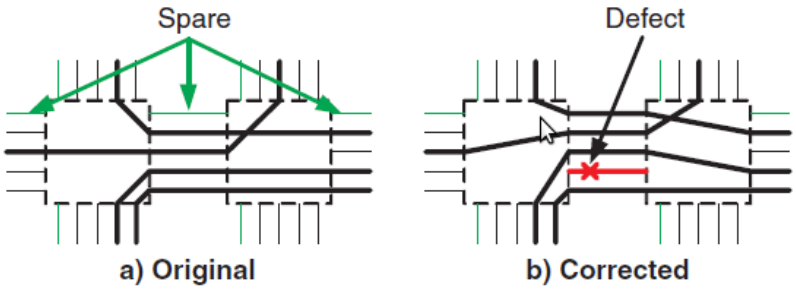


Figure 2.10: Fine-Grain redundancy [121].

Works reported in this chapter allowed us to propose solutions for analyzing then improving the FPGA basic blocks' fault tolerance. These solutions will be detailed in the following chapters.

Chapter 3

Analysis of the Basic Blocks' Fault Tolerance

In a fabless manufacturing context, the design of fault-tolerant circuits is performed so that their reliability would be as manufacturing-process-independent as possible. Thus, hardening the circuit is achieved by improving its logical masking ability. This is one of the reasons why we focus on reliability analysis methods assessing logical masking.

Since our target is to harden the basic blocks (cluster and switch box) in the mesh of clusters FPGA described in Section 1.2.3, we have to evaluate their initial fault tolerance. Moreover, hardening is costly in terms of resources and some designs do not benefit from enough budget to afford a full hardening. Thus, only a few parts of the circuit are selected to be hardened. This is called *selective hardening*. The concern is to choose the areas to be hardened. Therefore, after assessing the fault tolerance of the whole circuit, we need to determine the most "critical" parts in terms of reliability.

3.1 Analysis of the Cluster

3.1.1 Structure of the Cluster

Figure 3.1 represents the structure of a cluster having 10 CLBs, 24 primary inputs and 12 outputs. These characteristics were chosen in accordance with the Rent parameter defined in Section 1.2.3.

In this architecture, we have 4 crossbars 'Down' (see Figure 3.1) linking inputs to the CLBs. Each crossbar 'Down', as shown in Figure 3.2 is made up of 10 multiplexers 9:1, each of them having 6 primary inputs and 3 others coming from previous calculations of outputs, so as to re-utilize results and therefore favor locality. So, we have a total of 40 multiplexers 9:1 in all crossbars 'Down', rather than 40 multiplexers 36:1, if a full crossbar structure were chosen.

Thereafter, the outputs of the crossbar 'Down' will be distributed to the CLBs as inputs. The LUT have 4 inputs. Actually, this LUT size has been used in commercial FPGAs by Altera and gives one of the best performances [123], chiefly because LUT complexity grows exponentially with the number of inputs. Altera uses 4-input LUTs in Adaptive Logic Modules (ALMs) to get logic functions having a higher number of inputs.

Then, we have a crossbar 'Up' linking the CLB's outputs to CLBs of the same cluster (feedback to the inputs) and to the cluster outputs. The structure of a crossbar 'Up' is

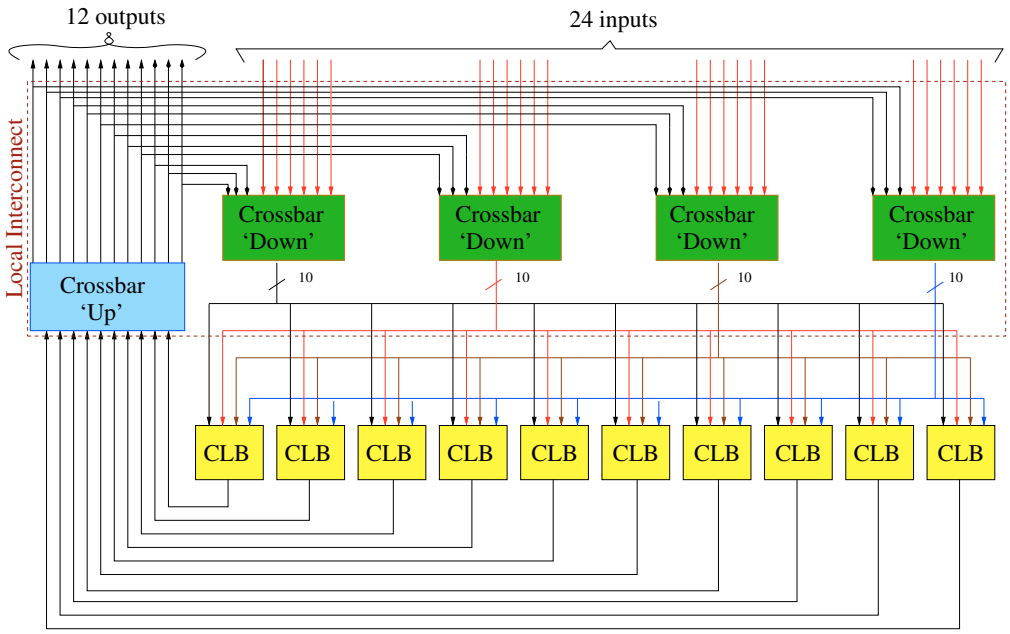


Figure 3.1: Structure of a mesh cluster.

perfectly similar to that of a crossbar 'Down', and it is composed of 12 multiplexers 10:1.

3.1.2 Analysis of the Impact of Simultaneous Errors on the Cluster Reliability

To compute reliability, we can use a simple and fast analytical method like SPR explained in Section 2.1.2.2. We are analyzing the impact of both soft errors and stuck-at defects on the cluster. The granularity level of analysis is the multiplexer 2:1. From this point on, we will call a multiplexer 2:1 a Mux2. A Mux2 can be considered as a 3-input, 1-ouput block. The third input corresponds to the selection bit. So, in case of soft errors, if the Mux2's reliability is noted q, the probability transfer matrix of the Mux2 is given in Equation 3.1, where the rows correspond to the inputs' combinations and the columns to the outputs'.

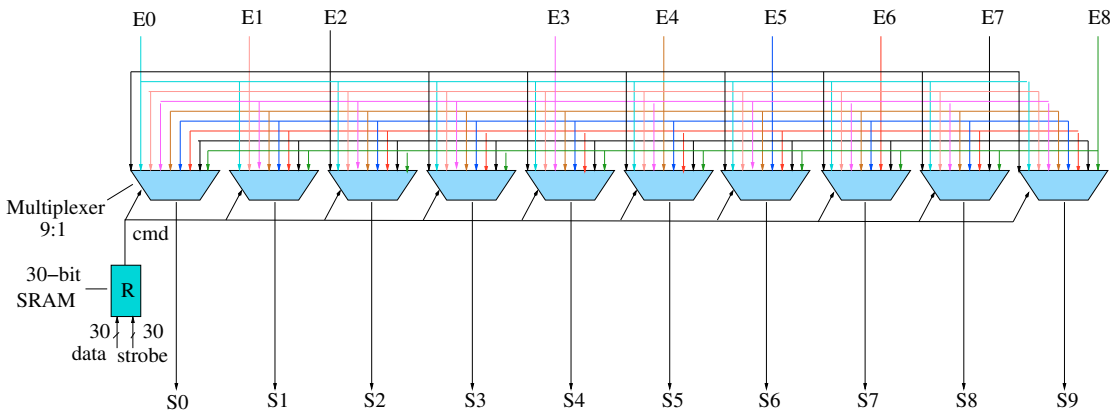


Figure 3.2: Structure of a Crossbar 'Down'.

$$PTM_{Mux2_q} = \begin{bmatrix} q & 1-q \\ q & 1-q \\ q & 1-q \\ 1-q & q \\ 1-q & q \\ q & 1-q \\ 1-q & q \\ 1-q & q \end{bmatrix} \tag{3.1}$$

The input patterns probability of the Mux2 are determined by the tensor product of its input signals probabilities, as expressed in Equation 3.2.

$$I_{Mux2} = SPM(in_1) \otimes SPM(in_2) \otimes SPM(in_3) \tag{3.2}$$

Finally, the signal probability matrix of the Mux2's output is given by Equation 3.3 where ITM'_{Mux2} is the transpose of the Mux2's ideal transfer matrix.

$$SPM(out_{Mux2}) = ITM'_{Mux2}.(I_{Mux2}.PTM_{Mux2}) \tag{3.3}$$

In our error analysis, we consider the following hypotheses:
- Any Mux2 in the CLBs and the crossbars are fault-prone;
- Since our objective is to analyze the cluster's inherent reliability, the cluster inputs are assumed fault-free;
- Configuration memory is supposed to be protected against faults.

We consider the most common types of errors: stucks-ats and bit flips. To model bit flips, the Mux2's PTM in Equation 3.1 is used. Stuck-at zero and stuck-at one defects are modeled through the PTMs of the Mux2s represented in the following, where s is the probability of stuck-at.

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ s & 1-s \\ s & 1-s \\ 1 & 0 \\ s & 1-s \\ s & 1-s \end{bmatrix}$$

Stuck-at zero PTM

$$\begin{bmatrix} 1-s & s \\ 1-s & s \\ 1-s & s \\ 0 & 1 \\ 0 & 1 \\ 1-s & s \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

Stuck-at one PTM

3.1.2.1 Methodology

The cluster is divided into three major blocks: a logic block composed of the LUTs, a 'Down' linking block composed of the 4 crossbars 'Down', and an 'Up' linking block composed of the crossbar 'Up'. The signal propagation with SPR is made through the three blocks, as shown in the flow graph of Figure 3.3. Actually, in each block, SPR needs the PTMs of all Mux2s, as well as the SPMs of the fault-free memory. These SPMs have 0.5 in their main diagonal and 0 in their antidiagonal. So are the SPMs of the primary

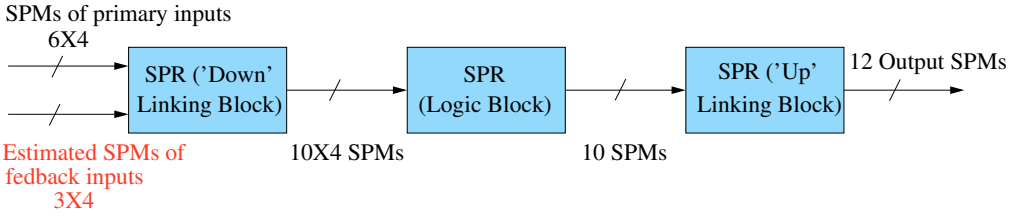


Figure 3.3: SPR applied on the cluster.

inputs. The SPMs of the feedbacked inputs (the three leftmost inputs of each crossbar 'Down' in Figure 3.3) can not be considered the same as those of the primary inputs, and have to be estimated. For this, we have to *cut* the feedback and procede as follows:

1. Choose arbitrarily a certain configuration memory: 160 selection bits in the 'Down' linking block, 160 SRAM bits in the logic block, and 48 selection bits in the 'Up' linking block.
2. Generate a binary uniformly distributed sequence for each input. The sequence length is equal to the number of iterations with which the simulation will be done.
3. Simulate the circuit with a large number of iterations (order of magnitude (10^4), without the feedback and having all Mux2s fault-free.
4. Observe statistically the binary distribution of the outputs, by counting the number of ones/zeros in each output and averaging over the number of iterations, so as to get the signal probabilities. Thus, the feedbacked SPMs are estimated.

After signal propagation with SPR, we calculate the signal reliability of each output as expressed in Equation 2.2. Therefore, the cluster reliability is given in Equation 3.4.

$$R_{Cluster} = \prod_{i=1}^{12} R_{output_i} \tag{3.4}$$

3.1.2.2 Results with a Fixed Memory Configuration

On the one hand, when varying the stuck-at probability s from 0.1% to 5% (as a mere example), the cluster reliability is computed and plotted in Figure 3.4 as a decreasing function of s, whether we have a stuck-at 0 or 1. Both curves are virtually alike. If we fix a minimum threshold for cluster reliability equal to 0.6, the maximum stuck-at probability is 1%.

On the other hand, when varying the Mux2's reliability q from 95% to 100%, the cluster reliability is computed and plotted in Figure 3.5 as an increasing function of q. The slope gets steeper and steeper starting from $q = 98\%$. This shows the impact of q on the cluster reliability which is only improved for high values of the Mux2's reliability. If the same threshold is considered, the minimum value of q is 99.5%, which corresponds to a maximum error probability of 0.5%. Hence, we deduce that the cluster is more vulnerable to bit flips than stuck-ats.

3.1.2.3 Results Varying Memory Configurations

We change the initial bitstream twice, and perform the same simulations again. Figure 3.6 shows the cluster reliability variations with an error probability range from 0% to

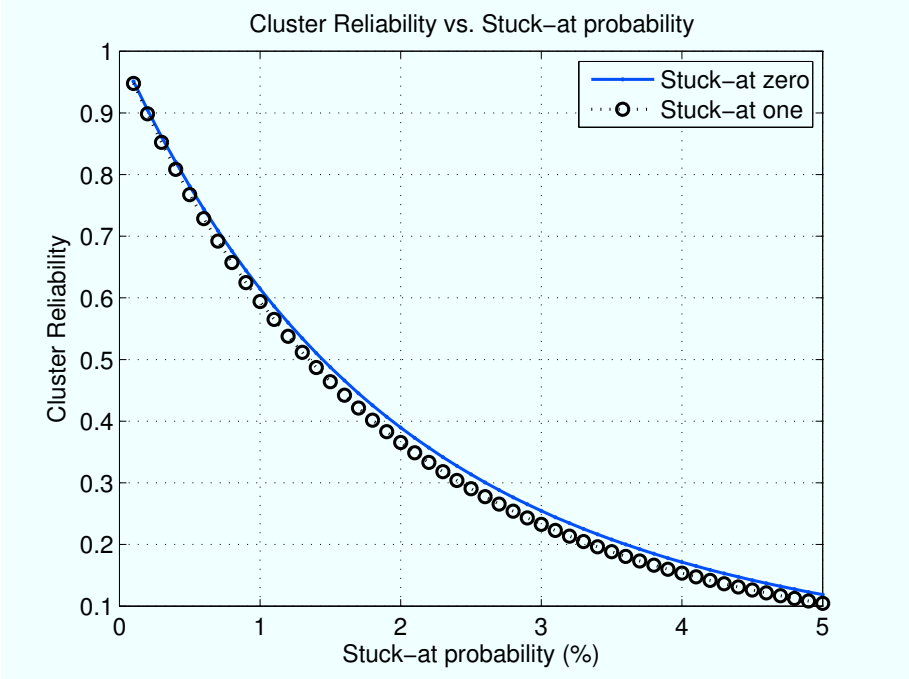


Figure 3.4: Cluster reliability variation with probability of stuck-at.

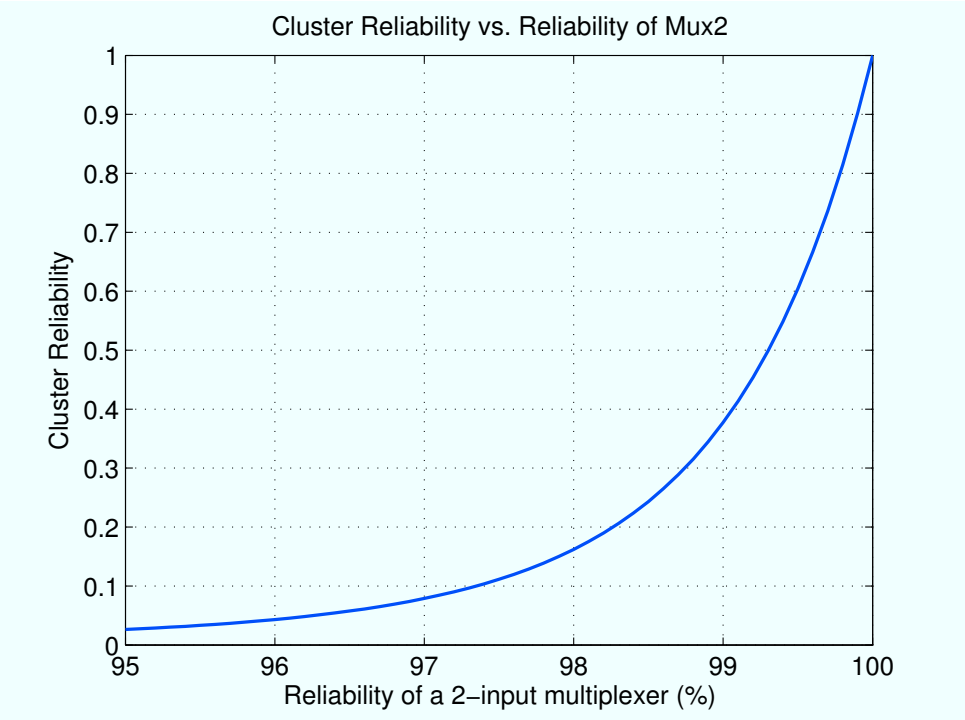


Figure 3.5: Cluster reliability variation with Mux2 reliability.

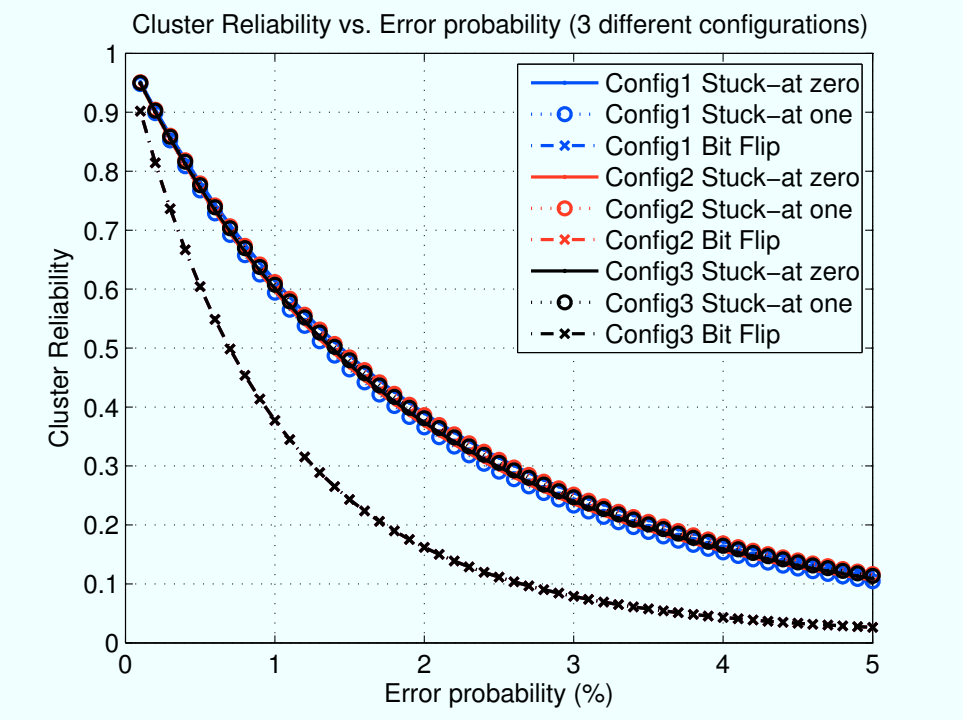


Figure 3.6: Cluster reliability variation with error probability.

5% for the three configurations.

For each memory configuration, we can notice that the cluster is more robust to stuck-at defects than to transient faults. Its reliability is virtually independent of the memory configuration, and this is more noticeable with bit flip errors where the three curves are perfectly superimposed.

3.1.3 Analysis of the Eligibility of the Cluster Components

We propose to make a selective hardening of components to improve the cluster's tolerance to faults, while minimizing the costs of this improvement. So, we are interested in sorting the cluster components by their contribution to increasing the cluster reliability. Hence, we keep the same analysis method used for multiple errors in Section 3.1.2, except that some multiplexers will be now hardened.

As the cluster proved to be more sensitive to bit flips, let us study the case of soft errors occurring in every Mux2, apart from a few that will be later specified. Each Mux2 has a value of reliability $q = 0.995$ (an example of value that could be modified), which results in a cluster reliability $R_{ref} = 0.6043$ that is our reference to assess the reliability improvement, as long as Mux2s are going to be hardened.

Actually, the hardening technique could be made either by design or by manufacturing. And for our simulations, a Mux2 is hardened if it has a value of reliability $q = 1$.

3.1.3.1 Hardening a Few Mux2s

In a first analysis, we are going to harden all Mux2s of a single block, while keeping the two other blocks faulty. Simulation results are summarized in Table 3.1, whose last

row gives the reliability gain G. The latter is defined relatively to the reference reliability value R_{ref} with faults in all Mux2s, as shown in Equation 3.5 where R is the reliability after hardening, and ΔR is the additive reliability gain.

Table 3.1 shows that the more blocks are hardened, the higher is the reliability gain. For example, hardening the logic block brings a bit more reliability gain (about 4% more) than hardening the 'Up' linking block, but the number of Mux2s that should be hardened is then multiplied by approximately 1.5. Moreover, hardening both logic and 'Up' linking blocks costs less and might achieve more reliability gain than hardening the 'Down' linking block. In this section, we are concerned with the reduction of hardening cost as much as possible.

Table 3.1: Hardening per block.

Blocks	**Logic**	**'Down' linking**	**'Up' linking**
Cluster reliability	0.7035	0.7586	0.6769
Number of Mux2s	150	320	108
Reliability Gain G	1.1641	1.2553	1.1201

$$G = \frac{R}{R_{ref}} = 1 + \frac{\Delta R}{R_{ref}} \tag{3.5}$$

Another metric, which is the *figure of merit* FM, is defined in Equation 3.6 where ΔR_{min} is the lowest additive reliability gain achieved by hardening one component of a considered block in the cluster.

$$FM = \frac{\Delta R}{\Delta R_{min}} \tag{3.6}$$

If only one LUT of the logic block is hardened, we obtain the results in Table 3.2. The LUT number is given according to where the inputs come from. For instance, the inputs of the first LUT come from each output $S0$ of a crossbar 'Down', as shown in Figure 3.2.

Table 3.2: Hardening per LUT.

LUT number	**{1,..,8}**	**{9,10}**
Reliability Gain G	1.0096	1.0394
Figure of Merit FM	1	4.1041

A difference in reliability improvement is noticed between the first 8 LUTs and the last two. The same asymmetry is noticed between the multiplexers 9:1 (Mux9s) of each crossbar 'Down' (see Table 3.3), since the outputs $S8$ and $S9$ of the last two Mux9s in each crossbar 'Down' feed the inputs of the two last LUTs. Nonetheless, no asymmetry is noticed in the 'Up' linking block, where the hardening of any multiplexer 10:1 (Mux10) gives a reliability gain of 1.0142.

Indeed, this asymmetry affecting both the upstream blocks of the cluster, is caused downstream by the architecture of the Mux10 in the crossbar 'Up'. This architecture is shown in Figure 3.7.a, where Mux2s are numbered from 1 to 9. The input wires of the fifth Mux2 are those coming directly from the outputs of LUTs 9 and 10. The fifth Mux2 is

Table 3.3: Hardening per Mux9.

Mux9 number	**{1,..,8}**	**{9,10}**
Reliability Gain G	1.0035	1.0144
Figure of Merit FM	1	4.1142

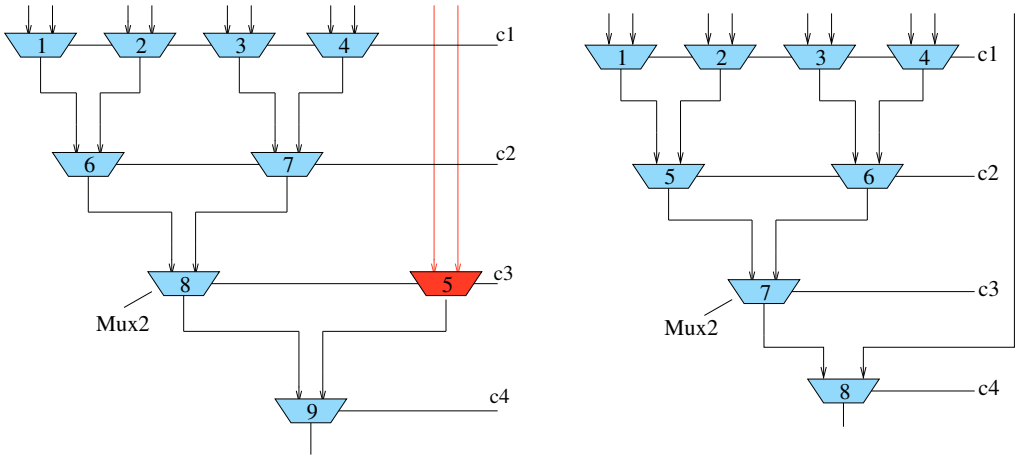


Figure 3.7: a. Mux10 b. Mux9

controlled by selection bit $c3$ which has a higher weight than $c1$ controlling the first four Mux2s whose inputs come from the first 8 LUTs. It should be noted that these results are independent of the value of q.

3.1.3.2 Hardening One Mux2

As long as the hardening technique is costly in terms of area, speed and power consumption, we are often to limit this technique to only one component of a circuit. From this standpoint, we will analyze the effect of hardening only one Mux2 in each cluster block, so that we can choose which Mux2 is the worthiest to be hardened.

Table 3.4 gives the reliability gain for each Mux2 in a Mux10 depicted in Figure 3.7.a. We see that the hardening of Mux2s of a same level (having the same selection bit) provides the same reliability gain. And the higher the level, the more improved the reliability.

Table 3.4: Hardening per Mux2 in a multiplexer 10:1.

Mux2 number	**{1,..,4}**	**{6,7}**	**{8,5}**	**9**
Reliability Gain G	1.0007	1.0012	1.0023	1.0048
Figure of Merit FM	1	1.7142	3.2857	6.8571

Eventually, we can conclude that the critical path is that connected to the fifth Mux2 in each Mux10 of the crossbar 'Up'. Table 3.5 and Table 3.6 give the reliability gain of each Mux2 in respectively a Mux9 (see Figure 3.7.b) and a LUT, in the critical path. The numbering of Mux2s in the LUT follows the same logic as for Mux9 and Mux10, starting

from lower levels to the higher ones.

Thus, we can deduce that the Mux2 of the last stage in the LUT of the critical path is the most eligible to be hardened, because it is the one which improves the most the cluster reliability, with a gain of 1.0096. So, a possible selective hardening strategy would concern the LUT of the critical path, either entirely or solely the most critical Mux2. Indeed, a reliability gain of approximately 1% for the cluster may seem insufficient but if hardening is done in all clusters of an FPGA, the reliability gain at system level will be then more important.

Table 3.5: Hardening per Mux2 in a multiplexer 9:1 of the critical path.

Mux2 number	**{1,..,4}**	**{5,6}**	**7**	**8**
Reliability Gain G	1.0006	1.0012	1.0024	1.0051
Figure of Merit FM	1	2	4	8.5

Table 3.6: Hardening per Mux2 in a LUT of the critical path.

Mux2 number	**{1,..,8}**	**{9,..,12}**	**{13,14}**	**15**
Reliability Gain G	1.0012	1.0023	1.0048	1.0096
Figure of Merit FM	1	1.9166	4	8

3.2 Analysis of the Switch Box

In the mesh of clusters FPGA topology, a cluster is surrounded by four switch boxes. The cluster's Inputs and Outputs (IOs) are directly routed by the switch box whose tree-like structure, containing upward and downward networks, was proposed in [59]. The switch box architecture is depicted in Figure 3.8. In fact, as shown in this figure, the cluster outputs get into two Upward Mini Switch Boxes (UMSBs) colored in red. Each UMSB is made up of three multiplexers 6:1 (Mux6). The multiplexers' outputs are controlled by the selection bits which are part of the whole configuration bitstream that is loaded while programming the FPGA for a given application. Then, the outputs of the UMSB are steered to two kinds of Downward Mini Switch Boxes (DMSBs). The DMSB2 would route the UMSB outputs to the same cluster again or to another one, and the DMSB1 would direct them somewhere else to another switch box. Each DMSB2 is composed of four multiplexers 4:1 and each DMSB1 of five multiplexers 5:1. Twice the number of DMSB1s defines the channel width that is the number of IOs between two adjacent switch boxes.

3.2.1 Reliability of the Switch Box Components

We consider the same assumptions as in the SPR analysis of the cluster. We take as an example $s = 0.005$ the probability of a Mux2 output to be stuck-at zero or one. This value of probability is just an example and proved not to have an impact on the subsequent results. Indeed, our analysis is a probabilistic modeling approach to compute

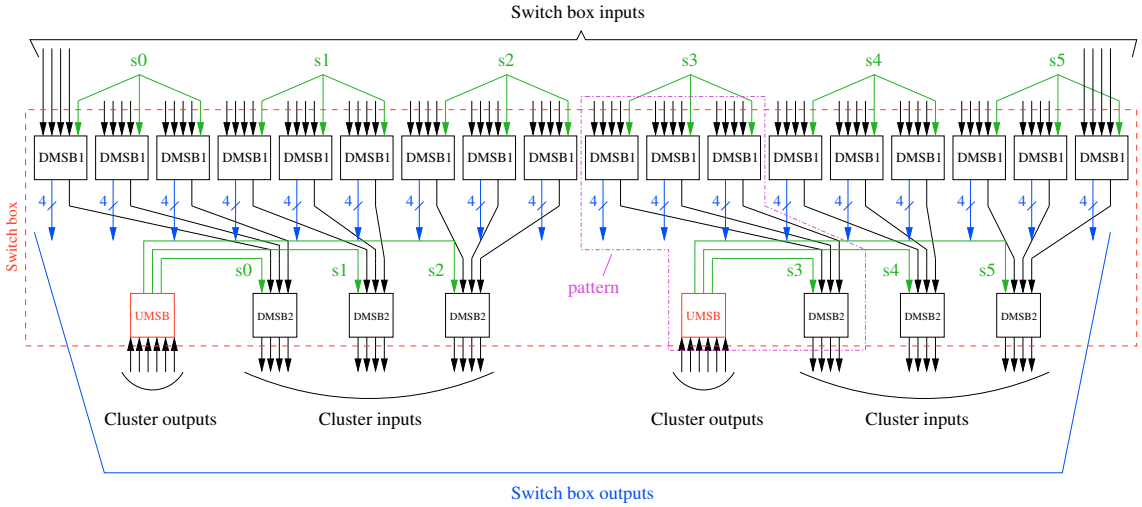


Figure 3.8: The switch box architecture.

the reliability of a circuit. It is perfectly compliant with the reality when using the most suitable probability value s provided by experimental tests.

SPR results are summarized in Table 3.7. Results for the stuck-at zero did not differ much from the stuck-at one. In a first analysis where every Mux2 was likely to fail, the reliability value of the switch box pattern equals 80.53%. Then, for a selective hardening purpose, we want to determine which block improves reliability the most.

Table 3.7: Reliability results for the switch box pattern.

Hardened Blocks	Reliability values (%)
None	80.53
UMSB	82.79
DMSB2	81.32
DMSB1	81.33

Thus, we harden all Mux2s of one given block in the switch box pattern. This means that the probability of stuck-at for the hardened Mux2 is $s = 0$. Results of selective hardening are given in the last three rows of Table 3.7. As it can be noticed, the UMSB is the most eligible component to be hardened in the switch box, bringing a reliability gain of roughly 2%. And as previously said, we remind that varying the value of s does not change the blocks' ranking. Nor does the ranking change with a soft error analysis.

Indeed, an additive gain of only 2% is realized by hardening the UMSB (Mux6). This can be explained by the fact that the Mux6 is still a small design whose number of transistors does not exceed 50. Furthermore, we should not forget that the UMSB is part of one single switch box. However, there could be thousands of switch boxes in an FPGA. Thus, the gain brought by each hardened UMSB should be then significant.

Chapter 4

Hardening at Multiplexer Level

This chapter proposes hardening solutions at the multiplexer level. The first solution uses NMR and concerns the logic block structure. Although the interconnect resources occupy almost 90% of an FPGA area [49], this ratio is inverted in terms of the actual use of resources. It means that an application mapped on an FPGA and using more than 80% of logic resources often uses only around 20% of the interconnect [124]. And in order to measure the importance of logic resources, it is interesting to know that for example, there are 474 240 LUTs in the Xilinx Virtex-6 XC6VLX760 [125]. Besides, the analysis performed in Section 3.1 reveals that the most critical component in the cluster is a LUT. Therefore, it is highly crucial to harden at least one CLB inside a cluster. A new CLB architecture baptized *Butterfly* is herein proposed. It is compared with other hardened CLB architectures in terms of reliability and cost penalties. Then, according to a proper design metric, the fittest architecture is selected.

The second hardening solution tries to avoid redundancy. In fact, since the routing network occupies the major FPGA area, it is highly important to have defect-tolerant switch boxes, without burdening their structures any more by adding extra hardware. In this context, we propose a method to identify the most reliable design among different candidates taken from a given library, thus avoiding to employ redundancy. This method is suitable for any library and can be generalized to any design. Our study case is the UMSB of the switch box that is a multiplexer 6:1 and that proved to be the most eligible one to be hardened according to the analysis performed in Section 3.2. Different possible architectures for the UMSB are built by assembling different standard cells from a 65nm industrial library. These architectures are studied under single defect injection by a tool that models several possible defects for a given design according to its extracted netlist. Eventually, the most robust architecture is picked.

4.1 Hardening the Cluster

This section presents a novel CLB structure having a *Butterfly* shape which embeds intrinsic redundance and is highly regular. Its tolerance to multiple simultaneous faults is analyzed and compared with the conventional CLB. And since the Butterfly architecture is based on NMR, it is greatly essential to have an optimized design of its voter. Otherwise, the voter is likely to add area overhead and degrade the reliability. Therefore, different structures for the voter are explored. The voter is first supposed fault-free in the fault tolerance analysis, then it is taken as fault-prone. Thereafter, a modified version of

the Butterfly architecture is examined. Then, both Butterfly architectures are compared to other hardened LUT designs with respect to reliability and cost penalties (area, power, delay and memory).

4.1.1 The Butterfly Architecture

4.1.1.1 Presentation

A conventional architecture for a 4-input LUT is depicted in Figure 4.1(a). As shown, the LUT's inputs A, B, C and D make 4 levels of multiplexers. The first level is controlled by selection bit A, that is the Least Significant Bit (LSB), and the fourth level by selection bit D, that is the Most Significant Bit (MSB). On the same figure, the 3-input LUT is represented in a dotted-line frame. If the 2-input multiplexer (Mux2) of the third level is affected by a bit flip error, the whole LUT-3 fails. However, if the bit-inversion error occurs in the second level, there is still half a chance that the transient fault can be masked. And if a Mux2 of the first level fails, there is a probability of $3/4$ that the LUT-3's output is correct. Therefore, the LUT's Mux2s are not of the same importance. That is why we call the conventional LUT architecture *Hierarchical*.

The idea behind the novel architecture is to break the hierarchy in the LUT architecture by duplicating Mux2s, so as to have the same number of them in each level. The novel architecture for a 4-input LUT is represented in Figure 4.1(b) where we can notice that the connections between the first and second levels of Mux2s involve four distinct Butterflies (a Butterfly shape as in Fast Fourier Transform (FFT) recursive algorithms [126]). Connections between the second and third levels of Mux2s engender two patterns of intermingled Butterflies. And we have one pattern of four twisted Butterflies between the third and fourth levels. The regularity of the patterns is verified as the number of LUT's inputs increases. Then, at the last stage of the Butterfly architecture, we have as many outputs as the number of Mux2s. Hence, a voter circuit is needed in order to decide what the final LUT's output will be.

4.1.1.2 Fault Tolerance Analysis

The behavior of the proposed Butterly design and that of the conventional one in response to bit-inversion errors are compared according to two different approaches. First, with the Matlab model, the tolerance to a single and a double transient fault is assessed. Then, with a full parameterizable HDL description that we developed for both architectures and the FIFA platform [3], we considered a large number of fault profiles, so as to validate simulation results and to have a broader scope of our analysis.

Preliminary Remarks Both approaches (simulations and emulations) for the study of fault tolerance/reliability have considered the following assumptions:

- Any Mux2 in the LUT are fault-prone. This means that single and multiple simultaneous faults are taken into account.
- The LUT inputs are assumed fault-free and configuration memory is supposed to be protected against faults. This is because our objective is to analyze the LUT's inherent fault tolerance/reliability.
- The voter is assumed fault-free by technology-hardening for instance, and performs a bitwise majority vote. If it is confronted with an equality between zeros and ones in the Mux2s' outputs, it chooses the first output by default.

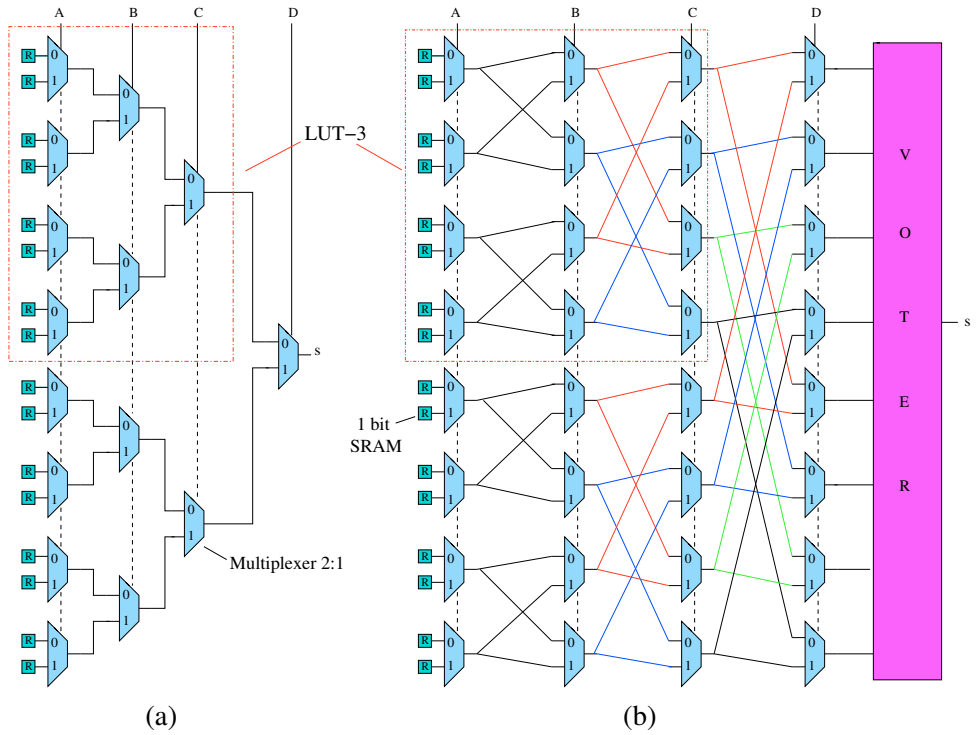


Figure 4.1: LUT-4 (a) Hierarchical (b) Butterfly

In order to compare the Butterfly and Hierarchical designs' fault tolerance, we evaluated the capabilities of fault masking offered by each solution. A fault masking means that the local fault has no effect on the CLB's output. So, the CLB is tolerant to such a fault. Considering that a bit flip is likely to happen in any level of Mux2s and taking into account all different combinations of LUT's inputs, we could calculate the masking rate P_c that is the probability of correct output. $P_c = \frac{m}{M}$ where m is the number of maskings and M is the number of all possible combinations of LUT's inputs and fault configurations. M is expressed in Equation 4.1, where N is the number of LUT's inputs, w is the number of Mux2s and k is the number of simultaneous faults.

$$M = 2^N C_k^w = 2^N \frac{w!}{(w-k)!k!} \tag{4.1}$$

Single and Double Fault Tolerance by Simulation We modeled a LUT-3 and a LUT-4 in Matlab, and analyzed the effect of a single ($k = 1$) then a double fault ($k = 2$). Simulation results are given in Figure 4.2 and Figure 4.3 for a single and a double fault respectively. We can observe the distribution of maskings along different levels of Mux2s.

Simulation results show that the Butterfly architecture, in all cases, has a higher masking rate than that of the Hierarchical architecture. So, the Butterfly is either as tolerant (in the first level of Mux2s) or more tolerant than the Hierarchical architecture, to single and double faults. Observing the histograms, we note that in the Hierarchical architecture, there are less and less maskings as we evolve to higher levels of Mux2s. However in the Butterfly architecture, the masking rate increases as we go from one level to a higher one. Thus, thanks to the Butterfly architecture, all levels of Mux2s are more tolerant to faults.

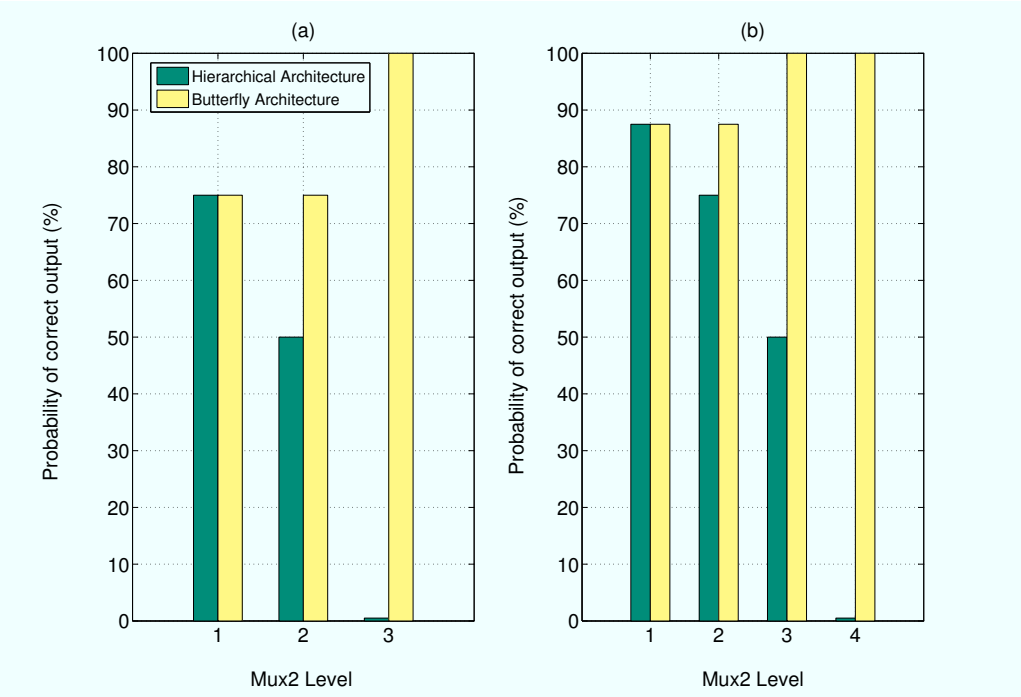


Figure 4.2: Single fault in (a) LUT-3 and (b) LUT-4

Fault Tolerance Generalization and Reliability The fault tolerance studied in the previous paragraph is generalized to a larger number of simultaneous faults and a larger number of inputs. Actually, a parameterizable Verilog description of both architectures was implemented. And the fault-injection-fault-analysis-based platform, that acts as a hardware accelerator using the Altera Cyclone© II 2C35 FPGA, analyzed the masking capabilities of the synthesized designs. Therafter, we could compute the reliability of the designs according to the PBR model [4] detailed in Section 2.1.2.4.

Emulation results

Thanks to the FIFA platform [3] presented in Section 2.1.1.2, we could inject as many simultaneous faults as the number of all Mux2s in the designs. Given the design and the number of faults, the platform returns the number of logical maskings, taking into account all possible combinations of the circuit's inputs. Table 4.1 shows fault analysis results for a Hierarchical and a Butterfly architecture, where N, k , m and P_c are as already defined above in this section.

Fault analysis results corroborate modeling simulation results, and show that the Butterfly architecture's fault tolerance surpasses that of the Hierarchical architecture for almost all combinations of (N, k).

As long as we have all values of P_c for LUT-3, we can compare the Butterfly's fault tolerance to that of a TMR-Hierarchical. Actually, we call a *TMR-Hierarchical* a triplicated Hierarchical design. For a design with w Mux2s, the overall masking rate $P_{c,o}$ is given by Equation 4.2. If the Hierarchical's overall masking rate is $P_{c,o}(H)$, that of the TMR-Hierarchical is expressed in Equation 4.3. Thus, the overall masking rates are presented in Table 4.2. We deduce that the Butterfly is more fault-tolerant than the TMR-Hierarchical for the LUT-3.

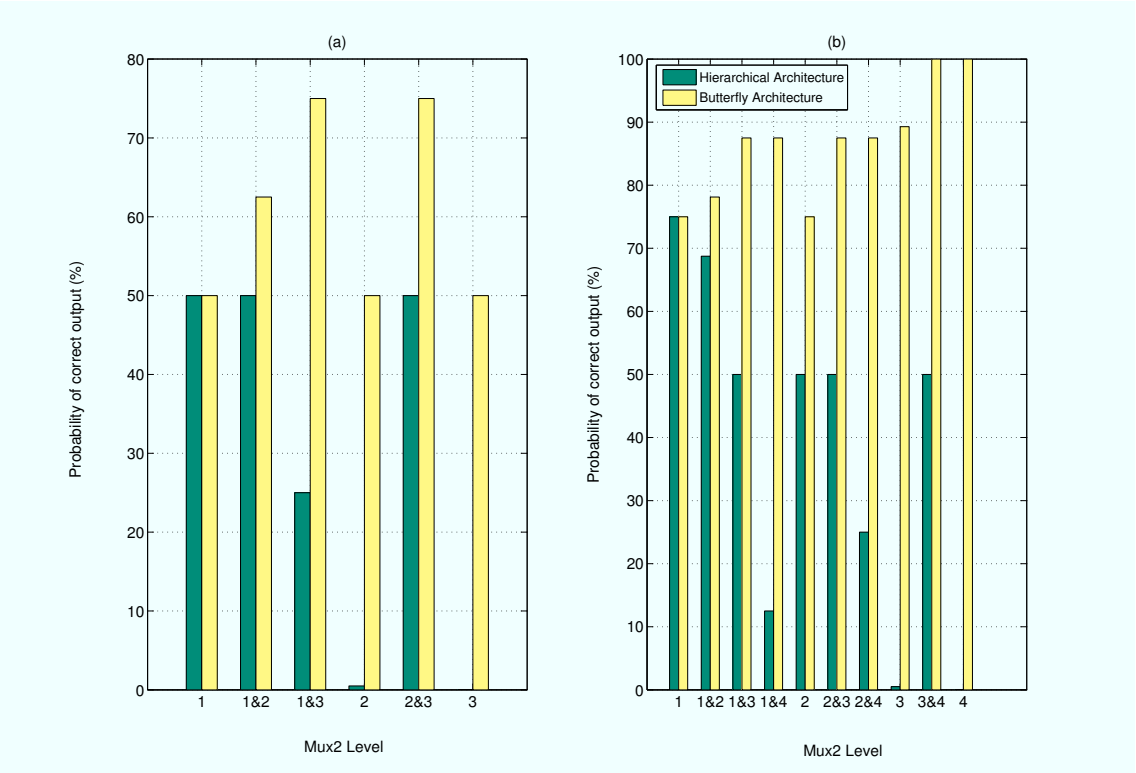


Figure 4.3: Double fault in (a) LUT-3 and (b) LUT-4

$$P_{c,o} = \frac{\sum_{k=1}^{w} m_k}{\sum_{k=1}^{w} M_k} \tag{4.2}$$

$$P_{c,o}(TMR - H) = 3 \times P_{c,o}(H)^2 \times (1 - P_{c,o}(H)) + P_{c,o}(H)^3 \tag{4.3}$$

Reliability of the two architectures

On the basis of the FPGA-based fault analysis results and the PBR model, we evaluated the reliability of the two architectures for a LUT-3 as an example. Every Mux2 is characterized by its reliability q, as defined in Section 2.1.2.4. The same notations used in that section are kept here.

Without loss of generality, we can take $q = 0.9971$ as an example to calculate the reliability of the LUT. We could compute reliability values for the LUT-3, especially that we had from FPGA-based fault analysis all values of m_k for $N = 3$. We could have calculated for both architectures all R_k terms of reliability as mentioned in Equation 2.9 of Section 2.1.2.4, but we realized that starting from R_3, terms become negligible within the sum. In fact, $\frac{R_3}{R_2}$ is in the order of 10^{-3} and 10^{-5} for Butterfly and Hierarchical architectures respectively. Computation results are shown in Table 4.3.

As we can see, R_0 corresponding to a fault-free operation is the predominant term, and it is superior for the Hierarchical architecture, as long as we have less Mux2s. However, the remaining terms R_k s.t. $k \neq 0$, are superior in the Butterfly architecture and take precedence in the sum, which makes the Butterfly architecture have a superior overall reliability. We can draw the same conclusion for the LUT-4 whose reliability results are

Table 4.1: Fault tolerance analysis

(a) Hierarchical

N	k	m	M	P_c (%)
3	1	32	56	57.14
	2	72	168	42.86
	3	128	280	45.71
	4	152	280	54.29
	5	96	168	57.14
	6	24	56	42.86
	7	0	8	0
4	1	176	240	73.33
	2	976	1680	58.1
	3	3696	7280	50.77
	4	10576	21840	48.42
5	1	832	992	83.87
	2	10720	14480	74.03
	3	91520	1.43×10^5	63.63
	4	582560	1.006×10^6	57.86
6	1	3648	4032	90.48
	2	103104	1.24×10^5	82.49
	3	1927360	2.54×10^6	75.84
	4	26813760	3.81×10^7	70.33

(b) Butterfly

N	k	m	M	P_c (%)
3	1	80	96	83.33
	2	344	528	65.15
	3	912	1760	51.82
	4	1808	3960	45.66
	5	2912	6336	45.96
	6	3696	7392	50
	7	3424	6336	54.04
	8	2152	3960	54.34
	9	848	1760	48.18
	10	184	528	34.85
	11	16	96	16.67
	12	0	8	0
4	1	480	512	93.75
	2	6928	7936	87.3
	3	63744	79360	80.32
	4	421216	5.75×10^5	73.21
5	1	2496	2560	97.5
	2	96032	101120	94.97
	3	2428928	2.62×10^6	92.39
	4	45404864	5.06×10^7	89.71
6	1	12160	12288	98.96
	2	1148992	1.17×10^6	97.91
	3	71986176	7.43×10^7	96.86

Table 4.2: Logical masking rates for the LUT-3

Hierarchical	**TMR-Hierarchical**	**Butterfly**
49.61%	49.42%	49.99%

shown in Table 4.4. The reliability gain obtained with Butterfly increases with the number of LUT's inputs. This can be checked for higher values of N.

For the Hierarchical and the Butterfly architectures, we calculated the reliability of LUT-3 and LUT-4, varying q in the interval $[0.95, 1]$. Results are shown in Figure 4.4 for the LUT-3 and the LUT-4. As we can see, reliability curves are increasing functions of q. And we can verify again that the Butterfly architecture is more reliable than the Hierarchical one.

4.1.1.3 Voter Architectures

From this standpoint on, let us note η the number of inputs to the voter. In this section, we introduce three possible designs for an η-input bitwise majority voter.

In fact, to make a decision based on the majority strategy, the voter needs to know the number of ones in the input sequence. If it is higher than $\frac{\eta}{2}$, the final output has to be one. Otherwise, it should be zero. We thought about two ways to perform this. The first way consists in sorting the input sequence and picking the $\frac{\eta}{2}$-th bit. The second way is to count the number of ones.

Table 4.3: Reliability of LUT-3

	Hierarchical	**Butterfly**
w	7	12
R_0	7.8390	7.7259
R_1	9.1196×10^{-2}	0.2247
R_2	5.9679×10^{-4}	2.81×10^{-3}
R_3	2.4107×10^{-8}	2.166×10^{-5}
r	0.9913	0.9942

Table 4.4: Reliability of LUT-4

	Hierarchical	**Butterfly**
w	7	12
R_0	15.3179	14.580
R_1	0.49	1.2721
R_2	7.904×10^{-3}	0.0534
R_3	8.7054×10^{-5}	1.42907×10^{-3}
r	0.9884	0.9941

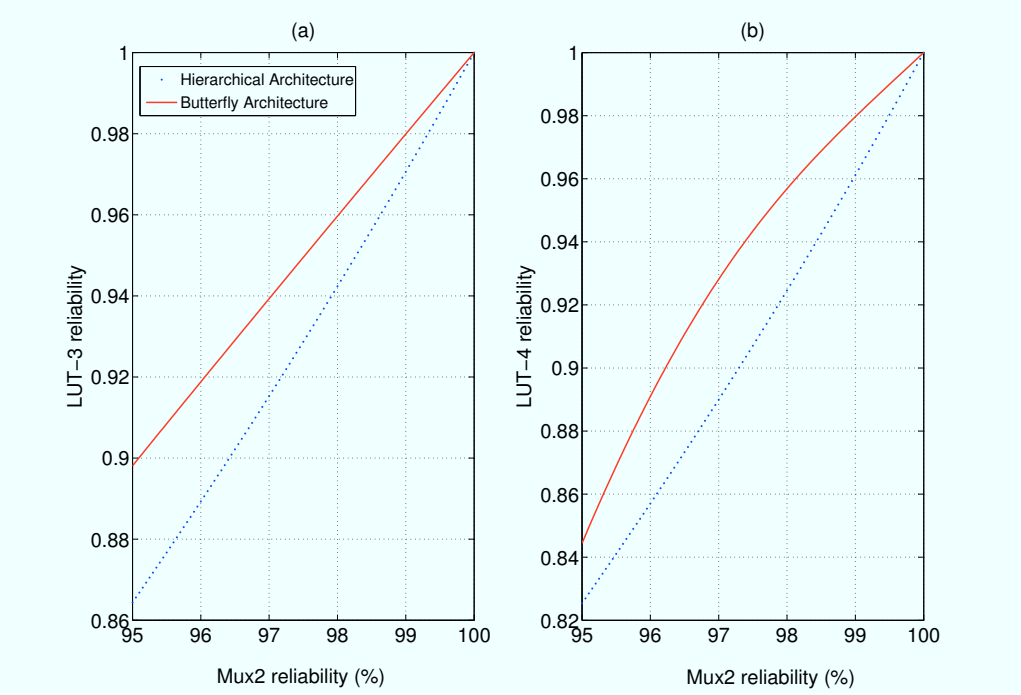


Figure 4.4: Reliability vs. q for (a) LUT-3 and (b) LUT-4

Parallelized BubbleSort Voter The bubble sort algorithm works by repeatedly stepping through the sequence to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order. In our case, the input sequence to be sorted is binary. Thus, the sorting is about putting all the maxima (the ones) and all the minima (the zeros) aside. The first two steps of the algorithm for an 8-input simple BubbleSort voter are depicted in Figure 4.5. It is made up of comparators (*Comp*). The number of comparators needed is equal to $\frac{\eta(\eta-1)}{2}$. And if we assume that one comparator needs T_c

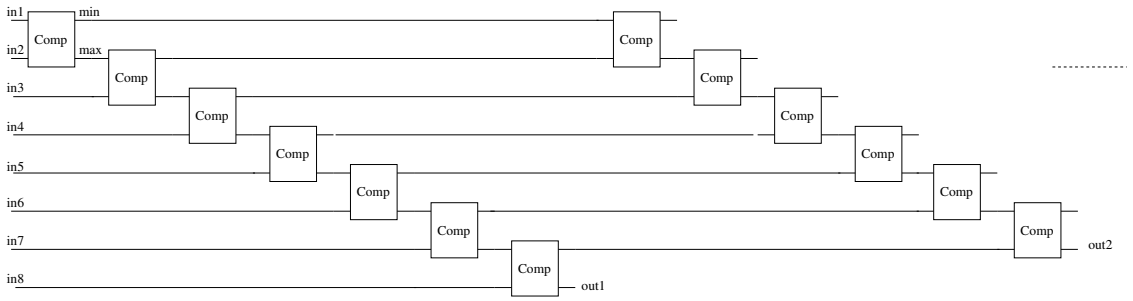


Figure 4.5: 8-input Simple BubbleSort voter design.

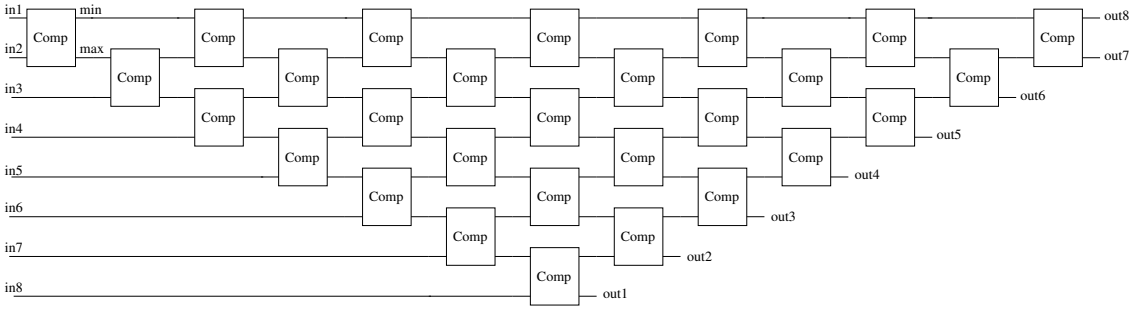


Figure 4.6: 8-input *Parallelized BubbleSort* voter design.

time to output the result, we will get the final sorted sequence after $\frac{\eta(\eta-1)}{2} \times T_c$.

Indeed, there is a way to reduce this time by making some comparators work in parallel. As soon as the inputs of a comparator are ready, the latter should treat them. Figure 4.6 shows an 8-input *Parallelized BubbleSort* voter. With this design, the final sorted sequence can be obtained after $(2\eta - 3) \times T_c$.

Yet, we could still alleviate the design by reducing the number of comparators. Since we are solely interested in the $\frac{\eta}{2}$-th bit of the sorted sequence, we should directly go for it. Figure 4.7 represents an 8-input optimized *Parallelized BubbleSort* voter. In the figure, the fourth maximum (max4) will be picked as the final output. Hence, some comparators were completey suppressed (the striped region on the figure) and some others became just half-comparators (represented with a slidelong bar across), i.e. they just output either the minimum or the maximum. Therefore, the number of comparators NB_{BS} used in the optimized *Parallelized BubbleSort* voter, and the computation time T_{BS} are given in Equations 4.4 and 4.5 respectively.

$$NB_{BS} = \frac{3}{8}\eta^2 - \left(\eta - \frac{5}{2}\right) \tag{4.4}$$

$$T_{BS} = \left(\frac{3}{2}\eta - 2\right) \times T_c \tag{4.5}$$

TwoByTwo Voter The baptized *TwoByTwo* voter is another kind of sorting voter that is represented in Figure 4.8 in the case of 8 inputs. It actually compares the adjacent input pairs alternatively through $(\eta - 1)$ stages. The number of comparators needed for this

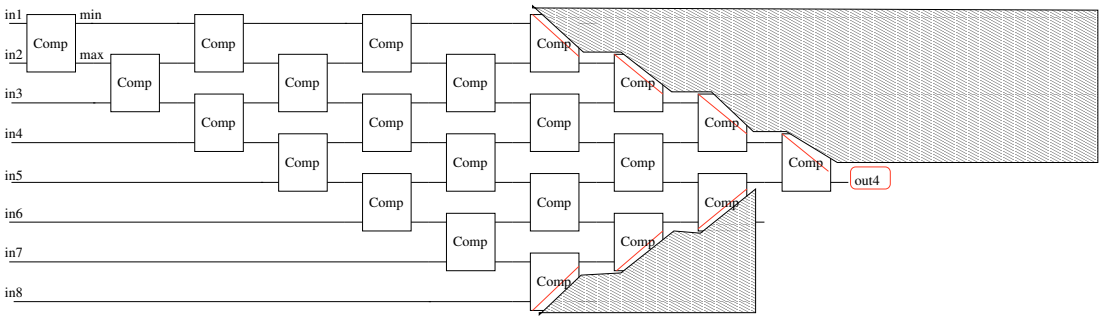


Figure 4.7: 8-input Optimized *Parallelized BubbleSort* voter design.

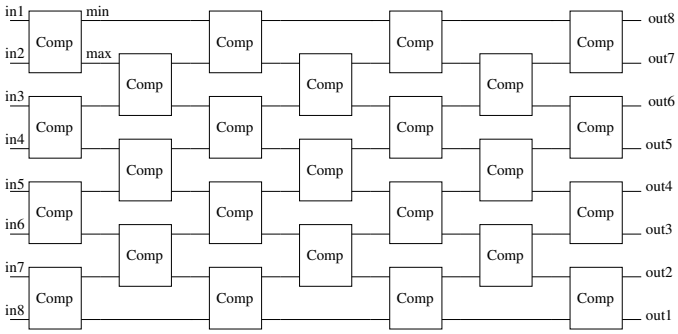


Figure 4.8: 8-input *TwoByTwo* voter design.

design is $2^{(2\gamma-1)} - 2^{\gamma} + 1$, where $\gamma = log_2(\eta)$. However, we can still reduce this number by picking just the fourth maximum, exactly like we did for the *Parallelized BubbleSort* voter. The optimized TwoByTwo design for 8 inputs is depicted in Figure 4.9. So, the number of comparators NB_{2By2} used in the optimized *TwoByTwo* voter, and the computation time T_{2By2} are given in Equations 4.6 4.7 respectively.

$$NB_{2By2} = \frac{3}{8}\eta^2 - \left(\eta - \frac{1}{2}\right) \tag{4.6}$$

$$T_{2By2} = (\eta - 1) \times T_c \tag{4.7}$$

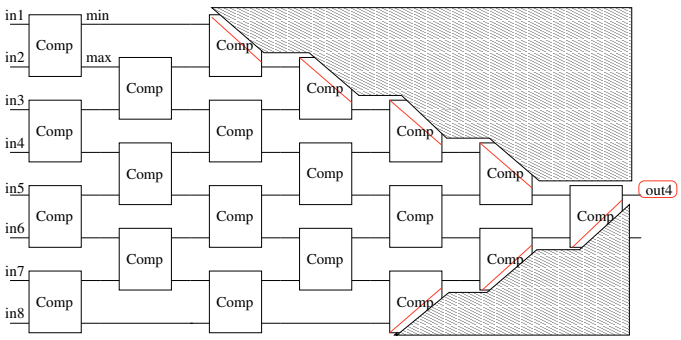


Figure 4.9: 8-input Optimized *TwoByTwo* voter design.

Since $NB_{2By2} < NB_{BS}$ and $T_{2By2} < T_{BS}$, the TwoByTwo design is cheaper and faster. Hence, we can already select the TwoByTwo over the Bubble Sort.

CarryAdd Voter The *CarryAdd* voter uses a different approach from sorting. If η is the number of inputs to the voter, the *CarryAdd* voter counts the bits to check whether the number of ones exceeds $\frac{\eta}{2}$. In order to optimize the design, it is pointless counting the actual sum. We had better add the carry bits till the $log_2(\frac{\eta}{2})$ weight. Then, we check if at least one of the carry bits is equal to one at that stage. If so, the final output is one. The 8-input *CarryAdd* voter design is shown in Figure 4.10. For this case, only two stages of full adders (*FA*) are required. Thereafter, the carry bits of weight 2 are assessed at the third stage. More generally, for an N-input LUT, the hardware H required by the *CarryAdd* voter is given in Equation (4.8) where FA, $AND2$ and $OR2$ stand for a full adder, a 2-input AND gate and a 2-input OR gate respectively.

$$H = (2^{N-1} - N)FA + (N-2)AND2 + OR2 \tag{4.8}$$

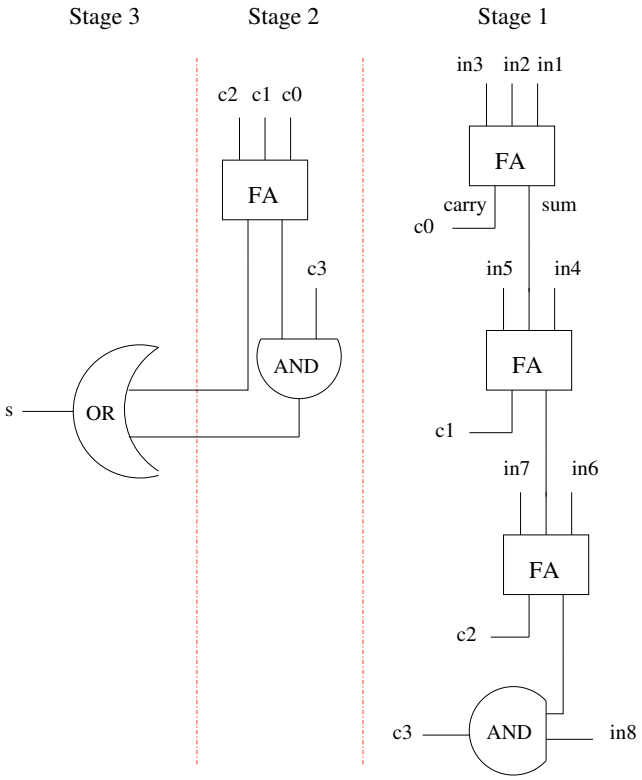


Figure 4.10: 8-input *CarryAdd* voter design.

As long as it is difficult to analytically compare the *TwoByTwo* and the *CarryAdd* voters in terms of area and speed, we resorted to a synthesis tool for ASICs that is Cadence Encounter RTL Compiler [127]. We have used STM 65nm CMOS technology and CORE65LPSVT standard cell library. This library is optimized for low power applications. The supply voltage is $1.2V$, the nominal junction temperature is $25°C$ and Verilog description has been used as the design entry to Cadence Encounter RTL Compiler. Dynamic power has been calculated at a clock frequency of $100MHz$. Results for 8-input

Table 4.5: Synthesis of the 8-input voters in STM 65nm CMOS.

8-input Voters	*TwoByTwo*	*CarryAdd*
Cell Area (μm^2)	85.28	41.60
Net Area (μm^2)	126.49	43.29
Total Area (μm^2)	211.77	84.89
Worst Path (ps)	624.60	707.10
Maximum Frequency (GHz)	1.601	1.414
Leakage Power (nW)	4.42	1.72
Dynamic Power (nW) @ $100MHz$	9928.09	8367.4

voters are given in Table 4.5. We note that the results are post synthesis with back-annotation (i.e. considering an accurate switching activity report) and do not consider the post layout routing.

From the synthesis results, we can deduce that the 8-input *CarryAdd* voter is less expensive in terms of area and power but is slower than its *TwoByTwo* counterpart.

The Butterfly LUT-4 architecture in its original version presents 8 inputs at the voter. Thus, we are going to elect either the *TwoByTwo* voter or the *CarryAdd* one, on the basis of the synthesis results shown in Table 4.5.

As a matter of fact, the *TwoByTwo* voter outpasses the *CarryAdd* by 149.46% of the total area overhead, while the latter is slower by only 11.69%. Besides, the *CarryAdd* consumes less power than the *TwoByTwo* by 15.74%. Hence, the *CarryAdd* voter is elected for the original Butterfly.

4.1.1.4 The Modified Butterfly

As we are willing to find a good tradeoff between reliability and area overhead, we had the idea to alleviate the Butterfly architecture by reducing the number of Mux2s at the last stage to only three. In this case, we merely have a TMR voter. Figure 4.11 represents the modified version of the Butterfly design.

4.1.2 Comparative Study of CLBs with Fault-Free Voters

We will be comparing 4-input LUTs, as long as they are used in the mesh of clusters FPGA architecture we are studying. Since the most appropriate voter is selected for the original LUT-4 Butterfly architecture, it is interesting to confront the latter with other LUT-4 designs so as to have an idea about the current state of the art.

4.1.2.1 Preliminary Remarks

The LUT-4 architectures that we are comparing are:
- The Hierarchical (Hr), having 15 Mux2s;
- The Hierarchical with TMR applied on the most critical Mux2 in the last level (Hr-3m), having 17 Mux2s;
- The original Butterfly (Bf), having 32 Mux2s;
- The modified Butterfly (Bf-M), having 25 Mux2s;

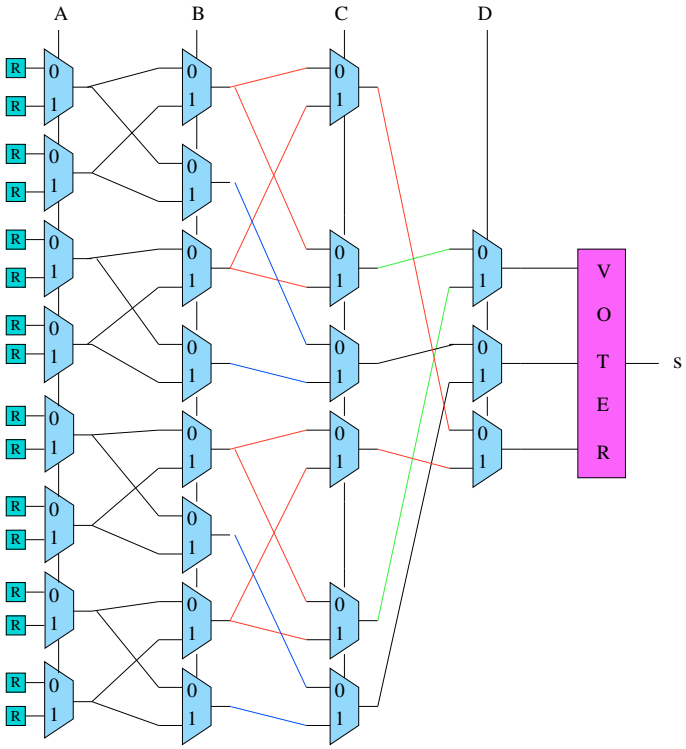


Figure 4.11: LUT-4 Butterfly Modified design.

– The triplicated Hierarchical (TMR-Hr), having 45 Mux2s;
– The architecture (Remap) described in [128] using the remap property of the logical functions and the Duplicate With Comparison (DWC) technique. This architecture is able to detect a single fault, and to correct it only if there is no self map in the Boolean function [128]. In this paper, we considered the described architecture without its detection circuitry, having 46 Mux2s;

The comparison criteria of the designs are:

– The area, maximal frequency and power using STM 65nm CMOS technology under the same conditions described in Section 4.1.1.3;
– The number of configuration bits required in the SRAM memory;
– The logical masking of k simultaneous faults that can occur in any Mux2, considering all possible configurations of the selection bits. We varied k from 1 to 4. In our case, all Mux2s are fault-prone. And a Mux2 fails when its output bit is inverted. Logical masking is calculated using the FIFA platform;
– The reliability of the LUTs where every Mux2 is characterized by its reliability q. Without loss of generality, we took $q = 0.96$ as an example to calculate the LUT-4's reliability. Using the logical masking figures of multiple simultaneous faults obtained with the FIFA platform, the reliability is computed according to the PBR model.

For the synthesis of Bf, we took into account the CarryAdd voter. For the synthesis of Bf-M, Hr-3m and TMR-Hr designs, we considered a standard TMR voter. And when computing the logical masking and overall reliability, we assumed the voters fault-free.

Table 4.6: Synthesis results for the LUT-4 architectures.

	Total Area (μm^2)	**Worst Path (ps)**	**Maximum Frequency (GHz)**	**Leakage Power (nW)**	**Switching Power (μW)**	**Configuration bits**
Hr	165.18	371.4	2.692	4.61	8.79	16
Hr-3m	193.27	513.1	1.948	5.40	12.63	16
Bf	357.15	513	1.949	10.06	30.46	16
Bf-M	266.78	514.7	1.942	7.9	19.06	16
TMR-Hr	492.33	480	2.083	14.01	26.88	48
Remap	490.07	565	1.769	14.73	23.40	40

Table 4.7: Logical masking and reliability computation results.

LUT-4	**k fault logical masking (%)**				**Reliability (%)**
	$k=1$	$k=2$	$k=3$	$k=4$	
Hr	73.33	58.1	50.77	48.42	85.8
Hr-3m	82.35	66.91	56.03	50.04	88.5
Bf	93.75	87.3	80.32	73.21	91.2
Bf-M	92	80.67	69.83	61.19	90
TMR-Hr	100	95.15	88.39	81.31	92.1
Remap	91.11	82.22	73.98	66.67	82.5

4.1.2.2 Comparison Results

Table 4.6 gives synthesis results for the mentioned LUT-4 architectures. Note that the last column is not part of the synthesis results. In fact, only the combinational part of the LUT-4 is synthesized and not the SRAM memory. Table 4.7 gives computation results of the logical masking and reliability.

In order to compare the architectures with respect to the Hr design, we are going to define some metrics:

- The masking gain of k faults defined as $M_k = \frac{M}{M_{Hr}}$ where M and M_{Hr} are the masking rates of a LUT-4 architecture under study and of Hr respectively;
- The reliability gain defined as $G_r = \frac{r}{r_{Hr}}$ where r and r_{Hr} are the reliabilities of a LUT-4 architecture under study and of Hr respectively;
- The performance degradation defined as $O_T = \frac{T}{T_{Hr}}$ where T and T_{Hr} are the worst paths of a LUT-4 architecture under study and of Hr respectively;
- The area overhead defined as $O_A = \frac{A}{A_{Hr}}$ where A and A_{Hr} are the total areas of a LUT-4 architecture under study and of Hr respectively;
- The power overhead defined as $O_P = \frac{P}{P_{Hr}}$ where P and P_{Hr} are the total power consumptions (leakage and switching) of a LUT-4 architecture under study and of Hr respectively;
- The memory overhead defined as $O_M = \frac{M}{M_{Hr}}$ where M and M_{Hr} are the sizes of SRAM memory in a LUT-4 architecture under study and of Hr respectively.

Table 4.8: Gains and overheads of the LUT-4 architectures with respect to the Hierarchical LUT-4 architecture.

LUT-4	G_r	M_1	M_2	M_3	M_4	O_T	O_A	O_P	O_M
Hr-3m	1.031	1.123	1.151	1.103	1.033	1.381	1.170	1.436	1
Bf	1.062	1.278	1.502	1.582	1.511	1.381	2.162	3.464	1
Bf-M	1.048	1.254	1.388	1.375	1.263	1.385	1.615	2.168	1
TMR-Hr	1.073	1.363	1.637	1.740	1.679	1.292	2.980	3.057	3
Remap	0.961	1.242	1.415	1.457	1.376	1.521	2.966	2.661	2.5

Table 4.8 gives computation results for the metrics we just mentioned. Examining the reliability and logical masking gains, we see that the TMR-Hr is ahead of all designs, then come the Bf and Bf-M. TMR-Hr is also the fastest just before Hr-3m. Nevertheless, the TMR-Hr is the worst in terms of SRAM memory requirements and it is actually the most consuming in area and power along with the Remap. On the other hand, the least costly design is the Hr-3m, then comes the Bf-M in second place.

From these metrics, we could also define another one that assesses a design in all its aspects. This metric Q is expressed in Equation (4.9) where $w_i, i = 1..10$ are weights applied on the metrics either to appreciate or to penalize them. These weights are determined by the designer according to the design requirements. Depending on the target application, the designer decides whether logical masking gain is more important than area overhead or vice-versa, and by which amount. As it is defined, Q is a quality metric for a design. The higher Q, the better the design.

$$Q = \frac{\left(\sum_{k=1}^{4} w_k M_k\right)^{w_5} G_r^{w_6}}{O_A^{w_7} O_P^{w_8} O_M^{w_9} O_T^{w_{10}}} \tag{4.9}$$

Two different examples for calculating the global metric Q are presented in Table 4.9. In the first example, the designer is giving the same importance to all parameters. Hence, all the weights in Q are set equal. Therefore, in this case, the global metric Q is the highest for Hr-3m, then for the Bf-M, and then for the TMR-Hr.

In the second example, the designer is more concerned about reliability and logical masking than about all overheads. This is the case for critical applications, e.g. spatial, military, health devices, where human safety is at stake [129]. For instance in our example, the designer would care the most for the overall logical masking and then for the reliability. Therefore, the weights attributed to the logical masking gain and reliability gain are respectively $w_5 = 4$ and $w_6 = 3$. Single and multiple faults masking gains are all judged of the same importance. This means $w_i = 1, i = 1..4$. On the other hand, since overheads do not matter, all their weights are equal, i.e. $w_i = 1, i = 7..10$. Computation results corresponding to this example are given in the second column of Table 4.9. Hence, in this case, the Bf-M design is ranked first. Then, come the Hr-3m in second place and the Bf in third place.

4.1.3 Comparative Study of CLBs with Fault-Prone Voters

Assuming a fault-free voter, the Butterfly proved to be more tolerant to multiple faults and more reliable than its conventional counterpart. In this section, voters are fault-

Table 4.9: Two examples for calculating the global metric Q.

LUT-4	Q example 1	Q example 2
Hr-3m	1.959	178.64
Bf	0.603	137.78
Bf-M	1.141	184.47
TMR-Hr	0.195	59.39
Remap	0.175	26.86

prone and we introduce another voter for the Butterfly, called *Cascade-TMR*, that is not only more area-efficient than the *CarryAdd*, but also more fault-tolerant. The *Cascade-TMR* is actually based on cascaded TMR voters. We consider three different TMR voters: the standard one and two other fault-tolerant voters. We inject multiple faults in the different CLB designs and assess their reliability, as we did in the previous section. After considering the synthesis results with STM 65nm CMOS technology, we compare the CLB designs with respect to the conventional architecture in terms of reliability gain on one hand, and performance degradation, area, power and memory overheads on the other hand. Since the Remap design studied in the previous section proved to be the worst one using any design metric, we will not include it in the current study.

4.1.3.1 The Cascade-TMR and the Fault-Tolerant TMR Voters

The baptized *Cascade-TMR* voter consists of TMR voters deployed in a cascade structure. The number of voters required depends on the number of inputs. For instance, with 8 inputs, four TMR voters are needed. This case is depicted in Figure 4.12, where there are three TMR voters in the first stage and then one last voter in the second stage. Note that one of the inputs (input 1 in Figure 4.12) is redundant.

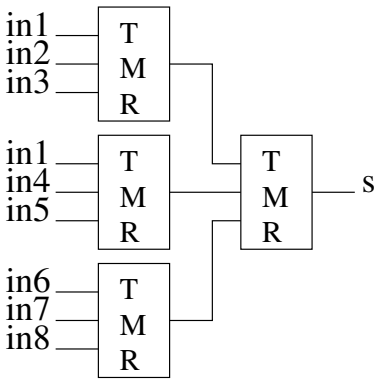


Figure 4.12: 8-input *Cascade-TMR* voter design.

Regarding the inner structure of the TMR voter, we consider three different architectures: the standard majority voter (*Stand*) and two other fault-tolerant voters. We call the first fault-tolerant TMR voter *Ban* [1]. It is represented in Figure 4.13. This voter proved to be tolerant to single faults. The second fault-tolerant TMR voter [2], involving more hardware than the first one, is called *Kshir* and is represented in Figure 4.14. It

was shown in [2] that if all the voter inputs are correct, then the voter output follows the input, irrespective of faults occurring in the voter circuit.

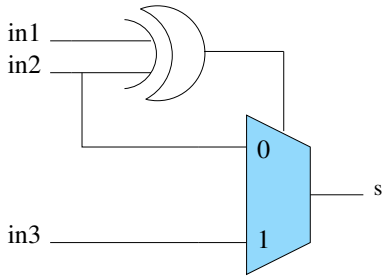


Figure 4.13: Ban's voter as in [1].

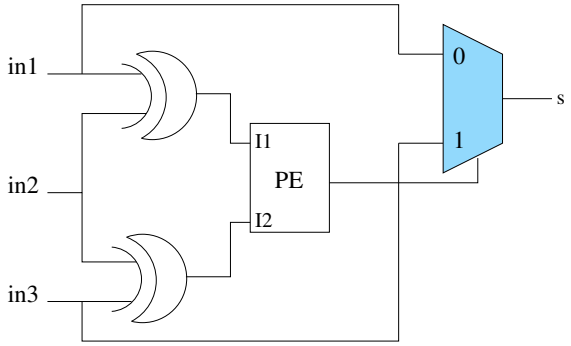
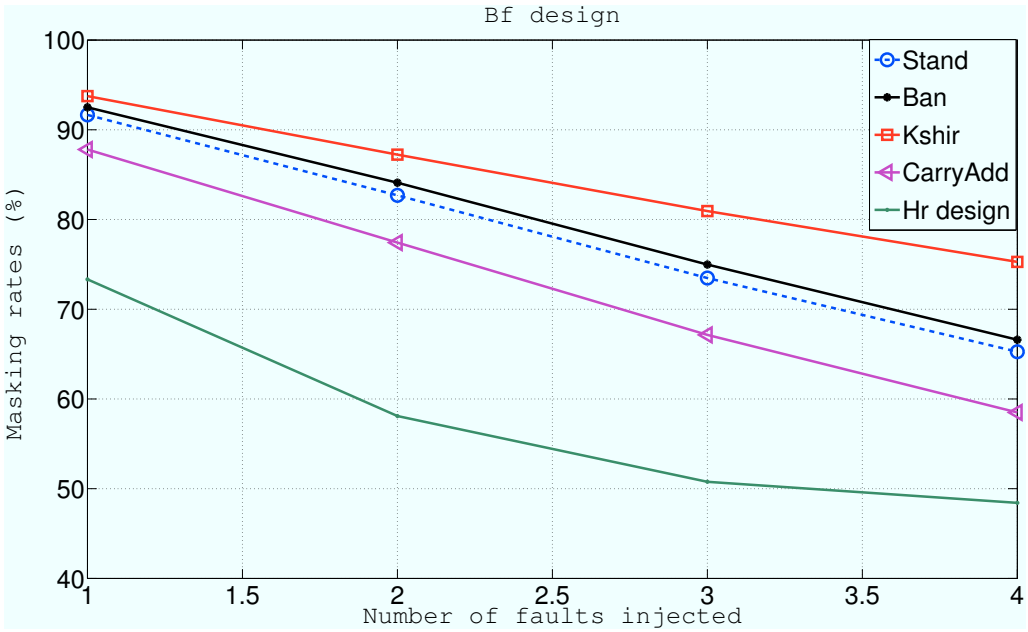


Figure 4.14: Kshirsagar's voter as in [2].

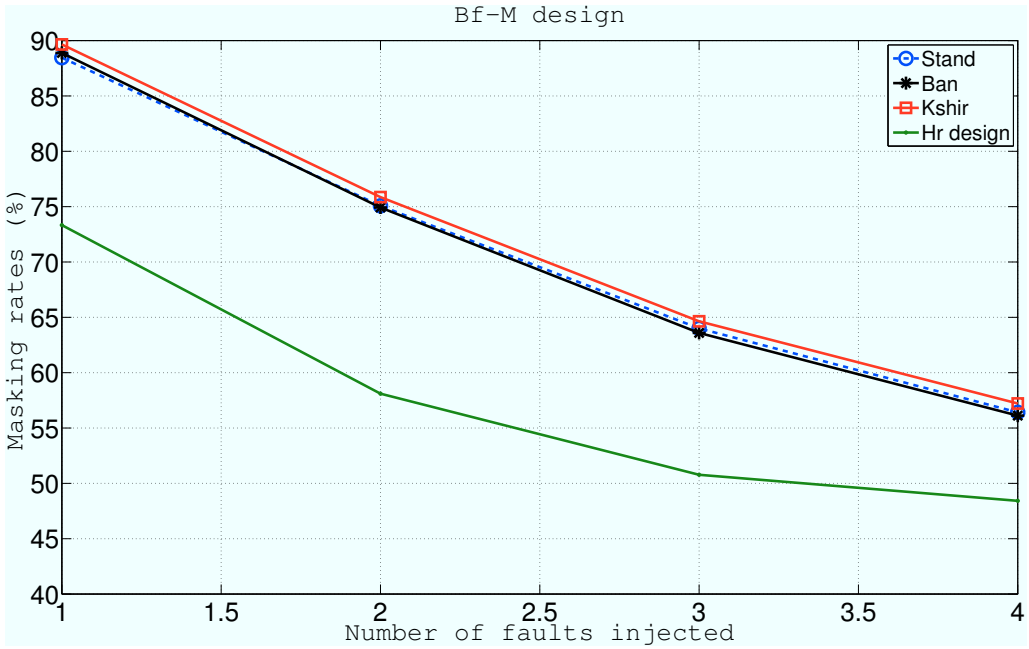
4.1.3.2 Fault Injection and Reliability

Before proceeding to fault injection, we synthesized the aforementioned CLB designs, so as to inject faults proportionally to the area of logic components. Using the FIFA platform, we injected up to 4 simultaneous bit flips in our designs under test. Results for the CLB designs, using different voters are shown in Figure 4.15 and Figure 4.16. The logical masking rates of each CLB design are confronted with those of the Hr, depicted with a full line on each subfigure. We notice that the TMR-Hr realizes the highest logical masking gains for any number of faults injected, and this is virtually independent of the voter choice. The lowest logical masking gains are accomplished by the *Hr-3m* design in which the use of the *Kshir* TMR voter increases the masking capability by 3% at the best case. More important increase in masking gain can be achieved with the cascaded *Kshir* voter for the Bf design. In this case, masking gains are in the order of 30% approaching those of TMR-Hr. The *CarryAdd* voter is the most vulnerable to bit flips.

Reliability is still calculated according to the PBR model, and is expressed in Equation (2.9) of Section 2.1.2.4. Reliability computation results for the CLBs are shown in Figure 4.17 and Figure 4.18. Each design's reliability is compared to the Hr design's which is equal to 85.81%. Since our prime target is to enhance the CLB's conventional design, we can already eliminate some designs that proved to be less reliable than the Hr: the Hr-3m with any voter and the Bf using the cascaded *Kshir* and the *CarryAdd* voters. As

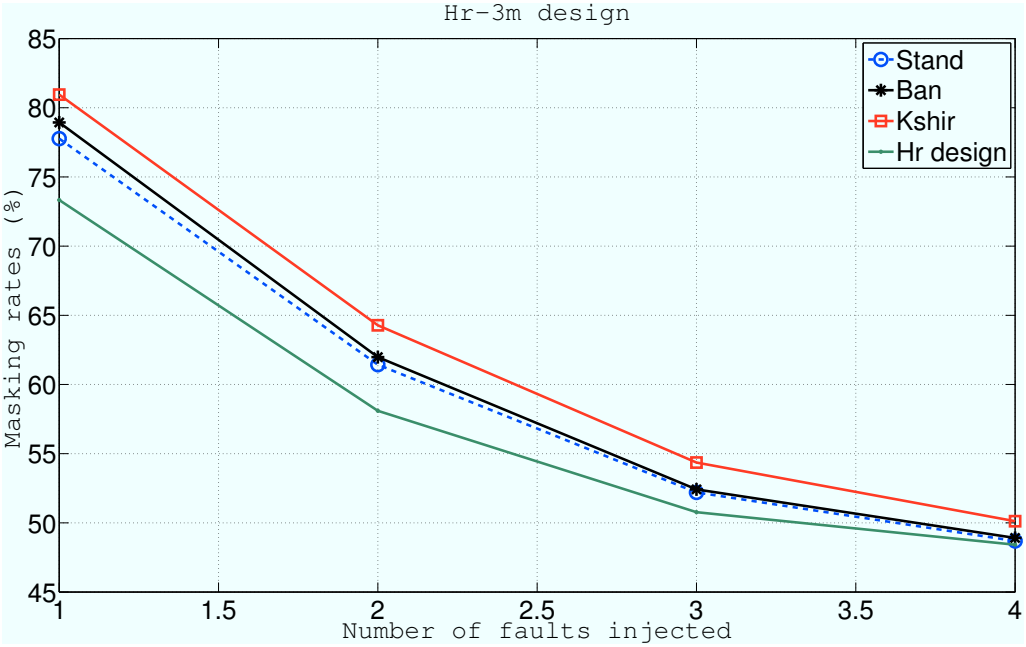


(a) Bf

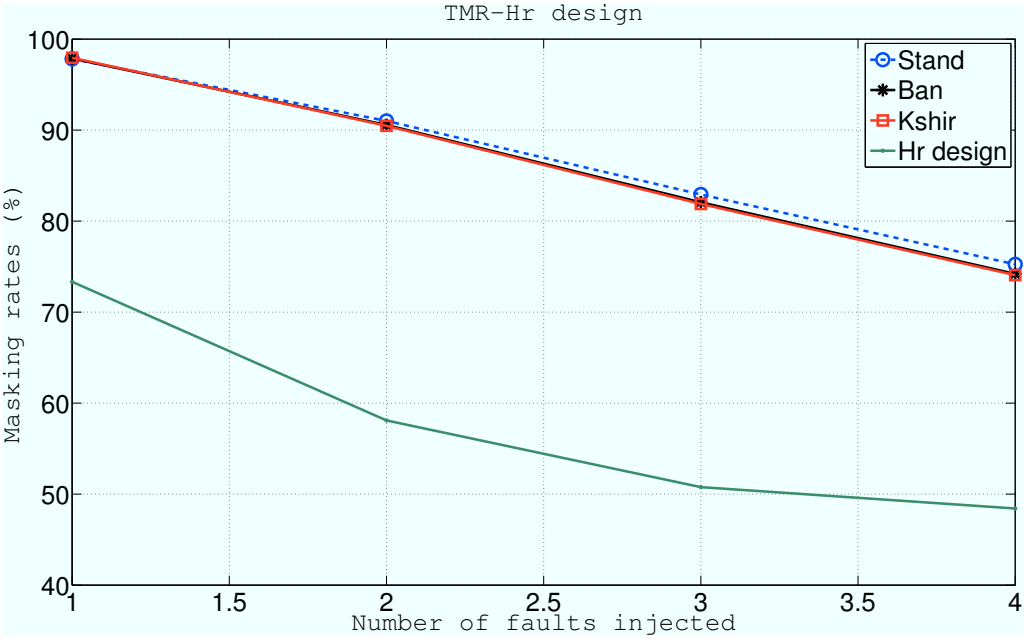


(b) Bf-M

Figure 4.15: Masking rates of hardened CLB designs (1)



(a) Hr-3m



(b) TMR-Hr

Figure 4.16: Masking rates of hardened CLB designs (2)

Table 4.10: Synthesis results for the LUT-4 architectures.

	Total Area (μm^2)	**Worst Path (ps)**	**Leakage Power (nW)**	**Switching Power (μW)**	**Configuration bits**
Hr	165.18	371.4	4.61	3.433	16
Bf Stand	360.22	642.4	9.15	6.909	16
Bf Ban	389.93	913.0	10.96	6.938	16
Bf-M Stand	266.78	514.7	7.00	5.510	16
Bf-M Ban	274.20	627.3	7.45	5.517	16
Bf-M Kshir	288.22	711.3	7.78	5.528	16
TMR-Hr Stand	492.33	480.0	12.51	9.715	48
TMR-Hr Ban	499.76	606.5	12.96	9.722	48
TMR-Hr Kshir	513.78	675.4	13.30	9.733	48

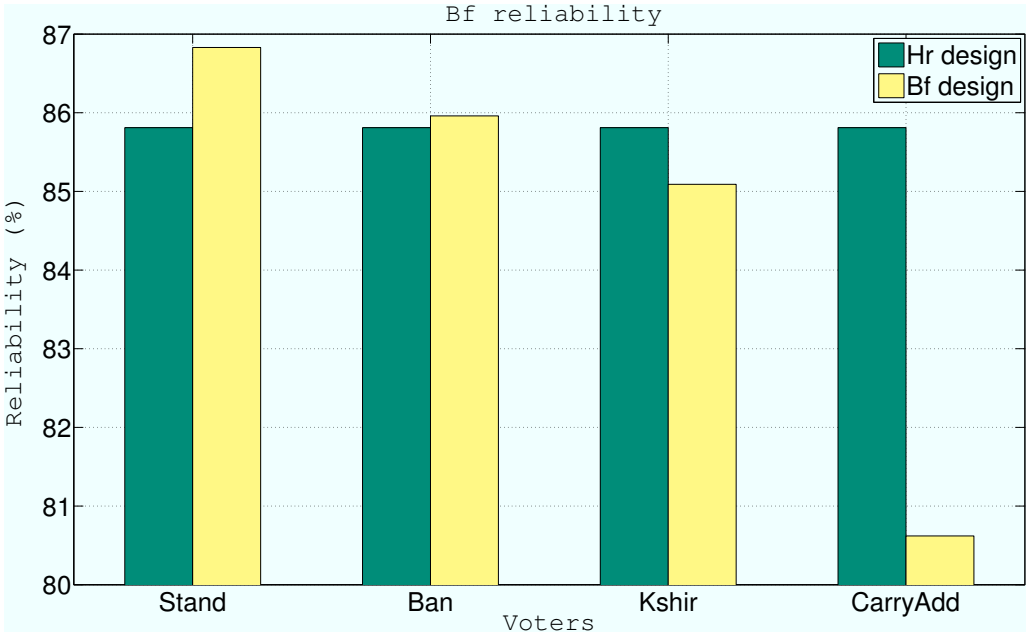
far as the Hr-3m is concerned, TMR is applied only on the most critical Mux2. The remaining Mux2s are not protected againts SETs, which explains the lowest masking gains achieved by Hr-3m. Thus, the terms $R_k, k \in [1, 4]$ in Equation 2.9 do not manage to compensate the low value of R_0, corresponding to a scenario where no faults occur. In fact, for any redundant design, the R_0 term is always lower than that of the non-redundant one. On the other hand, the voter in a TMR/NMR system is a critical component whose reliability determines the performances of the whole system. If the voter's probability of error outpasses a certain threshold, TMR/NMR degrades the initial reliability and becomes useless [1]. And the bigger the voter, the higher its probability of error. This can be checked in Figure 4.17 and Figure 4.18, where in each design the use of the least complex voters improves the reliability. For the Bf, the cascaded *Kshir* and the *CarryAdd* voters are so complex that they decrease the reliability of the whole design.

4.1.3.3 Synthesis Results and Quality Metric

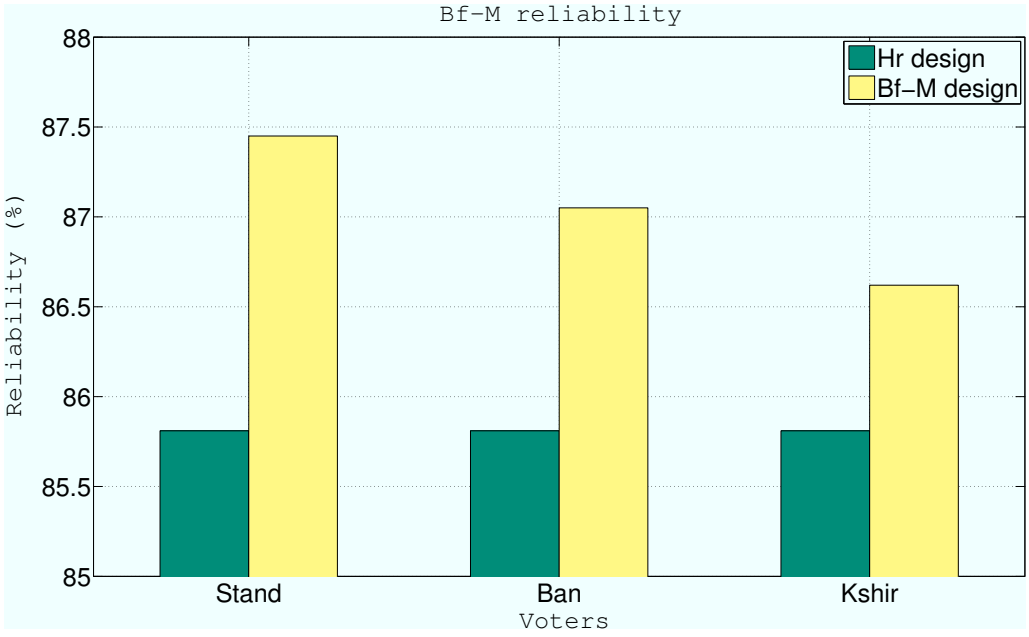
The retained hardened designs from the previous section are synthesized using Cadence RTL compiler under the same conditions as mentioned before. Then, a quality metric expressing the tradeoff between reliability gain and different overheads is defined to rank the LUT-4 architectures and select the best one.

Synthesis Results Table 4.10 gives synthesis results for the retained LUT-4 architectures, confronted with the Hr design. The last column is not part of the synthesis results. In fact, only the combinational part of the LUT-4 was synthesized.

We can see that, regardless of the voter, the Bf-M design has the lowest overheads in area, time, power and memory, and then comes the Bf followed by the TMR-Hr. Moreover, in all designs, the use of the standard TMR voter reduces the overheads, since the voter is synthesized in just one cell of the type HS65_LS_PAO2X4. However, the *Ban* and the *Kshir* TMR voters consist of respectively two and four cells of the CORE65LPSVT standard cell library.

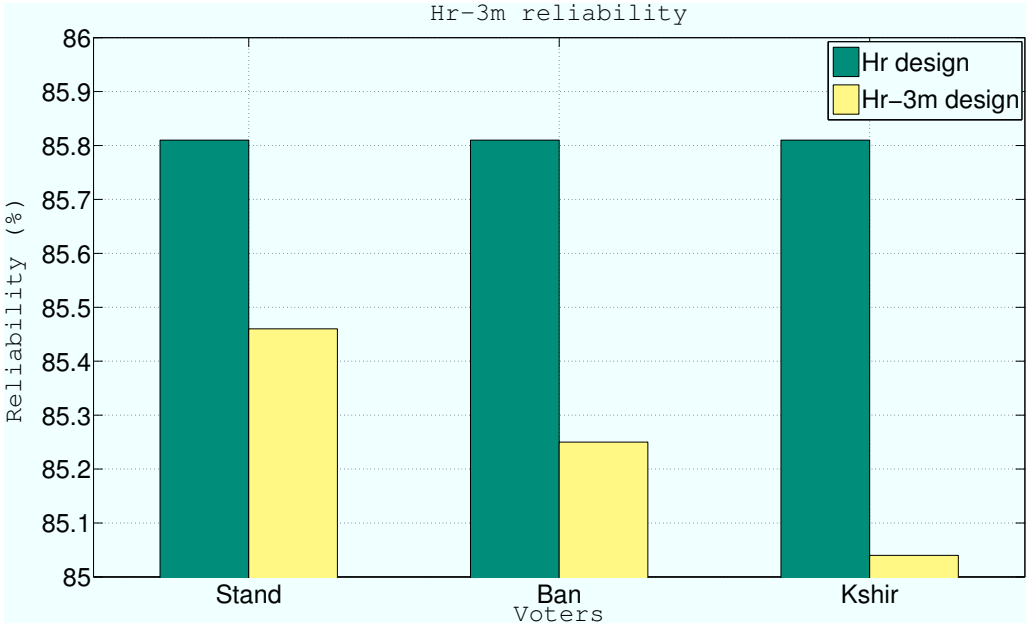


(a) Bf

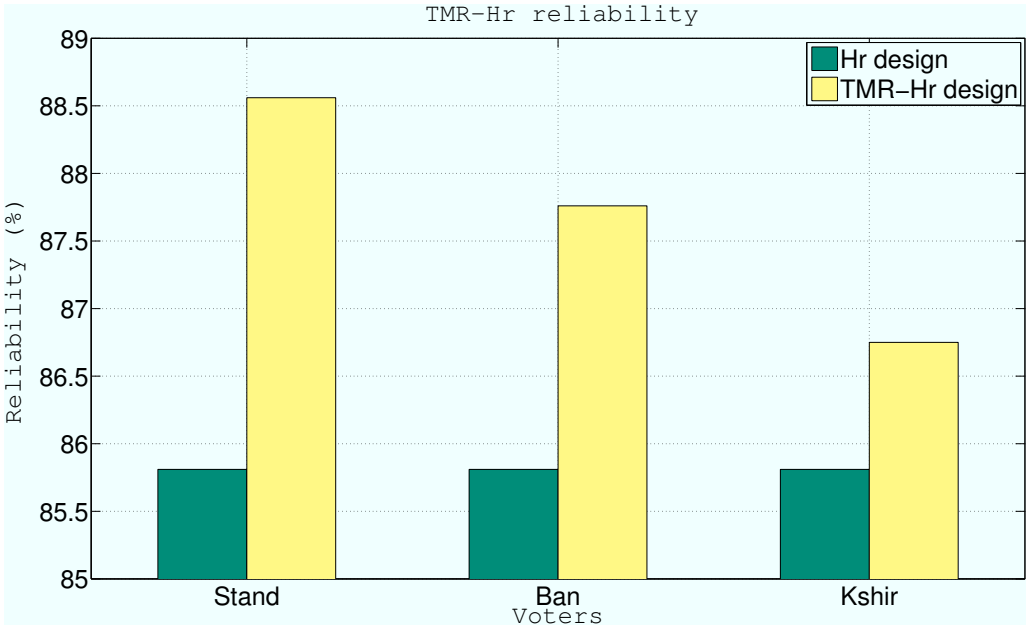


(b) Bf-M

Figure 4.17: Reliability of hardened CLB designs (1)



(a) Hr-3m



(b) TMR-Hr

Figure 4.18: Reliability of hardened CLB designs (2)

Table 4.11: Computation of the global metric Q.

LUT-4	Metric Q
Bf-M Stand	0.280
Bf-M Ban	0.220
Bf-M Kshir	0.180
Bf Stand	0.130
Bf Ban	0.085
TMR-Hr Stand	0.031
TMR-Hr Ban	0.024
TMR-Hr Kshir	0.021

Quality Metric The global design metric Q is expressed in Equation (4.10) where G_r, O_A, O_T, O_P and O_M are metrics as defined in the previous section. Values $w_i, i \in [1,5]$ are weights applied on the metrics as explained in the previous section.

In the presented example, the same importance is given to all parameters. Hence, all weights in Q are set equal. Computation results for the LUT-4 architectures are shown in Table 4.11, where designs are already ordered with respect to decreasing values of Q. Finally, since the global metric Q is the highest for Bf-M using the standard TMR voter, this design is selected as giving the best tradeoff between reliability gain and overheads.

$$Q = \frac{G_r^{w_1}}{O_A^{w_2} O_T^{w_3} O_P^{w_4} O_M^{w_5}} \tag{4.10}$$

4.2 Hardening the Switch Box

As long as the UMSB proved to be the most eligible component to be hardened in the switch box, a methodology is proposed in this section to enhance the inner defect tolerance of the UMSB while avoiding hardware redundancy. First, since the UMSB is a Mux6, we are building several Mux6 architectures by assembling different standard cells of the STM CORE65LPSVT library. Second, we use a Mentor Graphics Tessent CellModelGen tool [5] that wisely injects single defects into these architectures. This tool uses the extracted netlist of a design to model defects at transistor level. The way the tool works will be detailed in Section 5.2.1.2 of the next chapter dealing with hardening at transistor level.

Figure 4.19 represents different possible assemblings of standard cells from the STM CORE65LPSVT library for the Mux6. Indeed, the library contains Mux2, Mux3, Mux4 as sub-modules that can constitute the top level Mux6. A structural Verilog netlist was established for each architecture. Then, synthesis was carried out on Cadence RTL Compiler.

Table 4.12 summarizes simulation results obtained with the CellModelGen tool for all studied Mux6 architectures. From the figures in Table 4.12, we can conclude that the Gr43 has the lowest failure rates compared to its counterparts. Besides, it is the most compact design, solely composed of two standard cells: a multiplexer 4:1 and a multiplexer 3:1. There is no tradeoff to make in terms of reliability versus area overhead. Therefore, the

Gr43 is the best Mux6 architecture to be chosen among the ones studied. However, the Gr2 only constituted by multiplexers 2:1 is not the most optimal design since it is both area-costly and unreliable.

Table 4.12: Defect tolerance analysis results.

Mux6 architectures	**Gr422**	**Gr422(1)**	**Gr43**	**Gr332**	**Gr2**	**Gr2223**
Inserted bridges	390	377	406	401	371	382
Inserted opens	226	210	194	235	293	264
Failure rate (%)	12.11	12.19	10.43	12.37	12.57	12.17
Number of transistors	44	44	44	50	50	50

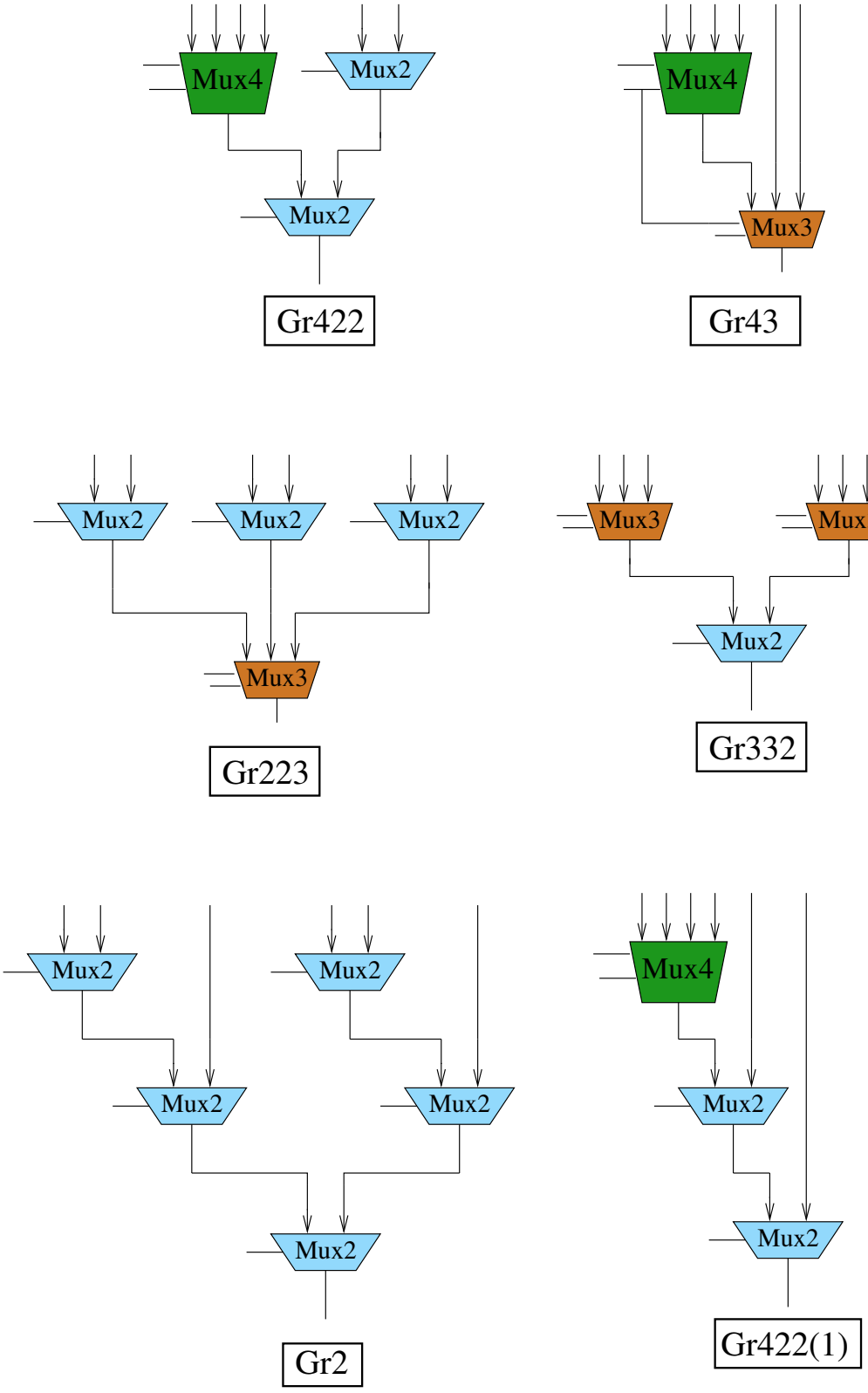


Figure 4.19: Mux6 studied architectures.

Chapter 5

Hardening at Transistor Level

This chapter aims at improving the robustness of the FPGA's basic blocks by proposing defect-tolerant Mux2 designs. Two different approaches are adopted. In the first approach, we examine single-ended solutions. We create a defect-tolerant Mux2, called *Z-Mux*, from Tristate cells of a 65nm industrial library. We compare the Z-Mux with the common Mux2 implemented with CMOS transmission gates, and with other Mux2 architectures. The proposed Z-Mux proved to be the most robust of all with respect to the modeled transistor defects. Besides, it consumes less area than other previous hardening methods. In the second approach, we explore two dual-rail Mux2 architectures. The first one is based on DCVS logic [113], and we propose a new second one called *Cross Logic*. We resort to the Mentor Graphics CellModelGen tool [5] to inject defects. This tool uses the extracted netlist of a design to model realistic physical defects. Finally, we compare these Mux2 architectures according to a design metric taking into account reliability gain and cost penalties.

5.1 Single-ended Designs

5.1.1 Mux2's Transistor Structures

In order to propose a defect-tolerant Mux2, we proceeded in two different ways.

First way, we explored four Mux2 design alternatives by changing the internal assembling of transistors, using the STM 65nm CMOS technology and CORE65LPSVT standard cell library. The transistor schematics of the three common Mux2 designs are represented in Figure 5.1 (transmission gates, nand gates and full custom). In all transistor schematics, signals $in0$ and $in1$ are the Mux2 inputs, $S0$ is the selection entry and out is the output signal. The proposed Z-Mux is the fourth Mux2 design, mainly composed of 2 Tristate cells. Figure 5.2 shows its transistor schematic.

Second way, we resorted to the TMR technique to improve the Mux2's defect tolerance. We applied TMR on the standard compact cell Trans-mux (cf. Figure 5.1(a)), and used two different voters: a standard TMR voter cell and a fault-tolerant TMR voter introduced in [1].

5.1.2 Defect Tolerance Analysis

The methodology and metrics used for defect tolerance analysis will be here explained.

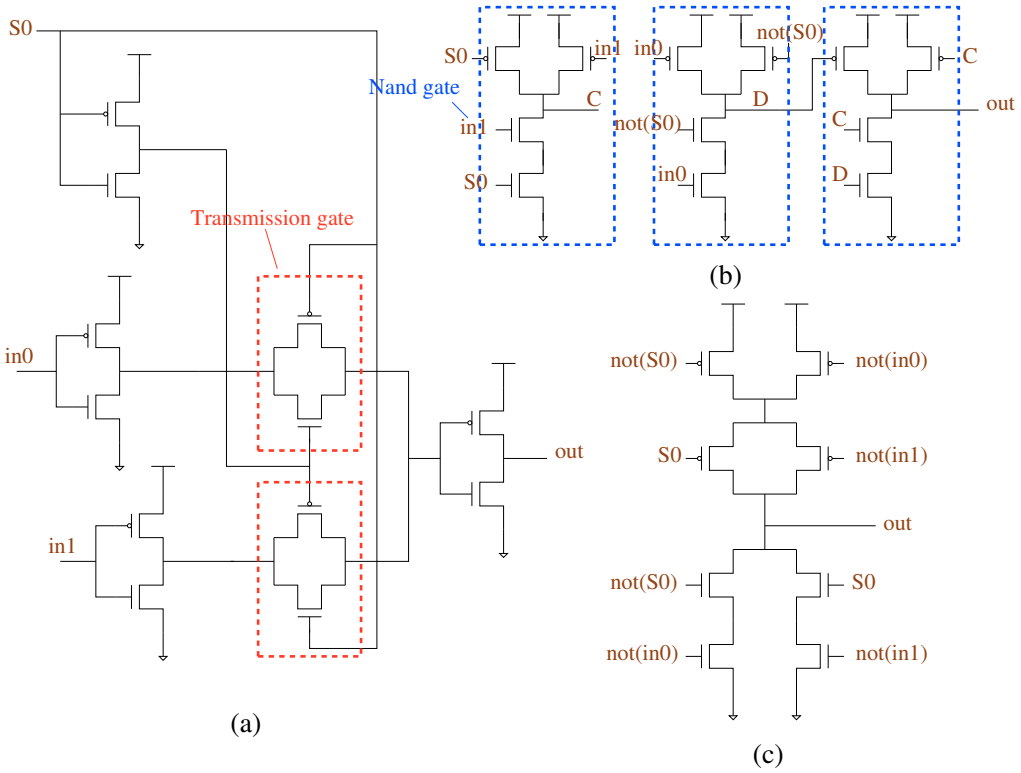


Figure 5.1: Simple Mux2 architectures: (a) Trans-mux, (b) Nand-mux, (c) FC-mux.

Defect modeling: We modeled single defects affecting the transistors. The most common defects are stuck-opens and stuck-shorts [98]. A stuck-open transistor is permanently OFF and is modeled by shorting the gate to the ground for the NMOS type, and to Vdd for the PMOS type. This is actually equivalent to removing the transistor from the netlist. A stuck-closed transistor is permanently ON and is modeled by shorting the drain and the source. Gate-drain and gate-source shorts were also considered.

Experimental setup: We generated random uniformly distributed binary sequences of length $Ns = 10^4$ for the three Mux2 inputs (including the selection bit) and we analytically computed the corresponding correct output sequence using Matlab. We stored all sequences in memory. Thereafter, to test the functionality of a given Mux2, we used the same generated input sequences in Piece Wise Linear (PWL) voltage sources so as to test different input transitions. The signals' period is $1.5ns$ and they were sampled at half of the period to make sure of getting a stable value. $Vdd = 1.2V$ is the supply voltage for the transistors of the cmos065 library used in the netlist. We performed a transient analysis on Cadence Spectre simulator, without injecting any defect in the Mux2 under test. Thus, we could verify the validity of our simulation model by comparing the obtained output sequence with the computed one stored in memory. Then, we performed the same analysis again while injecting a single transistor defect in the netlist. Finally, we compared the obtained output sequence to the correct one using metrics defined in Section 5.1.2.1. Figure 5.3 represents a block scheme for the experimental setup.

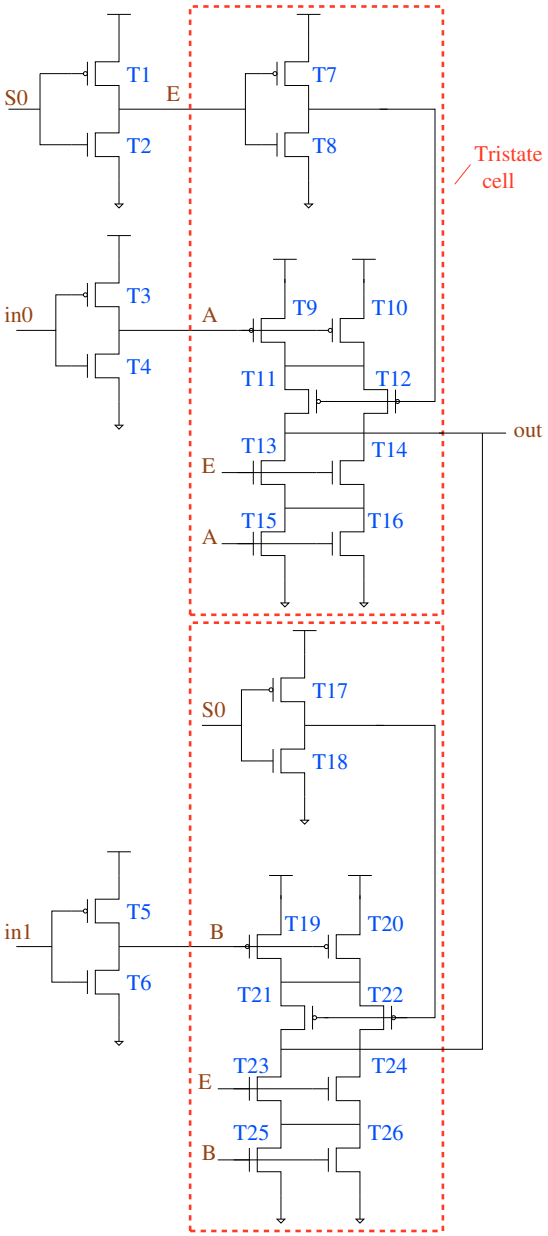


Figure 5.2: The Z-Mux architecture.

5.1.2.1 Defect Tolerance Metrics

Depending on whether we are analyzing the robustness of a given design to a particular defect or we are interested in the global defect tolerance of a design, we need different metrics.

Failure Profiles When injecting a single defect of type d in the transistor i, we assessed the failure rate and the Root-Mean-Square (RMS) error.

The failure rate is the binary error rate, estimated as in Equation (5.1) where $outBin$ and $outCorrBin$ correspond to logical values of the obtained and the correct output signals respectively. Considering an input sample j, we set $outBin(j) = 1$ if the obtained

output signal $out(j) \geq \frac{Vdd}{2}$, and $outBin(j) = 0$ otherwise. We checked the 'wisdom' of this decision by recomputing the failure rate after inserting a defect-free inverter at the circuit output (see Figure 5.3). $Ns = 10^4$ is the number of input samples. Indeed, if the Mux2 under test were defect-free, the obtained output signal would correspond to the correct output signal, in which case $outBin$ and $outCorrBin$ would be equal. However, because of a defective transistor, $outBin$ and $outCorrBin$ are likely to be different. Hence, the failure rate calculates the number of different bits between $outBin$ and $outCorrBin$ sequences and averages over the total number of samples.

$$F_d(i) = \frac{\sum_{j=1}^{Ns}(outBin(j) \oplus outCorrBin(j))}{Ns} \tag{5.1}$$

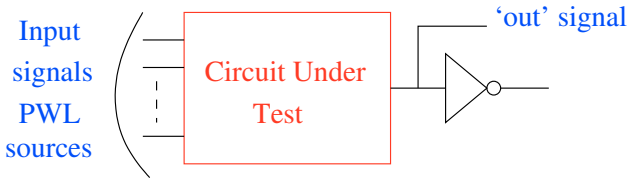


Figure 5.3: Experimental block scheme.

The RMS error is the distance between the obtained voltage output out and the correct one $outCorr$. It is expressed in Equation (5.2). The RMS error is a frequently used measure of the differences between values predicted by a model and the values actually observed.

$$RMS_d(i) = \sqrt{\frac{\sum_{j=1}^{Ns}(out(j) - outCorr(j))^2}{Ns}} \tag{5.2}$$

Overall Defect Tolerance The overall defect tolerance metrics are driven from the previously defined metrics when averaging over the number of transistors (Nt) and by all types of defects.

The masking rate is the masking ability of a given design, defined in Equation (5.3). It is deduced from the failure rate $F_d(i)$ which is a number between 0 and 1. For a given design, $\left(1 - \frac{\sum_{i=1}^{Nt} F_d(i)}{Nt}\right)$ is the masking rate of a defect d.

As a matter of fact, when a transistor i is affected by a defect d, the obtained output signal is not always wrong. For some input conditions, failures caused by a defect within the circuit appear in a non-sensitized path and thus can not propagate to the output. They are logically masked.

In addition to that, failures could be filtered out by electrical and/or temporal masking. A defective transistor might create a glitch that does not have enough duration or amplitude to reach the output of the circuit. Then, the glitch is said to be electrically masked. And regarding temporal masking, it occurs when the fault in the output signal does not arise at the sampling moment.

$$\tau = \frac{\sum_{d=1}^{4}\left(1 - \frac{\sum_{i=1}^{Nt} F_d(i)}{Nt}\right)}{4} \tag{5.3}$$

The conformity degree indicates to what extent the obtained voltage output is in accordance with the expected one. It is deduced from the RMS error metric $RMS_d(i)$ and is defined in Equation (5.4).

$$\kappa = \frac{\sum_{d=1}^{4} \left(1 - \frac{\sum_{i=1}^{Nt} RMS_d(i)}{Nt}\right)}{4} \tag{5.4}$$

5.1.3 Simulation and Comparison of Mux2 Designs

5.1.3.1 Simulation Results

Failure profiles for all simulated Mux2 designs were plotted. That of the Z-mux is shown in Figure 5.4. The transistors on the X-axis are numbered the same way as in Figure 5.2.

First of all, we notice that the shapes of the failure rate and the RMS error curves are in good agreement. Then, we can say in general that the most critical defect is the gate-drain short. When focusing on the core architecture, i.e. the Tristate cells only (transistors numbered from 7 to 26), we notice that the NMOS transistors (13, 14, 23, 24) controlled by the *enable* signal are the most sensitive ones to the gate-drain short. This is explained by the fact that the drain is directly connected to the output signal. On the other hand, the core architecture is generally immune to the stuck-open defect. Indeed, stuck-open tolerance is achieved with transistors connected in parallel as explained in [98]. So, we can verify that the core architecture duplicates all transistors in parallel, except those of the *enable* signal inverter. This explains the fact that the latter is the most vulnerable to the stuck-open defect.

Actually, such profiles provide us with the most sensitive parts of a design to a given defect. For instance, if we want to make an architecture more robust to stuck-closed defects, we can add transistors in series [98]. As far as the Z-mux is concerned, since it revealed more sensitivity to gate-drain shorts, we should find a hardening method against this kind of defect. We could resort to the MSO technique [100], which is completely immune to single gate shorts. Nonetheless, this technique induces too much area overhead, as it hardens one transistor by substituting it for a 12-transistor structure.

For all Mux2 designs, the failure rate and the RMS error are in accordance and give the same failure profile. Consequently, for further study of failure profiles, it is pointless considering both metrics. In fact, it would be more relevant to focus on the failure rate because it is easier to compute and it is used to get the masking rate which is involved in the reliability computation in Section 5.1.3.2. Furthermore, eventhough the RMS error gives more accurate measure of the difference between the obtained signal and the expected one, the failure rate is more appropriate if we are solely interested in the logical values of the output signal.

5.1.3.2 Comparison of Mux2 Designs

For all studied Mux2 designs, we computed the defect tolerance metrics and reported them in Table 5.1. We can see that the Z-Mux achieves the best performances in masking rate and conformity degree over all architectures. The defect tolerance gain realized by the Z-Mux, relatively to the standard Trans-mux, outpasses 10%. And, as far as the size is concerned, the Z-mux counts roughly twice as many transistors as the Trans-mux, but

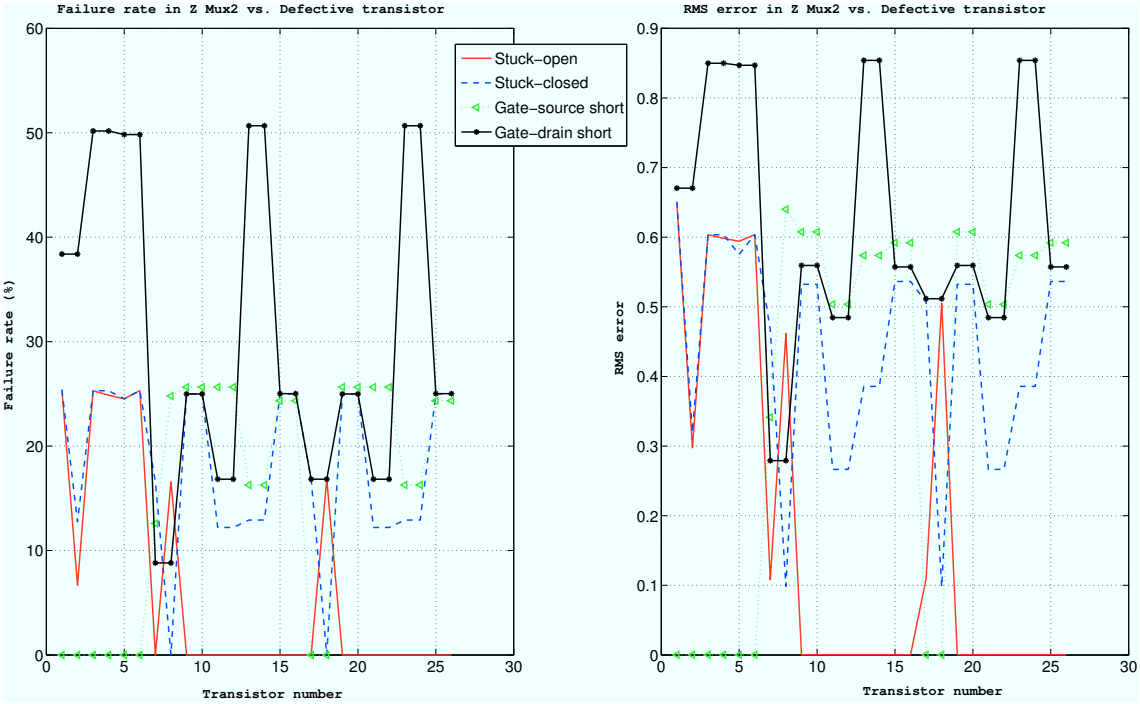


Figure 5.4: The Z-Mux failure profiles.

is approximately two times smaller than a TMR solution. That is to say that the Z-Mux is more area-efficient than the TMR.

Table 5.1: Defect tolerance metrics for the studied Mux2 architectures.

Mux2	**Trans**	**Nand**	**FC**	**stand. TMR**	**F.-T. TMR**	**Z**
$\tau(\%)$	71.75	72.49	75.32	74.12	77.72	82.30
$\kappa(\%)$	46.61	45.87	48.13	53.54	59.35	59.45
Nt	12	14	14	48	58	26

As a matter of fact, figures in Table 5.1 correspond to a 100% probability of defect occurrence, whereas in reality there is a certain probability that a defect takes place. We varied the probability of defect occurrence from 0 to 5% and plotted the reliability values that were computed on the basis of the masking rates. Figure 5.5 shows that the Z-mux is the most reliable architecture and the reliability gain increases along with the probability of defect occurrence.

5.2 Dual-rail Designs

In this section, we are presenting two defect-tolerant Mux2 designs based on differential logic. The first one uses DCVS logic. And we propose the second one that employs Cross Logic (CL).

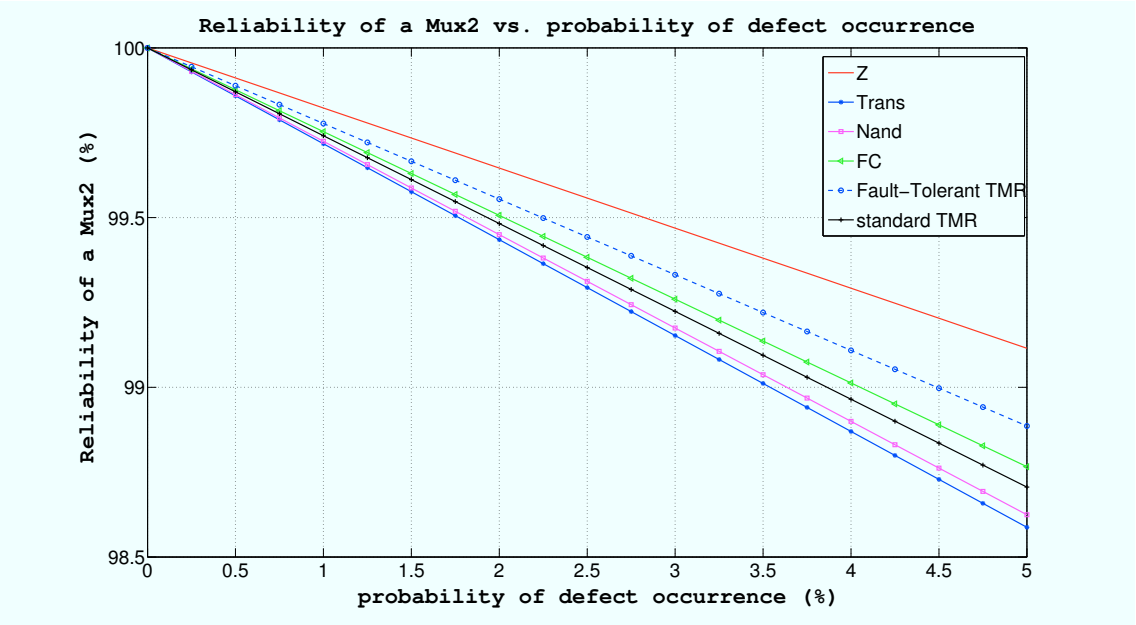


Figure 5.5: Reliability of the studied Mux2s.

5.2.1 The DCVS Mux2

5.2.1.1 Presentation

Figure 5.1(a) represents the transistor schematic of the single-ended standard Mux2 cell from the STM CORE65LPSVT standard cell library (MUX2X4). It is composed of two CMOS transmission gates and three inverters, where signals $in0$ and $in1$ are the Mux2 inputs, $s0$ is the selection entry and out is the output signal. In Section 5.1, the Z-Mux made up of two Tristate cells from the same library was introduced and proved to be more resilient to single transistor defects than the standard Mux2. However, it counts 26 transistors, which is more than the double area overhead. Hence, in this section, we thought of differential logic as a way to improve robustness and reduce area penalty at the same time.

In differential circuits, information is processed and transmitted in a redundant and complementary way, intrinsically offering an increased resistance to failures. Figure 5.6 depicts the proposed DCVS Mux2 built from 8 NMOS and 2 crossed PMOS transistors. The information and its complementary (out and $\overline{out}$ signals) are processed in two different paths. For each input combination, only one path drives the logic low value from the ground, and so polarizes the PMOS whose gate is connected to it, causing the complementary output to be at the logic high value. In fact, we noticed that the PMOS transistor sizing is an important parameter to drive correct signals and so to enhance the defect tolerance. This feature will be studied in the Section 5.2.1.2. Indeed, this architecture is expected to be more tolerant to open defects than bridges. For instance, if $sel = 1$, $A = 0$ and $B = 0$, then $out = 0$ thanks to three ON transistors. If one of the transistors controlled by $\overline{A}$ and sel is permanently OFF, the output will be correct all the same. So considering that particular input combination as an example, there is a probability of $\frac{2}{3}$ to get a correct output in the presence of an open defect. Simulations will prove later on that the proposed Mux2 architecture is actually more tolerant to opens than to bridges,

as theoretically expected.

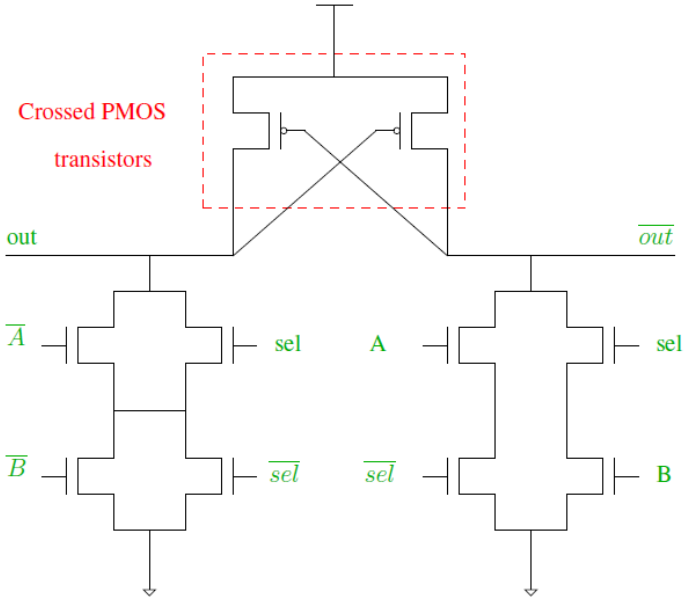


Figure 5.6: DCVS multiplexer.

5.2.1.2 Defect Tolerance Analysis

We first present the defect tolerance methodology that allowed us to assess the robustness of the proposed Mux2 architecture. Second, we analyze the impact of transistor sizing on the robustness gain. Then, we compare the proposed Mux2 architecture to the standard one and to two other Mux2 designs: the Z-Mux and the TMR solution.

– Defect emulation and robustness metric
With the Mentor Graphics Tessent CellModelGen tool [5], we could analyze the defect tolerance of each Mux2 architecture. The CellModelGen tool needs the extracted netlist of a design under study in order to inject single defects in it. Therefore, in a first step, we have to provide the tool with the layout extraction of each Mux2. Figure 5.7 represents the flow graph used to perform the proposed defect tolerance analysis. The GDSII (Graphic Data System) file for a standard Mux2 is already available in the library. We just added a filler cell to polarize transistor wells. But concerning the DCVS Mux2, we had to do a full-custom manual layout. After Design Rule Check and Layout Versus Schematic, the parasitic extraction generates a SPICE netlist that contains parasitic resistors and capacitors of all wires in the circuit. A portion of this extracted netlist including parasitic elements is depicted in Figure 5.8. As a matter of fact, the extracted capacitors are candidates for bridge defects. Resistors are candidates for open defects. The transistors are candidates for stuck-closed transistor also called transistor-on (Ton) and stuck-open transistor also called transistor-off (Toff) defects. The ports are candidates for port-open and port-bridge defects as shown in Figure 5.8 between the port A and the input $D0$ of a Mux2. Hence, this means that the CellModelGen tool considers realistic locations of bridges, opens, Ton/Toff, port-opens and port-bridges that are based on extracted capacitors, resistors, transistors and cell ports.

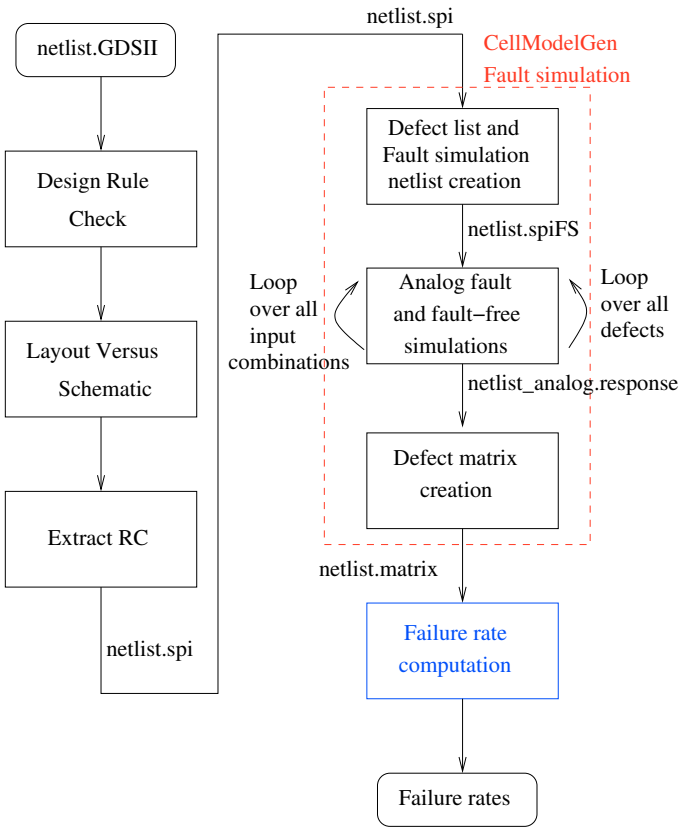


Figure 5.7: Defect tolerance analysis flow graph.

After establishing the defect list and creating a defective SPICE netlist (.spiFS) from the original defect-free one (.spi), the CellModelGen tool proceeds to the analog fault simulation. In this step, the effect of each potential defect for all possible input combinations is simulated. Actually, to emulate a faulty behaviour of a bridge, the tool inserts a resistor (for instance R=1 Ω) substituting a parasitic coupling between two interconnects. To emulate the faulty behaviour of an open defect, the resistor values are changed from their actual values to a high resistive value of, for example, $1\ G\Omega$. A Ton defect is emulated by inserting a resistor between the source and drain nets having a value of $1\ \Omega$. And concerning the Toff defect, a resistor of $1\ G\Omega$ is inserted into the source net and another one into the drain net.
On the other hand, each design is exhaustively simulated without defects in order to determine the golden voltage at the Mux2 output for every input combination. The simulations are analog transient analyses to determine the voltage at the Mux2 output. In each simulation run, the CellModelGen tool compares the output result of the analog fault simulation to the corresponding golden voltage and reports whether the defect was masked or not.
After the analog fault simulation, a defect matrix is created containing the important detection information for each defect and each stimulus. In the defect matrix, whenever a given defect is detected with a given input combination, it is signaled by a letter D. Equation 5.5 gives an example of a defect matrix where the rows correspond to the input combinations (stimuli) and the columns are the inserted

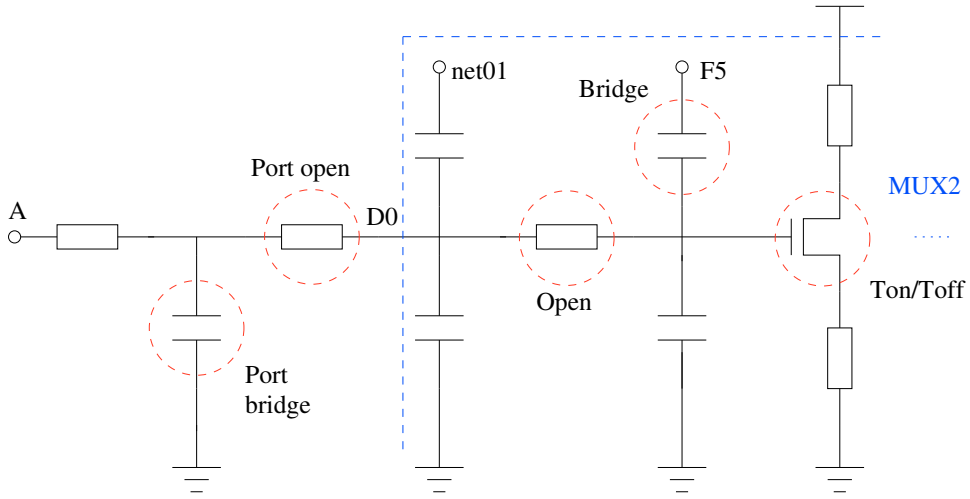


Figure 5.8: Portion of an extracted netlist.

defects.

$$Detected = \begin{bmatrix} D & - & D & ... & - & D & - \\ - & D & - & ... & - & - & - \\ . & & & & & & \\ . & & & & & & \\ . & & & & & & \\ - & - & D & ... & - & - & - \\ D & D & - & ... & - & - & - \end{bmatrix} \tag{5.5}$$

In fact, the CellModelGen tool is normally used for testability purposes to determine the test patterns, but we just needed the fault simulation part. Therefore, we adjusted the tool to satisfy our needs and we integrated the failure rate computation step (colored in blue in Figure 5.7) into the defect tolerance analysis flow.
In order to compute the failure rate F, we summed up all times when defects are detected and averaged over the total number of tests (that corresponds to the defect matrix size). The failure rate F is expressed in Equation 5.6 where n is the total number of inserted defects, d_j corresponds to a defect numbered j and m is the number of inputs. As the DCVS Mux2 has two outputs, its failure rate is expressed as in Equation 5.7 where the first two terms would be calculated as in Equation 5.6. The third term is the joint failure rate corresponding to the scenarios where both outputs are affected by a defect. We developed a Python routine which automatically computes such failure rate from the defect matrix. As a robustness metric, we use the masking rate that is defined as: $r = 1 - F$.

$$F = \frac{1}{2^m}\frac{1}{n}\sum_{i=0}^{2^m}\sum_{j=d_1}^{d_n} Detected(i,j) \tag{5.6}$$

$$F_{dcvs} = F_{out} + F_{\overline{out}} - F_{out \cap \overline{out}} \tag{5.7}$$

– Simulation results

As we did a full-custom design of the proposed Mux2 and we wanted it as compact as possible, we took in a first time the minimum channel width (equal to $w = 0.135\mu m$ in 65 nm technology) for all transistors. Table 5.2 points out that the DCVS Mux2 using $w = 0.135\mu m$ is globally more robust to single defects than the standard Mux2. And as it was expected, the DCVS robustness against open defects is even more significant (twice as important as the standard's). Nevertheless, the standard Mux2 is more tolerant to bridges. Then, hoping to better the DCVS robustness against bridges and to increase the overall robustness gain, we varied the PMOS channel width w from its minimal value to the double. Results are plotted in Figure 5.9 where both the failure rates of the DCVS and the standard architectures are confronted to see the robustness gain brought by the transistor sizing. Failure rates due to opens, to bridges and to both defects are represented. First of all, we can see that, for any PMOS channel width, the robustness gain in case of open defects is the highest. This remains in accordance with our expectations.

Table 5.2: DCVS Mux2 before and after transistor sizing.

Failure rate (%)	**Standard Mux2**	**DCVS using** $w = 0.135\mu m$	**DCVS using** $w = 0.17\mu m$
Global	26.66	25.58	23.52
Due to opens	23.43	12.5	12.5
Due to bridges	28.44	28.86	26.28

For the overall failure rate and the one due to bridges, the optimum is reached for the same PMOS channel width, corresponding to 25% of increase relatively to $0.135\mu m$, which equals $0.17\mu m$. When comparing the defect matrix corresponding to $w = 0.17\mu m$ to the ones corresponding to lower values of w, we noticed that there were more error maskings toward a correct logic one. Actually, when the channel width of PMOS transistors increases, their resistance decreases and so the output voltage increases. Hence, erroneous outputs that were supposed to be logic ones, were corrected thanks to the increase of the output voltage. Nonetheless, beyond $w = 0.17\mu m$, there were correct logic zeros that turned into incorrect logic ones, because of an excessive increase of w. Besides, it is worth noting that these results are independent of the layout, as the number and type of potential defects did not change. Finally, in terms of robustness, the optimum channel widths for the proposed Mux2 are $0.135\mu m$ and $0.17\mu m$ for the NMOS and the PMOS transistors, respectively. Corresponding failure rate figures are given in the last column of Table 5.2.

Furthermore, one might find these failure rates too high for a small component like the Mux2. Indeed, it is worth noting that the CellModelGen tool elaborates a pessimistic defect modeling. In reality, any parasitic coupling capacitor is replaced by a resistor value of 1Ω, regardless of the capacitance value. This unfairly increases the number of potential defects. And that is why the obtained failure rates are actually overestimated.

– DCVS Mux2 vs. other Mux2 architectures

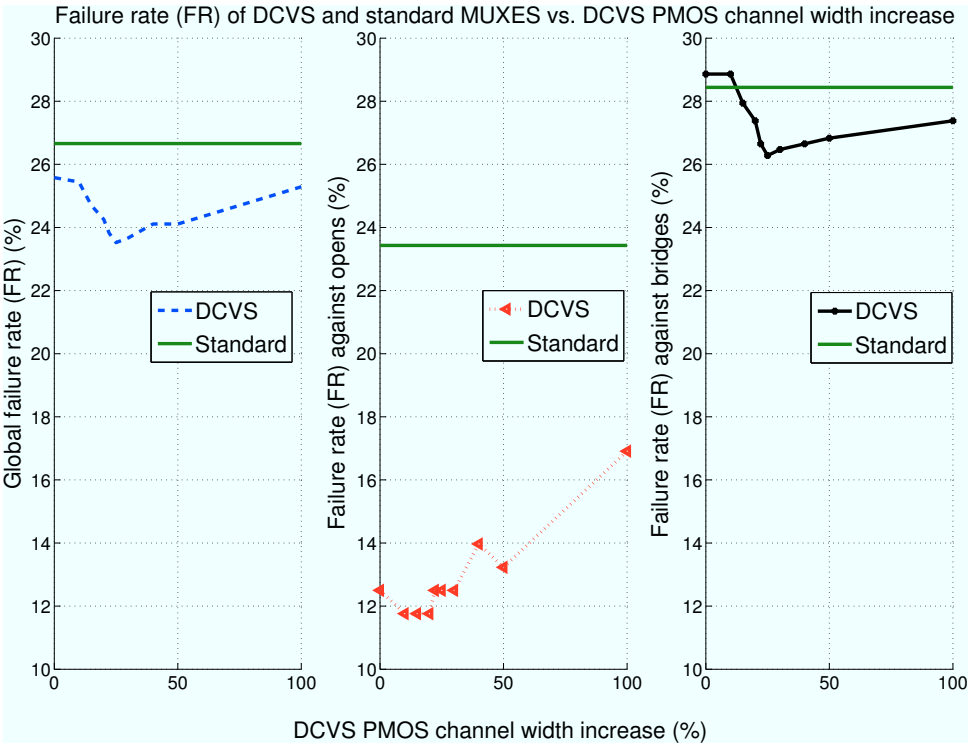


Figure 5.9: Impact of the PMOS channel width on the failure rate.

As long as we propose a defect-tolerant Mux2 causing little area overhead relatively to its standard counterpart, it is interesting to compare our solution to other existent hardened Mux2 architectures like the TMR and Z-Mux. Table 5.3 summarizes the defect tolerance analysis results for all studied Mux2 architectures. The first column reports the global masking rate r. As for r_{opens} and $r_{bridges}$, they are the masking rates due to opens and to bridges, respectively. One can see that the TMR solution achieves the highest masking rate gain. Then comes the Z-Mux architecture, followed by the DCVS design with a masking gain of 3%. However, if we observe masking rate gains due to open defects, the DCVS gain increases by more than three times, equalling 11%, approaches that of the TMR solution (13.5%) and surpasses the Z-Mux's by 3%. On the other hand, if we focus on the last column of the table, the proposed solution is the most compact one: about 6 and 7 times more compact than the Z-Mux and the TMR solution, respectively.

Table 5.3: Defect tolerance of the studied Mux2 architectures.

Mux2 design	r **(%)**	r_{opens} **(%)**	$r_{bridges}$ **(%)**	Area A (μm^2)
Standard	73.34	76.57	71.56	4.16
DCVS	76.48	87.50	73.72	6.02
Z-Mux	82.11	84.52	79.88	34.35
TMR	92.14	90.00	94.29	42.43

If we are interested in critical applications under area constraint, we need to express

the tradeoff between robustness gain and area penalty in the comparison of Mux2 architectures. Thus, we use a metric Q defined in Equation 5.8, where $Gr = \frac{r}{r_{stand}}$ is the robustness gain, $O_A = \frac{A}{A_{stand}}$ is the area overhead and (w_1, w_2) are weights chosen by the designer to decide whether the robustness or the area constraint is more important according to the design requirements. Obviously, the higher Q, the better the design.
We set $w_2 = 1$, and we consider different cases where robustness gain is a crucial design criterion $w_1 \geq w_2$. The weight on robutness gain is swept from 1 to 20 and the metric Q is assessed for all architectures to have them ranked. Figure 5.10 shows the comparison results. As we can see, even if robustness gain is 10 times more important than area penalty, the DCVS solution is ranked first ahead of TMR and Z-Mux.

$$Q = \frac{Gr^{w_1}}{O_A^{w_2}} \tag{5.8}$$

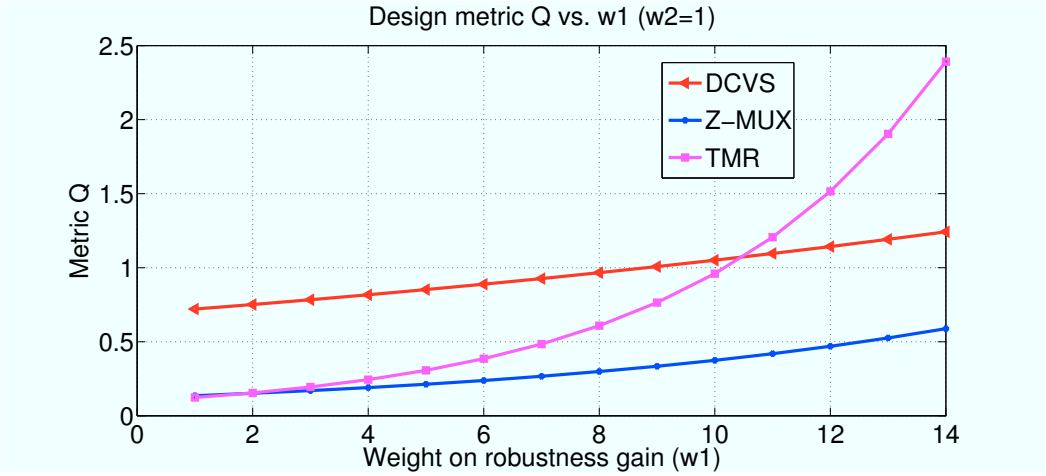


Figure 5.10: Design metric Q varying w1, while w2=1.

5.2.1.3 Improving the LUT Robustness with DCVS Mux2

The goal is to evaluate the robustness gain that could be realized by implementing a LUT (of any size N) with DCVS Mux2s instead of standard Mux2s. For instance, Figure 5.11 depicts a LUT-3 implementation with DCVS Mux2s, where $x_i, i \in [0, 7]$ would be the configuration bits stored in memory (the FPGA bitstream) and $s_i, i \in [0, 2]$ the LUT inputs, if this LUT corresponded to an FPGA logic block.

As long as the LUT is only made up of Mux2s, we could theoretically compute its failure rate from the failure rates of its elementary cells. To get the failure rate of a single Mux2 in a LUT-N from its failure rate FR_{mux} computed in Section 5.2.1.2, one should divide by the total number of multiplexers that is equal to $2^N - 1$. Then, assuming a uniform binary distribution of the primary inputs (selection bits), we reasoned stage by stage and tried to see, for each Mux2, when an error occurring in it would propagate to the output. We ended up with the Equation 5.9 expressing the failure rate of a LUT-N. This formula was verified for the LUT-2 with CellModelGen tool.

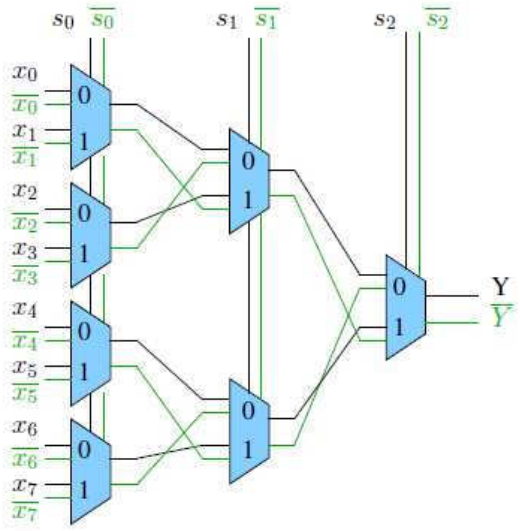


Figure 5.11: A LUT-3 in DCVS logic.

$$FR_{lut-N} = \frac{N \times FR_{mux}}{2^N - 1} \tag{5.9}$$

For different values of N, we computed the global LUT failure rate as well as the LUT failure rate due to open defects, using all standard Mux2s and all DCVS Mux2s. Results are shown in Figure 5.12. All curves are decreasing which is quite normal since we are increasing the LUT size while keeping the single defect model. The robustness gain is also decreasing with the LUT size. We can see that the LUT implemented with all DCVS Mux2s has always a lower failure rate than the one using all standard Mux2s. And we can observe that the robustness gain against open defects is more noticeable than the global robustness gain. For the LUT-4, the global robustness gain is roughly 1% and the one due to opens is approximately 3%. Although these gain values may seem very little, they become considerable for an FPGA composed of thousands of LUTs.

On the other hand, it should be noted that if one chooses the DCVS LUT design, there is an extra memory overhead apart from the one caused by the multiplexers. Indeed, an SRAM LUT-N in DCVS logic has $(2^N + N)$ SRAM cells more than the conventional LUT implemented in transmission gates.

5.2.2 Cross Logic: a New Logic for Defect-Tolerant Circuits

We propose to construct defect-tolerant cells using standard CMOS cells. The main idea is to obtain the output from two different paths such that if one path is defective, the second one remains fault-free and corrects the value given by the faulty path. To achieve our target, we have invented a new logic which we called Cross Logic (CL), inspired from DCVS logic, although their implementations are different.

One CL cell consists of the CMOS corresponding cell, its dual and two crossed CMOS inverters. We call dual cell the one generating the complementary outputs from the complementary inputs. Figure 5.13 depicts the general schematic for a CL cell where Q and $\overline{Q}$ correspond to the output signal and its complementary respectively. OP^* is the dual

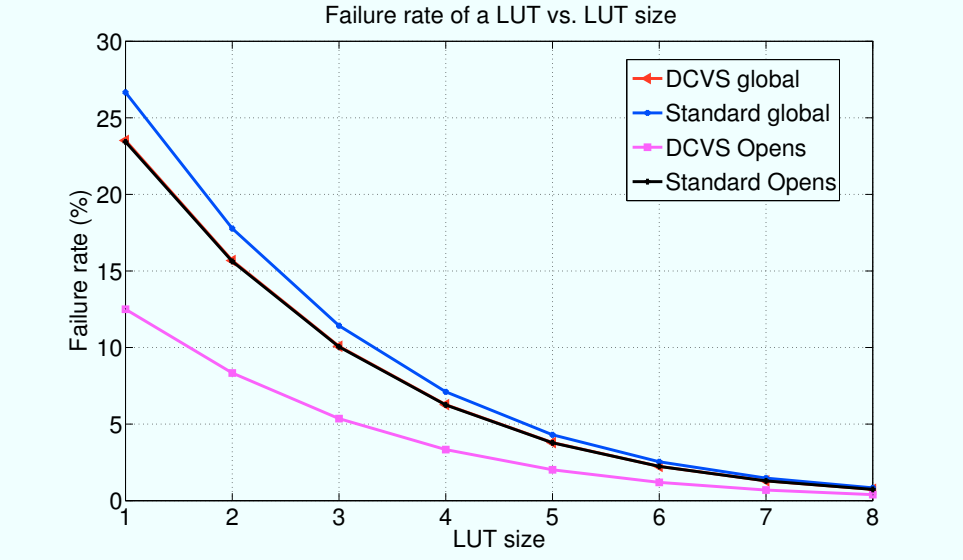


Figure 5.12: LUT failure rates versus LUT size.

of operator OP. For instance, the NAND gate has the NOR gate as dual. The Mux2 function is the dual of itself, i.e. if $Q = MUX2(sel, A, B)$ then $\overline{Q} = MUX2(\overline{sel}, \overline{B}, \overline{A})$. For simplicity reasons, we will explain CL with the NAND example.

Figure 5.14 represents the CL NAND cell. The CMOS NAND gate on the right side generates the output signal Q. The dual CMOS NOR gate on the left side generates the complementary signal of the output $\overline{Q}$. The pair of crossed inverters allows the generation of the signal output from its complementary and vice versa. Now, suppose a stuck-open defect affecting transistor T_1 which means that T_1 is permanently OFF. If the inputs $A = 1$ and $B = 0$, the output of the CMOS NAND Q would normally be in the state high impedence. However, thanks to the crossed inverters, the correct value of Q is imposed by $\overline{Q}$. Of course, the same reasoning could be applied wherever the stuck-open defect occurs. On the other hand, if the value of Q or $\overline{Q}$ is affected by any kind of bridge defects (port bridge, T_{on}...), it can be corrected by its complementary. However, one would wonder why the faulty value would not distort the corrected one. This actually depends on different parameters such as transistor sizing of the crossed inverters or their operating regions.

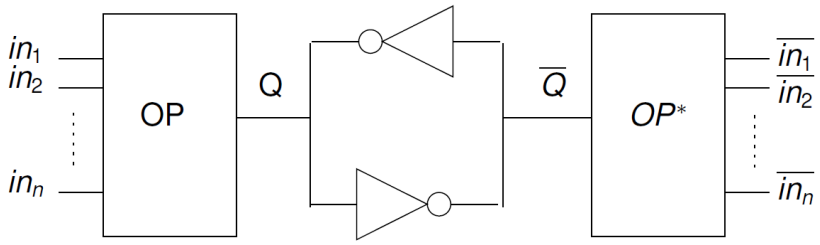


Figure 5.13: CL general schematic.

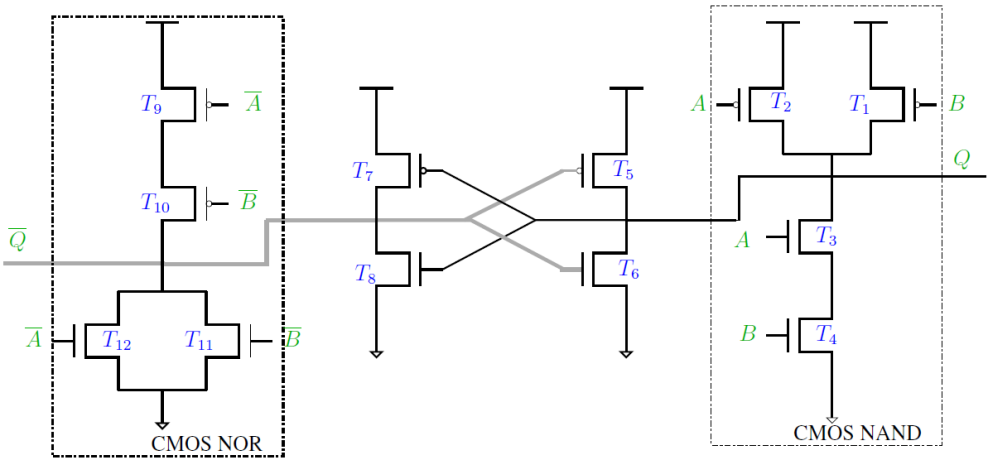


Figure 5.14: The CL NAND gate.

5.3 Comparative Study of the Multiplexer Designs

In this section, we are comparing the most interesting Mux2 architectures, namely the Z-Mux, the DCVS, the CL and the TMR solution, in terms of reliability and cost penalties (not only area but also delay and power consumption). Then, we are selecting the most convenient one according to a design metric we define.

5.3.1 Defect Tolerance Analysis

We perform the same defect tolerance analysis as described in Section 5.2.1.2 with CellModelGen tool. Table 5.4 summarizes the analysis results for the aforementioned Mux2 architectures. The first column reports the global masking rate r. As previously mentioned, r_{opens} and $r_{bridges}$, they are the masking rates due to opens and to bridges, respectively. The TMR solution still achieves the highest masking rate gain. Then comes the CL, followed by the Z-Mux architecture and the DCVS design. If we focus on masking rate gains due to open defects, the DCVS gain surpasses the Z-Mux's but still is outdistanced by the CL and TMR designs. It is worth noting that, unlike all other designs, the TMR Mux2 is more robust to bridges than to open defects.

Table 5.4: Defect tolerance of the studied Mux2 architectures.

Mux2 design	r **(%)**	r_{opens} **(%)**	$r_{bridges}$ **(%)**
Standard	73.34	76.57	71.56
DCVS	76.48	87.50	73.72
CL	83.43	91.50	77.08
Z-Mux	82.11	84.52	79.88
TMR	92.14	90.00	94.29

Actually, a design tradeoff is established among hardening techniques and the design vulnerability to a particular type of defects. For instance, the CL hardening technique turns out to be the most effective against open defects, so if the design is more affected

by opens than bridges, the CL is to be used instead of TMR.

5.3.2 Comparison Metric

We will show the penalties engendered by the hardened architectures in terms of area, power and delay. Then, we will define a design metric expressing the tradeoff between robustness gain and all overheads, and allowing to rank the Mux2 designs.

Table 5.5 shows that the DCVS is the most compact architecture, being 1.5 times bigger than the standard Mux2, while the Z-Mux, the CL and the TMR are approximately 8, 9 and 10 times bigger than their standard counterpart, respectively. Nevertheless, the DCVS has the worst features in terms of delay and average energy. It is 11 times slower and 4 times more power consuming than the standard. The most efficient design in both speed and energy is the Z-Mux architecture.

Table 5.5: Area, delay and energy of the studied Mux2 architectures.

Mux2	**Area (μm^2)**	**Delay (ps)**	**Average energy ($\mu W/GHz$)**
Standard	4.16	74.32	9.5467
DCVS	6.02	808	41.036
CL	36.78	97.78	23.786
Z-Mux	34.35	78.68	19.236
TMR	42.43	108.59	20.834

Since we are interested in critical applications under area, delay and power constraints, we need to express the tradeoff between robustness gain and all stated penalties in the comparison of Mux2 architectures. Thus, we use a metric Q defined in Equation 5.10, where the following metrics are defined relatively to the standard:

- $Gr = \frac{r}{r_{stand}}$ is the global robustness gain;
- $O_A = \frac{A}{A_{stand}}$ is the area overhead;
- $O_D = \frac{D}{D_{stand}}$ is the delay overhead;
- $O_E = \frac{E}{E_{stand}}$ is the energy overhead.

$$Q = \frac{G_r^{w_1}}{O_A^{w_2}.O_D^{w_3}.O_E^{w_4}} \tag{5.10}$$

It should be noted that $w_i, i \in [1, 4]$ are weights chosen by the designer to decide whether the robustness or the constraints are more important according to the application requirements. Obviously, the higher Q, the better the design. Table 5.6 gives an example where all parameters are of the same importance. So, all weights are set equal. We can see that, in this case, the Z-Mux architecture has the highest Q. Therefore, it is the most suitable one to be selected as a hardened design considering area, delay and energy constraints.

And as previously said, depending on the application requirements, one can think of another metric focusing only on opens or bridges.

Table 5.6: Comparison metric for the Mux2s.

Mux2	Q
DCVS	0.015
CL	0.039
Z-Mux	0.063
TMR	0.038

5.3.3 Improving the LUT Robustness

The global robustness gain that could be achieved, under the single defect model, in a LUT (of any size N) using hardened Mux2s instead of standard ones will be highlighted. Then, the most appropriate choice of Mux2 architecture will be discussed.

For different values of N, we computed the global LUT failure rate using only standard Mux2s and only hardened Mux2s. Results are shown in Figure 5.15. As expected, we can notice that the LUT implemented with only TMR Mux2s has always the lowest failure rate. So, one is tempted to select the TMR architecture for the Mux2. Yet, we should keep in mind that a single TMR Mux2 has one of the worst values of the design metric Q. It is 2.5% less than that of the Z-Mux. For the LUT-4 example, the global robustness gain using the Z-Mux architecture is roughly 2.5%. This gain can be doubled using the TMR Mux2 design. Nevertheless, the consequent penalties in terms of area, delay and power consumption will be certainly heavy, making the robustness gain worthless. Therefore, the most convenient Mux2 architecture to be used is the Z-Mux.

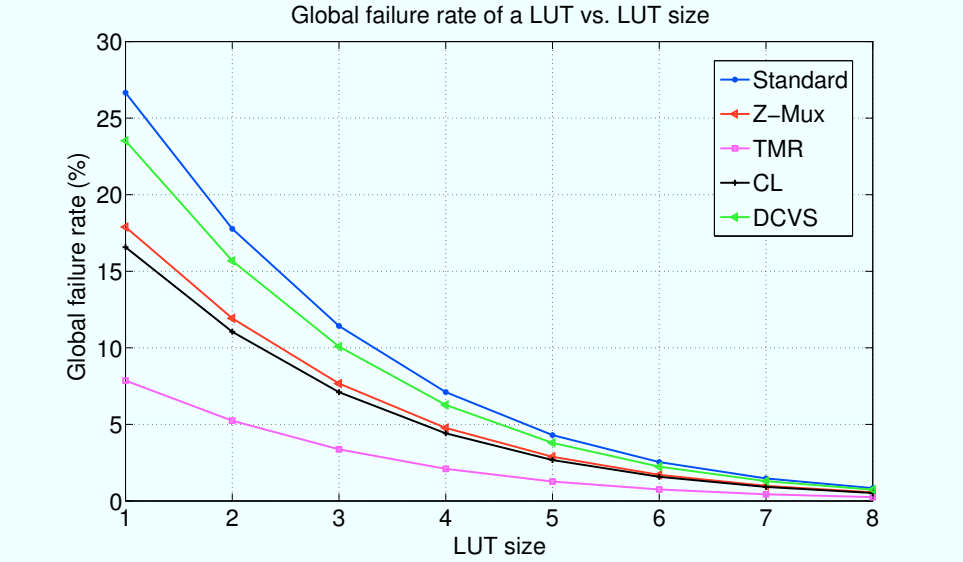


Figure 5.15: LUT global failure rates versus LUT size.

To put it in a nutshell, according to the application required level of reliability and the allowed extra costs, the designer has to define a proper metric that will let him select the most convenient design.

Conclusion

The present thesis dealt with fault tolerance in FPGAs that are surely affected by technology downscaling. Indeed, the tremendous development of CMOS technology has enabled an increasing integration density according to Moore's Law. However, this evolution trend is being slowed due to economic and physical limits. 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. Moreover, transients faults caused by charged particles also constitute a major threat for reliability. FPGAs are widely used in many critical applications like space and military, where reliability is a crucial design criterion. And modern FPGA architectures are mainly organized in clusters of configurable logic resources connected together by depopulated interconnect. So, this thesis aimed at hardening the basic blocks in a mesh of clusters FPGA, against permanent and transient faults.

The main contributions concerned the mesh of clusters FPGA's inherent structure. After selecting in the literature an appropriate method to evaluate the reliability of the basic blocks, the most vulnerable components were identified to be hardened. When hardening at multiplexer level, the first contribution was a new fault-tolerant architecture for the logic block, called *Butterfly*. It has the advantage of being generic for any number of CLB inputs, and it is more compact than a TMR solution. Then, design metrics taking into account both reliability gain and cost penalties were proposed to compare the Butterfly architecture to other hardened CLBs. These metrics help the designer to choose the most suitable architecture for a given application.

Another hardening solution was also proposed. It is a *redundanceless* scheme based on a "smart" synthesis that consists in seeking the most reliable design in a given founder library, instead of directly using a redundant solution. This scheme is actually part of a methodology using Mentor Graphics CellModelGen tool as a defect injector. The methodology can be easily integrated in a classical design flow.

When hardening at a finer level, different possible transistor assemblings were explored. First, a single-ended multiplexer design made up of two tristate cells from the STM CORE65LPSVT library proved to be more resilient to common transistor defects than other designs from the same library. Second, a dual-rail multiplexer design based on DCVS logic was an interesting alternative in terms of reliability and area. Another dual-rail multiplexer design was implemented according to a new logic named *Cross Logic*. Finally, to compare and rank all designs, we resorted to similar metrics to those used for CLBs.

Future avenues of research would be to upgrade the Mentor Graphics CellModelGen tool, that is limited to injecting single defects, in order to study the impact of multiple manufacturing defects on the multiplexer designs. Then, a comparison between single defect and multiple defect analyses would be worth discussing. Furthermore, we would

like to investigate the integration of the proposed hardening solutions, fabricate an FPGA prototype and perform experimental tests with real radiations attacking the circuit.

The results obtained in this thesis can be extended and used to manufacture the targeted mesh of clusters FPGA in the *Robust FPGA* ANR project. Most of the works covered in this thesis were published in appropriate conferences and journal, and those publications are listed in Appendix A. Some of them were the product of collaborations with other researchers within the same laboratory or with other partners of the *Robust FPGA* ANR project. We believe and hope that those publications have helped the evolution of this field and solved some of its challenges.

Appendix A

List of Publications

- *Analyzing and Alleviating the Impact of Errors on an SRAM-based FPGA Cluster,* Arwa Ben Dhia, Lirida Naviner and Philippe Matherat, in International On-Line Testing Symposium (IOLTS), 18th IEEE, 2012.

- *A New Fault-Tolerant Architecture for CLBs in SRAM-based FPGAs,* Arwa Ben Dhia, Lirida Naviner and Philippe Matherat, in Electronics, Circuits and Systems (ICECS), 19th IEEE International Conference on, 2012.

- *Tolérance aux défauts dans les FPGAs,* Arwa Ben Dhia et Lirida Naviner, Journées Nationales du Réseau Doctoral en Micro-nanoélectronique (JNRDM), 2012.

- *Automatic Selective Hardening Against Soft Errors: A Cost-based and Regularity-aware Approach,* Samuel Pagliarini, Arwa Ben Dhia, Lirida Naviner and Jean-François Naviner, in Electronics, Circuits and Systems (ICECS), 19th IEEE International Conference on, 2012.

- *Comparison of Fault-Tolerant Fabless CLBs in SRAM-based FPGAs,* Arwa Ben Dhia, Lirida Naviner and Philippe Matherat, in Latin American Test Workshop (LATW), 14th IEEE, 2013.

- *A defect-tolerant area-efficient multiplexer for basic blocks in SRAM-based FPGAs,* Arwa Ben Dhia, Samuel Pagliarini, Lirida Naviner, Habib Mehrez and Philippe Matherat, in Reliability of Electron Devices, Failure Physics and Analysis (ESREF), 24th European Symposium on, 2013.

- *A defect-tolerant area-efficient multiplexer for basic blocks in SRAM-based FPGAs,* Arwa Ben Dhia, Samuel Pagliarini, Lirida Naviner, Habib Mehrez and Philippe Matherat, in Microelectronics Reliability Journal, 2013.

- *Evaluating CLB Designs under Multiple SETs in SRAM-based FPGAs,* Arwa Ben Dhia, Lirida Naviner and Philippe Matherat, in Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 16th IEEE International Symposium on, 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, and Zied Marrakchi, in Field-Programmable Technology (FPT), 12th IEEE International Conference on, 2013.

- *Nouvelle architecture de BLE tolérante aux fautes dans les FPGAs SRAM,* Arwa Ben Dhia et Lirida Naviner, Journées Nationales du Réseau Doctoral en Micro-nanoélectronique (JNRDM), 2013.

- *SNaP: a Novel Hybrid Method for Circuit Reliability Assessment Under Multiple Faults,*

Samuel Pagliarini, Arwa Ben Dhia, Lirida Naviner and Jean-François Naviner, in Reliability of Electron Devices, Failure Physics and Analysis (ESREF), 24th European Symposium on, 2013.

- *SNaP: a Novel Hybrid Method for Circuit Reliability Assessment Under Multiple Faults,* Samuel Pagliarini, Arwa Ben Dhia, Lirida Naviner and Jean-François Naviner, in Microelectronics Reliability Journal, 2013.

- *Improving the Robustness of a Switch Box in a Mesh of Clusters FPGA,* Arwa Ben Dhia, Mariem Slimani and Lirida Naviner, in Latin American Test Workshop (LATW), 15th IEEE, 2014.

- *A Defect-Tolerant Multiplexer Using Differential Logic for FPGAs,* Arwa Ben Dhia, Mariem Slimani and Lirida Naviner, in Mixed Design of Integrated Circuits and Systems (MIXDES), 21st IEEE International Conference on, 2014.

- *Comparative study of defect-tolerant multiplexers for FPGAs,* Arwa Ben Dhia, Mariem Slimani and Lirida Naviner, in International On-Line Testing Symposium (IOLTS), 20th IEEE, 2014.

- *Cross Logic: a New Approach for Defect-Tolerant Circuits,* Mariem Slimani, Arwa Ben Dhia and Lirida Naviner, in IC Design and Technology (ICICDT), 11th IEEE International Conference on, 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, and Zied Marrakchi, in Computer Society Annual Symposium on VLSI (ISVLSI), 13th IEEE, 2014.

- *A dual-rail compact defect-tolerant multiplexer,* Arwa Ben Dhia, Mariem Slimani, Hao Cai and Lirida Naviner, Accepted for publication in Microelectronics Reliability Journal, January 2015.

- *Designing a Robust Mesh of Clusters FPGA,* Arwa Ben Dhia and Lirida Naviner, Accepted for publication by LAP LAMBERT Academic Publishing, January 2015.

Bibliography

[1] T. Ban and L. de Barros Naviner, "A simple fault-tolerant digital voter circuit in TMR nanoarchitectures," in *NEWCAS Conference (NEWCAS), 2010 8th IEEE International*, June 2010, pp. 269–272.

[2] R. Kshirsagar and R. Patrikar, "Design of a novel fault-tolerant voter circuit for TMR implementation to improve reliability in digital circuits," *Microelectronics Reliability*, vol. 49, no. 12, pp. 1573 – 1577, 2009.

[3] L. Naviner, J.-F. Naviner, G. dos Santos Jr., E. Marques, and N. P. Jr., "FIFA: A fault-injection-fault-analysis-based tool for reliability assessment at RTL level," *Microelectronics Reliability*, vol. 51, no. 9-11, pp. 1459–1463, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0026271411002162

[4] M. de Vasconcelos, D. Franco, L. de B. Naviner, and J.-F. Naviner, "Reliability analysis of combinational circuits based on a probabilistic binomial model," in *Circuits and Systems and TAISA Conference, 2008. NEWCAS-TAISA 2008. 2008 Joint 6th International IEEE Northeast Workshop on*, June 2008, pp. 310 –313.

[5] M. Graphics. Mentor Graphics Tessent CellModelGen Tool. [Online]. Available: http://www.mentor.com/products/silicon-yield/products/testkompress/

[6] J. Neumann, Ed., *Probabilistic logics and the synthesis of reliable organisms from unreliable components*. Automata Studies, January 1956.

[7] E. Moore and C. Shannon, "Reliable circuits using less reliable relays. I-II," *the Franklin Institute*, vol. 262, no. 43, pp. 191–208, 281–297, 1956.

[8] ITRS. International Technology Roadmap for Semiconductors. [Online]. Available: http://www.itrs.net/Links/2013ITRS/Summary2013.htm

[9] L. Di Cioccio, P. Gueguen, R. Taibi, T. Signamarcheix, L. Bally, L. Vandroux, M. Zussy, S. Verrun, J. Dechamp, P. Leduc, M. Assous, D. Bouchu, F. De Crecy, L.-L. Chapelon, and L. Clavelier, "An innovative die to wafer 3D integration scheme: Die to wafer oxide or copper direct bonding with planarised oxide inter-die filling," in *3D System Integration, 2009. 3DIC 2009. IEEE International Conference on*, Sept 2009, pp. 1–4.

[10] H. Goel and D. Dance, "Yield enhancement challenges for 90 nm and beyond," in *Advanced Semiconductor Manufacturing Conference and Workshop, 2003 IEEEI/SEMI*, 2003, pp. 262–265.

[11] S. N. Pagliarini, L. A. d. B. Naviner, and J.-F. Naviner, "Selective hardening methodology for combinational logic," in *Test Workshop (LATW), 2012 13th Latin American*, april 2012, pp. 1 –6.

[12] M. Stanisavljevic, A. Schmid, and Y. Leblebici, "Optimization of the averaging reliability technique using low redundancy factors for nanoscale technologies," *Nanotechnology, IEEE Transactions on*, vol. 8, no. 3, pp. 379–390, May 2009.

[13] G. G. D. S. Júnior, "Conception Robuste de Circuits Numériques à Technologies Nanométriques," Ph.D. dissertation, École Nationale Supérieure des Télécommunications, 2012.

[14] X. Lu, X. Su, J. Zeng, and H. Wang, "A single FPGA embedded framework for secondary user in cognitive network," in *Communication Technology (ICCT), 2010 12th IEEE International Conference on*, Nov 2010, pp. 881–884.

[15] J. Manikandan, M. Jayaraman, and M. Jayachandran, "Design of an FPGA-based electronic flow regulator (EFR) for spacecraft propulsion system," *Advances in Space Research*, vol. 47, no. 3, 2011.

[16] S. Habermann, R. Kothe, and H. Vierhaus, "Built-in self repair by reconfiguration of FPGAs," in *12th IEEE International On-Line Testing Symposium, IOLTS 2006.*, 2006.

[17] M. Berg, "Fault tolerance implementation within SRAM based FPGA designs based upon the increased level of single event upset susceptibility," in *12th IEEE International On-Line Testing Symposium, IOLTS 2006.*, 2006, p. 3 pp.

[18] C. Bolchini, C. Sandionigi, L. Fossati, and D. Codinachs, "A reliable fault classifier for dependable systems on SRAM-based FPGAs," in *On-Line Testing Symposium (IOLTS), 2011 IEEE 17th International*, July 2011, pp. 92 –97.

[19] F. Lahrach, A. Abdaoui, A. Doumar, and E. Chatelet, "A novel SRAM-based FPGA architecture for defect and fault tolerance of configurable logic blocks," in *Design and Diagnostics of Electronic Circuits and Systems (DDECS), 2010 IEEE 13th International Symposium on*, April 2010, pp. 305 –308.

[20] A. Avizienis, J.-C. Laprie, B. Randell, and Vytautas. (2000) Fundamental Concepts of Dependability. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.24.1400

[21] J. Laprie, A. Avizienis, and H. Kopetz, Eds., *Dependability: Basic Concepts and Terminology*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1992.

[22] C. Stapper, F. Armstrong, and K. Saji, "Integrated Circuit Yield Statistics," *Proceedings of the IEEE*, vol. 71, no. 4, pp. 453–470, 1983.

[23] C. Constantinescu, "Trends and challenges in VLSI circuit reliability," *Micro, IEEE*, vol. 23, no. 4, pp. 14–19, July 2003.

[24] I. Koren and Z. Koren, "Defect Tolerance in VLSI Circuits: Techniques and Yield Analysis," *Proceedings of the IEEE*, vol. 86, no. 9, pp. 1819–1838, 1998.

[25] B. Benware, C. Schuermyer, M. Sharma, and T. Herrmann, "Determining a Failure Root Cause Distribution From a Population of Layout-Aware Scan Diagnosis Results," *Design Test of Computers, IEEE*, vol. 29, no. 1, pp. 8–18, 2012.

[26] H. Walker and S. Director, "VLASIC: A Catastrophic Fault Yield Simulator for Integrated Circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 5, no. 4, pp. 541–556, October 1986.

[27] C. VinodChandra and S. Ramasamy, "Test pattern generation for benchmark circuits using lfsr," in *Computing, Communications and Networking Technologies (ICCCNT),2013 Fourth International Conference on*, July 2013, pp. 1–6.

[28] L. Lingappan, S. Ravi, and N. Jha, "Satisfiability-based test generation for nonseparable rtl controller-datapath circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, no. 3, pp. 544–557, March 2006.

[29] S. Reddy, I. Pomeranz, and S. Kajihara, "Compact test sets for high defect coverage," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 16, no. 8, pp. 923–930, Aug 1997.

[30] H. Hao and E. McCluskey, ""Resistive Shorts" within CMOS Gates," in *Test Conference, 1991, Proceedings., International*, Oct 1991, pp. 292–.

[31] S. Zhong, "Fault modelling and accelerated simulation of integrated circuits manufacturing defects under process variation," Ph.D. dissertation, University of Southampton, 2013.

[32] S. Kundu, S. T. Zachariah, S. Sengupta, and R. Galivanche, "Test Challenges in Nanometer Technologies," *J. Electron. Test.*, vol. 17, no. 3-4, pp. 209–218, Jun. 2001. [Online]. Available: http://dx.doi.org/10.1023/A:1012203009875

[33] C. Constantinescu, "Intermittent faults in VLSI circuits," in *Proceedings of IEEE Workshop on System Effects of Logic Soft Errors*, 2006.

[34] R. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *Device and Materials Reliability, IEEE Transactions on*, vol. 5, no. 3, pp. 305–316, Sept 2005.

[35] P. E. Dodd and L. W. Massengill, "Basic Mechanisms and Modeling of Single-event Upset in Digital Microelectronics," *Nuclear Science, IEEE Transactions on*, vol. 50, no. 3, pp. 583–602, Jun. 2003. [Online]. Available: http://dx.doi.org/10.1109/TNS.2003.813129

[36] N. Seifert, "Radiation-induced Soft Errors: A Chip-level Modeling Perspective," *Found. Trends Electron. Des. Autom.*, vol. 4, pp. 99–221, Feb. 2010. [Online]. Available: http://dx.doi.org/10.1561/1000000018

[37] C. Slayman, "Cache and Memory Error Detection, Correction, and Reduction Techniques for Terrestrial Servers and Workstations," *Device and Materials Reliability, IEEE Transactions on*, vol. 5, no. 3, sept. 2005.

[38] F. Monteiro, S. Piestrak, H. Jaber, and A. Dandache, "Fault-secure interface between fault-tolerant RAM and transmission channel using systematic cyclic codes," in *13th IEEE International On-Line Testing Symposium, IOLTS 2007.*, July 2007, pp. 199 –200.

[39] M. Haghi and J. Draper, "The 90 nm Double-DICE storage element to reduce Single-Event upsets," in *Circuits and Systems, 2009. MWSCAS '09. 52nd IEEE International Midwest Symposium on*, Aug. 2009, pp. 463 –466.

[40] E. Hwang, S. Jeon, R. Negi, B. Kumar, and M. Cheng, "Scrubbing with partial side information for radiation-tolerant memory," in *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, 2010, pp. 1941–1945.

[41] B. Narasimham, B. Bhuva, R. Schrimpf, L. Massengill, M. Gadlage, O. Amusan, W. Holman, A. Witulski, W. Robinson, J. Black, J. Benedetto, and P. Eaton, "Characterization of Digital Single Event Transient Pulse-Widths in 130-nm and 90-nm CMOS Technologies," *Nuclear Science, IEEE Transactions on*, vol. 54, no. 6, dec. 2007.

[42] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, 2002, pp. 389 – 398.

[43] D. T. Franco, "Fiabilité du Signal des Circuits Logiques Combinatoires sous Fautes Simultanées Multiples," Ph.D. dissertation, École Nationale Supérieure des Télécommunications, 2009.

[44] N. Seifert, P. Slankard, M. Kirsch, B. Narasimham, V. Zia, C. Brookreson, A. Vo, S. Mitra, B. Gill, and J. Maiz, "Radiation-induced soft error rates of advanced cmos bulk devices," in *Reliability Physics Symposium Proceedings, 2006. 44th Annual., IEEE International*, 2006, pp. 217–225.

[45] N. George and J. Lach, "Characterization of logical masking and error propagation in combinational circuits and effects on system vulnerability," in *Dependable Systems Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, June 2011, pp. 323 –334.

[46] Altera. FPGA - field-programmable gate array. [Online]. Available: http://www.altera.com/devices/fpga/stratix-fpgas/stratix-ii/stratix-ii/st2-index.jsp

[47] Xilinx, "Benchmark designs for the quartus university interface program (QUIP). Technical report," pp. 41–43, March 2005.

[48] V. Betz, J. Rose, and A. Marquardt, in *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, January 1999.

[49] Z. Marrakchi, H. Mrabet, and H. Mehrez, "Optimized local interconnect for cluster-based mesh FPGA architecture," in *Microelectronics, 2008. ICM 2008. International Conference on*, dec. 2008, pp. 15 –18.

[50] E. Amouri, Z. Marrakchi, and H. Mehrez, "Differential pair routing to balance dual signals of WDDL designs in cluster-based Mesh FPGA," in *Reconfigurable*

Communication-centric Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on, June 2011, pp. 1–4.

[51] I. Kuon, R. Tessier, and J. Rose, "FPGA Architecture: Survey and Challenges." Foundations and Trends in Electronic Design Automation,, 2008.

[52] D. Lewis, E. Ahmed, G. Baeckler, V. Betz, M. Bourgeault, D. Cashman, D. Galloway, M. Hutton, C. Lane, A. Lee, P. Leventis, S. Marquardt, C. McClintock, K. Padalia, B. Pedersen, G. Powell, B. Ratchev, S. Reddy, J. Schleicher, K. Stevens, R. Yuan, R. Cliff, and J. Rose, "The Stratix II logic and routing architecture," in *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, ser. FPGA '05. New York, NY, USA: ACM, 2005, pp. 14–20.

[53] W. Feng and S. Kaptanoglu, "Designing efficient input interconnect blocks for LUT clusters using counting and entropy," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 1, no. 1, pp. 6:1–6:28, Mar. 2008.

[54] Xilinx. FPGA - field-programmable gate array. [Online]. Available: http://www.xilinx.com/products/silicon-devices/fpga/index.htm

[55] Z. Marrakchi, H. Mrabet, and H. Mehrez, "Programmable Gate Array, Switch Box and Logic Unit for such an Array," Patent 7 795 911, September, 2010.

[56] M. Hutton, K. Adibsamii, and A. Leaver, "Timing-driven placement for hierarchical programmable logic devices," in *Proceedings of the 2001 ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays*, ser. FPGA '01. New York, NY, USA: ACM, 2001, pp. 3–11. [Online]. Available: http://doi.acm.org/10.1145/360276.360286

[57] Y.-T. Lai and P.-T. Wang, "Hierarchical interconnection structures for field programmable gate arrays," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 5, no. 2, pp. 186–196, June 1997.

[58] A. Aggarwal and D. 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*, Oct 1994, pp. 475–478.

[59] M. Zied, M. Hayder, A. Emna, and M. Habib, "Efficient Tree Topology for FPGA Interconnect Network," in *Proceedings of the 18th ACM Great Lakes Symposium on VLSI*, ser. GLSVLSI '08. New York, NY, USA: ACM, 2008, pp. 321–326. [Online]. Available: http://doi.acm.org/10.1145/1366110.1366186

[60] Z. Marrakchi, H. Mrabet, and H. Mehrez, "Evaluation of hierarchical FPGA partitioning methodologies based on architecture Rent parameter," in *Research in Microelectronics and Electronics 2006, Ph. D.*, 2006.

[61] J. Pistorius and M. Hutton, *Placement Rent Exponent Calculation Methods, Temporal Behaviour and FPGA architecture Evaluation*. SLIP, April 2003.

[62] M.-C. Hsueh, T. Tsai, and R. Iyer, "Fault Injection Techniques and Tools," *Computer*, vol. 30, no. 4, pp. 75 –82, Apr 1997.

[63] D. Bhaduri and S. Shukla, "NANOLAB-a tool for evaluating reliability of defect-tolerant nanoarchitectures," *Nanotechnology, IEEE Transactions on*, vol. 4, no. 4, pp. 381–394, July 2005.

[64] A. C. Marquez, A. S. Heguedas, and B. Iung, "Monte carlo-based assessment of system availability. a case study for cogeneration plants," *Reliability Engineering & System Safety*, vol. 88, no. 3, pp. 273 – 289, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0951832004001838

[65] J. Banks and J. Carson, *Discrete Event System Simulation*, Prentice Hall, Englewood Cliffs, NJ ed., Feb 2008.

[66] S.-A. Hwang, J.-H. Hong, and C.-W. Wu, "Sequential Circuit Fault Simulation Using Logic Emulation," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 17, no. 8, pp. 724–736, 1998.

[67] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Reorda, and M. Violante, "Exploiting Circuit Emulation for Fast Hardness Evaluation," *Nuclear Science, IEEE Transactions on*, vol. 48, no. 6, pp. 2210–2216, 2001.

[68] F. Faure, P. Peronnard, and R. Velazco, "Thesic+: A flexible system for SEE testing," in *Proceedings of RADECS*, 2002.

[69] F. Faure, R. Velazco, and P. Peronnard, "Single-event-upset-like fault injection: a comprehensive framework," *Nuclear Science, IEEE Transactions on*, vol. 52, no. 6, pp. 2205–2209, Dec 2005.

[70] G. Foucard, P. Peronnard, and R. Velazco, "Reliability limits of tmr implemented in a sram-based fpga: Heavy ion measures vs. fault injection predictions," in *Test Workshop (LATW), 2010 11th Latin American*, March 2010, pp. 1–5.

[71] P. Sedcole, B. Blodget, T. Becker, J. Anderson, and P. Lysaght, "Modular Dynamic Reconfiguration in Virtex FPGAs," *Computers and Digital Techniques, IEEE Proceedings -*, vol. 153, no. 3, pp. 157–164, 2006.

[72] L. Antoni, R. Leveugle, and B. Feher, "Using Run-time Reconfiguration for Fault Injection in Hardware Prototypes," in *Defect and Fault Tolerance in VLSI Systems, 2002. DFT 2002. Proceedings. 17th IEEE International Symposium on*, 2002, pp. 245–253.

[73] M. Aguirre, V. Baena, J. Tombs, and M. Violante, "A New Approach to Estimate the Effect of Single Event Transients in Complex Circuits," *Nuclear Science, IEEE Transactions on*, vol. 54, no. 4, pp. 1018–1024, 2007.

[74] M. Jeitler, M. Delvai, and S. Reichor, "Fuse - a hardware accelerated hdl fault injection tool," in *Programmable Logic, 2009. SPL. 5th Southern Conference on*, April 2009, pp. 89–94.

[75] C. Lopez-Ongil, M. Garcia-Valderas, M. Portela-Garcia, and L. Entrena, "Autonomous Fault Emulation: A New FPGA-Based Acceleration System for Hardness Evaluation," *Nuclear Science, IEEE Transactions on*, vol. 54, no. 1, pp. 252–261, 2007.

[76] L. Entrena, M. Garcia-Valderas, R. Fernandez-Cardenal, A. Lindoso, M. Portela, and C. Lopez-Ongil, "Soft Error Sensitivity Evaluation of Microprocessors by Multilevel Emulation-Based Fault Injection," *Computers, IEEE Transactions on*, vol. 61, no. 3, pp. 313–322, 2012.

[77] S. Krishnaswamy, G. Viamontes, I. Markov, and J. Hayes, "Accurate reliability evaluation and enhancement via probabilistic transfer matrices," in *Design, Automation and Test in Europe, 2005. Proceedings*, March 2005, pp. 282 – 287 Vol. 1.

[78] D. Franco, M. Vasconcelos, L. Naviner, and J.-F. Naviner, "Reliability of logic circuits under multiple simultaneous faults," in *Circuits and Systems, 2008. MWSCAS 2008. 51st Midwest Symposium on*, Aug. 2008, pp. 265 –268.

[79] D. T. Franco, M. C. Vasconcelos, L. Naviner, and J.-F. Naviner, "Signal Probability for Reliability Evaluation of Logic Circuits," *Microelectronics Reliability*, vol. 48, no. 8-9, pp. 1586 – 1591, 2008.

[80] R. Ogus, in *The probability of a correct output from a combinational circuit*, May 1975.

[81] S. Dokouzgiannis and J. Kontoleon, "Exact reliability analysis of combinational logic circuits," Dec. 1988.

[82] A. Bogliolo, M. Damiani, P. Olivo, and B. Ricco, "Reliability evaluation of combinational logic circuits by symbolic simulation," in *VLSI Test Symposium*, May 1995.

[83] J. Chen, J. Mundy, Y. Bai, S. m. C. Chan, P. Petrica, and R. I. Bahar, "A probabilistic approach to nano-computing," in *in IEEE non-silicon computer workshop*. IEEE Press, 2003.

[84] T. Rejimon and S. Bhanja, "Scalable probabilistic computing models using Bayesian networks," in *Circuits and Systems, 2005. 48th Midwest Symposium on*, Aug. 2005, pp. 712–715 Vol. 1.

[85] A. Abdollahi, "Probabilistic decision diagrams for exact probabilistic analysis," in *Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design*, ser. ICCAD '07. Piscataway, NJ, USA: IEEE Press, 2007, pp. 266–272. [Online]. Available: http://dl.acm.org/citation.cfm?id=1326073.1326128

[86] M. Choudhury and K. Mohanram, "Reliability Analysis of Logic Circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, no. 3, pp. 392–405, March 2009.

[87] S. Luckenbill, J.-Y. Lee, Y. Hu, R. Majumdar, and L. He, "RALF: Reliability Analysis for Logic Faults – An exact algorithm and its applications," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, March 2010, pp. 783–788.

[88] A. Darwiche, "Decomposable negation normal form," *J. ACM*, vol. 48, no. 4, pp. 608–647, Jul. 2001. [Online]. Available: http://doi.acm.org/10.1145/502090.502091

[89] J. T. Flaquer, J. Daveau, L. Naviner, and P. Roche, "Fast reliability analysis of combinatorial logic circuits using conditional probabilities," *Microelectronics Reliability*, vol. 50, no. 9-11, pp. 1215–1218, 2010, 21st European Symposium on the Reliability of Electron Devices, Failure Physics and Analysis. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0026271410003318

[90] K. Mohammadi, H. Jahanirad, and P. Attarsharghi, "Fast Reliability Analysis Method for Sequential Logic Circuits," in *Systems Engineering (ICSEng), 2011 21st International Conference on*, Aug 2011, pp. 352–356.

[91] J. Han, H. Chen, E. Boykin, and J. A. B. Fortes, "Reliability evaluation of logic circuits using probabilistic gate models," *Microelectronics Reliability*, vol. 51, pp. 468–476, 2011.

[92] S. Pagliarini, A. B. Dhia, L. de B. Naviner, and J.-F. Naviner, "SNaP: a Novel Hybrid Method for Circuit Reliability Assessment Under Multiple Faults," *Microelectronics Reliability*, 2013.

[93] J. von Neumann, "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," *Automata Studies*, vol. 34, pp. 43–99, 1956. [Online]. Available: http://www.cs.caltech.edu/courses/cs191/paperscs191/VonNeumann56.pdf

[94] D. Blough and G. Sullivan, "A comparison of voting strategies for fault-tolerant distributed systems," in *Reliable Distributed Systems, 1990. Proceedings., Ninth Symposium on*, Oct. 1990, pp. 136 –145.

[95] G. Dos Santos Jr., L. Naviner, B. Cousin, G. Deleuze, and L. Crétinon, "Procédé de durcissement logique par partitionnement d un circuit électronique," Patent 1 261 439, Novembre, 2012.

[96] A. El-Maleh and F. Oughali, "Enhancing Reliability of Combinational Circuits against Soft Errors by Using a Generalized Modular Redundancy Scheme," in *Electronic System Design (ISED), 2013 International Symposium on*, Dec. 2013, pp. 62–66.

[97] W. Pierce, "Failure-tolerant computer design," *Academic Press*, 1965.

[98] L. Anghel and M. Nicolaidis, "Defects tolerant logic gates for unreliable future nanotechnologies," in *Proceedings of the 9th international work conference on Artificial neural networks*, ser. IWANN'07. Berlin, Heidelberg: Springer-Verlag, 2007.

[99] A. Djupdal, "Evolving Static Hardware Redundancy for Defect Tolerant FPGAs," Ph.D. dissertation, Norwegian University of Science and Technology, 2008.

[100] A. Djupdal and P. Haddow, "Defect Tolerance Inspired by Artificial Evolution," in *Symposium on VLSI, 2008. ISVLSI '08. IEEE Computer Society Annual*, April 2008, pp. 28 –33.

[101] M. Takechi and T. Tokunaga, "Evolving hardware with genetic learning: A first step towards building a darwin machine," in *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*. The MIT Press, 1993, pp. 417–424.

[102] A. Eiben and J. Smith, "Introduction to Evolutionary Computing." Springer, 2003.

[103] F. Corno, M. Reorda, and G. Squillero, "A new evolutionary algorithm inspired by the selfish gene theory," in *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, May 1998, pp. 575–580.

[104] A. Thompson, "Evolving fault tolerant systems," in *Genetic Algorithms in Engineering Systems: Innovations and Applications, 1995. GALESIA. First International Conference on (Conf. Publ. No. 414)*, Sep 1995, pp. 524–529.

[105] A. Tyrrell, G. Hollingworth, and S. Smith, "Evolutionary strategies and intrinsic fault tolerance," in *Evolvable Hardware, 2001. Proceedings. The Third NASA/DoD Workshop on*, 2001, pp. 98–106.

[106] R. Canham and A. Tyrrell, "Evolved fault tolerance in evolvable hardware," in *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, vol. 2, 2002, pp. 1267–1271.

[107] A. Jackson, R. Canham, and A. Tyrrell, "Robot fault-tolerance using an embryonic array," in *Evolvable Hardware, 2003. Proceedings. NASA/DoD Conference on*, July 2003, pp. 91–100.

[108] E. Stefatos and T. Arslan, "An efficient fault-tolerant vlsi architecture using parallel evolvable hardware technology," in *Evolvable Hardware, 2004. Proceedings. 2004 NASA/DoD Conference on*, June 2004, pp. 97–103.

[109] D. Keymeulen, R. Zebulum, Y. Jin, and A. Stoica, "Fault-tolerant evolvable hardware using field-programmable transistor arrays," *Reliability, IEEE Transactions on*, vol. 49, no. 3, pp. 305–316, Sep. 2000.

[110] J. Lohn, G. Larchev, and R. DeMara, "A Genetic Representation for Evolutionary Fault Recovery in Virtex FPGAs," in *Proceedings Of The Fifth International Conference On Evolvable Systems (ICESâĂŹ03)*, 2003, pp. 47–56.

[111] A. Djupdal and P. C. Haddow, "The Route to a Defect Tolerant LUT Through Artificial Evolution," *Genetic Programming and Evolvable Machines*, vol. 12, no. 3, pp. 281–303, Sep. 2011. [Online]. Available: http://dx.doi.org/10.1007/s10710-011-9129-2

[112] L. Heller, W. Griffin, J. Davis, and N. Thoma, "Cascode voltage switch logic: A differential cmos logic family," in *Solid-State Circuits Conference. Digest of Technical Papers. 1984 IEEE International*, vol. XXVII, 1984, pp. 16–17.

[113] M. Stanisavljevic, A. Schmid, and Y. Leblebici, "Fault-tolerance of robust feedforward architecture using single-ended and differential deep-submicron circuits under massive defect density," in *Neural Networks, 2006. IJCNN '06. International Joint Conference on*, 2006, pp. 2771–2778.

[114] W.-J. Huang and E. McCluskey, "Column-Based Precompiled Configuration Techniques for FPGA," *Field-Programmable Custom Computing Machines, 2001. FCCM '01. The 9th Annual IEEE Symposium on*, pp. 137–146, March 2001.

[115] K. Inoue, Y. Nishitani, M. Amagasaki, M. Iida, M. Kuga, and T. Sueyoshi, "Fault detection and avoidance of FPGA in various granularities," in *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*, Aug. 2012, pp. 539–542.

[116] A. Doumar, S. Kaneko, and H. Ito, "Defect and fault tolerance FPGAs by shifting the configuration data," *IEEE Defect and Fault-Tolerance Symp.*, pp. 377–385, 1999.

[117] R. Kshirsagar and S. Sharma, "Fault Tolerance in FPGA through Horse Shifting," in *Emerging Trends in Engineering and Technology (ICETET), 2012 Fifth International Conference on*, Nov. 2012, pp. 228–232.

[118] J. Lach, W. Mangione-Smith, and M. Potkonjak, "Low overhead fault-tolerant FPGA systems," *IEEE (VLSI) Systems Transactions*, vol. 6, pp. 212–221, 1998.

[119] A. Yu and G. Lemieux, "FPGA defect tolerance: impact of granularity," *Field-Programmable Technology, 2005. Proceedings. 2005 IEEE International Conference on*, pp. 189–196, Dec. 2005.

[120] A. Corp., *Altera's patented redundancy technology dramatically increases yields on high-density APEX 20KE devices*, press release ed., November 2000.

[121] A. Yu and G. Lemieux, "Defect-tolerant FPGA switch block and connection block with fine-grain redundancy for yield enhancement," *IEEE Field Programmable Logic and Applications Proc.*, 2005.

[122] A. Doumar and H. Ito, "Design of switching blocks tolerating defects/faults in FPGA interconnection resources," in *Defect and Fault Tolerance in VLSI Systems, 2000. Proceedings. IEEE International Symposium on*, 2000, pp. 134–142.

[123] E. Ahmed and J. Rose, "The effect of LUT and cluster size on deep-submicron FPGA performance and density," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 12, no. 3, pp. 288 –298, March 2004.

[124] A. 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*, ser. FPGA '99. New York, NY, USA: ACM, 1999, pp. 69–78. [Online]. Available: http://doi.acm.org/10.1145/296399.296431

[125] Xilinx. FPGA - field-programmable gate array. [Online]. Available: http://www.alldatasheet.com/view.jsp?Searchword=Xc6vlx760

[126] R. Mersereau and T. Speake, "A unified treatment of Cooley-Tukey algorithms for the evaluation of the multidimensional DFT," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 29, no. 5, pp. 1011 – 1018, oct 1981.

[127] Cadence. Cadence Encounter RTL Compiler. [Online]. Available: http://www.cadence.com/products/ld/rtl_compiler/pages/default.aspx

[128] E. Sundar, V. Chandrasekhar, M. Sashikanth, V. Kamakoti, and V. Narayanan, "A novel CLB architecture to detect and correct SEU in LUTs of SRAM-based FPGAs," in *Field-Programmable Technology, 2004. Proceedings. 2004 IEEE International Conference on*, dec. 2004, pp. 121 – 128.

[129] S. Weininger and M. Pecht, "Exploring medical device reliability and its relationship to safety and effectiveness," in *Product Compliance Engineering, 2009. PSES 2009. IEEE Symposium on*, oct. 2009, pp. 1 –5.

Durcissement de circuits logiques reconfigurables

Arwa BEN DHIA

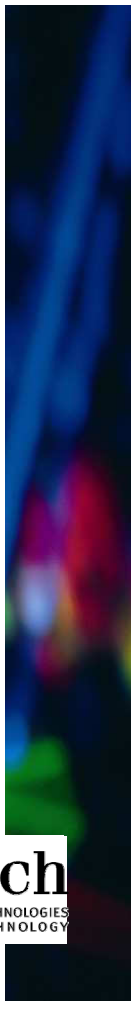
RESUME : Avec les réductions d'échelle, les circuits électroniques deviennent de plus en plus petits, plus performants, consommant moins de puissance, mais aussi moins fiables. En effet, la fiabilité s'est récemment érigée en défi majeur dans l'industrie micro-électronique, devenant un critère de conception important, au même titre que la surface, la consommation de puissance et la vitesse. Par exemple, les défauts physiques dûs aux imperfections dans le procédé de fabrication ont été observés plus fréquemment, affectant ainsi le rendement des circuits. Par ailleurs, les circuits nano-métriques deviennent pendant leur durée de vie plus vulnérables aux rayonnements ionisants, ce qui cause des fautes transitoires. Les défauts de fabrication, aussi bien que les fautes transitoires, diminuent la fiabilité des circuits intégrés. En avançant dans les nœuds technologiques, les circuits logiques programmables de type FPGA sont les premiers à entrer sur le marché, grâce à leur faible coût de développement et leur flexibilité qui leur permet d'être utilisés pour n'importe quelle application. Les FPGA possèdent des caractéristiques attrayantes, notamment pour les applications spatiales et aéronautiques, où la reconfigurabilité, les hautes performances et la faible consommation de puissance peuvent être exploitées pour développer des systèmes innovants. Néanmoins, les missions ont lieu dans un environnement rude, riche en radiations pouvant produire des erreurs soft dans les circuits électroniques. Ceci montre l'importance de la fiabilité des FPGA en tant que critère de conception dans les applications critiques. La plupart des FPGA commerciaux ont une architecture matricielle et leurs blocs logiques sont regroupés en *clusters*. Ainsi, cette thèse s'intéresse à la tolérance aux fautes des blocs de base (blocs logiques élémentaires (BLE) et boîtes d'interconnexion) dans un FPGA de type 'matrice de *clusters*'.

MOTS-CLEFS : Fiabilité ; Tolérance aux fautes ; FPGA

Hardening Basic Blocks in a Mesh of Clusters FPGA

ABSTRACT : As feature sizes scale down to nano-design level, electronic devices have become smaller, more performant, less power-consuming, but also less reliable. Indeed, reliability has arisen as a serious challenge in nowadays' microelectronics industry and as an important design criterion, along with area, performance and power consumption. For instance, physical defects due to imperfections in the manufacturing process have been observed more frequently, impacting the yield. Besides, nanometric circuits have become more vulnerable during their lifetime to ionizing radiation which causes transient faults. Both manufacturing defects and transient faults contribute to decreasing reliability of integrated circuits. When moving to a new technology node, Field Programmable Gate Arrays (FPGAs) are the first coming into the market, thanks to their low development and Non-Recurring Engineering (NRE) costs and their flexibility to be used for any application. FPGAs have especially attractive characteristics for space and avionic applications, where reconfigurability, high performance and low-power consumption can be fruitfully used to develop innovative systems. However, missions take place in a harsh environment, rich in radiation, which can induce soft errors within electronic devices. This shows the importance of FPGA reliability as a design criterion in safety and critical applications. Most of commercial FPGAs have a mesh architecture and their logic blocks are gathered into clusters. Therefore, this thesis deals with the fault tolerance of basic blocks (clusters and switch boxes) in a mesh of clusters FPGA.

KEY-WORDS : Reliability ; Fault tolerance ; FPGA



TELECOM ParisTech

EDITE

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY