UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação

Dissertação

A Fast Approximate Function Generation Technique to ATMR Design

Guilherme Barbosa Manske

Pelotas, 2022

Guilherme Barbosa Manske

A Fast Approximate Function Generation Technique to ATMR Design

Dissertação apresentada ao Programa de Pós-Graduação em Computação do Centro de Desenvolvimento Tecnológico da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Advisor: Prof. Dr. Leomar Soares da Rosa Júnior
Coadvisor: Prof. Dr. Paulo Francisco Butzen

Pelotas, 2022

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação na Publicação

M286f Manske, Guilherme Barbosa

A fast approximate function generation technique to ATMR design / Guilherme Barbosa Manske ; Leomar Soares da Rosa Junior, orientador ; Paulo Francisco Butzen, coorientador. — Pelotas, 2022.

58 f. : il.

Dissertação (Mestrado) — Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2022.

1. TMR. 2. Confiabilidade. 3. Computação aproximada. 4. Correlação. 5. ATMR. I. Rosa Junior, Leomar Soares da, orient. II. Butzen, Paulo Francisco, coorient. III. Título.

CDD : 005

Elaborada por Aline Herbstrith Batista CRB: 10/1737

Guilherme Barbosa Manske

A Fast Approximate Function Generation Technique to ATMR Design

Dissertação aprovada, como requisito parcial, para obtenção do grau de Mestre em Ciência da Computação, Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas.

Data da Defesa: 21 de junho de 2022

ABSTRACT

MANSKE, Guilherme Barbosa. **A Fast Approximate Function Generation Technique to ATMR Design**. Advisor: Leomar Soares da Rosa Júnior. 2022. 58 f. Thesis (Masters in Computer Science) – Technology Development Center, Federal University of Pelotas, Pelotas, 2022.

Technological advances provide the production of transistors on a nanometric scale, allowing the production of circuits with a higher transistor density and an increase in the circuit's operating frequency. However, this transistor miniaturization makes circuits more susceptible to faults. Single events transient can be generated by the collision of energetic particles (such as alpha particles, neutrons, or other heavy ions) with a node in the circuit. These transient events can cause a fault that becomes an error when propagated to a memory element or output. Triple Modular Redundancy is used to mask these faults by generating two copies of the original circuit and connecting the three modules with a majority voter. This strategy can mask 100% of faults that occur in one module. However, it has a high area cost, presenting more than 200% area increase with the modules and the majority voter. Approximate computing concept is used to create modules for the Triple Modular Redundancy architecture to mitigate this area increase. However, the area cost reduction happens in exchange for increasing the fault coverage rate. Therefore, optimizing the tradeoff between area increase and the fault coverage rate is important and challenging. Another critical metric to be optimized is the computational cost to generate these modules, although many methods proposed in the literature do not focus on this metric. The proposed method aims to optimize the computational cost while maintaining a good tradeoff between area increase and fault coverage. The method analyzes the correlation of each input variable with the output, using the variables that have the highest correlation with the output to approximate the modules. The area of the mapped circuit and the error rate of the circuit created using the approximate modules are extracted. The generated circuits have an area increase down to 86.02% and an error rate down to 3.61%. The generation of these functions has a low computational cost, with the largest circuits being approximated in 214 ms. The area and error rate values are worse than those obtained by other methods, but some of these methods take several hours to approximate the same functions used in our analysis.

Keywords: TMR. Reliability. Approximate Computing. Correlation.

RESUMO

MANSKE, Guilherme Barbosa. **Geração Rápida de Funções aproximadas dedicadas ao projeto de arquiteturas ATMR**. Orientador: Leomar Soares da Rosa Júnior. 2022. 58 f. Dissertação (Mestrado em Ciência da Computação) – Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2022.

O avanço tecnológico proporciona a produção de transistores em escala nanométrica, possibilitando a construção de circuitos com uma maior densidade de transistores e que funcionem com uma maior frequência de operação. Entretanto, essa miniaturização dos transistores faz com que os circuitos se tornem mais suscetíveis a falhas. Redundância Modular Tripla é uma técnica utilizada para mascarar essas falhas, gerando duas cópias do circuito original e ligando os três módulos em um votador majoritário. Essa técnica é capaz de mascarar 100% das falhas únicas que ocorram nos módulos, porém ela tem um custo elevado em termos de área, apresentando mais de 200% de aumento considerando os módulos e o votador. Para combater esse aumento de área, técnicas de computação aproximada podem ser utilizadas para criar os módulos para a técnica da Redundância Modular Tripla. Entretanto, a redução do custo de área acontece em troca de um aumento na taxa de cobertura de falhas. Portanto, é importante e desafiador otimizar o *tradeoff* entre aumento de área e taxa de cobertura de falhas. Outra medida importante a ser otimizada é o custo computacional para produzir esses módulos, já que muitos métodos propostos na literatura não possuem esta preocupação. O método proposto nesta dissertação visa otimizar o custo computacional, mantendo um bom *tradeoff* entre o aumento de área e a cobertura de falhas. O método analisa a correlação de cada variável de entrada com a saída, utilizando as variáveis que possuem maior correlação com a saída para aproximar os módulos. São extraídas a área do circuito mapeado e a taxa de erros do circuito criado utilizando os módulos aproximados. Os circuitos gerados apresentam um aumento de área de pelo menos 86,02% e uma taxa de falhas de pelo menos 3,61%. A geração dessas funções tem um baixo custo computacional, com os maiores circuitos sendo aproximados em 214 ms. Esses valores são inferiores aos obtidos por outros métodos disponíveis na literatura. Contudo, alguns desses outros métodos demandam várias horas para aproximar as mesmas funções utilizadas na nossa análise.

Palavras-chave: TMR. Confiabilidade. Computação Aproximada. Correlação.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

SET	Single Event Transient
TMR	Triple Modular Redundancy
ATMR	Approximate Triple Modular Redundancy
CED	Concurrent Error Detection
NMR	N-Modular redundancy
QAMR	Quadruple Approximate Modular Redundancy
PI	Prime Implicant
MEROP	Minterm/Maxterm to Expand/Reduce Original Prime Implicant
KM	Karnaugh Map
ATPG	Automatic Test Pattern Generation
GA	Genetic Algorithm
MOO	Multi-Objective Optimization
MOOGA	Multi-Objective Optimization Genetic Algorithm

CONTENTS

1 INTRODUCTION

Electronic devices are part of modern lives, and they can be found everywhere. Nowadays, most people have a smartphone that contains chips with billions of transistors in their pockets. Also, most modern home appliances have some electronic circuit that controls their functions. This dissemination is causing an increase in the demand for electronic circuits, and this increase will probably continue with autonomous cars, the Internet of Things, space missions, among others.

Transistor miniaturization (Fig. 1) is happening fast with the advances in technologies used in foundries. These advances increase the circuit's transistor density and operating frequency, producing more complex circuits. Today CPUs have up to 114 billion 5 nm transistors in a single chip (APPLE, 2022).

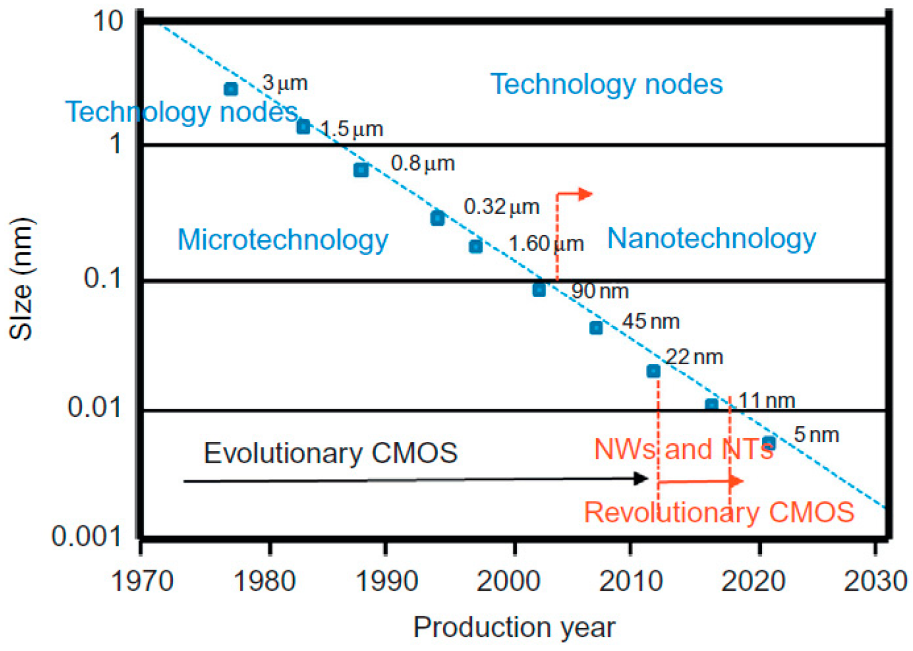


Figure 1 – Transistor size reduction over the years. Source: (RADAMSON et al., 2019)

Additionally, the transistor miniaturization makes circuits more susceptible to faults. Single event transients (SET) are the disturbances that can be generated by the collision of energetic particles, such as alpha particles, neutrons, or other heavy ions, with a node in the circuit. The disturbances can flip the gate output to the opposite value that it is supposed to be. These SETs can become errors if propagated to a memory element or an output. Once it becomes an error stored in the memory, it becomes harder, sometimes impossible, to detect and correct the error.

It is crucial to design circuits that have a low susceptibility to faults. There are multiple strategies to mask these faults, preventing them from becoming errors (MUNTEANU; AUTRAN, 2008). Electrical masking occurs when a SET is faster than the cutoff frequency of the circuit, this SET will then be attenuated. Temporal masking occurs when a SET happens in the combinational circuit, but the timing does not match with the latching window of the sequential circuit. Logical masking occurs when a SET happens in the combinational circuit, but it does not change the output of subsequent gates because its other inputs determine it.

The Triple Modular Redundancy (TMR) technique is a well-known method to mask these faults. This technique consists in creating two extra modules equal to the original and connecting them with a majority voter. This technique presents a 100% fault coverage rate when any SET occurs in one of the modules. The fault is logically masked in the majority voter that chooses the correct value from the other two modules. It is important to notice that this technique cannot mask a SET happening in the voter. Therefore, this technique is useful for critical circuits where any fault can cause a severe problem.

The TMR has a 100% fault coverage rate, but it comes with a high area overhead. The area overhead of the TMR technique is bigger than 200%. This area overhead can be a big problem for some projects that cannot use all this extra area. Also, a larger circuit has a higher power consumption, which usually is a metric that should be as low as possible. Lastly, the voter also adds a short delay in the circuit because the signal needs to go through the circuit and go through the voter afterward.

Some applications present a resilience to errors and can tolerate them. These applications do not need the output to be 100% correct and can use techniques to reduce their computational cost with a penalty in the accuracy. An example of these applications is image/video processing, where an image does not need to be perfect to be understood. Fig. 2 shows an example of how an increase in the number of errors. Fig. 2(a) is the image without any errors, and in Fig. 2(b) some error is added, and in Fig. 2(c) more error is added. However, it does not make it impossible to interpret what is in the image.

The error resiliency can be explored to wisely protect circuits, selective hardening the most crucial parts of the circuit or the outputs that cannot tolerate errors. Selective hardening makes it possible to reduce the costs of the circuit while it operates acceptably. There are two reasons to select a part of a circuit to be applied selective hardening. This part can affect important outputs that do not tolerate errors, so they need extra protection. Also, the part can be hardened because it presents a high fault

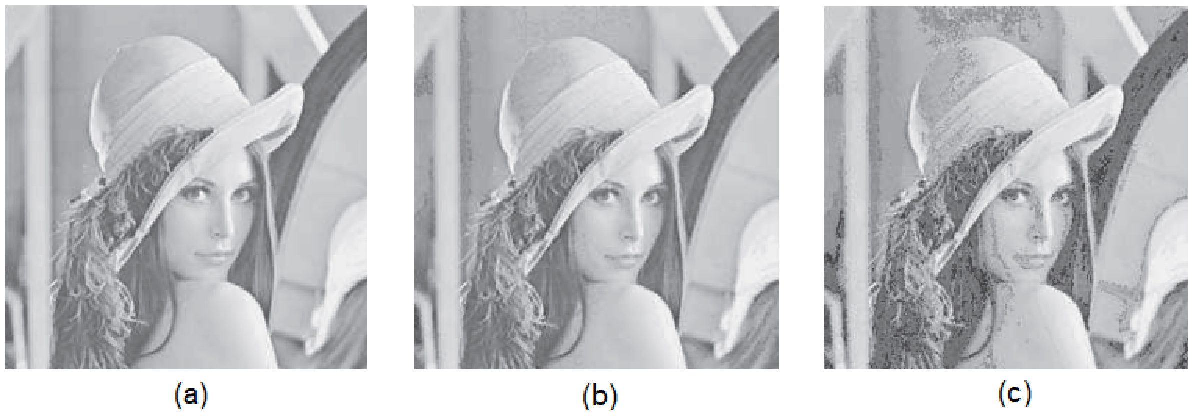


Figure 2 – Error resiliency shown in image processing. (a) has no errors, (b) adds some error, and (c) adds more error. Source: (ICHIHARA et al., 2015)

susceptibility, making selective hardening methods more effective.

The approximate computing concept is relevant for several applications resilient to approximations. Approximate computing has been used to make more efficient, simpler, and faster circuits (BOSIO et al., 2020). It generates less complex systems, reducing the output accuracy in a controlled manner while using fewer resources. It is crucial to analyze if the application, where the approximate computing technique will be applied, tolerates the inaccuracy added by this technique.

The approximate computing concept was explored to reduce the area overhead needed by the TMR technique in (GOMES et al., 2015a; HASSAN et al., 2018; ALBANDES et al., 2019). Approximate TMR (ATMR) uses approximate functions to create the TMR modules. For a set of approximate functions, it is necessary that, for each input, at least two of the three modules of the TMR architecture present the output equal to the output of the original function.

1.1 Motivation

The use of approximate computing to create the modules for the ATMR tends to reduce the area overhead compared with the TMR design. The area reduction achieved in ATMR, when compared to traditional TMR, occurs at the cost of a fault coverage rate reduction (GOMES et al., 2015a). The reduction in the fault coverage rate happens because the approximate functions are not entirely the same as the original function. Therefore, generating approximate functions that present a good tradeoff between fault coverage and area cost is very important and challenging. It is even harder to generate these functions with a low computational cost.

It is challenging to generate good approximate functions to be used in the ATMR. The ATMR design needs a good tradeoff between area overhead and fault coverage rate. This generation needs to meet different constraints depending on the project.

The project can limit the area overhead not to be higher than a certain value or limit the acceptable fault coverage rate. These limitations can happen in the same project. The generation of the approximate modules has to be thought carefully because the voter needs to be able to give the correct output in the absence of faults. The two approximate modules cannot differ from the original function for any input vector at the same time for the voter to give the correct output when no fault happens.

The computational cost required to generate the approximate functions cannot be left aside. This cost tends to be elevated because there are multiple possibilities to generate these approximation functions. Some proposed techniques rely on an exhaustive method to generate the approximate functions, limiting the size of the circuits that can be approximated, because the computational cost increases very fast with the circuit size. The authors usually continue their research by proposing heuristic methods to improve the computational cost of the method previously proposed while keeping similar results in area overhead and error rate (ER) tradeoff.

1.2 Objectives

The computational cost to generate the modules to the ATMR is usually left aside, with area overhead and ER being the main focus of analysis and improvements. Previous methods can take hours to create the approximate functions for circuits with no more than 14 inputs (ALBANDES et al., 2019). In (ARIFEEN et al., 2018), circuits with up to 117 inputs were used to generate ATMR designs, but there is no data about the computational cost available. Also, the area overhead and ER tradeoff cannot be ignored, so the method needs to return competitive values for these metrics. With this in mind, the main objectives of this work are:

- propose a fast method to generate approximate functions for ATMR designs.
- keep a good area overhead and ER tradeoff.
- prove the scalability of the proposed method.

A good literature review is crucial to understand the existing methods and their strong and weak points. This information determines if the proposed method is something needed and new in the literature or to think about a new method if necessary. Useful concepts are learned throughout this process and they can be used to develop this new method. Therefore, some secondary objectives of this work are:

- Learn about reliability.
- Research about the state of the art techniques to generate ATMR designs.
- Research about other redundant methodologies.

1.3 Organization

The rest of the dissertation is organized as follows. Chapter 2 provides background information that is important to understand the proposed method. Chapter 3 presents and analyses other works in the literature. Chapter 4 explains the concept used to generate the approximate functions and how it was implemented. Chapter 5 presents and discusses the results obtained using the proposed method. Finally, the conclusions and future directions of this work are presented in Chapter 6.

2 BACKGROUND

This chapter presents important concepts used as a base to develop our methodology. First, an introduction about approximate computing, explaining what it is, why use it and where it is used. Next, some fault-masking techniques will be presented and explained. After, the TMR technique will be explained, presenting its pros and cons. Also, it will be explained how to approximate computing techniques can be used to improve the design of circuits in the TMR architecture. Finally, some aspects that need attention when creating the approximate modules will be addressed.

2.1 Approximate Computing/Circuits

The approximate Computing paradigm aims to make more efficient, simpler, and faster circuits/systems (BOSIO et al., 2020). It generates less complex systems, reducing the output accuracy in a controlled manner while using fewer resources. It can be used for applications that can accept inaccuracies, reducing accuracy while reducing the area and power consumption. Also, it can reduce the area cost of redundancy reliable fault-tolerant techniques while keeping a low ER.

There are multiple Approximate Computing techniques, and they can be classified into two types. The first type work at the hardware level, with less accurate and high efficient energy components. The second type is software-level techniques that reduce calculations or accesses to memory to improve performance at the expense of precision in the results (APONTE-MORENO et al., 2018).

Approximate Computing can be used in multiple applications that can accept some level of inaccuracy. For example, it can be used in applications such as image processing, pattern recognition, and data mining (APONTE-MORENO et al., 2018). Also, there are multiple techniques for redundancy in HW/SW architectures. Concurrent Error Detection (CED) , depicted in Fig. 3, uses one approximate module and checkers to detect faults happening in the circuit (CHOUDHURY; MOHANRAM, 2008). N-Modular Redundancy (NMR) , illustrated in Fig. 4, uses approximate software to reduce the computational cost (BAHARVAND; MIREMADI, 2017). ATMR uses the original module plus two approximate modules and a voter to protect the circuit (GOMES; KASTENSMIDT, 2013).

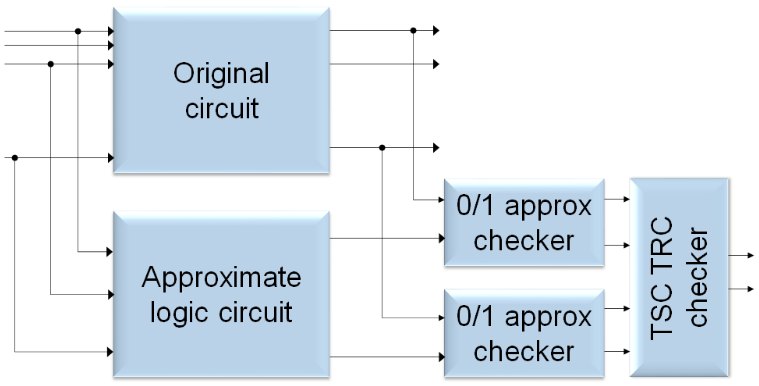
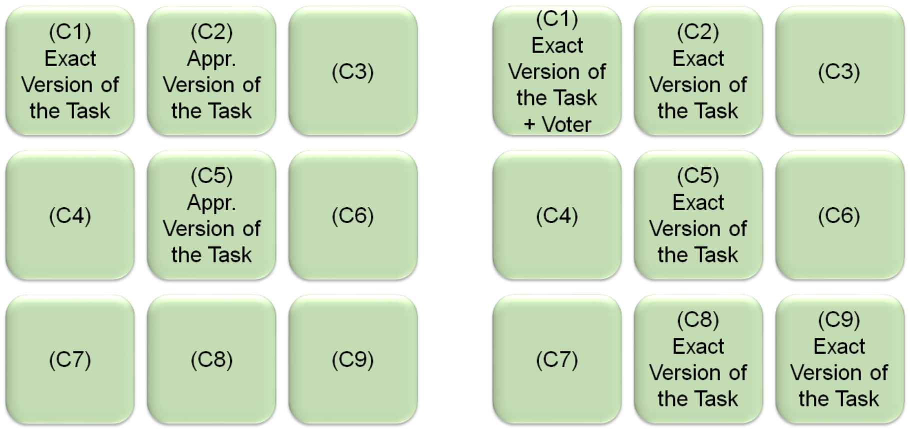


Figure 3 – Concurrent Error Detection


(C1) Exact Version of the Task
(C2) Appr. Version of the Task
(C3)
(C4)
(C5) Appr. Version of the Task
(C6)
(C7)
(C8)
(C9)
(C1) Exact Version of the Task + Voter
(C2) Exact Version of the Task
(C3)
(C4)
(C5) Exact Version of the Task
(C6)
(C7)
(C8) Exact Version of the Task
(C9) Exact Version of the Task
Figure 4 – N-Modular Redundancy

Strong approximate functions differ from the original for a small number of inputs. It means they have just a few unprotected vectors or a small Hamming distance from the original function. The weak approximate functions agree with the original function for a small number of inputs. It means that they have most of their vectors protected or a high Hamming distance from the original function.

2.2 Fault Masking

The increase in the fault susceptibility in modern circuits makes masking techniques crucial to the correct operation of a circuit. Masking fault techniques are important, so the circuit can still work even when a fault happens, preventing the fault becomes an error. There are multiple masking techniques, and some of them will be explained in this subchapter.

Temporal masking, or latching window masking, occurs when a SET happens in a combinational circuit but is not captured in the latching window of the sequential circuit, so it does not become an error. The latching window is a period of time where the sequential circuit reads its inputs. Fig. 5 presents a strategy to use a sequential circuit to mask some faults that do not happen in the latching window. It masks every fault that happens before or after the latching window. In Fig. 5, the fault happens after the latching window and therefore does not become an error.

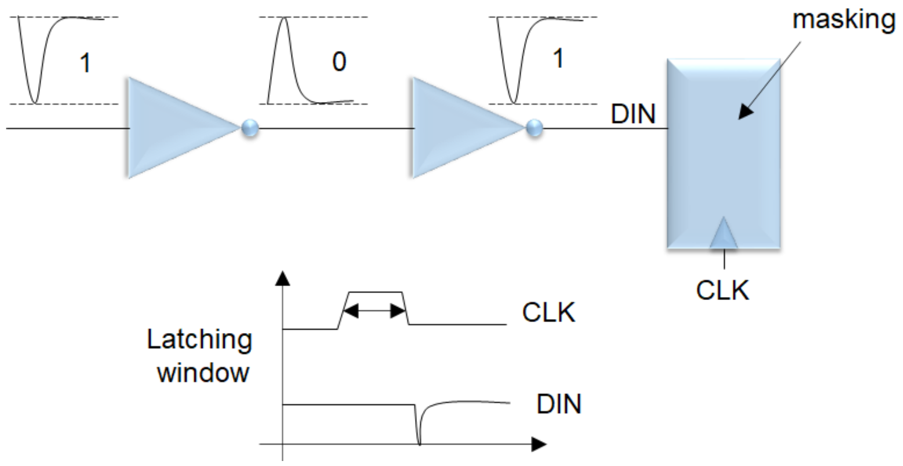


Figure 5 – Window masking technique. The fault arrives in the sequential circuit after the latching window and is masked.

Electrical masking occurs when a SET presents bandwidth higher than the cutoff frequency of the CMOS circuit, this SET will then be attenuated (MUNTEANU; AUTRAN, 2008). The pulse amplitude may reduce, the rise and fall times increase, and, eventually, the pulse may disappear, as shown in Fig. 6. On the other hand, since most logic gates are nonlinear circuits with a substantial voltage gain, low-frequency pulses with sufficient initial amplitude will be amplified, generating an error.

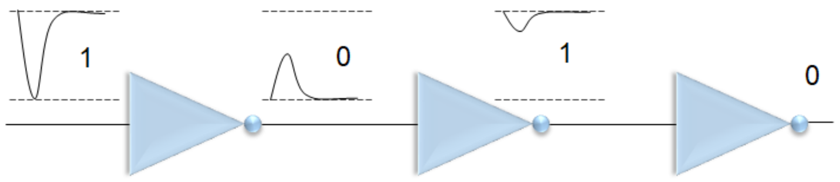


Figure 6 – Electrical masking technique. The logic gates of the circuit attenuate the SET.

Logical masking occurs when a particle strikes a portion of the combinational logic that cannot affect the output due to a subsequent gate whose result is entirely determined by its other input values (MUNTEANU; AUTRAN, 2008). For example, in Fig. 7(a), if a SET happens and it is propagated until the input of a NOR gate, but the other input is in the controlling state ('1'), then the SET will be masked and will not cause an error. The same thing happens in Fig. 7(b) using a controlling state ('0') in the NAND gate.

The miniaturization of the transistors causes a reduction in temporal and electrical masking techniques capacities. Shorter SETs can be latched using the temporal technique because the latching window opens more often in faster circuits. Also, lower

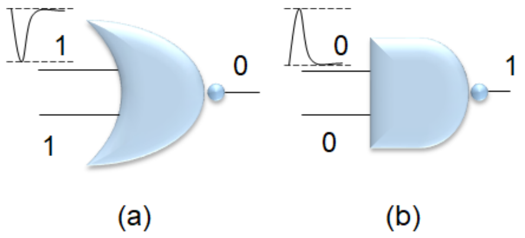


Figure 7 – Logical masking technique. The value in the other input masks the SET happening in the first input. In (a), the second input has the controlling state '1', and in (b), the second input has the controlling state '0'.

amplitude SETs can become an error because the circuits are working with lower voltages, making it harder to use the electrical masking technique. These characteristics make logical masking the main focus of improving the fault tolerance of circuits.

2.3 TMR / ATMR

The TMR is a well-known logical masking technique, and it is used to protect the circuit against SETs (VON NEUMANN, 1956). It uses three copies of the original circuit and a majority voter to compute the correct output. This technique presents a 100% fault coverage rate against SETs happening in the modules. It is important to notice that a fault happening in the voter will not be masked, so it needs to be protected using other techniques. The TMR technique is essential for critical circuits where any fault can cause a severe problem. Fig. 8 shows the TMR design using the original circuit and two copies connected to a voter to protect the circuit.

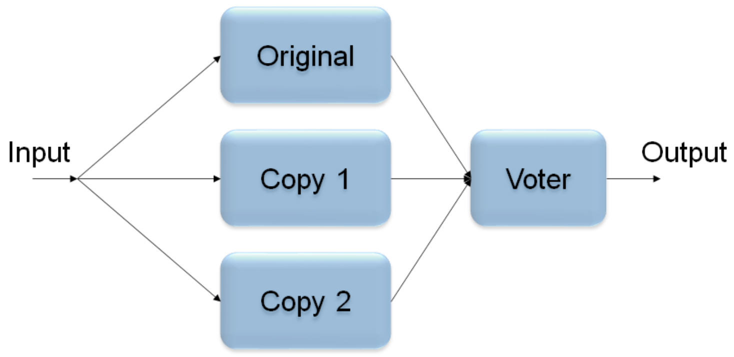


Figure 8 – TMR design with the original circuit and two copies connected to a voter

Fig. 9 shows an example of how this technique masks a SET happening in one of the modules. In Fig. 9(a), the circuit is operating with no faults. The inputs and outputs of all the modules are the same, and the voter has the correct output. In Fig. 9(b), a SET happens in the last module. The inputs are the same for all modules, but the original circuit and the copy 1 module have output '0', and the copy 2 module has output '1'. The majority voter receives two '0's and one '1' as input and returns the correct output '0'. SETs happening in the other modules would be masked in the same way.

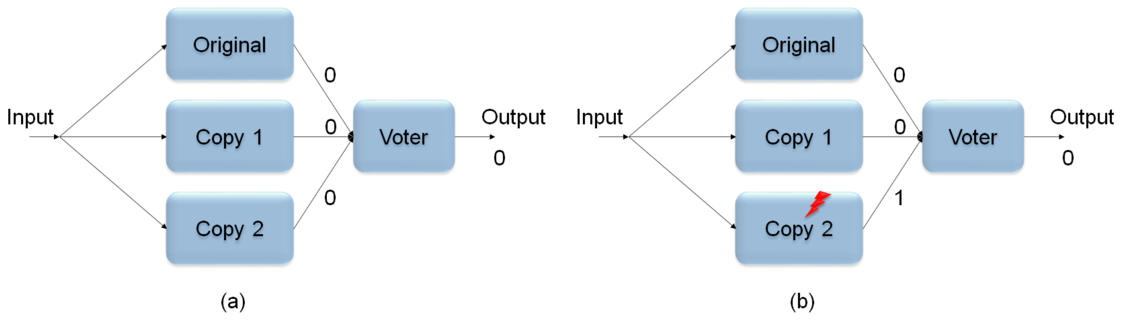


Figure 9 – Example of how the TMR technique masks a fault. (a) shows the design without a fault, and (b) shows the design when a fault happens in the copy 2 module.

The TMR is an easy technique to apply in any circuit, and it makes the circuit more reliable. However, the TMR design has more than 200% of area overhead, 200% in the modules, and an extra area used by the voter. This can be a big problem if the project constrains the area of the circuit, power consumption, or the delay that increases because of the voter. Therefore, it is crucial to look for ways to decrease these metrics while making the circuit as reliable as possible.

To reduce the area overhead, the approximate TMR (ATMR) design was proposed (GOMES; KASTENSMIDT, 2013) and its design is shown in Fig. 10. In the ATMR, approximate computing concepts are used to create the modules for the TMR architecture. It keeps one original circuit and uses two different approximate modules. This technique reduces the area overhead, but it also reduces the fault coverage rate of the design. Therefore, it is important and challenging to generate approximate functions that have a good tradeoff between area overhead and fault coverage rate. It is even more challenging to generate these functions with a low computational cost.

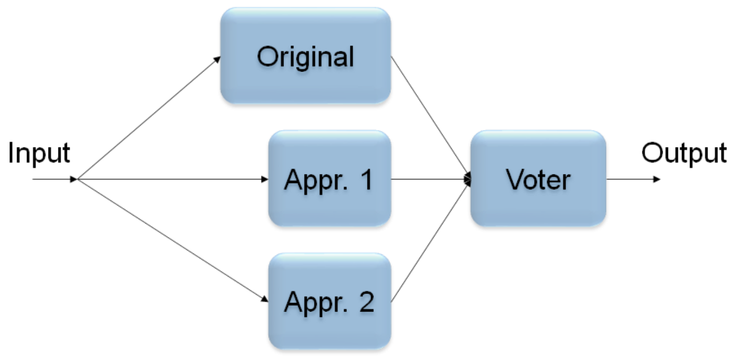


Figure 10 – ATMR design withe the original circuit and two approximate modules connected to a voter.

The Fig. 11 shows an example of how the ATMR technique masks a fault happening in one of the modules. In Fig. 11(a), the circuit is operating with no faults. The inputs are the same for all modules, the original circuit and the appr. 1 module have output ’0’ and the appr. 2 module has output ’1’. The majority voter receives two ’0’s and one '1' as input and returns the correct output '0'. In Fig. 11(b), a fault happens in the appr. 2 module. The inputs and outputs of all the modules are the same and the voter has the correct output. The majority voter receives three '0's as input and returns the correct output '0'. In Fig. 11(c), a fault happens in the appr. 1 module. The inputs are the same for all modules, the original circuit has output '0' and the appr. 1 module and the appr. 2 module have output '1'. The majority voter receives one '0' and two '1's as input, and it ends up returning the incorrect output '1'. In Fig. 11(d), a fault happens in the original circuit. The inputs are the same for all modules, the original circuit and the appr. 2 module have output '1' and the appr. 1 has the output '0'. The majority voter receives one '0' and two '1's as input, and it ends up returning the incorrect output '1'.

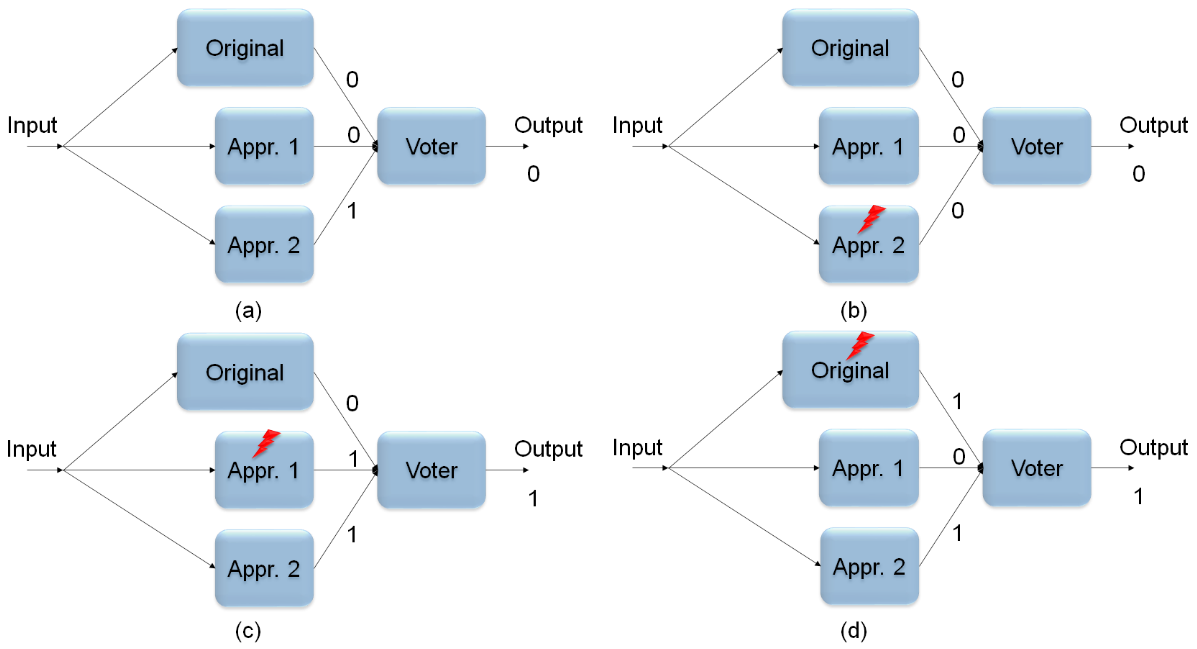


Figure 11 – Example of how the ATMR technique masks a fault. (a) shows the design without a fault, (b) shows the design when a fault happens in the appr. 2 module, (c) shows the design when a fault happens in the appr. 1 module, and (d) shows the design when a fault happens in the original module.

The challenge to generate the approximate functions is that all the input vectors need to still protected. To make the ATMR design functional, both modules cannot unprotect the same minterm, because it would cause the voter to give the wrong output even in the absence of faults. A well-known methodology to avoid both modules to unprotect the same minterm is to create an over-approximate module (H), or 0-approximation module, and an under-approximate module (F), or 1-approximation module. The original function is called G. The over-approximate module (H) protects all the outputs that used to be '1'. The under-approximate module (F) protects all the outputs that used to be '0'. Fig. 12 shows how the set of minterms of the three functions are related in an ATMR design.

The function G=(A*!C)+(!A*C)+(!A*!B*D)+(!A*B*!D)+(A*B*D)+(A*!B*!D) and some

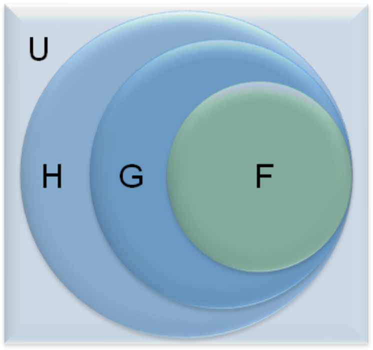


Figure 12 – Graphical relation between the minterms of the FGH functions. G is the original function, F is the under-approximate function, and H is the over-approximate function. Adapted from: (GOMES et al., 2015a)

approximate functions will be used to exemplify this methodology in Table 1. The first 4 columns are the inputs. Next, the column G is the original function. The next 4 columns are under-approximate functions and the last 4 are over-approximate functions. The last row indicates how many literals each function has. The red cells indicate which input vector was changed to create the approximate function. Any combination of an under and an over-approximate function with the original function will work properly in an ATMR design in the absence of faults.

Table 1 – Under and over-approximate functions to be used in ATMR and Full-ATMR designs.

Function					Under				Over			
A	B	C	D	G	F1	F2	F3	F4	H1	H2	H3	H4
0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	1	1	0	1	1	1	1	1	1	1
0	0	1	0	1	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1	1	1	1	1
0	1	0	0	1	1	0	1	1	1	1	1	1
0	1	0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	1	1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	1	1	1	1	1
1	0	1	0	1	1	1	1	0	1	1	1	1
1	0	1	1	0	0	0	0	0	0	0	1	0
1	1	0	0	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1
1	1	1	0	0	0	0	0	0	0	0	0	1
1	1	1	1	1	1	1	0	1	1	1	1	1
Lit.Count				16	13	13	13	13	14	14	14	14

The ATMR technique uses two approximate modules and the original module. The Full-ATMR is proposed in (GOMES et al., 2015a). The Full-ATMR approximates the original module too, creating a design with three approximate modules. This technique tends to reduce the area further than the ATMR design, but it is a big challenge to generate three good functions that can make the Full-ATMR design work. Only one module can differ from the original function for each input vector, so the design works properly in the absence of faults. Once three good approximate functions are found, the masking analysis can be done the same way that it was done with the ATMR design.

The strategy to generate over and under-approximate functions can be applied for the Full-ATMR too. The Full-ATMR can be formed using three functions of the same type (F/H). In Table 1, the functions F1,F2 and F3 can be used for this situation. Fig. 13(a) shows how the set of minterms of the three functions are related in this situation. Next, the Full-ATMR can be formed using two H functions and one F function. Using the functions in Table 1, the functions F1,H1 and H2 can be used for this situation. Fig. 13(b) presents how the set of minterms of the three functions are related in this situation. Lastly, the Full-ATMR can be formed using one H function and two F functions. In Table 1, the functions F1,F2 and H1 can be used for this situation. Fig. 13(c) presents how the set of minterms of the three functions are related in this situation.

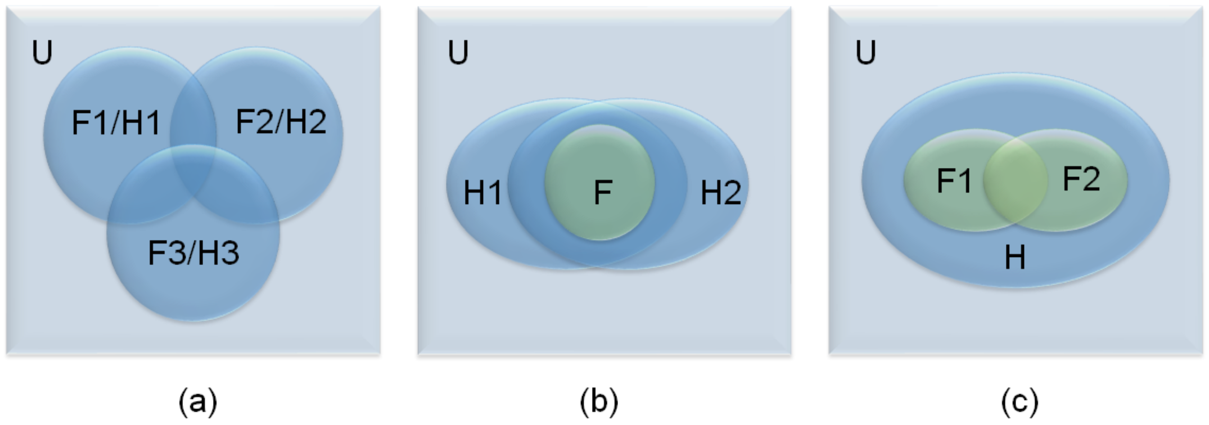


Figure 13 – There are three configurations to use under/over-approximate functions in Full-ATMR. (a) uses 3 under or 3 over-approximate functions. (b) uses 2 over and 1 under-approximate functions. (c) uses 1 over and 2 under-approximate functions. Adapted from: (GOMES et al., 2015a)

Some projects may not tolerate the reduction in reliability created by the ATMR and Full-ATMR designs, but the TMR design is still too expensive in the area cost. There is a new technique called Quadruple Approximate Modular Redundancy (QAMR) proposed in (DEVEAUTOUR et al., 2020). This technique uses four approximate modules and connects the outputs of these modules to 4 majority voters, but each module can be connected to only three voters. This method can get to 100% fault coverage and have a lower area overhead than the TMR. Fig. 14 shows the QAMR architecture.

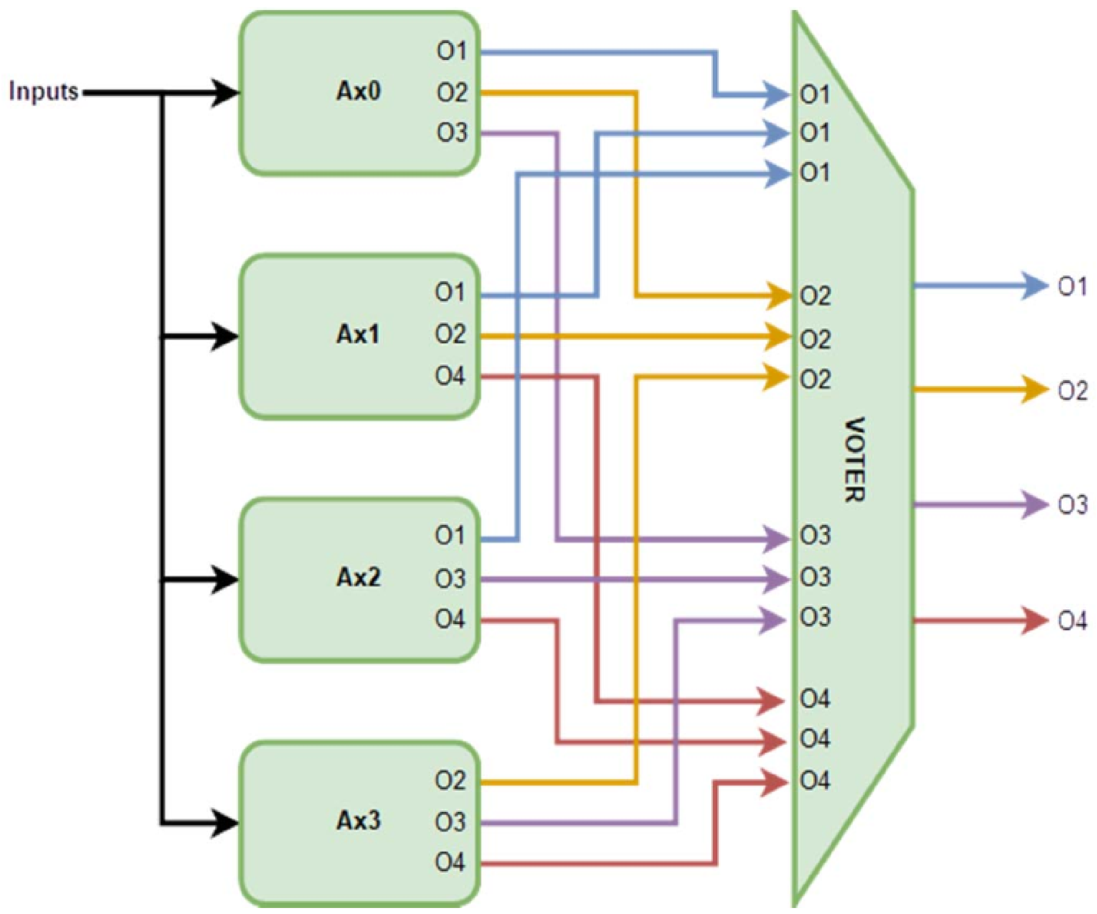


Figure 14 – QAMR layout. Source: (DEVEAUTOUR et al., 2020)

2.4 Considerations for building an ATMR

Applications - The ATMR design can be used in specific applications, where the circuit can tolerate a determined ER. An example would be in parts of a rover that would explore an inhospitable place. The rover does multiple separate experiments, so if a circuit necessary for only one of the experiments fails, it will not compromise the other experiments. However, if an error happens in the main circuits, it can compromise the whole mission.

Fault tolerance - Not all applications tolerate the same ER, so it is important to be flexible when creating the ATMR design. Also, it is important to avoid approximations at critical inputs. It should be noted that the inputs have different levels of importance, as certain inputs are more susceptible to errors than the rest. The generation of approximate functions for ATMR must emphasize the inputs that are not susceptible to errors and must ensure that the inputs that are more susceptible to errors remain intact. A different way to improve the reliability of the ATMR circuit is proposed in (GOMES; KASTENSMIDT, 2013). The authors analyze the impact of using transistor reorder and input reorder.

Tradeoff (area overhead x error rate) - A method that offers multiple options of approximate functions, with different area overhead and ER tradeoffs, to be used in the ATMR circuit is desirable. This is important because different projects have different constraints, so a solution can work better for a constraint and another solution can work better for a different constraint. However, an exhaustive generation of approximate functions can present high computational cost and this can be an impediment, especially for bigger circuits.

AMTR x Full-ATMR - The Full-ATMR design usually has a lower area overhead than the ATMR design. However, it is not always the case, as sometimes the approximate function can have an area bigger than the original function. The selection of the best approximate functions to build the Full-ATMR is a big challenge and some methods cannot produce Full-ATMR designs. Full-ATMR design is created in (GOMES et al., 2015a) using a slightly modified version of the Boolean factoring algorithm to generate the approximate modules.

Scalability - The computational cost is another key metric that needs to be optimized. Most proposed works do not prioritize the computational cost at first and the subsequent works try to optimize the computational cost with a small penalty in the area overhead and ER tradeoff. Most of these works do not have data about the computational cost available. In (GOMES; KASTENSMIDT, 2013; GOMES et al., 2014, 2015a), no data about the computational cost is available. However, tested circuits have no more than 8 inputs, which is a small circuit. In (HASSAN et al., 2018), circuits with up to 117 are tested, but no data about the computational cost is available. In

(ALBANDES et al., 2018; ALBANDES; AL., 2018; ALBANDES et al., 2019), circuits have up to 14 inputs and it takes more than 6 hours to generate the ATMR design in the worst case.

3 RELATED WORKS

Multiple works propose the use of approximate computing techniques to generate better modules for multiple modular redundancy designs (ARIFEEN; HASSAN; LEE, 2020). Some of these works will be presented in this chapter and compared at the end. It was decided to separate three groups of works to better explain their progress with time. The three groups are Boolean factoring, prime implicant (PI) expansions and reductions, and approximate library. Next, other important works in this area will be presented. Lastly, a comparison of the presented works was made to better view what exists in the literature.

3.1 Boolean Factoring

In (GOMES; KASTENSMIDT, 2013; GOMES et al., 2014, 2015a; GOMES, 2014), the author proposes an analysis of the impact of the input permutation and the transistor topology in ATMR and FATMR circuits. First, the author used to create the ATMR designs manually and test the impact of the input permutation and the transistor topology technique. Next, it is proposed the use of the Boolean factoring technique to generate the approximate functions. Finally, the authors propose a method to choose the best approximate functions to be used in the ATMR design.

In (GOMES; KASTENSMIDT, 2013), the author proposes two methods to customize an approximate module. The first method is the input permutation, which changes the inputs of the transistors without changing the topology or the logic function. In Fig. 15, we can see two different designs for a NAND gate that only differ in which input each transistor receives. Assuming that this design is being used in an ATMR design that the only unprotected vector is A = ‘1’ and B = ‘0’, which makes T2a and T1b to be ON, and T1a and T2b to be OFF. If a SET happens in node n1a, the fault will be propagated to the output and will become an error, because T2a is ON and this input vector is unprotected. However, the same will not happen if a SET happens in n1b, because T2b is OFF, so the fault will not propagate to the output. A SET occurring in the node n2a or n2b will not be masked and will become an error. Using this permutation, the amount of sensitive p-n junctions is reduced from 3/8 to 1/8.

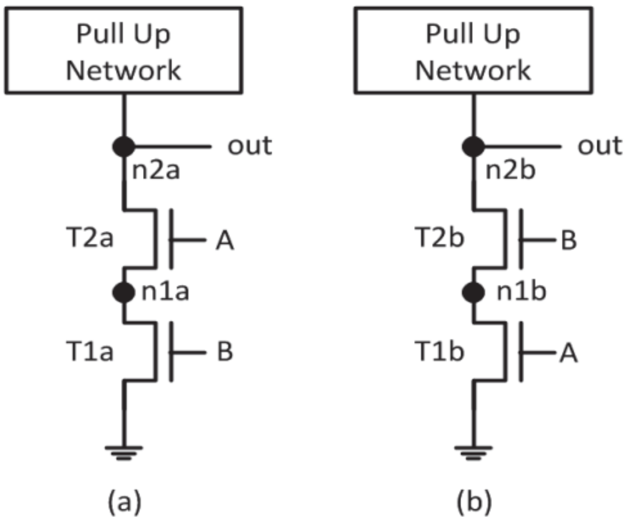


Figure 15 – Input permutation changes the input order to protect more p-n junctions. Source: (GOMES; KASTENSMIDT, 2013)

The second method is the topology variation and an example is shown in Fig. 16. It has a similar idea to the input permutation, but it changes the transistor disposition in the network. This method is better suited for complex gates. Fig. 16 shows two different topologies for the function not (A and (B or C)). Assuming that this design is being used in an ATMR design that the only unprotected vector is A = ‘0’, B = ‘0’, and C = ‘1’, which causes the transistors T2a and T2b to be ON and the other transistors to be OFF. If a SET occurs in node n1b, the fault will be propagated through T2b and it will become an error. On the other hand, if a SET occurs in node n1a, the fault propagates through T3a, because it is OFF. A SET occurring in node n2a or n2b will not be masked by any of the designs. This technique reduces the amount of sensitive p-n junctions for this function from 5/12 to 1/12.

In this first work, the author was using approximate functions for the ATMR design, but it was not explained how they were generated. In (GOMES et al., 2014), the author proposes the use of the Boolean factoring method to create the approximate functions and an example is shown in Fig. 17. First, a list of cofactors and cube cofactors are combined to create a set of allowed functions. Next, a bucket with all variables of the original function is created. The elements of this bucket are combined two by two, creating a new bucket with all the combinations of 2-literals functions. This process is repeated until you get in a bucket with the same amount of literals as the original function as shown in 17(a). The set of allowed functions is used to filter the functions during this process, removing functions that will not contribute to a good solution. In the end, the smaller subfunctions are inserted in bucket F and the subfunctions that are larger are put in bucket H shown in Fig. 17(b).

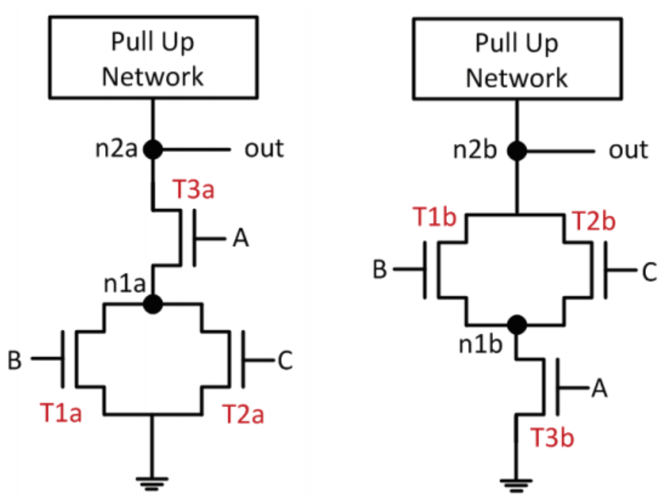


Figure 16 – Topology variation changes the topology of the transistor network to protect more p-n junctions. Source: (GOMES; KASTENSMIDT, 2013)

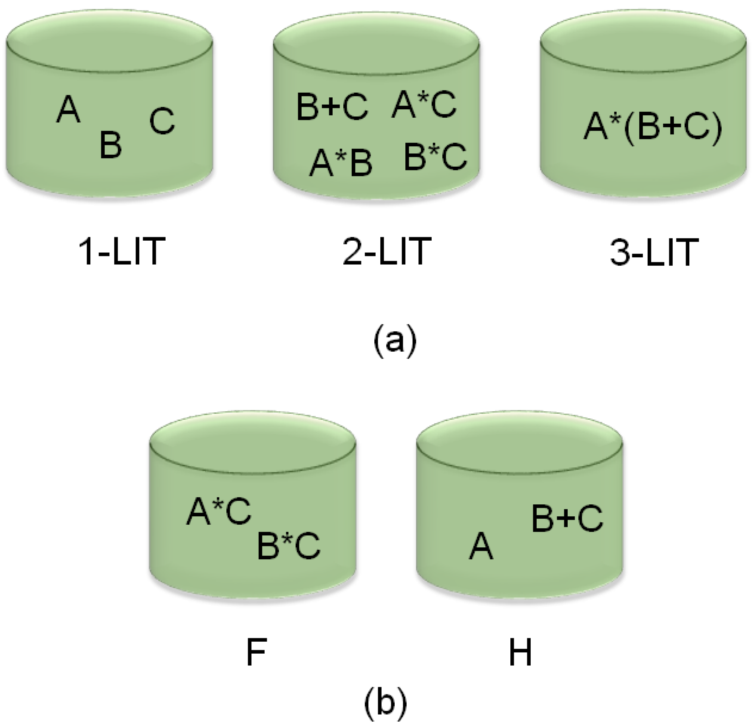


Figure 17 – Boolean factoring technique. (a) shows the buckets where the generated functions are stored in each step and (b) shows the selected under and over-approximate functions. Adapted from: (GOMES et al., 2014)

The use of Full-ATMR is proposed in (GOMES et al., 2015a,b). The author uses the Boolean factoring method to create the approximate modules for ATMR designs and Full-ATMR designs and compare them. The author explains in detail the methodology to choose the best approximate function candidates. The number of literals and the Hamming distance (number of unprotected vectors) are used to eliminate bad candidates. Table 2 shows five examples of possible H functions. The cost column is built by multiplying the number of literals by the Hamming distance, and this value is used to order the list of candidates. Comparing H1 and H6, the cost of H1 is smaller, but we can't remove H1 because it has a smaller area and it can be used as a tradeoff. However, if H1 and H14 are compared, the H14 function is bigger and has more unprotected vectors than the H1 function, so it can be considered a bad function that does not offer any tradeoff to be used. The same conclusion can be made about functions H3 and H4.

Table 2 – Evaluation of the approximate functions. H6 and H1 are selected as good approximate candidates. Adapted from: (GOMES et al., 2015b)

Cod	Function	Literals	Hamming	Cost
H6	(!D+!C)*(!E+!F)	4	3	12
H1	!D+!C	2	15	30
H14	(!E+!F)*(!B+(!D+!A))	5	9	45
H3	!B + (!D+!A)	3	23	69
H4	(!B+!D)+!C	3	23	69

The Boolean factoring technique is flexible, generating multiple approximate candidates that can be used in the final design, making it possible to choose the functions that better meet the project constraints. Also, this technique presents a good tradeoff between area overhead and protected p-n junctions, keeping the protected p-n junction ratio above 90% with only 33% area overhead in (GOMES et al., 2015a). Additionally, the use of input permutation and transistor topology techniques can increase the amount of protected p-n junctions. However, the Boolean factoring is an exhaustive method and no data about the computational cost is available, but all circuits tested have 8 or fewer inputs.

3.2 Prime Implicant Expansions and Reductionn

In (ARIFEEN et al., 2016; HASSAN et al., 2018; ARIFEEN et al., 2018), the authors propose a method to generate TMR circuits with the smallest area overhead possible within an acceptable threshold. First, the prime implicant expansions and reduction technique is used to generate the approximate modules for the ATMR architecture. The number of allowed complements is shared between the three modules as Fig. 18 shows. The number of allowed complements is the sum of N1, N2, and N3. Next, two approaches to improving this technique are presented. The first approach uses a heuristic method to generate the approximate functions, so it can be used for larger circuits. The second approach uses the concept of input vulnerability to better protect the circuit.

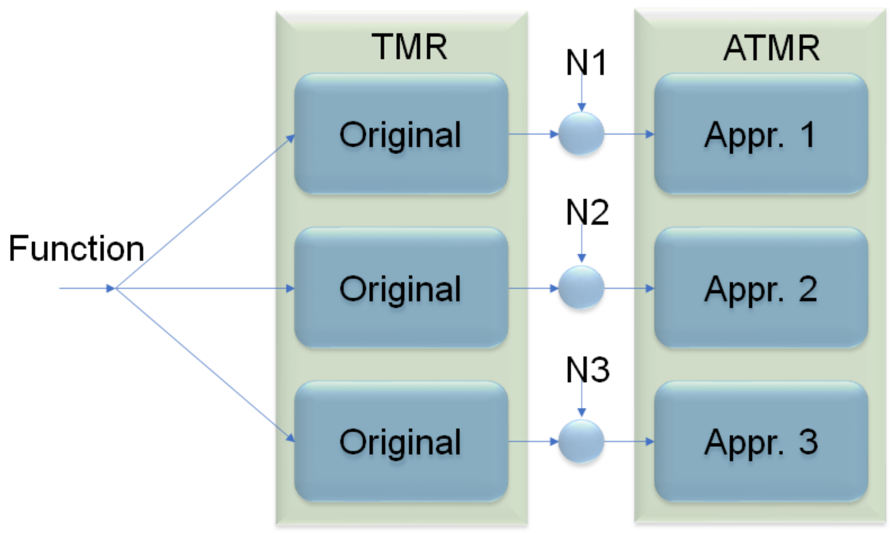


Figure 18 – The approximate functions are built based on the number of allowed complements (N1, N2, N3). Adapted from: (ARIFEEN et al., 2016)

In (ARIFEEN et al., 2016), it is presented a method that effectively uses the acceptable ER threshold and minimizes the area overhead to a minimum. The method uses the PI expansions and reduction technique to generate the approximate modules for the ATMR architecture, and it is shown in Fig. 19. Fig. 19(a) presents the Karnaugh map (KM) of the original circuit. Fig. 19(b) shows the first complement in the minterm "0011" which reduces the amount of cubes. In Fig. 19(c), the output of the minterm "0011" is locked in the original value, and another minterm is complemented ("0010"), reducing the number of cubes as well.In a similar way, Fig. 19(d) locks the output of the minterm "0010" and complement the next minterm in line ("0101"), which also reduces the number of cubes.

The method have to follow some steps to find the best approximate modules. First, a list of all minterm/maxterm to expand/reduce original PI (MEROP) is generated. Next, MEROPs are selected from the MEROP list until the allowed number of complements for each function is met. The selected MEROPs are complemented, obtaining the approximate functions. Every selected MEROP is blocked after its selection because the same minterm/maxterm cannot be complemented in two different approximations without compromising the correct operation of the ATMR circuit. The process, besides the MEROP list creation, is repeated until the maximum ER is obtained. During the process, area overhead local minimums are obtained and the global minimum is obtained at the end of the last round. The proposed method presents a literal count reduction when compared with (GOMES et al., 2014, 2015b). However, the method is exhaustive and cannot be used for larger circuits.

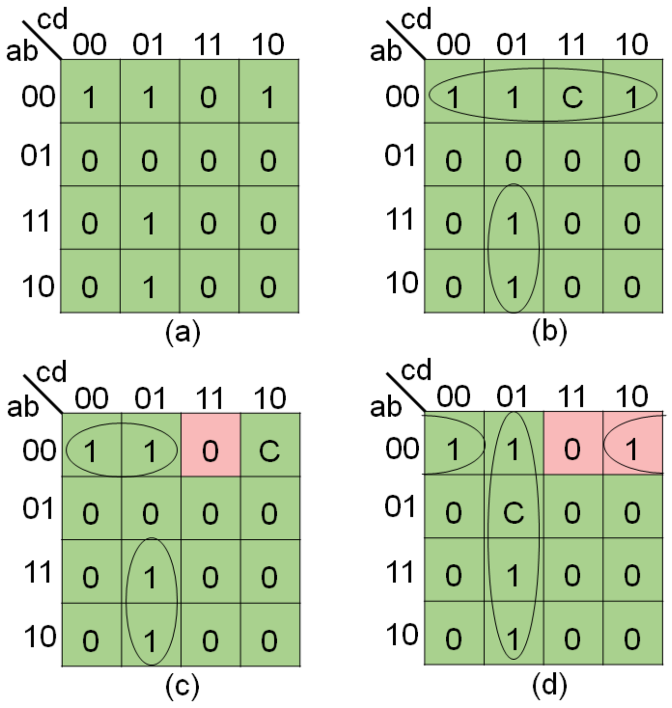


Figure 19 – Karnaugh maps of (a) original function, (b) approximate function 1, (c) approximate function 2, and (d) approximate function 3. Red cells highlight locked minterms. Adapted from: (HASSAN et al., 2018)

A heuristic to target larger circuits is proposed in (HASSAN et al., 2018) to improve the previous work. The heuristic uses the same MEROP list and complement selection technique as the previous work. However, after the selected MEROPs are complemented, the functions are ordered by literal count and just a determined number of functions will continue to be approximated. Using this heuristic will produce local minimums, but the global minimum can be not found in some cases. This work uses a clustering technique in the ABC tool(BRAYTON; MISHCHENKO, 2010) and it represents intermediate circuits with the internal nodes of Boolean circuits having 10-15 inputs for circuits with more than 15 inputs. The apex5 benchmark, a function with 117 inputs and 88 outputs, was approximated using the clustering technique, but there is no extra data about how long it takes to generate the approximate functions.

The concept of input vulnerability for ATMR was proposed in (ARIFEEN et al., 2018). Different input vectors can have different levels of importance. Therefore, a group of input vectors can be more vulnerable to errors than the others, as the error can easier propagate to the output. ATMR circuits are more susceptible to errors than the TMR circuits, so it is important to approximate critical inputs. The proposed method recognizes the critical input space using automatic test pattern generation (ATPG) and restricts the vulnerable input space as blocked when creating the MEROP list. The method is similar to the method presented in (HASSAN et al., 2018), adding a pre-blocking phase before creating the MEROP list and starting the same process. The ATPG tool identifies the input vectors that are more susceptible to errors and block them to be selected to be complemented in Fig. 20, blocking these input vectors throughout the process. The rest of the process happens the same way that was explained for Fig. 19. This happens so all three modules have the same output for this input vector and in case of a fault happening in one module, the other two modules will mask the fault with the voter. The pre-blocking stage reduces the search space for approximation candidates even further.

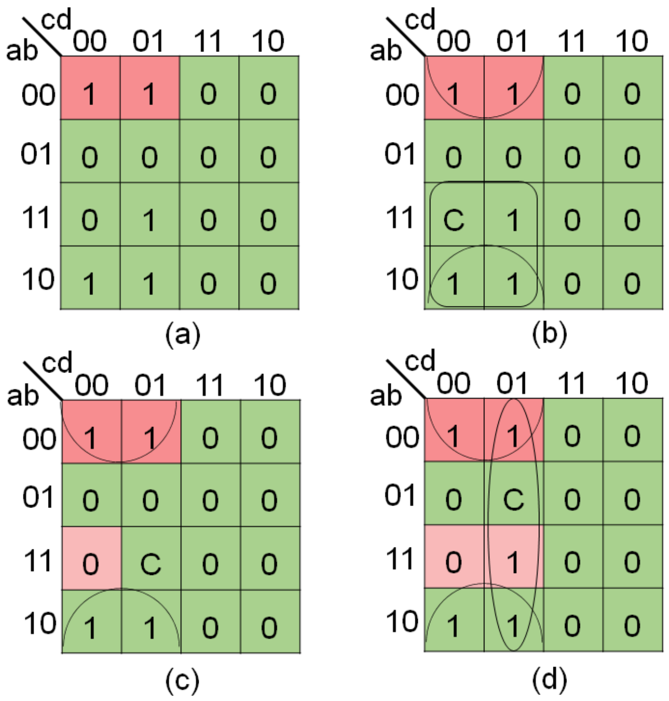


Figure 20 – Karnaugh maps of (a) original function, (b) approximate function 1, (c) approximate function 2, and (d) approximate function 3. Red cells highlight locked input vectors by the method, and dark red cells highlight the input vectors locked by the ATPG tool. Adapted from: (HASSAN et al., 2018)

3.3 Approximate Library

The generation of approximate circuits happens at a different abstraction level in (ALBANDES et al., 2018; ALBANDES; AL., 2018; ALBANDES et al., 2019; GOMES, 2018). An approximate library is used to replace the gates with predetermined approximate gates. An example of the use of the approximate library is shown in Fig. 21, where possible substitutions, to under or over-approximate a gate, are listed. Every work presented in this subchapter uses the approximate library technique to create the approximate functions. The improvements in later works are in the computational cost of choosing the best gate to be approximate and which approximate option is the best to be used.

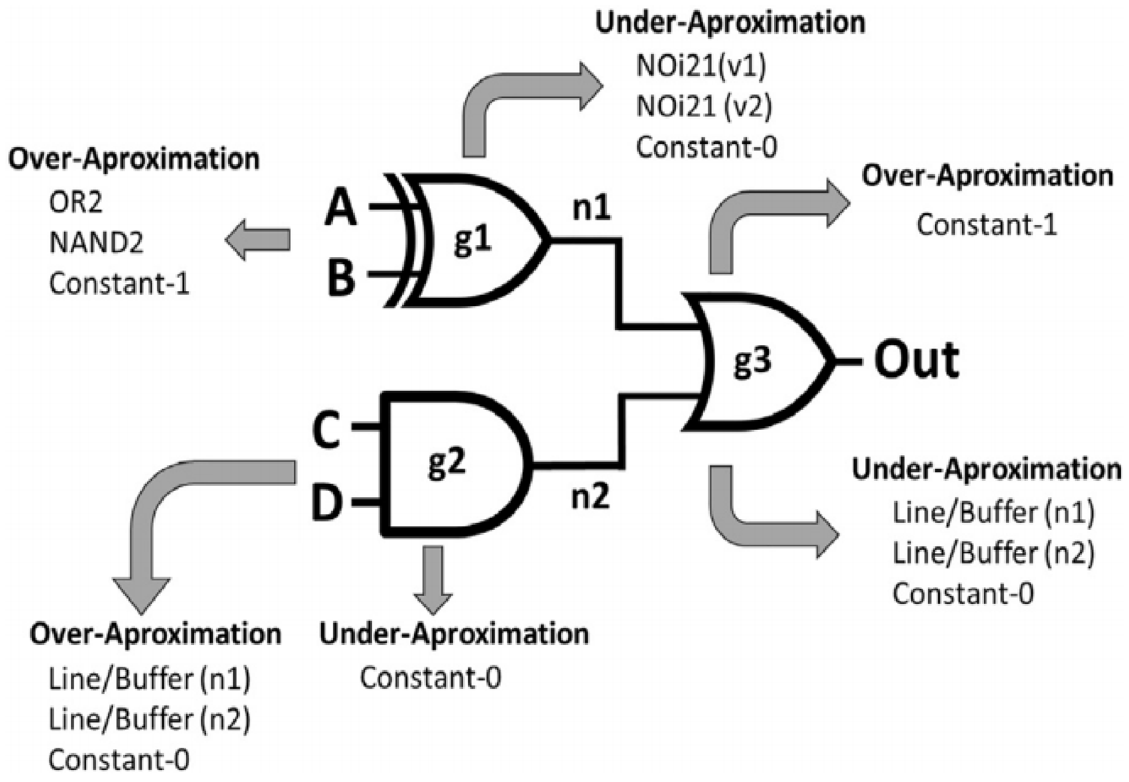


Figure 21 – The approximate library technique gives each gate in the circuit has multiple approximation options. Source: (ALBANDES et al., 2019)

In (ALBANDES et al., 2018), a genetic algorithm (GA) is used to generate new approximate functions, keeping the best functions for the next generation. Multi-Objective Optimization (MOO) is used to decide what are the best functions. MOO sorting is used to optimize the selection of the best function that depends on multiple metrics to be evaluated. The two metrics used in the MOO sorting are area overhead which needs to be minimized and fault coverage which needs to be maximized. The use of both techniques combined is known as MOOGA. This technique is exhaustive and works only in small circuits.

(ALBANDES; AL., 2018) proposes a heuristic guided by testability and observability techniques to assist in generating approximate functions with a lower computational cost. This heuristic is used, after the best candidate for approximation is found, to find the best gate transformation using the approximate library technique. Testability and observability techniques are used to evaluate the impact of each gate on circuit outputs. Testability values are obtained when a gate output is stuck-at a determined value and the impact in the circuit output is analyzed. The gate with the lowest testability measure is the best candidate to be approximated. Next, the heuristic is used to decide what is the best gate transformation. The best transformation is the one where the percentage of output bits that change when compared with the original circuit is the lowest. This process can use an exhaustive method or a random process to estimate the circuit difference.

An iterative approach for larger circuits is proposed in (ALBANDES et al., 2019). First, an ordered table with the testability measures is built, with the best candidate gates being on the top. Next, the first candidate gate is removed from the table, and the best gate transformation for this gate is evaluated. This table is used until all the gate measures are used or for a determined number of approximations. A new table is then generated to continue the approximation. An ATMR design is generated for each built table. The process continues until a new table cannot be generated anymore or for a determined number of approximations.

3.4 Additional Related Works

In (MOHANRAM; TOUBA, 2003), a partial masking scheme based on the TMR architecture is proposed. A combination of two techniques is used to reduce the area overhead of the TMR design. The first technique is the cluster sharing. The soft error susceptibility of certain nodes in the logic circuit can be orders of magnitude higher than that of the other nodes in the design. Therefore, the nodes with high soft error susceptibility are triplicated, and the clusters with low soft error susceptibility are used only once, sharing their logic with all modules. Since the nodes that have a higher soft error susceptibility are masked by the TMR design, it is possible to achieve a significant reduction in the soft error failure rate in this manner. The second technique used in the partial masking scheme is the dominant value. This technique exploits the fact that the logic ’0’ (logic ’1’) soft error susceptibility of certain primary outputs can be close to an order of magnitude higher than when it is at logic ’1’ (logic ’0’). The idea is to identify such outputs and replace triplication by duplication when it happens. The majority voter is replaced by an AND (OR) gate. For example, if an output has a logic ‘1’ failure rate one order of magnitude higher than its logic ‘0’ failure rate. The majority voter is replaced by an OR gate, and the part of the circuit that computes only this output is only doubled instead of triplicated.

In (SANCHEZ-CLEMENTE et al., 2012), the concept of unate functions is used to create the approximate functions. First, it is necessary to compute the parity of every input and gate for each output. Setting a gate to a determined logic value can generate a under/over-approximate function based on this value and the parity of this input/gate. In a network, lines with no parity can be duplicated and divided in two sets with even and odd parities. This temporarily creates a larger circuit that allows the application of the described technique. Duplications that are not removed after approximation can be removed later on by resynthesizing the approximate logic circuits.

In (DEVEAUTOUR et al., 2020), a novel modular redundancy technique is proposed. The QAMR uses four approximate modules to achieve the same reliability with a lower area overhead than the TMR. The approximation method needs to ensure that three replicas of the same output are always available. Therefore, the same majority voter used in the TMR technique, can be used in the QAMR technique. The approximate modules are generated by arbitrarily create complementary groups of outputs in a random manner. The main purpose of this works was to show the technique feasibility.

3.5 Final Considerations

There are multiple works trying to optimize the generation of approximate functions to ATMR designs. These works use different methodologies, benchmarks and evaluation metrics. Table 3 compare some of the works in literature. The first column shows the main author and year of publication. The second column present the methodology used in each work. The third column presents the benchmark used to test and evaluate the method. Finally, the last two columns show what area and ER metrics are used to present the results.

Table 3 – Comparison of different methodologies, benchmarks, area, and ER metrics used in literature.

[width=16cm]tabelacomp2.png

4 PROPOSED METHOD

This Chapter presents the method proposed to generate the approximate modules for the ATMR architecture with a low computational cost. A simple and efficient method would be necessary to reach this objective. Also, the method has to obtain a good area overhead and ER tradeoff. In order to reduce the area overhead, two options are presented, to create larger cubes (adding '1's) or to reduce the number of cubes (removing '1's). However, the proposed method should change the output the least possible to keep the ER as low as possible. The process explores the correlation between the input variables and the output and uses this concept to create the approximate modules. The method generates an over-approximate function and an under-approximate function.

Table 4 shows the truth table of the original function, that is the function that needs to be approximated. It is used, as a study case, the function S=(!A*B*C)+(A*!B)+(A*!C) to analyze the impacts of the correlation between input variables and output. This function has 4/50% of its outputs at '1'.

Table 4 – Truth table of the original function

A	B	C	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0
Output = 1			4

To develop the over-approximate function candidates, we check how many times each variable is '1' and the output is also '1'. Next, we use a logic OR gate between the input variables and the output. This guarantees that every output that used to be '1' still a '1' for these functions.

The Table 5 shows the correlation between each input variable and the output. The value is ’1’ when both variable and output are ’1’, and ’0’ otherwise. The last line presents how many times they were both ’1’, which is the correlation. For our study case, variable A has correlation 3, variable B has correlation 2 and variable C has correlation 2.

Table 5 – Number of times that each input variable is ’1’ and the output is ’1’.

Var = 1 & output = 1		
A	B	C
0	0	0
0	0	0
0	0	0
0	1	1
1	0	0
1	0	1
1	1	0
0	0	0
3	2	2

In Fig.22 we can see the KM of the original function and the over-approximate functions. Also, we can see the resulting SOP of each of these functions. We can see in Fig.22(a) the KM of the original function and the resulting SOP that has 7 literals. Next, in Fig.22(b) we can see the KM of the first approximate function and, because we can create a larger cube and reduce the number of cubes, the resulting SOP has only 3 literals. We can notice that it differs in only 1 minterm with the output, adding 1 uncovered vector. In a similar way, Fig.22(c) and Fig.22(d) show the KMs of other 2 approximate functions. In these cases, the literal count is reduced to only 2, but they also increase the number of uncovered minterms to 2.

In the same way, to develop the under-approximate function options, we check how many times each variable is ’0’ and the output is also ’0’. Next, we use a logic AND gate between the input variables and the output. This guarantees that every output that used to be ’0’ still a ’0’ for these functions.

Table 6 shows the correlation between each input variable and the output. The value is ’1’ when both variable and output are ’0’, and ’0’ otherwise. The last line shows how many times they were both ’0’, which is the correlation. For our study case, variable A has correlation 3, variable B has correlation 2 and variable C has correlation 2.

In Fig.23 we can see the KM of the original function and the under-approximate functions. Also, we can see the resulting SOP of each of these functions. We can see in Fig. 23(a) the KM of the original function and the resulting SOP that has 7 literals. Next, in Fig.23(b) we can see the KM of the first approximate function and, because we reduce the number of cubes, the resulting SOP has 4 literals and adds only 1

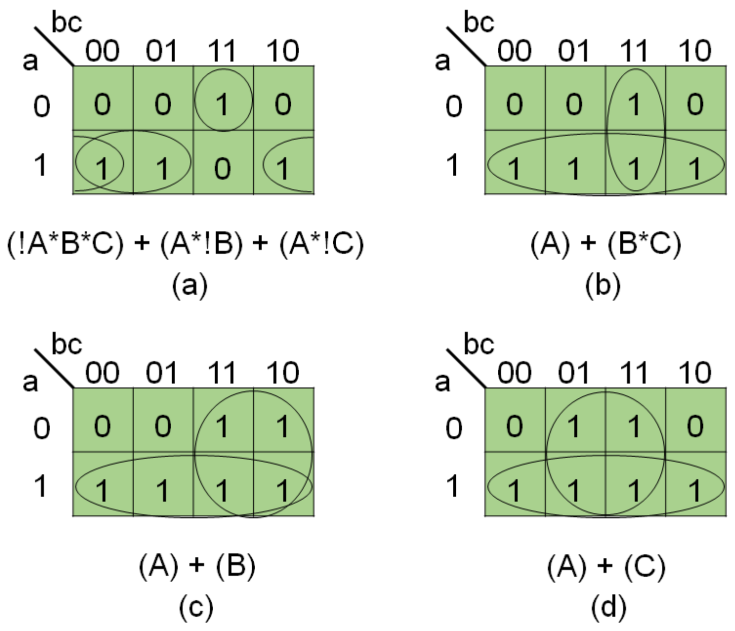


Figure 22 – Karnaugh maps of the original function and the generated over-approximate functions.

uncovered vector. In a similar way, Fig.23(c) and Fig.23(d) show the KMs of other 2 approximate functions. In these cases, the literal count is reduced to 6 because we reduced the number of cubes. It is important to notice that it can create smaller cubes (more literals), but the literal count usually ends up reducing.Also, they increase the number of uncovered minterms to 2 in both cases.

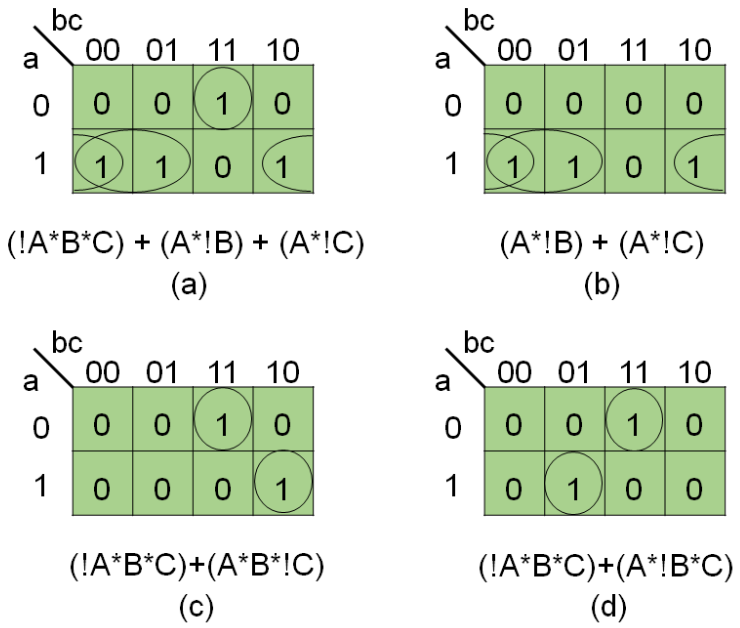


Figure 23 – karnaugh Maps of the original function and the generated under-approximate functions.

We can check Table 7 to decide what is the best variable to use to create the approximate functions. The first column is the circuit description. The next column

Table 6 – Number of times that each input variable is ’0’ and the output is ’0’.

Var = 0 & output = 0		
A	B	C
1	1	1
1	1	0
1	0	1
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
3	2	2

shows the correlation we got before. The last two columns show how many literals the function has and how many uncovered vectors it adds. The first 2 rows are the original circuit and the TMR design. The other rows are separated in groups of the two approximate functions and the ATMR design for each variable. Even though variable A has more literals in the over-approximate function, the ATMR design using the variable A presents the lowest area. Also, variable A adds the least amount of uncovered vectors (2), one for each approximate function. Variables B and C have less literals in the under-approximate functions, but their ATMR designs end up with more literals and more uncovered vectors.

We noticed that using the input variable that presents the highest correlation with the output is the best option to create the approximate functions for the ATMR architecture, adding a few uncovered vectors and reducing the area overhead when comparing to the TMR design.

In Fig. 24 we can see all the steps of our implementation. Input treatment, generation of the approximate functions, how to deal with multiple outputs circuits, simplification steps, technological mapping, generation of the ATMR design and fault injection analysis. In the end of the process, we have a Verilog file with the generated ATMR design, the area values and the ER of the original circuit and the ATMR design. Some of these steps will be explained in more details.

First, the input file is loaded in the data structure that represents the truth table of the circuit. In case the circuit has multiple outputs, the data about each output will be loaded separately. It is encouraged to use variable types with big sizes because the computational cost to decide the best variable to use and generate the approximate functions is reduced. The computational cost reduction happens because it requires less computational operations.

The core of the proposed method is the analysis of the correlation of each variable

Table 7 – Analysis of the number of literals, uncovered vectors, and the correlation of the input variables and the output of the ATMR designs for each input variable. An input variable with higher correlation generated an ATMR design with a lower area overhead and fewer uncovered vectors.

	Correlation	Literals	Uncovered Vectors
Original	-	-	-
TMR	-	21	0
A OR S	3	3	1
A AND S	3	4	1
ATMR A	3	14	2
B OR S	2	2	2
B AND S	2	6	2
ATMR B	2	15	4
C OR S	2	2	2
C AND S	2	6	2
ATMR C	2	15	4

with the output and how this is used to generate the approximate functions. This process is explained in more detail in Algorithm 1. The function ApprGeneration uses the data about how many inputs and outputs the circuits have and a vector of truth tables with all the outputs. This function returns a pair of approximated functions for every output.

To find out what is the best variable to generate the over-approximate function, or 0-approximation function, it is necessary to make a logical AND with the output (line 7). The AND is used because it is necessary to check when both the variable and the output are '1'. After this, it is necessary to count how many '1's the result of the logical operation has (line 8). Next, it is checked which of the variables has the biggest value (line 11), because this is the variable that has the biggest correlation with the output. Finally, the selected variable is used to generate the over-approximate function. A logical OR is done with the selected variable and the output to obtain the over-approximate function (line 21). The OR is used to keep all the '1's that are in the original output and it makes the approximate functions be the same to the variable that has the highest correlation. This whole process is repeated for every output (line 3).

A similar process is necessary to discover what is the best variable to generate the under-approximate function, or 1-approximation function. This generation requires the use of a logical NOR with the output (line 15). The NOR is used to check which vectors are '0' when the output is '0'. After this, it is necessary to count how many '1's the result of the previous operation has (line 16). Next, it is checked which of the variables has

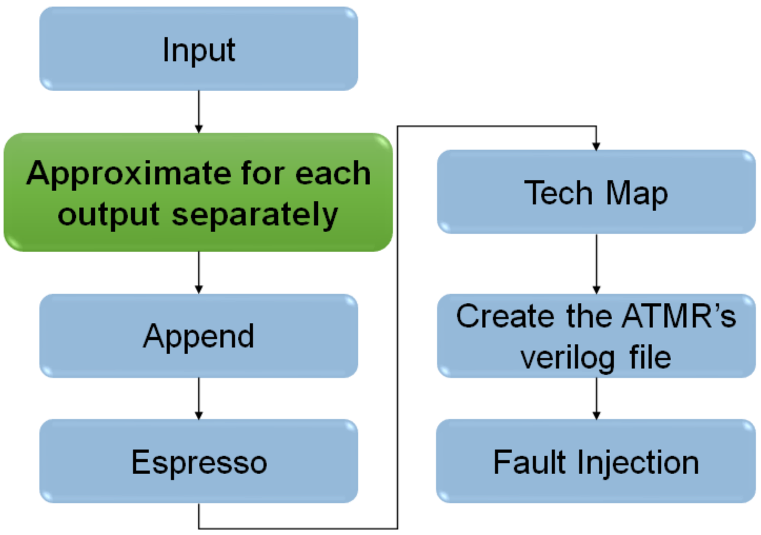

Input
Approximate for each output separately
Append
Espresso
Tech Map
Create the ATMR's verilog file
Fault Injection

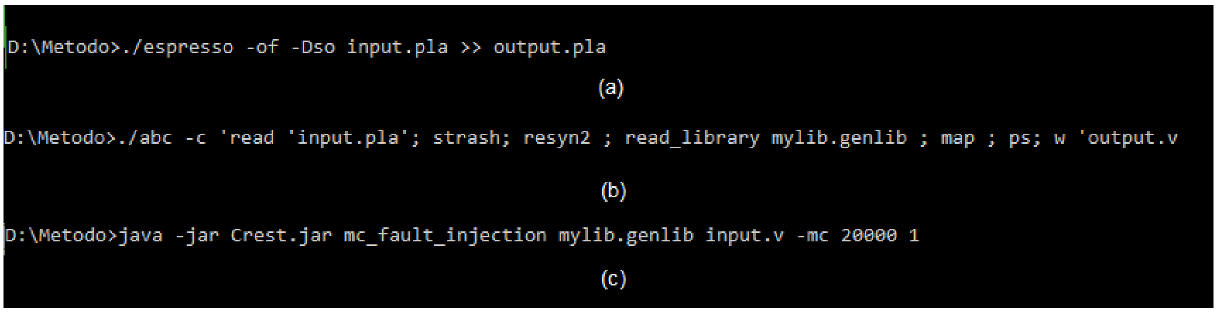
Figure 24 – Flowchart of the implementation. The input function is approximated, then an append is made for multiple output functions. Next, the function is simplified with ESPRESSO, and the technological mapping is done. Lastly, the final ATMR design is built and a fault injection tool is used to test the design's robustness.

Algorithm 1 Approximate Function Generation

```
1:  function APPRGENERATION($Ninputs, Noutputs, *ttable$)
2:      $output \leftarrow 0$
3:      while $output < Noutputs$ do
4:
5:          $input \leftarrow 0$
6:          while $input < Ninputs$ do
7:              $aux[input] \leftarrow ttable[output]$ AND $var[input]$
8:              $count[input] \leftarrow$ amount of bits '1' in $aux[input]$
9:              $input \leftarrow input + 1$
10:         end while
11:         $iover \leftarrow$ maxindex($count$)  ▷ Return the index of the biggest value in a vector
12:
13:         $input \leftarrow 0$
14:         while $input < Ninputs$ do
15:             $aux[input] \leftarrow ttable[output]$ NOR $var[input]$
16:             $count[input] \leftarrow$ amount of bits '1' in $aux[input]$
17:             $input \leftarrow input + 1$
18:         end while
19:         $iunder \leftarrow$ maxindex($count$)
20:
21:         $overappr \leftarrow ttable[output]$ OR $var[iover]$
22:         $underappr \leftarrow ttable[output]$ AND $var[iunder]$
23:         $output \leftarrow output + 1$
24:     end while
25: end function
```

the biggest value (line 19), because this is the variable that has the biggest correlation with the output. Finally, the selected variable is used to generate the over-approximate function. A logical AND is done with the selected variable and the output to obtain the first approximate function (line 22). The AND is used to keep all the ’0’s that are in the original output and it makes the approximate functions be the same to the variable that has the highest correlation. This whole process is repeated for every output (line 3).

After the generation of the approximate functions, it is necessary to make an append in the approximate circuits if the original circuit had multiple outputs. Next, the method uses ESPRESSO (RUDELL; SANGIOVANNI-VINCENTELLI, 2003) to simplify the functions, increasing the area reduction. Next, the ABC Tool (BRAYTON; MISHCHENKO, 2010) is used to further simplify the circuit and to do the technological mapping using a gate library. The mapping process returns a file with the Verilog description of the approximate circuit. An example of the ESPRESSO and ABC command lines used can be seen in Fig. 25.



```
D:\Metodo>./espresso -of -Dso input.pla >> output.pla
```

(a)

```
D:\Metodo>./abc -c 'read 'input.pla'; strash; resyn2 ; read_library mylib.genlib ; map ; ps; w 'output.v
```

(b)

```
D:\Metodo>java -jar Crest.jar mc_fault_injection mylib.genlib input.v -mc 20000 1
```

(c)

Figure 25 – (a) ESPRESSO command line used to simplify the function. (b) ABC command line used to do the technological mapping. (c) command line used to do the fault injection.

The final step is to create the ATMR Verilog file. To create this design, the original circuit is put together with the over and under-approximate functions and majority voters are added to complete the ATMR design. It may be necessary to rename some inputs, outputs and signals.

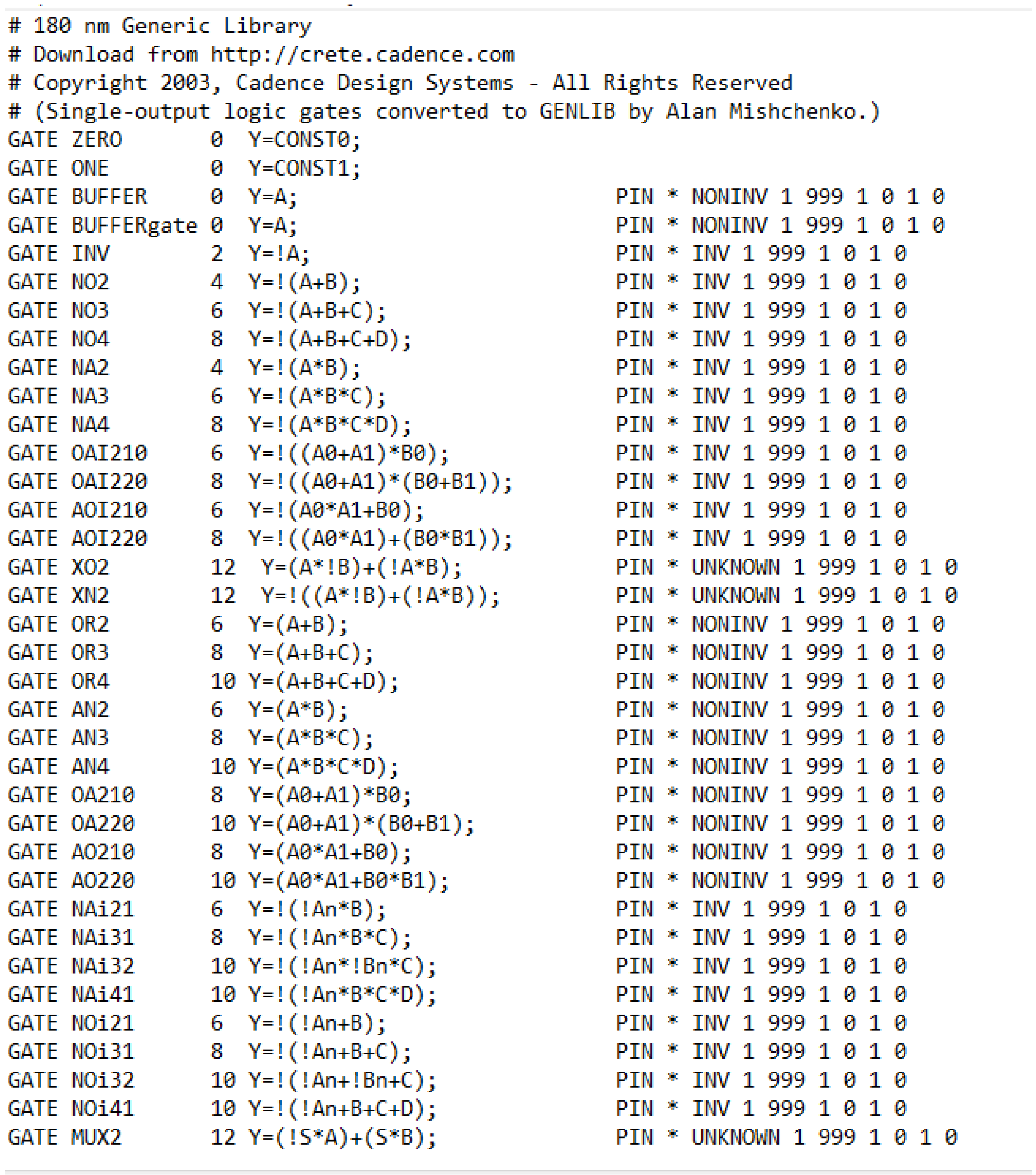
A fault injection tool (FARIAS, 2022) is used to to estimate how reliable the original circuit and the ATMR design are and the command line can be seen in Fig. 25 (c). It injects SETs in different gates of the circuits, one at a time, and checks the output of the voter. If the voter gives the wrong output, then an error is added to the analysis. The ER is the percentage of the SETs that are not masked by the ATMR design and become an error.

5 RESULTS

The method was developed in C and automatized using shell scripts to call the different functions and tools. Also, the script saves the output files in the correct folders and organize the result sheets. The method was tested in a group of circuits from the LGSynth93 benchmark. This group has 22 circuits and the results from seven of them were compared with (ALBANDES et al., 2019). The circuits have between 7 and 17 inputs and between 1 and 63 outputs. Some of the generated approximate modules presented a small area reduction, so it was created a version removing those approximate module and using the original module instead to reduce the ER. The custom library cell used in this work is presented in Fig. 26 and it is the same used in (ALBANDES et al., 2019). The library has 36 gates with 0 (constants) to four inputs and one output. The area overhead is calculated by adding up the areas of the original circuit and both approximate circuits. Then, the total area is divided by the area of the original circuit. The voter is not considered in the area analysis.

A Monte Carlo based fault injection tool was used to evaluate the ER of the original circuit and the generated ATMR design. The procedure injected 20k faults, propagated through a fault injection campaign. This fault injection procedure was chosen because of the excellent tradeoff between computational cost and accuracy. Consequently, this Monte Carlo statistical methodology provides better performance than traditional exhaustive approaches.

Table 8 presents the results of all investigated circuits. The first column is the name of the circuit, and the rows with a ”*” have the modules that present area larger than 80% of the original circuit exchanged for the original module. This is done to reduce the ER with a small area overhead penalty. The I/O columns indicates the amount of inputs and outputs of the circuit. Time indicates the execution time (ms) to generate the over-approximate module and the under-approximate module. The column Area Over. shows the area overhead when comparing the ATMR circuit with the original circuit, voters are not included in this area. The column ER Original presents the ER of the original circuit without any protection, column ER ATMR is the ER of the generated ATMR circuit, and the ER Ratio column shows how much the ER was reduced from



```
# 180 nm Generic Library
# Download from http://crete.cadence.com
# Copyright 2003, Cadence Design Systems - All Rights Reserved
# (Single-output logic gates converted to GENLIB by Alan Mishchenko.)
GATE ZERO       0  Y=CONST0;
GATE ONE        0  Y=CONST1;
GATE BUFFER     0  Y=A;                        PIN * NONINV 1 999 1 0 1 0
GATE BUFFERgate 0  Y=A;                        PIN * NONINV 1 999 1 0 1 0
GATE INV        2  Y=!A;                       PIN * INV 1 999 1 0 1 0
GATE NO2        4  Y=!(A+B);                   PIN * INV 1 999 1 0 1 0
GATE NO3        6  Y=!(A+B+C);                 PIN * INV 1 999 1 0 1 0
GATE NO4        8  Y=!(A+B+C+D);               PIN * INV 1 999 1 0 1 0
GATE NA2        4  Y=!(A*B);                   PIN * INV 1 999 1 0 1 0
GATE NA3        6  Y=!(A*B*C);                 PIN * INV 1 999 1 0 1 0
GATE NA4        8  Y=!(A*B*C*D);               PIN * INV 1 999 1 0 1 0
GATE OAI210     6  Y=!((A0+A1)*B0);            PIN * INV 1 999 1 0 1 0
GATE OAI220     8  Y=!((A0+A1)*(B0+B1));       PIN * INV 1 999 1 0 1 0
GATE AOI210     6  Y=!(A0*A1+B0);              PIN * INV 1 999 1 0 1 0
GATE AOI220     8  Y=!((A0*A1)+(B0*B1));       PIN * INV 1 999 1 0 1 0
GATE XO2        12  Y=(A*!B)+(!A*B);           PIN * UNKNOWN 1 999 1 0 1 0
GATE XN2        12  Y=!((A*!B)+(!A*B));        PIN * UNKNOWN 1 999 1 0 1 0
GATE OR2        6  Y=(A+B);                    PIN * NONINV 1 999 1 0 1 0
GATE OR3        8  Y=(A+B+C);                  PIN * NONINV 1 999 1 0 1 0
GATE OR4        10 Y=(A+B+C+D);                PIN * NONINV 1 999 1 0 1 0
GATE AN2        6  Y=(A*B);                    PIN * NONINV 1 999 1 0 1 0
GATE AN3        8  Y=(A*B*C);                  PIN * NONINV 1 999 1 0 1 0
GATE AN4        10 Y=(A*B*C*D);                PIN * NONINV 1 999 1 0 1 0
GATE OA210      8  Y=(A0+A1)*B0;               PIN * NONINV 1 999 1 0 1 0
GATE OA220      10 Y=(A0+A1)*(B0+B1);          PIN * NONINV 1 999 1 0 1 0
GATE AO210      8  Y=(A0*A1+B0);               PIN * NONINV 1 999 1 0 1 0
GATE AO220      10 Y=(A0*A1+B0*B1);            PIN * NONINV 1 999 1 0 1 0
GATE NAi21      6  Y=!(!An*B);                 PIN * INV 1 999 1 0 1 0
GATE NAi31      8  Y=!(!An*B*C);               PIN * INV 1 999 1 0 1 0
GATE NAi32      10 Y=!(!An*!Bn*C);             PIN * INV 1 999 1 0 1 0
GATE NAi41      10 Y=!(!An*B*C*D);             PIN * INV 1 999 1 0 1 0
GATE NOi21      6  Y=!(!An+B);                 PIN * INV 1 999 1 0 1 0
GATE NOi31      8  Y=!(!An+B+C);               PIN * INV 1 999 1 0 1 0
GATE NOi32      10 Y=!(!An+!Bn+C);             PIN * INV 1 999 1 0 1 0
GATE NOi41      10 Y=!(!An+B+C+D);             PIN * INV 1 999 1 0 1 0
GATE MUX2       12 Y=(!S*A)+(S*B);             PIN * UNKNOWN 1 999 1 0 1 0
```

Figure 26 – Custom library cell used in ABC tool to do the technological mapping.

the ER Original value to the ER ATMR value. The generated ATMRs have an average area overhead of 126.78% while reducing the average ER from 27.16% to 12.21%, a 46.25% reduction. The generation of these functions is fast, with the largest circuit taking only 214.804ms to be approximated.

Table 8 – Benchmark data and time,area, and ER results obtained by the proposed method.

Bench			Metrics				
Circuit	Inputs	Outputs	Time(ms)	Area Overhead	ER Original	ER ATMR	ER Ratio
5xp1	7	10	0.289	118.97%	44.86%	19.67%	43.85%
9sym	9	1	0.094	103.36%	6.44%	2.95%	45.81%
9symml	9	1	0.095	103.36%	6.82%	2.87%	42.08%
al2	16	47	126.456	122.81%	59.61%	29.37%	49.27%
al2*	16	47	124.212	124.12%	59.61%	28.58%	47.94%
alcom	15	38	52.699	113.71%	59.92%	29.03%	48.45%
alcom*	15	38	51.378	115.43%	59.92%	28.51%	47.58%
alu1	12	8	1.241	230.65%	44.74%	19.74%	44.12%
alu1*	12	8	1.213	200.00%	44.74%	0.00%	0.00%
alu2	10	6	0.414	115.64%	14.24%	7.04%	49.44%
alu4	14	8	5.403	118.22%	10.94%	4.67%	42.69%
amd	14	24	14.731	107.06%	18.06%	11.77%	65.17%
amd*	14	24	14.567	121.17%	18.06%	11.07%	61.30%
apex4	9	19	0.829	109.74%	12.70%	6.48%	51.02%
b2	16	17	45.958	128.96%	13.28%	5.80%	43.67%
b12	15	9	11.276	139.81%	33.66%	15.36%	45.63%
b12*	15	9	11.568	151.85%	33.66%	8.56%	25.43%
clip	9	5	0.332	133.46%	30.39%	11.83%	38.93%
ex5	8	63	1.938	128.80%	30.15%	22.43%	74.39%
ex5p	8	63	1.860	114.81%	30.47%	20.43%	67.05%
ex7	16	5	13.294	122.50%	38.49%	14.88%	38.66%
gary	15	11	13.455	122.97%	9.39%	4.56%	48.56%
gary*	15	11	13.561	142.16%	9.39%	0.93%	9.90%
intb	15	7	9.110	91.76%	9.08%	4.43%	48.79%
max1024	10	6	0.362	158.52%	12.77%	5.37%	42.05%
misex3	14	14	9.233	86.02%	8.58%	6.99%	81.47%
prom1	9	40	1.628	107.98%	14.38%	6.49%	45.13%
vda	17	39	214.804	115.95%	26.18%	12.17%	46.49%
Average			26.500	126.78%	27.16%	12.21%	46.25%

The misex3 circuit has the smallest area overhead, and more detail is presented in Fig. 9. The first two columns present basic information about the circuit and how long it took to generate the approximate functions. The other columns show area data in the first two rows and ER data in the last two rows. The area columns present data about the original circuit and both approximate modules area, the area of the TMR and ATMR designs, and the area overhead of the ATMR design. Information about the total amount of injected faults, detected errors, and the ER for both original and

ATMR design are presented. The misex3 circuit has 14 inputs and 14 outputs, and the generated ATMR has 86.02% area overhead and 8.13% ER, which is 75.51% of the ER of the original circuit. The over-approximate module has only 11.27% of the original circuit area, and the under-approximate module has 74.76%.

Table 9 – Misex3's ATMR design data.

Circuit	misex3	Area	Original	Over	Under	TMR	ATMR	Overhead
Inputs	14		4508	508	3370	13524	8386	86.02%
Outputs	14	Error Rate	Original	Errors	ER	ATMR	Errors	ER
Time (ms)	9.233		20000	2152	10.76%	20000	1625	8.13%

The alcom circuit has one small module and one big module, and more detail is presented in Fig. 10. The alcom circuit has 15 inputs and 38 outputs, and the generated ATMR (a) has 113.71% area overhead and 43.19% ER, which is 54.92% of the ER of the original circuit. The over-approximate module has only 15.43% of the original circuit area, and the under-approximate module has 98.29%. As the under-approximate module has more than 80% of the original circuit, a second version was created with a second original module instead of the under-approximate module. The generated ATMR (b) has 115.86% area overhead and 41.86% ER, which is 53.22% of the ER of the original circuit. The increase in area overhead and decrease in ER are similar, so the project constraints would determine which design is better.

Table 10 – Alcom's ATMR designs data.

Circuit	alcom	Area	Original	Over	Under	TMR	ATMR	Overhead
Inputs	15		350	54	344	1050	748	113.71%
Outputs	38	Error Rate	Original	Errors	ER	ATMR	Errors	ER
Time (ms)	52.699		20000	15729	78.65%	20000	8638	43.19%

(a)

Circuit	alcom*	Area	Original	Over	Under	TMR	ATMR	Overhead
Inputs	15		350	54	350	1050	754	115.43%
Outputs	38	Error Rate	Original	Errors	ER	ATMR	Errors	ER
Time (ms)	51. 537		20000	15729	78.65%	20000	8371	41.86%

(b)

The gary circuit has the biggest reduction in ER, and more detail is presented in Fig. 11. The gary circuit has 15 inputs and 11 outputs, and the generated ATMR (a) has 122.97% area overhead and 6.79% ER, which is 46.73% of the ER of the original circuit. The over-approximate module has 80.81% of the area of the original circuit and the under-approximate module has 42.16%. A second version was created using a second original module instead of the under-approximate module because the over-approximate module has more than 80% of the original circuit. The second ATMR design (b) has 142.16% area overhead and 3.67% ER, which is 25.22% of the ER of the original circuit. The ER of the second design is reduced to 53.98% of the ER of the first design, with 15.60% extra area overhead. The decision of the best design depends on the project constraints again, but the ER reduction is more significant.

Table 11 – Gary's ATMR designs data.

Circuit	gary	Area	Original	Over	Under	TMR	ATMR	Overhead
Inputs	15		1428	1154	602	4284	3184	122.97%
Outputs	11	Error	Original	Errors	ER	ATMR	Errors	ER
Time (ms)	13.455	Rate	20000	2906	14.53%	20000	1358	6,79%

(a)

Circuit	gary*	Area	Original	Over	Under	TMR	ATMR	Overhead
Inputs	15		1428	1428	602	4284	3458	142.16%
Outputs	11	Error	Original	Errors	ER	ATMR	Errors	ER
Time (ms)	13.567	Rate	20000	2906	14.53%	20000	733	3.67%

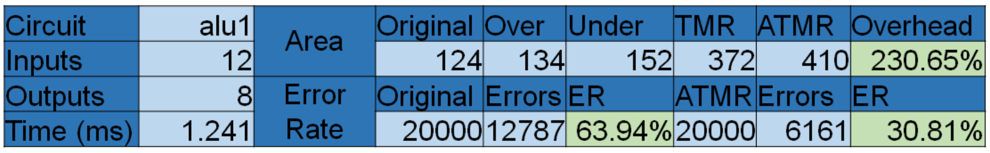
(b)

The alu1 circuit has both approximate functions larger than the original function Fig. 12. The alu1 circuit has 12 inputs and 8 outputs, and the generated ATMR (a) has 230.65% area overhead and 30.81 % ER, which is 46.73% of the ER of the original circuit. The over-approximate module has 108.06% of the original circuit area and the under-approximate module has 122.58%. A second version was created as a TMR, with three original modules because both approximate modules have a larger area than the original function. The second ATMR design (b) has 200% area overhead and 17.9% ER, which is 27.99% of the original circuit ER. The ER of the second design is reduced to 58.10% of the first design ER. Also, the area overhead was reduced by 13.29% when compared with the first design. The second design is a better design because it has lower area overhead and ER.

The area overhead values are compared with (ALBANDES et al., 2019) in Fig. 27. The values obtained by the proposed method are used to normalize the y axis at 1 and compare them with the values of the smallest and the biggest designs generated by (ALBANDES et al., 2019). The area overhead presented by the proposed method is always between smallest and biggest. In most cases it is close to the middle, but in some cases like the intb and misex3 circuits, the generated ATMR has an area overhead close to smallest. However, in circuit max1024 the area overhead is close to biggest.

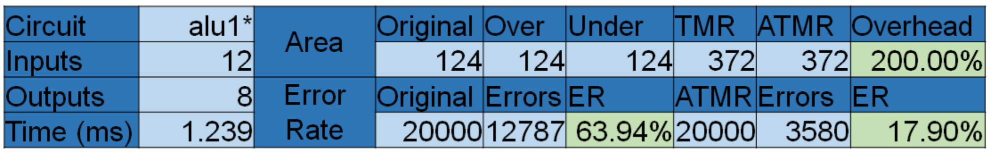
The obtained ER values are compared with (ALBANDES et al., 2019) in Fig. 28. The values obtained by the proposed method are used to normalize the y axis at 1 and compare them with the values of the smallest and the biggest designs generated by (ALBANDES et al., 2019). The ER of the generated designs are close to the ER

Table 12 – Alu1's ATMR design data.



Circuit	alu1	Area	Original	Over	Under	TMR	ATMR	Overhead
Inputs	12		124	134	152	372	410	230.65%
Outputs	8	Error Rate	Original	Errors	ER	ATMR	Errors	ER
Time (ms)	1.241		20000	12787	63.94%	20000	6161	30.81%

(a)



Circuit	alu1*	Area	Original	Over	Under	TMR	ATMR	Overhead
Inputs	12		124	124	124	372	372	200.00%
Outputs	8	Error Rate	Original	Errors	ER	ATMR	Errors	ER
Time (ms)	1.239		20000	12787	63.94%	20000	3580	17.90%

(b)

obtained by smallest, which is a loss for the proposed method, because smallest has smaller area overhead ending up with a better tradeoff of these two metrics. The biggest designs are more reliable than the generated designs and some of biggest designs are more than 3 orders of magnitude more reliable than the generated designs.

The obtained computational costs are compared with (ALBANDES et al., 2019) in Table 13. The first column shows the circuit, the second and third columns present the input and outputs data. The fourth and fifth column present the computational costs of the proposed method and (ALBANDES et al., 2019) computational cost to generate approximate modules for the same circuits. It is possible to see that the proposed method generates the ATMR functions up to 7 orders of magnitude faster on average. The next three columns show the data of the area obtained by the developed method and the biggest and smallest circuits produced by (ALBANDES et al., 2019). The area of the developed method is always between the two values. The last three columns show the ER comparison. First it presents the ER of the proposed method and the ER of the biggest and smallest circuits produced by the other work. The ER of the proposed method is worse than the ER obtained by (ALBANDES et al., 2019) in most cases and need improvements in future works.

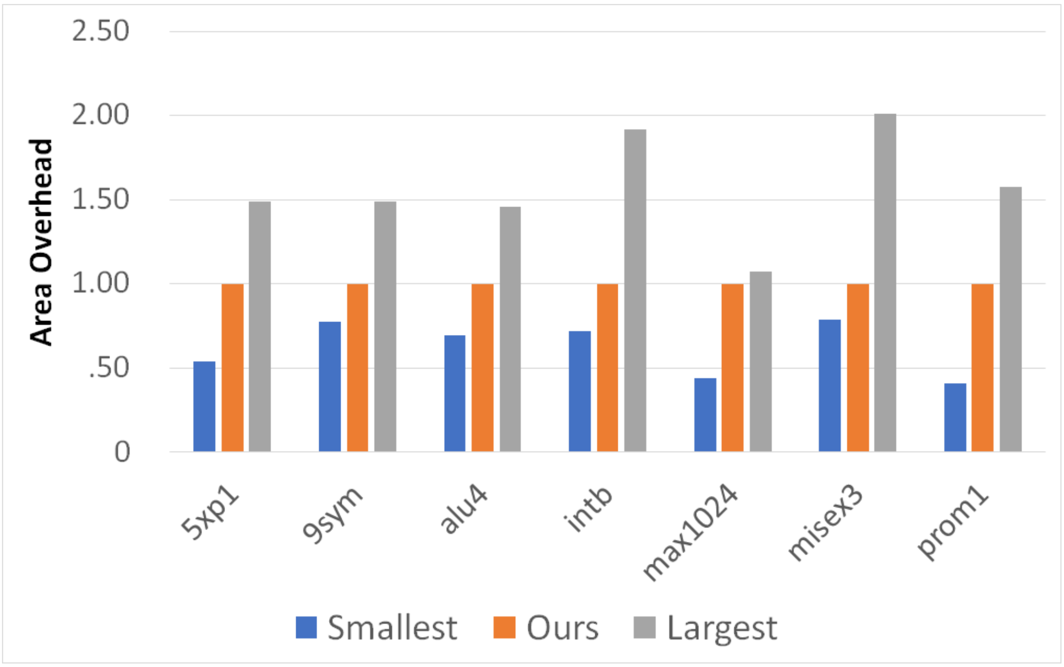


Figure 27 – Normalized Area Overhead. The area obtained by the proposed method is set as 1 and it is compared with (ALBANDES et al., 2019) smallest and biggest designs.

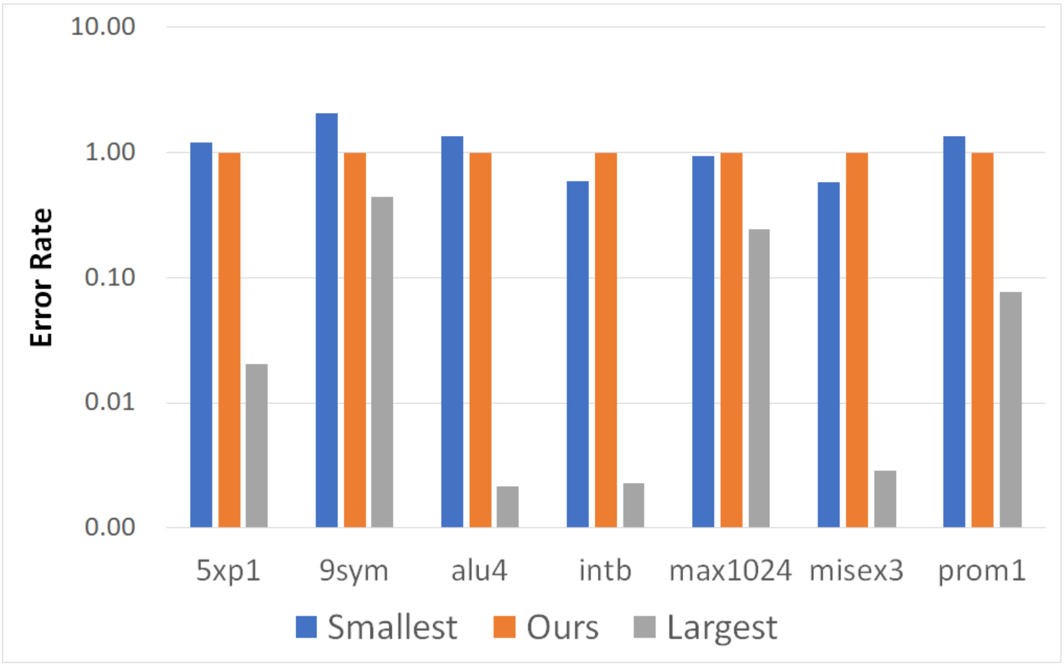


Figure 28 – Normalized Error Rate. The ER obtained by the proposed method is set as 1 and it is compared with (ALBANDES et al., 2019) smallest and biggest designs.

Table 13 – Area overhead, ER and computational cost comparison with (ALBANDES et al., 2019)

Bench			Time		Area			Error Rate		
Circuit	Inputs	Outputs	Ours	Other work	Ours	Other work		Ours	Other work	
						Biggest	Smallest		Biggest	Smallest
5xp1	7	10	0.289ms	-	118.97%	177.00%	64.00%	19.67%	0.40%	23.90%
9sym	9	1	0.094ms	-	103.36%	154.00%	80.00%	2.95%	1.30%	6.10%
alu4	14	8	5.403ms	4h45m	118.22%	172.00%	82.00%	4.67%	0.01%	6.30%
intb	15	7	9.110ms	1h50m	91.76%	176.00%	66.00%	4.43%	0.01%	2.60%
max1024	10	6	0.362ms	3m	158.52%	170.00%	70.00%	5.37%	0.30%	5.00%
misex3	14	14	9.223ms	6h30m	86.02%	173.00%	68.00%	6.99%	0.02%	4.02%
prom1	9	40	1.628ms	35m	107.98%	170.00%	44.00%	6.49%	0.50%	8.80%

6 CONCLUSIONS

The correlation between an input variable and the output of the circuit was analyzed. It showed that the input variable that presents the highest correlation is the best candidate to generate the approximate functions for the ATMR architecture in the proposed method. This variable adds the least amount of unprotected vectors and usually has the smallest area value.

The proposed method is not flexible with the tradeoff between area overhead and ER. The method receives an input and just returns the ATMR design that was generated using the variable with the highest correlation. This needs to be addressed by future works because the tradeoff flexibility is an important aspect when generating approximate modules for an ATMR design. The tradeoff flexibility helps the method to generate designs that meet the project constraints most of the time.

The generated modules do not have the best area overhead and ER values. Therefore, the method needs to improve these metrics with no big penalties in the computational cost to generate the modules. At the same time, the method is fast and some modules have a large area reduction, so it can be used combined with other solutions.

The low computational cost of generating the modules for ATMR designs is key to turning feasible using ATMR in complex designs. The proposed method generates ATMR designs with a low computational cost. All the circuits were approximated with a low computational cost, with the largest circuit taking only 215ms to be approximated. The method generates ATMR 7 orders of magnitude faster on average when compared to (ALBANDES et al., 2019). Therefore, the proposed method has higher scalability capable of embracing higher complexity designs.

6.1 Future Works

In future works, we plan to improve the technique to keep computational cost over control while allowing a better tradeoff between area overhead and ER. The tradeoff flexibility is another key aspect that the method does not have yet. Other interesting concepts showed up in the research. Some of the future works to improve this method are:

- Improve area overhead and ER values
- Approximate every X lines in the truth table. This way, the method can have the flexibility aspect.
- Find a way to apply the correlation concept directly in the PLA file. This would make approximate bigger circuits possible.

The developed method presents great results in the computational cost, but the area overhead and ER values usually are worse than the results presented by other methods. It is important to optimize these two metrics when designing an ATMR circuit. Therefore, it is crucial to improve the area overhead and ER values in future works. However, the low computational cost cannot be heavily penalized for these improvements.

Another aspect that needs improvement in the developed method is the results flexibility. The method currently delivers a single solution at the end of the execution. However, different projects may require different optimizations to meet their constraints. A possible solution would be to divide the truth table into X parts and check the correlation of the input variables with the output for every part. This division would allow two different approaches to flexibilize the results. First, distinct values of X could be used to create multiple ATMR designs, creating one design for each value of X. Another alternative would be to set a value for X and use a different amount of parts to generate the approximations.

The main impediment to running the method for bigger circuits is the data structure. The approximate function generation step creates PLA files with all the lines of the truth table, and these files become exponentially bigger with bigger circuits. Applying the correlation concept directly in the input file could make it possible to generate approximate modules for more complex circuits. This improvement would make the method becomes more scalable.

Other future works do not depend on the proposed method. It can be used other techniques to generate the modules and a different amount of approximate modules. Some of these future works are:

- Test more circuits/benchmarks
- Propose different techniques to generate the approximate functions. For instance, using artificial intelligence.
- Explore the FATMR, QAMR, and other different architectures.

A more scalable method allows further testing, using different benchmarks and more complex circuits. This would make the results presented by the developed method more reliable. Analyzing the results obtained by testing different benchmarks, it would be possible to understand how the method would work for different types of applications. Also, it would be easier to compare the results with other methods, because it would test the circuits that are used to test other methods.

Other techniques to generate approximate functions for ATMR designs can be tested. For instance, artificial intelligence can be used to generate approximate functions. The use of artificial intelligence would be challenging because it would need a careful selection of parameters and hyper-parameters to optimize the results, test sets to avoid over-fitting, choice of models, and other decisions required by artificial intelligence techniques.

Different architectures can be tested as well. FATMR can produce designs with lower area overhead, but it is harder to meet the requirements. The proposed method can be improved to approximate the module that stays the same as the original function. This process needs to be done in a way that the voter has the correct output in the absence of faults. The QAMR is a good option to reduce the area overhead without losing the 100% fault coverage rate.

REFERENCES

ALBANDES, I.; AL. et. Design of approximate-TMR using approximate library and heuristic approaches. **Microelectronics Reliability**, [S.l.], v.88-90, p.898–902, 2018.

ALBANDES, I. et al. Improving approximate-TMR using multi-objective optimization genetic algorithm. In: LATIN-AMERICAN TEST SYMPOSIUM (LATS), 2018. **Anais. . .** [S.l.: s.n.], 2018. p.1–6.

ALBANDES, I.; MARTINS, M.; CUENCA-ASENSI, S.; KASTENSMIDT, F. Building ATMR circuits using approximate library and heuristic approaches. **Microelectronics Reliability**, [S.l.], v.97, p.24–30, 06 2019.

APONTE-MORENO, A.; MONCADA, A.; RESTREPO-CALLE, F.; PEDRAZA, C. A review of approximate computing techniques towards fault mitigation in HW/SW systems. In: IEEE 19TH LATIN-AMERICAN TEST SYMPOSIUM (LATS), 2018., 2018. **Anais. . .** [S.l.: s.n.], 2018. p.1–6.

APPLE, N. **Apple unveils M1 Ultra, the world's most powerful chip for a personal computer**. https://nr.apple.com/d2l1v3s8D5.

ARIFEEN, T.; HASSAN, A. S.; LEE, J.-A. Approximate triple modular redundancy: A survey. **IEEE Access**, [S.l.], v.8, p.139851–139867, 2020.

ARIFEEN, T.; HASSAN, A. S.; MORADIAN, H.; LEE, J. A. Input vulnerability-aware approximate triple modular redundancy: Higher fault coverage, improved search space, and reduced area overhead. **Electronics Letters**, [S.l.], v.54, n.15, p.934–936, 2018.

ARIFEEN, T. et al. Probing Approximate TMR in Error Resilient Applications for Better Design Tradeoffs. In: EUROMICRO CONF. ON DIGITAL SYSTEM DESIGN (DSD), 2016. **Anais. . .** [S.l.: s.n.], 2016. p.637–640.

BAHARVAND, F.; MIREMADI, S. G. LEXACT: Low energy N-modular redundancy using approximate computing for real-time multicore processors. **IEEE Transactions on Emerging Topics in Computing**, [S.l.], v.8, n.2, p.431–441, 2017.

BOSIO, A. et al. Exploiting Approximate Computing for implementing Low Cost Fault Tolerance Mechanisms. In: DESIGN & TECHNOLOGY OF INTEGRATED SYSTEMS IN NANOSCALE ERA (DTIS), 2020., 2020. **Anais...** [S.l.: s.n.], 2020. p.1–2.

BRAYTON, R.; MISHCHENKO, A. ABC: An academic industrial-strength verification tool. In: INTERNATIONAL CONFERENCE ON COMPUTER AIDED VERIFICATION, 2010. **Anais...** [S.l.: s.n.], 2010. p.24–40.

CHOUDHURY, M. R.; MOHANRAM, K. Approximate logic circuits for low overhead, non-intrusive concurrent error detection. In: DESIGN, AUTOMATION AND TEST IN EUROPE, 2008. **Proceedings...** [S.l.: s.n.], 2008. p.903–908.

DEVEAUTOUR, B.; TRAIOLA, M.; VIRAZEL, A.; GIRARD, P. Qamr: an approximation-based fully reliable tmr alternative for area overhead reduction. In: IEEE EUROPEAN TEST SYMPOSIUM (ETS), 2020., 2020. **Anais...** [S.l.: s.n.], 2020. p.1–6.

FARIAS, C. R. **Procedimento para estimativa da susceptibilidade à radiação de circuitos combinacionais**. 2022. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal do Rio Grande, Grupo de Sistemas Digitais e Embarcados. Programa de Pós-Graduação em Computação., Rio Grande.

GOMES, I. A. C. **Uso de redundância modular tripla aproximada para tolerância a falhas em circuitos digitais**. 2014. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal do Rio Grande do Sul, Instituto de Informática. Programa de Pós-Graduação em Computação., Porto Alegre.

GOMES, I. A. C. **Use of approximate triple modular redundancy for fault tolerance in digital circuits**. 2018. Tese (Doutorado em Ciência da Computação) — Universidade Federal do Rio Grande do Sul, Instituto de Informática. Programa de Pós-Graduação em Microeletrônica., Porto Alegre.

GOMES, I. A.; KASTENSMIDT, F. G. Reducing TMR overhead by combining approximate circuit, transistor topology and input permutation approaches. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI), 2013. **Anais...** [S.l.: s.n.], 2013. p.1–6.

GOMES, I. A. et al. Methodology for achieving best trade-off of area and fault masking coverage in ATMR. In: LATIN AMERICAN TEST WORKSHOP-LATW, 2014. **Anais...** [S.l.: s.n.], 2014. p.1–6.

GOMES, I. A. et al. Using only redundant modules with approximate logic to reduce drastically area overhead in TMR. In: LATIN-AMERICAN TEST SYMPOSIUM (LATS), 2015. **Anais...** [S.l.: s.n.], 2015. p.1–6.

GOMES, I. A. et al. Exploring the use of approximate TMR to mask transient faults in logic with low area overhead. **Microelectronics Reliability**, [S.l.], v.55, n.9-10, p.2072–2076, 2015.

HASSAN, A.; ARIFEEN, T.; MORADIAN SARDROUDI, H.; LEE, J. A. Generation Methodology for Good-Enough Approximate Modules of ATMR. **Journal of Electronic Testing**, [S.l.], v.34, 12 2018.

ICHIHARA, H.; INAOKA, T.; IWAGAKI, T.; INOUE, T. Logic simplification by minterm complement for error tolerant application. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER DESIGN (ICCD), 2015., 2015. **Anais. . .** [S.l.: s.n.], 2015. p.94–100.

MOHANRAM, K.; TOUBA, N. A. Partial error masking to reduce soft error failure rate in logic circuits. In: IEEE SYMPOSIUM ON DEFECT AND FAULT TOLERANCE IN VLSI SYSTEMS, 18., 2003. **Proceedings. . .** [S.l.: s.n.], 2003. p.433–440.

MUNTEANU, D.; AUTRAN, J.-L. Modeling and simulation of single-event effects in digital devices and ICs. **IEEE Transactions on Nuclear science**, [S.l.], v.55, n.4, p.1854–1878, 2008.

RADAMSON, H. H. et al. Miniaturization of CMOS. **Micromachines**, [S.l.], v.10, n.5, p.293, 2019.

RUDELL, R.; SANGIOVANNI-VINCENTELLI, A. Exact minimization of multiple-valued functions for PLA optimization. In: **The Best of ICCAD**. [S.l.]: Springer, 2003. p.205–216.

SANCHEZ-CLEMENTE, A.; ENTRENA, L.; GARCÍA-VALDERAS, M.; LÓPEZ-ONGIL, C. Logic masking for SET mitigation using approximate logic circuits. In: IEEE 18TH INTERNATIONAL ON-LINE TESTING SYMPOSIUM (IOLTS), 2012., 2012. **Anais. . .** [S.l.: s.n.], 2012. p.176–181.

VON NEUMANN, J. Probabilistic logics and the synthesis of reliable organisms from unreliable components. **Automata Studies: Annals of Mathematics Studies. Number 34**, [S.l.], n.34, p.43, 1956.