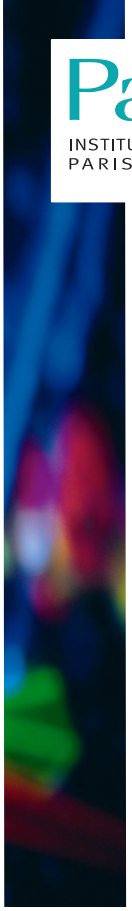


ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2012-ENST-039

EDITE - ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité « Communications et Electronique »

présentée et soutenue publiquement par

Gutemberg GONÇALVES DOS SANTOS JÚNIOR

le 6 septembre 2012

Conception Robuste de Circuits Numériques à Technologie Nanométrique

Directeurs de thèse : **Mme. Lirida NAVINER**
M. Jean-François NAVINER

Jury

M. Raoul Velazco, Directeur de recherche, CNRS-TIMA	Rapporteur
M. Luis Entrena, Professeur, Université Carlos III Madrid	Rapporteur
M. Habib Mehrez, Professeur, UPMC, LIP6	Examinateur
M. Gilles Deleuze, Chercheur senior, EDF R&D	Examinateur
Mme. Lirida Naviner, Professeur, Télécom Paristech	Directrice de Thèse
M. Jean-François Naviner, Maître de conférences, HDR, Télécom Paristech	Directeur de Thèse

T H È S E

TELECOM ParisTech
École de l'Institut Télécom - membre de ParisTech

École Doctorale
d'Informatique,
Télécommunications
et Électronique de Paris

Thèse

Robust Design of Deep-Submicron Digital Circuits

Gutemberg GONÇALVES DOS SANTOS JÚNIOR

Directeurs de thèse Pr. Dr. Lirida NAVINER
Dr. Jean-François NAVINER

"Education is the most powerful weapon which you can use to change the world."

Nelson Mandela

"Il n'y a aucune connaissance de la terre qui ne commence par l'imagination. Lorsqu'elle disparaît, lorsque se brise la création par l'imaginaire, la curiosité s'évanouit avec elle et le savoir s'épuise."

Francesco Alberoni

Acknowledgement

Working on a Ph.D has been an extraordinary experience. It would not have been possible to succeed without the support of many people who gave of their time tirelessly and patiently in order to make the difficult moments less unbearable.

First and foremost I offer my sincerest gratitude to my supervisors, Dr Lirida Naviner and Dr Jean-François Naviner, without whose sincerity, knowledge, encouragement and assistance this study would not have been successful. I'm deeply thankful for the excellent atmosphere you created for doing research, where everybody could contribute and learn from each other. Besides, you acted not only as a supervisor, but more importantly as a real friend, and I appreciate that from my heart.

I would like to express my deepest gratitude to EDF R&D for the financial support and the opportunity to carry out my research studies. I would like to extend my greetings to Bastien Cousin, Laurent Cretinon, Gilles Deleuze, Anne-Lise Didierjean, Laurent Doireau, Sandrine Legruel, and Philippe Mathevon, whose advice, support and patience were fundamental during my studies.

I would like to thank Dr Luis Entrena and Dr Raoul Velazco for agreeing to be the reviewers of this dissertation. Your knowledge and remarks were vital for the improvement of the current work.

Many thanks to all my friends for sharing their enthusiasm for and comments on my work. I would like to thank Alban Gruget, Arthur Liraneto, Arwa Ben Dhia, Bruno Lyra, Chadi Jabbour, Cibele Trinca, Daniel Caon, Davi Bibiano, Dimitri Edouard, Eduardo Ferraz, Elaine Crespo, Eric Bouton, Fabrice Linot, Farhan Mirani, Florent Lozac'h, Hasham Khushk, Joana Silveira, Julie Gaudin, Maí Correia, Márcia Costa e Silva, Mariem Slimani, Pietro Maris, Sami Mekki, Samuel Pagliarini, Sereuja Zier, Shivam Bhasin, Tian Ban, Yang Liu, and all the other students in Télécom Paristech and friends I made in Paris for the great moments we spent together. Just remembering the happy hours in the "butte-aux-cailles", the picnics in the parks and on the banks of the Seine, the travels, the dinners in Maisel, and all the parties we went make me laugh and feel good. I'm truly grateful to all of you guys, you are fantastic people, and I hope to see you all again very soon.

I would like to thank as well Chantal Cadiat, Florence Besnard, Zouina Sahnoune and

all the employees of Télécom Paristech who welcomed me and made my stay so memorable. You helped me so much with my problems, bureaucracy and so many other things that I can't really imagine finishing this thesis without your help.

I'm especially thankful to my family for their unflagging love, and for always supporting me during difficult times, listening to my worries and complaints.

Abstract

The design of circuits to operate at critical environments, such as those used in control-command systems at nuclear power plants, is becoming a great challenge with the technology scaling. These circuits have to pass through a number of tests and analysis procedures in order to be qualified to operate. In case of nuclear power plants, safety is considered as a very high priority constraint, and circuits designed to operate under such critical environment must be in accordance with several technical standards such as the IEC 62566, the IEC 60987, and the IEC 61513. In such standards, reliability is treated as a main consideration, and methods to analyze and improve the circuit reliability are highly required.

The present dissertation introduces some methods to analyze and to improve the reliability of circuits in order to facilitate their qualification according to the aforementioned technical standards. Concerning reliability analysis, we first present a fault-injection based tool used to assess the reliability of digital circuits. Next, we introduce a method to evaluate the reliability of circuits taking into account the ability of a given application to tolerate errors. Concerning reliability improvement techniques, first two different strategies to selectively harden a circuit are proposed. The first one is based on the assumption that some output bits of a circuit may be more important for a given application than the others. Then, the proposed technique drives the reliability improvement effort to those bits. The other technique uses a cost function in order to automatically select the best candidates to be hardened. Finally, a method to automatically partition a TMR design based on a given reliability requirement is introduced.

French Summary

Introduction

Depuis l'avènement de la micro-électronique, ce domaine n'a pas cessé de prendre de l'ampleur. Les technologies de fabrication ont vécu une évolution exponentielle comme prévu par la Loi de Moore [1, 2]. Comme résultat, les dispositifs électroniques deviennent de plus en plus petits, plus performants et moins chers.

Afin de continuer l'évolution de la micro-électronique même après l'arrivée des dimensions submicroniques, les chercheurs doivent surpasser des défis comme la considération des phénomènes physiques qui auparavant étaient négligeables et maintenant sont prépondérants, comme les forces de Casimir et de Van Der Waals [3]. De plus, les systèmes d'interconnections sont devenus très complexes avec l'arrivée du schéma de connexion 3-D [4]. En fait, l'augmentation de la quantité de composants dans la même puce et l'augmentation de la complexité des interconnections font croître la probabilité de défaillance des composants. En même temps, l'augmentation des fréquences d'opération augmente la probabilité des erreurs de synchronisation [5]. En conséquence, une réduction du rendement de fabrication aussi bien que de la fiabilité des circuits intégrés est attendue [6–10].

Avec l'augmentation de la probabilité de fautes dans les circuits numériques, les systèmes développés pour les environnements critiques comme les centrales nucléaires, les avions et les applications spatiales doivent être certifiés selon des normes industrielles. Cette thèse est un résultat d'une coopération CIFRE entre l'entreprise Électricité de France (EDF) R&D et Télécom Paristech. EDF est l'un des plus gros producteurs d'énergie au monde et possède de nombreuses centrales nucléaires. Les systèmes de contrôle-commande utilisé dans les centrales sont basés sur des dispositifs électroniques, qui doivent être certifiés selon des normes industrielles comme la CEI 62566 [11], la CEI 60987 [12] et la CEI 61513 [13] à cause de la criticité de l'environnement nucléaire. En particulier, l'utilisation des dispositifs programmables comme les FPGAs peut être considérée comme un défi du fait que la fonctionnalité du dispositif est définie par le concepteur seulement après sa conception physique. Le travail présenté dans ce mémoire porte sur la conception de nouvelles méthodes d'analyse de la fiabilité aussi bien que des méthodes d'amélioration de la

fiabilité d'un circuit numérique.

La fiabilité dans les circuits numériques

Un circuit électronique peut être vu comme l'assemblage d'un certain nombre de composants électroniques de telle façon qu'il produit une fonctionnalité souhaité. Cette fonctionnalité peut être garantie si on considère que les composants sont exempts de fautes. Malheureusement les dispositifs électroniques sont susceptibles de défaillances occasionnées par des mécanismes naturels comme les impuretés dans les matériaux et les variations de paramètres, entre autres. Pour être précis, il y a une certaine probabilité qu'un circuit numérique va fournir la fonctionnalité souhaitée pendant un période de temps. Cette probabilité est connue comme fiabilité et peut être définie comme suit : la fiabilité est l'aptitude d'un dispositif à accomplir une fonction requise dans des conditions données pour une période de temps donnée [14].

La fiabilité d'un circuit électronique peut être calculée selon (1). Son comportement par rapport au temps peut être divisé en 3 phases (voir FIGURE 1) :

1. Taux de défaillance décroissant → Cette phase de vie est aussi appelée période de jeunesse.
2. Taux de défaillance sensiblement constant → C'est aussi appelé période de vie utile du dispositif.
3. Taux de défaillance croissant → Correspond à la période de vieillissement du circuit

$$R(t) = e^{-\int_0^t \lambda(x)dx} \tag{1}$$

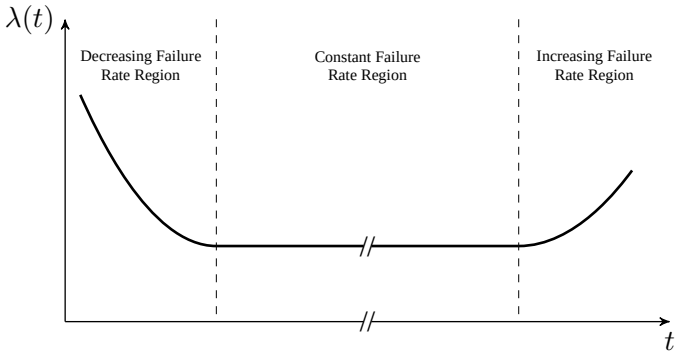


FIGURE 1 – Courbe en baignoire

Parmi les types de fautes qui peuvent occasionner une défaillance du système électronique, les fautes transitoires sont particulièrement une menace à cause de leur comporte-

ment aléatoire et leur grande probabilité d'occurrence. Les fautes transitoires peuvent être occasionnées par différents phénomènes physiques comme par exemple les particules alpha, les rayons cosmiques et les interférences électromagnétiques. De plus, la susceptibilité des circuits électroniques à ces types de phénomènes augmente avec la réduction de la taille des composants. Pour faire face à ces erreurs, les concepteurs peuvent utiliser des méthodes de durcissement d'un circuit intégré. Cela représente toujours un surcoût en surface, en consommation ou en vitesse. Ainsi, les méthodes d'analyse de la fiabilité d'un circuit intégré deviennent de plus en plus importantes avec l'évolution de la technologie. C'est grâce à ce type d'analyse que les concepteurs peuvent identifier les zones de défaillance potentielles, la nécessité d'ajout de redondance, la nécessité d'un système de sauvegarde, etc. De plus, les méthodes d'analyse de la fiabilité peuvent être utilisées comme un outil pour mesurer la performance de différentes stratégies de durcissement d'un circuit intégré.

Plusieurs méthodes d'analyse de la fiabilité d'un circuit ont été reportées dans la littérature. C'est bien connu qu'une analyse optimale doit prendre en considération autant d'information que possible du circuit lui-même aussi bien que de l'application cible. En dépit de cela, la plupart de méthodes d'analyse de fiabilité considère quelques simplifications dans les modèles mathématiques comme la considération de fautes simples, de signaux non corrélés, etc. En outre, peu de travaux sur l'analyse de la fiabilité ont été effectués tenant en compte l'importance des résultats du circuit pour l'application cible. En fait, beaucoup d'applications présentent la capacité de tolérer un certain nombre et certains types d'erreurs. En considérant cette information, un concepteur peut mieux contrôler l'ajout de redondance afin d'éviter un surcoût trop élevé. La première contribution de ce travail est une technique nommée « effective reliability » qui prend en considération la tolérance aux erreurs d'une application pour évaluer la fiabilité du circuit.

Effective Reliability

L'augmentation considérable du nombre d'erreurs attendue dans les circuits avec l'évolution de la technologie a inspiré les discussions sur la tolérance aux erreurs depuis la sortie du « *2001 International Technology Roadmap for Semiconductors* (ITRS) ». Le concept de tolérance aux erreurs a été introduit comme un paradigme orienté à l'application pour faire face aux variations du processus, aux défauts et au bruit [7]. L'idée principale est de que certaines applications présentent la capacité de tolérer un certain nombre et certains types d'erreurs à condition qu'ils soient limités à un certain niveau de sévérité défini par l'application. En effet, plusieurs applications multimédia présentent cette caractéristique grâce au fonctionnement des sens humains comme la vue, l'audition et l'odorat, qui ne peuvent pas s'apercevoir de la présence de certains types d'erreurs. Ce mémoire se réfère à ce type de phénomène comme les masquages des erreurs par l'application et introduit la

classification d'erreurs suivante :

- Erreurs critiques → Ce sont les erreurs qui peuvent occasionner un grand impact dans les résultats produits par un circuit ;
- Erreurs non-critiques → Ce sont les erreurs qui sont masquées par l'application.

Cette classification des erreurs prend en considération l'usage des résultats produits par un circuit. Par conséquence, un facteur très important qui affecte ce type de classification est l'approche de codification utilisée pour représenter l'information dans la sortie du circuit. En fait, l'impact d'une inversion d'un bit de sortie du circuit dépend directement de son poids, c'est-à-dire de sa signifiance relative par rapport au mot de sortie. En dépit de cela, le concept traditionnel de fiabilité d'un circuit (appelé fiabilité nominale dans ce mémoire) est basée sur le paradigme de passer ou échouer, c'est-à-dire il ne prend pas en considération l'importance d'un bit de sortie comme décrit en (2).

$$R_{nom} = \prod_{i=0}^{M-1} q_i \tag{2}$$

Pour faire face à ces problèmes, nous proposons le concept de « effective reliability » comme décrit par (3) et (4). Dans ce cas, le terme R_{ack} représente la probabilité qu'une erreur soit masquée par l'application cible. Cela veut dire que le terme R_{ack} prend en considération les erreurs qui sont classées comme non-critiques alors que le terme $R_{\overline{ack}}$ considère les erreurs critiques. La classification d'une erreur en critique ou non-critique prend en compte des métriques de qualité qui sont considérées pertinentes par rapport à l'application cible. Ce mémoire introduit aussi deux métriques de qualité différentes basées d'importance d'un bit (voir (5) et (6)) et d'erreur relative (voir (7) et (8)).

$$R_{eff} = R_{nom} + R_{ack} \tag{3}$$

$$R_{eff} = 1 - R_{\overline{ack}} \tag{4}$$

$$R_{eff} = \prod_{i=0}^{M-1} q_i + \sum_{k=1}^{T+1} \sum_{r=1}^{C_k^{T+1}} \gamma_{k,r} \tag{5}$$

$$R_{eff} = 1 - \sum_{k=1}^{T+1} \sum_{r=C_k^{T+1}+1}^{C_k^M} \gamma_{k,r} - \sum_{k=T+2}^{M} \sum_{r=1}^{C_k^M} \gamma_{k,r} \tag{6}$$

TABLE 1 – Valeurs de la fiabilité de chaque bit de sortie de l'APPR8

Sortie	Fiabilité (q_i)
b_0	99.80%
b_1	99.48%
b_2	99.31%
b_3	99.24%
b_4	99.20%
b_5	99.18%
b_6	99.17%
b_7	99.16%
b_8(retenue)	99.36%

$$R_{eff} = \prod_{i=0}^{M-1} q_i + \sum_{a=0}^{2^H-1} p(a) \sum_{k=1}^{k_{max}} \sum_{r=1}^{C_k^M} \gamma_{k,r}.u\left(\delta_{max} - \delta(k,r,a)\right) \tag{7}$$

$$R_{eff} = 1 - \sum_{a=0}^{2^H-1} p(a) \sum_{k=1}^{k_{max}} \sum_{r=1}^{C_k^M} \gamma_{k,r}.u\left(\delta(k,r,a) - \delta_{max}\right) \tag{8}$$

Résultats

Prenons comme exemple un additionneur parallèle à propagation retenue de 8 bit (APPR8), construit à partir de 8 additionneurs de 1 bit en chaîne, dans lequel la probabilité de défaillance de chaque porte logique est égale à 99.9%. Supposons que la contrainte de fiabilité minimale de l'APPR8 soit $R_{min} = 95\%$ et que l'application cible présente la capacité de tolérer des erreurs aussi grandes que 2% du résultat correct ($\delta_{max} = 2\%$).

La fiabilité de chaque bit de sortie du APPR8 a été évaluée en utilisant la méthode SPR-MP [15], et les résultats sont illustrés dans le Tableau 1. La concept de fiabilité nominale peut donc être calculée selon (9). En analysant le résultat pour la fiabilité nominale un concepteur ira conclure que l'APPR8 ne respecte pas la contrainte de fiabilité minimale et que le circuit a besoin d'être durci. En considérant la méthode TMR (*Triple Modular Redundancy*) pour réaliser cette procédure, l'architecture durcie avec moins de surface mais qui encore respecte la contrainte de fiabilité minimale cause un surcoût en surface de 75%.

$$R = \prod_{i=0}^{8} q_i = 94.06\% \tag{9}$$

Rappelons que l'application cible présente la capacité de tolérer des erreurs s'ils ne dépassent pas la contrainte d'erreur relative $\delta_{max} = 2\%$. Prenons donc le concept de « effective reliability » pour analyser la fiabilité du circuit. Dans ce cas, la fiabilité du circuit

TABLE 2 – Valeurs de R_{eff} pour différentes tolérances aux erreurs (APPR8)

Erreur Relative (δ_{max})	Fiabilité
0.5%	94.23%
1.0%	94.64%
1.5%	94.96%
2.0%	95.22%
2.5%	95.44%
3.0%	95.62%
3.5%	95.77%
4.0%	95.92%
4.5%	96.05%
5.0%	96.16%

dépend de la capacité de l'application cible de tolérer des erreurs (voir Tableau 2). C'est bien noté qu'en considérant une capacité de tolérance d'erreurs $\delta_{max} = 2\%$ la fiabilité du circuit pour cette application est égale à 95.22%, c'est-à-dire la contrainte de fiabilité minimale est déjà respectée et en fait il n'y a pas besoin d'ajout de redondance.

Le concept de « effective reliability » est très intéressant pour les applications dans lesquelles un certain nombre d'erreurs peut être toléré. Par contre, dans les applications appelées critiques comme les centrales nucléaires, les avions et les satellites, l'occurrence d'une seule erreur peut causer des conséquences sévères. En fait, les circuits développés pour ces environnements ont besoin d'une couverture de test qui s'approche de 100%. Pour faire face à ces problèmes, la deuxième contribution de ce travail est un outil basée sur Verilog appelé FIFA (Fault-Injection-Fault-Analysis) développé pour accélérer les tests exhaustifs dans les circuits intégrés.

L'outil FIFA

C'est déjà bien connu que l'injection de fautes est une approche intéressant pour analyser le fonctionnement des circuits intégrés en présence de fautes. L'idée principale est d'injecter des fautes dans le circuit de forme aléatoire ou contrôlée et analyser si la faute est propagée jusqu'à la sortie. Comme les circuits intégrés deviennent de plus en plus complexes avec l'évolution de la technologie, le temps nécessaire pour atteindre un niveau élevé de couverture de test est très important, voire prohibitif. Cela devient un problème pour les applications qui ont besoin de tel niveau de couverture, et de ce fait le développement des nouvelles méthodes que puissent accélérer la procédure d'injection de fautes devient nécessaire. La deuxième contribution de ce travail est un outil appelé FIFA (Fault-Injection-Faut-Analysis) qui a été développé comme un « hardware IP » pour accélérer l'analyse de fiabilité basée sur l'injection de fautes. Cet outil est adapté à différents modèles de fautes

et à des fautes multiples.

L'architecture de mise en œuvre de l'outil FIFA est illustrée dans la FIGURE 2. Nous pouvons noter qu'il y a deux versions du dispositif sous test (DUT). Le module «DUT REF» correspond à une version idéale du dispositif sous test, tandis que le module appelé «DUT FAULTY» est une copie du «DUT REF» dans laquelle des saboteurs ont été ajoutés . Un saboteur est un dispositif électronique capable de changer la valeur logique d'un nœud du circuit. Le saboteur qui a été développé pour l'outil FIFA supporte quatre types de fautes différentes : les inversions de bit, les collages à zéro, les collages à un, et les hautes impédances. Son schéma est illustré dans la FIGURE 3.

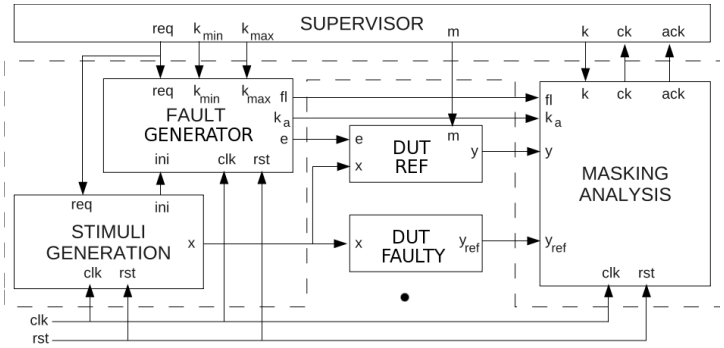


FIGURE 2 – Schéma général de l'outil proposé

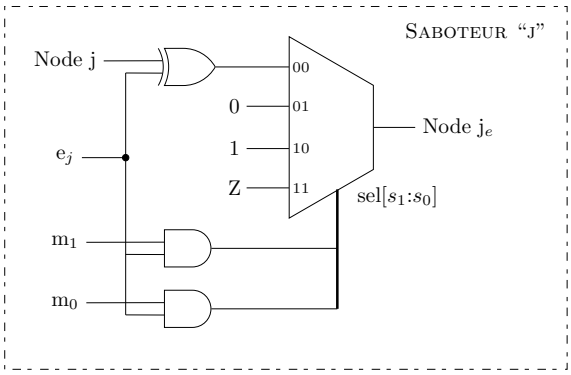


FIGURE 3 – Schéma général d'un saboteur

L'évaluation de la fiabilité d'un circuit à partir des résultats fournis par l'outil FIFA est basée sur la méthode PBR [16]. Le module «FAULT GENERATOR» a été développé pour générer des erreurs en ordre croissante de multiplicité, de sorte que le concepteur peut limiter le nombre de tests à effectuer s'il connaît le nombre maximal de fautes simultanées.

Afin d'analyser la performance et la quantité de ressources utilisées par l'outil FIFA, nous l'avons comparé avec une plateforme reportée dans la littérature nommée FuSE [17].

TABLE 3 – Plate-forme Fuse vs. outil FIFA

	Fuse [17]	FIFA
ALUTs	2157	817
Registres	694	467
Fréquence maximale	75.1MHz	109.87MHz

L'implémentation a été faite dans un STRATIX II EP2S180F1508C3 et il a été considéré un circuit avec $N = 10$ saboteurs et $P = Z = 32$ entrées et sorties. Les résultats de comparaison peuvent être analysés dans le Tableau 3. Les résultats de synthèse de l'outil FIFA sont présentés dans la FIGURE 4.

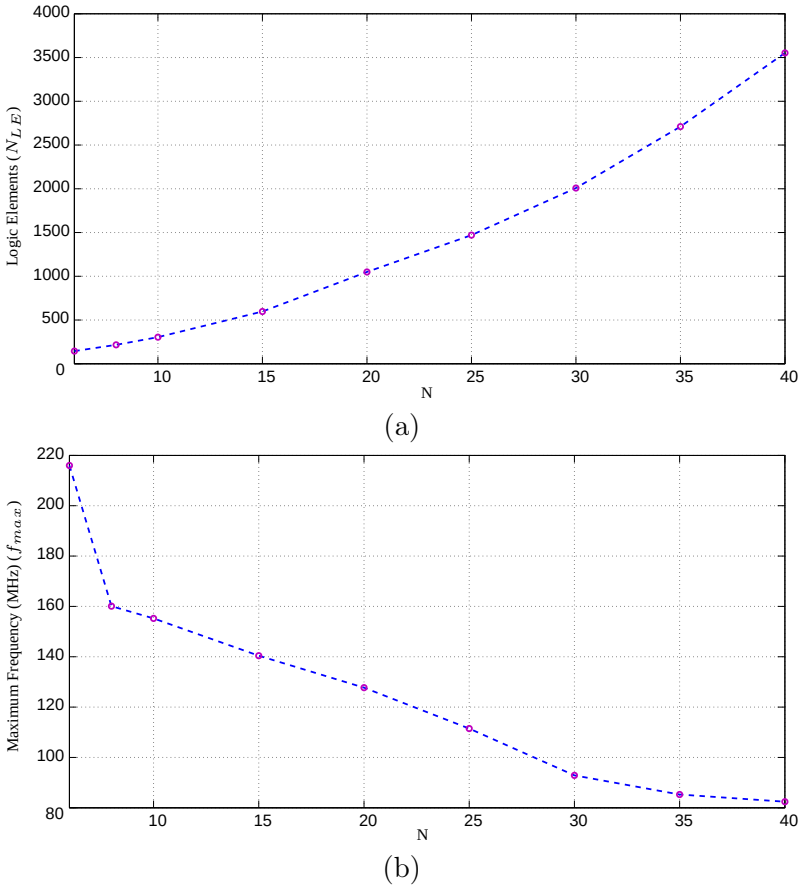


FIGURE 4 – Résultats de la synthèse de l'outil (jusqu'à N erreurs simultanées) : (a) nombre d'éléments logiques nécessaires dans le FPGA (b) fréquence maximale d'injection de fautes

Magré la bonne performance de l'outil FIFA, l'analyse de la fiabilité de circuits complexes reste très coûteuse en temps. Pour surmonter ce problème, nous proposons d'utiliser la technique de parallélisme. La FIGURE 5 illustre une architecture parallèle très simple pour l'outil FIFA. Le problème avec cette approche est la grande surface additionnelle qui

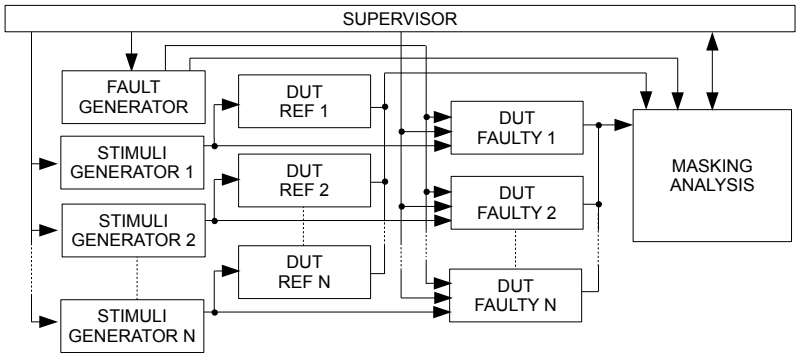


FIGURE 5 – Une simple architecture parallèle pour la FIFA

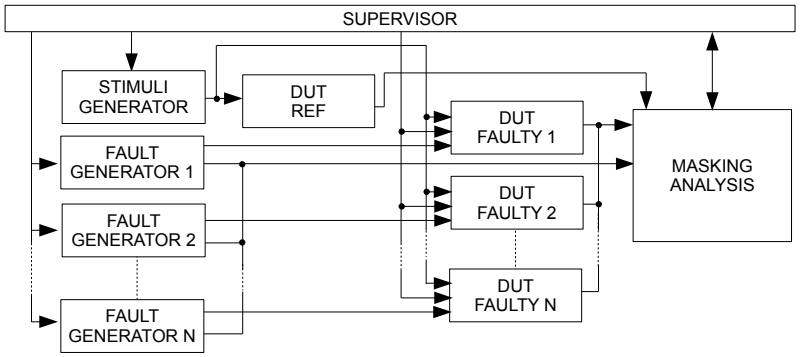


FIGURE 6 – L'architecture en parallèle proposée pour la FIFA

devient nécessaire à cause de la réplication des modules «STIMULI GENERATOR», «DUT FAULTY» et «DUT REF». En fait, les modules «DUT REF» et «DUT FAULTY» répliqués requièrent une surface qui peut être significative par rapport à celles des autres modules. Une architecture qui évite la réplication de «DUT REF» est illustrée dans la FIGURE 6. Dans ce cas il est nécessaire de répliquer seulement les modules «DUT FAULTY» et «FAULT GENERATOR».

Afin d'implémenter correctement l'architecture illustrée dans la FIGURE 6, il faut bien distribuer la génération de vecteurs de fautes entre les N modules «FAULT GENERATOR». C'est important de remarquer que la génération de fautes est faite en ordre croissante concernant le nombre de fautes simultanées. En conséquence, il faut développer un algorithme pour calculer les vecteurs de fautes qui iront initialiser chaque module «FAULT GENERATOR». Cet algorithme a été developpé en utilisant quelques régularités numériques comme décrit dans le Chapitre 2. Les résultats, présentés dans la FIGURE 7, prouvent la bonne distribution de la génération de vecteurs de fautes.

Jusqu'à ce point il a été introduit deux méthodes pour analyser la fiabilité d'un circuit numérique. Cette analyse est généralement utilisée pour certifier le fonctionnement correct du circuit pendant sa période de vie utile. Si sa fiabilité ne respecte pas la contrainte de fiabilité minimale, les concepteurs peuvent réaliser le durcissement du circuit. Générale-

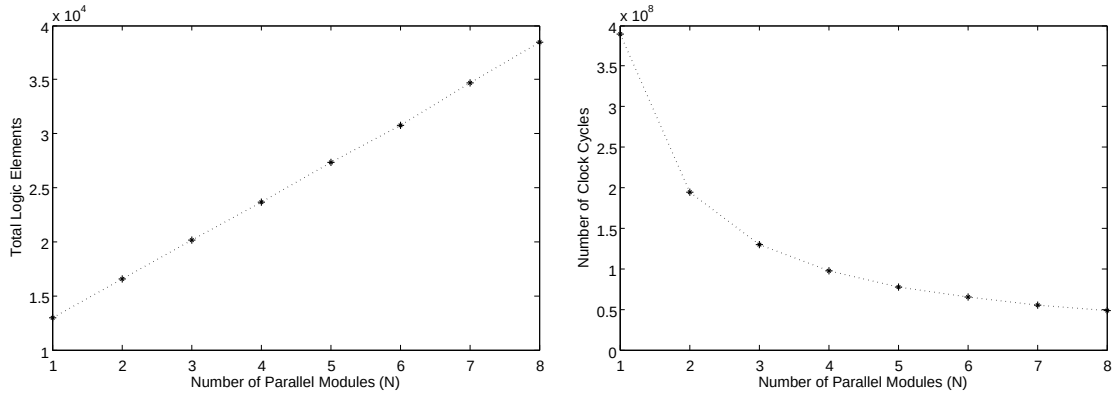


FIGURE 7 – Performance de l'architecture en parallèle proposée pour la FIFA

ment, la protection partiale d'un circuit contre défaillances est suffisante pour la plupart des applications. Ainsi, le développement des nouvelles méthodes basée sur une procédure de durcissement sélectif devient nécessaire. La troisième contribution de ce travail concerne le développement de deux techniques pour identifier les portes logiques les plus critiques et ainsi permettre de réaliser le durcissement sélectif d'un circuit.

Durcissement Sélectif

Les techniques de durcissement sélectif d'un circuit offrent un bon compromis entre l'augmentation de sa fiabilité et le surcoût correspondant. Ces techniques consistent fondamentalement de deux étapes : les portes ou blocs logiques sont analysés et ordonnés selon leur susceptibilité aux fautes et la probabilité que ces fautes produisent une défaillance du système ; ensuite, les portes ou blocs logiques les plus critiques sont protégés en utilisant une technique de durcissement choisie par le concepteur. La difficulté de mise en œuvre du durcissement sélectif réside dans l'identification des portes ou blocs logiques les plus critiques pour une application. La première technique proposée dans ce travail considère l'utilisation des résultats produits par un circuit comme le facteur déterminant de la criticité d'un bloc logique.

Évitement des erreurs critiques dans les circuits intégrés

Un problème présent dans la plupart des méthodes de durcissement sélectif est négliger le profil d'utilisation des résultats d'un circuit par l'application cible. En fait, le concept de fiabilité nominale ne prend pas en compte la quantité d'information que chaque bit de sortie contient pour évaluer la fiabilité d'un circuit. Ce fait peut être illustré en considérant 3 architectures d'un circuit additionneur de 4 bits. La fiabilité de chaque bit de sortie ($\mathbf{y} = b_3 b_2 b_1 b_0$) de ces 3 architectures est donnée dans le Tableau 4. La fiabilité nominale

TABLE 4 – Fiabilité pour les bits de sortie de 3 architectures différentes d'un additionneur de 4-bit

Architecture	b_3	b_2	b_1	b_0	$R_{nominal}$	$R_{practical}$
1	99%	99%	99%	95%	92.18%	97.63%
2	95%	99%	99%	99%	92.18%	94.17%
3	98%	99%	99%	95%	91.25%	96.64%

peut être calculée selon (10), et les résultats correspondants sont aussi disponibles dans le Tableau 4. Concernant les valeurs pour la fiabilité nominale de ces architectures, un concepteur conclura que les architectures 1 et 2 sont également fiables. Cependant, en analysant la fiabilité de chaque bit de sortie pour ces architectures, il est évident que l'architecture 1 fournit des résultats plus en conformité avec l'application que l'architecture 2.

$$R_{nominal} = \prod_{i=0}^{M-1} R_i \tag{10}$$

Pour faire face à ce problème, ce travail propose le concept de fiabilité pratique. Basée sur le fait que chaque bit de sortie d'un circuit peut avoir une importance différente pour une certaine application, la fiabilité pratique utilise un facteur k_i pour déterminer la sévérité d'une erreur dans un bit de sortie spécifique (voir (11)). Par exemple, dans le cas d'utilisation d'une codification binaire, la valeur de k_i est calculée selon (12). Ce concept corrige le problème décrit par l'exemple du additionneur 4 bits comme illustré dans le Tableau 4

$$R_{practical} = \prod_{i=0}^{M-1} R_i^{k_i} \tag{11}$$

$$k_i = \frac{1}{2^{(M-1)-i}} \tag{12}$$

L'utilisation de la fiabilité pratique dans le processus de durcissement sélectif d'un circuit est illustrée avec l'additionneur de 4-bit de la FIGURE 8. Dans ce cas, la première étape consiste à identifier les portes logiques les plus critiques selon le modèle de fiabilité pratique. Les résultats sont présentés dans le Tableau 5. La méthode [18] a été aussi implémentée et les résultats ont été comparés avec la méthode proposée en considérant la même contrainte de surface (voir Tableau 6).

Les résultats présents dans le Tableau 6 prouvent l'efficacité de la méthode proposée. En fait, il peut être noté que le gain en fiabilité est plus marqué pour les bits les plus

significatifs du circuit (les plus critiques). En conséquence, le circuit durcit par la méthode proposée exhibe la plus grande fiabilité pratique.

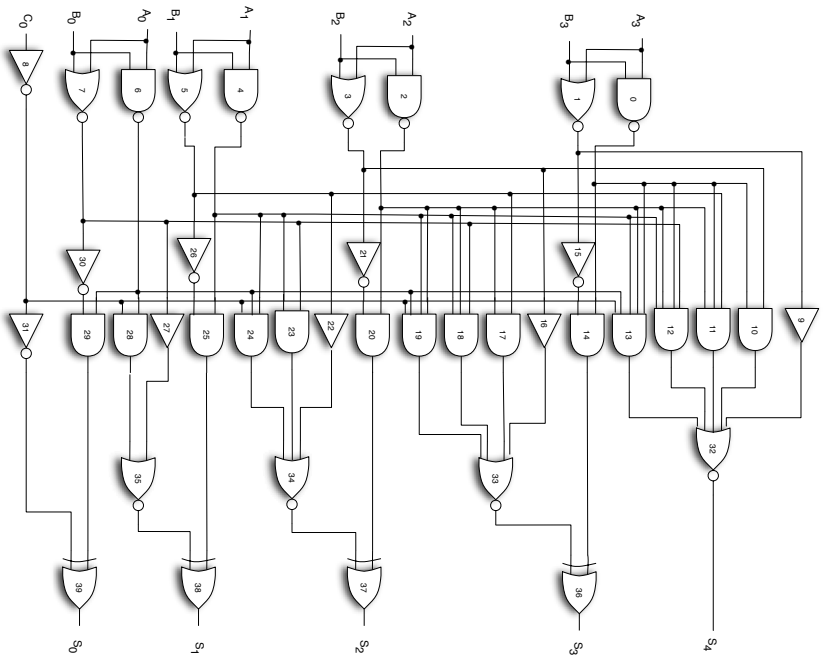


FIGURE 8 – Schéma en portes logiques du circuit 74283

L'utilisation d'une fonction de coût pour déterminer les portes critiques

La méthode décrite dans les paragraphes précédents ne s'applique pas à toutes les applications. En fait, plusieurs applications ne présentent pas une différence d'importance entre les bits de sortie du circuit. Pour ces applications, ce travail propose aussi une méthode basée sur des fonctions de coût pour automatiser le processus d'identification des portes logiques critiques. Cette méthode utilise le modèle SPR pour évaluer la fiabilité d'un circuit et déterminer les blocs logiques offrant la meilleure relation entre gain en fiabilité et coût.

Prenons un circuit composé de K portes logiques $[g_i \cdots g_k]$ pour lesquelles les fiabilités sont représentées par $[q_i \cdots q_k]$ et la fiabilité total du circuit par R. Donc, en considérant un gain en fiabilité dans la porte logique g_i, la fiabilité total du circuit devient R_i^*. Dans ce cas, deux portes logiques g_i et g_j peuvent contribuer différemment pour la fiabilité totale du circuit (R_i^* et R_j^*). Il faut définir donc un paramètre pour qu'une fonction de coût puisse être utilisée. Dans ce travail, nous proposons un paramètre appelé « hardware affinity » (Cha_i) qui peut être lié à n'importe quelle contrainte du circuit. Par exemple, le Tableau 7 utilise la surface des portes logiques obtenue par une synthèse basée sur la bibliothèque de Synopsis [19]. Il faut noter que le durcissement des portes logiques avec une valeur de Cha_i plus grande sera prioritaire. Une fonction de coût C_i peut donc être exprimée comme

TABLE 5 – Classification pour les portes logiques du circuit 74283

g_i	S_0	S_{0_w}	S_1	S_{1_w}	S_2	S_{2_w}	S_3	S_{3_w}	S_4	S_{4_w}	$\sum e_w$	$CritFac$
0	0	0	0	0	0	0	384	3072	192	3072	6144	36
1	0	0	0	0	0	0	384	3072	320	5120	8192	38
2	0	0	0	0	384	1536	192	1536	96	1536	4608	33
3	0	0	0	0	384	1536	320	2560	160	2560	6656	37
4	0	0	384	768	192	768	96	768	48	768	3072	25
5	0	0	384	768	320	1280	160	1280	80	1280	4608	32
6	384	384	192	384	96	384	48	384	24	384	1920	14
7	384	384	320	640	160	640	80	640	40	640	2944	23
8	512	512	256	512	128	512	64	512	32	512	2560	22
9	0	0	0	0	0	0	0	0	320	5120	5120	35
10	0	0	0	0	0	0	0	0	288	4608	4608	34
11	0	0	0	0	0	0	0	0	272	4352	4352	31
12	0	0	0	0	0	0	0	0	264	4224	4224	29
13	0	0	0	0	0	0	0	0	272	4352	4352	31
14	0	0	0	0	0	0	512	4096	0	0	4096	27
15	0	0	0	0	0	0	384	3072	0	0	3072	24
16	0	0	0	0	0	0	320	2560	0	0	2560	21
17	0	0	0	0	0	0	288	2304	0	0	2304	20
18	0	0	0	0	0	0	272	2176	0	0	2176	18
19	0	0	0	0	0	0	288	2304	0	0	2304	20
20	0	0	0	0	512	2048	0	0	0	0	2048	17
21	0	0	0	0	384	1536	0	0	0	0	1536	13
22	0	0	0	0	320	1280	0	0	0	0	1280	12
23	0	0	0	0	288	1152	0	0	0	0	1152	10
24	0	0	0	0	320	1280	0	0	0	0	1280	12
25	0	0	512	1024	0	0	0	0	0	0	1024	7
26	0	0	384	768	0	0	0	0	0	0	768	6
27	0	0	320	640	0	0	0	0	0	0	640	4
28	0	0	384	768	0	0	0	0	0	0	768	6
29	512	512	0	0	0	0	0	0	0	0	512	2
30	384	384	0	0	0	0	0	0	0	0	384	0
31	512	512	0	0	0	0	0	0	0	0	512	1
32	0	0	0	0	0	0	0	0	512	8192	8192	39
33	0	0	0	0	0	0	512	4096	0	0	4096	27
34	0	0	0	0	512	2048	0	0	0	0	2048	15
35	0	0	512	1024	0	0	0	0	0	0	1024	8
36	0	0	0	0	0	0	512	4096	0	0	4096	28
37	0	0	0	0	512	2048	0	0	0	0	2048	16
38	0	0	512	1024	0	0	0	0	0	0	1024	9
39	512	512	0	0	0	0	0	0	0	0	512	3

TABLE 6 – Analyse de fiabilité du circuit 74283

Fiabilité	Sans durcissement	Méthode en [18]	Méthode proposée
S_0	94.07%	94.97%	94.07%
S_1	92.39%	93.26%	92.39%
S_2	91.80%	92.65%	92.43%
S_3	91.33%	92.17%	93.07%
S_4	94.60%	95.51%	97.15%
$R_{nominal}$	68.93%	72.24%	72.63%
$R_{practical}$	87.29%	88.89%	90.65%

en (13).

TABLE 7 – Paramètre Cha_i pour quelques cellules logiques

Cellule	**Surface** (μm^2)	Cha_i
INVX0	5.5296	1
NAND2X0	5.5296	1
NOR2X0	5.5296	1
AND2X1	7.3728	0.75
OR4X1	10.1376	0.55
XOR3X1	22.1184	0.25

$$\begin{aligned} Rg_i &= R_i^* - R \\ C_i &= Rg_i / Cha_i \end{aligned} \tag{13}$$

La méthode proposée évalue la fiabilité du circuit et identifie les portes logiques qui seront durcies jusqu'à ce qu'un niveau minimal de fiabilité 'T' soit atteint. En utilisant cette méthodologie pour les circuits du benchmark ISCAS [20], deux profils pour la fonction de coût ont été obtenus (voir FIGURE 9 et 10). Le premier présente une décroissance très marquée juste après le début de la courbe, et le deuxième présente la formation des plateaux jusqu'à la fin de l'évaluation.

A partir de ce constat, ce travail propose aussi deux heuristiques pour trouver un point d'arrêt pour la méthode. Le premier est appelé l'heuristique de la somme des éléments et est calculé selon (14). Dans ce cas, C_0 représente la valeur de la fonction de coût du meilleur candidat pour le durcissement, et K est une contrainte empirique choisie par le concepteur. Le deuxième s'appelle l'heuristique basée sur le pourcentage et utilise un point

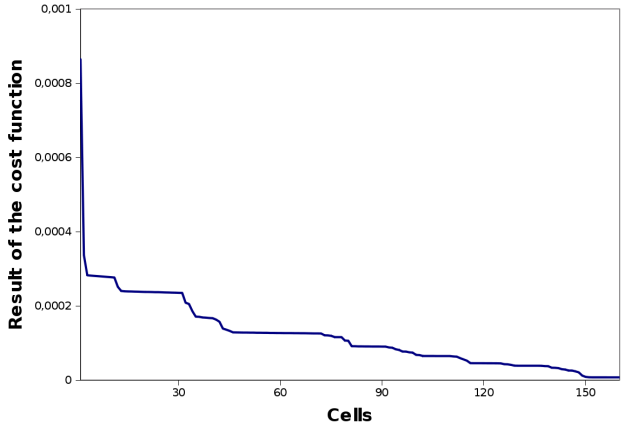


FIGURE 9 – Profil de la fonction de coût pour le circuit *c432*

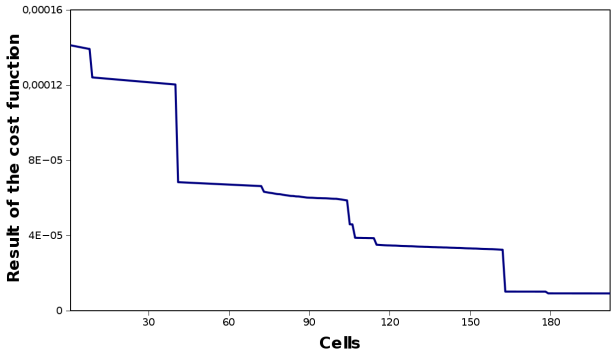


FIGURE 10 – Profil de la fonction de coût pour le circuit *c499*

d'arrêt pour l'algorithme égal à $X\%$ du valeur de C_0. L'utilisation des deux heuristiques peut être analysée dans la FIGURE 11.

$$\sum_{i=2}^{j} C_i \quad \leq \quad K \times C_0 \tag{14}$$

Les deux heuristiques ont été utilisées pour réaliser le durcissement de plusieurs circuits du benchmark ISCAS. Les résultats de cette utilisation sont indiqués dans les Tableaux 8 et 9.

Le durcissement sélectif peut offrir un bon compromis entre le gain en fiabilité du système et l'ajout de redondance nécessaire. Cela est une caractéristique fondamentale pour la plupart des circuits. Cependant, il y a des applications qui requièrent un niveau de fiabilité très élevé. Pour ces applications un durcissement sélectif peut ne pas être suffisant et l'utilisation de méthodes offrant un niveau de protection plus haut est intéressante, même au pris d'un surcoût significatif de surface. La quatrième contribution de ce travail

TABLE 8 – Résultats pour l'heuristique de la somme des éléments, $K = 10$

Circuit	**Nombre de portes**	**Surface original (μm^2)**	**Portes durcit**	**Surface durcit (μm^2)**	**Surcoût en surface**
c17	6	33.1776	6	99.5328	200%
74283	40	306.5096	20	547.9688	78.7%
c432	160	1134.4672	33	1541.4208	35.8%
c499	202	2155.1680	12	2414.1504	12.0%
c1355	546	3194.7328	11	3316.3840	3.8%
c1908	880	5273.7488	13	5417.5184	2.7%
c2670	1269	8018.0632	19	8233.7176	2.6%
c3540	1669	10855.1824	25	11177.7424	2.9%
c5315	2307	15293.5992	20	15518.4696	1.4%

TABLE 9 – Résultats pour l'heuristique basée sur la pourcentage, $X = 50\%$

Circuit	**Nombre de portes**	**Surface original (μm^2)**	**Portes durcit**	**Suface durcit (μm^2)**	**Surcoût en surface**
c17	6	33.1776	5	88.4736	166.6%
74283	40	306.5096	9	406.0424	32.5%
c432	160	1134.4672	2	1187.5264	4.6%
c499	202	2155.1680	41	2854.6752	32.4%
c1355	546	3194.7328	201	5647.1232	76.7%
c1908	880	5273.7488	119	6611.912	25.3%
c2670	1269	8018.0632	10	8128.6552	1.4%
c3540	1669	10855.1824	8	10963.9312	1.2%
c5315	2307	15293.5992	15	15459.4872	1.1%

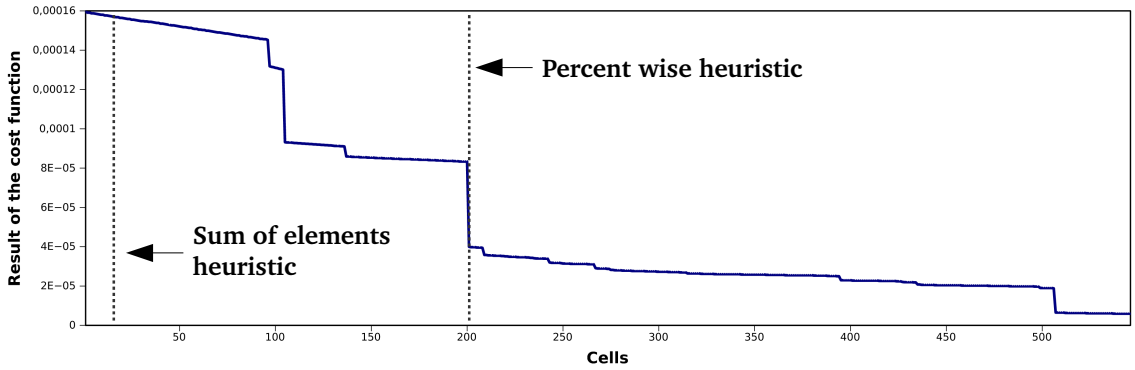


FIGURE 11 – L'utilisation des deux heuristiques dans le circuit *c1355*

est une méthode pour optimiser la procédure de partitionnement d'un circuit TMR de telle façon que un niveau de fiabilité très élevé soit assuré.

Optimisation du placement des arbitres dans un circuit TMR

La méthode TMR est souvent utilisée pour réaliser le durcissement d'un circuit intégré. L'idée générale est très simple : trois répliques du circuit fournissent les résultats pour un arbitre qui juge quel est la sortie exacte en utilisant normalement le critère de majorité (voir FIGURE 12). Dans ce cas, même avec la présence d'une erreur dans la sortie d'un module, l'arbitre peut fournir la sortie exacte. Donc, la fiabilité totale du circuit initial (R_m) devient R_{cir} comme illustré dans (15).

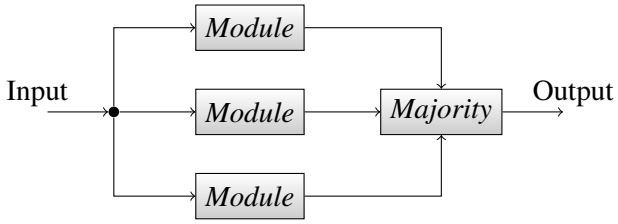
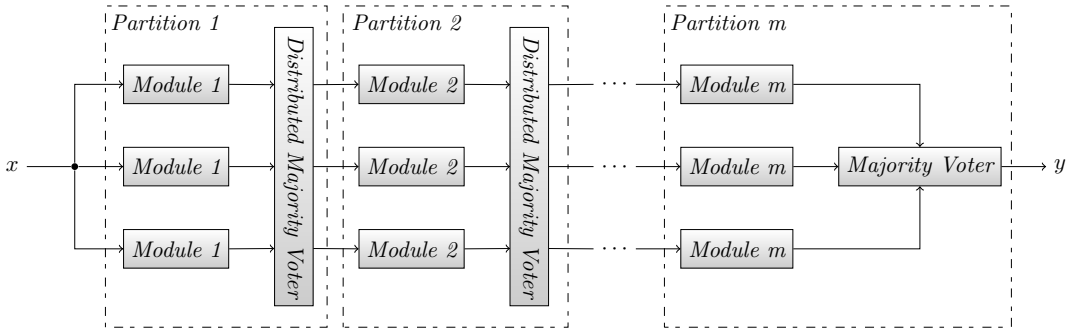


FIGURE 12 – Schéma en blocs de la méthode TMR

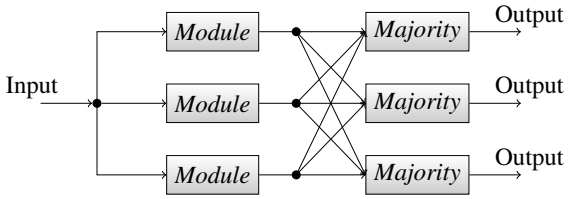
$$\begin{aligned} R_{cir} &= R_m^3 + 3R_m^2(1 - R_m) \\ R_{cir} &= 3R_m^2 - 2R_m^3 \end{aligned} \tag{15}$$

En dépit de sa simplicité, la méthode TMR offre un bon niveau de protection contre défaillances. Si une application requiert un niveau de fiabilité plus élevé que celui fourni par le TMR, il est possible de réaliser un partitionnement du circuit de telle façon qu'il soit composé par 'm' modules et sa fiabilité soit donnée par (16). En considérant que

chaque module est protégé par TMR (voir FIGURE 13), la fiabilité total du circuit peut être déterminée selon (17).



(a) Schéma TMR utilisant des partitions



(b) Distributed majority voter

FIGURE 13 – Circuit protégé par des partitions TMR

$$R_{C_1} = \prod_{k=1}^{n} (R_{m_k}) \tag{16}$$

$$R_{C_{1_{TMR}}} = \prod_{k=1}^{n} (3R_{m_k}^2 - 2R_{m_k}^3) \tag{17}$$

L'approche de partitionnement d'un circuit TMR peut offrir un niveau très élevé de fiabilité qui dépend de deux facteurs principaux : la quantité 'n' des modules et le placement des arbitres pour ces modules. Donc, une question qui devient intéressante est comment déterminer ces deux facteurs d'une façon optimale pour qu'un niveau de fiabilité minimale R_{min} soit atteint.

Ce problème peut être divisé en deux parties. En considérant une quantité 'n' de partitions, il faut évaluer la valeur de la fiabilité de chaque module R_{m_k} pour que la fiabilité totale $R_{C_{1_{TMR}}}$ soit maximisée. Cela peut être obtenu avec la méthode des multiplicateurs de Lagrange appliquée dans les équations (16) et (17) conforme décrit en (18).

$$g(R_{m_n}, R_{m_{n-1}}, \cdots, \lambda) = \prod_{k=1}^{n}(3R_{m_k}^2 - 2R_{m_k}^3) - \lambda(\prod_{k=1}^{n}(R_{m_k}) - R_{C_1}) \tag{18}$$

La résolution de ce problème implique une série de dérivées partielles comme décrit ci-après :

$$\begin{cases} \frac{\partial}{\partial R_{m_1}} g = (6R_{m_1} - 6R_{m_1}^2)\prod_{k=2}^{n}(3R_{m_k}^2 - 2R_{m_k}^3) - \lambda\prod_{k=2}^{n}(R_{m_k}) = 0 \\ \frac{\partial}{\partial R_{m_2}} g = (6R_{m_2} - 6R_{m_2}^2)(3R_{m_1}^2 - 2R_{m_1}^3)\prod_{k=3}^{n}(3R_{m_k}^2 - 2R_{m_k}^3) - \lambda R_{m_1}\prod_{k=3}^{n}(R_{m_k}) = 0 \\ \frac{\partial}{\partial R_{m_3}} g = (6R_{m_3} - 6R_{m_3}^2)\prod_{k=1}^{2}(3R_{m_k}^2 - 2R_{m_k}^3)\prod_{k=4}^{n}(3R_{m_k}^2 - 2R_{m_k}^3) - \lambda R_{m_1}R_{m_2}\prod_{k=4}^{n}(R_{m_k}) = 0 \\ \quad\vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ \frac{\partial}{\partial R_{m_n}} g = (6R_{m_n} - 6R_{m_n}^2)\prod_{k=1}^{n-1}(3R_{m_k}^2 - 2R_{m_k}^3) - \lambda\prod_{k=1}^{n-1}(R_{m_k}) = 0 \end{cases}$$

Par inspection, il peut être déterminé qu'une solution qui maximise la valeur de $R_{C_{1_{TMR}}}$ en considérant l'utilisation de 'n' modules est $R_{m_1} = R_{m_2} = \cdots = R_{m_n} = R_{C_1}^{1/n}$. C'est-à-dire, le circuit doit être partitionné en modules de fiabilités aussi identiques que possible. En utilisant ce résultat, l'équation (17) devient (19). De plus, ce résultat permet l'évaluation du nombre de modules 'n' qui doivent être utilisés pour qu'un niveau de fiabilité minimale R_{min} soit atteint.

$$R_{C_{1_{TMR}}} = (3R_{C_1}^{2/n} - 2R_{C_1}^{3/n})^n \tag{19}$$

La littérature ne reporte pas l'existence d'outils permettant la mise en œuvre automatique de la procédure de partitionnement d'un circuit TMR. Pour faire face à ce problème, nous proposons une méthode basée sur une idée très simple. D'abord, il faut visualiser l'évolution de la fiabilité du signal par rapport aux niveaux logiques comme illustré dans la FIGURE 14. Dans ce cas, un niveau logique est défini comme un nœud qui présente la capacité de devenir la frontière d'un module.

Si un arbitre est inséré dans un niveau aléatoire d'un circuit, une augmentation de la fiabilité est attendue dans ce même niveau (voir FIGURE 15). En fait, le gain en fiabilité est obtenu à cause de la capacité que possède l'arbitre de corriger des erreurs simples.

C'est bien connu que le gain en fiabilité obtenu par l'utilisation de la méthode TMR dépend de la fiabilité du module R_m comme illustré par (15). En même temps, la réduction de la fiabilité du signal illustrée dans la FIGURE 14 est aussi proportionnelle à R_m. Donc, la limitation de la réduction de la fiabilité du signal d'un circuit génère la création des

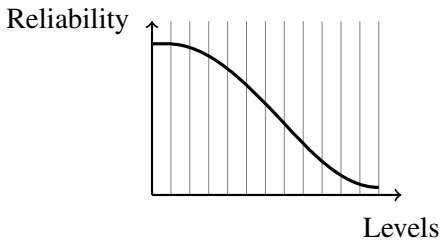


FIGURE 14 – Comportement de la fiabilité du signal d'un circuit par rapport aux nombre de niveaux

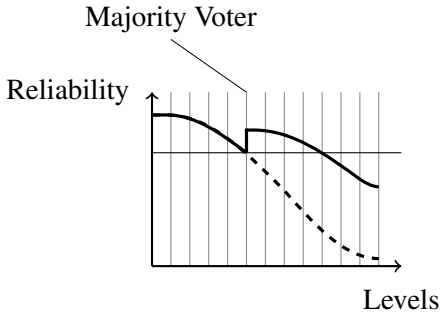


FIGURE 15 – L'insertion d'un arbitre dans un niveau aléatoire du circuit

blocs logiques de fiabilités aussi égales que possible. Cela est la première conclusion pour obtenir une solution optimale. Si ce seuil est défini par la fiabilité minimale du circuit, le résultat sera un nombre des modules 'n' très proche de la valeur optimale. Cette procédure est illustrée dans la FIGURE 16.

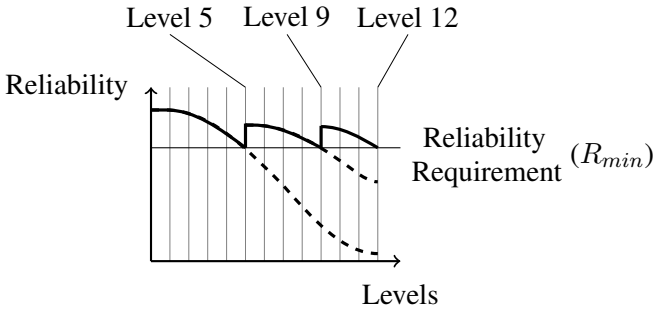


FIGURE 16 – Circuit TMR utilisant une distribution du processus de vote

Ce résultat peut être mieux analysé en considérant un circuit simple comprenant 10000 inverseurs logiques identiques (même fiabilité $R_m = 99.99\%$) connectés dans une structure en cascade (voir FIGURE 17).

La fiabilité totale du circuit R_c peut être évaluée à l'aide de l'outil SPR ($R_c = 56.7654\%$). En admettant que la contrainte de fiabilité minimale est $R_{min} = 99.9\%$, la

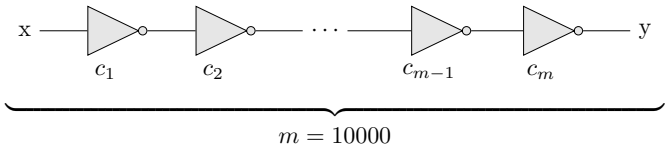


FIGURE 17 – Inverseurs logiques en cascade

méthode proposée insère 1000 arbitres. Tous les modules ont la même quantité de composants (N=10) sauf le premier (N=11) et le dernier (N=9). Il résulte de (19) que la quantité minimale de modules pour attendre le valeur de R_{min} est égal à 961. Cependant, cela voudrais dire que chaque partition devrait avoir 10.4058 inverseurs ce qui est impossible. La solution la plus proche possible sera l'utilisation de 10 inverseurs par module, ce qui correspond à la quantité de 1000 arbitres insérés, la même obtenue par la méthode proposée. La FIGURE 18 illustre le comportement de la méthode proposée en considérant plusieurs valeurs de R_{min}. Il peut être observé que les résultats sont proches des résultats optimaux pour tous les valeurs de R_{min} considérés.

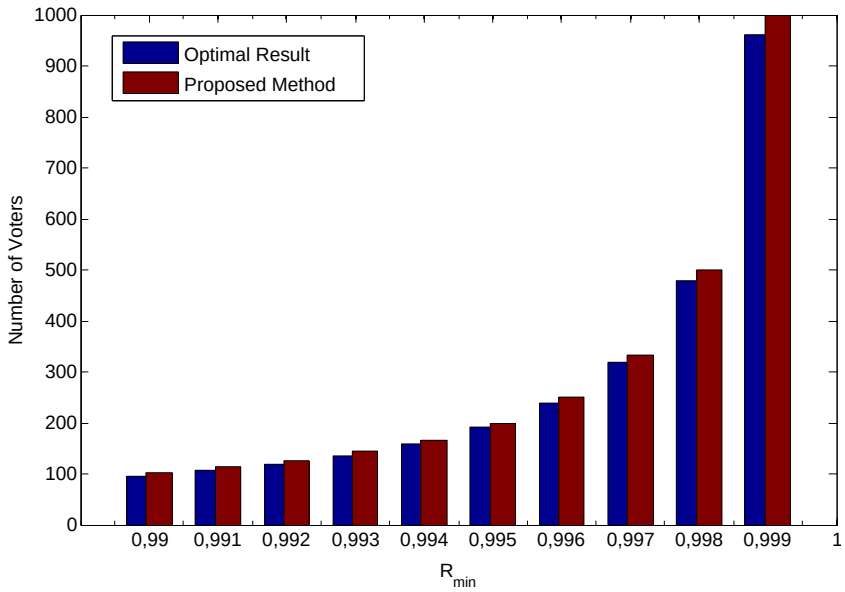


FIGURE 18 – Nombre d'arbitres insérés par la méthode proposée

Conclusion

Avec l'évolution technologique, la fiabilité joue un rôle de plus en plus important dans la conception des circuits intégrés. L'analyse de fiabilité doit être utilisée dans le flot de conception du circuit pour identifier le besoin d'utilisation des techniques de durcissement. Dans la littérature il y a plusieurs techniques d'analyse et de durcissement d'un circuit intégré. Cependant, ces téchinques présentent des limitations du fait de la complexité

d'analyse. En conséquence, le développement des nouvelles méthodes d'analyse aussi bien que des techniques de durcissement deviennent nécessaires. Dans cette thèse, plusieurs méthodes et outils d'analyse et durcissement ont été proposés.

Concernant l'analyse de fiabilité, la métrique mathématique «effective reliability» permet la prise en considération de la tolérance aux erreurs de l'application cible pour évaluer la fiabilité du circuit, alors que l'outil FIFA a été développé pour accélérer le processus d'analyse de fiabilité basée sur l'injection de fautes.

Dans le domaine de durcissement d'un circuit, deux techniques ont été développées pour identifier les portes logiques les plus critiques d'un circuit. Cela est une étape fondamentale pour l'application d'un processus de durcissement sélectif. Pour les applications qui ont besoin d'un niveau de fiabilité très élevé, un méthode automatique de partitionnement d'un circuit TMR a été élaborée. La performance optimale de cette méthode a été prouvée mathématiquement aussi bien que par simulation.

Symbols and Abbreviations

q	Gate reliability
$1 - q$	Gate unreliability
q_i	Reliability of bit i
λ, $\lambda(t)$	Failure rate
Cha_i	Hardening affinity parameter
$R(t)$, $\hat{R}(t)$	Reliability function
R_i	Reliability of bit i
$u(t)$	Step function
R_{ack}	Probability of errors being masked according to the application
R_{nom}	Nominal reliability
R_{eff}	Effective reliability
$R_{practical}$	Practical reliability
ALM	Adaptive logic module
ASMBL	Advanced silicon modular block
AUED	All-unidirectional error detecting code
BUED	Burst unidirectional error detecing code
CCC	Clock conditioning circuit
CD	Code distance
CLB	Configurable logic block
CMOS	Complementary metal-oxide-semiconductor
CMT	Clock management tile
DCM	Digital clock management
DMA	Direct memory access
DRAM	Dynamic random-access memory
DUT	Device under test

ECC	Error correcting code
EDC	Error detecting code
EDF	Électricité de France
EEPROM	Electrically erasable programmable read-only memory
EMI	Electromagnetic interference
EPROM	Erasable programmable read only memory
FA	Full Adder
FIT	Failures in time
FPGA	Field-programmable gate array
HALT	Highly accelerated life test
HD	Hamming distance
IEC, CEI	International electrotechnical commission
IP	Intellectual property
ITM	Ideal transfer matrix
ITRS	International technology roadmap for semiconductors
JEDEC	Joint electron devices engineering council
JTAG	Joint test action group
LAB	Logic array block
LE	Logic element
LET	Linear energy transfer
LSB	Least significant bit
LUT	Look-up table
MBU	Multiple-bit upset
MCU	Multiple-cell upset
MOSFET	Metal-oxide-semiconductor field-effect transistor
MSB	Most significant bit
MTBF	Mean-time-between-failures
MTTF	Mean-time-to-failure
NMR	N-modular redundancy
OTP	One time programmable
PBR	Probabilistic binomial reliability model
PIP	Programmable interconnect point
PLL	Phase-locked loop
PTM	Probabilistic transfer matrix
RAM	Random-access memory
RTL	Register transfer level
SBD	Soft breakdown

SEE	Single-event effect
SEFI	Single-event functional interrupt
SEL	Single-event latch-up
SER	Soft error rate
SET	Single-event transient
SEU	Single-event upset
SPICE	Simulation program with integrated circuit emphasis
SPR	Signal probability reliability model
SPR-MP	SPR multi-path model
SRAM	Static random-access memory
STMR	Selective triple modular redundancy
TMR	Triple modular redundancy
TSC	Totally self-checking
t-UED	t-unidirectional error detecting code
ULA	Ultra-low alpha
VHDL	Very high speed integrated circuit hardware description language
VLSI	Very-large-scale integration

Contents

Introduction **44**

1 Background on Reliability **51**

1.1 Introduction . . . 51

1.2 Reliability analysis . . . 51

1.2.1 Faults in VLSI circuits . . . 55

1.2.2 Reliability issues in FPGAs . . . 59

1.2.3 Prior works on reliability analysis . . . 62

1.3 Reliability improvement of integrated circuits . . . 68

1.3.1 Modular redundancy . . . 68

1.3.2 Voting strategies . . . 71

1.3.3 Selective Hardening . . . 72

2 FIFA Tool **75**

2.1 Introduction . . . 75

2.2 FIFA Tool . . . 75

2.2.1 FIFA Architecture . . . 77

2.2.2 Reliability Assessment . . . 81

2.2.3 Synthesis Results . . . 82

2.2.4 Parallelizing the FIFA Fault Generation . . . 84

2.2.5 Results . . . 88

2.2.6 Conclusion . . . 89

3 Effective Reliability **91**

3.1 Introduction . . . 91

3.2 Error tolerance . . . 92

3.3 Effective reliability . . . 94

3.4 Quality metrics . . . 95

3.4.1 Definitions . . . 95

3.4.2 Quality metric 1: bit significance . . . 96
3.4.3 Quality metric 2: relative error . . . 97
3.5 Simulation results . . . 98
3.5.1 Median filter . . . 98
3.5.2 8-bit ripple carry adder . . . 100
3.5.3 4-bit multiplier . . . 102
3.6 Conclusion . . . 104

4 Selective Hardening **107**
4.1 Introduction . . . 107
4.2 Avoiding Critical Errors in Integrated Circuits . . . 107
4.2.1 Nominal reliability . . . 107
4.2.2 Practical reliability . . . 108
4.2.3 Selectively applying TMR . . . 109
4.3 Using a Cost Function to Detect Critical Gates . . . 114
4.3.1 Cost function profiling . . . 116
4.3.2 Experimental results . . . 119
4.3.3 Comparison with related works . . . 120
4.4 Conclusion . . . 121

5 Optimizing Voter Placement for TMR Systems **123**
5.1 Introduction . . . 123
5.2 TMR approach . . . 123
5.3 Partitioning a TMR design . . . 125
5.4 Problem of automatically inserting voters . . . 128
5.5 Proposed method . . . 129
5.6 Conclusion . . . 134

6 Concluding Remarks **135**

A Other Methods for Reliability Improvement of ICs **139**
A.1 Fault detection and correction . . . 139
A.1.1 Basic principles . . . 139
A.1.2 Fault detection techniques . . . 140
A.2 Evolvable hardware . . . 149

B Basics on FPGAs **153**
B.1 FPGA technologies . . . 153
B.1.1 Fusible link technology . . . 153

B.1.2 Antifuse 154
B.1.3 Static memory technology 156
B.1.4 Flash technology 157
B.1.5 Summary 159
B.2 FPGAs architectures 159
B.2.1 Altera 159
B.2.2 Xilinx 162
B.2.3 Actel 165
B.2.4 Lattice 167

Conclusion **169**

Glossary **171**

Notations **171**

Bibliography **184**

List of Figures

1.1 Number of functioning parts of a circuit at time t . . . 53
1.2 Bathtub curve . . . 54
1.3 Residue induced intermittent fault in a DRAM chip . . . 57
1.4 Effects of a high-energy ion hitting a semiconductor device . . . 58
1.5 Fault simulation approach proposed by Ogus . . . 63
1.6 PTM representation for an AND gate . . . 64
1.7 Basic interconnection models of PTM . . . 64
1.8 SPR matrix for the output of a 2-input OR gate . . . 65
1.9 Propagation of the SPR matrices through a circuit . . . 65
1.10 TMR concept envisaged by Von Neumann . . . 69
1.11 TMR performance regarding reliability improvement . . . 69
1.12 TMR with three majority voters . . . 70
1.13 Word-Voter proposed in [21] . . . 71

2.1 General scheme of the proposed tool . . . 78
2.2 General scheme of a saboteur . . . 78
2.3 FIFA Timing diagram of communication signals . . . 80
2.4 Example of a step by step execution of Algorithm 1 . . . 81
2.5 Synthesis results of the FIFA tool . . . 83
2.6 A simple parallel architecture for FIFA . . . 85
2.7 The proposed parallel architecture for FIFA . . . 85
2.8 Example of a fault pattern generation sequence . . . 86
2.9 Pascal's triangle . . . 87
2.10 Total logic elements . . . 89
2.11 Number of clock cycles . . . 89

3.1 Example of a logical masking . . . 92
3.2 Bit-flip occurrence in a sine wave . . . 93
3.3 General schema for reliability calculation . . . 95

3.4 Example of matrix **E** considering 3 errors . . . 96
3.5 Comparison between the original and the noisy "Lena" pictures . . . 100
3.6 Structure of FA (full adder) block . . . 101
3.7 Schema of FA block . . . 101
3.8 Structure of a 4-bit multiplier block . . . 103

4.1 4-bit fast adder circuit . . . 110
4.2 74283 gate-level schematic . . . 111
4.3 Simulation results for the 74283 circuit . . . 114
4.4 Cost function profile for the circuit *c432* . . . 117
4.5 Cost function profile for the circuit *c499* . . . 118
4.6 Both heuristics applied to the circuit *c1355* . . . 118

5.1 TMR block scheme . . . 123
5.2 Reliability gain using TMR . . . 124
5.3 Partitioning a TMR design . . . 125
5.4 C_1 comprises n modules serially interconnected . . . 126
5.5 Plot of $R_{m_1} = \frac{R_{C_1}}{R_{m_2}}$ for different values of R_{C_1} . . . 127
5.6 Reliability of a circuit versus its number of levels . . . 129
5.7 Insertion of a majority voter . . . 130
5.8 Distributing the voting process of a TMR circuit . . . 130
5.9 Cascade of inverters . . . 131
5.10 Number of voters inserted by the proposed technique . . . 131
5.11 Circuit 74283 - Gate level . . . 132

A.1 Example of a duplex comparison scheme . . . 139
A.2 Computer memory using parity checking . . . 141

B.1 Programmable circuit concept . . . 154
B.2 Programmable circuit with intact fusible links . . . 154
B.3 Programmed circuit with output $Y = A + \bar{B}$ . . . 154
B.4 Programmable circuit with intact antifuses . . . 155
B.5 Programmed circuit with output $Y = A + \bar{B}$ . . . 155
B.6 Static Memory Cell . . . 157
B.7 Flash memory cell - ProASIC3 . . . 158
B.8 ALM High-Level Block Diagram . . . 160
B.9 LE Block Diagram . . . 161
B.10 Example of devices using the ASMBL architecture . . . 163
B.11 High-level block diagram of a CLB in Spartan FPGAs . . . 164

B.12 AX C-Cell and R-Cell 166
B.13 AX SuperCluster Arrangement 167
B.14 PFU block diagram 168

List of Tables

1.1 Programming technology properties summary 60

2.1 Fuse platform vs. FIFA tool 84

3.1 Effective reliability evaluation for different error tolerances 99
3.2 Reliability values for the output bits of a full adder 101
3.3 R_{eff} for different error tolerances (CRA8) 102
3.4 Reliability values for each output bit of the 4-bit multiplier 103
3.5 R_{eff} for different error tolerances (MUL4) 104

4.1 Reliability values of three different architectures of an adder 108
4.2 Error analysis for the gates of the circuit 74283 112
4.3 Reliability Analysis of 74283 113
4.4 Hardware affinity (Cha_i) parameters for some cells 116
4.5 Results for the sum of elements heuristic, $K = 10$ 119
4.6 Results for the percent wise heuristic, $X = 50\%$ 120

5.1 Placement of the voters for the circuit 74283 133
5.2 Reliability of gates based on their area 133
5.3 Placement of the voters for the 74283 circuit 134

A.1 3-bit Berger code – B_0 scheme 142
A.2 Bose code for data words comprising 4 bits 143
A.3 Syndrome Table for Hamming (7,4) code 147

B.1 Programming technology properties summary 159
B.2 Altera Devices Comparison 162
B.3 Xilinx Devices Comparison 164
B.4 Actel Devices Comparison 167
B.5 Lattice Devices Comparison 169

Introduction

The first electronic computer was built in Antanasoff's Iowa State College in 1942 [22] and used rather unreliable components. Improve the system reliability was a major concern and techniques such as duplexing with comparison, triplication with voting, control codes, among others, were proposed. Indeed, important researches were done by J. Von Neumann, E. F. Moore and C. E. Shannon using redundancy as a mean to build reliable systems from less reliable components [23, 24].

Since then, the integrated circuit technology has underwent an exponential evolution as predicted by the *Moore's law* [1, 2]. Nowadays devices are shrinking into the deca-nanometer range, allowing the fabrication of chips containing billions of transistors, and operating at very high speeds (multiple GHz). In such scale, new physical phenomena, such as Van Der Waals and Casimir forces, appear leading to new fabrication methodologies and affecting the components reliability [3]. Further, interconnect systems are becoming very complex, particularly with the introduction of the 3-D die integration scheme [4]. In fact, the higher density of integrated circuits together with the higher complexity of the interconnections lead to a higher probability of erroneous components in a die. Meanwhile, the higher frequencies pose strict limits to timing, thus also increasing the probability of timing errors [5]. In other words, a reduction in manufacturing yield is expected, as well as in the overall circuit reliability [6–10].

Consequently, faults have become more and more likely to occur in deep-submicron technologies. Permanent faults can be significantly reduced by performing deep investigations during offline testing [25]. On contrary to that, transient faults depend on environmental conditions, and therefore they randomly occur during circuit operation. In the past, these faults used to be a concern only on the design of memories. However, the technology scaling has increased the susceptibility of combinational blocks to thermal bit-flips, radiation events, among others, so that their resulting error rates are approaching those of memories [26, 27]. This is a serious menace to circuits designed to operate under critical environments such as nuclear power plants, avionics, among others, and therefore solutions to construct fault-tolerant circuits are necessary.

This dissertation is a result of a CIFRE partnership between EDF R&D and Télécom

Paristech. The motivations for this work can be explained as follows. EDF is one of the world's largest producers of electricity with main activity in nuclear power. EDF's control-command systems are based on electronics devices/circuits. Nuclear power plants consider safety as a very high priority in their systems, and electronic circuits must be in accordance to several technical standards such as the IEC 62566 [11], the IEC 60987 [12], and the IEC 61513 [13] in order to be qualified to operate in such critical environments. Particularly, the use of programmable devices poses a great challenge to be qualified since the functionality of the IC is not defined by the founder of the physical component, but by the designer of the application. Further, the technical standard IEC 62566 states that the benefits accomplished by the use of redundancy in an electronic circuit must be balanced with the corresponding increase in the system complexity as well as in the fault coverage. Because of that, methods to analyze and to improve the reliability of electronic circuits to be used in nuclear power plants are a major concern.

The main objective of the current work is to propose methods to analyze and to improve the reliability of circuits in order to facilitate their qualification according to the aforementioned technical standards. Therefore, different strategies that allows both to achieve a very high level of reliability in a circuit and to control the amount of redundancy adding are required. These methods must be developed in such a way that they can be used to construct circuits using programmable devices as well as for circuits tailored for a specific function, e.g. ASICs. Also, the proposed solutions must be able to be appended to traditional design flows of integrated circuits.

A fault-tolerant integrated circuit is generally obtained by the properly use of redundancy, whether it be temporal or spatial. The addition of redundancy, however, directly affects some attributes of the circuit, such as performance and surface, thus increasing the overall system cost and complexity. Because of that, the choice of which fault tolerant approach to use for a given scenario involves a multi-criteria optimization problem, taking into account all the specified design constraints. Therefore, whether a circuit is intended to mission critical applications or in the case reliability can be relaxed in order to avoid the increase in the complexity of the circuit, reliability analysis plays a crucial role in its design flow. Methodologies to assess the reliability of circuits have been extensively researched over the last years. As a matter of fact, an optimal reliability analysis lies on the use of as much information as possible about the circuit itself as well as about the target application. However, most of the existing techniques assume simplifications on mathematical models such as single faults, uncorrelated signals, among others. Further, not much has been done in order to consider the usage profile of the circuits' results when calculating its reliability. In spite of that, many applications exhibit the ability to tolerate some kinds of errors. By considering such ability, a designer can obtain more accurate results, which can

avoid unnecessary over costs.

Based on that fact, the current work proposes a technique to cope with such problem. The proposed technique, named *effective reliability*, can take into account the masking effect provided by the target application in order to evaluate its reliability. This technique works alongside quality metrics such that it is possible to differentiate critical from non-critical errors. In this case, an error is said to be non-critical if it can be tolerated by the target application. Two possible quality metrics are also proposed in the current work.

Effective reliability is of great use for applications in which some errors can be tolerated. However, mission critical applications demand a high-degree of confidence, and they have low or no interest in accepting any kind of error. Indeed, these circuits usually require deep investigations to predict its behavior considering the occurrence of faults, so that the reliability of its results can be asserted. Generally, such circuits demand a test coverage approaching 100%, and therefore methods to accelerate exhaustive testing are necessary.

In order to cope with this problem, this work also proposes a Verilog-based platform to exhaustively analyze the behavior of a logical circuit considering the occurrence of faults. The proposed platform, named FIFA (Fault-Injection-Fault-Analysis), is based on the Probabilistic Binomial Reliability model (PBR), which can evaluate the reliability of a circuit based on its logical masking ability. One of the great advantages of the FIFA platform is that it is easily customized. Further, it supports several fault models as well as the ability to inject single and multiple simultaneous faults. It is important to highlight that the fault pattern generation as well as the fault injection can be performed without the need of any device reconfiguration. Also, the proposed platform allows the evaluation of approximated reliability values by considering a maximum number of simultaneous faults to be injected. In order to allow that, the platform generates the fault patterns in an ascending order regarding the number of simultaneous faults. The flip side of the coin is that such fault pattern generation sequence imposes strict difficulties to be parallelized. In order to address this problem, the current work also proposes a solution based on number patterns to elaborate a parallel design for the FIFA platform, which can significantly reduce the required computing time.

Although reliability analysis plays an important role during the design phase of an integrated circuit, methods to improve its reliability are more and more desired in deep-submicron technologies. Partial fault tolerant designs are usually enough for some applications, and therefore methods based on selective hardening are very suitable.

Based on that fact, this work proposes two methods to identify the critical gates of a circuit in order to apply selective hardening. The first one is based on the criticality of the output bits regarding the usage profile of the results. In other words, it drives the reliability improvement effort to better protect the output bits that are considered more critical to

the target application. By doing that, the proposed methodology can automatically select a set of gates, based on an area overhead constraint, such that the probability of occurrence of critical errors is minimized. The second one uses a parameter similar to a hardening cost in order to drive the methodology using accurate cost values for hardening each gate. In addition, two heuristics are introduced as a means to determine when selective hardening is no longer feasible.

Although partial fault tolerance techniques are good solutions for some applications, this is not the case for mission critical ones. Indeed, most of the time such applications demand the most reliable system possible. In such context, Triple Modular Redundancy (TMR) is a fault-tolerant technique often used despite its huge area overhead. This is because TMR has proven to be a very simple, effective solution to the correction of single faults. Further, several tools were developed in order to automatically apply TMR to a circuit, which simplifies the whole process of circuit hardening. However, as the dimensions of integrated circuits continue to shrink, the probability of occurrence of MBUs increases as well. Therefore, methods that can deal with multiple simultaneous faults are highly desired.

One possible solution to that is the use of other modular redundancy techniques such as 5MR, 7MR, etc., but the area overhead is generally prohibitive. On the other hand, the heart of TMR is the majority voter block, responsible to mask the faults occurring in the circuit. Indeed, this block can correct any single fault or detect any double faults occurring in the circuit. Delegate the majority decision to several modules across the circuit has been proved to be a great cost-effective solution to correct multiple simultaneous faults. This technique, known as partitioned TMR, can increase the fault tolerance of a traditional TMR system by slightly increasing the corresponding area overhead. However, determine the number of majority voters to be used and their corresponding placements are not trivial tasks. Indeed, the voter insertion process directly affects the timing performance, the area, and the reliability of the obtained circuit. This problem is yet more complicated in case of FPGAs because certain nets are not allowed to be cut by voters, or this is not desirable.

Finally, the current work proposes an algorithm to tackle this problem. Given a reliability requirement, the proposed algorithm can automatically detect the best amount of voters as well as their placements in order to partition a TMR design. Further, the reliability gains achieved by the proposed method approach those obtained with an optimal TMR partitioning. Indeed, by using this method, only the first and last partitions do not always have optimal sizes. Last but not least, the method was developed in order to be applied in both VLSI and FPGA circuits by analyzing gate and primitive netlists, respectively.

The current work is organized as follows. First, some basics on reliability analysis and reliability improvement techniques are presented in Chapter 1. Next, Chapter 2 introduces

a fault-injection tool developed during this thesis as a means to analyze and validate a fault tolerant design. Chapter 3 introduces the concept of *effective reliability* of an integrated circuit. Chapter 4 presents two techniques to selectively harden a circuit. The first is based on the usage results of the output, while the latter uses heuristics and a hardening cost function in order to automatically select the best candidates to be protected. Chapter 5 presents a technique to automatically insert partitioning voters into a TMR design. Finally, a review of other methods existent in the literature to improve the reliability of a circuit is presented in Appendix A, and a review of some popular technologies and architectures of FPGAs is available in Appendix B.

Chapter 1

Background on Reliability

1.1 Introduction

Until the sixties, the consumer expectation when buying something was to receive a product that performed well its functionality at the time it left the manufacturer. This expectation evolved over the years, and the product bought today must perform the required function free of failures for a specified period of time [28]. This brings the concept of reliability of a product, which can be defined as the probability of a given item to perform its required function under stated conditions for a stated time interval [14].

Since then, reliability has become a very important attribute for most of industrial products. In case of electronic circuits, reliability is a main consideration when designing nanoscale devices. Indeed, factors associated to technology scaling such as manufacturing precision limitations, devices parametric variations, supply voltage reduction, among others, are increasing the likelihood of faults in electronic circuits, thus decreasing their reliabilities. Therefore, methods that can analyze the reliability of a circuit in order to provide feedback for the elaboration of robust designs are highly desired.

This chapter presents some basics on reliability. First, Section 1.2 introduces some important concepts on reliability prediction. Next, techniques to improve the reliability of a circuit based on modular redundancy are discussed in Section 1.3.

1.2 Reliability analysis

An electronic circuit is composed of a set of electronic components interconnected in such a way that a given functionality, described by a circuit specification, is provided. Assuming that a circuit is well specified and that it contains only fault-free elements, the desired functionality can be always guaranteed. However, electronic devices are susceptible to some natural and human-made mechanisms, e.g. impurities in materials, device param-

eters variations, errors in the specification of a circuit, among others, which can affect the state of individual electronic components. These mechanisms, here called *faults*, are of great importance for reliability engineering because they are responsible for the occurrence of *errors* in electronic circuits. However, not every error will succeed to propagate to the output of a circuit and then affect the final results. As a matter of fact, the propagation of errors depends on the interaction among the electronic components of a design. If an error succeeds to reach the final output of a circuit, it will lead to the occurrence of results that are not in accordance with the circuit specification, also know as *failures*. Otherwise, it is said that the error was *masked*. This abnormal behavior is becoming more and more likely to happen with the downscaling of electronics. Because of that, reliability analysis has become an important step on the design flow of integrated circuits.

Reliability analysis can be performed in several phases of the circuit development. However, the most effective way is to perform the analysis while still on its design phase [29,30]. This is because the correction of a design can be performed before physically fabricating it, thus reducing the time-to-market and the cost of the circuit in case of the validation process fails. In order to do that, it is important to deeply understand the possible causes of failures, so that they can be anticipated and prevented. Therefore, one of the most common forms of reliability analysis is the reliability prediction. It refers to the estimation of the failure rate of electronic components and of the overall system. This prediction contributes to define the initial, maintenance and total system costs, for example. By predicting the reliability of a circuit, designers can evaluate the feasibility of a given design, revealing potential failure areas and the need for environmental control systems. They can also determine the need of redundant systems, back-up systems, among others. Further, reliability analysis can be used as a tool to compare the performance of different fault-tolerant strategies, measuring the reliability improvement achieved by using a given technique and the corresponding overhead in terms of area and/or timing [31].

In order to estimate the failure rate of a given circuit, let us first assume that it is composed of n statistical identical and independent parts that were put into operation at time $t = 0$. Then, the number of parts of this circuit that did not yet fail at time t can be represented by a continuous decreasing step function $u(t)$ as shown in Figure 1.1. Based on this curve, the empirical reliability of a circuit can be obtained by (1.1). A direct application of the law of large numbers ($n \to \infty$) yields that $\hat{R}(t)$ converges to the reliability function $R(t)$ [28].

$$\hat{R}(t) = \frac{u(t)}{n} \tag{1.1}$$

Let us now define $\hat{\lambda}(t)$ as the empirical failure rate given by (1.2). Then, it can be shown

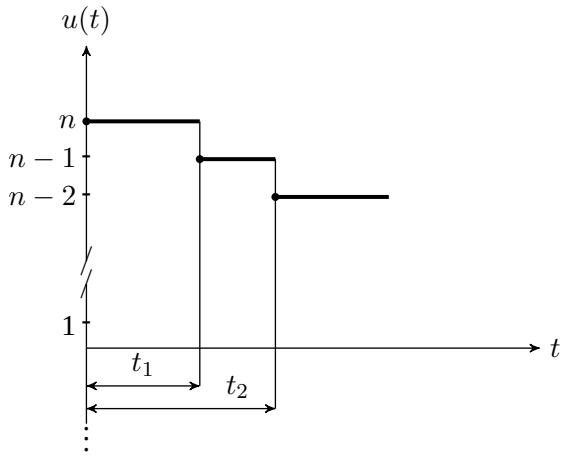


Figure 1.1: Number of parts of a circuit that not yet failed at time t

that this equation converges to the failure rate expressed in (1.3) for $n \to \infty$, $\delta_t \to 0$ and $n\delta_t \to 0$ [28].

$$\hat{\lambda}(t) = \frac{u(t) - u(t+\delta_t)}{u(t)\delta_t}$$
$$\hat{\lambda}(t) = \frac{\hat{R}(t) - \hat{R}(t+\delta_t)}{\delta_t \hat{R}(t)} \tag{1.2}$$

$$\lambda(t) = \frac{-\frac{dR(t)}{dt}}{R(t)} \tag{1.3}$$

Considering that the circuit operates perfectly at time $t = 0$, that is $R(0) = 1$, the reliability function can be expressed as shown in (1.4).

$$R(t) = e^{-\int_0^t \lambda(x)dx} \tag{1.4}$$

Equation (1.4) shows that the reliability function depends on the behavior of the failure rate $\lambda(t)$, which has a typical shape as represented in Figure 1.2. Due to its shape, this curve is denominated *bathtub curve*, and it can be split into three different regions:

- *Decreasing Failure Rate*: corresponds to the failures that occur when the circuit is first introduced as a result of momentary weakness in materials or in the production process. During this period, $\lambda(t)$ can also oscillate [28]. In order to reduce the infant mortality occurring during this phase, manufacturers use stress tests (often called burn-in) to accelerate the aging of the devices in such a way that they can reach their useful life before going to the market.
- *Constant Failure Rate*: during this period, $\lambda(t)$ can be approximated by a constant.

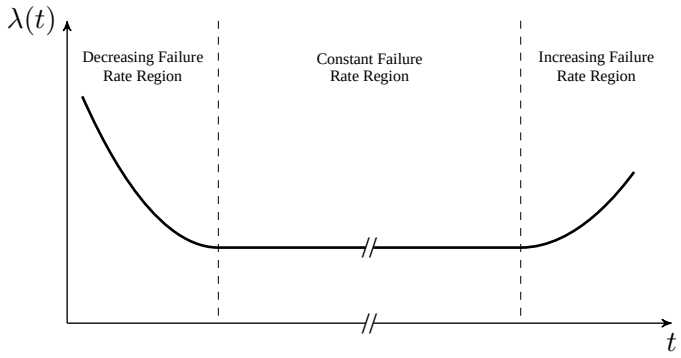


Figure 1.2: Bathtub curve representing the typical shape of the failure rate of a circuit

This region corresponds to the useful life of the circuit.

– *Increasing Failure Rate*: this part represents the end of the circuit's lifetime due to wear out and aging.

The failure rate of an integrated circuit is often expressed in terms of *failures in time* (FIT), where 1 FIT means 1 failure in 10^9 device hours. Considering that a circuit is composed of k different components, the failure rate is expressed as shown in (1.5), where N_k stands for the number of components of type k. Notice that manufacturers generally use highly accelerated life tests (HALT) in order to estimate the failure rate of their integrated circuits. More details about HALT procedures can be seen in [32], in the JEDEC JESD74A [33], and in the MIL-STD-883H [34].

$$\lambda_{circuit} = \sum_{1}^{k} N_k \lambda_k \tag{1.5}$$

Since the failure rate of a circuit is constant during its useful life, the reliability expression (1.4) becomes (1.6). It can be seen that the reliability of a circuit is a measure that depends on the time of the circuit operation, which is not very practical. Because of that, another useful metric is also available to analyze the behavior of integrated circuits considering the occurrence of failures. This metric, called *Mean-Time-Between-Failures* (MTBF), can be evaluated by expression (1.7). Notice that MTBF is used to systems that are repaired after the occurrence of a failure. In case a circuit is replaced after a failure, the metric *Mean-Time-To-Failure* (MTTF) is often used instead of MTBF. More details about metrics to analyze the reliability of a system can be seen in the book of David J. Smith [35].

$$R(t) = e^{-\lambda t} \tag{1.6}$$

$$MTBF = \int_0^\infty R(t)dt = \int_0^\infty e^{-\lambda t}dt$$
$$MTBF = \frac{1}{\lambda} \tag{1.7}$$

As a matter of fact, a number of reliability analysis methods are available in the literature. Basically, these methods are used to analyze either the *functional reliability*, which is the probability that a given circuit will perform its specified function, or the *signal reliability*, which stands for the probability that the output data is correct [8]. The current work considers the latter analysis, which generally takes into account the logical masking ability of a design.

Reliability analysis plays an important role in the design process of a circuit. In order to develop a product with a stated reliability requirement, appropriate investigations of failure rate and failure mode must be done. The results produced by these investigations lead to the evaluation of the reliability of the product. However, due to uncertainties such as simplifications in mathematical modeling, inaccuracies in the investigations of the failure rate, among others, these results present a limited precision [28]. Moreover, an investigation of the required functionality, the types of faults that are likely to occur, and the environmental conditions in which the circuit will perform its task, should be carried out. Indeed, in order to design high-reliable systems, we should consider as many aspects as possible during the reliability analysis phase. Then, let us start by reviewing the types of faults that affect VLSI systems in Section 1.2.1.

1.2.1 Faults in VLSI circuits

The reliability of a VLSI circuit is related to its capacity to correctly operate considering the occurrence of faults [8]. Regarding their persistence, these faults can be classified into three categories:

– *Permanent Faults*: represent irreversible physical changes in the device, which permanently affect the specified logic function. They generally occur due to imperfections on the design process, and therefore they can be significantly reduced during offline testing [25]. However, permanent faults may also appear during the useful life of a circuit due to different reasons such as aging and wear out, for example. In this case, they are generally preceded by the occurrence of intermittent faults [36].

- *Intermittent Faults*: manifest themselves as random physical changes, caused generally by unstable or marginal hardware [37]. For example, due to minor changes in temperature, vibrations, among others, a borderline electrical connection may become an intermittent connection.
- *Transient Faults*: generally caused by environmental conditions such as electromagnetic interference (EMI) and ionizing radiation. Because of that, they randomly occur during circuit operation.

It is important to notice that intermittent and transient faults manifest themselves very similarly. However, as stated above, intermittent faults reflect the existence of unstable or marginal hardware, and therefore they tend to occur in bursts and at the same location. Besides that, intermittent faults can be mitigated by the repair of the faulty circuit [37].

The effects of intermittent faults on the reliability of integrated circuits was deeply analyzed by Constatinescu in [37–40]. In such works, he stated that several phenomena are capable to produce intermittent faults in deep-submicron technologies. For instance, due to the reduction in dimensions of integrated circuits, electromigration may increase the resistance of narrower sections in the devices, thus leading to the occurrence of delay faults. Besides, if in the past larger transistors could handle small amounts of manufacturing residues, this is not anymore true in deep-submicron technologies. Indeed, these small quantities of residues may now lead to the occurrence of intermittent contacts. For example, by performing a series of experiments in data servers in [40], Constatinescu have noticed a memory exhibiting such problem (see Figure 1.3). Also, due to soft breakdown (SBD) effects present in ultrathin gate oxides, fluctuating current leakages are expected to increase and may exhibit the same characteristics of intermittent faults [37]. In other words, the work of Constatinescu have shown that the rate of intermittent faults in electronics circuits tend to increase with the downscaling of electronics. However, he has also shown that techniques developed to mitigate transient faults can also reduce the number of intermittent faults [40].

Transient faults, also known as *soft errors*, are a major concern for the design of electronic circuits because of their random nature. They are responsible for one of the highest error rates in electronic circuits. Because of that, the current work is focused on the reliability of VLSI circuits to soft errors, with a special attention to the susceptibility of FPGAs to this kind of errors.

Transient faults are caused by several different physical phenomena such as alpha particles, cosmic rays, interconnect noise, electromagnetic interference, among others. For instance, the reduction of the supply voltage coupled with a higher VLSI integration have led to a great increase on the susceptibility of integrated circuits to energetic particles. In past technologies, these errors used to be a concern only in dense radiation environments

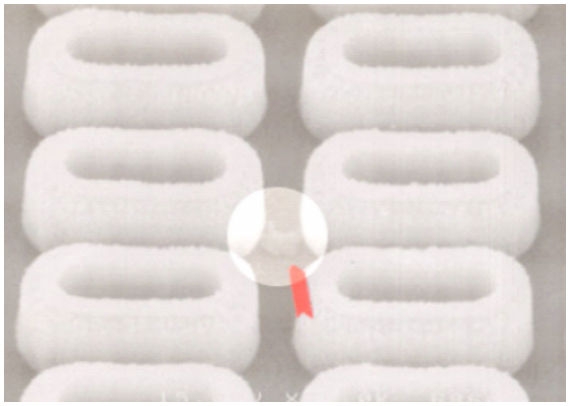


Figure 1.3: Residue induced intermittent fault in a DRAM chip [40]

such as space. However, newer technologies are making the devices susceptible to such particles even at ground level. Indeed, radiation-induced soft errors have the potential to become the most severe cause of failures in electronic devices if not mitigated [41]. In order to deal with such threat, it's important to understand how electronic circuits behave in the presence of such particles.

1.2.1.1 Sources of ionizing radiation

Radiation particles that can cause soft errors in electronic devices are mainly generated by two different mechanisms at the terrestrial environment: alpha particles and cosmic rays. When one of such energetic particles hits a semiconductor device, specially if near the reverse-biased junction (the most sensitive part of a circuit to radiation particles), it interacts with the electrons in the material during its passage until it loses all of its kinetic energy. The result is the appearance of a cylindrical track of electron holes, with a very high carrier concentration. The higher the energy of the particle, the longer is the distance it travels. Such phenomenon is illustrated in Figure 1.4(a). When this ionization track is close to the depletion region, the carriers are fast collected by the electric field, thus generating a high current/voltage transient at the corresponding node, which can persist approximately one nanosecond (see Figure 1.4(b)) [41]. Next, a phase where the charges are collected by diffusion begins, generating a low-current pulse as shown in Figure 1.4(c). If the collected charge exceeds the critical charge, it can cause the well known single-event effect (SEE).

Single-event effects can take many forms depending on the magnitude of the disturbance generated by the hitting particle, which relies on its linear energy transfer (LET), and on which component of the circuit it occurred. If the SEE generates enough of charge disturbance that the state of a bit in a register, flip-flop, latch, or memory cell, is flipped, it causes a single-event upset (SEU). Since nodes of circuits are close to each other, an

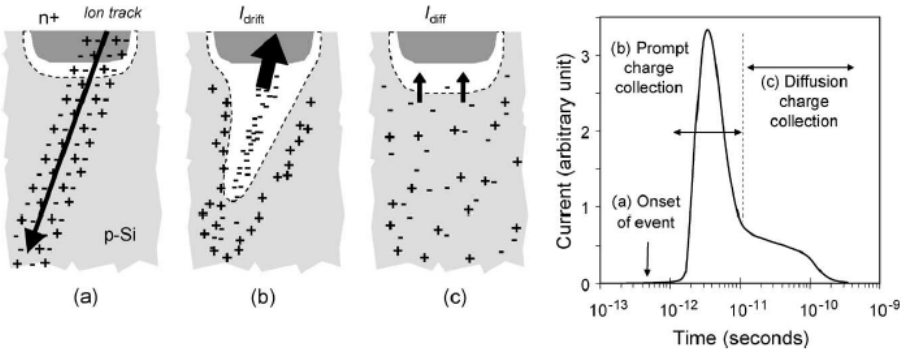


Figure 1.4: Effects of a high-energy ion hitting a semiconductor device [41]. (a) particle hitting the device - (b) charges being rapidly collected by the electric field - (c) charge collection by diffusion

SEE may propagate through several paths, thus sharing the charge effect of the particle among different nodes. If the corresponding particle is of very-high energy, the state of several bits in a circuit may be reversed, thus causing a multiple-cell upset (MCU). If these faulty bits are located in the same word, it is called a multiple-bit upset (MBU). SEEs occurring in combinational logic generates a single-event transient (SET), which can propagate through the logic and, if latched by a memory element, will become an SEU. Other kinds of soft errors may still occur due to single-event effects. If critical bits of a system, such as those of the configuration memory of an FPGA device are affected by an SEE, it may directly lead the device to malfunction. In this case, the error is called a single-event functional interrupt (SEFI). Last but not least, an SEE may turn on the CMOS parasitic bipolar transistors between well and substrate, thus generating a single-event latch-up (SEL). With so many threats generated by a radiation event, let us analyze how the main mechanisms responsible for generating ionizing particles behave at ground level.

Alpha particles used to be the main cause of radiation-induced soft errors in silicon devices in late 1970s. In fact, impurities presented in the package materials have the potential to emit a high rate of alpha particles, which can produce a high number of soft errors. However, with the improvement of the fabrication process of electronic circuits, materials can be highly purified and the rate of alpha particles emissions can be well controlled. This does not mean that the problem of alpha particles is completely solved. This problem was significantly reduced in current technologies, but it still plays a special role in VLSI and FPGA reliability [42]. For the sake of comparison, the rate of alpha particles emission went from a level of $100\alpha/cm^2/h$ in older technologies to levels below $0.001\alpha/cm^2/h$ in current technologies [41]. Indeed, devices that have a rate of alpha particles emissions below $0.002\alpha/cm^2/h$ are said to be ultra low alpha (ULA). Normally

direct alpha counting techniques must be employed in order to guarantee that an electronic device is ULA. In this context, one of the main challenges in future technologies regarding alpha particles is to verify if all materials reaches or exceeds the ULA grade [41].

Cosmic rays are particles generated by interactions of galactic cosmic rays with the Earth's atmosphere. As a matter of fact, the Earth's atmosphere is constantly hit by high-energy particles originated from galactic cosmic rays. The flux of such particles is modulated by some mechanisms such as solar wind and the earth's magnetic field. This generates a flux that depends on the latitude, longitude, altitude and solar activity of a location. Neutrons are one of the main resulting particles of such interactions, and since they can have a high amount of energy, they are the most likely cause of radiation-induced soft errors due to cosmic rays. In order to be aware of the intensity of the neutron flux in a given city/location, the Joint Electron Device Engineering Council (JEDEC) developed some models based on the actual flux occurring at sea level in New York City (JEDEC Standard 89A). It is important to notice that neutrons cannot directly generate ionization in silicon, but they can react with chip materials so that such phenomenon is generated. In fact, neutrons hits produce a series of elastic and inelastic reactions so that a burst of smaller particles are created. The higher the energy of the neutron, the higher is the probability of occurrence of high-energy bursts. And since the energy (LET) of such bursts are significantly higher than that of alpha particles, neutrons exhibit a higher probability to cause an SEU. Indeed, the occurrence of MCU and SEL are mainly due to high-energy neutron effects [41,43]. Another interesting, not to say challenging, characteristic of cosmic neutron flux is that they cannot be significantly reduced at the chip level by the use of shielding, keep-out zones, or high purity materials [41]. For instance, concrete has proved to reduce the high-energy portion (E>10 MeV) of the cosmic-ray neutron spectrum at a rate of 2.3× per foot of concrete thickness, while the total neutron flux is reduced at a rate of 1.6× [44]. Therefore, although the Soft Error Rate (SER) generated by cosmic rays can, for example, be reduced in a nuclear plant surrounded by many feet of concrete, for domestic use very little can be done. Because of that, the use of design hardening is an attractive solution against soft errors caused by cosmic rays.

Soft errors are yet more challenging in FPGAs devices because of their memory cells. Section 1.2.2 deals with radiation-induced soft errors in FPGAs devices.

1.2.2 Reliability issues in FPGAs

Field Programmable Gate Arrays (FPGAs) are integrated circuits very flexible in the context that they can be customized after manufacturing. They are composed of programmable logic blocks and interconnects that can be configured to implement basically any kind of digital logic, and programmable input/output blocks which allow the configu-

ration of most of industrial communication standards. Further, circuits can be described by using general hardware languages such as Verilog and VHDL. By doing that, a given circuit can be easily used in different projects and FPGAs, thus increasing the flexibility of such devices.

Several technologies are available to construct the programmable blocks of FPGAs. Basically, they can be classified into two types: One Time Programmable (OTP) and Reprogrammable. The programmable interconnections available in OTP devices operate in such a way that, once programmed, they are physically wired and therefore cannot be changed anymore. However, in case of reprogrammable devices, the configurable interconnections are made of memory elements which enables the reprogrammable capability. Then, reprogrammable devices contains two types of memory:

- *User memory*: responsible to keep the data required for the application.
- *Configuration memory*: responsible for defining the configuration logic (interconnections and logic functions) of the implemented circuit.

The most important technologies used in FPGAs are: SRAM, Antifuse, and Flash. Table 1.1 summarizes the major characteristics of such technologies. It is important to note that an ideal technology would be a nonvolatile, reprogrammable, providing low resistance and parasitic capacitances, and using a standard CMOS process. None of the existent technologies can satisfy such requirements.

Table 1.1: Programming technology properties summary [45]

	SRAM	**Flash**	**Antifuse**
Volatile	Yes	No	No
Reprogrammable	Yes	Yes	No
Storage Element Size	High	Moderate	Low
Manufacturing Process	Standard CMOS	Flash Process	Special Antifuse Process
In-System Programmable	Yes	Yes	No
Switch Resistance	$\sim$500 – 1000Ω	$\sim$500 – 1000Ω	$\sim$20 – 100Ω
Switch Capacitance	$\sim$1 – 2 fF	$\sim$1 – 2 fF	$<$ 1 fF
Programmable Yield	100%	100%	$>$ 90%

1.2.2.1 Susceptibility of FPGA technologies to radiation

FPGAs' technologies play an important role regarding the susceptibility of such devices to radiation events. For instance, antifuse-based devices are relatively immune to soft errors due to radiation. The main reason is that once an antifuse is programmable, a particle can not change its state [46, 47]. However, antifuse devices still have user memory which is susceptible to soft errors. Therefore, techniques to mitigate transient errors in antifuse devices may yet be required for critical applications.

SRAM-based FPGAs are of special concern because they are one of the most used FPGAs on the market, and yet one of the most susceptible devices to radiation. The foremost reason is that SRAM-FPGAs, as the name implies, are mainly composed of SRAM cells, one of the most vulnerable elements to soft errors [48]. The sensitivity of a memory device to radiation particles depends on several factors such as the capacitance of the node, the operating voltage, the volume of the depletion region, and the strength of the feedback transistors. In the past, SRAM devices were very robust to soft errors due to the use of higher operating voltages and stronger transistors, therefore requiring a very high-energy charge in order to reach the switching threshold and produce a soft error. Further, in case of SRAM cells, the speed in which the circuit can react also plays a major role. Indeed, the speed of an SRAM cell directly affects the time in which the feedback circuit can restore the corrupted node. Generally, considering the same technology, the slower is the speed of the cell, the more robust that cell is to radiation. However, the technology scaling has reduced the dimensions of transistors, the node capacitance and the operating voltage, which increased the sensitivity of SRAM cells to radiation particles. Further, the operation frequency of SRAM devices is rapidly increasing with technology scaling. In spite of that, these factors have been counterbalanced by the evolution of the process technology in deep-submicron devices ($< 0.25\mu m$), which lowered the junction collection efficiency and maintained the sensitivity of a single-bit SRAM cell almost constant [41,49]. It is important to notice that, despite this fact, the exponential growth in the amount of SRAM required in current electronic devices has led to a higher probability of occurrence of SEUs in current SRAM-based FPGAs.

Flash-based FPGAs are alternatives to SRAM-based devices that can provide the same reconfiguration capability while still presenting a nonvolatile storage capability. In order to do that, a flash memory cell contains a floating gate, located between the control gate and the MOSFET structure, encased in a very good dielectric [49]. The floating gate is used to store the bit value, and writing and erasing operations are performed by applying a relatively high voltage for a few milliseconds. Because of that, flash-based devices are more robust to radiation than SRAM-based ones. Further, unlike mainstream flash-based devices that are conceived focusing on speed and size, flash cells built to provide the reconfiguration mechanism to FPGAs feature a far more robust construction. Indeed, they can be considered very robust to particles originating from cosmic rays [50].

1.2.2.2 Faults in FPGA devices

The most common faults occurring in FPGA devices due to radiation particles are SETs and SEUs. As explained in Section 1.2.1.1, single-event transients are transient voltage pulses that can be propagated through the circuit and then generate errors. Notice that

SETs can occur in several elements such as combinatorial logic, PLLs, and charge pumps, among others. On the other hand, a single-event upset represents a change/flip in the state of a memory element (a bit-flip). SEUs in FPGAs can occur either in the block memory or in the configuration memory. In the former case, the SEU will affect data bits required by the application, and they are generally mitigated by the use of error correcting codes and/or scrubbing. The last case is generally restricted to SRAM-based devices, and it can generate a Single-Event Functional Interrupt (SEFI).

SEFIs are of great concern for the reliability of electronic circuits due to their severity. In fact, an SEFI changes the state of a programmable interconnect, and therefore it can affect the functionality of the device. Because of that, some manufacturers, e.g. Xilinx, use larger and more robust transistors for the configuration memory in order to reduce the occurrence of SEFIs [51]. Generally, an SEFI can cause one of the following symptoms:

- Changing the functionality of a logic module or embedded block;
- Shorting of a signal to power or ground;
- Bridging of two signals;
- Changing the direction or standard of an I/O;
- Breaking of a routing connection.

In spite of that, FPGAs contain millions of configurations' bits as a means to cover with all the interconnection/logical function possibilities that a design may require. Therefore, only a small fraction (generally 10% to 30% [49]) of the configuration memory is actually used in most of FPGA's designs.

Now that the susceptibility of integrated circuits to the different types of faults was explained, let us analyze in Section 1.2.3 some methods stated in the literature to evaluate the reliability of a circuit.

1.2.3 Prior works on reliability analysis

Over the years, many works have been proposed to study the behavior of logic circuits in the presence of different types of faults [15, 16, 52–55]. The first model used to represent faults in logic circuits was the "stuck-at" model [52]. This model considers that most of the failures in a circuit are due to permanent faults, and that they manifest themselves by driving a logical signal level in a circuit node/line to stuck at a constant value, i.e., 0 or 1. Although it has been proved that most of short-circuit type faults or bridging faults can be covered by this model [53], efficient algorithms for this purpose were not easily conceived. In order to tackle this problem, *Ogus* proposed a probabilistic model of logic circuits in which he introduced the concept of *signal reliability* [54].

The *signal reliability* of the output of a circuit is defined as the probability that this output is correct. As stated in [54], the proposed model allows to evaluate the reliability

of a circuit by performing straightforward operations so that it can be easily automated. The idea is to inject into the circuit one fault pattern each time, whether it be a single fault or a multiple fault, and analyze the presence of errors at the output. The procedure of fault injection is shown in Figure 1.5. Notice that two versions of the Device Under Test (DUT) are available, one fault-free and one fault-prone. In this case, stuck-at faults are injected in the fault-prone copy B and its output is compared with the fault-free version A.

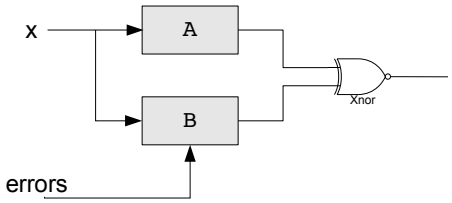


Figure 1.5: Fault simulation approach proposed in [54]

The probabilistic model presented in [54] deals only with stuck-at faults, and therefore it can not be used to model bit-flips, which is of major concern for the analysis of Single Event Effects (SEEs). Further, it does not take into account the individual contribution of the circuit gates for the final reliability value, which could provide additional information in order to harden the desired circuit [8].

A method that can deal with soft errors produced by SEEs was proposed in [55]. The proposed method, named Probabilistic Transfer Matrix (PTM), is based on probabilistic matrices that correlates the inputs and outputs of logic circuits. In order to do that, the topology of the logic circuit as well as the individual reliability of its logic gates must be taken into consideration.

The PTM model is based on a very simple idea: the operation of an error-free logic circuit can be defined by its truth table, that is its behavior is deterministic. However, if a circuit is composed of fault-prone components, its output can present different results for the same input sequence due to the occurrence of errors. Taking this fact into consideration, the PTM model modifies the truth table of a logic circuit in order to behave as a non-deterministic function. This means that a probability of occurrence of an error, denoted by p, is now considered. By doing that, the truth table of an AND gate, for example, becomes the one shown in Figure 1.6.

Using the probabilistic matrices, the authors have then defined a set of basic operations that must be performed to interconnect different logic elements. This is shown in Figure 1.7.

The PTM model can be directly applied to analyze soft and permanent errors induced by bit-flips or stuck-at faults, for example. However, the size of the probabilistic matrices grows exponentially with the bit width associated with the input and the output of a

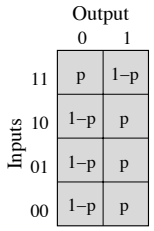


Figure 1.6: PTM representation for an AND gate [55]

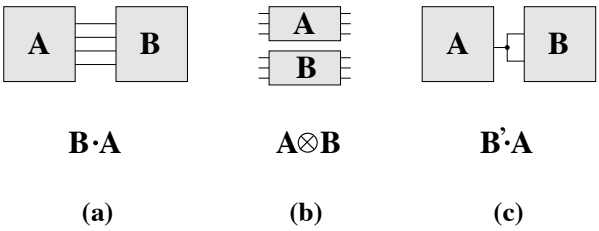


Figure 1.7: Basic interconnection models: (a) serial - (b) parallel - (c) fanout

circuit. This leads to intractable computation times and huge memory storage needs in order to analyze the reliability of large circuits. In order to deal with such drawbacks, the *Signal Probability Reliability* (SPR) model was proposed in [15]. The SPR model is based on the signal reliability concept, and therefore it assumes that the cumulative effects of multiple faults in a circuit can be used to evaluate the probability that the output is correct [8].

The SPR model relies on the consideration that a logic signal can take four different values: correct 0 (0_c), incorrect 0 (0_i), correct 1 (1_c), and incorrect 1 (1_i). The probabilities of a signal x to take one of these four values are organized in a 2×2 matrix as shown in (1.8). The reliability of a signal can then be obtained by adding the values corresponding to the correct operation of the circuit (0_c and 1_c). By convenience, the SPR model uses probabilistic transfer matrices (PTMs) and ideal transfer matrices (ITMs) to represent the fault-prone and the fault-free behavior of the logic components [8].

$$Signal = \begin{bmatrix} P(x = 0_c) & P(x = 1_i) \\ P(x = 0_i) & P(x = 1_c) \end{bmatrix} \tag{1.8}$$

The SPR matrix representing the output of a logic element g_i can be evaluated by performing matrices operations as shown in (1.9). In this case, the INPUT_{gi} matrix can be obtained by calculating the Kronecker product of the SPR matrices of the inputs of g_i. For the sake of illustration, Figure 1.8 illustrates this operation for a 2-input OR gate for which the inputs are uniformly distributed.

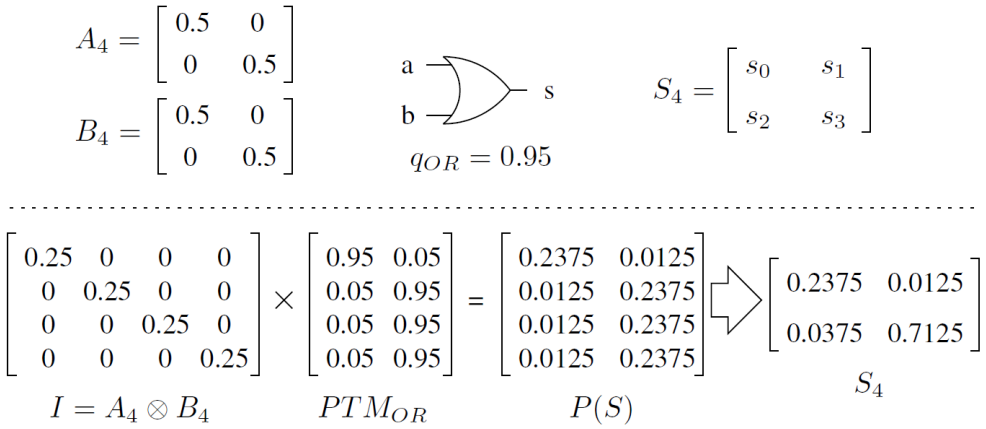


Figure 1.8: SPR matrix for the output of a 2-input OR gate

$$\mathrm{SPR}_{\mathrm{out}_i} = \mathrm{ITM}'_{g_i} \times (\mathrm{INPUT}_{g_i} \times \mathrm{PTM}_{g_i}) \tag{1.9}$$

The evaluation of the signal reliability for the output of a circuit can be performed by the propagation of the signal matrices as shown in Figure 1.9. In this example, the inputs are assumed as being equiprobable, that is, a uniform probability distribution is considered.

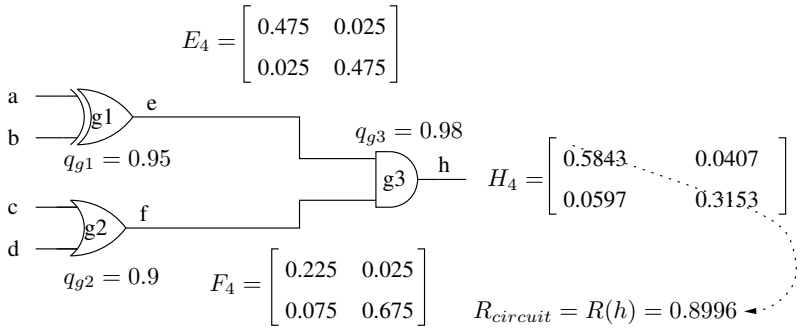


Figure 1.9: Propagation of the SPR matrices through a circuit [8]

It can be noted that the complexity of the SPR algorithm is linear regarding the number of logic elements in the circuit. However, the SPR method cannot deal with reconvergent fanouts due to the presence of signal correlations. In order to tackle this problem, a multi-path version of the signal probability reliability model named SPR-MP was also proposed in [15]. In this case, the method evaluates the contribution of each element of the SPR matrix representing the reconvergent fanout node. Because of that, the complexity of the algorithm grows from a linear complexity to 4^N, where N is the number of reconvergent fanouts in the circuit.

Based on the idea of *signal reliability*, the PBR (Probabilistic Binomial Reliability)

model was proposed. This is a probabilistic model that analyzes the reliability of combinational circuits using a probabilistic binomial distribution [16]. Although both the probabilistic model proposed in [54] and the PBR model use fault injection simulation/emulation in order to evaluate the signal reliability of the output, the last one uses the binomial model as a means to represent SEEs in combinational circuits. Indeed, the PBR model considers that an error in a given gate results in an inversion of the output signal (a bit-flip).

In order to understand the PBR approach, let us consider a generic logic circuit C_1 with input and output vectors $\mathbf{x}$ and $\mathbf{y}$, respectively. Let us also consider that C_1 is composed of w fault-prone elementary gates and m inputs, and that the probability of failure of a gate is represented by $1-q$. Then, the probability of occurrence of k simultaneous errors in C_1 can be evaluated as shown in (1.10).

$$f(q) = (1-q)^k q^{w-k} \tag{1.10}$$

In the PBR model, an error pattern is represented as a vector $\mathbf{e}$ of w bits where the bit $e_i = 1$ indicates a bit-flip in the gate g_i. By using that, the number of 1's in the vector $\mathbf{e}$ represents the number of injected errors in a given moment. Let us now denote the set of vectors $\mathbf{e}$ containing k simultaneous errors as $\mathbf{e}_{w:k}$. Then, the error-free situation is represented by $\mathbf{e}_{w:0}$, and the number of errors patterns associated with the occurrence of k simultaneous errors is given by $C_k^w = \frac{w!}{(w-k)!k!}$. Therefore, for a given error pattern $\mathbf{e}_{w:k}(l)$ and a given input $\mathbf{x}_j$, the boolean expression (1.11) represents the occurrence of error masking in the output $\mathbf{y}$.

$$\overline{y(\mathbf{x}_j, \mathbf{e}_{w:0}) \oplus y(\mathbf{x}_j, \mathbf{e}_{w:k}(l))} = 1 \tag{1.11}$$

Considering a uniform probability distribution for the input vectors, the signal reliability for the output $\mathbf{y}$ can be evaluated by (1.12), where c_k is a masking coefficient obtained by (1.13)

$$R = \frac{1}{2^m} \sum_{k=0}^{w} f(q) c_k \tag{1.12}$$

$$c_k = \sum_{l=1}^{C_k^w} \sum_{j=0}^{2^m-1} \overline{y(\mathbf{x}_j, \mathbf{e}_{w:0}) \oplus y(\mathbf{x}_j, \mathbf{e}_{w:k}(l))} \tag{1.13}$$

Notice that (1.12) evaluates an accurate value for the signal reliability of the output based on the aforementioned assumptions. It can be noted that in order to do that, an exhaustive calculation must be performed since all the possible C_k^w error configurations must be considered, which is very time consuming. Nevertheless, a high number of si-

multaneous errors are not always likely to happen. Therefore, an approximate value for the signal reliability of the output can be obtained by limiting the number k of injected errors. This can speed up the evaluation process in the detriment of accuracy. Further, the PBR approach separates the logical masking calculation from the statical analysis of the circuit. By doing that, once a given circuit architecture has its logical masking ability characterized by fault injection, the reliability can be evaluated for different technologies without the need of re-performing the fault injection analysis. Indeed, what need to be done is to re-evaluate Equation (1.12) by considering a different value of 'q'.

As a matter of fact, fault injection has been considered very useful to analyze the behavior of digital circuits in the presence of faults. Many fault-injection platforms have been proposed over the years. For instance, FuSE is a platform proposed in [17] that supports both emulation and simulation-based fault injection campaigns, and therefore it provides a good flexibility. On the flip side of the coin, the collection of data is performed by reading text files that tends to be big and difficult to interpret. THESIC+ is another fault-injection platform conceived as a means to characterize radiation-induced faults in digital architectures [56]. Although this platform has been used to different purposes such as reproduce the results of radiation ground testing for microprocessors [57], and analyze the robustness of TMR systems implemented in FPGAs [58], it focuses on the effects of single-event upsets occurring in the memory elements of a design. One interesting and recently proposed approach to deal with SET effects is the AMUSE platform [59]. This platform emulates the effects of transient faults by considering a multilevel approach for fault injection. Indeed, the fault injection is performed at gate level in order to provide delay accuracy, and the fault propagation is performed at RTL level in order to speed up the process.

As can be seen, many methods to analyze the reliability of digital circuits are available in the literature, each with its pros and cons. The choice of which technique to use depends on some factors such as the purpose of the reliability analysis, the type of faults are expected to occur, among others. In the current work, the reliability analysis is performed focusing on a fabless manufacturing, for which a designer will elaborate a reliable circuit by modifying its architecture. Moreover, the design of reliable circuits is performed in such a way that its robustness will be as technology independent as possible. Therefore, the target is to improve the reliability of a circuit by maximizing its logical masking ability. Because of that, techniques to characterize the reliability of a circuit based on its logical masking ability are highly required. In this thesis two methods were develop in order to cope with that. The first one is a fault injection tool based on the PBR approach, while the second one is an implementation of the SPR algorithm. Both methods are explained in Appendix 2.

1.3 Reliability improvement of integrated circuits

As the expected number of errors and defects increases with the technology scaling, fault-tolerance techniques are more and more required. It is well known that the reliability of a circuit can be improved through the properly use of redundant components [23,60–62]. Basically, redundancy techniques can be classified as follows [63]:

- *Modular Redundancy or Fault Masking* → redundant components are used to mask the effect of a fault so that it will not reach the primary output of a system. The most used modular redundancy technique is the TMR (Triple Modular Redundancy), in which a system is triplicated and the output is obtained through a voting process. This technique was applied in important scientific programs over the years. As an example, the TMR technique was applied in the Saturn V [64], a rocket used by NASA's Apollo and Skylab programs from 1967 until 1973.
- *Fault Detection and Correction* → this mechanism is based on two steps: fault detection and corrective action. Fault detection can be classified as concurrent and periodic [63]. In the first one, the fault detection procedure must be executed concurrently with the system operation. A traditional technique that uses this principle is to duplicate a system and check if any output mismatch exists. In the other one, a diagnosis routine is carried out to search for errors. If any error is encountered in the circuit, a corrective action is executed. As an example, a routine to reconfigure the fault area of the circuit can be performed in order to correct it.
- *Hybrid Redundancy* → this method benefits from both mechanisms presented above. It can use a modular redundancy technique to detect and mask errors and a fault detection redundancy technique to provide additional corrective actions.

Although many techniques to improve the reliability of an integrated circuit are available in the literature, this thesis focus on the modular redundancy approach. This is because, despite its simplicity, this technique can provide a great performance as will be shown in Section 1.3.1. A review of other methods to improve the reliability of a circuit can be found in Appendix A.

1.3.1 Modular redundancy

The concept of triple modular redundancy (TMR) was originally envisaged by Von Neumann in his work "*Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components*" [23]. This concept is based on three identical modules and a majority voter that will compute the output of the circuit, as shown in Figure 1.10.

Considering the majority voter does not fail, i.e., it is a perfect voter, the output of the circuit will be correct if at most one of the three redundant modules fails. Therefore, the

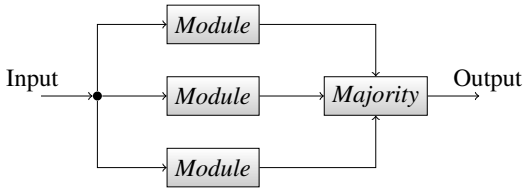


Figure 1.10: TMR concept envisaged by Von Neumann

reliability of the circuit presented in Figure 1.10 can be evaluated by (5.1), where R_M is the reliability of one module.

$$R_{TMR} = R_M^3 + 3R_M^2(1 - R_M)$$
$$R_{TMR} = 3R_M^2 - 2R_M^3 \tag{1.14}$$

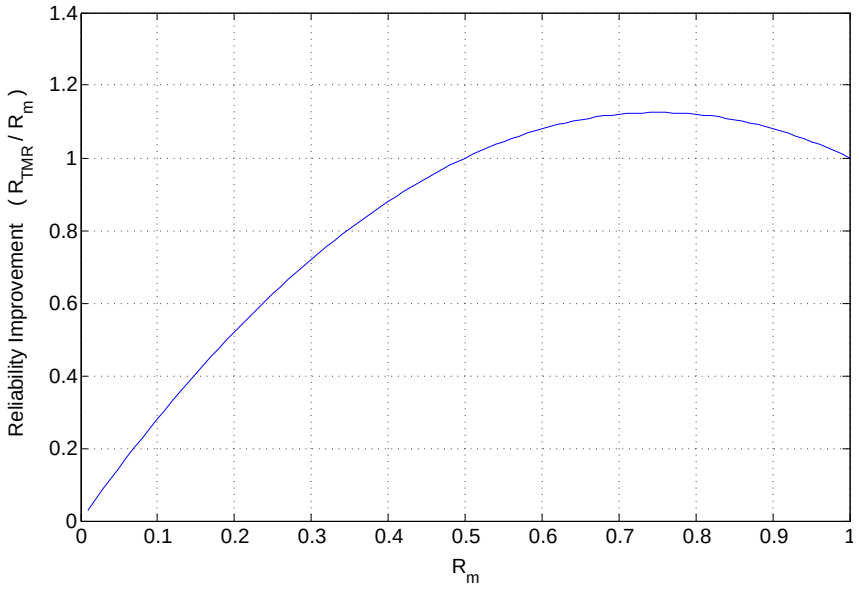


Figure 1.11: TMR performance regarding reliability improvement

Analyzing (1.14), it can be noted that the TMR technique does not improve the reliability of a circuit if the reliability of each module is less than 0.5. Moreover, the behavior of this function, illustrated in Figure 1.11, shows us that the gain obtained with TMR has second-order effect when the reliability of each module is very near unity. Thus, the size of each module is a crucial role for TMR performance regarding reliability improvement since when we increase the size of a circuit, generally the reliability decreases. Further mathematical details regarding TMR performance can be found in [60, 65].

The system architecture shown in Figure 1.10 takes into account that the majority

voter does not fail. However, considering the majority voter as an imperfect voting circuit, the TMR performance is compromised. A single error presented in the majority voter will result in an erroneous output. In order to minimize this problem, a different architecture, proposed by Cohn [66] and shown in Figure 1.12, may be utilized . In this case, if a single majority voter fails, the output is still considered correct.

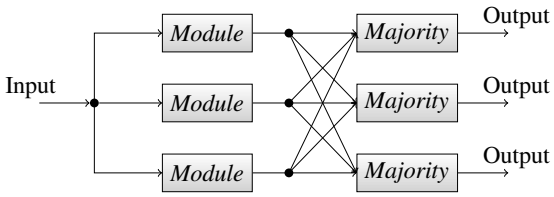


Figure 1.12: TMR with three majority voters

Despite the widely use of TMR regarding reliability improvement, we must deal with some constraints. The area penalty required by this technique is higher than three times the area of a circuit. Therefore, before applying TMR architecture to a circuit, we must investigate its pros and cons. The work presented in [67] analyzes area and performance penalty whether to use TMR or Hamming Code in different digital modules. The results have shown that TMR is more appropriated for modules using single registers and Hamming Codes for groups of registers.

Regarding the robustness of TMR in SRAM-Based FPGAs, the work in [68] have shown that TMR is not able to effectively protect the circuit against SEUs affecting the configuration memory. According to their results, the percentage of faults that escape TMR could reach 13% for the analyzed architectures. However, the adoption of a smart floorplan that isolates the three modules can significantly reduce the number of faults that escapes TMR. The work in [58] analyzed the weakness of the TMR strategy implemented in SRAM-based FPGAs . They performed a series of radiation tests in a Xilinx Virtex-II device in which a cryptocore application was used as a device under test (DUT). The results have shown that the voter may not be able to detect some single faults occurred on the configuration bits. In 2010, a method to protect different modules against short-effects of SEUs by separating two routed nodes from each other by at least two programmable interconnect points (PIPs) was proposed in [69]. Using the proposed approach, a TMR system is isolated in such a way that an SEU cannot affect two modules. At the same year, an algorithm to improve the reliability of TMR designs in SRAM-based FPGAs against multiple cell upsets was introduced in [70]. The algorithm, called PHAM (Placement Hardening Algorithm for Multiple cell upsets), takes into account the FPGA physical layout to determine the best locations for each block of a TMR system. The results have shown that circuits using the proposed algorithm can achieve 34 times better robustness

than the ones using the standard TMR approach.

The intrinsic capability to mask erroneous results is an attractive characteristic to use TMR in fault-tolerant systems. This capability is provided by the voting process, making it an essential part of a TMR system. Therefore, let us review some important voting strategies proposed over the years in Section 1.3.2.

1.3.2 Voting strategies

The most used voting process is the *Majority Voter*, proposed by John von Neumann in 1965 [23]. This approach evaluates the system output based on a majority rule. Thus, considering a TMR system, a 2-out-of-3 rule is used.

Despite its frequently use, the majority process presents some limitations such as the fact that it does not consider common-mode failures, i.e., failures that affect more than one module of the system. In order to take these failures into consideration, an alternative voter process named *Word-Voter* was proposed in [21]. The word-voter is applicable to TMR systems with a multiple-bit output. A diagram of a 2-bit word-voter is shown in Figure 1.13. It compares the output word from each module and if at least two of them are equal, the system will produce a correct output. If all the output words are different, it is a signal that multiple errors have occurred and the system output is compromised. Thus, an error signal is activated and corrective procedures are carried out.

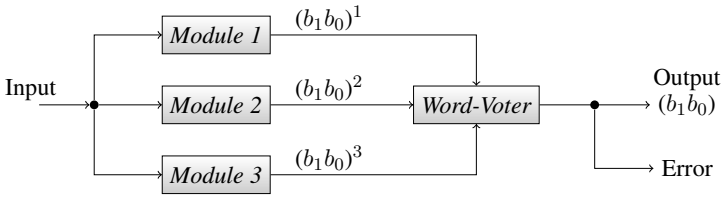


Figure 1.13: Word-Voter proposed in [21]

The concept of TMR can be generalized by using the term N-Modular Redundancy (NMR). For instance, a 5MR technique based on a majority process would use a 3-out-of-5 rule. However, when using a higher numbers of modules than TMR, it may be acceptable to use a more relaxed rule to evaluate the system output. As an example, it may be acceptable to use a 2-out-of-5 rule for a 5MR system. This voting process is named *Plurality Voter* [71]. Obviously, this strategy cannot be used in case of a bit-based voter, but it is perfectly usable if a word voter is available.

Several other voting techniques are available in the literature. For instance, the *Weighted Voter* [72] uses the outputs of the modules to evaluate the mean value considering that each module output has a weight. Notice that the system output can be distinct from all

the module results. The *Generalized Median Voter* [73] supposes that the output space is a metric space. Then, it analyzes the outputs of the modules and discards the pairs with maximum reciprocal distance. Finally, the median result is used as the system output. A theoretical investigation of the aforementioned voter strategies regarding redundant systems is available in [73].

The voting techniques discussed above do not take into account the historical behavior of each module to judge what is the correct output. As a matter of fact, the presence of a permanent error in one of the three modules will increase the likelihood that a transient fault in one of the other two will affect the final result. In order to tackle this problem, several history-based voters were proposed over the years [74–76]. In 1999 an approach based on the inexact majority voting was proposed in [74]. It uses the history of correct computations to select the most reliable module in case of agreement. Later in 2001, a voter based on weighted voting was introduced in [75]. This voter uses the history information to modify the weights of each module in order to use the most reliable results when evaluating the system output. An alternative approach regarding the exact majority organ was proposed in [76]. The main idea is to identify the reliability of each module based on its temporal behavior. In order to do that, Dotan proposed the use of indices to represent the historical behavior of the modules. Indeed, each module has an index that is incremented each time the corresponding module computes a correct output. Then, the most reliable module is the one selected to transmit its computation as the output of the system.

With so many voting procedures existent in literature, we need to compare and evaluate theirs pros and cons. The work in [77] analyzed seven different voting strategies in their work entitled "Experimental Comparison of Voting Algorithms in Cases of Disagreement". They used several simulated scenarios and a software error-injection tool to analyze the robustness of the following voters: majority, plurality, median, weighted average, linear prediction, first order prediction and three domain. Among them, the majority and plurality voters produced the lowest number of catastrophic errors. The median voter produced the largest number of correct results. However, it produced the largest number of catastrophic errors as well. The three domain voter showed a compromise providing relatively low number of catastrophic errors while keeping the number of correct results large.

1.3.3 Selective Hardening

As discussed earlier, soft errors have become a serious concern in deep-submicron technologies. These faults used to be a concern only in the design of memories, but with the downscaling of electronics, it is expected that the number of soft errors in combinational circuits will exceed those occurring in unprotected memories [78]. A number of hardening

techniques have been proposed over the years in order to reduce the probability of transient errors in logical circuits. These techniques are based on adding redundancy, whether it be spatial or temporal, which increases some attributes of the design such as cost, power consumption, and performance, among others. With the decrease in the feature size, the number of logic gates is usually very large, and protecting every gate in a circuit may not be a good solution to mainstream devices. Therefore, cost-effective solutions based on protecting only the most critical gates of a design, also known as selective hardening, have been considered a promising solution to limit the protection cost of such techniques [18, 78–84].

Therefore, selective hardening is based on two steps: first, gates are analyzed and ranked according to their susceptibility to soft errors and to cause a circuit malfunction; next, the most critical gates of a circuit are protected. The difficultness to implement a selective hardening technique is to define which is the best subset of gates/blocks that must be protected in order to meet a given reliability requirement, and because of that, many methods have been proposed to perform such selection procedure [18, 78–84]. In [18], a method to selectively apply triple modular redundancy (STMR) into FPGAs was proposed. The idea is to calculate the sensitivity of the gates of a circuit based on their input probabilities, and then select subcircuits to be protected by analyzing the longest cascade chain of sensitive gates starting from the primary outputs and backtracking through the circuit. Although the method does not guarantee 100% protection against single-faults, the savings in area overhead reached 65% for some circuits when compared to a traditional TMR technique. In [80], a strategy based on the gate-level information was proposed in order to deal with this problem. The proposed method does not take into account any low-level electrical or timing information as a means to select the critical gates of a design while still on its early design phase. The basic idea is to define a factor c such that the probability of an erroneous system output p_{err} is reduced to $c \times p_{err}$, considering $p_{err_{min}} < c < 1$. This calculation is performed based on a static logical masking estimation of a circuit, which is independent of the circuit technology. Although the selection procedure does not take into account other masking phenomena such as electrical and latching window, simulations of the hardened circuit considering these phenomena were performed and the results suggest that these masking mechanisms have little influence when selecting critical nodes in a circuit. Later in [83], the authors have evaluated the validity of choosing critical nodes of a circuit based only on its logical masking ability and have come to the same conclusion.

Thus, considering logical masking, the main idea is to classify the composing blocks (i.e., standard cells) of a circuit according to their relative significance with respect to the reliability of the circuit. With the classified list of blocks it is possible to apply selective hardening either by using hardening by design techniques or by more generic fault tolerance techniques like Triple Modular Redundancy (TMR). By using an additional hardening

affinity parameter, a trade-off between the hardening cost of a block and the reliability gain is then clearly established. Chapter 4 will introduce a method that can take that into consideration and uses some heuristics in order to automatically select the best candidates to be protected.

Selective hardening can also be applied based on the error tolerance concept. As shown in [83, 85, 86], some applications have the ability to tolerate some kind of errors, and therefore just a subset of them should be mitigated. Chapter 3 introduces a method to evaluate the reliability of a circuit taking into account the error tolerance of a given application, and chapter 4 proposes two techniques to selectively harden a circuit. The first one selects the most critical gates of a circuit based on the impact of an error in the output of a system, while the next one uses a cost function parameter and two different heuristics to automatically select the best candidates to be protected.

Chapter 2

FIFA Tool

2.1 Introduction

It is well-known that the increased density of circuitry associated with a reduction in the supply voltage have decreased the effects of electrical and temporal masking, thus increasing the likelihood of transient and multiple faults in integrated circuits [87]. For this reason, the prediction of circuit behavior when exposed to faults is becoming more and more important in deep submicron technologies [4].

This chapter introduces a fault-injection based tool developed during this thesis to analyze the robustness of integrated circuits against faults. The proposed tool, named FIFA, is used as a means to validate the construction of fault-tolerant designs along the current work.

2.2 FIFA Tool

Fault injection has been considered very useful to evaluate the behavior of computing systems in the presence of faults [17]. The basic idea of such approach is to produce or simulate faults during system operation, and then observe whether they produce a device failure. Several methodologies can be used in order to inject faults in a circuit. Basically, they can be classified into two types: software or hardware.

Software-based approaches are normally performed as a device simulation by using a netlist description such as SPICE and VHDL/Verilog. If the number of fault-prone components or test vectors considered for fault injection is too high, the required time to perform the simulation procedure can become prohibitive [59]. Hardware-based approaches are a good solution to accelerate the testing procedures. In this case, the testing procedure is executed by emulating the target circuit, and therefore it requires a physical device [57].

The aim of the current work is to build reliable systems by improving their logical

masking ability. Based on that, this Section presents a new tool designed as a hardware IP to accelerate the Fault Injection and Fault masking Analysis (FIFA) approach. The proposed tool was implemented on FPGA, and the analysis is performed at register transfer level (RTL). The FIFA tool is fully parameterizable, allowing the designer to adapt it for analysis of practically any digital circuit. In addition, this IP can help the designer to establish efficient trade-offs between cost (time, amount of FPGA resources) and completeness of the analysis. Unlike previous works, the FIFA tool deals with several fault models and no FPGA reconfiguration is necessary to simulate different fault patterns for the same circuit [88].

The FIFA tool has shown a great performance for generating different fault patterns. However, if multiple simultaneous faults are considered, that is, if several gates may fail at the same time t (fault multiplicity $k > 1$), the number of tests for exhaustive analysis in large circuits may become prohibitive. Indeed, although the hardware implementation provides a reasonable calculation speed, the reliability evaluation is still intractable in such case.

In order to diminish this drawback, two possible solutions can be taken into consideration: first, the occurrence of multiple simultaneous faults in an integrated circuit depends on some of its properties as well as on certain characteristics of the environment in which the circuit is supposed to operate. A good solution is then to limit the number of simultaneous faults to be injected based on such characteristics. This can significantly reduce the required computing time and yet provide great approximated results. Based on that fact, the FIFA tool uses the PBR model as a means to evaluate the reliability of a circuit. This model can provide approximate values for the reliability of a circuit by considering a maximum number of simultaneous errors to be injected. On the flip side of the coin, this feature requires that the fault generation sequence be performed in an ascending order regarding the number of simultaneous faults, that is, first all single faults are generated, then the double faults, and so on. This poses strict difficulties to perform the second solution to reduce the computing time: efficiently parallelize the calculation. As a matter of fact, an optimal parallel implementation relies on a balance of the circuit operations among the parallel modules. However, for the fault pattern generation sequence explained above, this requirement is not easily fulfilled. This is because of the difficulties to calculate which is the z^{th} generated fault pattern on this sequence.

Section 2.2.4 makes use of number patterns to tackle this problem. It introduces an efficient algorithm that can calculate which is the z^{th} generated fault pattern using simple operations. In addition, this algorithm is used to conceive a parallel architecture for the FIFA tool. Results have shown that the proposed architecture can optimize the parallel computation while keeping the area overhead as low as possible. This is done as a means

to increase the number of parallel copies that can be synthesized in a given FPGA support.

2.2.1 FIFA Architecture

As stated above, the FIFA tool is a hardware IP developed to accelerate the fault injection and fault masking analysis approach. In order to understand the functioning of such tool, let us first consider a digital circuit DUT for which we are interested in analyzing its robustness against faults. The basic idea behind the proposed tool is to inject faults in DUT and observe whether this internal fault will produce a device failure. Therefore, some kind of mechanism is required in order to allow the injection of faults during circuit runtime. The fault injection mechanism available in the FIFA tool is based on saboteurs. A saboteur comprises a small set of components which provides the capability to alter the values contained in a circuit node. Thus, appending such components to the nodes of DUT allows the injection of single or multiple faults.

The FIFA tool contains, among other items, two versions of the DUT: one fault-free (DUT REF) and one fault-prone (DUT FAULTY). The analysis of the robustness against a specific fault f_1 in the DUT takes two steps: first, we inject the internal fault f_1 in DUT FAULTY by enabling the corresponding saboteur(s); next, we compare the outputs of both circuits in order to detect any mismatch. Notice that this procedure is done considering a given input i for both circuits, DUT REF and DUT FAULTY. If the injection of f_1 doesn't modify the circuit's output, we can say that the circuit is robust to such fault, that is, f_1 was masked.

Let us now define a **test configuration** of DUT as a couple comprising a given input and a given fault. For a specific set of test configurations, the proposed tool analyses the error masking capabilities of DUT, and determines its corresponding **masking coefficient**. The masking coefficient of a circuit represents the number of test configurations for which it generates correct outputs. Thus, this coefficient is directly related to the **robustness** of a circuit. In our case, we classify the error masking coefficient according to the number k of simultaneous faults injected. Then, we define c_k as the masking coefficient representing the robustness of DUT regarding the occurrence of k simultaneous faults.

2.2.1.1 Defining the components of the FIFA tool

Figure 2.1 shows the proposed tool which comprises the following modules:

- DUT REF – A fault-free version of DUT.
- DUT FAULTY – A faulty version of the device under test. Programmable saboteurs are appended to the nodes of DUT for which we would like to inject faults. These programmable saboteurs support four different fault models: bit-flip (Single Event Upset

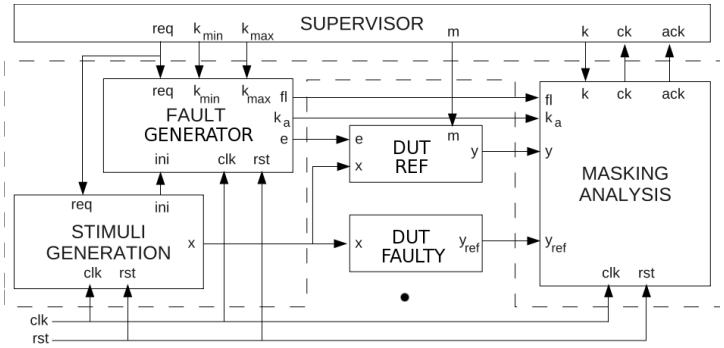


Figure 2.1: General scheme of the proposed tool

or Multiple Bits Upset), stuck-at-0, stuck-at-1, and open circuit (high impedance). In order to emulate a fault in a node j, the corresponding saboteur must be activated (see Figure 2.2). This is done by a control signal (bit e_j in bus $\mathbf{e}$). If $e_j = 0$, the node j is supposed to be fault-free. If $e_j = 1$, the node j is supposed to be faulty. The fault model to be used is selected by the signal $\mathbf{m}[m_1 : m_0]$.

– STIMULI GENERATION – Generates the data inputs for DUT REF and DUT FAULTY (bus $\mathbf{x}$).

– FAULT INJECTION – Generates the control signals to activate/deactivate the saboteurs in DUT FAULTY (bits e_j of bus $\mathbf{e}$). This module was implemented according to the work presented in [89]. We took into account the second algorithm presented in this work, which can generate all the possible C_k^N vectors $\mathbf{e}$ for a given number of simultaneous faults $k_{min} \leq k \leq k_{max}$. This algorithm is explained in Section 2.2.1.3

– MASKING ANALYSIS – Compares the outputs provided by DUT REF (bus $\mathbf{y}_{ref}$) and DUT FAULTY (bus $\mathbf{y}$) in order to evaluate the masking coefficients ($\mathbf{c}_k$ values).

– SUPERVISOR – Manages the communication signals among modules ($\mathbf{m}$, req, $\mathbf{k}$, $\mathbf{k}_{min}$ and $\mathbf{k}_{max}$).

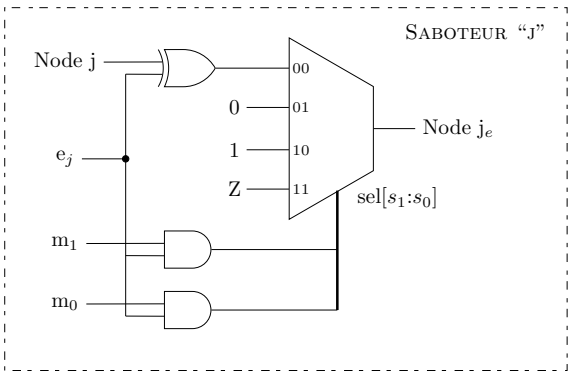


Figure 2.2: General scheme of a saboteur

2.2.1.2 Explaining the communication signals

The FIFA tool is a synchronous circuit operating on the rising edges of a clock signal (clk). In order to better understand the functioning of the proposed tool, it is a prime concern to comprehend the purpose of the communications signals used during operation (see Figure 2.1).

First of all, signal $\mathbf{k}_a$ represents the number of simultaneous faults to be used in the current test configuration. Considering a DUT FAULTY containing N saboteurs, the FAULT INJECTION module generates the set of $C_{k_a}^N = \binom{N}{k_a}$ different $\mathbf{e}$ vectors corresponding to all possible occurrences of k_a errors. In other words, each vector $\mathbf{e} = [e_{N-1} : e_0]$ contains exactly k_a bits at logic value 1 as a means to activate the desired k_a saboteurs. Signals $\mathbf{k}_{min}$ and $\mathbf{k}_{max}$ indicate the minimum and maximum number of simultaneous errors to be considered, respectively. The tool considers fault multiplicity in ascending order. It means that the test configuration starts with $\mathbf{k}_a = \mathbf{k}_{min}$ and concludes with $\mathbf{k}_a = \mathbf{k}_{max}$.

Initialization is done with the asynchronous signal rst, which is active at logic level 0. Indeed, when $rst = 0$, signals fl, ack, and all bits of buses $\mathbf{e}$, $\mathbf{x}$ and $\mathbf{c}_k$ are set to zero.

Signal req is used to indicate a reliability analysis request. If $req = 1$, both STIMULI GENERATION and FAULT INJECTION modules are enabled. The first one generates all the possible values of $\mathbf{x}$ in ascending order: $0 \leq \mathbf{x} \leq (2^Z - 1)$, where Z represents the width of the bit-vector $\mathbf{x}$. The second one generates the bit-vector $\mathbf{e}$ responsible to activate/deactivate the desired saboteurs. When the last $\mathbf{x}$ value is generated, the STIMULI GENERATION module sends a signal $ini = 1$ to the FAULT INJECTION module. This signal enables the corresponding module to generate the next bit-vector $\mathbf{e}$ and to reinitialize the bit-vector $\mathbf{x}$ (i.e. $\mathbf{x} = 0$).

Signal fl enables the MASKING ANALYSIS to compare $\mathbf{y}_{ref}$ and $\mathbf{y}$ in order to evaluate the masking coefficients $\mathbf{c}_k$.

Signal ack indicates that the analysis is finished ($ack = 1$), that is, all masking coefficients $\mathbf{c}_k$ are now available. Then, if we want to retrieve a specific $\mathbf{c}_k$ value, we can use the input $\mathbf{k}$ and the output $\mathbf{c}_k$ from the MASKING ANALYSIS module.

Figure 2.3 shows a timing diagram with the tool signals. It considers a circuit in which N nodes may fail. Notice that the MASKING ANALYSIS module requires the simultaneous activation of both signals, rst and fl, in order to start its functioning.

2.2.1.3 Fault Injection Module

As stated above, the FAULT INJECTION MODULE is responsible to control the activation/deactivation of the saboteurs presented in DUT FAULTY. Considering the presence of N saboteurs, each fault pattern is represented by a vector $\mathbf{e} = [e_{N-1}e_{N-2}\cdots e_1e_0]$, where

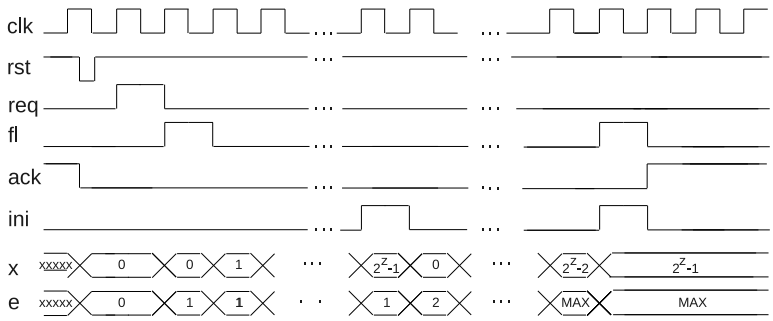


Figure 2.3: Timing diagram of communication signals in the tool ($MAX = 2^N - 2^{N-k_{max}}$)

$e_i = 1$ activates the saboteur i. Therefore, given a number of simultaneous errors k, the Fault Injection Module generates the set of C_k^N vectors $\mathbf{e}$ containing exactly k bits equal to 1.

This fault pattern generation is done based on an algorithm proposed in [89] comprising 4 tasks:

- Task 1 - Set $e_i = 1$ for all $0 \leq i \leq k-1$, and $e_i = 0$ for all $k \leq i \leq N-1$. This task is performed only once for each value of k
- Task 2 - Perform a search from LSB towards MSB in order to find the position m of the first bit 0 after a bit 1 in the previous vector
- Task 3 - Create a temporary vector $\mathbf{t} = [t_{N-1} \cdots t_0]$ by flipping the bits e_m and e_{m-1} of vector $\mathbf{e}$
- Task 4 - Permutate bits t_i and t_{m-2-i} of vector $\mathbf{t}$ for all $0 \leq i \leq m-2$

In order to generate all the possible vectors $\mathbf{e}$ considering a range of simultaneous errors $k_{min} \leq k \leq k_{max}$, Algorithm 1 is performed.

Algorithm 1 Fault Pattern Generation

```
1:  k ← k_min
2:  for k ≤ k_max do
3:      e_1^k ← Task1(N, k)
4:      j ← 2
5:      for j ≤ C_N^k do
6:          m ← Task2(e_{j-1}^k)
7:          t ← Task3(e_{j-1}^k, m)
8:          e_j^k ← Task4(t, m)
9:          j ← j + 1
10:     end for
11: end for
```

In an effort to better understand Algorithm 1 realization, let us consider the generation of the whole set of vectors $\mathbf{e} = [e_2 e_1 e_0]$ for $k_{max} = 2$. In this case, the step by step execution

is shown in Figure 2.4.

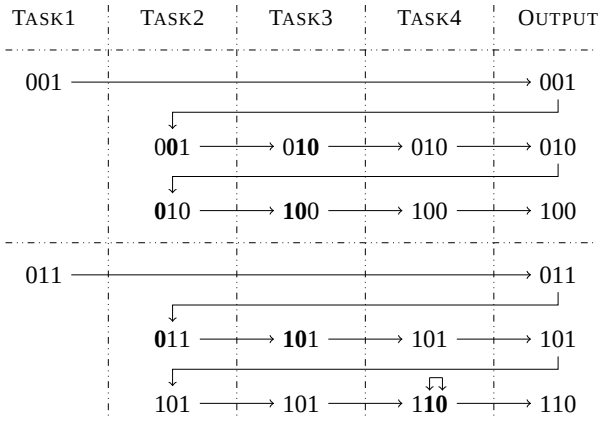


Figure 2.4: Example of a step by step execution of Algorithm 1 considering $N = 3$ and $k_{max} = 2$

2.2.2 Reliability Assessment

The main purpose of the FIFA tool is to characterize the logical masking ability of a given circuit architecture. This ability can be used in order to evaluate its reliability by using the PBR model [16] explained in Chapter 1. According to such model, the reliability of a circuit can be calculated by (2.1), where:

- N is the number of gates that may fail.
- q represents the reliability of a gate, that is, the probability that it doesn't fail. We consider all gates in a circuit as having the same reliability value.
- $f(k) = (1-q)^k q^{N-k}$ denotes the probability that k gates fail simultaneously. Notice that more complex models can also be used to evaluate this term as shown in [16].
- c_k denotes a coefficient related to the masking of k simultaneous errors in a circuit. Considering that the target circuit has Z input bits, it can be calculated using (2.2).

$$R = \sum_{k=0}^{N} f(k)ck \tag{2.1}$$

$$c_k = \sum_{j=0}^{2^Z-1} p(x_j) \left(\sum_{l=1}^{C_k^N} \overline{y(x_j, e_{N:0}) \oplus y(x_j, e_{N:k}(l))} \right) \tag{2.2}$$

The exact reliability value can be calculated by considering all the possible input vectors and fault patterns as shown in (2.1). However, an approximated value of the reliability can be obtained if a maximum number of simultaneous errors $k_{max} < w$ is considered, which significantly reduce the required computing time. Indeed, for most of real world systems,

w simultaneous faults are not likely to happen.

2.2.3 Synthesis Results

We have implemented a full parameterizable HDL description of the proposed tool. Indeed, parameters such as the number of fault-prone gates (N), the number of stimuli data bits (Z), and the number of output bits (P) can be selected in order to match a target design DUT.

In order to evaluate the implementation cost of the proposed tool, we have synthesized several versions of the IP using a `STRATIX II EP2S180F1508C3` FPGA from Altera®. Each implemented version has considered a different number of fault-prone gates (N). Even though the implementations have considered the same number of input/output bits ($Z = 5$, $P = 3$), the proposed IP can deal with any value of Z and P.

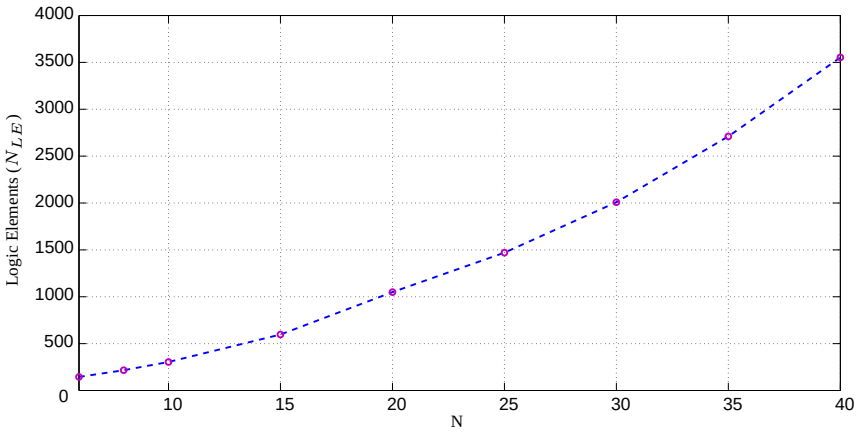
The synthesis results are shown in Figure 2.5. Only the main components of the FIFA tool are taken into consideration. They are: the STIMULI GENERATION, the FAULT INJECTION, and the MASKING ANALYSIS modules, which are responsible for fault injection and fault masking analysis.

When $N = 6$, the tool implementation requires less than 0.1% of the FPGA resources ($N_{LE} = 144$) and $f_{max} = 215.98MHz$. Even if we consider a large number of fault-prone nodes, the proposed IP remains very compact. For example, with $N = 40$, the tool implementation requires only 2% ($N_{LE} = 3555$) of the available LEs in the target FPGA. In the case of $N = 40$, more than 80 millions test configurations can be generated in every second ($f_{max} = 82.41MHz$).

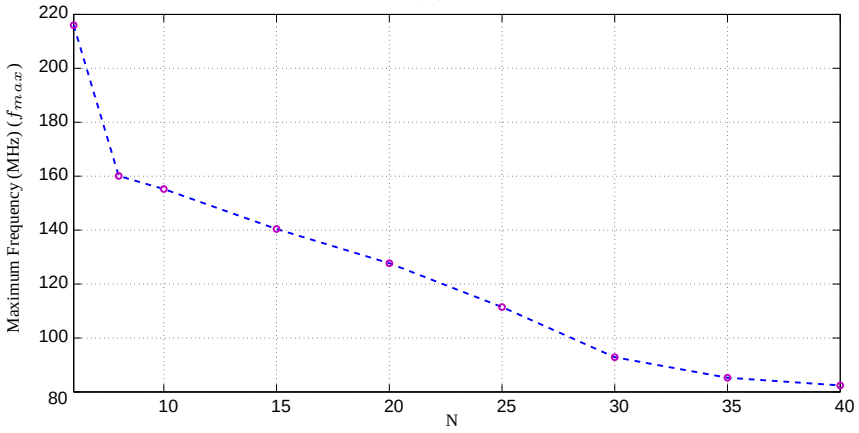
We compare the performance of the FIFA tool with the FuSE HDL platform proposed in [17]. The comparison in terms of resource requirements and performance is shown in Table 2.1. Both implementations have considered $N = 10$ and $P = Z = 32$. Notice that the Fuse platform only deals with single faults, while the FIFA tool deals with single and multiple faults ($k_{max} = N$). Nevertheless, the proposed IP is more efficient in terms of time and resource requirements. If the occurrence of several simultaneous faults (large values of k_a) is not probable, we can still optimize the IP implementation by reducing the width of the buses $\mathbf{c}_k$, $\mathbf{k}_{min}$, $\mathbf{k}_{max}$, $\mathbf{k}_a$ and $\mathbf{k}$.

Other tools, such as those presented in [90] and [91], have a temporal cost that grows with the complexity of the fault model. Instead of this, the FIFA tool presents a fault model which has no significant impact on its performance.

The tool presented in [92] only supports permanent stuck-at faults, and the tools [17] and [93] only deal with single faults. Unlike those works, FIFA tool deals with single and multiple faults, and it supports permanent and transient faults as described in Section 2.2.1.1.



(a)



(b)

Figure 2.5: Synthesis results of the tool (up to N simultaneous errors): (a) number of logic elements required in the FPGA (b) maximum frequency for error generation

Table 2.1: Fuse platform vs. FIFA tool

	Fuse [17]	FIFA
ALUTs	2157	817
Registers	694	467
Maximum Frequency	75.1MHz	109.87MHz

As the fault injection approach can become very time-consuming for large circuits, a dedicated hardware may be used as a means to accelerate calculations [94]. Despite this solution have been used in [88], each test configuration requires reprogramming the FPGA. This significantly reduces the efficiency, even if partial reconfiguration is used. The proposed tool is composed in such a way that it avoids reprogramming the FPGA during the analysis of a given circuit. Indeed, using the available control signals we can select not only in which nodes we would like to inject faults, but also the fault model(s) to be used.

The FIFA tool was implemented from scratch without benefiting from any proprietary libraries. Thus, all the required functions together with the memory control access (DMA - Direct Memory Access) were implemented using standard cells, making the FIFA tool widely flexible to be used with any FPGA. The modifications should be restricted to the supervisor module, where we need to specify the communication interface between the tool and the computer according to availability in the target FPGA. Furthermore, the FIFA tool allows the designer to control the time complexity as well as the pertinence of the test configurations. Although such tool has shown to be capable to inject faults in a circuit in a reasonable speed, the reliability evaluation may be still intractable in case of large circuits. In order to reduce this drawback, a parallel approach for the FIFA tool is presented in next Section.

2.2.4 Parallelizing the FIFA Fault Generation

As a matter of fact, the performance of the FIFA tool can be improved by using parallel computation. A very simple parallel architecture for this tool is shown in Figure 2.6. In this case, each STIMULI GENERATOR is responsible for generating a different set of inputs. Theoretically, this approach can reduce the computing time by a factor of T_{comp}/N. Nevertheless, the area overhead is extremely high, thus limiting the number N of parallel copies that can be synthesized in an FPGA device. Indeed, since N input vectors are being generated each time, N fault-free DUTs are needed in order to enable the output comparison performed by the MASKING ANALYSIS block. One solution to that would be to save all the fault-free output values in a memory, but the required amount of resources may become prohibitive.

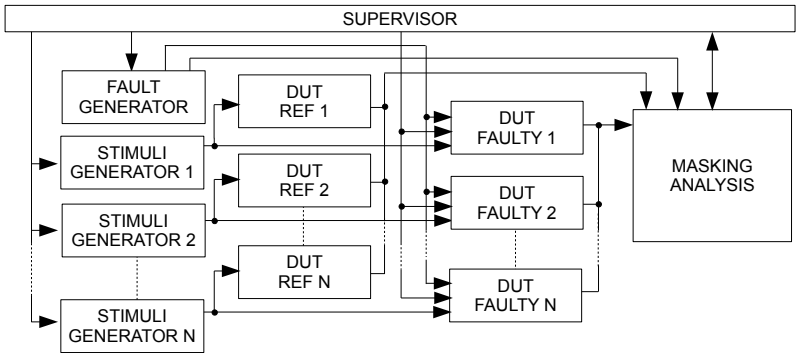


Figure 2.6: A simple parallel architecture for FIFA

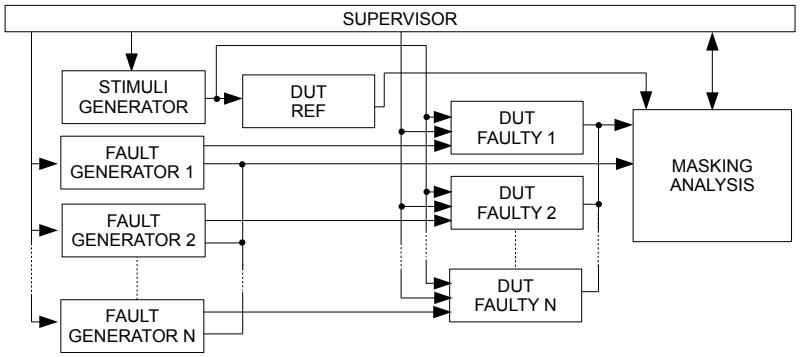


Figure 2.7: The proposed parallel architecture for FIFA

We propose another architecture to parallelize the FIFA tool, shown in Figure 2.7. In this case, the tool synthesizes N FAULT GENERATORS, each one responsible for generating a different set of fault patterns. This approach avoids the duplication of the fault-free DUT, thus reducing the area overhead compared to the previous solution.

The parallel computation can be optimized if the FAULT GENERATORS produce the same number of fault patterns, that is, if they are well balanced. In order to do that, each FAULT GENERATOR must be initialized by two main parameters: the number of fault patterns to be generated, and the initial vector $\mathbf{e}$. As the fault pattern generation sequence is done for $k = \{1, 2, \cdots, w\}$, the calculation of the initial vector $\mathbf{e}$ is not a trivial task. Indeed, the difficultness to implement the architecture shown in Figure 2.7 lies on this calculation, for which a solution is proposed in Section 2.2.4.1.

2.2.4.1 Calculating the initial vectors

Let us consider the fault pattern generation sequence for a 6-bit vector $\mathbf{e}$ shown in Figure 2.8. The aim is to find an algorithm that, based on this generation sequence, can find which is the z^{th} fault pattern $\mathbf{e}$.

In order to do that, let us explore some number pattern in this sequence. First, let us

IDX	k = 1	k = 2	k = 3	k = 4	k = 5	k = 6
0	000001	000011	000111	001111	011111	111111
1	**000010**	**000101**	**001011**	**010111**	**101111**	
2	**000100**	000110	001101	011011	110111	
3	**001000**	**001001**	001110	011101	111011	
4	**010000**	001010	**010011**	011110	111101	
5	**100000**	001100	010101	**100111**	111110	
6		**010001**	010110	101011		
7		010010	011001	101101		
8		010100	011010	101110		
9		011000	011100	110011		
10		**100001**	**100011**	110101		
11		100010	100101	110110		
12		100100	100110	111001		
13		101000	101001	111010		
14		110000	101010	111100		
15			101100			
16			110001			
17			110010			
18			110100			
19			111000			

Figure 2.8: Example of a fault pattern generation sequence

analyze the behavior of left-shifting the left-most bit '1' in the first vector of each column. This is represented by the bold vectors shown in Figure 2.8. Extrapolating the fault-pattern generation for a w-bit vector $\mathbf{e}$, we obtain the following sequences representing the indices of the bold vectors:

- $k = 1 \rightarrow \{1, 2, 3, 4, 5, \cdots\}$
- $k = 2 \rightarrow \{1, 3, 6, 10, 15, \cdots\}$
- $k = 3 \rightarrow \{1, 4, 10, 20, 35, \cdots\}$
- $k = 4 \rightarrow \{1, 5, 15, 35, 70, \cdots\}$
- $k = 5 \rightarrow \{1, 6, 21, 56, 126, \cdots\}$

Notice that the number pattern shown above corresponds to the columns of the Pascal's triangle illustrated in Figure 2.9. Therefore, for a given k, the n^{th} element of the corresponding sequence can be evaluated by (2.3). This allows us to obtain any bold vector by performing left-shifting operations on the left-most bit '1' in the first vector of a column. For example, in order to obtain the vector in column $k = 2$ with index 10, we should perform four left-shift operations ($\mathbf{e}(\mathbf{1}) = 00001\mathbf{1} \rightarrow \mathbf{e}(\mathbf{10}) = \mathbf{1}00001$) because 10 is the fourth element in the corresponding sequence ($elem_{4_2} = 10$).

$$elem_{n_k} = \frac{n(n+1)\cdots(n+k-1)}{\mathrm{K!}} \tag{2.3}$$

Let us now explore some other number pattern to allow the evaluation of non-bold vectors. First of all, let us denote Φ_k^j as the set of vectors in a column k for which the left-most bit '1' is located at the position j of $\mathbf{e} = (e_w e_{w-1} \cdots e_1)$. For example, in Figure 2.8 we have $\Phi_2^4 = \{001\mathbf{001}, 001\mathbf{010}, 001\mathbf{100}\}$. Notice that the elements of Φ_k^j show an interesting

	K = 1	K = 2	K = 3	K = 4	K = 5	K = 6	K = 7	K = 8	K = 9	
1										
1	1									
1	2	1								
1	3	3	1							
1	4	6	4	1						
1	5	10	10	5	1					
1	6	15	20	15	6	1				
1	7	21	35	35	21	7	1			
1	8	28	56	70	56	28	8	1		
1	9	36	84	126	126	84	36	9	1	
1	10	45	120	210	252	210	120	45	10	1

Figure 2.9: Pascal's triangle

behavior: they differ from each other only by the bits $\{e_{j-1} \cdots e_1\}$ shown in bold. Further, these bits are generated in the same sequence as the vectors in the column $k-1$. This pattern holds true for any two successive columns in Figure 2.8.

We can then explore this number pattern in order to elaborate a recursive algorithm to evaluate the z^{th} vector in a column k. First, let us consider the following case of study: the 63 fault patterns shown in Figure 2.8 have to be produced by two FAULT GENERATORS. In this case, the first module is responsible for generating the first 32 vectors, and the second one for generating the other 31. Because of that, we have to find the 33^{th} vector $\mathbf{e}$ in order to initialize the second FAULT GENERATOR.

The first step of the proposed algorithm is to find the column of Figure 2.8 that contains the 33^{th} fault pattern $\mathbf{e}$. This can be easily obtained by using combinatorial functions, which indicates that this vector belongs to the third column ($C_1^6+C_2^6 < 33 < C_1^6+C_2^6+C_3^6$). Further, it can be seen that it corresponds to the 12^{th} vector $\mathbf{e}$ in the column $k=3$ because $33-(C_1^6+C_2^6)=12$.

Since the indices of the elements in Figure 2.8 start from 0, the first step is to use (2.3) to find which is the highest value of n such that $elem_{n_3} \leq 11$. In this case, this corresponds to $n=3 \rightarrow elem_{3_3}=10$. Then, repeating the same procedure for $k=2$ such that $elem_{n_2} \leq (11-10)$, it can be found $n=1 \rightarrow elem_{1_2}=1$. Notice that this recursive procedure is repeated until the sum of the elements $elem_{n_k}$ equals the index of the target vector. From these results, we can extract the amount of left-shift operations that must be performed for each bit '1'. In this case, we have to perform n left-shift operations in e_3 ($n=3 \rightarrow \mathbf{e}=$ **1**00011) and e_2 ($n=1 \rightarrow \mathbf{e}=$ 100**1**01). Therefore, the 33^{th} vector in Figure 2.8 is **1**00**1**01. Algorithm 2 illustrates the steps of calculating the z^{th} generated fault pattern.

Algorithm 2 Evaluating the z^{th} generated fault pattern

1: $k \leftarrow 1$;
2: SumCombination $\leftarrow C_k^w$;
3: **while** SumCombination $\leq z$ **do**
4: $k \leftarrow k + 1$;
5: SumCombination $\leftarrow$ SumCombination + $\mathrm{C_k^w}$;
6: **end while**
7: Index $= z -$ SumCombination;
8: **for** $i = 1$; $i \leq k$; $i = i + 1$ **do**
9: $\mathrm{bit}_i = 1 \ll (i - 1)$;
10: **end for**
11: **for** $i = k$; $k > 0$; $i = i - 1$ **do**
12: $kn \leftarrow 1$;
13: Calculate $elem_{i_{kn}}$;
14: **while** $elem_{i_{kn}} \leq$ Index **do**
15: $\mathrm{bit}_i = \mathrm{bit}_i \ll (1)$;
16: $kn = kn + 1$;
17: Calculate $elem_{i_{kn}}$;
18: **end while**
19: Index = Index $- elem_{i_{kn-1}}$;
20: **if** Index == 0 **then**
21: break;
22: **end if**
23: **end for**
24: FinalVector = 0;
25: **for** $i = 1$; $i \leq k$; $i = i + 1$ **do**
26: FinalVector = FinalVector $\vee \mathrm{bit}_i$;
27: **end for**

2.2.5 Results

The implementation cost of the proposed architecture was analyzed with a 74283 fast adder, using a DE2 FPGA board from Altera. For each implementation, a different number of parallel modules was considered, from 1 up to 8 copies. This is the maximum number of copies that could be synthesized on this device. The amount of resources used by these implementations is shown in Figure 2.10. It can be noted that the resources grow in an almost linear rate with respect to the number of copies of the DUT. Further, the amount of logic elements doubles only for $N = 5$, i.e. only about 20% of area overhead is necessary to add a parallel module for this case.

The performance of the FIFA parallel implementation was analyzed regarding the number of clock cycles required to evaluate the reliability of the 74283 fast adder. The effect of up to 5 simultaneous bit-flips into the circuit was considered. The simulation results are shown in Figure 2.11. Notice that the computation time is reduced by a factor very close

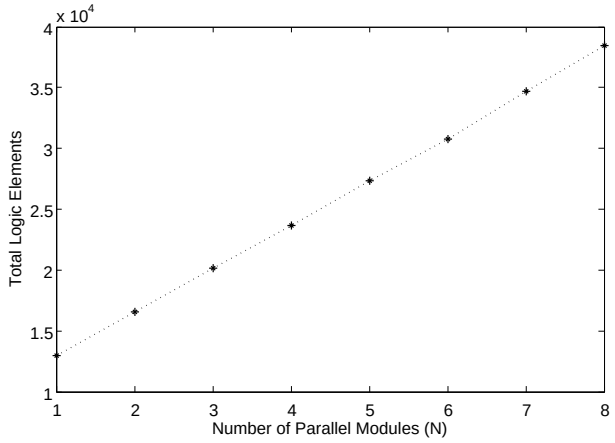


Figure 2.10: Total logic elements

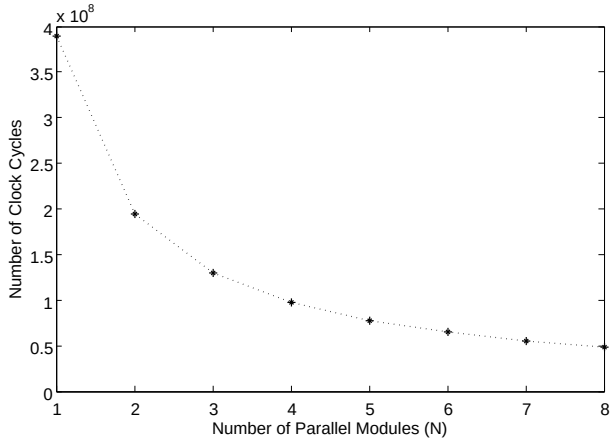


Figure 2.11: Number of clock cycles

to the theoretically value of $1/N$, what shows the efficiency of the proposed architecture.

2.2.6 Conclusion

This chapter presented a new tool based on fault injection for fault robustness analysis of digital circuits. The proposed tool, named FIFA, allows the designer to establish trade-offs between complexity and completeness of the analysis. The developed IP is fully parameterizable. Synthesis results have shown that it exceeds those reported in the literature in terms of area efficiency and performance. The FIFA tool deals with single/multiple simultaneous faults as well as permanent/transient faults. Moreover, if high fault multiplicity ($\mathbf{k}_a > \mathbf{k}_{max}$) is unlikely, all fault injections and tests related to $\mathbf{k}_a > \mathbf{k}_{max}$ can be avoided without diminishing accuracy in the analysis process.

In addition, an elaborate algorithm that calculates the z^{th} element in the fault pattern generation sequence was introduced. The corresponding sequence is of great interest for fault-injection tools because it can be used to inject faults according to their multiplicity.

The proposed algorithm was used to conceive a new parallel architecture for the FIFA tool. The algorithm is required in order to balance the amount of fault patterns among the parallel fault generators, a must for optimal parallel implementations. This architecture can reduce the calculation time by a factor of $1/N$, where N is the number of parallel modules. This shows that the parallel modules are well balanced, what is highly desired. At the same time, replications of fault-free DUTs are avoided, thus keeping the area overhead as low as possible. Indeed, extrapolating for very large circuits, the area overhead will converge to close to 50% of the original area per additional module.

Chapter 3

Effective Reliability

3.1 Introduction

As stated in Section 1.2, the reliability of a logic circuit is emerging as an important concern that may limit the benefits of technology scaling in nanometric dimensions [95–97]. The reliability of a circuit is a measure of its susceptibility to permanent, intermittent and transient faults [15]. Faults in integrated circuits can produce errors, but an error will not necessarily propagate to the final output of a circuit and produce a failure. Basically, three different kinds of masking effect can avoid the propagation of an error:

- *Logical masking*: occurs when the propagation of a fault is blocked by a subsequent logical gate whose output is completely determined by the other input. An example of logical masking can be seen in Figure 3.1.
- *Electrical masking*: occurs when a fault is attenuated during its propagation so that it does not have enough duration/amplitude to affect the result of the circuit.
- *Temporal masking or latching-window masking*: occurs when the current/voltage transient generated by a fault reaches the input of a synchronous cell outside its storage window (not at the clock transition).

These three mechanisms are directly related to the capacity of a circuit to tolerate faults. However, the downscaling of electronics is reducing both its electrical and temporal masking abilities [95, 98]. On the other hand, the technology scaling is not affecting the logical masking ability because it depends only on the topology of the logical circuit [8]. Indeed, several fault-tolerant approaches are based on increasing the logical masking ability of a circuit so that a fault will not reach its final output. These fault-tolerance techniques generally bring some design penalties such as area, cost, and performance. In spite of that, other potentially fault masking capabilities can be derived from the target application and were not extensively explored. In fact, the error impact on a circuit output is conditioned by the usage of its results. This can lead to the existence of errors that can be accept-

able/bearable for a specified application. In other words, certain applications can tolerate small errors, and we can explore this fact to improve the resulting design. In the rest of the current work, this phenomenon will be referred as **application masking** of errors, which is closely related to the concept of error tolerance.

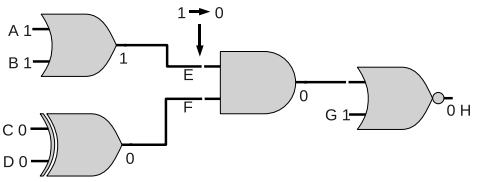


Figure 3.1: Example of a logical masking. Notice that when F assumes the value '0' and occurs a SET in E, the output H rests unchanged and is not affected by this error.

3.2 Error tolerance

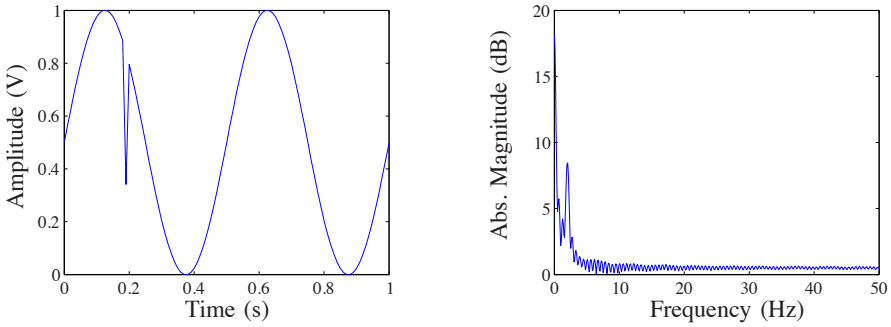
The drastically increase in the number of soft and permanent faults expected with technology scaling has inspired the discussion about error tolerance since the release of the 2001 *International Technology Roadmap for Semiconductors* (ITRS). This report states that: "Relaxing the requirement of 100% correctness for devices and interconnects may dramatically reduce costs of manufacturing, verification, and test. Such a paradigm shift is likely forced in any case by technology scaling, which leads to more transient and permanent failures of signals, logic values, devices, and interconnects." [99]. This is done as a means to reduce the cost of the final design since this has been the greatest threat to the continuation of the semiconductor roadmap. This menace continues to be true in the ITRS 2011, and it's the principal message of the last ITRS Design Report [4].

The error tolerance concept was first introduced as an application-oriented paradigm to deal with process variations, defects, and noise in [7]. The main idea is that some applications such as audio and video, have the ability to tolerate certain types of errors as long as they are restricted to a certain level of error severity given by the target application. For instance, most of multimedia applications inherently have this ability because of the functioning of human's senses such as sight, hearing, and smell, which can mask the effect of some errors.

Application masking of errors, also called application-level resilience by [83], has been extensively researched over the past years [7, 83, 86, 100]. For instance, a methodology to analyze the error tolerance of applications was proposed in [100]. In this work, the authors have analyzed the impact of errors in the quality of the audio signal provided by a digital telephone-answering device. Then, they have investigated the correlation between

the position of an erroneous output bit and the corresponding impact on speech quality. The results have shown that the contribution of each output bit can be different, and that errors below a specified threshold are acceptable. A study of the resilience of a JPEG compressor to errors was carried out in [86]. In this study they proposed some models to calculate the error severity to the JPEG compressor as well as a method to selectively harden such circuit.

Another element that can influence the error tolerance of a given application is the approach of coding used to represent the desired information. As a matter of fact, the impact of an erroneous output bit also depends on its weight, i.e. its significance relative to the output word. In order to demonstrate that, let us consider the output of an audio signal in which an SEU occurred in a sample as shown in Figure 3.2(a) [101]. It can be noted that the error has a pulse characteristic in time domain, and that the amplitude of such pulse depends on the bit position. Consequently, a bit-flip presents a white noise behavior in the frequency domain, and its power also depends on the erroneous bit position as shown in Figure 3.2(b). This means that the amount of white noise added by the occurrence of a bit-flip depends on the bit position, and since the addition of a small quantity of noise can be masked by the human hearing, some errors can be acceptable/bearable for an audio application. In fact, the most significant bits carry more information about the signal, therefore they deserve special attention. Errors presented in the less significant bits may be even acceptable.



(a) Bit-flip occurrence at b_{M-1}

(b) Frequency representation of the signal with error

Figure 3.2: Bit-flip occurrence in a sine wave representing an output of a circuit

In spite of that, most of testing and reliability evaluation techniques analyze the output of a circuit in a pass versus fail paradigm. By doing that, they ignore the real impact of an error on the final result, which can lead to the conception of suboptimal designs.

This chapter introduces a new model for reliability calculation named **effective reliability** [101, 102]. Unlike the traditional concept for reliability evaluation, effective reliability takes into consideration application-specific characteristics to give us a fairly value

for the reliability of a circuit. Indeed, using pertinent quality metrics, it can evaluate the reliability of a circuit considering that errors below a specified threshold are acceptable. Therefore, effective reliability is not based only on fault tolerance, but also makes use of the error tolerance concept [7]. In addition, two metrics are proposed in order to calculate the effective reliability of a circuit based on bit significance and relative error rules.

3.3 Effective reliability

Let $\mathbf{y}_i = b_{M-1}b_{M-2}\cdots b_1b_0$ be defined as a vector of M bits that represents the output of a circuit. In this case, bit b_0 stands for the LSB (Least Significant Bit). Also, let us define the reliability of bit b_i as q_i. Considering an application with independent outputs, the nominal reliability can be evaluated by (3.1) [54].

$$R_{nom} = \prod_{i=0}^{M-1} q_i \tag{3.1}$$

It can be noted that the nominal reliability concept does not differentiate the impact of each output bit to the final reliability of the circuit. This means that such concept is based on a pass versus fail paradigm, and therefore does not consider any kind of error tolerance. In order to allow that, we propose a model for reliability calculation which takes into account the effective impact of errors on the target application (see Figure 3.3). This new concept, named **effective reliability**, can be evaluated by (3.2) or (3.3)

$$R_{eff} = R_{nom} + R_{ack} \tag{3.2}$$

$$R_{eff} = 1 - R_{\overline{ack}} \tag{3.3}$$

where R_{ack} is the probability of errors being masked according to the application, that is, errors that are acceptable for the target application are neglected, and $R_{\overline{ack}}$ is the probability of errors not being masked according to the application. Notice that two different terms are presented in (3.2): R_{nom} and R_{ack}. The first one is related to the logical masking ability of the design, while the latter one is related to its application masking ability. The general procedure to evaluate the effective reliability of a circuit is represented in Figure 3.3.

One important characteristic of the effective reliability concept is that it is based on the assumption that for a given application, errors can be classified into two categories: critical and noncritical. In this context, an error is considered critical if it significantly impacts the quality of the result, or if it directly causes a circuit malfunctioning. Because of that, these errors cannot be accepted. On the other hand, noncritical errors have no

significant impact on the output such that they cannot be perceived by the final user (they can be tolerated).

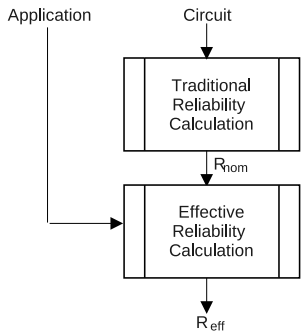


Figure 3.3: General scheme for reliability calculation of a bit-vector from a logic circuit that supplies a target application circuit

The evaluation of R_{ack} or $R_{\overline{ack}}$ is performed according to quality metrics that are considered pertinent from the application's point of view. For example, due to physical mechanisms presented in human ear, the frequency response of our auditory system is not linear. As a result, we have a better perception at some frequencies than others; and we can use this to define a set of errors that are acceptable for our application. Exploiting this fact, we notice that $R_{eff} \geqslant R_{nom}$. Thus, reliability constraints for the logic circuit can be relaxed so that area over cost can be minimized.

The next Section presents two quality metrics that can be used to evaluate the effective reliability of a circuit.

3.4 Quality metrics

3.4.1 Definitions

Let us suppose that the output of a specific circuit is used to control an application. In this context, each circuit output result is represented by $\mathbf{y_i}$, defined in Section 3.3. Considering the occurrence of k simultaneous errors, there are $C_k^M = \binom{M}{k}$ different situations concerning the locations (indices) of the faulty bits in $\mathbf{y_i}$. For instance, if $M = 4$ and $k = 3$, we have 4 different situations ($C_3^4 = 4$) for the occurrence of 3 simultaneous errors in $\mathbf{y_i}$: $\{b_2b_1b_0, b_3b_1b_0, b_3b_2b_0, b_3b_2b_1\}$. Based on that, we can define the following elements:

- $\mathbf{w}^{k\times 1}$ as a column vector that represents the indices of the k faulty bits in $\mathbf{y_i}$.
- $\mathbf{E}^{k\times C_k^M}$ as a matrix created by the concatenation of all possible $\mathbf{w}^{k\times 1}$ vectors for a specified k (see Figure 3.4 for an example). Each element of $\mathbf{E}^{k\times C_k^M}$ is represented by $e_{i,r}$.

– $\gamma_{k,r}$ as the probability of occurrence of k errors in $\mathbf{y_i}$ distributed according $\mathbf{w}_r$. This parameter can be evaluated by (3.4).

$$\boldsymbol{E}^{k\,x\,C_k^M} = [\boldsymbol{w}_1 \ldots \boldsymbol{w}_{C_k^M}] = \overbrace{\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 2 \\ 1 & 1 & 2 & 2 & 1 & 2 & 3 & 2 & 3 & 3 \\ 2 & 3 & 3 & 3 & 4 & 4 & 4 & 4 & 4 & 4 \end{bmatrix}}^{C_k^M = C_3^5 = 10} \Big\} k=3$$

$$\underbrace{\qquad\qquad}_{C_k^{M-1} = C_3^4 = 4} \qquad \underbrace{\qquad\qquad\qquad}_{C_{k-1}^{M-1} = C_2^4 = 6}$$

Figure 3.4: Example of matrix $\mathbf{E}$ considering 3 errors in $\mathbf{y}_i = b_4 b_3 b_2 b_1 b_0$

$$\gamma_{k,r} = R_{nom} \cdot \prod_{i=1}^{k} \left(\frac{1}{q_{e_{i,r}}} - 1 \right) \tag{3.4}$$

In the next Section these elements will be used to define quality metrics to evaluate the effective reliability of a system.

3.4.2 Quality metric 1: bit significance

The first quality metric proposed in the current work is based on bit significance. In this case, we consider that any faulty bit located below a specified position is acceptable. This threshold value, represented by T, depends on the target application. For example, if we consider $T = 2$ for a specific application, it means that any faulty bit located in b_2, b_1 and/or b_0 can be tolerated. With this in mind, we can now evaluate R_{ack} and $R_{\overline{ack}}$ by (3.5) and (3.6), respectively. Moreover, the effective reliability can be calculated using (3.7) or (3.8).

$$R_{ack} = \sum_{k=1}^{T+1} \sum_{r=1}^{C_k^{T+1}} \gamma_{k,r} \tag{3.5}$$

$$R_{\overline{ack}} = \sum_{k=1}^{T+1} \sum_{r=C_k^{T+1}+1}^{C_k^M} \gamma_{k,r} + \sum_{k=T+2}^{M} \sum_{r=1}^{C_k^M} \gamma_{k,r} \tag{3.6}$$

$$R_{eff} = \prod_{i=0}^{M-1} q_i + \sum_{k=1}^{T+1} \sum_{r=1}^{C_k^{T+1}} \gamma_{k,r} \tag{3.7}$$

$$R_{eff} = 1 - \sum_{k=1}^{T+1} \sum_{r=C_k^{T+1}+1}^{C_k^M} \gamma_{k,r} - \sum_{k=T+2}^{M} \sum_{r=1}^{C_k^M} \gamma_{k,r} \tag{3.8}$$

3.4.3 Quality metric 2: relative error

Relative error refers to an evaluation of a difference between two measures normalized with respect to the true measure. One approach to calculate the relative error is shown in (3.9), where $\mathbf{y}_i$ is the correct measure and $\tilde{\mathbf{y}}_i$ is the erroneous measure.

$$\delta = \frac{|\tilde{\mathbf{y}}_i - \mathbf{y}_i|}{\mathbf{y}_i} \tag{3.9}$$

Using this concept, we can define a maximum acceptable value for the relative error (δ_{max}) based on the target application. In order to do that, let us first suppose a system in which the input word comprises a vector of H bits. Considering that $p(a)$ represents the probability of the input to assume a value a, R_{ack} and $R_{\overline{ack}}$ can be evaluated according to (3.10) and (3.11), respectively,

$$R_{ack} = \sum_{a=0}^{2^H-1} p(a) \sum_{k=1}^{k_{max}} \sum_{r=1}^{C_k^M} \gamma_{k,r}.u\left(\delta_{max} - \delta(k,r,a)\right) \tag{3.10}$$

$$R_{\overline{ack}} = \sum_{a=0}^{2^H-1} p(a) \sum_{k=1}^{k_{max}} \sum_{r=1}^{C_k^M} \gamma_{k,r}.u\left(\delta(k,r,a) - \delta_{max}\right) \tag{3.11}$$

where k_{max} represents the maximum number of simultaneous errors considered, $u(t)$ is a step function centered at the origin, and $\delta(k,r,a)$ is evaluated according (3.9). The erroneous measure $\tilde{\mathbf{y}}_i$ in such expression is obtained considering the occurrence of k simultaneous errors distributed according to $\mathbf{w}_r$.

Finally, we can evaluate the effective reliability using (3.12) or (3.13).

$$R_{eff} = \prod_{i=0}^{M-1} q_i + \sum_{a=0}^{2^H-1} p(a) \sum_{k=1}^{k_{max}} \sum_{r=1}^{C_k^M} \gamma_{k,r}.u\left(\delta_{max} - \delta(k,r,a)\right) \tag{3.12}$$

$$R_{eff} = 1 - \sum_{a=0}^{2^H-1} p(a) \sum_{k=1}^{k_{max}} \sum_{r=1}^{C_k^M} \gamma_{k,r}.u\left(\delta(k,r,a) - \delta_{max}\right) \tag{3.13}$$

Notice that both expressions (3.12) and (3.13) only provide the same effective reliability value when considering $k_{max} = M$ simultaneous errors. This task is computationally intensive and may be intractable for large circuits. However, this consideration may be too pessimistic for the specified application. The analysis of both expressions considering different values of k_{max} can give us a possible solution to tackle this problem. As a matter of fact, expressions (3.12) and (3.13) provides a different pair of values for each $k_{max} < M$. The first one gives us a pessimistic value for effective reliability and the other one an optimistic value. The difference between these two results reduces as k_{max} approximates

M. Therefore, expressions (3.12) and (3.13) can be used to produce boundaries in order to estimate the effective reliability. When both results are closer enough from each other, we can stop the calculation and estimate the effective reliability for the target application.

3.5 Simulation results

For illustration of the proposed approach, let us consider the design of three circuits commonly used in digital signal processing applications: a median filter, an 8-bit ripple carry adder (CRA8), and a 4-bit multiplier (MUL4). For the median filter, the bit significance metric will be used, while for the two latter the relative error metric is considered.

3.5.1 Median filter

In the field of image processing, a set of noise reduction algorithms is often required. These algorithms generally present different properties so that one algorithm is more suitable to deal with one type of noise than another. In particular, the median filter is a nonlinear digital filtering technique commonly used because of its great performance when coping with "speckle" and "salt and pepper" noise.

Let us suppose for our case of study that we have to implement a median filter to process images with the following requirements:

- Each image has 256×256 pixels varying from 0 to 255;
- The system must have an 8-bit input and an 8-bit output;
- The filter's reliability must be higher than 95%.

Let us now consider that the filter architecture we chose led us to an implementation in which each output bit has a reliability of 99%. In the following, we can analyze both the traditional and the proposed reliability evaluation methods as following:

3.5.1.1 Traditional reliability calculation

The traditional reliability calculation that considers independent outputs can be evaluated by (3.14).

$$R_{nom} = \prod_{i=0}^{M-1} q_i = 92.27\% \tag{3.14}$$

In this case, it can be seen that the reliability requirement was not fulfilled by our median filter. Therefore, procedures must be carried out to solve this problem. In order to respect the reliability requirement stated above, a possible solution is to make use of

redundancy techniques to increase the reliability of the median filter. Using that, the system will fit the reliability requirement in exchange for area, cost, and power increase.

Nevertheless, a careful analyze of the median filter application puts in evidence that this problem could be treated from another point of view. Using the concept of nominal reliability we do not take into account any application-specific characteristic in order to evaluate the reliability of the system. However, in systems such as the one used in our case study, small errors can be tolerated without compromising the system performance. Unlike the traditional method for reliability evaluation, effective reliability can take into consideration such characteristics. Therefore, it may give us a fairly evaluation for the reliability of this circuit with respect to a pertinent quality metric.

3.5.1.2 Effective reliability calculation

As a first step to evaluate the effective reliability for our case of study, we need to define an acceptable quality metric from the application's point of view. In this case, we chose the bit significance approach discussed in Section 3.4.2. The next step is to evaluate the effective reliability for different error tolerances. As shown in Table 3.1, we can then be aware of the minimum error tolerance value necessary to meet the reliability requirement. Notice that one important contribution of the effective reliability concept is to provide a reliability value that depends on the application masking ability of the circuit. Indeed, the reliability of a circuit is now characterized not only by the presence of errors in the output of a circuit, but with the usage profile of the results provided by the circuit. By doing that, the reliability of a given circuit 'A' may be higher for one application than for another.

Table 3.1: Effective reliability evaluation for different error tolerances

T (bit)	$\mathbf{R_{eff}}$
b_0	93.21%
b_1	94.15%
b_2	95.01%
b_3	96.06%
b_4	97.03%
b_5	98.01%
b_6	99.00%
b_7	100.00%

According to Table 3.1, if our application can tolerate any faulty bit localized between b_0 and b_2 (i.e., $T = 2$), the effective reliability of this system is 95.01%. Therefore, the project requirements are already fulfilled and we do not need to add extra hardware to

improve the reliability of the median filter. In order to check if this error tolerance is acceptable for our system, we performed a subjective analyze relative to noisy pictures. Each picture was contaminated with gaussian noise using the required error tolerance range (faulty bits localized between b_0 and b_2). An example of the well-known "Lena" used in this comparison procedure can be seen in Figure 3.5. As can be noted, the noise is almost unperceived by humans, and then can be tolerated by our application. As a result of that, we can conclude that the effective reliability of our system is already greater than 95%, fitting well the specified requirements, and that no extra hardware is required.

(a) Original "Lena" (b) Noisy "Lena"

Figure 3.5: Comparison between the original and the noisy "Lena" pictures

3.5.2 8-bit ripple carry adder

The ripple carry adder circuit (CRA8) was constructed by cascading 8 FA blocks (see Figure 3.6), where each logic gate is supposed to have reliability 99.9%. Furthermore, let us suppose that the CRA8's reliability must be higher than $R_{min} = 95\%$ and that the target application can accept/tolerate errors as high as 2% of the correct result ($\delta_{max} = 2\%$).

The reliability of the output bits of the CRA8 circuit can be evaluated by using the SPR-MP technique proposed in [15] and explained in Section 1.2. The results are shown in Table 3.2.

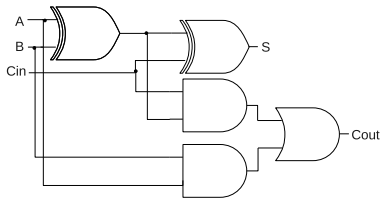


Figure 3.6: Structure of FA (full adder) block

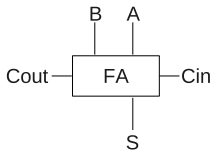


Figure 3.7: Schema of FA block

Table 3.2: Reliability values for the output bits of the 8-bit full adder (CRA8)

Output	Reliability (q_i)
b_0	99.80%
b_1	99.48%
b_2	99.31%
b_3	99.24%
b_4	99.20%
b_5	99.18%
b_6	99.17%
b_7	99.16%
b_8(carry)	99.36%

3.5.2.1 Design based on nominal reliability

Using the traditional reliability concept defined in Eq. 3.1 and values in Table 3.2, the reliability for CRA8 is calculated as shown in (3.15). This analysis leads the designer to assume that the specifications are not met, and that the use of a fault tolerance technique is necessary.

$$R = \prod_{i=0}^{8} q_i = 94.06\% \tag{3.15}$$

Let us consider TMR (Triple Modular Redundancy) as the fault-tolerance technique to be applied in order to improve the reliability of this circuit [23]. TMR corresponds to the triplication of a module (in this case, a FA) and the selection of one among the three outputs according to a majority vote. Triplication of a FA implies the use of two voters (one for sum and another for the carry bit). Considering that the relationship between the voter's area (S_V) and the FA's area (S_{FA}) are such that $S_{FA} = 2 \times S_V$, applying TMR to a FA implies in triplicating the required area for this block. If we analyze all the possible architectures with FA modules triplicated, the configuration that can meet the reliability

Table 3.3: R_{eff} for different error tolerances (CRA8)

Relative Error (δ_{max})	Reliability
0.5%	94.23%
1.0%	94.64%
1.5%	94.96%
2.0%	95.22%
2.5%	95.44%
3.0%	95.62%
3.5%	95.77%
4.0%	95.92%
4.5%	96.05%
5.0%	96.16%

requirement while still minimizing the area overhead is obtained by protecting two FAs with TMR. In this case, the area overhead is 75%. Let us now analyze the same circuit by using the effective reliability concept.

3.5.2.2 Design based on effective reliability

Differently of nominal reliability, effective reliability takes into account the usage of the results produced by the circuit when evaluating its reliability. The main idea is to define a threshold based on a pertinent quality metric that can be used to classify errors into critical and noncritical. In order to do that, let us consider the relative error metric stated in Section 3.4.3. The equations to evaluate the effective reliability value for a given error tolerance (δ_{max}) are given in (3.2) and (3.10). We have evaluated the effective reliability for different values of δ_{max} as a mean to illustrate the relationship between this value and the error tolerance ability of the target application. The obtained results are shown in Table 3.3. Given that the target application tolerates errors as high as 2%, we notice that the reliability is over than 95%. Therefore, our system has already met the specifications and no TMR is required.

3.5.3 4-bit multiplier

The MUL4 was constructed using the architecture shown in Figure 3.8. In this case, each full-adder FA was built according Figure 3.6 and Fig 3.7. Let us now consider that this project has a reliability requirement of $R_{min} = 90\%$, and that the target application can tolerate errors as high as 5% ($\delta_{max} = 5\%$).

The reliability of the output bits of the MUL4 circuit can be evaluated by using the

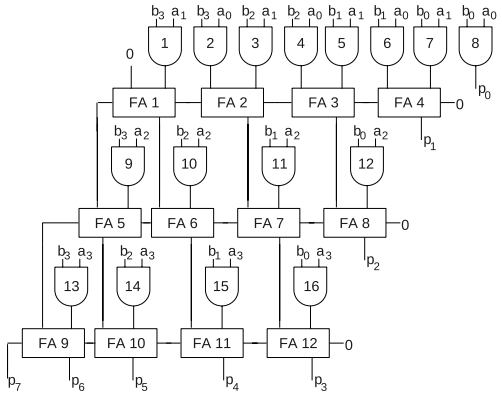


Figure 3.8: Structure of a 4-bit multiplier block

Table 3.4: Reliability values for each output bit of the 4-bit multiplier

Output	Reliability
p_0	99.90%
p_1	99.70%
p_2	99.25%
p_3	98.43%
p_4	97.55%
p_5	97.30%
p_6	97.89%
p_7	99.16%

SPR-MP technique [15]. In order to do that, we have assumed that the reliability of a logic gate is $q = 99.9\%$. The corresponding results are shown in Table 3.4. Let us now analyze the results by using the nominal reliability and the effective reliability concepts.

3.5.3.1 Design based on nominal reliability

Using the values presented in Table 3.4, we can evaluate the reliability according the traditional concept as shown in (3.16). In this case, the designer is also leaded to assume that the specifications were not met, and that the use of a fault tolerance technique is necessary.

$$R_{mul} = \prod_{i=0}^{7} q_i = 89.65\% \tag{3.16}$$

Next, we have analyzed the use of the TMR technique to improve the multiplier reli-

Table 3.5: R_{eff} for different error tolerances (MUL4)

Relative Error (δ_{max})	Reliability
0.5%	89.65%
1.0%	89.67%
1.5%	89.71%
2.0%	89.75%
2.5%	89.81%
3.0%	89.87%
3.5%	89.92%
4.0%	89.98%
4.5%	90.07%
5.0%	90.12%

ability such that the expected requirements are met. Every component, AND gates and FAs shown in Figure 3.8, was considered as a possible module where we might apply TMR. The first step was to analyze all possible architectures using TMR in one module. In this case, there was no architecture that could fit the reliability requirement ($R_{min} = 90\%$). However, by applying TMR into two modules, 27 different architectures can reach R_{min}. Among them, we chose the one with less area overhead. This is obtained applying TMR into gates AND1 and AND16. Assuming that all logic gates have the same area, the total area overhead for this architecture is 11.84%. Let us now consider the analysis by using the effective reliability concept.

3.5.3.2 Design based on effective reliability

We evaluated the effective reliability for different error tolerances as shown in Table 3.5. Analyzing the results presented in this Table, we can conclude that the reliability of this circuit is over 90% if it can tolerate error as high as $\delta_{max} \geq 4.5\%$. Therefore, assuming the example presented in this Section, which $\delta_{max} = 5\%$, the multiplier has already met the required specifications without using any fault tolerance technique.

3.6 Conclusion

In this chapter we have proposed a new concept for reliability evaluation named *effective reliability*. Unlike the traditional concept for reliability evaluation, effective reliability can take into account specific characteristics of the target application to give us a fairly value for the reliability of a circuit. Indeed, using pertinent quality metrics, it evaluates the reliability of a circuit considering that errors below a specified threshold are acceptable.

Therefore, effective reliability is not based only on fault tolerance, but also make use of the error tolerance concept.

In addition, two important quality metrics were also proposed. Both, bit significance and relative error metrics, may be used in a range of digital signal processing applications.

Simulation results were presented in Section 3.5. Three circuits often used in digital signal processing applications were considered as the case of studies in order to demonstrate the importance of the proposed concept. In fact, when we do not consider the application when evaluating the reliability of a circuit, the calculated value can be too pessimistic. Then, unnecessary procedures may be carried out in order to fit the reliability requirement for a specific project, increasing cost and area penalty.

Chapter 4

Selective Hardening

4.1 Introduction

As discussed earlier, selective hardening techniques offer a good compromise between reliability improvement and area overhead. Based on that fact, this chapter introduces two different approaches to apply selective hardening in integrated circuits. The first one, introduced in Section 4.2, is based on the fact that errors may have different consequences for different applications. For instance, in a binary output word, errors located in the most significant bits tend to be more critical than errors located in the least significant bits. Therefore, the proposed technique drives the effort of reliability improvement to the bits that will most impact the output of a circuit. In addition, a metric that allows to assign different weights to different output bits of a system, named practical reliability, is also introduced. The second approach, introduced in Section 4.3, uses a parameter similar to a hardening cost in order to allow designers to drive the methodology using accurate cost values for hardening each gate [103]. Further, two heuristics are proposed as a means to determine when selective hardening is no longer feasible.

4.2 Avoiding Critical Errors in Integrated Circuits

4.2.1 Nominal reliability

Let $\mathbf{y} = b_{M-1}b_{M-2}\cdots b_1b_0$ be a vector of M bits representing the output of a circuit. The nominal reliability [54, 101] of a circuit is defined as the probability that it produces correct outputs, i.e., the probability that all $b_i \in \mathbf{y}$ are correct 0(s) and 1(s). Considering that the output bits are independent, this value is conventionally expressed as in (4.1), where R_i stands for the reliability of b_i.

Table 4.1: Reliability for the output bits of three different architectures of a 4-bit adder

Architecture	b_3	b_2	b_1	b_0	$R_{nominal}$	$R_{practical}$
1	99%	99%	99%	95%	92.18%	97.63%
2	95%	99%	99%	99%	92.18%	94.17%
3	98%	99%	99%	95%	91.25%	96.64%

$$R_{nominal} = \prod_{i=0}^{M-1} R_i \tag{4.1}$$

Let us now suppose that the circuit's output is coded using a binary scheme such that b_{M-1} and b_0 stand for the Most Significant Bit (MSB) and the Least Significant Bit (LSB), respectively. Error(s) occurring in MSB(s) will result in more remarkable disparities than in any other bit. By contrast, errors in LSB(s) may even be masked by the target application [101]. In spite of that, nominal reliability assigns equal reliability costs to the output bits as can be seen in (4.1). In fact, two different architectures of a logic function may have the same nominal reliability value, but one may still be more likely to provide results with greater disparities than the other. For instance, let us suppose that a designer obtained three different architectures of a 4-bit adder and he has to select one of them by comparing theirs reliability values. Table 4.1 illustrates the values of the reliability of the output bits of such architectures.

Analyzing the nominal reliability values for the obtained architectures, *Architecture 1* and *Architecture 2* are selected as the best solutions. Indeed, no distinction can be made between these two architectures regarding the nominal reliability value. However, if the output of this circuit is coded using a binary scheme, it is more likely that the first architecture will provide better results (smaller disparities) than will the second. Ideally, the reliability analysis should take into account the amount of information each bit of an output carries (or its importance) in order to assign progressively great costs to them. In order to tackle this problem, a new metric to analyze the reliability of a circuit is presented in Section 4.2.2.

4.2.2 Practical reliability

Practical reliability is a metric that can take into account the importance of each output bit of a circuit. It can be evaluated as shown in (4.2). The weight factor k_i allows a designer to control the importance of a specific output bit b_i to the output of the circuit. Notice that if $k_i = 1$ for all $0 \leq i \leq M-1$, the practical reliability expression (4.2) becomes

the nominal reliability expression (4.1). Also, by setting $k_i = 0$, the metric does not take into account errors in b_i. This is very useful to circuits that can tolerate some errors (application masking [101] or application-level resilience [83]). If a simple standard binary representation is considered, then k_i can be calculated as shown in (4.3).

$$R_{practical} = \prod_{i=0}^{M-1} R_i^{k_i} \tag{4.2}$$

$$k_i = \frac{1}{2^{(M-1)-i}} \tag{4.3}$$

Although the proposed metric does not evaluate the true reliability of a circuit, it takes into account both the reliability and the importance of an output bit to the system and merge these information in a single number to simplify the analysis. For instance, let us analyze the architectures shown in Table 4.1. It can be noted that the practical reliability values are different from the values obtained with nominal reliability. Actually, even the order of the best architectures changes with the proposed metric. *Architecture 2*, which before was considered the best architecture together with *Architecture 1*, now is considered as the worst choice. This is due to the low reliability value of its MSB. In fact, practical reliability "punishes" architectures that present low reliability in critical bits, thus providing a more realistic result for a given target application. This metric will be used in Section 4.2.3 as the basis for a method to selectively apply TMR into a circuit.

4.2.3 Selectively applying TMR

Although TMR can provide a great level of protection against faults, the area overhead required by such technique is quite high. To diminish this drawback, this chapter proposes a method to selectively apply TMR to digital circuits. The main idea is to rank gates or blocks to be protected based on critical factors. In the current work, a critical factor takes into account not only the probability that an error will be introduced by a gate, but also how critical this error will be for the target application as will be seen in Section 4.2.3.2.

4.2.3.1 Case study

In order to explain and validate the proposed method, a 4-bit fast adder (*74283*) is used. This fast adder is illustrated in Figure 4.1.

4.2.3.2 Identifying critical gates

Selective TMR is realized by the classification of constituent gates of a circuit [104]. The fast adder has 9 inputs, 5 outputs, and is composed of 40 logic gates. All gates are

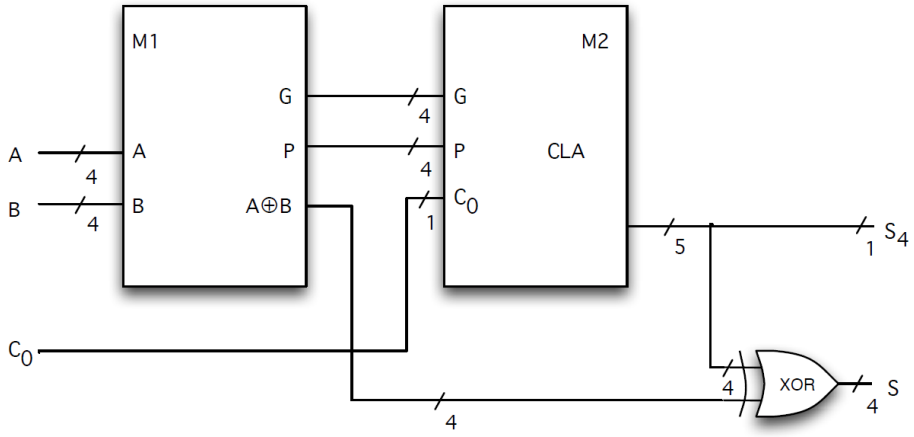


Figure 4.1: 4-bit fast adder circuit

considered as being fault-prone. Further, it is considered that these blocks (g_i ($i \in [0, 39]$)) are independent, and that they are labeled as shown in Figure 4.2.

The procedure to detect which are the critical gates of this circuit takes two steps: first, a fault emulation tool, named FIFA and described in [105], is used to inject bit-flip faults due to radiation effects; next, **critical gates** are detected by analysis of errors that appeared in the outputs.

In the following work it is considered only the occurrence of single faults so that the tool injects just one per clock cycle. If the occurrence of multiple simultaneous faults is likely, the tool can be configured to deal with that.

Finally, the results produced by the original and the faulty circuits are compared bit by bit. If these results are different, it is concluded that the injected fault has been propagated to the output bits. Otherwise it is concluded that the fault has been masked.

The fault injection emulation is performed in order to detect the critical factors. The idea is to inject a single fault in a gate g_i and analyze the output for all possible input vectors. Then, for each output bit b_z, the number of errors S_z related to a single fault in g_i is evaluated (see Table 4.2). The columns S_{z_w} correspond to weighted versions of S_z. The issue is to define proper weights so that S_{z_w} reflects the relevance of each gate to each output.

In the case of the adder circuit, as the output is given as a binary number, S_{z_w} is obtained as shown in (4.4). Notice that there are 2^9 possible input logic values for each faulty gate. All the emulation results for the adder circuit are shown in Table 4.2.

$$S_{w_z} = 2^z \cdot S_z \tag{4.4}$$

The **critical gates** are detected according to the results presented in Table 4.2. Notice that the rightmost column in Table 4.2 gives the critical factor for a gate g_i. The higher

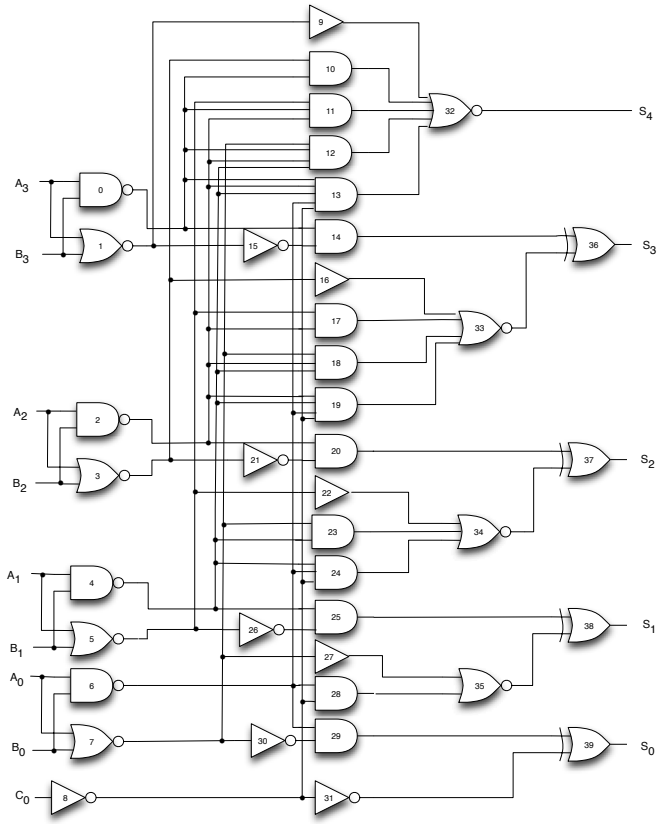


Figure 4.2: 74283 gate-level schematic

Table 4.2: Error analysis for the gates of the circuit 74283

g_i	S_0	S_{0_w}	S_1	S_{1_w}	S_2	S_{2_w}	S_3	S_{3_w}	S_4	S_{4_w}	$\sum e_w$	$CritFac$
0	0	0	0	0	0	0	384	3072	192	3072	6144	36
1	0	0	0	0	0	0	384	3072	320	5120	8192	38
2	0	0	0	0	384	1536	192	1536	96	1536	4608	33
3	0	0	0	0	384	1536	320	2560	160	2560	6656	37
4	0	0	384	768	192	768	96	768	48	768	3072	25
5	0	0	384	768	320	1280	160	1280	80	1280	4608	32
6	384	384	192	384	96	384	48	384	24	384	1920	14
7	384	384	320	640	160	640	80	640	40	640	2944	23
8	512	512	256	512	128	512	64	512	32	512	2560	22
9	0	0	0	0	0	0	0	0	320	5120	5120	35
10	0	0	0	0	0	0	0	0	288	4608	4608	34
11	0	0	0	0	0	0	0	0	272	4352	4352	31
12	0	0	0	0	0	0	0	0	264	4224	4224	29
13	0	0	0	0	0	0	0	0	272	4352	4352	31
14	0	0	0	0	0	0	512	4096	0	0	4096	27
15	0	0	0	0	0	0	384	3072	0	0	3072	24
16	0	0	0	0	0	0	320	2560	0	0	2560	21
17	0	0	0	0	0	0	288	2304	0	0	2304	20
18	0	0	0	0	0	0	272	2176	0	0	2176	18
19	0	0	0	0	0	0	288	2304	0	0	2304	20
20	0	0	0	0	512	2048	0	0	0	0	2048	17
21	0	0	0	0	384	1536	0	0	0	0	1536	13
22	0	0	0	0	320	1280	0	0	0	0	1280	12
23	0	0	0	0	288	1152	0	0	0	0	1152	10
24	0	0	0	0	320	1280	0	0	0	0	1280	12
25	0	0	512	1024	0	0	0	0	0	0	1024	7
26	0	0	384	768	0	0	0	0	0	0	768	6
27	0	0	320	640	0	0	0	0	0	0	640	4
28	0	0	384	768	0	0	0	0	0	0	768	6
29	512	512	0	0	0	0	0	0	0	0	512	2
30	384	384	0	0	0	0	0	0	0	0	384	0
31	512	512	0	0	0	0	0	0	0	0	512	1
32	0	0	0	0	0	0	0	0	512	8192	8192	39
33	0	0	0	0	0	0	512	4096	0	0	4096	27
34	0	0	0	0	512	2048	0	0	0	0	2048	15
35	0	0	512	1024	0	0	0	0	0	0	1024	8
36	0	0	0	0	0	0	512	4096	0	0	4096	28
37	0	0	0	0	512	2048	0	0	0	0	2048	16
38	0	0	512	1024	0	0	0	0	0	0	1024	9
39	512	512	0	0	0	0	0	0	0	0	512	3

Table 4.3: Reliability Analysis of 74283

Reliability	No hardening	Method in [18]	Proposed Method
S_0	94.07%	94.97%	94.07%
S_1	92.39%	93.26%	92.39%
S_2	91.80%	92.65%	92.43%
S_3	91.33%	92.17%	93.07%
S_4	94.60%	95.51%	97.15%
$R_{nominal}$	68.93%	72.24%	72.63%
$R_{practical}$	87.29%	88.89%	90.65%

the factor number is, the more critical the gate will be.

In fact, critical factors are assigned to the gates according to the number of weighted errors in Table 4.2. If the number of weighted errors equals, gates that are nearer the primary outputs receive higher priorities. If the number of weighted errors and the distance to the primary outputs are both identical, gates presenting more reconvergent fanouts are considered more critical. Gates for which these three parameters are equal receive the same critical factor.

4.2.3.3 Reliability analysis and comparison

Subsequent to classifying the critical gates, the reliability of the circuit is evaluated using the SPR analysis [15], which was explained in Section 1.2.3. Let us now consider TMR as the chosen redundancy technique to harden a gate, and that the area overhead constraint allows a designer to protect up to 5 gates. According to the critical factors presented in Table 4.2, gates g_{32}, g_1, g_3, g_0 and g_9 are selected by the proposed method as the five candidates to be protected. The method named STMR presented in [18], under the same area overhead constraint, applies TMR in gates g_{32}, g_{36}, g_{37}, g_{38} and g_{39}.

The reliability of the output bits for the original circuit and for the redundant configurations can be obtained by the SPR technique. Table 4.3 shows the reliability results for the respective configurations considering $q = 0.99$ for the gates not protected by TMR and $q = 1$ for the hardened gates.

Analyzing the results presented in Table 4.3, it can be seen the effectiveness of the proposed approach. Indeed, the proposed method to selectively apply TMR into a circuit shows a greater increase on the reliability of the most significant bits of the circuit (see Table 4.3). For instance, the reliabilities of S_0 and S_1 (LSBs) do not present any increase compared to the original circuit. Besides, the reliability of S_4 (MSB) presents the highest

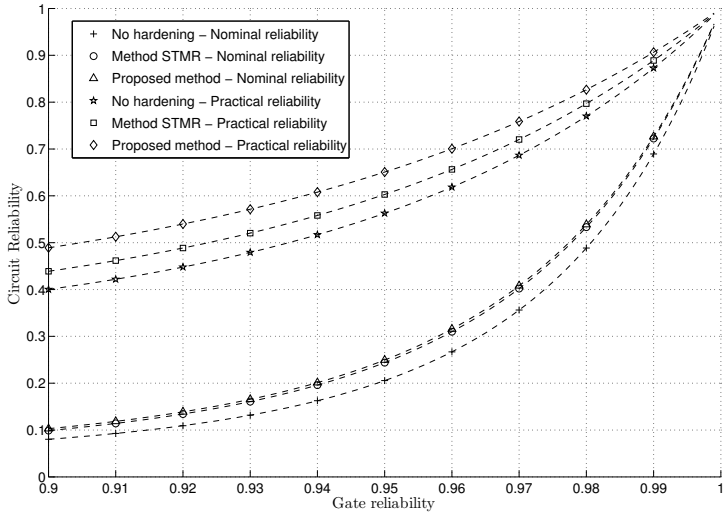


Figure 4.3: Simulation results for the 74283 circuit

improvement as expected, once it is considered the most critical bit for this application.

It can be also noted that, under the same area overhead, the nominal reliability increases by almost the same amount with both methods (see Figure 4.3). In fact, nominal reliability assigns equal reliability costs to the output bits. In spite of that, practical reliability results can handle this problem, and can indeed provide a sharper distinction between this two hardened architectures as shown in Figure 4.3.

4.3 Using a Cost Function to Detect Critical Gates

The reliability of a given circuit is the degree of confidence observed in the outputs of this circuit, given a scenario in which faults are expected to occur with a given probability. From the analysis point of view these faults could be either defects or transients induced by single event effects.

In this work, the SPR algorithm [15] is applied to obtain the reliability figures of a circuit. As stated in section 1.2.3, SPR uses both the reliability of the gates and signal reliability computation to determine the cumulative effect of multiple faults. The effort that is required to evaluate each gate of the circuit (in order to find the best hardening candidate) is only possible because the complexity of the SPR algorithm is linear with the number of gates in the circuit.

Let us consider that a given circuit comprises K gates $[g_i \cdots g_k]$. Each gate has an associated reliability, given by $[q_i \cdots q_k]$. The circuit as a whole has a reliability value R. Then, if we consider any reliability change (i.e., improvement) of a single gate g_i brings in its new reliability to q_i^*, the circuit's reliability becomes R_i^*. Two different gates, g_i and g_j, may have different contributions to the reliability of the circuit, therefore producing

different values R_i^* and R_j^*.

It is important to note that the SPR algorithm is not 100% accurate. The sources of inaccuracies come from incorrect evaluation of (multiple) reconvergent fanout branches. An accurate analysis is possible using the multi-pass algorithm described in [15], referred as SPR-MP. It is well known that both algorithms produce different values for the reliability of a circuit. Yet, in [27] it has been shown that SPR is capable of estimating the critical nodes (from a hardening perspective) with a small degree of error (in comparison with SPR-MP).

In our methodology we assume that a hardening technique is applied, and such technique is able to improve the reliability of a gate such that $q_i^* = 1$. This is a simplification, not a restriction, other values are also possible. Then, for all gates of the circuit we perform an evaluation run of the SPR algorithm. In each evaluation run we select a gate g_i, allow q_i^* to be 1, and obtain the new reliability value R_i^*.

After all evaluation runs are performed, we obtain an ordered list of R_i^* values. At this point one could select the gate with the highest R_i^* to be hardened. This is a common approach applied in many works [104, 106]. Yet, this approach could be considered naive since it does not take into account the hardening cost of each gate. Both mentioned works define a maximum area target that cannot be surpassed.

Thus, the goal of this research is to establish a trade-off between the costs of hardening a gate against the costs of hardening any other gate. In order to do so, a new parameter is introduced to express the hardening affinity of a gate, given by Cha_i. This parameter defines how easy/hard it is to harden a gate. The Cha_i value of each gate type is user-defined and it must be constrained in the interval [0,1]. The higher the value of Cha_i the better it is. This parameter is generic and can be used to express any type of hardening trade-off: area, delay, power or combinations of the previous. The decision of which circuit characteristic should be used to define Cha_i falls to the user.

In Table 4.4 we show some of the values that were used in our experiments. These values are extracted from an actual 90nm standard cell library provided by Synopsys [19]. In our experiments we considered only the area to calculate the hardening affinity. For each gate we have divided the area of the smallest inverter (INVX0) in the library by the given gate actual area, in order to normalize all the Cha_i values. Negated cells benefit from the CMOS natural inversion and have a higher Cha_i value.

It is then possible to apply the Cha_i values in a cost function which takes into account both the cost and the reliability gain. The reliability gain (or reliability difference) is given by Rg_i, and it is the difference from the circuit reliability before and after a single gate g_i was hardened:

Table 4.4: Hardware affinity (Cha_i) parameters for some cells

Cell	**Area** (μm^2)	Cha_i
INVX0	5.5296	1
NAND2X0	5.5296	1
NOR2X0	5.5296	1
AND2X1	7.3728	0.75
OR4X1	10.1376	0.55
XOR3X1	22.1184	0.25

$$Rg_i = R_i^* - R \tag{4.5}$$

These values are then used in a cost function that is expressed as follows:

$$C_i = Rg_i/Cha_i \tag{4.6}$$

Once the value of C_i has been obtained for all gates, these are sorted and the highest value is chosen. The gate that corresponds to the highest value of C_i is then assumed to be hardened and the new circuit reliability (R_i^*) is obtained.

In [27] this reliability value is compared against a user-given reliability target 'T'. If the reliability is lower than 'T', the algorithm starts again and all gates still not hardened are considered as candidates. Otherwise, if the target is met, the algorithm ends and outputs the ordered list of gates to be hardened. In the next section, an approach based on two heuristics is presented as a mean to automatically set a reliability improvement target.

4.3.1 Cost function profiling

The methodology described in Section 4.3 was applied to several ISCAS benchmark circuits [107]. The profile of the cost function was then obtained for circuits of different sizes and topologies. Figures 4.4 and 4.5 illustrate the cost function profile for the circuits *c432* (a channel interrupt controller) and *c499* (32-bit single-error-correcting circuit). These circuits were chosen particularly because they represent two contrastive profiles that are of interest.

The illustrations in both figures were obtained using the parameters $q_i = 0.999$ and $q_i^* = 1$. Other combination of values cause slight changes in the plots, i.e., the profile of the function remains the same. In other words, the profile of the function is highly related to the logic masking capabilities and the affinity of each gate. The closer a gate is to the **y** axis, the better candidate for hardening it is.

The illustration in Figure 4.4 represents a profile that contains a fast drop in the

function, observed in the very first gates. Circuits that have some degree of regularity (e.g., adders and multipliers) have a profile with some similarities with the one in Figure 4.5, where a 'step-like' pattern is observed. Each 'step' or plateau represents a set of gates that has a similar functionality in the circuit, therefore they can be hardened in any given order. Taking into account both profiles that were presented, we have defined two heuristics to decide when selective hardening starts to impose an impractical cost. Those heuristics are explained in details in the next subsections.

4.3.1.1 Sum of elements heuristic

This heuristic was defined to create a stop point when the sum of the C_i terms from the elements that were already hardened reaches a threshold. Let C_0 be the value of the cost function for the best hardening candidate. Then the target becomes to find a value j such that:

$$\sum_{i=2}^{j} C_i \quad \leq \quad K \times C_0 \tag{4.7}$$

where K is an empirically chosen constant. In other words, the threshold is defined as K times the value of the cost function for the first hardened gate. This heuristic can be interpreted as an integral that sums the area under a curve. For the sake of comparison, we have set the parameter $K = 10$ for all the case studies used in this work.

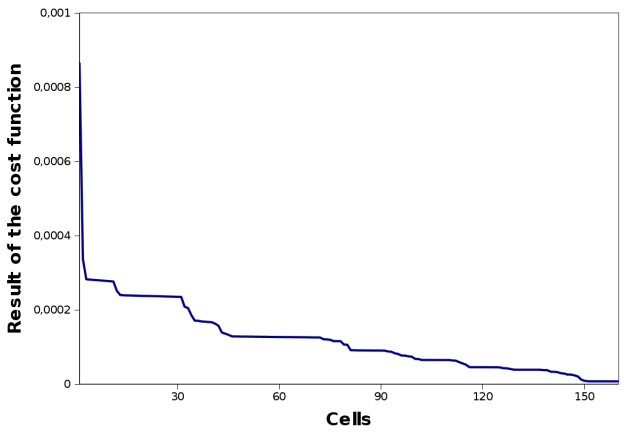


Figure 4.4: Cost function profile for the circuit *c432*

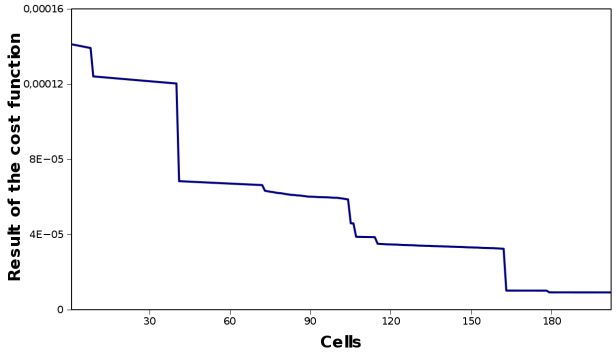


Figure 4.5: Cost function profile for the circuit *c499*

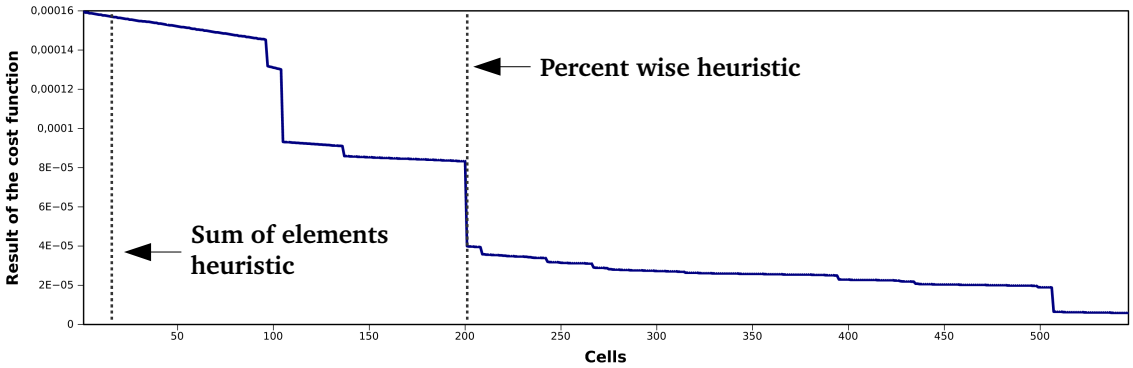


Figure 4.6: Both heuristics applied to the circuit *c1355*

4.3.1.2 Percent wise heuristic

This heuristic was defined to create a stop point at the first C_i value that is lower than $X\%$ of the first term (C_0). This heuristic can be interpreted as an horizontal threshold value. When the function crosses that threshold it is no longer feasible to perform selective hardening for the remaining gates.

For the sake of comparison, in the following we have empirically set the parameter $X = 50\%$. In other words, any gate that improves the circuit reliability with a C_i value that is less than half of C_0 should not be hardened, i.e., we only harden cells that are at least half as effective as the first candidate.

4.3.1.3 Comparing the heuristics

Both heuristics were applied to the circuit *c1355* (which is also a 32-bit single-error-correcting circuit). Figure 4.6 contains the plot of the cost function for all elements of the target circuit. The dashed vertical lines represent the points where the heuristics decided that selective hardening was no longer feasible.

Deciding which parameter value is more appropriate for each circuit is a complex task.

Table 4.5: Results for the sum of elements heuristic, $K = 10$

Circuit	**Number of gates**	**Original area (μm^2)**	**Hardened gates**	**Hardened area (μm^2)**	**Area increase**
c17	6	33.1776	6	99.5328	200%
74283	40	306.5096	20	547.9688	78.7%
c432	160	1134.4672	33	1541.4208	35.8%
c499	202	2155.1680	12	2414.1504	12.0%
c1355	546	3194.7328	11	3316.3840	3.8%
c1908	880	5273.7488	13	5417.5184	2.7%
c2670	1269	8018.0632	19	8233.7176	2.6%
c3540	1669	10855.1824	25	11177.7424	2.9%
c5315	2307	15293.5992	20	15518.4696	1.4%

For instance, for the circuit *c1355*, the first heuristic would select 11 gates for hardening, while the second heuristic would select 201 gates. Hardening 201 out of 546 gates (around 36%) might be a hard assignment, since most of the times the area budget will not allow for such hardening (the total circuit area would become 76% larger).

Nevertheless, selecting 11 out of 546 gates (around 2%) might be a better and more suitable choice. Along the same lines, applying the percent wise heuristic to the circuit *c432* would result in only 2 gates being selected for hardening, which could left some of the hardening budget unused.

In the next section, results for other circuits are presented. Also, the discussion regarding which heuristic (and associated parameter) is more appropriate for which scenario is extended.

4.3.2 Experimental results

The methodology described in Section 4.3 was applied to several ISCAS benchmark circuits. Each gate from each circuit was set using $q_i = 0.9999$. The results are presented in tables 4.5 and 4.6. The former table contains the results for the first heuristic defined in subsection 4.3.1.1 (with $K = 10$) while the latter contains the results for the second heuristic defined in subsection 4.3.1.2 (with $X = 50\%$).

In tables 4.5 and 4.6, the meaning of each column is as follows: the column denoted "Original area" contains the sum of the area of each gate in each circuit (therefore placement utilization rate and routing overhead are not considered). The column denoted "Hardened gates" contains the number of gates that are selected for hardening. Then, the column denoted "Hardened area" contains the circuit area of the hardened version of the circuit, while the column denoted "Area increase" contains that same value but percent wise.

A fairly simple assumption was made: when hardening a gate its area become three times larger than before. This metric is inspired by classical Triple Modular Redundancy

Table 4.6: Results for the percent wise heuristic, $X = 50\%$

Circuit	Number of gates	Original area (μm^2)	Hardened gates	Hardened area (μm^2)	Area increase
c17	6	33.1776	5	88.4736	166.6%
74283	40	306.5096	9	406.0424	32.5%
c432	160	1134.4672	2	1187.5264	4.6%
c499	202	2155.1680	41	2854.6752	32.4%
c1355	546	3194.7328	201	5647.1232	76.7%
c1908	880	5273.7488	119	6611.912	25.3%
c2670	1269	8018.0632	10	8128.6552	1.4%
c3540	1669	10855.1824	8	10963.9312	1.2%
c5315	2307	15293.5992	15	15459.4872	1.1%

(TMR), although other techniques with different metrics might be applied (e.g., hardening by design). The additional area that would be required for a majority voter, given TMR is considered, is neglected. Therefore the area figures given in the tables are a minimum value estimate for TMR.

An analysis of the area increase values in Table 4.5 reveals that the sum of elements heuristic is not prone for small circuits, causing a large overhead for the circuits *74283* and *c432*. For the smallest of the circuits (*c17*) the heuristic decides that all gates should be hardened, which is unacceptable when the goal is selective hardening. Nevertheless, this can be avoided by using a smaller value for the parameter K (e.g., $K = 1$ elects 2 cells while $K = 2$ selects 4 cells for hardening). This is not the case for the area increase values in Table 4.6. There is no value for the parameter X that will be a good fit for all circuits or even for a group of circuits. Therefore, it is quite harder to apply the percent wise heuristic.

4.3.3 Comparison with related works

A straightforward comparison with other methodologies is not simple since the goals are usually different. If comparing a methodology is hard, it is even harder to compare the heuristics proposed in this work. A simple solution adopted by related works is to define a limit or target for hardening. In [108] a simple limit L is defined as the maximum number of gates to be hardened. In references [106] and [104] a hardening limit in terms of area increase is applied. In [27] the authors have defined the hardening target as a relative improvement in the reliability of the circuit. None of the mentioned works perform an evaluation of how hard it was to reach a hardening limit/target. This is the reason why we have studied the profile of the cost function.

The work of [109] has a similar target as the one described in this work. The hardening is achieved by increasing the gate size of some critical nodes in the circuit but no hardening

against defects is mentioned. Although this is a valid solution, it can be quite complicated to apply it in a commercial design flow since the choices of logic gates are limited. Thus the technique presented here is a more general solution since it is library and technology independent. The overheads mentioned in [109] are not directly comparable.

Nevertheless, in qualitative terms it is easily observed that certain gates have a larger impact in the reliability of the circuit than others. This observation is highlighted in [104, 106, 108, 109]. In our experiments this was also observed. There are some particular cases, like the one illustrated in Figure 4.4, where choosing the correct gate to harden has a large impact in the overall circuit reliability.

4.4 Conclusion

In a context where defects and soft errors are a growing concern, two methods to selectively apply TMR to digital circuits were proposed. The first one detects critical gates by taking into account not only the probability of error occurrence, but also the impact of such error to the system. Simulation results have shown the effectiveness of the proposed approach. Although the reliability of the circuit obtained with the proposed method is only slightly different from the one stated in [18], the reliability enhancement is most present in the most critical bits of the system. In fact, the ability to drive the reliability improvement effort to critical bits of a system is of great use for applications in which some output bits are more critical than others. In addition, a new metric to analyze the reliability of a circuit with a multiple-bit output is also presented. This metric, named *practical reliability*, allows a designer to assign different weights to different output bits in order to reflect different error costs to a system. It has been shown that the proposed metric provides additional information to the designer so that he can better analyze the reliability of a system. The second one is based on some heuristics that provide a better understanding of the costs related to selective hardening applied for combinational logic in digital circuits. Furthermore, we have also dealed with multiple faults to determine the actual reliability of the circuits in our analysis. The results present the use of the methodology in conjunction with a standard cell library from an actual vendor, where the trade-off between area and reliability gain is highlighted. Thus, the methodology can be integrated in commercial design flows in a very straightforward manner.

Chapter 5

Optimizing Voter Placement for TMR Systems

5.1 Introduction

This chapter introduces a method to automatically partition a TMR design based on a given reliability requirement. First, some basics on TMR will be introduced in section 5.2. Next, a mathematical analysis of TMR partition will be explained in section 5.3, and some problems regarding the insertion of majority voters will be presented in section 5.4. The proposed method is explained in section 5.5. Concluding remarks are given in section 5.6

5.2 TMR approach

Triple modular redundancy is a well-known fault-tolerance technique based on a very simple concept. Three identical logic modules (or black boxes) performing the same task feed the inputs of a majority voter, which is responsible to evaluate the final output (see Figure 5.1). Since the outputs of the modules are binary and the number of inputs of the majority voter is odd, an unambiguous majority opinion can be provided as the output [60].

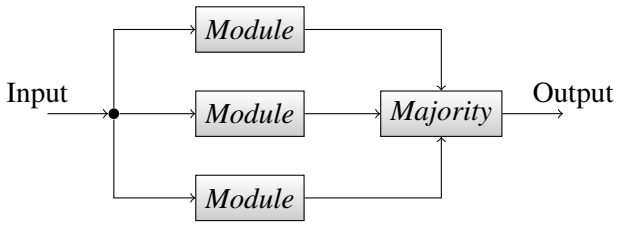


Figure 5.1: TMR block scheme

If it is considered that the majority voter does not fail, the reliability of a redundant

circuit protected by TMR (R_{cir}) can be defined as a function of the reliability of one module (R_m) as shown in (5.1). Notice that it is also assumed that the failures of the redundant modules are independent.

$$\begin{aligned} R_{cir} &= R_m^3 + 3R_m^2(1 - R_m) \\ R_{cir} &= 3R_m^2 - 2R_m^3 \end{aligned} \tag{5.1}$$

Further information regarding the use of TMR to improve the reliability of a circuit can be extracted from (5.1). First of all, it can be seen that if $R_m \leq 0.5$, the use of TMR will not increase the circuit's reliability R_{cir}. Next, if $R_m \approx 1$, the increase in reliability achieved is so small that the area penalty required by this technique may not worth it. Indeed, as shown in Figure 5.2, the gain in reliability (R_{cir}/R_m) acquired by the use of TMR depends on the reliability of the block R_m.

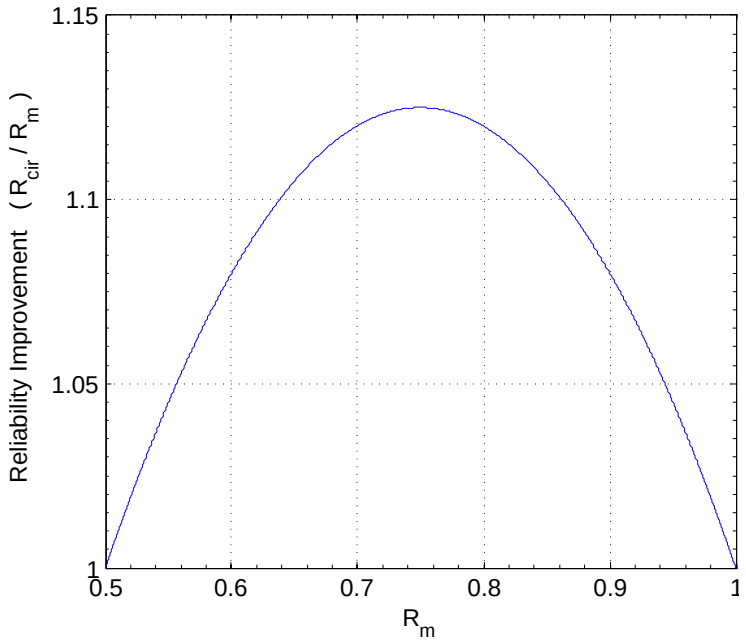
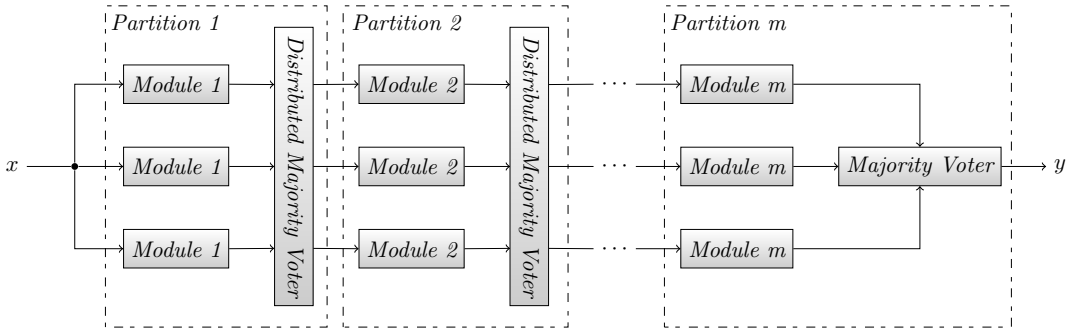


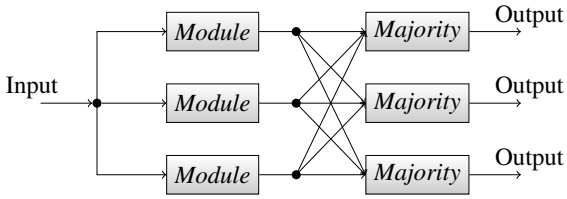
Figure 5.2: Reliability gain using TMR

Despite its simplicity, triple modular redundancy is a fault-tolerance technique that can yield great results. Circuits protected by TMR can provide correct outputs even if one redundant module fails. In general, the larger the size of a logic module, the more likely is the occurrence of multiple errors. However, if two redundant modules produce erroneous results at the same time in a TMR circuit, the final output is incorrect. Because of that, it would be a more efficient approach to partition a large logic module into 'n' logic partitions

as shown in Figure 5.3 [110]. In this case, the circuit will provide incorrect outputs only if two logic modules located at the same logic partition produce erroneous results. Notice that each logic module in Figure 5.3(a) is protected by three redundant majority voters, as shown in Figure 5.3(b), because of two main reasons: first, this avoids a single point of failure if the majority voter fails; and second, three different paths are available to be connected to the next module.



(a) TMR scheme with logic partitions



(b) Distributed majority voter

Figure 5.3: Partitioning a TMR design

Although the reliability tends to increase with the decrease in the module size, the amount of resources required for the addition of the majority voters may be too costly. Nevertheless, it has been proved that placing voters only at the final output may be not sufficient to avoid errors [110]. Because of that, there is a major need to detect the best amount of voters and the best locations to insert them into the circuit in order to meet a given reliability requirement. In order to tackle this problem, let us start by performing a mathematical analysis about partitioning a TMR design in the next section.

5.3 Partitioning a TMR design

Let be C_1 a circuit whose reliability is represented by R_{C_1}. Considering that this circuit is composed of n modules serially interconnected as shown in Figure 5.4, the reliability of the circuit C_1 can be expressed as follows:

$$R_{C_1} = \prod_{k=1}^{n} (R_{m_k}) \tag{5.2}$$

Assuming that each module is now implemented as a TMR version of itself, that is the TMR version of the circuit C_1 comprises n partitions, and considering that the voters do not fail, the reliability of the TMR circuit can be evaluated as shown in (5.3).

$$R_{C_{1_{TMR}}} = \prod_{k=1}^{n} (3R_{m_k}^2 - 2R_{m_k}^3) \tag{5.3}$$

Considering the use of n partitions, let us evaluate what is the value of the reliability of each block R_{m_k} such that (5.3) is maximized. In order to do that, we can use (5.2) and (5.3) and the method of Lagrange Multipliers as shown below.

$$g(R_{m_n}, R_{m_{n-1}}, \cdots, \lambda) = \prod_{k=1}^{n} (3R_{m_k}^2 - 2R_{m_k}^3) - \lambda(\prod_{k=1}^{n} (R_{m_k}) - R_{C_1}) \tag{5.4}$$

Then, performing a series of partial derivatives, the following system of equations can be found.

$$\begin{cases} \frac{\partial}{\partial R_{m_1}} g = (6R_{m_1} - 6R_{m_1}^2) \prod_{k=2}^{n} (3R_{m_k}^2 - 2R_{m_k}^3) - \lambda \prod_{k=2}^{n} (R_{m_k}) = 0 \\ \frac{\partial}{\partial R_{m_2}} g = (6R_{m_2} - 6R_{m_2}^2)(3R_{m_1}^2 - 2R_{m_1}^3) \prod_{k=3}^{n} (3R_{m_k}^2 - 2R_{m_k}^3) - \lambda R_{m_1} \prod_{k=3}^{n} (R_{m_k}) = 0 \\ \frac{\partial}{\partial R_{m_3}} g = (6R_{m_3} - 6R_{m_3}^2) \prod_{k=1}^{2} (3R_{m_k}^2 - 2R_{m_k}^3) \prod_{k=4}^{n} (3R_{m_k}^2 - 2R_{m_k}^3) - \lambda R_{m_1} R_{m_2} \prod_{k=4}^{n} (R_{m_k}) = 0 \\ \vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ \frac{\partial}{\partial R_{m_n}} g = (6R_{m_n} - 6R_{m_n}^2) \prod_{k=1}^{n-1} (3R_{m_k}^2 - 2R_{m_k}^3) - \lambda \prod_{k=1}^{n-1} (R_{m_k}) = 0 \end{cases}$$

By inspection, it can be seen that $R_{m_1} = R_{m_2} = \cdots = R_{m_n} = R_{C_1}^{1/n}$ is a solution of such system, and it can be shown that this point is a maximum of (5.2) for $0 \leq R_{C_1} \leq 1$. However, the function represented by (5.3) is restricted to the region represented by (5.2) for $0 \leq R_{C_1} \leq 1$, and therefore it presents discontinuities. Because of that, we have to

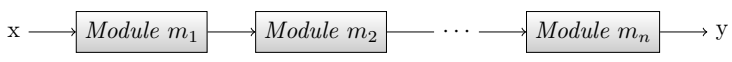


Figure 5.4: C_1 comprises n modules serially interconnected

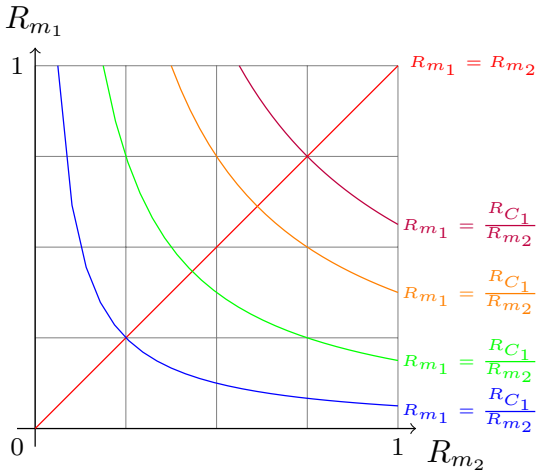


Figure 5.5: Plot of $R_{m_1} = \frac{R_{C_1}}{R_{m_2}}$ for different values of R_{C_1}

proof that there is no point in the boundary of such region that is a maximum of the function. In order to do that, let us analyze the simple case of $n = 2$, for which the system of equations is shown below:

$$g(R_{m_1}, R_{m_2}, \lambda) = (3R_{m_1}^2 - 2R_{m_1}^3)(3R_{m_2}^2 - 2R_{m_2}^3) - \lambda((R_{m_1}R_{m_2}) - R_{C_1}) \tag{5.5}$$

$$R_{C_1} = R_{m_1}R_{m_2} \tag{5.6}$$

The behavior of the constraint function (5.6) for different values of R_{C_1} is shown in Figure 5.5. It can be seen that in order to analyze the boundary of such region, we must set $R_{m_1} = 1$, which leads to $R_{m_2} = R_{C_1}$, or $R_{m_2} = 1$, which leads to $R_{m_1} = R_{C_1}$. In this case, equation (5.5) becomes (5.7), which is the same as the equation for a TMR circuit with a unique partition. Considering perfect voters, it is known that the reliability of a circuit increases with the number of voters. Therefore, (5.7) provides a value of reliability that is smaller than (5.5), that is the points in the boundary do not represent a maximum of the function. This result can be generalized for n dimensions since each time we set a R_m value to 1, the set of equations will correspond to the analysis of a system with $n-1$ partitions. Thus, in order to achieve a maximum TMR reliability, we have to partition the design into blocks of as nearly reliability as possible, that is $R_{m_1} = R_{m_2} = \cdots = R_{m_n} = R_{C_1}^{1/n}$. The same result can be seen in [60] for the case of imperfect voters. Based on that, (5.8) represents the reliability of a TMR circuit with n equal partitions, which maximizes the TMR performance. Notice that (5.8) can be used to evaluate the minimum number of partitions n required to meet a given reliability $R_{C_{1_{TMR}}}$.

$$g(R_{m_1}, R_{m_2}, \lambda) = (3R_{C_1}^2 - 2R_{C_1}^3) \tag{5.7}$$

$$R_{C_{1_{TMR}}} = (3R_{C_1}^{2/n} - 2R_{C_1}^{3/n})^n \tag{5.8}$$

5.4 Problem of automatically inserting voters

As stated above, TMR is a technique widely used in the construction of fault tolerant circuits and systems. However, manually apply TMR to a circuit and insert the required voters is an error-prone task, so the automation of such procedure is an important requirement. Tools such as Xilinx XTMR [111] and BYU-LANL Partial TMR [79] can automatically apply TMR in case of FPGAs. For instance, XTMR tool provides several features such as triplicate the inputs, the clock, the majority voters, and yet insert synchronization voters to feedback loops. The last one intends to synchronize the sequential logic state of the TMR modules when a scrubbing process is performed to correct the effects of SEEs. In spite of that, not much is provided in order to partition a design and insert voters such that a minimum reliability requirement is met.

As a matter of fact, in order to design high-reliable circuits, a traditional TMR implementation may not be enough, and the use of partitioning voters, i.e. voters that intends to partition a design in order to increase its reliability, may be required. In [112] the authors have concluded that the number and the placement of voters in a TMR design directly affect its fault tolerance ability, and therefore cleverly insert voters may be a good approach to build high-reliable systems. However, the decision of the quantity as well as the placement of such voters is very complicated since it is governed by several constraints, such as timing, area and reliability. Further, the insertion of voters in some nodes of a circuit may not be allowed or desired. For instance, some FPGAs contain dedicated route connections to implement some specific functions such as a ripple-carry adder that does not allow the insertion of a voter.

Methods to automatically insert voters generally rely on simple rules such as the insertion of voters after every flip-flop. Although simple, this technique ensures that only one voter will be inserted per timing path, which reduces the timing penalty caused by the voter insertion process. On the flip side of the coin, this technique may insert more voters than the necessary to meet a given reliability requirement. This insertion procedure can be performed in different levels of a design. For instance, in case of FPGAs the voter insertion is generally performed using FPGA primitives such as LUTs and flip-flops. In next section

we introduce a method that can automatically insert voters in a design, whether it be an ASIC of FPGA, based on a given reliability requirement.

5.5 Proposed method

In order to develop our methodology, let us first analyze the behavior of the reliability of a circuit as a function of its number of levels. In such context, a level can be defined as a circuit node with the capability to become the edge of a partition. In practical circuits these prospective points can be for example registered outputs. A very useful representation for the reliability of a circuit regarding its number of levels is the *signal reliability* concept explained in Section 1.2.3. This is because such concept can provide the probability of a correct result in any circuit node. The expected behavior of the signal reliability versus the number of levels can be seen in Figure 5.6. Notice that it usually decreases with the number of levels.

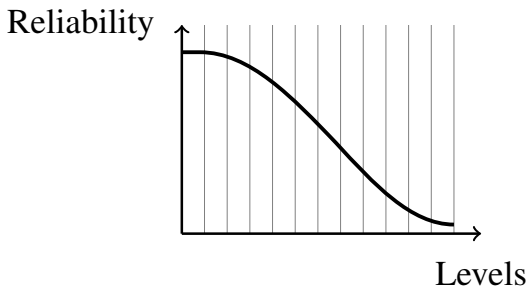


Figure 5.6: Reliability of a circuit versus its number of levels

When a majority voter is inserted in an arbitrary level of a TMR circuit, an increase in the signal reliability of such level is expected as shown in Figure 5.7. This is due to the logical masking ability provided by the voter decision, which, as it is well known, can correct the occurrence of any single error presented in its inputs.

From these two figures shown above, a very simple idea can be elaborated as a means to provide the edges of the modules of a TMR implementation, defining then their sizes. Supposing that a given circuit has a minimum reliability requirement, the edges can be identified by the levels in which the signal reliability is as close as possible to the minimum reliability requirement as shown in Figure 5.8.

Although the method lies in a very simple idea, the provided results are very close to the optimal ones. As shown in Section 5.3, it has been proved that to achieve a maximum TMR reliability, a circuit must be subdivided into modules of as nearly equal reliability as possible [60]. Using the proposed approach, only the first and the last modules may not have the same size. As a matter of fact, two different characteristics contributes to obtain

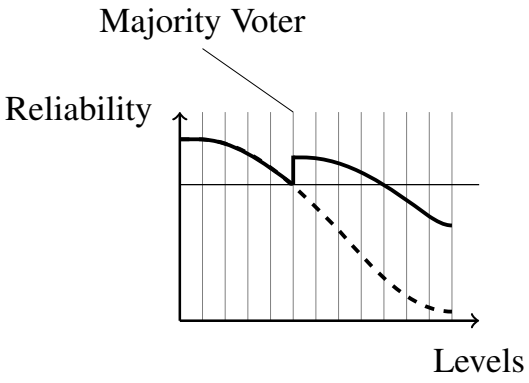


Figure 5.7: Insertion of a majority voter

good results with the proposed method. First, the reliability gain obtained with a TMR system depends on the reliability of the module R_m as shown in (5.1). Next, the decrease in signal reliability experienced when a signal passes through a given block is proportional to the reliability of such block. Therefore, limiting the signal reliability decrease to the same value leads to the creation of blocks with reliability as close as possible to each other, thus approaching the optimal solution.

For the sake of illustration, let us consider a simple circuit comprising 10000 inverter gates with equal reliabilities ($R_m = 99.99\%$) and interconnected in a cascade structure as shown in Figure 5.9. The reliability of this circuit, represented by R_c, can be evaluated using the SPR tool, which leads to a result of $R_c = 56.7654\%$. Assuming that the minimum reliability requirement is $R_{min} = 99.9\%$, the proposed method evaluates that 1000 voters must be inserted in order to meet such requirement. All the created partitions have the same number of components ($N = 10$), with exception of the first ($N = 11$) and the last ($N = 9$) ones. Follows from (5.8) that the minimum number of partitions to achieve R_{min} is 961 in this case. However, in order to split the circuit into 961 partitions with the same

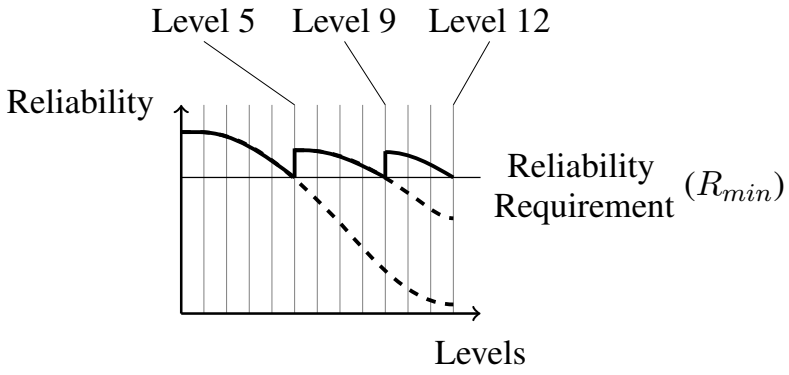


Figure 5.8: Distributing the voting process of a TMR circuit

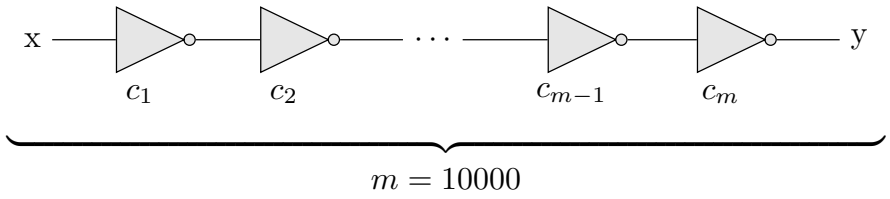


Figure 5.9: Cascade of inverters

size, each partition should contain 10.4058 inverter gates, which is not possible. Therefore, follows that the best amount of modules in each partition should be 10, which leads to the result of 1000 voters as well. Figure 5.10 illustrates the results obtained with (5.8) and with the proposed method for different values of R_{min}. Notice that the optimal values evaluated by (5.8) are not yet rounded such that the division of the circuit into partitions with the same size can be realized.

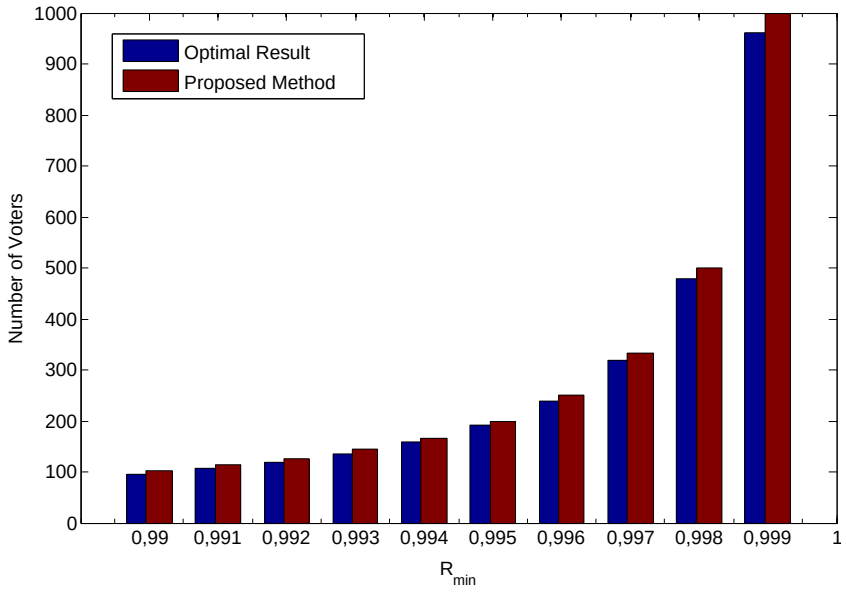


Figure 5.10: Number of voters inserted by the proposed technique

Although the results provided by the proposed method regarding the cascade circuit shown in Figure 5.9 are very close to the optimal results, this is a special case and its performance regarding more complex circuits must also be analyzed. Thus, let us now consider a 74283 4-bit fast adder for which the gate-level architecture is shown in Figure 5.11. As can be seen, this circuit comprises 40 logic gates, 9 inputs and 5 outputs. First of all, let us consider that the output of any logic gate is a prospective point to insert a majority voter, and that the reliability of any of such gates is 99%. Remember that the primary outputs of the circuit have already majority voters. Table 5.1 shows the obtained results for three different reliability requirements (R_{min}). It can be noted that the first two points

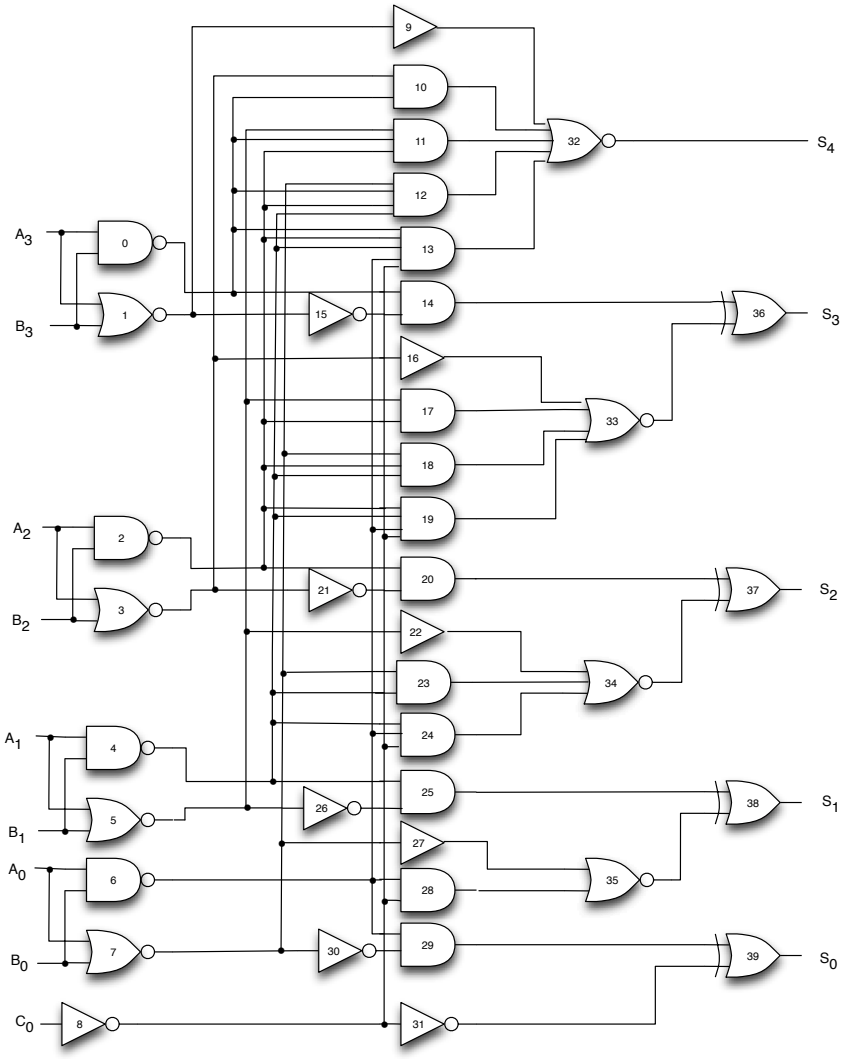


Figure 5.11: Circuit 74283 - Gate level

in which the method inserts voters are the outputs of the gates 33 and 34. As a matter of fact, these two gates are points in which several signals converge (fanins) leading to a more accentuated decrease in the signal reliability. Also, as the reliability of the logic gates are assumed to be the same, paths in which more logic gates are presented tends to contain more voters. For instance, voters are placed in points in which the signal passes through at least 3 logic gates for $R_{min} = 97\%$.

The results obtained with the proposed method also depend on the reliability of each logic gate. In the last example, we have considered that any logic gate has reliability $q = 99\%$. However, let us now remove such consideration and assume that the reliability of a gate depends on its area. The area of the logic gates are obtained from an actual $90\eta m$ standard cell library provided by Synopsys [19]. We considered that the reliability of an

Table 5.1: Placement of the voters for the circuit 74283

R_{min}	Number of Voters	Voter Placement (Output of Gates)
0.96	2	33, 34
0.97	7	14, 20, 25, 29, 33, 34, 35
0.98	11	13, 14, 19, 20, 24, 25, 28, 29, 33, 34, 35

Table 5.2: Reliability of gates based on their area

Gate	Area (μm^2)	Reliability
Inverter	5.5296	0.99
Buffer	5.5296	0.99
2-input NAND	5.5296	0.99
2-input NOR	5.5296	0.99
2-input AND	7.3728	0.9867
3-input AND	8.2944	0.985
3-input NOR	8.2944	0.985
4-input NOR	9.2160	0.9833
4-input AND	10.1376	0.9817
2-input XOR	13.8240	0.975
5-input NOR [a]	14.7456	0.9733
5-input AND [b]	16.5888	0.97

a. Built with a 4-input NOR and a 2-input NOR gates
b. Built with two 3-input AND gates

inverter gate is $q = 99\%$, and we derive the other reliabilities based on the relation between the area of a gate and the area of the inverter. These values are shown in Table 5.2. Based on such reliability values, we have performed another analysis of the 74283 circuit using the proposed approach for three different values of R_{min}. The obtained results are shown in Table 5.3. Observe that the placement as well as the quantity of the voters are different from the results presented in Table 5.1. Now, for $R_{min} = 96\%$, 4 majority voters are inserted into the outputs of gates 13, 33, 34 and 35. For instance, gate 13 is the gate with highest probability of failure since it has the biggest area, and therefore a majority voter is now inserted in its output.

Obviously, the 74283 circuit is very small when compared to circuits produced nowadays. However, it is just a case study in order to demonstrate the performance of the proposed method when dealing with non-cascade structures. In fact, this method is based on the SPR tool, which has some interesting characteristics that allows a very good scaling ability. For instance, the method can be applied to different levels of abstraction of a circuit such as gate level, block level, among others. Also, the complexity of the SPR algorithm is linear regarding the number of gates/blocks of a design, and therefore can be

Table 5.3: Placement of the voters for the 74283 circuit

R_{min}	Number of Voters	Voter Placement (Output of Gates)
0.95	2	33, 34
0.96	4	13, 33, 34, 35
0.97	9	13, 14, 19, 20, 25, 29, 33, 34, 35

applied to relative large circuits. Moreover, the consideration of the signal reliability of a circuit to decide the points in which we should insert voters allows to deal with circuits with complex structures and yet provide good results.

5.6 Conclusion

The current chapter presented an automatic method to partition a TMR design. This is of great interest for systems in which safety is a major concern so that reliability must be as high as possible. Although the proposed method is based on a simple idea, the results are very close to optimal. Also, the algorithm is based on the SPR method, which has a linear complexity related to the number of nodes of a circuit. Therefore, it can be applied to relative large circuits. In addition, it is important to note that the whole decision process can be easily automated. In fact, a tool written in C language was already developed in order to validate the proposed idea. Such implementation can deal with gate level descriptions of circuit as well as with primitives of FPGAs devices.

Chapter 6

Concluding Remarks

As electronic circuits shrinks down, methods to analyze and fabricate reliable circuits are more and more required. The present research provides some new models and tools that can be used to design reliable circuits such that these circuits can more easily meet the requirements of the technical standard IEC 62566.

The main objective of this work was to propose methods to analyze and to improve the reliability of circuits in order to facilitate their qualification according to technical standards such those useful for EDF. The necessary steps to achieve this goal have been presented in the chapters of the report. First, several studies already reported in the literature were reviewed. This reading produced Chapters 1 and Annexes A and B of the current manuscript.

Concerning reliability analysis, this work proposed the concept of *effective reliability*, which provides a model to evaluate the reliability of a circuit taking into account the ability of a given application to tolerate faults, here called application error masking. The results have shown the importance of considering such concept when evaluating the reliability of a circuit. As a matter of fact, *effective reliability* can provide a more precise result about the reliability of a circuit so that unnecessary area overhead can be avoided.

An interesting technique that can provide a tradeoff between the amount of redundancy adding and the reliability improvement is selective hardening, which is based on protecting only the most critical gates/blocks of a circuit. In this work, two different methodologies were proposed to deal with that. The first one takes into account that some output bits are more critical than others for a given application, and that these bits should be more protected. The other uses a hardware function and some heuristics to automatically detect the best candidates in a design to be protected.

Concerning critical applications in which the reliability requirement is usually very high, a method to automatically partition a TMR design based on a desired reliability level was introduced. The method has shown to provide results very close to optimal, and

that it can be easily integrated in design tools to be used in VLSI as well as in FPGA circuits.

Further, a tool based on fault injection, named FIFA, was also developed in order to analyze and validate fault tolerant designs explored in this work. Although most of the tests performed with the tool were based on exhaustive analysis, FIFA can be easily used to inject randomly faults as well as to use just a set of inputs, controlled by the designer, that are considered more pertinent for the application.

The perspectives of the current work can be explained as follows. Concerning reliability analysis, the concept of *effective reliability* can be expanded by the development of new quality metrics to measure the impact of errors in the output of a circuit. By doing that, designers can better control the set of errors that can be tolerated by a given application. The FIFA tool can be upgraded by analyzing and then limiting the amount of nodes to inject multiple faults. For example, multiple faults could be limited to a region in the design. Concerning reliability improvement techniques, the idea of drive the reliability effort to the most critical bits of an application can be extended by implementing other weight functions that reflect the impact of an error in non-binary output words. Also, the method to partition a TMR design can be improved if the signal dependencies resulted from reconvergent fanouts are taken into consideration.

The research performed during this thesis originated several publications as can be seen bellow. Also, a patent of the method to partition a TMR design is under analysis.

- **Effective metrics for reliability analysis** [101] - Oral presentation at the 53rd Midwest Symposium on Circuits and Systems (MWSCAS) 2010;
- **Using error tolerance of target application for efficient reliability improvement of digital circuits** [102] - Oral presentation at the 21st European Symposium on the Reliability of Electron Devices, Failure Physics and Analysis (ESREF) 2010 and published as a special issue of the *Microelectronics Reliability* journal;
- **An approach for efficient reliability improvement of digital circuits** - Poster presentation at Gdr SoC-SiP, Lyon, France, 2011;
- **FIFA: A fault-injection-fault-analysis-based tool for reliability assessment at RTL level** [105] - Oral presentation at the 22nd European Symposium on the Reliability of Electron Devices, Failure Physics and Analysis (ESREF) 2011 and published as a special issue of the *Microelectronics Reliability* journal;
- **Exploring the Feasibility of Selective Hardening for Combinational Logic** [103] - Accepted for poster presentation at the 23rd European Symposium on the Reliability of Electron Devices, Failure Physics and Analysis (ESREF) 2012 and published as a special issue of the *Microelectronics Reliability* journal;
- **Net Hardening: an Approach for Selective Hardening Concerning Multi-**

ple Faults - Submitted to IEEE Transactions on Nuclear Science (TNS).

Appendix A

Other Methods for Reliability Improvement of ICs

A.1 Fault detection and correction

A.1.1 Basic principles

This redundancy technique is based on the detection of errors in a circuit in order to perform a corrective action. The fault detection is usually carried out by internal mechanisms, as well as the corrective action. A circuit that has the ability to detect an internal fault is called a self-checking circuit.

Self-checking circuits have received a lot of attention over the years [113,114]. The basic idea is to encode the information in such a way that errors can be detected. Figure A.1 illustrates a self-checking circuit using fault detection redundancy. In this example, the error detection mechanism is based on a simple comparison between two copies of the same module, and is named duplex scheme. It is important to note that this scheme does not provide intrinsically error correction capability. Thus, when the error flag reports the occurrence of an error, we must carry out additional procedures in order to resolve the problem.

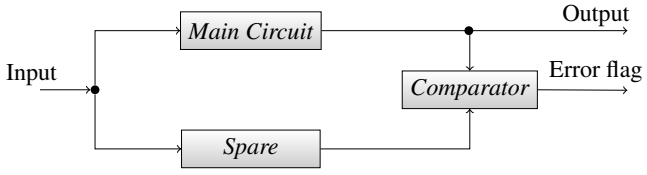


Figure A.1: Example of a duplex comparison scheme

In order to use fault detection redundancy in a circuit, we must be careful when choosing

the fault detection technique and the recovery procedure to be used. If a single fault is presented in such mechanisms, the correct operation of the circuit is compromised. Section A.1.2 introduces a brief state of the art of fault detection techniques existent in the literature.

A.1.2 Fault detection techniques

In order to perform a fault detection procedure in a circuit, the addition of redundancy is a necessary design penalty. This redundant information is generally used to implement a code technique in which each output data is represented as a codeword. There are two types of codes that can be used in such procedure: Error Detecting Code (EDC) and Error Correcting Code (ECC) [115]. EDCs have the ability to identify error(s) in a data word, but they cannot restore its(their) true value. Therefore, a post processing step is often required when using such codes. ECCs can do both detect an error in a codeword and restore the corresponding correct value. Thus, an ECC can be configured to do both detect and mask the effect of errors in a circuit.

The characteristic of how many erroneous bits can be identified and/or corrected by a given code is based on the *Hamming Distance* (HD) principle. The HD of a given code can be defined as the number of bits in which two codewords differ. Derived from this principle, the term *Code Distance* (CD) is often used to define the minimum HD between any two symbols presented in a given code. In general, if we want to detect d errors in a circuit, we must have a code with $CD \geq d+1$. Moreover, in order to detect and correct t errors, then $CD \geq 2t+1$, and to detect d and correct t, or fewer errors, the $CD \geq t+d+1$, where $t \leq d$ [115]. For example, suppose a system in which the set of codewords allowed by this circuit is given by $C = \{000, 111\}$. In this case, the Code Distance is given by $CD = 3$; what means that this code can detect 2, or fewer errors, or correct 1. Indeed, the code will interpret the received codeword as the most closer word presented in C. As an example, if the system receives the word 001, it will be corrected to 000. Notice that this example illustrates the behavior of a TMR system.

A.1.2.1 Error detecting codes

Error detecting codes are often used in techniques based on fault detection redundancy. These codes provide error detecting capabilities but lack any error correcting capability. Then, further procedures are required in order to execute a correction procedure.

A.1.2.1.1 Parity code

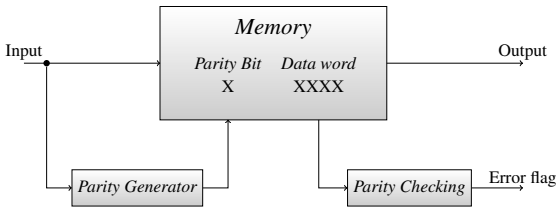


Figure A.2: Computer memory using parity checking

The simplest EDC is the *Parity Checking* [114]. The main idea is to append an extra bit to a binary word in such a way that the number of 1's is odd or even. When examining the codeword, a parity checker counts the number of 1's to verify the presence of errors. Codes in which codewords are constructed appending check bits are named *systematic codes*. They have the advantage that no decoding is required to get the final output.

The parity code has a code distance $CD = 2$, i.e., this code allows the detection of one faulty bit without correction. Figure A.2 illustrates a computer memory that uses the parity code to detect errors presented in its data. Notice that we just added two blocks to the basic memory structure: the parity generator and the parity checking. Both circuits are simple, resulting in a low area overhead.

Because of the aforementioned features, scientists have been researching the use of parity codes for many decades. The work in [116] proposed a procedure for synthesizing multilevel circuits with concurrent error detection in which all errors caused by single stuck-at faults were detected using a parity-check code. The proposed procedure automatically searches for the parity-check code that will require the least amount of area to implement. The work proposed in [114] compared the performance of twelve different combinational circuits implemented into an Spartan-2 FPGA using four different error detecting techniques: duplication with comparison, parity checking, Bose-Lin code, and Berger code. The results have shown that for an FPGA, the best solution regarding area overhead and performance decrease is the parity checking technique.

A.1.2.1.2 Berger code

Berger code is a systematic all-unidirectional error detecting code (AUED) proposed by J. M. Berger in 1961 [117]. Unidirectional errors states for either a $(0 \rightarrow 1)$ or $(1 \rightarrow 0)$ transition, but not both. This type of errors is particularly important to VLSI designs since Pradhan [118] has shown that a large number of errors presented in such designs are unidirectional. The main idea of Berger code is to append check bits to the data word. These check bits are obtained according two encoding schemes: B_0 or B_1. In the first

scheme, the check bits are obtained counting the number of 0's and representing the value into binary. In the other one, the encoding scheme uses the number of 1's represented using 1's complement. Table A.1 illustrates a 3-bit Berger code with B_0 encoding scheme.

Table A.1: 3-bit Berger code – B_0 scheme

Data word	Check bits	Codeword
000	11	00011
001	10	00110
010	10	01010
011	01	01101
100	10	10010
101	01	10101
110	01	11001
111	00	11100

Design of totally self-checking (TSC) circuits based on Berger codes was first introduced in [119] in 1974. The checker used 2^k gates to translate the Berger codewords to 1-out-of-2^k codewords. Therefore, it was impractical and inefficient. Later, a more efficient design procedure for Berger code was introduced in [120]. The method nevertheless failed to deliver a 2-output TSC checker for Berger codes with $I = 2^{r-1}$ information bits [121]. Intending to tackle this problem, the work in [122] proposed the generalized Berger check partitioning method that allowed TSC Berger code checkers be constructed from TSC m-out-of-n checkers . However, the proposed approach was not entirely self-testing as shown by Piestrak afterwards [121]. Meanwhile, Piestrak also proposed the first correct design approach for such circuits [123].

As stated above, Berger code can detect all unidirectional errors presented in a data word if there is no error in the check bits. However, the Berger code is not efficient since it requires a quite large number of check bits. A more efficient solution is the Bose Code presented below.

A.1.2.1.3 Bose codes

The first Bose code that will be presented in this report is the Bose AUED code. These code is also an all-unidirectional error detecting code, as the Berger code, but it is more efficient since it requires fewer check bits. Indeed, it requires exactly d check bits to detect any unidirectional error presented in a data word comprising 2^d bits. Table A.2 shows the Bose code for a 4-bit data word. Considering the number of zeros in a data word being represented by N_o, the rules to be followed in order to encode a data word using the Bose code are:

- $(N_o = 0)$ or $(N_o = 2^d) \rightarrow$ Complement the first (2^{d-1}) bits and append $C_b = (2^{d-1} - 1)$ as check bits;
- $(2^{d-1}) \leq N_o \leq (2^d - 1) \rightarrow$ Append $C_b = N_o$ to the data word in binary (same as Berger code);
- $1 \leq N_o \leq (2^{d-1} - 1) \rightarrow$ Append $C_b = (N_o - 1)$ to the data word in binary.

Table A.2: Bose code for data words comprising 4 bits

Data word	Check bits	Codeword
0000	01	110001
0001	11	000111
0010	11	001011
0011	10	001110
0100	11	010011
0101	10	010110
0110	10	011010
0111	00	011100
1000	11	100011
1001	10	100110
1010	10	101010
1011	00	101100
1100	10	110010
1101	00	110100
1110	00	111000
1111	01	001101

Although Bose has succeeded to improve the efficient of the Berger's code, in digital circuits data can be represented with a large number of bits. Based on that, we can define a threshold 't' in such a way that we can neglect the probability of occurrence of more than 't' unidirectional errors in an output word. For these scenarios, we can use t-unidirectional error detecting codes (t-UED) – codes able to detect up to 't' faulty bits – instead of the codes shown above.

Based on that, Bose and Lin have proposed in 1985 various t-UED codes, known as Bose-Lin codes, in their work entitled "*Systematic unidirectional error-detecting codes*" [124]. Indeed, they developed optimal codes that require 2, 3 and 4 check bits in order to detect 2, 3 and 6 unidirectional errors, respectively. Considering that N_o denotes the number of 0's in a data word, the check bits (C_b) for each of the optimal codes can be evaluated as follows:

- *2-UED code* $\rightarrow C_b = N_o \bmod 4$
- *3-UED code* $\rightarrow C_b = N_o \bmod 8$
- *6-UED code* $\rightarrow C_b = (N_o \bmod 8) + 4$

Subsequently, in 1986, Bose have proposed a burst unidirectional error-detecting (BUED) code in [125].These codes can detect burst errors with length up to 2^{r-1} using only r check bits. The coding format uses the following relation to evaluate the check bits:

- *BUED code* $\rightarrow C_b = N_o \; mod \; 2^r$

The codes shown above were extensively used in totally self-checking (TSC) checker designs [126–129]. In the work presented in [126], easily-testable checker designs for three different codes were proposed: Bose-Lin, Bose and Blaum code. These checkers were proved to be TSC under the stuck-at fault model. Later, a modular method for designing checkers for Bose-Lin and Bose codes was proposed in [127], which resulted in designs more efficient regarding area and performance than previous approaches. Moreover, in the same work, these checkers were proved to be TSC under a more realistic fault model including: stuck-at, transistor stuck-on, transistor stuck-open, resistive bridging faults and breaks. In 2003, the work in [128] proposed TSC checkers for 3 different codes: Borden, Bose-Lin and Bose. The proposed checkers can perform well under very weak assumptions, what makes them perfect for use as embedded checkers. Afterwards, the same authors have also proposed checkers for the Bose AUED code in [129].

A.1.2.1.4 Borden code

The Borden code is an optimum t-unidirectional error detecting code proposed by Borden in 1982 [130]. This code is defined as follows:

Definition: *If* $C(m,n)$ *is the set of codes of length 'n' for which exactly 'm' bits are ones, then the union of all such codes with 'm' being the set of values congruent to* $\lfloor n/2 \rfloor \mathrm{mod}(t+1)$ *is known as the Borden* (n,t) *code* [115].

In order to illustrate this concept, let us assume $n = 12$ and $t = 2$. In this case, all values for 'm' that are congruent to the expression $\lfloor n/2 \rfloor \mathrm{mod}(t+1)$ are $m = 2, 6, 10$. This means that this code is composed by any word of length 12 of which either 2, 6, or 10 bits are 1, making these codes non-systematic in nature. The Borden code (n,t) can detect 't' unidirectional errors. Then, for $t = 1$, the Borden code becomes the parity code.

Several works have been done over the years to benefit from the optimality of Borden codes. A common way to efficient use such codes is designing self-checking circuits. In 1989, Jha proposed the first design for totally self-checking circuits based on Borden codes, improving their overall applicability [131]. Later, the work in [132] proposed a method to efficiently design self-checking circuits based on a class of Borden codes. Checkers designed with the proposed approach reduced implementation costs up to 98.5%. However, they are limited to a class of Borden codes. A systematic method of designing totally self-checking

circuits for a larger class of Borden codes (even code lengths) were proposed in [133]. They significantly reduced the implementation costs compared to Jha's method. In 2006, Tarnick presented a self-checking circuit based on Borden codes and AN arithmetic very suitable for use as embedded checkers [134].

A.1.2.2 Error correcting codes

Unlike EDCs, error correcting codes are codes that provides the ability to detect and locate n errors presented in a data word. Then, simple procedures can be carried out in order to correct the faults.

A.1.2.2.1 Hamming codes

Hamming codes were introduced in 1950 [135], and marked the beginning of the coding theory. The author was a pioneer computer scientist, and during his researches, he found that computers require error correcting capabilities. In order to do that, he created a sophisticated pattern based on parity checking that can detect up to two simultaneous bit errors or correct one, but not both simultaneously.

First of all, Hamming developed during his studies a nomenclature to describe systems using coding techniques. His idea was to describe this as an (n, k) code where 'n' stands for the number of bits of the codeword and 'k' stands for the number of data bits. Based on that, a system that is composed of words with 7 data bits and 1 parity bit is represented as an (8,7) code.

A Hamming code is then defined as an (n, k) code that respects the following relations: $k = 2^r - r - 1, \quad n = 2^r - 1, \quad r \geq 3$, where '$r$' represents the number of check bits. Considering that each bit in a codeword is numbered from 1 to n, the parity check bits are located in positions that are multiple of two. For example, if we have a codeword encoded with Hamming (7,4), the codeword structure is $P_1P_2D_1P_3D_2D_3D_4$, where P represents a parity bit and D a data bit.

In order to encode a specific data word using a Hamming code, we must first define $\mathbf{H} := (\mathbf{I_{n-k}}|\mathbf{A})$ as a parity checker matrix and $\mathbf{G} := (\mathbf{A^T}|\mathbf{I_k})$ as a generator matrix. The parity checker matrix comprises an identity matrix ($\mathbf{I_{n-k}}$) and a matrix ($\mathbf{A}$) that contains all nonzero binary combinations that do not appear in the columns of the neighboring identity matrix. For a Hamming (7,4) code, a possible parity checker matrix is shown below.

$$\mathbf{H} = \left(\begin{array}{ccc|cccc} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right)$$

The generator matrix $\mathbf{G}$ can be evaluated considering that $\mathbf{HG^T} = \mathbf{0}$. In this case, the corresponding generator matrix $\mathbf{G}$ is shown below.

$$\mathbf{G} = \left(\begin{array}{ccc|cccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right)$$

In order to demonstrate the procedure required to encode a data using a Hamming code, let us use both matrices defined above. First of all, we multiply the data word by the generator matrix $\mathbf{G}$. Considering that the data word is $\mathbf{w} = 1010$ and that we use even parity, the codeword can be obtained as follows:

$$\mathbf{c} = \mathbf{wG} = (1010)\left(\begin{array}{ccc|cccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right) = (1011010)$$

The decoding process starts checking if there is any error in the codeword. This procedure is done multiplying the received codeword by $\mathbf{H}$. The resulting value is called syndrome and identifies the faulty bit. If the codeword is correct, the syndrome must be zero. In order to demonstrate that, let us first assume that the received codeword is 1011010. In this case, the syndrome must be zero as shown below.

$$\mathbf{s} = \mathbf{cH^T} = (1011010) \left(\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \hline 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{array} \right) = (000)$$

Let us now suppose that the received codeword was 1011110, *i.e.*, the received codeword has one error located at bit 5. Then, we can evaluate the syndrome of this codeword and next locate the faulty bit using Table A.3. Once an error has been detected, corrective procedures such as scrubbing are usually carried out [136].

$$\mathbf{s} = \mathbf{cH^T} = (1011110) \left(\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \hline 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{array} \right) = (101)$$

Table A.3: Syndrome Table for Hamming (7,4) code [115]

Error Vector	Syndrome
1000000	100
0100000	010
0010000	001
0001000	110
0000100	101
0000010	011
0000001	111

Hamming codes are widely used in computer memory (RAM) because of their simplicity. The work in [67] evaluates the area efficiency of this technique, and points to a lower area increase when compared to TMR for codes capable to correct one bit-flip. Regarding multiple upsets, a technique based on Hamming code and Reed Solomon code that was emulated in a Virtex FPGA was introduced in [137]. For a memory composed of 128-bit rows, the proposed technique requires only 19 parity bits per row. In 2007, a new technique

to protect memories against multiple errors named *Matrix code* was proposed in [138]. It is based on Hamming and parity codes, and the idea is to organize the data bits into a matrix in such a way that rows are protected with Hamming code, meanwhile columns are protected with parity code. The results have shown that the relation detection/correction coverage per cost is better when comparing to both Reed-Muller and Hamming codes.

An important research in reconfigurable computers using Hamming codes was realized in Dassault Laboratories by Maison [139]. This work, called MECRA (*Maquette Expérimentale de Calculateur à Reconfiguration Automatique*), consisted in realizing an ultra-reliable and self-reconfigurable computer. It used a redundant Hamming code that can detect up to two simultaneous errors and spare blocks to provide a reconfigurability mechanism to correct errors. The MECRA project has shown a significant increase in the ratio reliability/cost for the proposed computer.

A.1.2.2.2 Cyclic codes

Cyclic codes are a special type of linear error correcting codes introduced by Prange in 1957 [140]. These codes were developed in such a way that both encoding and error correcting procedures can be easily implemented using shift-registers [141].

The main idea behind any code is to add extra bits in order to allow error detection and possible correction. Regarding cyclic codes, the codeword comprises k binary bits representing the data and $n-k$ check bits. Then, the codeword can be interpreted as a polynomial where each bit stands for a polynomial coefficient. As an example, a binary word such as $w = 101011$ can be interpreted as $w = x^5 + x^3 + x + 1$.

A cyclic code is based on a generator polynomial $P(x)$ of degree $n-k$. Any codeword that is divisible by $P(x)$ is considered as a valid codeword. Thus, in order to encode a data represented by $D(x)$ using a cyclic code, we must divide $x^{n-k}D(x)$ by $P(x)$, and add the remainder $R(x)$ to the result as shown in (A.1).

$$C(x) = \frac{x^{n-k}D(x)}{P(x)} + R(x) \tag{A.1}$$

An important aspect about cyclic codes is that all algebraic operations are done using module two arithmetic. Then, addition and subtraction provides the same result and the division operation can be drastically simplified. Indeed, it is well known that the division operation can be done using just shift registers and modulo two adders (a simple exclusive or) [141].

The ability of a cyclic code to detect and correct errors depends on the generator

polynomial used to encode a data word. The simplest polynomial with more than one term ($P(x) = x + 1$) is capable to detect any odd number of errors as shown in [141].

Subsequently to the introduction of the cyclic codes, several works were done based on it. In 1962, Lars-Henning Zetterberg proposed a family of binary cyclic codes based on Galois fields, discovered by the french mathematician Evariste Galois, which are capable to correct multiple errors [142]. Among them, Zetterberg codes with parameters $(2^u + 1, 2^u + 1 - 2u)$ deserve special attention due to their large code rate and their high decoding speed [143]. Indeed, Zetterberg codes with u odd were proved to be able either to correct all weight-2 errors with a small number of exceptions or become a truly 2-error correctable code multiplying the generator polynomials by $(x - 1)$ [144].

The most well-known family of cyclic-based codes is the Reed-Solomon family proposed in 1960 [145]. They are non-binary cyclic codes, which are constructed and decoded using finite field arithmetic. Reed-Solomon codes can detect up to t faulty bits when adding t check bits to the data. They are widely used in digital communication and storage systems. Another important capability of such code is the high reconfigurability. The work in [146] used this feature to propose a fault tolerant solid state mass memory for space applications. Using the proposed approach, the system can be reconfigured to deal either with latency problems (using smaller symbols) or data integrity (using longer symbols).

A.2 Evolvable hardware

Evolvable hardware is a new concept introduced by Adrian Thompson in 1995 [147] that uses evolutionary algorithms when designing a specific circuit. This technique can be used either to build circuits with high masking capability or to design reconfigurable circuits that can perform well in the presence of a fault. In fact, this concept brings together reconfigurable hardware, artificial intelligence, fault tolerance and autonomous systems.

Evolutionary algorithms are based on the biological evolution process. The main idea is that a population of different individuals competes among them to participate in the creation (reproduction process) of the next generation. This competition forms a selection mechanism equivalent to the natural selection mechanism described by Darwin in his evolution theory. In the case of circuits, each individual is a different electronic system attempting to solve the same problem [148]. The selection process is done by a measurable number, called fitness, which evaluates the quality of each solution. Then, all the designs that succeed the selection process are able to participate in the creation of the next generation of individuals.

Artificial evolution in circuits is generally used regarding two cases: system design and online adaptation of existing systems. In the first one, evolution algorithms are used

in order to create a hardware design that can perform a required action, but it stops changing after a good design is found (evolved hardware). Regarding fault tolerance, Adrian Thompson in [149] has shown that for some circumstances, the evolution process will automatically tend to create designs that are insensitive to some faults. The next one, however, continues the evolution process throughout its existence, allowing the system to adapt itself when an environmental change or an error occurs (evolvable hardware). One important point that we must highlight here is that the evolution process improves the system in a non-monotonic way, *i.e.*, some solutions in generation n can be worse than solutions in generation $n-1$. Therefore, precautions are required in order to keep the system functioning well during the evolution process.

In the last decade several researches were done regarding evolutionary-based techniques for fault tolerant systems. In 2000, the work in [150] used GA (genetic algorithm) to derive a population of different individuals for adaptation of electronic circuits. They compared two methods to achieve fault-tolerant designs: one based on fitness definition, and other based on population. In the fitness method, predefined faults are introduced during the evolutionary process, and the best fit individual is selected as a robust design solution. In the population-based approach, the evolutionary process is done without injecting errors a priori. Next, faults are injected in the best fit individual, and its performance is evaluated. If the fitness is too low for any injected fault, mutants are analyzed. If no mutant can provide acceptable behavior, the GA is restarted. Their simulation results have shown that the population-based design outperforms the fitness-based method in both analog and digital circuits.

In 2001, the work in [151] investigated the properties of messy gates, a model of gate-like components with added random noise, using evolutionary algorithms. The random designs created by these algorithms exploited redundancy in such a way that the evolved circuits exhibited implicit fault-tolerance to stuck-at-faults. In [152], the authors examined the ability of evolutionary techniques to include fault tolerance into the evolutionary process, and how it reacts in respect to real life faults. In 2003, the work presented in [153] applied repair techniques based on evolutionary algorithms into four circuits implemented using an FPGA: quadrature decoder, 3-by-3-bit multiplier, 3-by-3-bit adder, and 4-to-7 decoder. The fault simulation used was "hard-wiring" the individual LUTs values in order to simulate either stuck-at-0 or stuck-at-1 errors. The experimental results have pointed out evolutionary repair techniques as a great replacement or a supplement to traditional fault tolerance techniques such as TMR. Moreover, the increase in circuitry using evolutionary algorithms remains almost constant relative to the number of electronic components, unlike TMR that requires a linear increase in circuit area.

The work in [154] have introduced a method to take into account the correlation of

circuits during the fitness analysis of the evolutionary algorithm. Using this, it is possible to design an ensemble of circuits in such a way that the correlation in the fault pattern is reduced. They have shown that when using these uncorrelated circuits combined with an NMR approach, the reliability of the ensemble is increased. Later, the work in [155] reported the results of a comparison between the population-based and the correlation-based methods. The study points out the size of the ensemble of circuits required by the population-based method as a practical disadvantage when compared to the correlation-based method. Indeed, the fitness function used in the population algorithm leads to a set of circuits comprising one circuit performing well without faults and others performing well in respect to each detectable fault. In the worst case, the number of circuits can be $N + 1$ where N is the number of detectable faults. In the case of the correlation based-method, every circuit will provide an *acceptable* level of performance whether faults are present or not, leading to a much smaller size of the ensemble of circuits.

Appendix B

Basics on FPGAs

An FPGA (*Field Programmable Gate Array*) is an integrated circuit comprising configurable blocks and configurable interconnects. These configurable structures allow the designer to program the device in order to perform a desired function after the manufacturing process (field-programmable). FPGAs can be classified into two types:

- OTP (*One-Time Programmable*) FPGA: the device is designed in such a way that it can be programmed only one time.
- Reprogrammable FPGA: the device can be reprogrammed through software to perform another task.

In order to be able to program a circuit after manufacturing, certain mechanisms are required. In this chapter, examples of these mechanisms will be reviewed, and their prime characteristics will be revealed. First of all, technologies related to OTP devices will be introduced. Next, mechanisms that allow the fabrication of reconfigurable devices will be presented.

B.1 FPGA technologies

To explain the concepts related to FPGAs' technologies, a simple circuit, shown in Figure B.1, will be used. This figure represents a configurable circuit with potential links that can be used to define the circuit function.

B.1.1 Fusible link technology

The fusible link technology was one of the first mechanisms that allowed a designer to configure a circuit after manufacturing [156]. As shown in Figure B.2, the main idea is to use fuses to establish the connections between the elements. In the beginning, all the fuses are intact. In order to configure a device, a high voltage is applied in the desired fuses

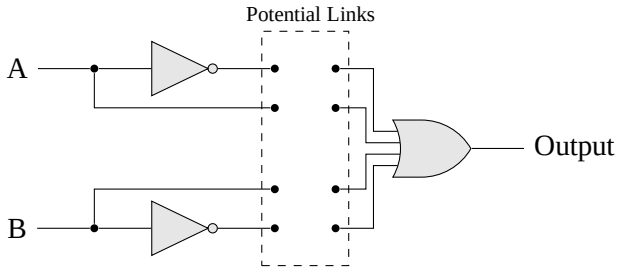


Figure B.1: Programmable circuit concept

to burn them out. An example of a programmed circuit is shown in Figure B.3. In this example, two fuses were burned out and the output provided now is $Y = A + \bar{B}$.

It can be noted that devices using fusible link technology are OTP (*One-Time Programmable*) because after burning a fuse out, there is no way to return to its original state.

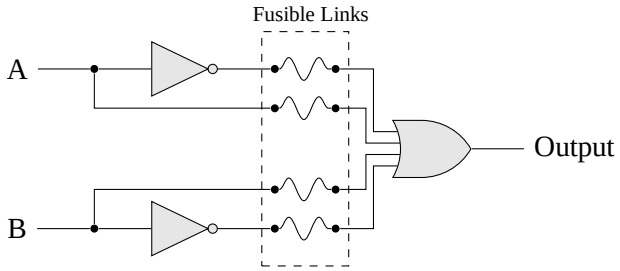


Figure B.2: Programmable circuit with intact fusible links

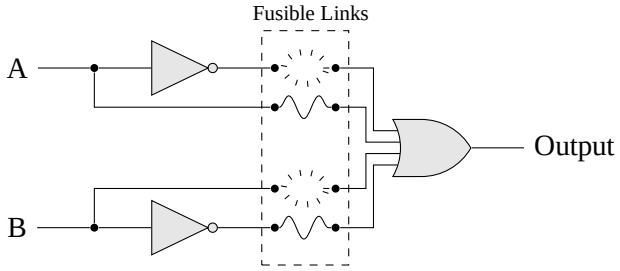


Figure B.3: Programmed circuit with output $Y = A + \bar{B}$

B.1.2 Antifuse

The antifuse technology emerged as an alternative choice to fusible link technology. It operates in an opposite way in respect to the fuses [45, 156]. The antifuses present a high resistance when not programmed and they can be interpreted as an open circuit, as shown

in Figure B.4. After a high voltage is applied, the resistance decreases and a link is formed in such a way that the device can be one-time programmed by the designer. An example of a programmed device with output $Y = A + \bar{B}$ can be seen in Figure B.5.

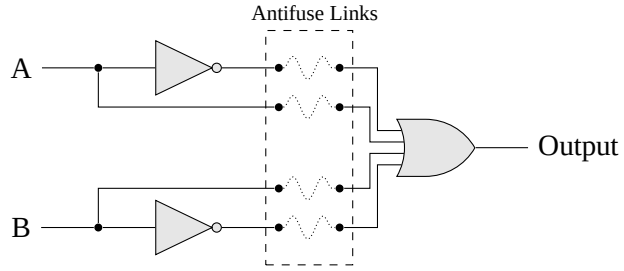


Figure B.4: Programmable circuit with intact antifuses

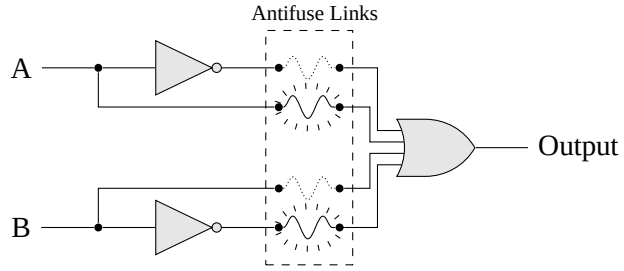


Figure B.5: Programmed circuit with output $Y = A + \bar{B}$

B.1.2.1 Actel antifuse technology - PLICE®

PLICE® (*Programmable Low-Impedance Circuit Element*) is an antifuse-based technology designed by Actel and available in some families of FPGAs. Each antifuse is constituted by a dielectric between an n+ layer and a poly-Si layer [157]. The behavior of these elements is to present a high resistance (normally bigger than 100 Megaohms) when not programmed and a low resistance (200-500 ohms) when programmed [158].

The configuration of these devices is done by applying a programming voltage in the desired antifuses in such a way that the dielectric is ruptured. Thus, a conductive link between the n+ and poly-Si layers is created. Notice that once programmed, an antifuse cannot be recovered to its original state. In other words, an antifuse FPGA is an one-time programmable device.

Nowadays Actel also uses metal-metal antifuses in some FPGA families. This type of antifuse mainly has two advantages over a poly-silicon antifuse. The first one is that poly-silicon devices require extra space to connect wiring layers, creating additional parasitic capacitance. The connections in metal-metal antifuses are stablished direct to metal

(the wiring layers). The second virtue of metal-metal antifuses is that they have smaller programming resistance (of approximately 25 ohms) [159].

B.1.2.2 Quicklogic antifuse technology - ViaLink®

ViaLink® is also an antifuse-based technology that uses an amorphous silicon layer to provide low resistances (of approximately 30 ohms) [160]. In order to programme this element, a voltage is applied to the top electrode and the bottom electrode is grounded. Thus, a conductive link is created by moving electrode material into the amorphous silicon [161]. A ViaLink® device offers a superior level of security, concerning intellectual property, when compared to SRAM-based FPGAs, ASICs or gate arrays [162]. Using this technology, the vendor assures that even if the FPGA is decapped, it is not possible to determine the location of programmed antifuses and relate this to design functionality [162]. This high level protection is provided mainly by the following characteristics:

- There is no serial data stream on power up, avoiding unauthorized copies of the program during boot up.
- Each antifuse contains a top similar to metal, making hard to distinguish whether or not an antifuse was programmed;
- QuickLogic devices contain a large number of elements, and the relationship among the programmed antifuses is proprietary information. Thus, if even just one antifuse is identified incorrectly during the reverse engineering process, the perfect functionality is compromised;
- The JTAG port available on the device has a security bit to avoid unauthorized access to the flip-flop values by thirds.

B.1.3 Static memory technology

Static Random Access Memory (SRAM) is a semiconductor memory widely used in FPGAs devices. The term *static* states that this type of memory do not need to be periodically refreshed in order to maintain its contained data, unlike dynamic memories. SRAM technology is based on static memory cells, such as the one shown if Figure B.6, that provide the reconfigurability mechanism. These cells are mainly used in multiplexers, in order to interconnect signals, and in LUTs (lookup-tables), used to implement logic functions in SRAM-based FPGAs [45].

Nowadays, the majority of FPGAs are based in static memory technology. Memory elements in SRAM-FPGAs can be divided as follows:

- *Configuration memory*: used to map a circuit into an FPGA. In other words, configuration bits are used to define interconnections, combinatorial functions, among

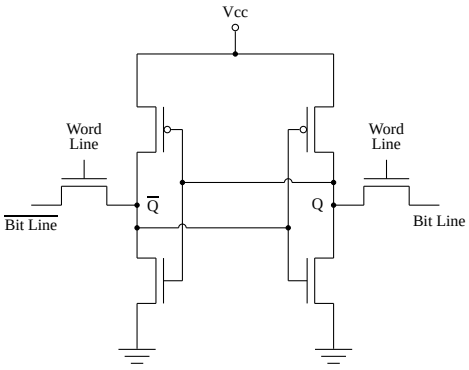


Figure B.6: Static Memory Cell

others. This memory comprises more than 99% of memory bits in an FPGA and is supposed to remain unchanged during execution [48].

– *User memory*: responsible for the current state of a circuit, i.e., the runtime information. The content of this memory is supposed to change during execution.

One of the foremost advantages in using SRAM-based devices is that it can be programmed in an indefinite number of times [45,156]. Besides, these cells are compatible with CMOS technology and no special circuit is needed. There are some drawbacks, however, presented in this technology that must be highlighted [45]:

– *Size*: The actual size of SRAM cells is bigger compared to techniques such as antifuses. A typical cell generally requires 6 transistors.

– *Volatility*: SRAM-based cells lose data information when power is down. Therefore, external devices, such as EEPROM or flash memories, are required to store the information when the FPGA is powered down, and then load it again when powered up.

– *Security*: Devices based on SRAM cells, when powered up, require a boot up process to perform their functionality. During this process, the information can be intercepted and stolen by unauthorized people. Thus, techniques such as encryption must be used to diminish this risk.

– *Electrical properties*: Multiplexers in SRAM-based FPGAs are implemented using transistors. Each transistor has significant resistance and capacitance that need to be considered when designing a circuit. Higher resistances and capacitances result in higher delays, thus reducing the system performance.

B.1.4 Flash technology

Flash memory technology emerged as a mix of EPROM and EEPROM technologies. The main difference is that in flash devices a large chunk of memory can be erased at one

time, while in EPROM/EEPROM devices only one bit is erased individually (so the term "flash" devices) [163].

Flash memory cells are made from floating gate transistors, a special type of transistor in which the gate is electrically isolated. This isolation creates a floating node in DC, which allows the transistor to keep the charge contained in it for long periods of time. Indeed, these cells do not lose their content even when the cell is powered down (nonvolatile memory). Using phenomena such as Fowler-Nordheim tunneling and hot carrier injection, it is possible to program the memory cell modifying the charge contained in the floating gate.

Flash-based memory cells can be used as programming elements in FPGAs in order to provide nonvolatility and reconfigurable programming, such as in Actel's ProASIC3 [164]. The programming cell of these devices comprises two transistors sharing a floating gate, which stores the programming information (see Figure B.7). The transistor located at the left side, named *sensing transistor*, is smaller and is responsible for writing and verification of the floating gate. The other transistor is named *switching transistor*, and it's used in 3 situations: to connect or separate routing nets, to configure VersaTile logic, and to erase the floating gate.

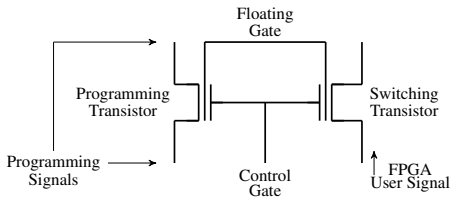


Figure B.7: Flash memory cell - ProASIC3

As mentioned earlier, flash-based technology provides nonvolatility to FPGAs. Therefore, flash-based FPGAs do not need external resources to store and load configuration data in contrast to SRAM-based devices. Indeed, a flash-based device is ready to perform its function upon power-up. Flash-based memory cells are also more area efficient than an SRAM-based cell, since the last one requires up to six transistors to implement the programmable storage.

Despite the virtues of flash-based FPGAs, some disadvantages are also presented in such devices. Perhaps the most important drawback is that they cannot be reprogrammed an infinite number of times. In fact, flash-based FPGAs such as the Actel's ProASIC3 have an endurance of 500 programming cycles and a program retention of 20 years for correct operating conditions [165].

B.1.5 Summary

The most important technologies used in FPGAs are: SRAM, Antifuse, and Flash. Table B.1 summarizes the prime characteristics of such technologies. It is important to note that an ideal technology would be a nonvolatile, reprogrammable, providing low resistance and parasitic capacitances, and using a standard CMOS process. None of the existent technologies can satisfy such requirements [45].

Table B.1: Programming technology properties summary [45]

	SRAM	**Flash**	**Antifuse**
Volatile	Yes	No	No
Reprogrammable	Yes	Yes	No
Storage Element Size	High	Moderate	Low
Manufacturing Process	Standard CMOS	Flash Process	Special Antifuse Process
In-System Programmable	Yes	Yes	No
Switch Resistance	$\sim$500 – 1000Ω	$\sim$500 – 1000Ω	$\sim$20 – 100Ω
Switch Capacitance	$\sim$1 – 2 fF	$\sim$1 – 2 fF	$<$ 1 fF
Programmable Yield	100%	100%	$>$ 90%

B.2 FPGAs architectures

In this Section, some FPGAs architectures from the most known vendors will be discussed.

B.2.1 Altera

B.2.1.1 Stratix® family

The Stratix® family is the high-end line of Altera FPGAs and is designed to achieve a high performance with low power consumption. In order to obtain this, important features were added over the generations of this family [166]:

- *Adaptive Logic Module* (ALM): Each ALM contains two 6-input LUT (Look-Up Table) that can be configured to work as an 8-input LUT. A high level block diagram of an ALM can be seen in Figure B.8.
- *Programmable Power Technology*: The new version of Stratix® FPGAs allows the configuration of speed paths. In other words, some blocks may run in higher speeds than others. This allows to speed up critical paths and slow down paths that do not need to run in high speeds, improving performance and power consumption;
- *Design Security*: Stratix® FPGAs use SRAM cells to store the configuration memory. Thus, each time the system is powered up, the data must be loaded. In order to

protect the intellectual property of the implemented design, it is possible to define a 128-bit or 256-bit key to encrypt the bitstream and avoid unauthorized copies.

– *Configurable High-Speed I/Os*: The I/O set can be configured according to a defined application, changing, for example, the electrical characteristics.

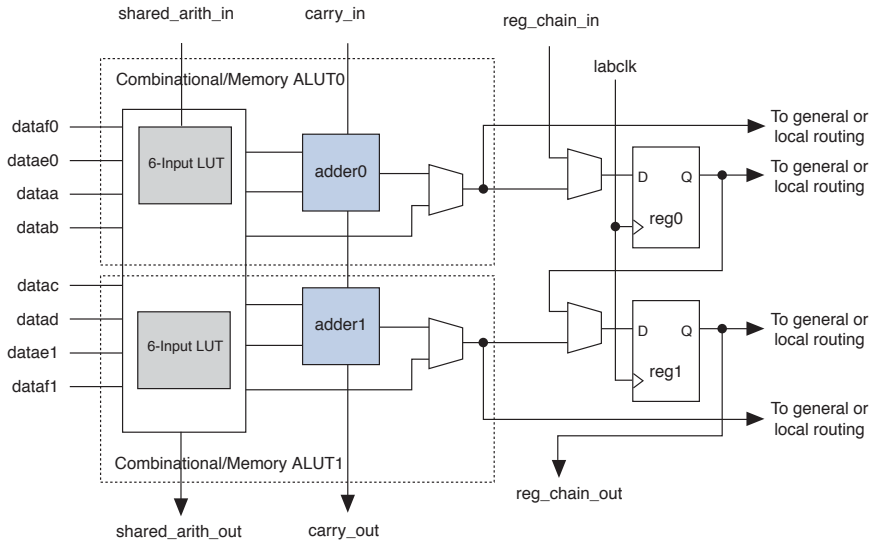


Figure B.8: ALM High-Level Block Diagram [166]

B.2.1.2 Arria® family

The Arria® Family is the midrange line of Altera FPGAs. It is designed as a cost and power sensitive device aiming transceiver-based applications. In order to provide fast interface connections for applications, Arria® FPGAs offer PCI Express, Gigabit Ethernet and Serial RapidIO interfaces. Other important features are available in this family [167]:

– *Logic Array Block* (LAB): Each LAB consists of 10 ALMs and other resources as carry chains, local interconnect and shared arithmetic. These FPGAs also dispose of MLABs that are basically LABs with SRAM-memory capability. These blocks allows a reduction in required routing, thus improving performance;

– *SEU Mitigation*: A built-in circuit may be used to detect data corruption due to soft errors in the configuration memory. Single Event Upsets (SEUs) were discussed in Section 1.2.2.

B.2.1.3 Cyclone® family

The Cyclone® Family is the low-cost FPGAs series offered by Altera. It was designed to provide power and cost savings as a result of features presented in its architecture as shown above [168]:

- *Logic Element* (LE): Logic elements use 4-input LUT to implement the logic functions. Each LE, illustrated in Figure B.9, can be configured to operate in two modes:
  - *Normal Mode*: Used for combinatorial functions and logic applications;
  - *Arithmetic Mode*: Used for implementing adders, counters, accumulators and comparators.
- *I/O Features*: The inputs and outputs of a Cyclone® FPGA can be configured according to the application. It can be chosen parameters such as bus hold, delay, pull-up resistors and even control the slew-rate to optimize signal integrity.

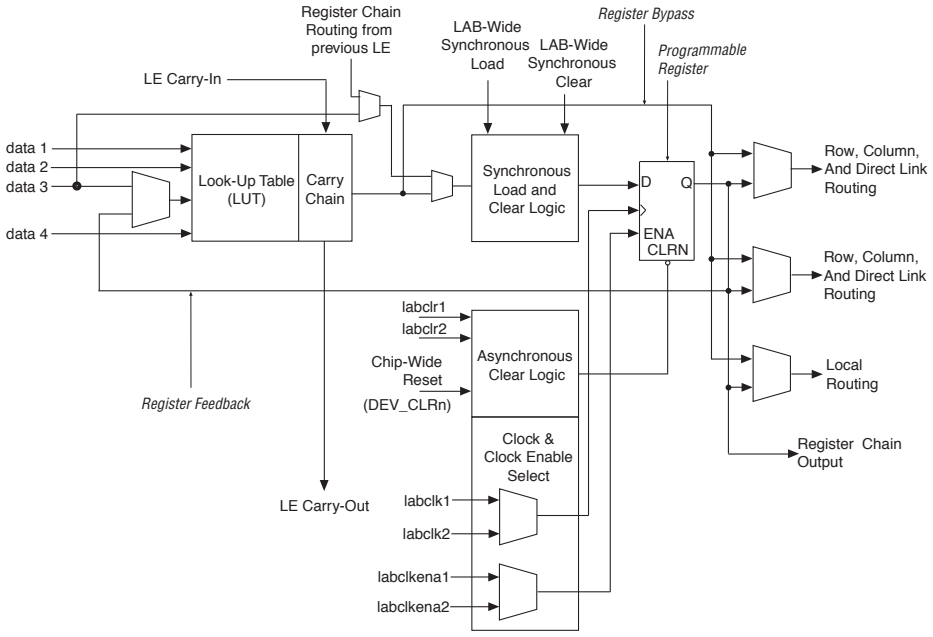


Figure B.9: LE Block Diagram [168]

B.2.1.4 HardCopy® family

HardCopy® are mask-programmable devices, called structured ASICs, that emerged as a midterm between FPGAs and ASICs. These devices use metal connections that are substantially smaller than that used in FPGAs, thus improving performance and reducing power consumption. Besides, unused logic blocks and clock trees are not powered up.

The HardCopy® architecture was designed in such a way that it is totally compatible with Stratix® FPGAs [169]. Moreover, the design tool is the same for HardCopy® and Stratix® devices, so allowing the use of reprogrammable FPGAs to design and test the circuit before make a "hard copy". The typical design flow can be resumed as follow:

- User develops and verifies a design using a Stratix® device;
- A netlist is created and sent to Altera;

– Altera sends the HardCopy® device using almost the same architecture of the initial design. Notice that the new device has smaller hard connections, thus improving performance and reducing power consumption.

B.2.1.5 Devices comparison

Table B.2: Altera Devices Comparison

Family	Logic Elements (KLEs)	Embedded Memory (Mbits)	Technology (nm)
Stratix IV	Up to 813.1	Up to 22.6	40
Stratix III	Up to 338.0	Up to 15.9	65
Stratix II	Up to 179.4	Up to 8.9	90
Hardcopy IV	Up to 813.1	Up to 20.2	40
Hardcopy III	Up to 338.0	Up to 15.9	40
Hardcopy II	Up to 179.4	Up to 8.4	90
Cyclone IV	Up to 149.8	Up to 6.3	60
Cyclone III	Up to 198.5	Up to 7.8	65
Cyclone II	Up to 68.4	Up to 1.1	90
Arria II	Up to 256.5	Up to 8.3	40
Arria	Up to 90.2	Up to 4.3	90

B.2.2 Xilinx

B.2.2.1 Virtex® family

Virtex® Family is the high-end line of FPGAs from Xilinx. It provides a high performance while keeping the power consumption low. The architecture used in this family of FPGAs is called ASMBL (*Advanced Silicon Modular Block*) [170] and it was first introduced in 2003. It allowed Xilinx to fast and cost-effectively assemble multiple platforms targeted to different application domains, providing the right choice of capabilities for a specific design. Thus, developers can select a platform with optimal features for a target application.

The ASMBL is a column-based architecture where each column represents a sub-system with specific capabilities, e.g., memory, I/Os, DSP, etc. Therefore, a domain-specific chip can be accomplished choosing the columns according to the desired functionality, as shown in Figure B.10. For example, a chip for speech processing must have more columns devoted to DSP functions than a chip targeted for an application in the logic domain. This architecture also alleviates problems such as:

– *I/O and Array Dependency*: The ASMBL architecture uses a flip-chip packaging process. It allows bonding pads to be located anywhere on the chip, not just on the periphery. Thus, a chip can accommodate more I/O pads just by devoting more

column to I/O functions [171];

- *Power and Ground Scaling*: In order to improve the power-grid distribution and to reduce the on-chip parasitic voltage drop, a chip design requires additional power and ground pads. The ASMBL architecture simplifies the task of uniform power distribution allowing to place the pads anywhere in the chip.
- *Hard-IP Scaling*: The Intellectual Property (IP) scaling problem is reduced from two dimensions to one dimension by the use of a column-based architecture.

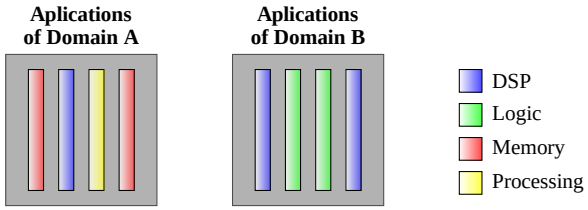


Figure B.10: Example of devices using the ASMBL architecture

B.2.2.2 Spartan® family

The Spartan® series of FPGAs from Xilinx are built as a low-cost and low-power device. The main resource for implementing combinatorial and sequential logic is the configurable logic block (CLB). A CLB comprises two vertical slices that can transfer data using a switch matrix as shown in Figure B.11. Each slice includes miscellaneous logic, eight flip-flops and four 6-input LUTs. There are three types of slices in Spartan-6 devices [172]:

- *SLICEM*: Each SLICEM can be used as LUTs (one 6-input or two 5-input), as memory (one 64-bit or two 32-bit RAM) or as shift registers (a single 32-bit or two 16-bit). Besides, for arithmetic operations, there is a high-speed carry chain to propagate the signals;
- *SLICEL*: Contains all the features of SLICEMs except the memory and shift register functions;
- *SLICEX*: Contains all the features of SLICELs except the arithmetic carry option and the wide multiplexers.

The clock management in Spartan-6 FPGAs is done by six Clock Management Tiles (CMTs). Each CMT has two Digital clock management (DCM) and one Phase-Locked Loop (PLL) that can be used individually or concatenated. Thus, digital designers can use either digital or analog clock management in Spartan-6 devices.

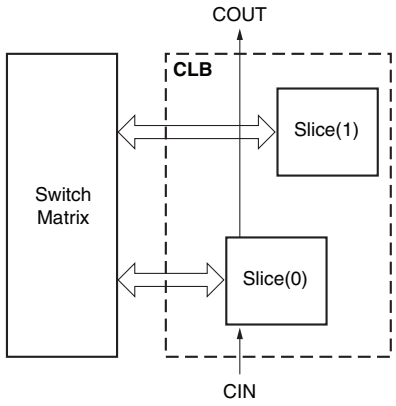


Figure B.11: High-level block diagram of a CLB in Spartan FPGAs [173]

B.2.2.3 EasyPath® family

EasyPath® devices are a midterm between FPGAs and ASICs that provide significant cost reduction with low-risk. The idea is to use Virtex® FPGAs to design and test a circuit and use the EasyPath® as an option to reduce its cost. This migration between devices can be done without additional design constraints. Both architectures are totally compatible and the cost savings are achieved by the reduction in the effective die size. In other words, unused gates, abundant routing and programmable multiplexers available in Virtex® FPGAs are removed [174].

The EasyPath-6 is the high-end device in this family and there is no minimal order quantities. After the submission of the compiled design files to Xilinx, the user will receive the EasyPath® devices in six weeks.

B.2.2.4 Devices comparison

Table B.3: Xilinx Devices Comparison

Family	Logic Cells (KCells)	Embedded Memory (Mbits)	Technology (nm)
Virtex-6	Up to 760	Up to 38	40
Virtex-5	Up to 330	Up to 18	65
Virtex-4	Up to 200	Up to 10	90
Spartan-6	Up to 150	Up to 4.8	45
Spartan-3	Up to 75	Up to 1.8	90

B.2.3 Actel

B.2.3.1 IGLOO® family

The IGLOO® family of FPGAs from Actel is well known due to its low power consumption. Basically, two technologies can be highlighted that reduce the amount of power consumption [175]:

- *Flash*Freeze*: This technology turns off I/Os and clocks to reduce the power consumption. When this mode is activated, the device consumes as little as $5\mu W$ and can rapid recovery to operation mode.
- *Low Power Active Capability*: The device remains completely functional while consuming just $12\mu W$.

An IGLOO® FPGA is a reprogrammable device that uses flash memory. Since this technology is nonvolatile, IGLOO® FPGAs do not need to load the information at power-up, avoiding unauthorized copies of the bitstream. Besides, it also includes a technology called FlashLock to hide its content until a host controller is able to authenticate itself through a 128-bit key.

The hardware architecture used in IGLOO® FPGAs is basically composed of VersaTiles. Each VersaTile can be configured as a 3-input function (3-input LUT), a D-type flip-flop or a latch.

In order to manage the clock signal, the IGLOO® device provides six CCCs (Clock Conditioning Circuit). One CCC has a PLL and the other five allow clock spine access and clock delay operations.

B.2.3.2 Fusion® family

Fusion® FPGAs are devices designed to mixed-signal integrated circuits, i.e., circuits that has both analog and digital on a single semiconductor die [176]. It also uses flash-based technology and its architecture is similar to the IGLOO® one. Fusion® FPGAs include analog components such as:

- *Configurable ADC*: it supports 8-, 10- and 12-bit modes;
- *32:1 Input Analog MUX*: channels 0 and 31 are dedicate. They can be used to monitor the core power supply and the device temperature, respectively.
- *Analog Quad I/O Structure*: Each structure can be used as one of various built-in circuit combinations.

B.2.3.3 ProASIC® family

The ProASIC® family was designed to provide a reprogrammable device with a cost per unit similar to an ASIC device. It uses nonvolatile flash technology and its architecture

is also similar to the IGLOO® one [177].

B.2.3.4 Axcelerator® family

Actel's Axcelerator® family was designed to provide high performance and high density. It uses an antifuse-based technology, denominated PLICE®, in which the programmable interconnect elements are located between two layers of metal. An important feature included in such devices is denominated *FuseLock technology* as a reference to the security level that is provided.

The Axcelerator® series uses the AX architecture that provides two types of logic modules [178] shown in Figure B.12:

- *Combinatorial Cell (C-Cell)*: Each C-cell can be configured to perform more than 4000 combinational functions up to 5-inputs.
- *Register Cell (R-Cell)*: Each R-cell contains a flip-flop with asynchronous clear and preset. Moreover, it features a programmable clock polarity that can be configured in a register-by-register basis.

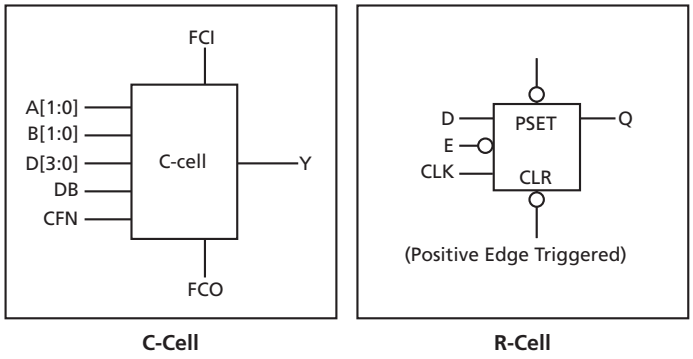


Figure B.12: AX C-Cell and R-Cell [178]

The AX architecture is organized into Core Tiles. A Core Tile is composed by an array of 336 SuperClusters and four SRAM blocks. Each SuperCluster comprises two Clusters. A Cluster is a block that contains two C-cells, a single R-cell and two Transmit (TX) and two Receive (RX) routing buffers. Superclusters are arranged in such a way that two combinational modules are side-by-side. This pattern, shown in Figure B.13, minimizes the delay for two-bit carry logic, thus improving arithmetic performance.

Relative to the I/O structure, every Axcelerator® device supports a range of operation voltages ($1.5V$, $1.8V$, $2.5V$ and $3.3V$). The Axcelerator® I/Os are organized in banks and it is possible to configure the I/O structure according to at least 14 different standards.

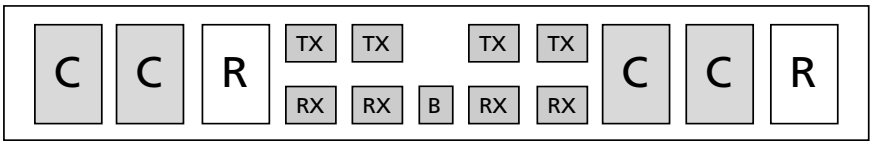


Figure B.13: AX SuperCluster Arrangement [178]

B.2.3.5 MX family

Actel MX family offers a cost-effective design solution at 5V. It uses antifuse technology and its capacity ranges from 3,000 to 54,000 gates. The architecture of MX devices are composed basically by [179]:

- *Logic Modules*: Each MX device contains three types of logic modules:
  - *Combinatorial (C-module)*: this block can be configured to implement a combinatorial logic function;
  - *Sequential (S-module)*: an S-module can implement a combinatorial logic function with the addition of a sequential element;
  - *Decode (D-module)*: these modules contain wide-decode circuitry and are arranged around the periphery of the device.
- *Dual-Port SRAM Modules*: the SRAM modules of MX devices contain independent read and write ports with independent clocks. As a result, these modules may be used to implement high-speed buffered applications. Unused modules can be used to implement other user logic.

B.2.3.6 Devices comparison

Table B.4: Actel Devices Comparison

Family	System Gates (x1000)	Embedded RAM (Kbits)	Technology (nm)
IGLOO	Up to 3000	Up to 504	130
ProASIC3	Up to 3000	Up to 504	130
Fusion	Up to 1500	Up to 270	130
Axcelerator	Up to 2000	Up to 295	150
MX	Up to 54	Up to 2.5	450

B.2.4 Lattice

B.2.4.1 EC® family

Lattice EC® family of FPGAs was designed using SRAM technology to provide devices with high performance while keeping the cost as low as possible. Its architecture is

composed basically by [180]:

- *Logic Modules*: EC® devices contain logic blocks organized in a two-dimensional array. There are two kinds of logic modules:
  - *Programmable Functional Unit (PFU)*: it can be configured to perform logic, arithmetic, RAM and ROM functions. Each PFU contains 4 slices. The first three slices are composed of two 4-input LUTs and a register. The remaining one comprises only two 4-input LUTs. A typical PFU is depicted in Figure B.14;
  - *Programmable Functional Unit Without RAM (PFF)*: it contains building blocks for logic, arithmetic and ROM functions.
- *sysMEM EBR Blocks*: A sysMEM Embedded Block Ram can implement either a single port, a dual port, or a pseudo dual port memory. It is a dedicated memory block that can be configured as RAM or as ROM.
- *sysDSP Slice*: this slice allows the implementation of typical functions for digital signal processing applications such as Finite Impulse Response filters, Fast Fourier Transform, among others. Each DSP slice supports different data widths to provide highly parallel implementations for DSP functions.
- *Serializer and Deserializer Channels (SERDES)*: each SERDES module contains independent 8bit/10bit encoder/decoder that allows to serialize and/or deserialize data.

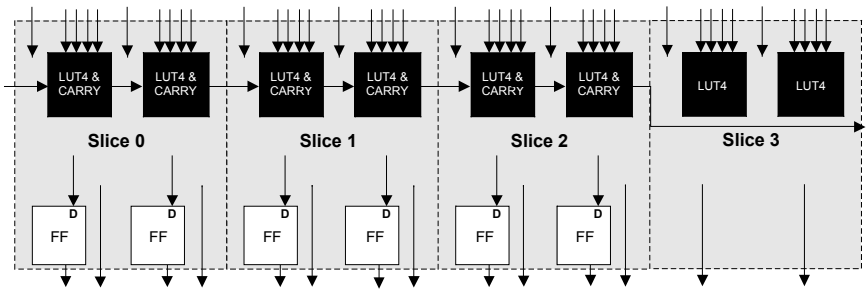


Figure B.14: PFU block diagram [180]

B.2.4.2 XP® family

Lattice XP® family provides devices with a flash-based architecture denominated flexiFLASH. As a result, these devices are instant-on, i.e., they do not need to load data during the boot-up process.

The flexiFLASH architecture is arranged in a similar manner in respect to the EC® architecture shown above.

B.2.4.3 Devices comparison

Table B.5: Lattice Devices Comparison

Family	**LUTs (x1000)**	**Embedded RAM (Mbits)**	**Technology (nm)**
ECP3	Up to 149	Up to 6.8	65
ECP2	Up to 95	Up to 5.3	90
XP2	Up to 40	Up to 0.9	90

Bibliography

[1] J. Larus, "Spending Moore's dividend," *Communications of the ACM*, vol. 52, June 2009.

[2] G. Moore, *Cramming more components onto integrated circuits, Electronics, Volume 38, Number 8, April 19, 1965.* January 1965.

[3] M. Dragoman and D. Dragoman, *Nanoelectronics: Principles and Devices.* Artech House Publishers, 2nd ed., October 2008.

[4] ITRS2011, "International technology roadmap for semiconductors." available: http://www.itrs.net/links/2011ITRS/Home2011.htm, 2011. [Online; accessed 09-July-2012].

[5] M. Stanisavljevic, A. Schmid, and Y. Leblebici, "Optimization of the averaging reliability technique using low redundancy factors for nanoscale technologies," *IEEE Transactions on Nanotechnology*, vol. 8, pp. 379–390, June 2009.

[6] M. Stan, P. Franzon, S. Goldstein, J. Lach, and M. Ziegler, "Molecular electronics: from devices and interconnect to circuits and architecture," *Proceedings of the IEEE*, vol. 91, pp. 1940–1957, January 2003.

[7] M. Breuer, S. Gupta, and T. Mak, "Defect and error tolerance in the presence of massive numbers of defects," *IEEE Design & Test of Computers*, vol. 21, pp. 216–227, June 2004.

[8] D. T. Franco, *Signal Reliability of Combinational Logic Circuits under Multiple Simultaneous Faults.* PhD thesis, December 2008.

[9] J. Vial, A. Virazel, A. Bosio, P. Girard, C. Landrault, and S. Pravossoudovitch, "Is triple modular redundancy suitable for yield improvement?," *IET Computers & Digital Techniques*, vol. 3, pp. 581–592, December 2009.

[10] M. Mirza-Aghatabar, M. Breuer, S. Gupta, and S. Nazarian, "Theory of redundancy for logic circuits to maximize yield/area," *in Proceedings of 13th International Symposium on Quality Electronic Design (ISQED)*, pp. 663 –671, March 2012.

[11] International Electrotechnical Commission (IEC), "IEC 62566: Nuclear power plants – Instrumentation and control important to safety – Development of HDL-programmed integrated circuits for systems performing category a functions," January 2012.

[12] International Electrotechnical Commission (IEC), "IEC 60987: Nuclear power plants – Instrumentation and control important to safety – Hardware design requirements for computer-based systems," August 2007.

[13] International Electrotechnical Commission (IEC), "IEC 61513: Nuclear power plants – Instrumentation and control important to safety – General requirements for systems," August 2011.

[14] J.-C. Laprie, ed., *Dependability: Basic concepts and Terminology*, vol. 5th of *Dependable Computing and Fault-Tolerant Systems*. Springer-Verlag Publisher, 1992.

[15] 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.

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

[17] M. Jeitler, M. Delvai, and S. Reichor, "FuSE - a hardware accelerated HDL fault injection tool," *in Proceedings of 5th Southern Conference on Programmable Logic (SPL)*, pp. 89–94, May 2009.

[18] P. Samudrala, J. Ramos, and S. Katkoori, "Selective triple modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs," *IEEE Transactions on Nuclear Science*, vol. 51, pp. 2957–2969, January 2004.

[19] Synopsys Armenia Educational Department, "SAED 90nm generic library." available: http://www.synopsys.com/Community/UniversityProgram. [Online; accessed 16-July-2012].

[20] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the ISCAS-85 benchmarks: a case study in reverse engineering," *IEEE Design & Test of Computers*, vol. 16, pp. 72–80, January 1999.

[21] S. Mitra and E. McCluskey, "Word-voter: a new voter design for triple modular redundant systems," *in Proceedings of 18th IEEE VLSI Test Symposium*, pp. 465–470, January 2000.

[22] A. R. Burks and A. W. Burks, *The First Electronic Computer: The Atanasoff Story*. December 1989.

[23] J. V. Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," *Automata Studies*, pp. 1–56, January 1956.

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

[25] P. Lala, "Transient and permanent fault injection in VHDL description of digital circuits," *Circuits and Systems*, pp. 192–199, February 2012.

[26] N. Miskov-Zivanov and D. Marculescu, "Multiple transient faults in combinational and sequential circuits: A systematic approach," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, pp. 1614–1627, October 2010.

[27] S. Nascimento Pagliarini, L. Alves de Barros Naviner, and J.-F. Naviner, "Selective Hardening Methodology for Combinational Logic," *in Proceedings of IEEE Latin-American Test Workshop (LATW)*, April 2012.

[28] A. Birolini, *Quality and reliability of technical systems: theory, practice, management.* Springer-Verlag Publisher, 1st ed., May 1994.

[29] P. D. T. O'Connor and A. Kleyner, *Practical reliability engineering.* John Wiley & Sons Ltd, 5th ed., 2012.

[30] M. Nanda and S. Rao, "A modified and effective system-engineering life cycle for critical systems," *in Proceedings of 4th Annual IEEE Systems Conference*, pp. 103 –108, april 2010.

[31] ITEM Software, "Reliability prediction basics." available: http://www.reliabilityeducation.com/ReliabilityPredictionBasics.pdf, 2007. [Online; accessed 09-July-2012].

[32] "Early Life Failure Rate Calculation Procedure for Semiconductor Components," Standard JESD74A, JEDEC Solid State Technology Association, USA, 2007.

[33] J. Plante, "Alternative test methods for electronic parts," tech. rep., NASA Electronic Parts and Packaging (NEPP) Program, 2004.

[34] "Test Method - Microcircuits - Method 1005.9," Standard MIL-STD-883H, Department of Defense, USA, 2010.

[35] D. J. Smith, *Reliability, Maintainability and Risk: Practical Methods for Engineers.* Butterworth-Heinemann Publisher, 8th ed., 2011.

[36] C. Constantinescu, "Trends and challenges in VLSI circuit reliability," *IEEE Micro*, vol. 23, pp. 14–19, July 2003.

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

[38] C. Constantinescu, "Impact of deep submicron technology on dependability of VLSI circuits," *in Proceedings International Conference on Dependable Systems and Networks*, pp. 205–209, February 2002.

[39] C. Constantinescu, "Dependability benchmarking using environmental test tools," *in Proceedings of Reliability and Maintainability Symposium (RAMS)*, pp. 567–571, January 2005.

[40] C. Constantinescu, "Intermittent faults and effects on reliability of integrated circuits," *in Proceedings of Reliability and Maintainability Symposium (RAMS)*, pp. 370–374, February 2008.

[41] R. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Transactions on Device and Materials Reliability*, vol. 5, pp. 305–316, September 2005.

[42] S. Martinie, J. L. Autran, S. Sauze, D. Munteanu, S. Uznanski, P. Roche, and G. Gasiot, "Underground experiment and modeling of alpha emitters induced soft-error rate in cmos 65 nm sram," *IEEE Transactions on Nuclear Science*, no. 99, p. 1, 2012.

[43] T. Merelle, F. Saigne, B. Sagnes, G. Gasiot, P. Roche, T. Carriere, and M.-C. Palau, "Alpha induced SEU and MBU rates evaluation for advanced srams by monte-carlo simulations," *in Proceedings of 8th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, pp. E3–1 –E3–6, September 2005.

[44] "Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices," Standard JESD89A, JEDEC Solid State Technology Association, USA, 2006.

[45] I. Kuon, R. Tessier, and J. Rose, "Fpga architecture: Survey and challenges," *Foundations and Trends® in Electronic Design Automation*, vol. 2, pp. 135–253, January 2007.

[46] C. Maxfield, ed., *FPGAs: World Class Designs*. Newnes Publisher, 1st ed., April 2009.

[47] J. McCollum, "ASIC versus antifuse FPGA reliability," pp. 1–11, March 2009.

[48] H. Asadi, M. Tahoori, B. Mullins, D. Kaeli, and K. Granlund, "Soft error susceptibility analysis of SRAM-based FPGAs in high-performance information systems," *IEEE Transactions on Nuclear Science*, vol. 54, pp. 2714–2726, December 2007.

[49] "Understanding single event effects (SEEs) in FPGAs," tech. rep., Microsemi - Actel, September 2011.

[50] "FPGA reliability and the sunspot cycle," tech. rep., Microsemi - Actel, September 2011.

[51] D. White, "Considerations surrounding single event effects in FPGAs, ASICs, and processors," tech. rep., Xilinx FPGAs, March 2012.

[52] E. J. McCluskey and F. W. Clegg, "Fault equivalence in combinatorial logic networks," *IEEE Transactions on Computers*, vol. C-20, pp. 1286–1293, November 1971.

[53] K. C. Y. Mei, "Bridging and stuck-at faults," *IEEE Transactions on Computers*, vol. C-23, pp. 720–727, July 1974.

[54] R. Ogus, "The probability of a correct output from a combinational circuit," *IEEE Transactions on Computers*, vol. C-24, pp. 534–544, June 1975.

[55] K. N. Patel, J. Hayes, and I. Markov, "Evaluating circuit reliability under probabilistic gate-level fault models," *in Proceedings of the International Workshop on Logic and Synthesis*, pp. 59–64, 2003.

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

[57] F. Faure, R. Velazco, and P. Peronnard, "Single-event-upset-like fault injection: a comprehensive framework," *IEEE Transactions on Nuclear Science*, vol. 52, pp. 2205–2209, December 2005.

[58] 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 Proceedings of 11th Latin American Test Workshop (LATW)*, pp. 1–5, January 2010.

[59] 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," *IEEE Transactions on Computers*, vol. 61, pp. 313 –322, March 2012.

[60] R. E. Lyons and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM Journal of Research and Development*, vol. 6, pp. 200–209, January 1962.

[61] N. Pippenger, "Developments in "the synthesis of reliable organisms from unreliable components"," *in Proceedings of Symposia in Pure Mathematics*, vol. 50, pp. 311–324, January 1990.

[62] N. Aymerich, S. D. Cotofana, and A. Rubio, "Adaptive fault-tolerant architecture for unreliable technologies with heterogeneous variability," *IEEE Transactions on Nanotechnology*, vol. 11, pp. 818 –829, July 2012.

[63] E. J. McCluskey, J. F. Wakerly, and R. C. Ogus, "Technical report," *Stanford*, pp. 1–110, January 1975.

[64] M. M. Dickinson, J. B. Jackson, and G. C. Randa, "Saturn V launch vehicle digital computer and data adapter," *AFIPS '64 (Fall, part I): Proceedings of the October 27-29, 1964, fall joint computer conference, part I*, November 1964.

[65] L. Edmonds, "Analysis of single-event upset rates in triple-modular redundancy devices," *JPL Publication 09-6 – National Aeronautics and Space Administration (NASA)*, February 2009.

[66] M. Cohn, "Redundancy in complex computers," *in Proceedings of the National Conference on Aeronautical Electronics*, May 1956.

[67] R. Hentschke, F. Marques, F. Lima, L. Carro, A. Susin, and R. Reis, "Analyzing area and performance penalty of protecting different digital modules with hamming code and triple modular redundancy," *in Proceedings of 15th Symposium on Integrated Circuits and Systems Design*, pp. 95–100, January 2002.

[68] L. Sterpone and M. Violante, "Analysis of the robustness of the tmr architecture in sram-based fpgas," *IEEE Transactions on Nuclear Science*, vol. 52, pp. 1545–1549, November 2005.

[69] X. She and S. Trimberger, "Scheme to minimise short effects of single-event upsets in triple-modular redundancy," *IET Computers & Digital Techniques*, vol. 4, pp. 50–55, January 2010.

[70] L. Sterpone and N. Battezzati, "A new placement algorithm for the mitigation of multiple cell upsets in SRAM-based FPGAs," *in Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1231–1236, January 2010.

[71] A. Grnarov, J. Arlat, and A. Avizienis, "Modeling of software fault-tolerance strategies," *in Proceedings of the 11th Annual Pittsburgh Modeling and Simulation Conference*, pp. 571–578, May 1980.

[72] R. B. Broen, "New voters for redundant systems," *Transactions of ASME, Journal of Dynamic Systems, Measurement, and Control*, March 1975.

[73] P. R. Lorczak, A. K. Caglayan, and D. E. Eckhardt, "A theoretical investigation of generalized voters for redudant systems," *in Proceedings of IEEE International Symposium on Fault-Tolerant Computing Systems*, pp. 444–451, January 1989.

[74] G. Latif-Shabgahi and S. Bennett, "Adaptive majority voter: a novel voting algorithm for real-time fault-tolerant control systems," *in Proceedings of 25th EUROMICRO Conference*, vol. 2, pp. 113–120, 1999.

[75] G. Latif-Shabgahi, J. M. Bass, and S. Bennett, "History-based weighted average voter: A novel software voting algorithm for fault-tolerant computer systems," *in Proceedings of 9th Euromicro Workshop on Parallel and Distributed Processing*, pp. 1–8, January 2001.

[76] Y. Dotan, N. Levison, R. Avidan, and D. Lilja, "History index of correct computation for fault-tolerant nano-computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, pp. 943–952, August 2009.

[77] J. Bass, G. Latif-Shabgahi, and S. Bennett, "Experimental comparison of voting algorithms in cases of disagreement," in *Proceedings of the 23rd EUROMICRO Conference*, pp. 516–523, September 1997.

[78] C. Zhao, Y. Zhao, and S. Dey, "Intelligent robustness insertion for optimal transient error tolerance improvement in VLSI circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, pp. 714–724, June 2008.

[79] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin, "Improving FPGA design robustness with partial TMR," *in Proceedings of 44th Annual IEEE International Reliability Physics Symposium*, pp. 226–232, March 2006.

[80] C. Zoellin, H. Wunderlich, I. Polian, and B. Becker, "Selective hardening in early design steps," *European Test, 2008 13th*, pp. 185–190, June 2008.

[81] X. She and P. K. Samudrala, "Selective triple modular redundancy for single event upset (SEU) mitigation," *in Proceedings of NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pp. 344–350, January 2009.

[82] O. Ruano and J. Maestro, "A methodology for automatic insertion of selective TMR in digital circuits affected by SEUs," *IEEE Transactions on Nuclear Science*, vol. 56, pp. 2091–2102, January 2009.

[83] I. Polian and J. Hayes, "Selective hardening: Toward cost-effective error tolerance," *IEEE Design & Test of Computers*, vol. 28, pp. 54 –63, May-June 2011.

[84] M. Augustin, M. Gossel, and R. Kraemer, "Selective fault tolerance for finite state machines," *in Proceedings of IEEE 17th International On-Line Testing Symposium (IOLTS)*, pp. 43–48, July 2011.

[85] I. Polian, D. Nowroth, and B. Becker, "Identification of critical errors in imaging applications," *in Proceedings of 13th IEEE International On-Line Testing Symposium (IOLTS)*, pp. 201–202, July 2007.

[86] D. Nowroth, I. Polian, and B. Becker, "A study of cognitive resilience in a JPEG compressor," *in Proceedings of 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 32–41, 2008.

[87] W. Zhang and T. Li, "Microarchitecture soft error vulnerability characterization and mitigation under 3d integration technology," *in Proceedings of 41st IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 435–446, December 2008.

[88] L. Antoni, R. Leveugle, and M. Feher, "Using run-time reconfiguration for fault injection in hardware prototypes," *in Proceedings of 17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, pp. 245–253, 2002.

[89] E. C. Marques, N. M. Paiva, L. A. B. Naviner, and J. F. Naviner, "A new fault generator suitable for reliability analysis of digital circuits," *in Proceedings of Argentine School of Micro-Nanoelectronics Technology and Applications (EAMTA)*, pp. 41–45, January 2010.

[90] J. Boue, P. Petillon, and Y. Crouzet, "MEFISTO-L: a VHDL-based fault injection tool for the experimental assessment of fault tolerance," *in Proceedings of Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing*, pp. 168–173, January 1998.

[91] J. C. Baraza, J. Gracia, D. Gil, and P. J. Gil, "Improvement of fault injection techniques based on VHDL code modification," *in Proceedings of Tenth IEEE International High-Level Design Validation and Test Workshop*, pp. 19–26, December 2005.

[92] K.-T. Cheng, S.-Y. Huang, and W.-J. Dai, "Fault emulation: A new methodology for fault grading," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, pp. 1487–1495, January 1999.

[93] D. Kammler, J. Guan, G. Ascheid, R. Leupers, and H. Meyr, "A fast and flexible platform for fault injection and evaluation in verilog-based simulations," *in Proceedings of Third IEEE International Conference on Secure Software Integration and Reliability Improvement (SSIRI)*, pp. 309–314, July 2009.

[94] I. Mavroidis and I. Papaefstathiou, "Accelerating hardware simulation: Testbench code emulation," *in Proceedings of International Conference on ICECE Technology (FPT)*, pp. 129–136, December 2008.

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

[96] V. Chandra and R. Aitken, "Impact of technology and voltage scaling on the soft error susceptibility in nanoscale cmos," *in Proceedings of IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems (DFTVS)*, pp. 114–122, January 2008.

[97] M. K. Goparaju, A. K. Palaniswamy, and S. Tragoudas, "A fault tolerance aware synthesis methodology for threshold logic gate networks," *in Proceedings of IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems (DFTVS)*, pp. 176–183, January 2008.

[98] 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 Proceedings of 44th Annual IEEE International Reliability Physics Symposium*, pp. 217–225, January 2006.

[99] ITRS2001, "International technology roadmap for semiconductors." available: http://www.itrs.net/Links/2001ITRS/Design.pdf, 2001. [Online; accessed 12-July-2012].

[100] M. Breuer, "An illustrated methodology for analysis of error tolerance," *IEEE Design & Test of Computers*, February 2008.

[101] E. C. Marques, G. G. S. Junior, L. A. B. Naviner, and J. F. Naviner, "Effective metrics for reliability analysis," *in Proceedings of 53rd IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 237–240, August 2010.

[102] G. dos Santos, E. Marques, and L. Naviner, "Using error tolerance of target application for efficient reliability improvement of digital circuits," *Microelectronics Reliability*, January 2010.

[103] S. Pagliarini, G. dos Santos, L. de B. Naviner, and J.-F. Naviner, "Exploring the feasibility of selective hardening for combinational logic," *Microelectronics Reliability*, no. 0, 2012.

[104] T. Ban and L. Naviner, "Progressive module redundancy for fault-tolerant designs in nanoelectronics," *Microelectronics Reliability*, vol. 51, no. 9–11, pp. 1489–1492, 2011.

[105] L. A. B. Naviner, J. F. Naviner, G. G. dos Santos Jr., E. C. Marques, and N. M. P. Jr., "FIFA: A fault-injection–fault-analysis-based tool for reliability assessment at RTL level," *Microelectronics Reliability*, vol. 51, pp. 1459–1463, October 2011.

[106] E. C. Marques, L. A. de Barros Naviner, and J.-F. Naviner, "An efficient tool for reliability improvement based on tmr," *Microelectronics Reliability*, vol. 50, no. 9–11, pp. 1247–1250, 2010.

[107] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinatorial benchmark circuits and a target translator in FORTRAN," *in Proceedings of International Symposium on Circuits and Systems*, pp. 663–698, June 1985.

[108] I. Polian, S. Reddy, and B. Becker, "Scalable calculation of logical masking effects for selective hardening against soft errors," *in Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 257 –262, April 2008.

[109] Q. Zhou and K. Mohanram, "Cost-effective radiation hardening technique for combinational logic," in *in Proceedings of IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 100–106, November 2004.

[110] X. Wang, "Partitioning triple modular redundancy for single event upset mitigation in FPGA," *in Proceedings of International Conference on E-Product E-Service and E-Entertainment (ICEEE)*, pp. 1–4, November 2010.

[111] B. Bridgford, C. Carmichael, and C. Wei Tseng, "Single-event upset mitigation selection guide," tech. rep., Xilinx, March 2008.

[112] F. Kastensmidt, L. Sterpone, L. Carro, and M. Reorda, "On the optimal design of triple modular redundancy logic for sram-based fpgas," *in Proceedings of Design, Automation and Test in Europe (DATE)*, vol. 2, pp. 1290–1295, March 2005.

[113] N. K. Jha and S. J. Wang, "Design and synthesis of self-checking VLSI circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, pp. 878–887, June 1993.

[114] T. Stankovic, M. Stojcev, and G. Djordjevic, "Design of self-checking combinational circuits," *in Proceedings of 6th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Service (TELSIKS)*, vol. 2, pp. 763–768 vol.2, January 2003.

[115] M. Abd-El-barr, *Design And Analysis of Reliable And Fault-tolerant Computer Systems*. World Scientific Pub Co Inc, 1st ed., 2006.

[116] N. Touba and E. McCluskey, "Logic synthesis of multilevel circuits with concurrent error detection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, pp. 783–789, January 1997.

[117] J. Berger, "A note on error detection codes for asymmetric channels," *Information and Control*, vol. 4, no. 1, pp. 68–73, 1961.

[118] D. Pradhan, "A new class of error-correcting/detecting codes for fault-tolerant computer applications," *IEEE Transactions on Computers*, vol. C-29, pp. 471–481, January 1980.

[119] M. J. Ashjaee, "Totally-self-checking check circuits for separable codes," *Ph.D. Thesis Iowa Univ., Iowa City.*, August 1976.

[120] M. A. Marouf and A. D. Friedman, "Design of self-checking checkers for Berger codes," *in Proceedings of International Symposium on Fault-Tolerant Computing (FTCS)*, pp. 179–184, June 1978.

[121] S. Piestrak, "Comments on "novel totally self-checking Berger checker designs based on generalized Berger code partitioning"," *IEEE Transactions on Computers*, vol. 51, pp. 735–736, January 2002.

[122] T. Rao, G. Feng, M. Kolluru, and J. Lo, "Novel totally self-checking Berger code checker designs based on generalized Berger code partitioning," *IEEE Transactions on Computers*, vol. 42, pp. 1020–1024, January 1993.

[123] S. Piestrak, "Design method of a class of embedded combinational self-testing checkers for two-rail codes," *IEEE Transactions on Computers*, vol. 51, pp. 229–234, January 2002.

[124] B. Bose and D. J. Lin, "Systematic unidirectional error-detecting codes," *IEEE Transactions on Computers*, vol. C-34, pp. 1026–1032, January 1985.

[125] B. Bose, "Burst unidirectional error-detecting codes," *IEEE Transactions on Computers*, vol. C-35, pp. 350–353, January 1986.

[126] N. Jha, "Design of totally self-checking checkers for Bose-Lin, Bose and Blaum codes," *in Proceedings of the 32nd Midwest Symposium on Circuits and Systems*, pp. 32–35 vol.1, January 1989.

[127] X. Kavousianos and D. Nikolos, "Modular TSC checkers for Bose-Lin and Bose codes," *in Proceedings of 17th IEEE VLSI Test Symposium*, pp. 354–360, 1999.

[128] S. Tarnick, "Self-testing embedded checkers for Bose-lin, Bose, and a class of Borden codes," *in Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pp. 1162–1163, January 2003.

[129] S. Tarnick, "Single-output embedded checkers for systematic unordered codes," *in Proceedings of 10th IEEE International On-Line Testing Symposium (IOLTS)*, pp. 45–51, January 2004.

[130] J. M. Borden, "Optimal asymmetric error detecting codes," *Information and Control*, vol. 53, no. 1–2, pp. 66–73, 1982.

[131] N. Jha, "A totally self-checking checker for Borden's code," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, pp. 731–736, January 1989.

[132] T. Haniotakis, A. Paschalis, and D. Nikolos, "Efficient totally self-checking checkers for a class of Borden codes," *IEEE Transactions on Computers*, vol. 44, pp. 1318–1322, January 1995.

[133] G. Biswas and I. Sengupta, "A design technique of TSC checker for Borden's code," *in Proceedings of Tenth International Conference on VLSI Design*, pp. 529–530, January 1997.

[134] S. Tarnick, "Embedded Borden 2-UED code checkers," *in Proceedings of 12th IEEE International Symposium on On-Line Testing (IOLTS)*, pp. 173–175, January 2006.

[135] R. Hamming, "Error detecting and error correcting codes," *Bell System Technical Journal*, vol. 26, April 1950.

[136] A. Saleh, J. Serrano, and J. Patel, "Reliability of scrubbing recovery-techniques for memory systems," *IEEE Transactions on Reliability*, vol. 39, pp. 114–122, January 1990.

[137] G. Neuberger, F. Lima, L. Carro, and R. Reis, "A multiple bit upset tolerant SRAM memory," *ACM Transactions on Design Automation of Electronic Systems (TODAES*, vol. 8, October 2003.

[138] C. Argyrides, H. Zarandi, and D. Pradhan, "Multiple upsets tolerance in SRAM memory," *in Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 365–368, May 2007.

[139] F. Maison, "The MECRA: A self-reconfigurable computer for highly reliable process," *IEEE Transactions on Computers*, vol. C-20, pp. 1382–1388, January 1971.

[140] E. Prange, "Cyclic error-correcting codes in two symbols," tech. rep., AFCRC-TN-57-103, Air Force Cambridge Research Center, 1957.

[141] W. W. Peterson and D. T. Brown, "Cyclic codes for error detection," *Proceedings of the IRE*, pp. 228–235, January 1961.

[142] L.-H. Zetterberg, "Cyclic codes from irreducible polynomials for correction of multiple errors," *IRE Transactions on Information Theory*, vol. 8, pp. 13–20, January 1962.

[143] S. Dodunekov and J. Nilsson, "Algebraic decoding of the Zetterberg codes," *IEEE Transactions on Information Theory*, vol. 38, pp. 1570–1573, January 1992.

[144] M.-H. Jing, Y. Chang, C.-D. Lee, J.-H. Chen, and Z.-H. Chen, "A result on Zetterberg codes," *IEEE Communications Letters*, vol. 14, pp. 662–663, January 2010.

[145] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *SIAM Journal on Applied Mathematics*, vol. 8, pp. 300–304, January 1960.

[146] G. Cardarilli, M. Ottavi, S. Pontarelli, M. Re, and A. Salsano, "Fault tolerant solid state mass memory for space applications," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 41, pp. 1353–1372, January 2005.

[147] A. Thompson, "Evolving electronic robot controller that exploit hardware resources," *in Proceedings of the Third European Conference on Advances in Artificial Life*, pp. 640–656, 1995.

[148] G. W. Greenwood and A. M. Tyrrell, *Introduction to evolvable hardware: A Practical Guide for Designing Self-Adaptive Systems*. Wiley-IEEE Press, 1 ed., November 2006.

[149] A. Thompson, "Evolving fault tolerant systems," *in Proceedings of First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA)*, pp. 524–529, January 1995.

[150] D. Keymeulen, R. Zebulum, Y. Jin, and A. Stoica, "Fault-tolerant evolvable hardware using field-programmable transistor arrays," *IEEE Transactions on Reliability*, vol. 49, pp. 305–316, January 2000.

[151] J. Miller and M. Hartmann, "Evolving messy gates for fault tolerance: some preliminary findings," *in Proceedings of The Third NASA/DoD Workshop on Evolvable Hardware*, pp. 116–123, January 2001.

[152] R. Canham and A. Tyrrell, "Evolved fault tolerance in evolvable hardware," *in Proceedings of Congress on Evolutionary Computation (CEC)*, vol. 2, pp. 1267–1271, January 2002.

[153] G. V. Larchev and J. D. Lohn, "Evolutionary based techniques for fault tolerant field programmable gate arrays," *in Proceedings of International Conference on Space Mission Challenges for Information Technology*, pp. 1–8, June 2006.

[154] T. Schnier and X. Yao, "Using negative correlation to evolve fault-tolerant circuits," *in Proceedings of 5th International Conference on Evolvable Systems: From Biology to Hardware*, pp. 35–46, January 2003.

[155] G. Greenwood and M. Joshi, "Evolving fault tolerant digital circuitry: Comparing population-based and correlation-based methods," *in Proceedings of IEEE Congress on Evolutionary Computation (CEC)*, pp. 2796–2801, January 2009.

[156] C. Maxfield, *The Design Warrior's Guide to FPGAs: Devices, Tools and Flows*. Newnes, May 2004.

[157] E. Hamdy, J. McCollum, S. Chen, and S. Chiang, "Dielectric based antifuse for logic and memory ICs," *IEEE International Electron Devices Meeting (IEDM)*, December 1988.

[158] N. G. Jacobson, *The In-System Configuration Handbook: A Designer's Guide to ISC*. Springer, 1 ed., November 2003.

[159] J. Wang, B. Cronquist, J. McCollum, F. Hawley, D. Yu, R. Chan, R. Katz, and I. Kleyner, "Total dose and SEE of metal-to-metal antifuse FPGA," *in Proceedings of 2nd Conference on Military and Aerospace Applications of Programmable Devices and Technologies*, January 1999.

[160] Quicklogic, "Quicklogic reliability report," pp. 1–21, November 1998.

[161] K. Gordon and R. Wong, "Conducting filament of the programmed metal electrode amorphous silicon antifuse," *IEEE International Electron Devices Meeting (IEDM)*, pp. 27–30, 1993.

[162] Quicklogic, "Security in quicklogic devices," *QuickLogic White Paper*, pp. 1–10, July 2002.

[163] B. Matas and C. D. Subercasaux, *Memory 1997: Complete coverage of DRAM, SRAM, EPROM, and flash memory ICs*. Integrated Circuit Engineering Corp., 1st ed., January 1997.

[164] Microsemi, "ProASIC3 fpga fabric user's guide," 2011.

[165] Microsemi, "ProASIC3 flash family FPGAs datasheet," April 2012.

[166] Altera Corporation, "Stratix V device handbook," March 2012.

[167] Altera Corporation, "Arria V device handbook," March 2012.

[168] Altera Corporation, "Cyclone V device handbook, volume 1: Device overview and datasheet," December 2011.

[169] Altera Corporation, "Hardcopy IV device handbook," April 2012.

[170] Xilinx, "Xilinx DS150 virtex-6 family overview," March 2009.

[171] C. Souza, "IP columns support app-specific FPGAs." available: http://eetimes.com/electronics-news/4046520/IP-columns-support-app-specific-FPGAs, 2003. [Online; accessed 16-July-2012].

[172] P. Alfke, "Xilinx spartan-6 FPGA user guide lite." available: http://www.eetimes.com/design/programmable-logic/4015237/Xilinx-Spartan-6-FPGA-User-Guide-Lite, 2009. [Online; accessed 19-July-2012].

[173] Xilinx, "Spartan-6 FPGA configurable logic block," February 2010.

[174] S. Bapat, "Easypath-6 technology: Fast, simple, risk-free FPGA cost reduction," tech. rep., Xilinx, November 2009.

[175] Actel, "IGLOO® handbook," December 2008.

[176] Actel, "Fusion® handbook," December 2008.

[177] Actel, "ProASIC®3 handbook," December 2008.

[178] Actel, "Axcelerator® family FPGAs," October 2009.

[179] Actel, "40MX and 42MX FPGA families," April 2009.

[180] Lattice, "LatticeECP3 family handbook," September 2009.