ON THE ROBUSTNESS OF BEHAVIOURAL CIRCUIT DESIGN: FROM FAULT-TOLERANCE TO HARDWARE SECURITY

ANJANA BALACHANDRAN

PhD

The Hong Kong Polytechnic University
2021

The Hong Kong Polytechnic University

Department of Electronic and Information Engineering

On the Robustness of Behavioural Circuit Design: From Fault-Tolerance to Hardware Security

ANJANA BALACHANDRAN

A thesis submitted in partial fulfilment of the

requirements for the degree of

Doctor of Philosophy

July, 2020

Certificate of Originality

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

———————————————- (signed)

ANJANA BALACHANDRAN (Name of student)

Abstract

The continuous demand for electronic products is pushing the semiconductor industry to deliver new and more advanced products at shorter time frames. These rapid developments have partially been made possible with the continuous shrinking of transistors, also known as Moore's law. One of the problems with smaller transistors is that they are also more susceptible to transient errors. These errors can affect the correct operation of electronic equipment, and hence, their reliability. Especially, transient errors could turn out to be fatal in safety-critical applications. Thus, it is natural to expect a higher level of reliability in such safety-critical systems.

Reliability here refers to the systems' ability to be available during adverse conditions. Factors that prevent these systems from being available can be unintentional or intentional. Unintentional factors that induce a fault include atmospheric radiations, harsh environmental conditions or energized particles. Intentional factors are mainly due to the fact that many Integrated Circuit (IC) design companies these days are fabless and they rely on offshore foundries to manufacture their chips. These ICs could be maliciously altered by rogue elements within the design and manufacturing chain, preventing such systems from functioning correctly. Moreover, complex System-on-chips (SoCs) now include a wealth of information that attackers try to extract to profit economically. This ranges from stealing the Intellectual Property (IP) of the designs to secret encryption keys stored in the SoCs. Thus, it is important that IC design seamlessly integrates fault tolerance techniques to mitigate faults induced due to unintentional factors with hardware security as a new design parameter.

The main fault tolerance approaches have relied on building N-Modular Redundant system (NMR) mainly at the Register Transfer Level (RTL). With the increase in logic density, modern ICs are now complex System-on-Chips (SoCs). Thus, new design methodologies have been developed to move up the level of Very Large Scale Integration (VLSI) design abstraction from the RTL to the behavioural level. Thus, it is crucial to revisit the problem of fault-tolerant VLSI design for SoCs designed using higher-levels of abstractions.

Moreover, hardware security primarily relies on logic locking techniques including key

gate insertions at the RT-level or gate level. Modern High-Level Synthesis (HLS) tools allow to more effectively lock behavioural IPs and make them more robust against several types of attacks. Thus, revisiting hardware security using HLS could open new paths towards designing secure ICs.

This thesis deals with these important issues and proposes techniques to make behavioural SoCs more robust against soft errors and protect these SoCs from malicious alterations or IP theft through functional locking. The main goal is to address these important issues by raising the level of VLSI design abstraction from the RT-level to the behavioural level.

Acknowledgements

As I approach the end of my doctoral studies, I would like to take this opportunity to thank people who have played a vital role in helping me complete this roller-coaster journey. First and foremost, I would like to thank my supervisor Dr. Benjamin Carrion Schafer from the bottom of my heart for his constant support and guidance at both my high and low points during this journey. I am indebted to Dr. Schafer for the time and efforts he has invested in guiding me despite the distance and time zones between us. Once again thank you Dr. Schafer for making this journey of my life count and for inspiring me with your knowledge and patience.

I would also like to take this opportunity to thank my chief supervisor, Prof. Francis C.M. Lau for being the strength at times of adversities and helping me in all possible ways. I would like to express my deep gratitude to Prof. Lau for taking time out of his busy schedule to discuss my research progress. Thank you once again, Prof. Lau for being the light at the end of the tunnel.

I wish to thank Dr. Farah Naz Taher for being a great collaborator and friend. I am glad that I did not have to travel alone throughout this journey because I had great colleagues for which I would like to thank Anushree and Nandeesh for being my support system all these years. I am truly grateful to my friends Tanuja, Sheena, Seema and Hayley for constantly motivating me and helping me stay focused.

I would like to thank my family though I still believe thanking them would not be enough to explain my gratitude towards them. I am extremely thankful to my parents for all the sacrifices they have done for giving me the best in life. I would like to take this opportunity to thank my mother for being my role model and teaching me endurance in life. This journey would have never started if not for her. Life is short as my father always says “No time to lose”. Thanks to my father for continually bringing me back on track and running the race along with me. I would also like to thank my brother for his love and affection. I would also like to thank my four-legged siblings Sheldon and Ginger, for being an emotional support system all my life. Finally, I would like to thank my husband Manu

for his inevitable love and support. He has been my greatest strength, and I am indebted to him for life for all the sacrifices he has made for us.

To my family.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ALAP	As Late As Possible
ASAP	As Soon As Possible
ASIC	Application Specific Integrated Circuit
BDL	Behavioural Descriptive Language
BIP	Behavioural Intellectual Property
CDFG	Control Data Flow Graph
CED	Concurrent Error Detection
CF	Control Flow
CWB	CyberWorkBench
CMF	Common Mode Failure
DF	Data Flow
DFS	Depth First Search
DIMP	DIversity Metric based on circuit Path analysis
DIP	Distinguishing Input Pattern
DSE	Design Space Exploration
DWC	Duplicate With Compare
EMI	Electromagnetic Interference
FI	Fault Injection
FIR	Finite Impulse Response
FLL	Fault analysis-based Logic Locking
FM	Fault Model
FPGA	Field Programmable Gate Array
FRU	Field Replaceable Unit
FS	Fault Simulation

FSM	Finite State Machine
FU	Functional Unit
GA	Genetic Algorithm
HDL	Hardware Description Language
HLS	High Level Synthesis
HS	Hardware Security
HW	Hardware
HWAcc	Hardware Accelerator
IC	Integrated Circuit
IP	Intellectual Property
LET	Linear Energy Transfer
LUT	Look Up Table
MAE	Mean Absolute Error
MBU	Multi Bit Upset
MPSoC	Multiprocessor System-on-chip
NMR	N Module Redundancy
NoC	Network on Chip
PFC	Primary Flight Control
RAM	Random Access Memory
RLL	Random Logic Locking
ROM	Read-Only Memory
RTL	Register Transfer Level
SAT	SATisfiability
SEFI	Single Event Functional Interrupt
SEU	Single Event Upset
SDC	System of Difference Constraints
SoC	System-on-Chip

SLL	Strong-interference based Logic Locking
TMR	Triple Modular Redundancy
UART	Universal Asynchronous Receiver Transmitter
VHDL	Very High Speed IC Hardware Description Language
VLSI	Very Large Scale Integration

Chapter 1

Introduction

A product's success in any market depends on several factors that revolve around customer expectations, which vary between affordability and dependability. In many ICs it is often essential to focus on dependability rather than affordability mainly because of the applications they are used in. The threshold of dependability may vary based on the industry where the product will be used. In some industries like avionics and aerospace, the threshold is very high, while in other industries, price is more important than reliability. Some examples include toys, calculators and watches where a failure naturally would not lead to catastrophic events. Still, reliability is one of the deciding factors in many industries and should be addressed in all cases as it seriously impacts the brand image if not taken into account.

The term reliability in this context refers to a system's availability during adverse conditions that could prevent it from functioning normally. A failure in an electronic system can lead to catastrophic events leading to the loss of human lives and large economic losses. There exist several factors that prevent a system from being available. They could be due to *unintentional* interference such as radiation-induced faults or harsh environmental conditions, or due to *intentional* interference caused by third party actions such as malicious alterations on an IP, IP theft, overproduction or

unauthorized use. The existing techniques used to mitigate the former and the latter are fault tolerance and hardware security, respectively. This thesis investigates novel techniques to mitigate such radiation-induced faults and threats posed by intentional alterations of IPs when moving to a higher VLSI design abstraction level. Thus, this work revisits both fault tolerance and hardware security when using HLS.

There are many precursors that could lead to faults in an electronic system. In this thesis, we mainly deal with single-even upsets (SEUs) due to cosmic particles, high-energy alpha particles, electromagnetic interference (EMI) or power-supply disturbances [53]. Moreover, with continuous scaling of transistors, ICs have become more susceptible to SEUs. Therefore, it is imperative to build reliability into these electronic systems' design methodology such that catastrophic malfunctioning of electronic systems never happens.

The main problem is that it is virtually impossible to guarantee that a design or system is completely reliable. Thus, often they are designed to guarantee a minimum quality service even in the presence of errors. The ability of a system to work even in the presence of an error is called *fault tolerance* [42] [30].

Traditional fault-tolerant systems are based around the concept of replicating the same module N times, also called the N-modular redundancy (NMR) technique. Out of all NMR systems, two are the most popular. The first, where two copies of the same module performing the same function are used along with a comparator that compares the results and issues a warning signal if the outputs do not match, is called duplication with comparison (DWC). DWC can capture errors within a module, but cannot mask the error. Thus, the system needs to be re-initialized from a well-known valid previous state when an error is detected. The second approach mask errors by instantiating three modules in parallel, also called triple modular redundancy (TMR). In TMR, the voter chooses the results based on the majority of outputs received. This can completely mask an error. The obvious drawback is the area

overhead associated with TMR systems, which is over 200% the area of single module systems (two additional modules and the voter are needed).

One weakness of DWC and TMR systems is that they cannot protect against Common-Mode Failures (CMFs). A CMF is a result of failure induced in more than one module simultaneously [40]. A formal definition of a CMF is given below:

"A common-mode failure (CMF) is the result of an event(s) which, because of dependencies, causes a coincidence of failure states of components in two or more separate channels of a redundancy system, leading to the defined system failing to perform its intended function." [50]

An event that triggers a CMF could be unintentional (EMI, power-supply disturbances or radiation effects) or intentional (e.g., Hardware Trojans). Design mistakes could also be a reason for a CMF as both the modules possess identical design faults. In an effort to reduce CMFs, the concept of diversity was introduced. For example, the Boeing 777 has three processor architectures designed by three different companies Intel, AMD and Motorola for its primary flight computer (PFC) [100]. Another way to achieve diversity is by involving two separate teams to implement the function using different resources as done in space applications.

This is extremely costly and only possible for very few domains. Therefore, it is highly recommended to have automatic ways to create diverse systems. Thus, part of this thesis deals with the automatic generation of diverse N-modular redundancy systems such that CMFs can be detected.

Securing hardware is also an important factor because nowadays, most of the IC design companies are fabless. These companies sometimes contract third party companies for the IC design back-end and contract foundries offshore to manufacture their chips. This makes such companies vulnerable to other companies making malicious alterations that would eventually cause system failure when a specific trigger condition is reached. Such critical alterations if made in safety-critical applications

could cause not only catastrophic damages but also irreversible loss of trust in the company.

Functional locking [73] is one among the commonly used techniques to protect ICs. The main idea behind is to add extra gates called key gates into an existing design in order to prevent it from working normally when an incorrect key is applied to those gates. Some of the drawbacks of the existing techniques that insert these locks at gate-level or RT-level are timing closure issues and third parties' ability to detect the inserted locks and, hence, make them prone to removal attacks. Limiting factors like these act as a catalyst to try and improve the existing techniques. Thus, the last part of this thesis introduces a new locking mechanism that minimises these limitations by inserting the locking mechanism at the behavioural level.

1.1 Contributions of This Thesis

Fig. 1.1 shows a pictorial view of the complete thesis using a heterogeneous SoC as an example. The three main contributions are highlighted in three different hardware accelerators, as the techniques developed mainly apply to them. In particular:

1. The first contribution presents a method to exploit the time that hardware accelerators are not used in shared bus SoCs to re-compute their results (redundancy in time). This work uses cycle-accurate simulation models of complete SoCs to accurately extract each hardware accelerator's timing slack that in turn allows understanding the amount of time each hardware accelerator is idle to recompute its task.

2. The second contribution investigates if the diversity between two RTL descriptions are coherent, with diversity between its corresponding gate netlists to help accelerate finding functional equivalent design pairs that are most diverse. This contribution also introduces the use of predictive models to estimate the

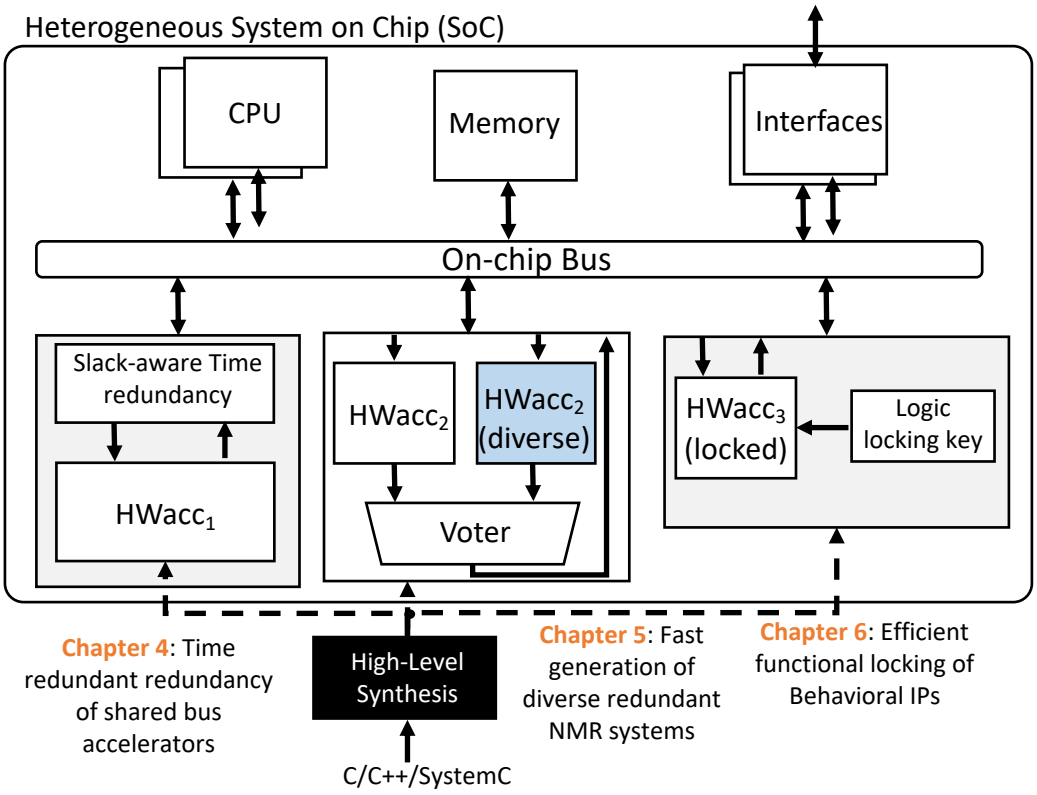

Figure 1.1: Overview of thesis contributions.

diversity between two functionally equivalent RTL implementations to avoid having to do full stuck-at-fault pair simulation at the gate netlist to find diverse hardware implementations.

3. The final contribution presents an efficient way to lock hardware accelerators given as behavioural descriptions for HLS. This method makes use of the way that HLS synthesizes behavioural descriptions in the form of Finite State Machine (FSM) + datapath to increase the complexity of finding the correct key by splitting the key by FSM states. The attacker thus has to try all possible permutations. Moreover, this task inserts locking primitives in each HLS control step based on the timing slack reported by the HLS tool, avoiding any timing closure issues.

1.2 Thesis Outline

Chapter 2: This chapter briefly explains the background of fault tolerance. This will help to better appreciate the need for a more reliable and sustainable system. *Chapter 3:* This chapter describes how C-based VLSI design methodologies work and in particular HLS, highlighting its main advantages over traditional RT-Level based VLSI design methodologies. *Chapter 4:* This chapter proposes an efficient fault tolerance strategy for complex MPSoCs by making use of the normally unused slack times between HWAcc computations. *Chapter 5:* This chapter focuses on spatial redundancy and proposes a C-based design technique that helps identify design pairs with greater diversity to build fault tolerant system against CMFs. The later part of the chapter introduces a predictive model to estimate the diversity index between two designs. *Chapter 6:* This chapter proposes a functional locking technique that makes use of HLS synthesis capabilities to lock hardware accelerators described at higher abstraction level, and avoids them from being misused along the supply chain. *Chapter 7:* This chapter discusses the conclusion and future directions.

Chapter 2

Fault Tolerance

This chapter introduces basic concepts around fault tolerance that will help better appreciate this thesis' contributions. It will primarily focus on describing the main steps that might lead to a failure in an electronic system and typical mechanisms proposed to avoid the same failure.

2.1 Introduction

Shrinking geometries, lower operating voltages, higher operating frequencies and higher density circuits have led to an increased sensitivity to soft errors. These soft errors occur when a radiation event causes a disturbance large enough to reverse a bit in the circuit. When the bit is flipped in a critical control register or configuration memory, e.g., in a Field-Programmable Gate Array (FPGA), it can cause the circuit to malfunction. At the same time, current SoCs are getting more complex and typically include multiple processors, memories and especially a variety of HWAccs. These HWAccs are extremely important as the breakdown of Dennard's scaling, and the continuation of Moore's law implies that power densities are reaching nuclear reactor levels. The term dark silicon has been coined to refer to the part of an IC which has to be kept dark (off) due to this increase in power densities [20]. The

solution to this problem is to adapt the architecture to the application through pluralization and customization using HWAccs.

Fault tolerance can be described as the ability of a system to continue working after the occurrence of a fault. Because of the ubiquitous nature of electronic systems based around ICs, it is extremely important to seamlessly integrate fault tolerance into the VLSI design process. In cost-sensitive products, these fault-tolerant systems should not penalize the cost of an IC, while for safety-critical applications, fault tolerance is a must, as not having a reliable system could have catastrophic consequences. Most prior works on VLSI hardware reliability make use of module redundancy, assuming that each module is exactly the same. Multiple module replicas implementing the same logic function are executed in different hardware channels. A voting scheme detects an output mismatch. Another approach makes use of time redundancy by recomputing the result using the same hardware channel. The advantage of the latter approach is that virtually no hardware overhead is required compared to the spatial redundancy case where the system requires 100% to 200% area overheads for the dual and triple modular redundancy case.

The end goal of a fault tolerant system is to develop a dependable design that can deliver its intended level of services to its users even at adverse situations such as module failure.

2.2 Fault to Failure

A *fault* is an unexpected event that can cause the hardware to behave differently than expected. Fault can be permanent or transient. In the case of permanent faults, the symptom is kept over the IC's lifetime, while transient fault manifests only during a short period of time. Some examples of permanent faults are wires that break within

the chip because of causes such as electromigration. A typical example of transient fault is SEU caused by radiations that result in single bit-flips.

Faults can lead to errors which in turn can lead to the failure of the system, but are often also masked away. It is therefore important to understand the relationship between faults, errors and failures.

Fault: An unexpected event that causes the hardware circuit to be different than its originally intended design.

Error: The effect of a fault occurrence that denies the system from delivering the correct information.

Failure: The non-performance of a system where it fails to work as expected due to its underlying errors.

A fault triggers an error that causes a system failure, but not all faults induce a system failure. This explains that some faults could be dormant for a long time while some could be identified immediately, i.e. some errors are reflected at the output while the others are not. Fig. 2.1 shows the transition of a working system from fault occurrence to failure.

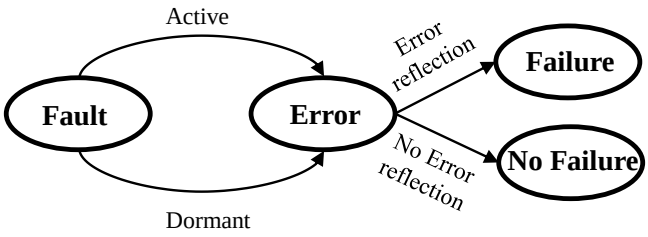

Figure 2.1: Fault to failure.

Faults happen due to several reasons such as incorrect specification, implementation, or fabrication of the design. Apart from these causes there are other critical external factors such as environmental disruptions or human actions that could be intentional or unintentional. Thus, the sources of faults could be grouped into four

types: incorrect specification, incorrect implementation, fabrication defects, and external factors [21].

1: Incorrect specification results from a wrong specification given as an algorithm, architecture or a customer requirement. These faults may occur due to certain aspects being left out while designing, such as operating conditions of the IC and so on. A system with these faults could work flawlessly most of the time while failing at specific operating conditions. These are called specification faults.

2: Incorrect implementation is when an implementation does not realize the function given in the specification. These are called design faults that arise from poor component selection, logic mistakes, poor synchronization between blocks or poor timing closure.

3: Fabrication defects are results of manufacturing imperfections, component wear-outs or device defects. In early days fabrication defects were an important reason for device failure, thus introducing fault tolerance while now the advancements in semiconductor technologies have drastically reduced fabrication defects.

4: External Factors are due to particle strikes that flip bits within the electronic component. These are often called single event upsets.

In this thesis, we extensively focus on mitigating faults triggered by the fourth category of fault triggers that are the external factors.

2.3 Effect of External Factors on Semiconductors

Radiation-induced soft errors are the biggest threat posed on semiconductors. Commercial digital electronic components face the major threat of failure due to radiation-induced soft errors. There are a number of radiations present in the Earth's atmosphere that are strong enough to cause data disruptions or even permanent failures on digital electronic components [4]. When a bit flips in a system

it is called data disruption. Some of them do not cause permanent damage which means that the device will return back to its normal state. These errors are called "soft-errors". Certain radiations cause irreversible damage thus making permanent changes on to the system and hence called "hard-errors" [4]. When the radiation flips a single bit, it is called SEU. If the radiation is of higher energy, it is capable of flipping a number of bits. This multiple bit flip is called multi bit upset (MBU).

Devices like FPGAs and DRAMs have control registers which are critical for their functioning. The effect of radiation can make permanent changes on the control circuitry. This event occurrence is termed as Single Event Functional Interrupt (SEFI) as it interrupts the system from functioning properly causing a significant threat to reliability [41]. Memory errors opposed to these events are of less impact as some of the errors do not get triggered or do not produce a wrong output.

Air-crafts and space-crafts are applications that are subjected to a significant amount of radiations. The electronic components here are prone to soft errors due to cosmic rays, alpha rays and other particles with energy that happen more often higher in the atmosphere. The magnitude of disturbance that an ion could cause depends on the Linear Energy Transfer (LET). In a silicon substrate, for every 3.6eV energy loss, an electron hole pair is produced. The energy lost by an ion depends on the material it traverses through and on the particle energy. When the particle is of high energy and traverses through a dense material, it loses more energy.

Fig. 2.2 shows the path taken by an ion. As the ion moves through the substrate it creates a pool of electron hole pairs with a sub-micron radius with high carrier concentration (a). Later the ionization track traverses to the depletion region after which the electric field collects all the carriers resulting in a voltage/current transient

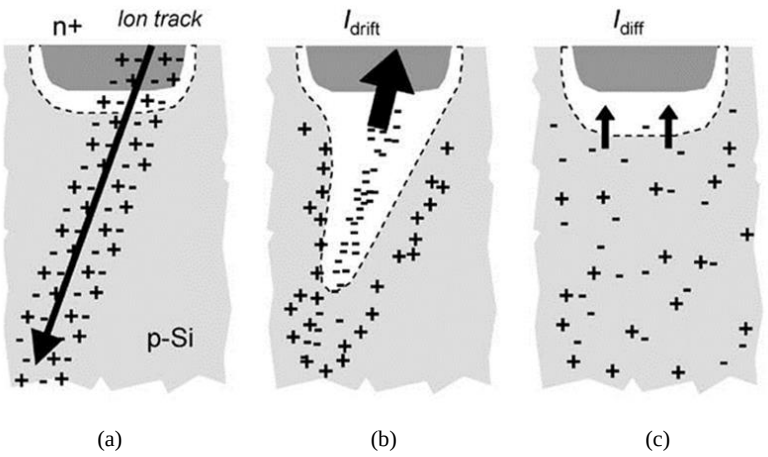

Figure 2.2: Charge generation and collection phases in a reverse-biased junction and the resultant current pulse caused by the passage of a high-energy ion [4].

at that node (b). The size of the funnel is a function of substrate doping. The funnel distortion increases for decreased substrate doping. This "prompt" collection phase is completed within a nanosecond and is followed by a phase where diffusion begins to dominate the collection process (c). The additional charge is collected as electrons diffuse into the depletion region on a longer time scale (hundreds of nanoseconds) until all excess carriers have been collected, recombined, or diffused away from the junction area. The corresponding current pulse resulting from these three phases is also shown in Fig. 2.2. If the energy lost by an ion is less than the critical charge called Q_{crit}, it does not create an SEU. But when Q_{coll}, which is the collective energy lost, is greater than Q_{crit}, SEU or MBU occurs. Several techniques used to overcome these radiation effects are discussed below. The work carried out clearly explains a new technique to speed up the process while being able to reduce soft error occurrences [4].

2.4 Fault Tolerance in Modern SoC

Heterogeneous SoCs have multiple components such as processors, memory blocks, interfaces and HWAccs. For a fault free computation, it is important that all these components are fault-tolerant but fault tolerance techniques used by each one of them varies. This subsection briefly discusses the fault tolerance strategies used in each of the typical SoC component.

2.4.1 Microprocessor

In distributed systems, checkpointing and rollback recovery techniques were used to make them fault-tolerant by bringing a system back to a consistent state after a failure has occurred [9]. Usually consistent states are saved during a system's fault free execution so that in case of a failure, a recomputation could start from this state rather than starting from the first. In [35] the shortcomings of a checkpoint and rollback recovery were discussed, including inconsistencies due to inter-process dependencies that often forced processes that did not fail to rollback. This phenomenon is called "domino effect". Authors in [93] proposed an adaptive independent checkpoint strategy that prevents such inter-dependencies. The rollback in one process does not influence the other processes in a distributed system.

The use of multiple processors has revolutionized the way computations were handled. With the use of multi-cores, software could now become parallel, at the same time it is important to make sure that the computations are unaffected by hardware errors. The tasks done in parallel have close similarities in the control-data. In [90] the authors proposed BLOCKWATCH that utilizes this similarity to check for any hardware errors during runtime.

Instruction level parallelism in modern processors were leveraged to build SWIFT [71]. SWIFT aims to duplicate instructions executed during the programs slack time

reclaiming unused resources for the computations. SWIFT was used to check for the control flow path's correctness by inserting checkers at the basic blocks. Extensive work with respect to fault tolerance in modern CPUs could be found in [59, 70, 96].

2.4.2 Memory Protection

For a system to maintain its integrity, it needs to protect its memory elements or storage units. Several techniques exist and they vary from module replication to error-correcting codes. Many error-correcting codes for building fault-tolerant memory elements have been proposed. Error-correcting code is a process that encodes a data within a much larger data set through replication so that a small level of error does not affect the credibility of the retrieved data. Using parity checker is a simple way to detect errors in memory [26] where a parity bit is added to the data string. Each time a data is requested to be retrieved from a memory unit the parity of that data is computed. An error is flagged if a single bit flip has occurred while this cannot detect multiple bit flips. It can only detect an error but cannot correct it. Hamming code was introduced in [23] to detect two-bit error or correct single bit errors. This restriction lets Hamming code to be used only when the error rate is low. A low error rate is usually observed in computer memories i.e. RAM, where bit error occurrences are rare, making Hamming code the best choice.

2.4.3 Interface

Data being transferred through peripheral interfaces are equally at the risk of being affected by transient errors. Synchronous and asynchronous interfaces alike are affected by SEUs. The universal asynchronous receiver transmitter (UART), which is a common asynchronous interface. It receives data in parallel from the data bus and transmits it serially. The data transferred via UART has a *start* bit, a *stop* bit and a *parity* bit before the stop bit to detect a single bit flip. Each data bit was sampled

once but the inaccuracy led to a technique called enhanced UART (eUART) [24], eUART samples a data bit 16 times and the voter helps to recover from the fault. This extensive sampling mainly aims to retrieve a fault free data bit that could lie outside the lifetime of a transient error.

2.4.4 Hardware Accelerators

Hardware accelerators are the differentiating components of modern-day SoCs. This thesis focuses on protecting these dedicated design units from being affected by transient errors. The base of fault tolerance in hardware accelerators is laid on redundancy where modules are replicated N times to ensure the delivered output's integrity. This technique is called Concurrent Error Detection (CED)[49]. Redundancy in fault tolerance can be categorized into two types

- Spatial redundancy;
- Temporal redundancy.

The following sections briefly explain the above techniques.

Spatial Redundancy

Since 1960's, concurrent error detection techniques like duplication with comparison, triple module redundancy and parity codes have been used. These techniques work based on the following principle. Assume that the system considered for a specific application realizes a function F and its corresponding output be F(i) where i is the input sequence that triggered the output. CED techniques make use of another module, either the replication of the original system or a module that checks the integrity of the data [48]. Both combinational and sequential logic designs use this CED technique. Following are the advantages of using CED: (1) Early detection of an error, thus preventing data integrity issues; (2) Quickly work on the corrective actions

to avoid further failures; and (3) To identify the Field Replaceable Unit (FRU) which is either a chip or a board. In the past, these CED techniques were used in IBM mainframes. The HP enterprise server, IBM S/360, IBM Enterprise System/9000 Type 9021 processors, IBM S/390 consisting of G4 CMOS processor chip, VAX 8600 and modules from companies like Tandem (Compaq), Hitachi, Sperry/Univac and many other companies used CED techniques [87].

Commonly used CED methods and their drawbacks are explained below [40]. A brief description of various fault tolerance techniques, especially different forms of CED could be found in [49]. Techniques for fault tolerance have improved over the past years but still a number of cases are yet to be solved. The mundane fault tolerant techniques are the Duplication With Comparison (DWC) and Triple Module Redundancy (TMR).

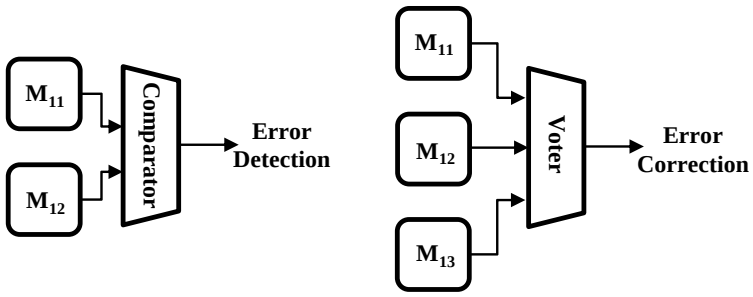

Figure 2.3: (a) Duplicate With Comparison (DWC). (b) Triple Module Redundancy (TMR).

Duplication with Comparison technique compares the outputs of two modules which are exactly the same [50]. If both outputs match, it means the system is error free and if it differs it indicates the presence of a transient or permanent fault. Fig. 2.3 (a) shows a DWC used for error detection.

According to [51], triple module redundancy is a fault tolerance technique in which a system is replicated thrice to make it fault tolerant. All the three modules are the same and perform the same task with the received input. Each of these outputs is

passed as inputs to a voter which later decides the final output based on majority. If all the three outputs are the same, it means the system is fault free and the outputs are safe to use. On the contrary, if an output differs, it indicates the presence of fault or an error. Fig. 2.3 (b) shows a TMR scheme used in a fault-tolerant system.

Common Mode Failure: TMR systems have the capability to correct/mask errors unless the system has multiple SEUs at more than one module at the same time. This is called Common Mode Failure (CMF) [52]. To understand this scenario better, let us assume $output_1$ and $output_2$ are the same while $output_3$ is different from them. The voter would choose $output_1$ or $output_2$ as its final output. What if $output_3$ is the correct output while the other two were faulty outputs which accidentally happened to be the same? In this case, the system's data integrity is lost and is dangerous if it happens in an aircraft or a space-craft. The worst-case scenario in any fault-tolerant system is that two outputs are identical but wrong. Thus, mechanisms to avoid this are required.

Diversity and its relation with CED: CED is a fault tolerance technique that duplicates the system which implements a function. The duplex system in Fig. 2.3 (a) has two identical modules. When design diversity is introduced, both modules are non-identical but realize the same function [45]. This detection technique also works based on comparison, but now both the modules have their own differences which can help avoid one main cause of CMFs, which is design error. Design errors are errors caused due to wrong designing, defects during fabrication or unnoticed dormant bugs. The output from both modules are compared. However, in this case, the possible fault pairs are reduced.

"A CMF is the result of an event(s) which, because of dependencies, causes a coincidence of failure states of components in two or more separate channels of a redundancy system, leading to the defined system failing to perform its intended function" [52].

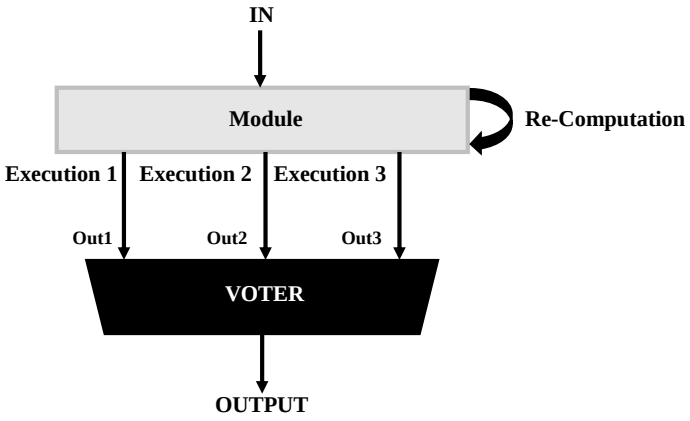

Figure 2.4: Redundancy in time.

Several studies are being conducted in an effort to overcome CMFs. The main idea is to automatically generate diverse redundant systems by perturbing the logic gate netlist such that two functional equivalent modules are obtained which manifest faults at their outputs differently [53, 50, 51]. This allows the voter to detect when faults have happened.

Temporal Redundancy

Spatial redundancy is fast and can fully mask errors (in the TMR case). Unfortunately, the area and power overheads are too large for many applications, and hence, other approaches have been developed to minimize the cost of fault-tolerant systems.

One such solution is to replicate in time, also called temporal redundancy. Temporal redundancy recomputes the same computation multiple times on the same hardware channel. It has the benefit that it reduces the area overhead of the circuit as the same hardware channel is re-used to recompute the output, as shown in Fig. 2.4. Similar to space redundancy, in temporal/time redundancy the output can be recomputed once or twice to either detect if an error has happened or to mask it. Time-based redundancy technique has proven its ability as a low-cost fault tolerance technique to reduce the effect of SEUs.

Time-based redundancy techniques exploit the fault's temporal nature to achieve fault tolerance as recomputations performed after the lifetime of this fault will be error free.

Temporal redundancy as a standalone fault tolerance technique is less studied while there exist several techniques that suggest an interplay of both space and temporal redundancy. In [76], the authors have elaborated the strengths of a transient fault which could span over multiple cycles in the temporal domain as well as multiple units in the spatial domain. They address this issue by engaging temporal redundancy along with spatial redundancy techniques. The authors in [58] proposed a re-execution strategy that shifts operands for consecutive computations. Most of the time-based redundancy techniques are used alongside error detection systems so as to start its recovery process via recomputation of the same task in the event of fault detection. In [6], the authors proposed using idle processors to compute the same task being done on an active processor. Later, several studies have shown that this method has higher performance penalties due to complex communication requirements [79]. A typical time-based redundancy phenomenon these days aims at bringing about a low-overhead solution for reliable electronics. The core idea is to make use of any available slack time to recompute if an error has occurred. If a block remains idle, they use it to recompute; else the system remains unreliable.

The main problem with time-redundant fault-tolerant systems is that they cannot detect permanent faults as the hardware channel is exactly the same. Thus, a permanent fault on the hardware channel would manifest itself on every re-execution. Moreover, time-redundancy can typically not be used in time-sensitive systems required to meet demanding deadlines.

2.5 Summary

This chapter has introduced the basic concepts around fault-tolerance in VLSI circuit design and typical approaches that are being widely used to build fault-tolerant circuits. The main approach is by adding redundancy in the system by computing the same operation on different identical hardware channels or by recomputing the same operation on the same hardware channel multiple times.

Chapter 3

C-Based VLSI Design

In the year 1965, Gordon E. Moore made an observation according to which the number of transistors in an IC would double every 18 to 24 months. This observation was called the Moore's law which still holds good after 5 decades [55]. The central technology that has fuelled Moore's law is transistor scaling. One of the main problems is that the power density has not remained constant since entering the sub-micron region. This implied that the benefits from Moore's law have significantly diminished as chips could not just achieve higher performance from simply scaling transistors down. This has led to a paradigm shift in computer architecture. Most complex ICs are now heterogeneous SoCs composed of multiple embedded processors that offload computationally intensive applications to dedicated HWAccs. These heterogeneous SoCs have also increased the complexity of the design methodology. Thus, new methodologies are required to keep up with consumers' demand.

Early days of IC design technology relied on a capture-and-simulate design methodology where block diagrams of the chip architecture were laid out manually or semi-manually which would later be transformed into its corresponding circuit schematic. This methodology did not scale well, and newer design methodologies were required to increase the design productivity. This was achieved through the

use of hardware description languages (HDLs) like Verilog and VHDL. Complex circuit designs could be specified using one of these HDLs, and logic synthesis tools would convert them into efficient gate netlist. This design methodology is still widely used. But with the further advent of technology, newer design methodologies are required to further increase the VLSI design productivity. One such methodology is HLS which takes an untimed behavioural description as input and generates efficient Verilog or VHDL code for the given function. With HLS an algorithm could be converted into a digital circuit [17]. HLS is more efficient than HDLs as designers do not need to specify any low-level details like clocks and resets. They just have to focus on the actual functionality of the application and leave it up to the HLS tool to generate the detailed optimized circuit.

This chapter covers the basics of HLS as well as system-level design features that most commercial HLS tools possess to build complete SoCs at the behavioural level. This thesis leverages some of the unique features of HLS in the context of fault tolerance and security. It is therefore vital to understand how HLS works to better appreciate the contributions made in this work.

3.1 High Level Synthesis

Fig. 3.1 shows an overview of the complete HLS process. HLS takes an untimed behavioural description (i.e. ANSI-C or C++) and a set of constraints as input, and generates efficient RTL code in Verilog or VHDL. The main steps include the parsing of the behavioural description to check for syntactical errors and to generate the control data flow graph (CDFG). The actual core of HLS processes consists of the following three steps: (1) allocation, (2) scheduling and (2) binding. These steps are all interdependent. Finally, the back-end of the flow generates the RTL code.

The next subsections describe these steps in detail.

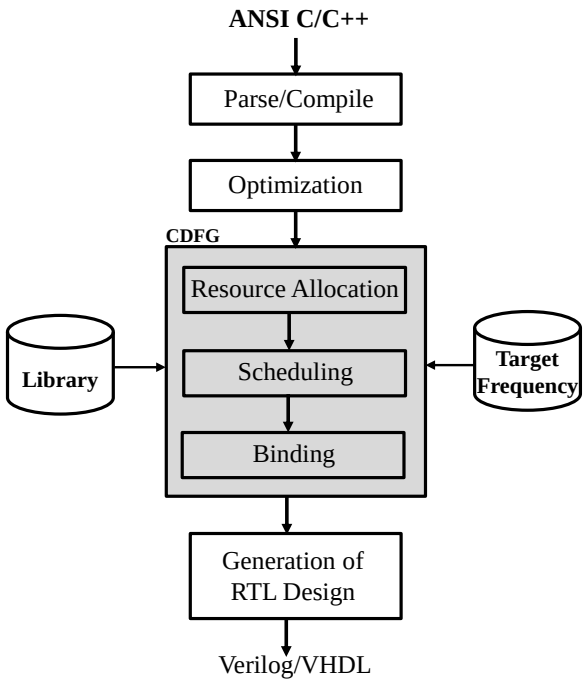

Figure 3.1: High Level Synthesis Design Flow.

3.1.1 Parsing

The behavioural input is first parsed and checked for syntactical errors. This step also performs technology independent optimizations, e.g., constant propagation and dead code elimination. The parser then converts the parsed code into a CDFG over which the next steps are executed. Fig. 3.2 shows a simple example program, and Fig. 3.3 shows the generated CDFG after this stage.

3.1.2 Resource Allocation

Allocation, being the first step after parsing, allocates the hardware resources required to execute the functionality described in the behavioural input. Commercial HLS tool often allocates the maximum number of functional units (FUs) to maximize parallelism that can be extracted from the CDFG. The functional unit count is a variable and can be modified by the user through a resource constraint file. This forces

```
L1    int main (){
L2    int in1,in2,in3,in4,…,in11;
L3    int a, d, g;
L4    int out1, out2, out3;
L5    a       = int1+int2;
L6    out1 = (a-in3)*3
L7    out2 = in4+in5+in6;
L8    d       = in7*in8;
L9    g       = d+in9+in10;
L10  out3= in11*5*g;
                }
```

Figure 3.2: Behavioural input example.

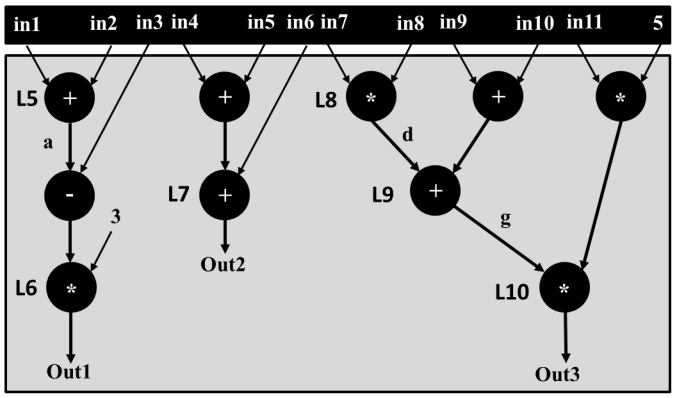

Figure 3.3: Data Flow Graph.

the resultant circuit to re-use some of the FUs mapped to multiple operations. This optimization is called resource sharing and is often used to trade-off area reduction with increased circuit latency. The total number of final resources will depend on the next HLS steps. The scheduling stage schedules different portions of the CDFG on individual clock cycles based on the delay constraint and the number of available resources. It should be noted that intuitively fewer FUs means smaller circuit are generated. But for FPGAs this does not hold good as the cost of the muxes required to share a FU most of the times out-weights the cost of the FU saved [28].

3.1.3 Scheduling

Followed by allocation is the scheduling process. This phase maps the operations in the CDFG onto individual clock steps based on their delay and the specified target synthesis frequency (f_{target}) and the resource available from the previous stage.

Multiple FUs can be chained together in the same clock step if there are enough resources, and their delay does not exceed $1/f_{target}$. Multiple clock cycles might be needed for some operations as they might need more time to complete their operations or due to clock cycles with small periods. The same is the case when the FU delay is too large. This is called multi-cycle operations.

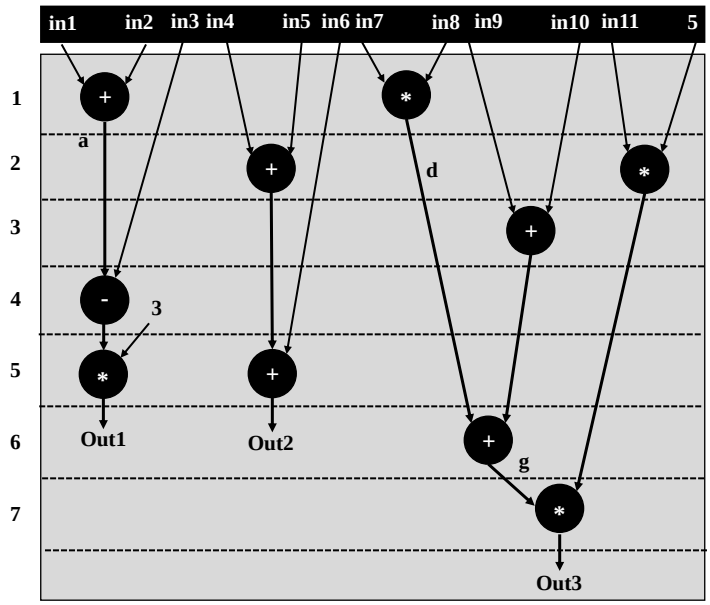

Figure 3.4: ASAP Scheduling with only 1 adder and 1 multiplier.

The scheduling process is an important step in the transformation of a behavioural input to an RTL implementation as it directly influences the design latency. This factor makes it difficult to decide on the best possible scheduling algorithm. The authors in [16] have proven that scheduling the execution of a design with the given resource constraints is an NP-hard problem. In an effort to solve this shortcoming

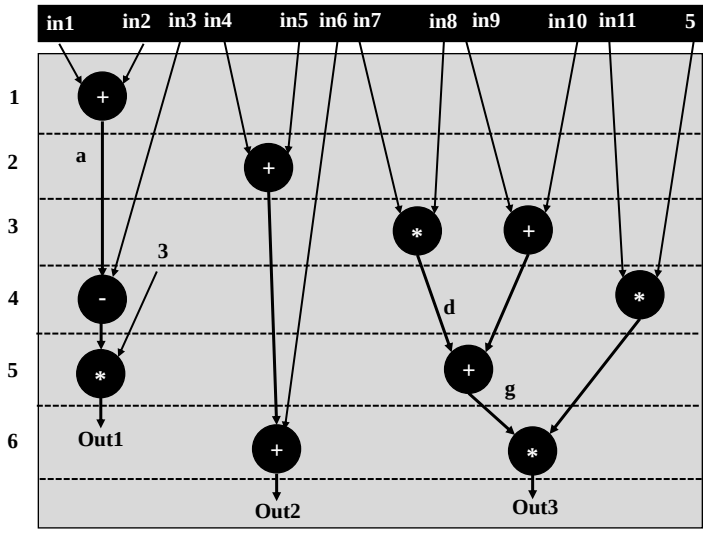

Figure 3.5: ALAP Scheduling with only 1 adder and 1 multiplier.

multiple heuristics have been proposed while the existing scheduling algorithms could be classified into two categories:

1. Data-Flow (DF) based scheduling,
2. Control-Flow (CF) based scheduling.

DF-based scheduling: Data-flow-intensive applications make use of DF-based scheduling. DF-based scheduling can further be classified into timing-constrained and resource constrained scheduling. Two of the most common scheduling algorithms due to their simplicity are As Soon As Possible (ASAP) and As Late As Possible (ALAP). Fig. 3.4 and Fig. 3.5 show the scheduling result of the sample code when only one adder and one multiplier is allowed. ASAP schedules the operations in such a way that the computation takes place at the earliest, i.e., when the input data is ready and when there are sufficient clock cycles available to perform the computation. In the ALAP case, the method tries to put the operation as late as possible. An operation is delayed up to the maximum limit for which it can be postponed. As seen in this case, ALAP scheduling method leads to better results as only six clock cycles are needed, while the ASAP method requires seven. The latency of the circuit

could be shorted in this case if the number of FUs is increased to allow two adders and two multipliers. In this case, the latency would be 3 clock cycles.

Other scheduling algorithms include force-directed scheduling [69], which is time-constrained scheduling where the primary focus is to balance computations within the available time step. On the contrary resource-constraint scheduling algorithms such as list scheduling [62] [31] makes a function based priority list of operations that are ready to be executed with the available resources.

CF-based scheduling: Control-flow-intensive applications make use of CF-based scheduling algorithms. Path-based scheduling algorithm [12] is one of those algorithms that deal with control-flow-intensive applications by scheduling each path as fast as possible. Loop-directed scheduling [7] is based upon depth-first search (DFS). The average-case performance is optimized while keeping an account of the loop iterations during DFS. Wavesched [39] is a scheduling algorithm that schedules ready operations in a manner that mimics wave-propagation. By scheduling independent loops together as well as unrolling it, the performance is enhanced. Advanced heuristics these days are capable of identifying hidden details that could be used to increase parallelism. SPARK introduces a set of code transformations to fit into high-level synthesis framework [27]. These code transformations are applied dynamically during scheduling using a global list-scheduling-based heuristic. CF-intensive applications are driven largely by I/O timing constraints due to its dependence on external circuits. Relative scheduling [37], being one of the earliest scheduling algorithms, manages minimum/maximum timing constraints efficiently. The authors in [54] have proposed VOTAN which makes use of a retiming-based approach in rescheduling a timed VHDL by using a behavioural code transformation strategy while still maintaining the original I/O timings. ILP-based scheduling [10] [25], symbolic scheduling [77] [89] and constraint-programming based scheduling [38] are exact scheduling approaches that retain the original I/O timing constraints. The schedul-

ing efficiencies of the above-mentioned algorithms vary from one case to the other. For example, DF-based scheduling cannot handle CF-intensive designs efficiently. On the other hand CF-based scheduling means [12] [39] [7] [77] in its worst case scenario has an exponential time complexity and is inefficient for large designs. With increasing complexities in SoC designs, there exists designs with a combination of intensive computations, controls and communications in addition to various time/area/power constraints. Scheduling algorithms like system of difference constraints (SDC) [16] convert a set of scheduling constraints into a system of difference constraints. The constraint equation generated matrix is capable of generating an integral solution which can be directly translated into a valid scheduling.

3.1.4 Binding

The last step in the HLS process following the scheduling phase is the binding stage, where every scheduled operation is bound to a FU in the given technology library. Design variables and operations need to be bound to specific storage units and functional units, respectively. The binding stage depends upon the allocation stage that decides the number of functional units used for the design. Register binding technique was proposed in [61] where the life time of a variable was divided into two intervals. The HAL system [15] proposes a technique where the number of registers used for binding is decided based on a weighted clique partitioning method. There exist other binding techniques such as BUD/DAA, EMUCS, HYPER, etc [2] [46] [15].

3.1.5 RTL Generation

The back-end of the HLS process finally generates the RTL code in VHDL or Verilog. Fig. 3.6 shows a typical RTL architecture generated after HLS. HLS generated RTL consists of an FSM and a datapath [57]. The datapath contains all of the FUs, while

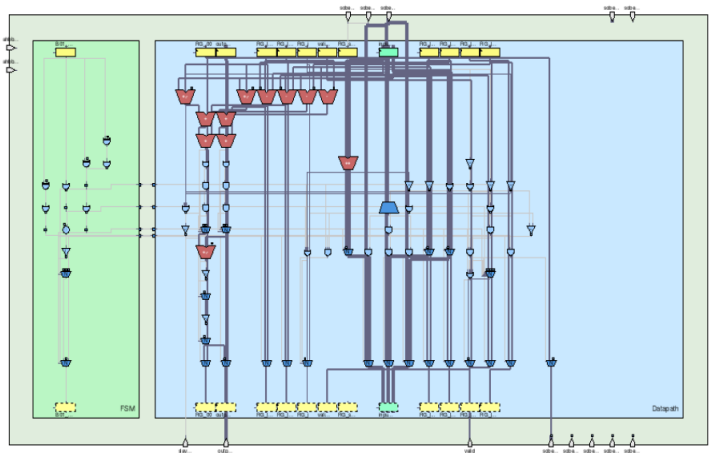

Figure 3.6: RTL Architecture after HLS (FSM+Datapath) from commercial HLS tools NEC CyberWorkBench [57].

the FSM generates the control signals to synchronize the data through the datapath correctly.

3.2 Advantages of HLS

This section describes some of the unique advantages of moving up the level of abstraction from the RT-level to the behavioural level. These advantages are exploited in this thesis, and hence, it is important to review them. They are: (1) the ability to generate micro-architectures from the same behavioural description with unique area vs performance and power trade-offs and (2) the ability to generate and simulate complete SoCs. Fig. 3.7 shows an overview of these advantages. The next subsections describe these two features in detail.

Advantage 1: Faster Design Cycle: RTL design requires to manually specify all the intricacies of the hardware using low-level HDLs. The HDL is then synthesized using logic synthesis tools to generate its gate level equivalent. From Fig. 3.7 it could be seen that a 1-million gate gate-level netlist can be generated by using 300K lines of RTL code, but the same could be easily generated using a 40K lines of C code.

This is why using HLS can accelerate the design cycle due to fewer lines of code that is required to generate the same 1-million gate gate-level netlist.

Advantage 2: Faster Simulation and Verification: HLS tools allow to generate simulation model at different abstraction levels ranging from transaction to cycle-accurate, which are multi-fold faster than RTL and gate netlist simulations. In [21], it has been estimated that behavioural simulation is 1000× faster than logic simulation while a cycle-accurate simulation is 100× faster. Thus, using HLS, the design verification cycle can be reduced, which in turn shortens the design cycle.

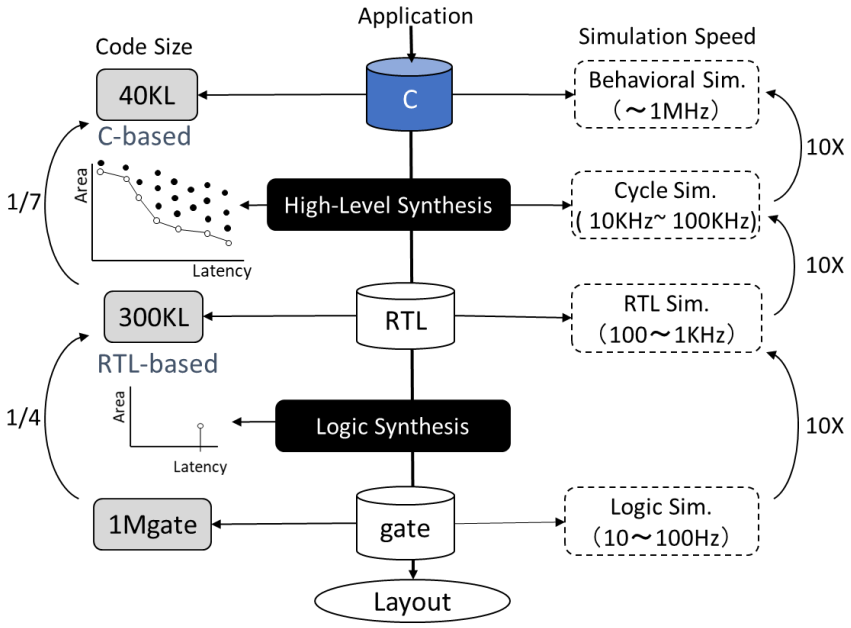

Figure 3.7: Advantages of using modern High-Level Synthesis tools [21].

Advantage 3: High-Level Synthesis Design Space Exploration: HLS allows to generate different micro-architectures from the same untimed behavioural description by simply setting different combinations of synthesis options. These are often called *knobs*, and they allow to generate a larger number of designs quickly that is impossible at the RT-level. This process can be fully automated with an automated design space explorer that sets these knobs automatically and evaluates their effect on the final circuit. This area of research has been very prolific and much work has been

done in this domain. A recent survey published highlights the main contributions done in this domain [75]. Fig. 3.8 shows a flow diagram of a typical exploration flow with the three main exploration knobs.

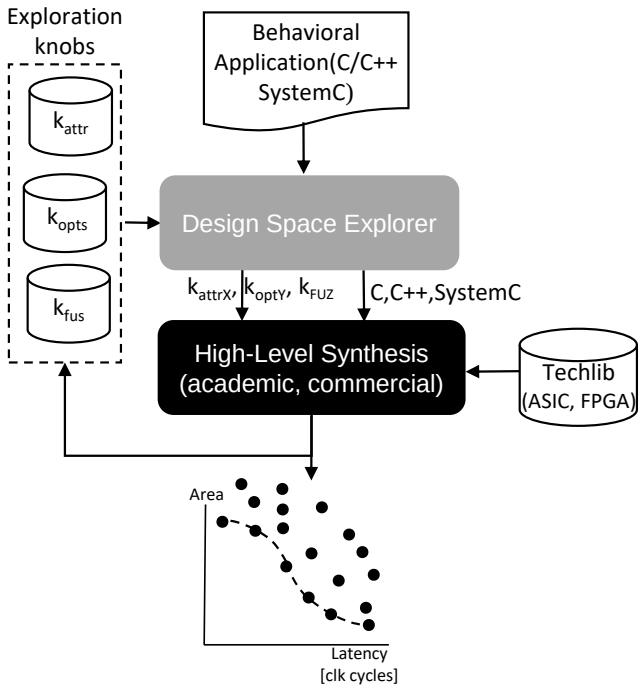

Figure 3.8: HLS DSE overview.

The following are the three different knobs.

Knob1 (k_{attr}): Synthesis directives: This first synthesis option is the most powerful because it allows to synthesize different constructs in the C code based on the user's preferences. For example, a loop could be fully unrolled, partially unrolled, not unrolled or pipelined; an array can be synthesized as a memory or registers. HLS tools allow to specify how these constructs should be synthesized by using pragmas inserted directly at the source code. Different mixes of these pragmas lead to unique designs with unique properties.

Knob2 (k_{opts}): Global synthesis options: These global synthesis options affect the

entire behavioural description and allow to control factors e.g., the target synthesis frequency and the FSM encoding scheme. They affect the entire HLS process.

Knob3 (k_{fus}): Functional unit constraints: This last knob controls the number of FUs that can be used to synthesize the description. This will control the amount of resource sharing in the final circuit.

An automatic HLS design space explorer's main goal is to set these knobs automatically such that the Pareto-optimal micro-architectures are found quickly. Because the search space grows supra-linearly with the number of knob combinations, many heuristics have been proposed.

Although this work does not directly deal with this topic, it will use HLS DSE to find a diverse set of micro-architectures to increase the fault-tolerance against CMF.

Advantage 4: Behavioural System-Level Design: The last advantage of raising the abstraction level is that complete heterogeneous SoCs can now be created using only untimed behavioural descriptions as input. This is typically done through the use of synthesizable bus interfaces and bus generation tools. It also allows *stitching* of different components through standard buses. One of the main advantages of this approach is that complete fast simulation models can be created for the entire SoC.

Fig. 3.9 shows the block diagram of a simple SoC composed of two masters and one slave connected through an on-chip bus. The on-chip bus that connects the different components could be an AHB, AXI of CoreConnect. Using synthesizable APIs make switching between buses elementary as only a functional call is required, significantly helping to experiment between bus types, sizes and arbiters. This approach makes the process of SoC design much easier when compared to designing at RT-Level.

This feature will be used in our first contribution to generate complete behavioural SoC. Through accurate cycle-accurate simulations, the time that each of the slaves is idling is investigated such that this time is used to re-compute its workload without any extra runtime overhead.

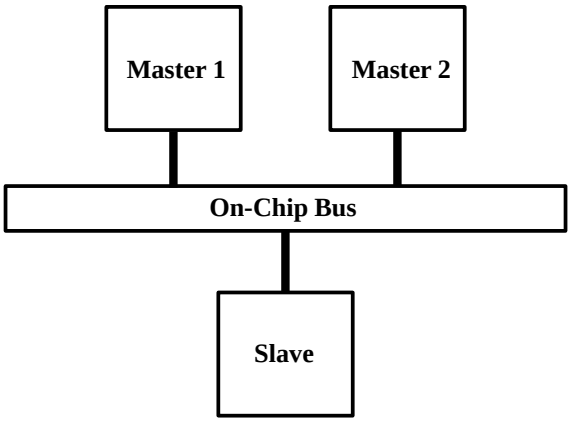

Figure 3.9: Master-Slave architecture.

3.3 HLS Tool Review

This last section reviews commercial and academic HLS tools currently available and their most distinct features.

There are multiple commercial and academic HLS tools of different levels of maturity in the market. Some target specifically ASICs (Stratus[11], Catapult[47] and CWB[57]) although the RTL generated can also be used for FPGAs, while others target FPGAs (Vivado[92] and HLS Compiler[29]). Table 3.1 summarizes the list of commercial tools and lists the input languages supported and what kind of *exploration* knobs that are allowed.

Table 3.1: Overview of Commercial HLS tools, their supported input languages and knobs.

	HLS tool	Input Languages	knob1 (pragmas)	knob2 (global)	knob3 (FCNT)
ASIC	Cadence Stratus [11]	C/C++/SystemC	x	x	x
	Mentor Catapult [47]	C++/SystemC	x	x	x
	NEC CyberWorkBench[57]	C/BDL/SystemC	x	x	x
FPGA	Vivado HLx [92]	C/C++/SystemC	x	(x)	
	HLS Compiler [29]	C++	x	(x)	

There are also multiple academic HLS tools available. Some include LegUP[13], GAUT[18] and BAMBU[63].

Each of these tools support different input languages like ANSI C, C++ or SystemC. They also have different amount of controllability in terms of the *knobs* supported.

3.4 Summary

This chapter has described the need to raise the level of abstraction from low-level HDLs to C and presented the main steps behind HLS. It has also introduced some of the most salient aspects of HLS that this thesis will leverage to build fault-tolerant systems.

Chapter 4

Fault Tolerant C-based MPSoCs using Time Based Redundancy Technique

4.1 Introduction

Fault tolerance can be achieved through different forms: either spatial or temporal. Spatial redundancy requires the same module to be replicated several times, and voter decides on the final output. One of the problems with spatial redundancy, aka NMR redundancy, is the high overhead involved. In the case of DWC 100% and in the TMR case 200%. In addition, the area of the voter needs to be included.

In certain cost-sensitive application, using spatial redundancy may be prohibitive. Thus, this chapter focuses on time based redundancy in C-based MPSoCs. The method presented in this chapter leverages the latest system-level design capabilities of commercial HLS tools that allow the design, simulation and verification of complete SoCs at the behavioural level. Our proposed method builds complete MPSoCs at the behavioural level, which contains a variety of loosely coupled hardware accelerators

(HWAccs) mapped as slaves onto a memory mapped shared bus. Since a single bus is shared by a number of HWAccs, each of them accesses the bus in turns based on the arbitration policy. We call this waiting time *slack*. The proposed method builds fast cycle-accurate simulation models of the entire SoC and extracts the slack time in each HWAcc. This time is in turn used to recompute the operation at the accelerators twice or thrice to maximize the data integrity. This recomputation methodology can detect the presence of transient faults or can even mask them completely. Although the proposed method cannot guarantee complete fault tolerance, experimental results show that especially for larger MPSoCs it can in most of the cases at least recompute the output twice and thus detect if a fault has occurred. This work mainly targets transient faults.

One of the key contributions of the work presented in this chapter is that the proposed system is fully described at the behavioural level. This is important because it allows for cycle-accurate model generators to precisely measure the slack in each HWAcc mapped as a slave in the MPSoC and hence get accurate fault tolerance results prior to the IC fabrication.

4.2 Target Platform and Overall Automated Flow

Fig. 4.1 shows the MPSoC platform, which is the target of this proposed work that was presented in [3]. It could be seen that the system has M number of masters and N number of slaves. The slaves are loosely coupled HWAccs that are mapped onto the MPSoCs. In order to build this type of platform, a fully automated MPSoC generator is created. The MPSoC generator takes as inputs N BIPs in synthesizable ANSI-C or SystemC code for HLS and generates automatically different MPSoC configurations with M number of masters and N slaves interconnected through a standard shared

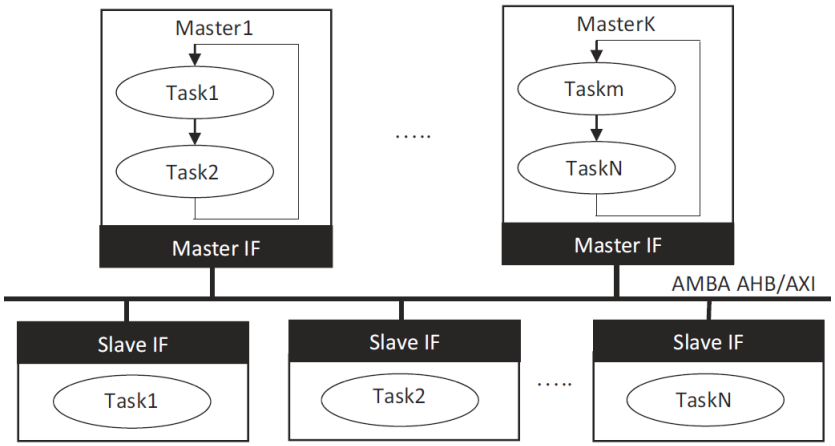

Figure 4.1: Target MPSoC platform [3].

bus (e.g. AMBA AHB or AXI). The BIPs are synthesized as slaves in the system, while the masters emulate processors executing different tasks. The masters then send data to the slaves continuously repeating the same sequence. The HLS tool used in this work allows to generate a completely synthesizable C-Based MPSoC by using a bus interface library. This allows our method to quickly generate new MPSoC configurations and simulate these to evaluate their fault tolerance using a cycle-accurate simulation.

The main idea conceived in this work is to make use of the slack time available.

Slack Time and Bus Congestion

A complex system is built to process multiple information at the same time. It consists of several processors and dedicated hardware accelerators that perform unique tasks. They are all bound together into a single entity by an on-chip shared bus. Each of these processors sends out data to any hardware accelerators if they are loosely coupled and to their respective ones if they are tightly coupled. The limiting factor is that there is only one bus for all transactions, i.e., for both sending data and receiving data. The bus access is based on the priority rule set by the user.

Each hardware accelerator in such a system receives the necessary inputs from the processor via bus and start computing, after which it waits over several clock cycles for it to send the output back to the processor via bus again. The time that it waits is known as its slack time. Fig. 4.2 shows the computation and wait time partitions for three different hardware accelerators that share the same bus. The computation and wait times for each accelerator are unique. This data transfer traffic within the bus is called bus congestion.

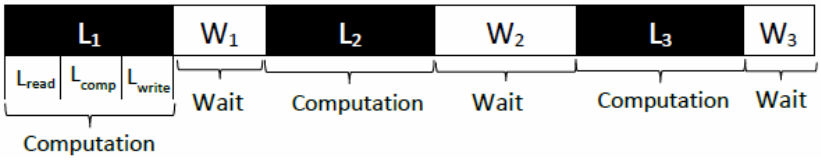

Figure 4.2: Wait time W and Computation time L of different HWAccs.

C-Based design methodology helps in the easy generation of bus interfaces and also lets us easily calculate the slack time between different processes using the cycle-accurate model generator. These features thus make the building process of an MPSoC far simpler.

4.2.1 Behavioural MPSoC Generation

A completely synthesizable MPSoC can be generated using the bus generator and cycle-accurate model generator provided by the commercial HLS tool used for this work (Cybus). Building such an MPSoC itself is one of the unique feature of this work. Fig. 4.3 shows an overview of the complete automated flow. The flow is colour coded. The grey parts denote parts developed by us and black parts indicate third party tools (e.g.HLS). The flow takes as inputs N behavioural IPs with their test benches which will be mapped as loosely coupled HWAccs on the SoC, thus $S = \{BP_1, BP_2, ..., BP_N\}$, information about the bus structure (e.g. arbiter type,

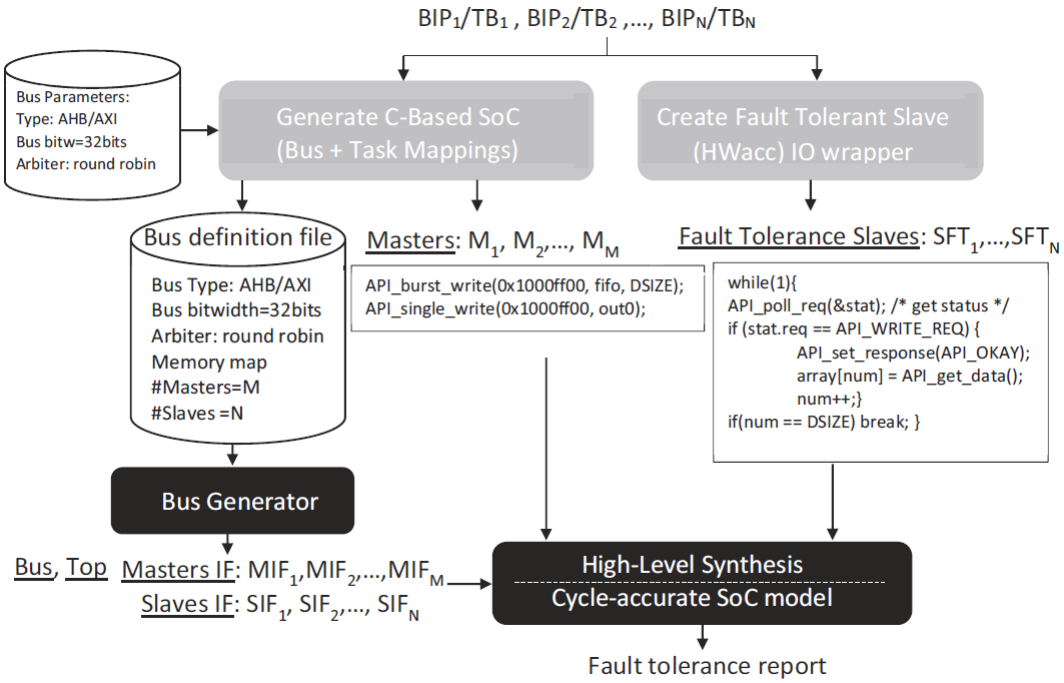

Figure 4.3: C-Based MPSoC generation flow [3].

round robin used by default), bus bandwidth (32-bits used by default), type of bus (AHB or AXI) and the number of masters.

Given below are the steps to be followed before the bus generation process.

Step 1: Inclusion of Bus Interface: From the generation flow diagram shown in Fig. 4.3, it is clear that the bus generation process is separate. The inbuilt APIs help in interfacing. All the masters in this MPSoC use single write or burst write APIs, which are function calls to write the test inputs into the bus. The commercial HLS tool (CWB) used for this work has the ability to generate bus interfaces. Any modifications made on the bus does not affect the master or slave modules as it is isolated. This is an advantage of using C-Based VLSI design over the traditional RTL design methods. AMBA AHB bus interface is used in this design.

Step 2: Bus Definition and Generation: After all the BIPs are ready, the next step is to generate the bus and all its corresponding interfaces like the master interfaces and the slave interfaces. CWB understands the user requirements through the bus definition file also called the bdef file. This file has individual information about the masters, the slaves and the memory address to which they are mapped to, the bit-width information of the bus required, and also the arbiter information (round-robin or fixed priority). The bus generator with the help of the bdef file generates master and slave interfaces, the bus and a top module which binds everything together into a single entity. All the above CWB generated designs are in ANSI-C and are completely synthesizable.

Step 3: Generation of Cycle-accurate model: Each and every process has to be synthesized separately after which the whole system is ready for a cycle accurate simulation. The necessary input sequences are fed in to simulate the top module. The simulation result clarifies how long each slave works and how long it waits for the next input or to write back the computed output. The data from the simulation helps us to understand about the time available for recomputation.

This module is modified to perform recomputation with whatever slack time the slave has with it. After each computation it recalculates and checks the output's authenticity instead of waiting for the bus to write back the output. Once the bus signal goes high, it writes back the final result after re-computing and re-checking.

4.2.2 Proposed Fault Tolerance Architecture Using Available Slack Time

The MPSoC targeted in this work has multiple loosely coupled HWAccs. The difference between tightly coupled and loosely coupled HWAcc is that the former architecture follows one master one slave pattern thus making the slave accessible only to

its master while the latter one lets any master access any slave. Creating MPSoCs at the behavioural level has the unique benefit of creating cycle-accurate models of the entire system and precisely measuring the slack available in the system based on bus congestion caused by read/write overheads. This in turn allows to understand the fault tolerance of the system by accurately estimating the number of times a task could be recomputed.

Fig. 4.4 shows the partitions that help in re-computing without any loss of data. This partitioning can help the system perform any task execution on the HWAcc. The HWAcc is split into three parts to keep all the work separate. The three parts are (1) Read module, (2) Computation module, and (3) Write module. Tasks done by each of these parts are explained in detail. Fig. 4.4 shows the proposed architecture with partitioned read, HWAcc, and write modules.

Read Process: The read process is the first part of the slave, which waits for initiation from the master. The behavioural description of the slave makes use of synthesizable APIs provided by the HLS tool. Synthesizable APIs are function calls to read data from the bus. The code keeps polling the bus to check if there is any request from the bus. Once the request status receives a write request from the bus, then the read process is all set to receive its new data. Once it has received the data entirely from the bus, flag_{inew} goes high. Along with it, idata passes the data to the computation module. The signal flag_{inew} is an output from the read process while it is an input into the HWAcc. Its main function is to intimate the arrival of a new input sequence to the HWAcc.

HWAcc: It is a part of the code which computes or performs the function for which the HWAcc was designed for. Here in this work odata is calculated, which is later passed on to the store/write module. After the first computation in normal

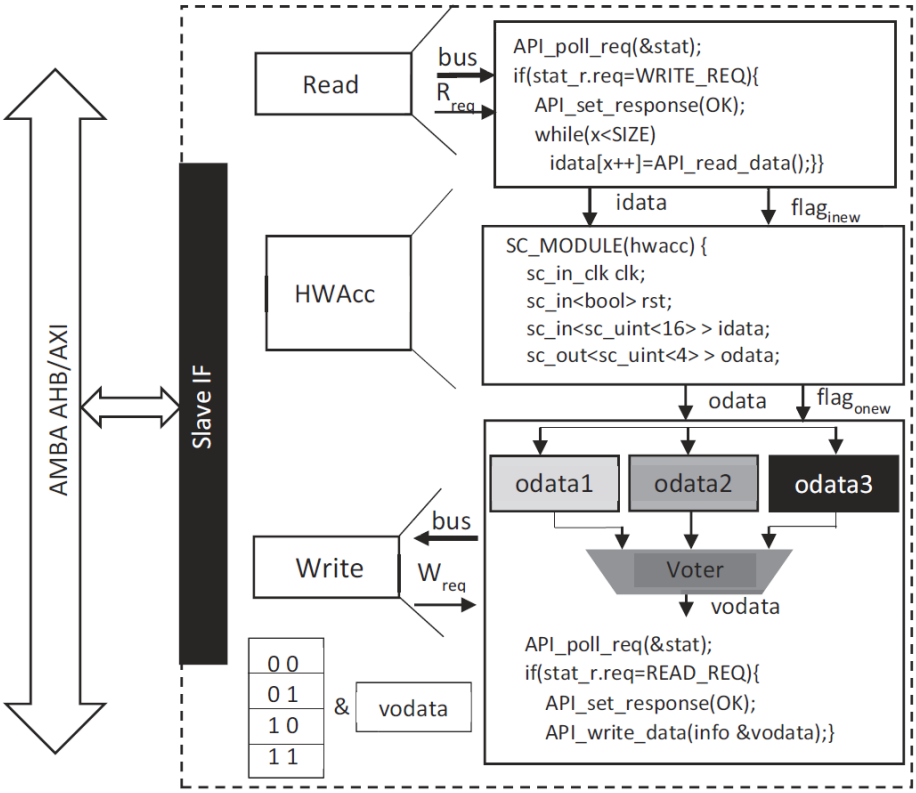

Figure 4.4: Proposed Architecture [3].

cases, the slave keeps polling the bus to check if there is read request from the bus in order to take back the computed odata onto the master. In our method, the slave re-computes until the flag_{inew} goes high, which means a new input sequence has been received, and new computation has to start. To clearly understand, let us assume that idata was received and the first computation was successful, but it took a long time thus not writing back the output. Now that there is time for the next read request to arrive from the bus, our HWAccs have excess time called the slack time to re-compute and check if the previous computation was free from transient errors. When the computation is in progress and if flag_{inew} goes high, it completes the current computation and proceeds to compute for the new input data received. After each computation, the data is passed to the next module to store, compare and write. When it writes the output odata of a brand new input sequence, then flag_{onew}

Table 4.1: Information encoding for master.

Code	Fault	Description
00	U	No re-execution accomplished. No guarantee on integrity of the result.
01	N	Computation re-executed twice. Integrity of the result conserved.
10	Y	Computation re-executed twice but results do not match. Fault detected.
11	N	Computation re-executed thrice. Fault detected and corrected using a majority voter.

goes high.

Write Process: The write process is responsible for storing each and every computation odata into different registers based on the iterations. The flag_{onew} signal indicates if it is a new output or it is a re-computation result. The first computed output is stored in odata1, the next in odata2 and the third computation in odata3. After which the voter decides the correct output based on majority. Not all cases are the same because few slaves do not have enough time to recompute thrice. The next task to be kept in mind is how to rate the output if it is a safe one or a misinterpreted output. To answer this, our work has implemented a way with which the output is verified thus called verified output data (vodata). The output vodata itself explains its authenticity. This is done by adding two bits at the MSB position, which says about the data integrity. If the slave did not get sufficient time and could compute only once, it cannot be safe to use or is not for sure the correct output. Under these circumstances, "00" is appended to the MSB bits of vodata. If the slack time of the HWAcc was sufficient to recompute, i.e., if it could compute twice, then it must be able to detect the presence of any transient error even if it cannot correct it. In this case, if the outputs are the same, then it is safe. For this case, the output is

Table 4.2: Complex System Benchmarks.

Bench	S1	S2	S3	S4	S5	S6
Ave8	1		1	1		1
		1			1	
Bsort			1	1		1
		1			1	
FIR			1	1		1
		1			1	
Sobel	1		1	1		1
HWAccs	2	3	4	4	3	4
Masters	1	1	1	2	2	4

appended with "01" which indicates computation happened twice and the results did match with each other and "10" would be appended to vodata if odata1 and odata2 did not match. When the slave has enough slack time to re-compute thrice, then the fault can be detected and corrected which means the output is the safest in this case. "11" is appended to the MSB bits of vodata to indicate that the output was computed thrice and also matched to produce the majority. In this case, SEU errors are completely masked.

It should be noted that no FIFO is used at the output registers as it is assumed that the master will always read the output before sending new data to the same slave. The regular structure of this architecture makes it fully parameterizable and hence can be used with any HWAcc with very little area overhead. The main disadvantage of this method is that it cannot guarantee fault tolerance if performance degradation is not allowed. In case that some degree of performance degradation is allowed, then the proposed architecture could lead to fault tolerance by always guaranteeing that the result will be computed three times. In this case, the masters would have to wait until the slaves have finished with the triple computations. One other obvious drawback of this method is that of the extra power consumption due to the recomputation of the outputs. It is nevertheless assumed that any fault tolerance system includes this extra power consumption in their power budgets.

4.3 Experimental Results

Different computationally intensive application, amenable to HW acceleration, were selected and grouped together into complex systems in order to test our proposed method. These designs were taken from the open source Synthesizable SystemC Benchmark suite (S2CBench) [74]. Table 4.2 shows how these complex benchmarks were formed. The first column indicates the name of benchmark used from S2C benchmark suite. Columns 2 to 7 are 6 complex systems S1-S6. The 1's underneath each of these columns corresponds to different benchmarks used as HWAcc in that system. The last two rows report the total number of slaves used in each system and below that the total number of masters instantiated in each system. The ave8 computes the average of 8 numbers, bsort sorts 8 numbers, fir is a 9-tap FIR filter and sobel a 3x3 edge detection algorithm. Although these designs are relatively small, they should help as a proof of concept of our proposed method. The experiments were run on an Intel dual 2.40GHz Xeon processor machine with 16 GBytes of RAM running Linux Fedora release 19. The HLS tool used is CyberWorkBench v.5.5 [57]. The target architecture, as mentioned previously, is a multi-core processor system with as many masters as BIPs, with a 32-bit AMBA AHB bus using a round robin arbiter. The target technology is Nangate's 45nm Opencell technology and the HLS target frequency for all of the processes in the system is set to 100MHz.

Fig. 4.5 shows the histograms, in percentage of the times that each HWAcc in the system could only compute the output once ("00"), twice ("01" "10") and thrice ("11"). The encoded numbers in the x-axis correspond to the code returned by each HWAcc after each computation. As mentioned previously, the experiments do not involve any fault injection, hence the cases "01" (computation was repeated twice and results match) and "10" (computation was repeated twice but results do not match) are grouped together.

From the results, it can be observed that the more complex the system becomes,

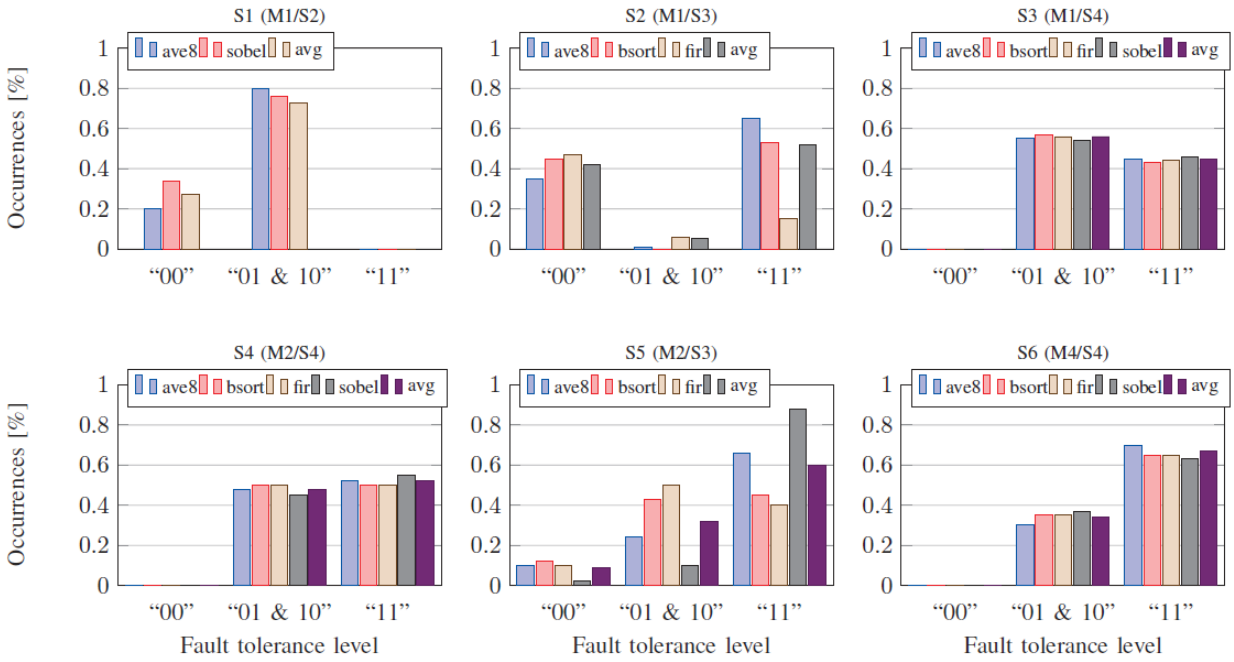

Figure 4.5: Fault tolerance level for the different SoC configurations [3].

the larger the percentage of times that the HWAccs could recompute the outputs. Intuitively this makes sense as the system only has a single shared bus, which means that the masters and slaves have to wait longer to get access to the bus, thus leaving more time to the slaves to recompute the results. It can be particularly observed that for half of the systems (i.e. S3, S4 and S6), the slaves always had enough time to at least re-compute the results once.

Table 4.3 shows the area overhead introduced by our proposed architecture vs a typical architecture without any fault tolerance mechanism (no FT) and also the area savings when compared with a typical DWC and TMR systems, considering only the slave portions of the system (excluding the area of the masters and buses). From the results, it can be seen that the area overhead seems considerable. This is mainly due to the fact that the actual benchmarks are relatively small compared to the fixed area overhead required to store the recomputation of the result as well as the voting scheme. For larger HWAccs this overhead will become negligible. This

Table 4.3: Experimental results: Area overhead and simulation running time.

Bench	**Area Overhead vs. no FT [%]**	**Overhead vs. DWC [%]**	**Overhead vs. TMR [%]**	**Running time [h:m:s]**
S1	75	5	30	0:52:43
S2	53	18	30	0:28:26
S3	62	12	34	1:01:25
S4	57	11	33	0:54:32
S5	52	22	30	0:31:15
S6	61	10	32	1:03:26
Avg.	60	13	32	0:48:38

table also shows that the area savings over traditional spatial redundancy methods is on average 13% and 32%compared to duplicating the same HWAcc (DWC) and triplicating the HWAcc (TMR) respectively. This overhead difference is expected to grow with larger HWAccs. Thus, our proposed method is much more scalable that a traditional spatial replication approach.

In terms of running time, it can be observed that the simulation of the complete systems can be performed in a reasonable time, being on average 48 minutes. Thus it is not necessary to prototype these systems onto an FPGA. It should be obviously noted that the running time increases with the complexity of the system and thus for systems with larger number of masters and HWAcc with more complicated interconnects (e.g. NoCs), it might be necessary to prototype the system onto an FPGA. Anyway, we believe that this step could be easily automated in our flow, but leave it for future work.

Scalability of the proposed fault-tolerant architecture: In the proposed fault-tolerant MPSoC, we assume that a master sends new data to a slave only after it receives the output computed by the slave for the last sent data. This assumption was made to ease the process of initial investigation on using slack time efficiently to re-compute to ensure data integrity. The results have proven that the slack time could be efficiently used by the slave to recompute and that the MPSoC becomes robust

against transient errors with increasing complexity. But with increasing complexities, there is a need to analyze its scalability. The fault-tolerant MPSoC here only makes use of three storage registers to store results of maximum three computations that could be repeated for a given input. But in real-time, read and write requests from the buses could be asynchronous which means a new data could be sent even before a read request from the bus. This asynchronous behaviour will only increase with increasing design complexities, and it could also cause loss of data. In order to address this scalability issue, a FIFO could be introduced onto the write back module to store the final results computed by the voter. This would help in streamlining the output data transfer back to the master without loss of data. It should be understood that this will lead to an extra area overhead, but a relative comparison against a much complex MPSoC will only lead to a permissible overhead.

4.4 Summary and Conclusion

This chapter presents a fully automated behavioural MPSoC design flow based on a parameterizable architecture to enable time redundancy of loosely coupled HWAccs of behavioural MPSoCs. The system concurrently re-computes the output at each HWAcc while receiving and sending data from and to the master to determine if any soft errors have happened. Experimental results show that depending on the complexity of the SoC, each HWAcc has more or less slack to recompute the results two or three times, achieving in this last case full fault tolerance for SEUs. The main benefit of this proposed flow is that the area overhead is small compared to traditional module redundancy systems, but the main drawback is that it cannot guarantee full fault tolerance if the performance of the system should be preserved.

Chapter 5

Common Mode Failure Mitigation

Most previous work on VLSI design reliability is based around time or space redundancy where in all cases the underlying hardware channel is exactly the same. Reliability is tackled very differently in the software domain. In contrast to most hardware faults, software faults (bugs) exist in every software instance. Therefore, the same software program cannot just be replicated and executed onto different hardware channels. In order to address this issue, software developers rely on code diversity. A classic approach to add diversity is to set up multiple teams working in parallel and independently on the same design. This is an expensive method but still the standard approach when extremely reliable systems are required. In order to make software diversity more efficient, much research has been done to automate this process. One of the main techniques investigated is diverse compiling. This chapter investigates if similar techniques could be applied to C-based VLSI design to find functionally equivalent, but diverse micro-architectures from a single input behavioural description for HLS.

As introduced in chapter 3, raising the abstraction level has some distinct advantage over the traditional RTL design process. One salient advantage is the ability to generate micro-architectures with unique characteristics from the same behavioural

description. The authors in [86] showed that HLS DSE could be used to generate different types of NMR systems by combining different types of micro-architectures generated automatically. These NMR systems mainly focus on detecting SEU in individual modules. One crucial problem is that electronic systems are also prone to Common-Mode Failures (CMFs). CMF imply that multiple modules in the NMR system could present a fault at the same time. This can lead to catastrophic failures if the output of two modules is the same and wrong, as the voter cannot detect the fault. For example, an aircraft crashed because of a common vibration mode that affected all three parts of a TMR system [8]. One way to address CMFs is by building diverse redundant systems. The main goal is to force the output of the two modules to be wrong but different from one another when an SEU occurs. In this case, the voter can detect the error and take pro-active measures.

Because HLS allows generating micro-architectures of different characteristics automatically, it looks tempting to study if HLS can be used to generate a more diverse set of NMR systems to mitigate the effect of CMFs. Moreover, the main problem with traditional approaches to estimate the diversity is that all possible stuck-at-fault pairs between the two netlists need to be fault modelled, which is computationally too expensive. Thus, this chapter tries to investigate if the diversity computation can be done at higher levels of abstraction, e.g., at the RT-level or HLS level, and also study if predictive models could be used to find the more diverse micro-architectures at the RT-level.[1]

[1]This work has been done in collaboration with Farah N. Taher [22]. My specific contributions have been the study of predictive models at the RTL level as well as investigating if diversity could be estimated at the RT-level as opposed to the gate netlist level and compare the same with diversity predictions at the behavioural level.

5.1 Hardware Diversity

So far, the main way to generate diverse system has been to perturb the logic gate netlist of the modules to be replicated such that the two netlists are as diverse as possible [50]. The main problem is how to measure the degree of diversity between two modules? Two approaches have been proposed up to date, the first called Dmetric and the second DIMP.

Diversity Metric (Dmetric): In [53], the authors introduced the first metric to measure the diversity between two functionally equivalent circuit called Dmetric. In this case, diversity could be defined as the number of stuck-at-fault pairs that lead to the same wrong output as a fraction of all possible stuck-at-fault combinations. Formally the diversity can be calculated as follows:

$$K_{i,j} = \frac{inputs\_that\_jointly\_detect\_output\_flip}{total\_no\_of\_inputs} \tag{5.1}$$

$$D_{ij} = 1 - (K_{ij}) \tag{5.2}$$

where $K_{i,j}$ is the percentage of the total number of stuck-at-fault pairs that lead to the same result from all the different stuck-at-fault pair combinations and $D_{i,j}$ is the diversity metric. The higher the $D_{i,j}$, the more diverse the two modules are.

Let us consider two functionally equivalent designs D_1 and D_2. Let the probability of fault occurrence in D_1 and D_2 be f_i and f_j , respectively. Let us assume (f_i,f_j) is a fault pair. The diversity D_{ij} value for the fault pair is the conditional probability that the two implementations do not produce identical errors, given that the faults f_i and f_j have occurred [53]. For any input sequence, the implementations could produce one of the following outcomes: (1) both the outputs can be correct; (2) one of the two implementations produce the correct output while the other produces the wrong output; (3) both implementations produce incorrect and different outputs; or (4) both outputs can be wrong but identical.

DIversity Metric based on circuit Path analysis (DIMP): Although very accurate, one of the main problems with the Dmetric is that it is too computationally intensive as it requires all possible fault pairs to be simulated. Thus, recently the authors in [1] presented a DIversity Metric method based on circuit Path analysis (DIMP) that estimates the diversity between two designs based on the structural analysis of their gate netlist. Thus, no simulations are required, making the method much faster. This method's base lies on the assumption that diversity depends on the gate order through which input PI_i traverses to reach output PO_j. This assumption means that micro-architectures are very closely related if the different logic paths from input to output have the same gates. Thus, DIMP generates a list of all logic paths between primary inputs PI_i and primary outputs PO_j in both implementations and compares them. The formal definition of DIMP is given as follows:

$$DIMP = \sum weight_{(p^1_{i,j}, p^2_{i,j})}.(1 - overlap_{(p^1_{i.j}, p^2_{i.j})}) \tag{5.3}$$

$$MaxDIMP = \sum weight_{(p^1_{i,j}, p^2_{i,j})}, DV = \frac{DIMP}{maxDIMP} \tag{5.4}$$

In the above equation, weight refers to the maximum gate count of the paths being compared while overlap refers to the number of gates repeated across paths in the same order. The diversity of a design pair tends to 0 when both implementations are identical; and it tends to 1 when they are dissimilar, making them less susceptible to CMF.

One of the DIMP problems is that the authors do not show that design pairs with high DIMP also translate into low CMFs as they do not compare their results with the Dmetric results. Dmetric estimates the actual robustness of the system towards CMFs as it generates stuck-at-fault pairs.

Based on the limitations exposed here of these two diversity metrics, the question that we tried to address is to find a fast and accurate model to predict the diversity

of two functional equivalent modules in order to guide a HLS DSE in finding the two most diverse micro-architecture pairs automatically.

5.1.1 Raising the level of Abstraction to Increase Diversity

The original work on hardware diversity perturbs a given gate netlist to generate two functionally equivalent netlists with higher Dmetric [50]. The problem with this approach is that only a minimal amount of diversity could be extracted. Raising the level of abstraction from gate netlist to RT-level and then to behavioural level should increase the amount of diversity obtained, thus, making NMR systems more robust against CMF.

Fig. 5.1 depicts three cases of the same DWC fault-tolerant system. The first one instantiates the exact same micro-architecture twice. Thus, the system has no diversity at all and is very vulnerable to CMFs. The second case shows how diverse systems could be generated by performing gate-level exploration. In this case, logic synthesis options are modified like timing constraints, max fanout and logic depths. This leads to different types of gate netlists with some degree of diversity. Finally, further raising the level of abstraction to C allows to generate a much larger pool of candidates by setting different mixes of the synthesis knobs explained in chapter 3. In particular, the synthesis directives are specified as pragmas.

The main idea behind this work is to extend this explorer to find design pairs with the highest diversity. For this, a quick diversity estimator is needed to guide the explorer right after the HLS process instead of performing a full Dmetric calculation or DIMP evaluation at the gate netlist, which would involve a full logic synthesis and in the case of Dmetric stuck-at-fault simulation.

To systematically study the accuracy of the proposed flow, we implement an evaluation framework that compares the following diversity methods: Dmetric at the gatenetlist (original method and most accurate), Dmetric at the RT-level, and DIMP

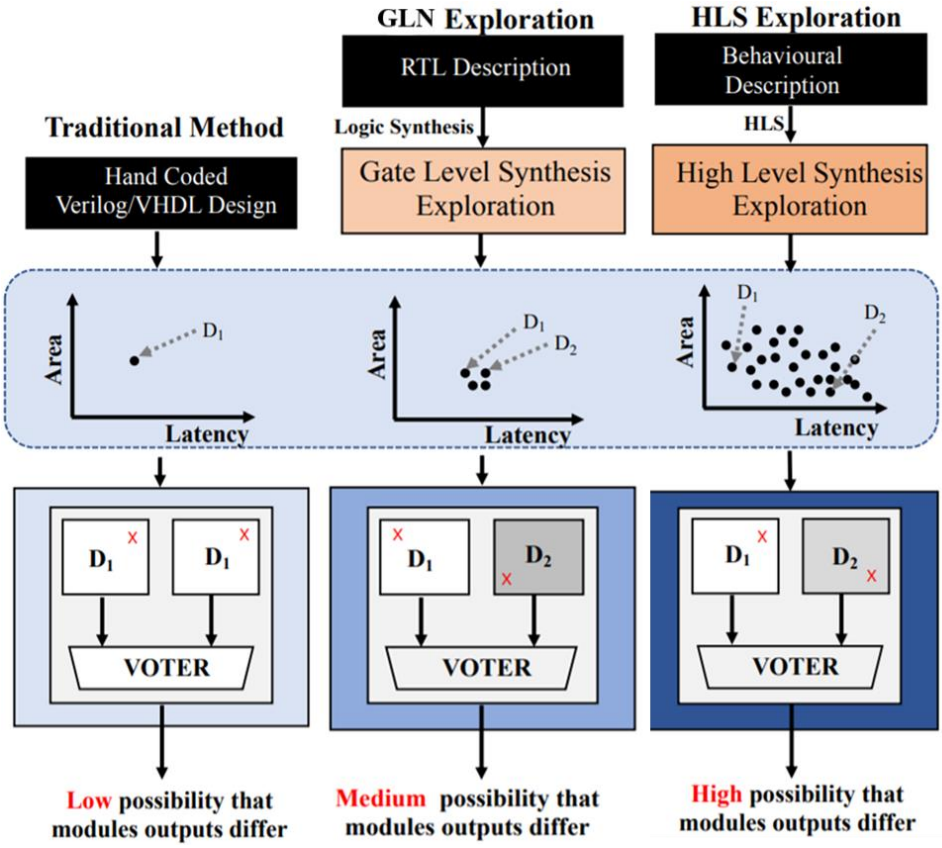

Figure 5.1: Comparison of expected of diversity, hence, protection against CMF between: (a) Traditional method based on identical hardware channel, (b) Gate-Level exploration, and (c) HLS DSE.

at the gate netlist as shown in Fig. 5.2. For the two Dmetric approaches, a full stuck-at-fault simulation is needed for all the stuck-at-fault pairs. The obvious advantage of doing this at the RT-level is that it has fewer fault pairs. The main problem is that the Dmetric value will not translate into finding the most robust system against CMFs. The last step is to mine this data to generate a predictive diversity model that can drive the HLS DSE to find the two most diverse micro-architectures with the result reported by the HLS tool.

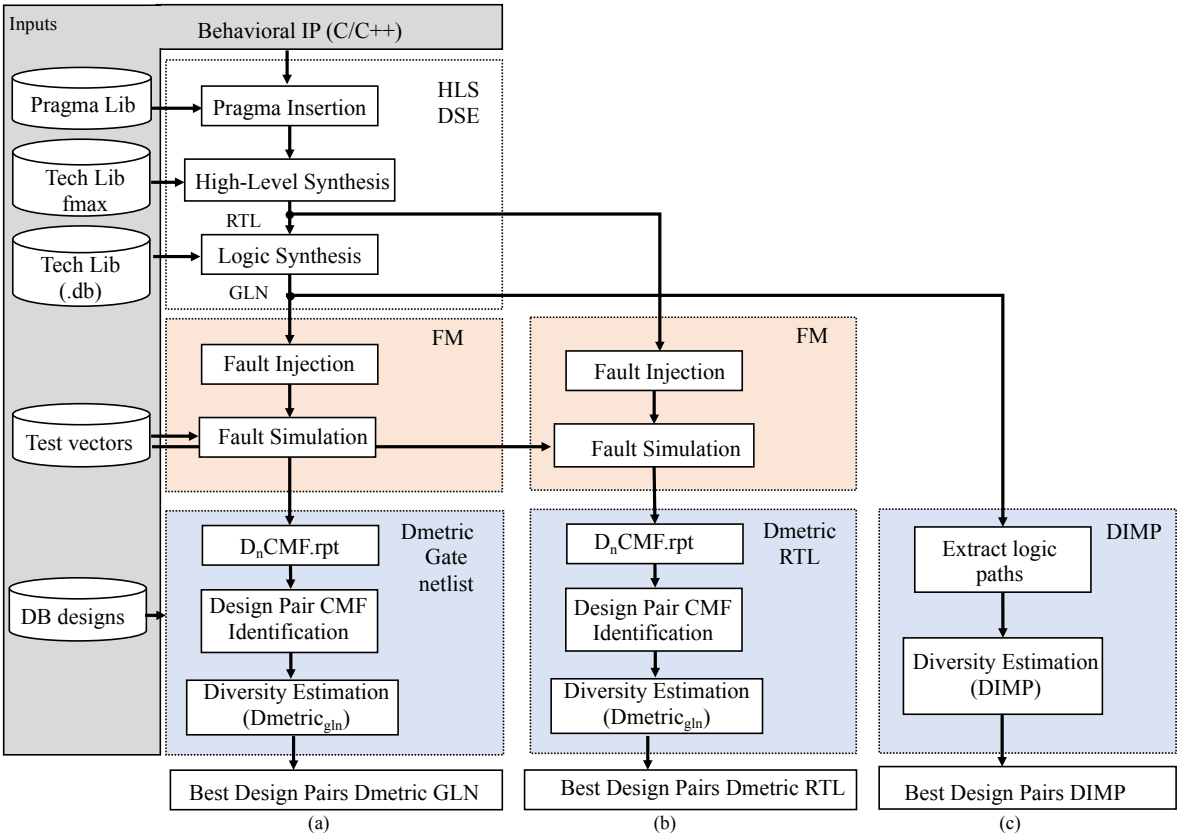

Figure 5.2: Proposed HLS DSE flow to find diverse micro-architectures using: (a) Dmetric on gate netlist, (b) Dmetric on RTL and (c) DIMP on gate netlist.

5.2 Proposed Diversity Estimation Methodology

This work aims first at automating the design process of building a space redundant fault-tolerant system that has a better potential to mitigate CMF by utilising the strengths of HLS DSE. The second objective is to find a quicker way to determine the diversity of two micro-architectures generated right after HLS, i.e., without having to perform a full Dmetric or DIMP calculation. In this section, we address the first topic and investigate how well DIMP approximates Dmetric for diversity and also compare the results obtained from Dmetric using RTL (Verilog) instead of the gate netlist. This is important because the alternative methods would tremendously speed up the diversity computation.

Fig. 5.2 shows the overall flow. The input consists of an untimed behavioural description, a library of pragmas for it, and the test vectors required to evaluate the effect of stuck-at-faults on the design. The output is the design pairs with the highest diversity. Additionally, constraints like maximum latency or maximum area can be specified. The proposed flow calculates both DIMP and Dmetric for diversity as shown in Fig. 5.2. This will help us to understand if DIMP could be used to mitigate CMFs. The flow consists of three steps as follows:

Step 1: Design Space Exploration: The first phase generates different RTLs from a given untimed behavioural description. For this, different mixes of pragmas are given to this untimed behavioural description (BIP). HLS is executed, and the generated RTL is synthesized. The results of this first step is a gate netlist for one micro-architecture generated by the HLS explorer.

Step 2: Fault modelling: This step takes as input an RTL code and a gate-level netlist generated in the first step and performs stuck-at-fault simulations for every possible stuck-at-fault location. For larger gate-level netlist the number of stuck-at-faults might be too large. Thus the number of stuck-at-fault locations can be limited by a user constraint as a percentage of the total points to be evaluated

The number of stuck-at-faults in an RTL code is obviously much smaller than in a gate netlist. The main objective here is to study if the Dmetric obtained from the RTL stuck-at-fault comparison is *usable* when compared against the more accurate gate netlist method.

Step 3: Dmetric and DIMP Computation: This last step computes the Dmetric and the DIMP diversity metrics for all the design pairs generated so far. The results are three different diversity metrics: $Dmetric_{gate}$, $Dmetric_{RTL}$ and $DIMP_{gate}$

The process of identifying joint detectability for each fault pair is tedious with excessive iterative searches throughout the generated reports of both designs under

consideration. In an attempt to reduce the number of iterations, the existing Dmetric formulation has been slightly modified. The modified equation was derived as follows:

$$D = \sum_{(f_i,f_j)} p(f_i, f_j).d_{ij} \tag{5.5}$$

$$= \frac{1}{M}\sum_{(f_i,f_j)} d_{ij} = \frac{1}{M}[1 - \frac{k_1}{2^n} + 1 - \frac{k_2}{2^n} + ... + 1 - \frac{k_n}{2^n}]$$

$$= \frac{1}{2^n M}[2^n - k_1 + 2^n - k_2 + ... + 2^n - k_n] = \frac{1}{2^n M}[M2^n - [k_1 + k_2 + ... + k_n]]$$

$$= \frac{M2^n}{M2^n}[1 - [\frac{\sum_{(f_i,f_j)} k_{ij}}{M2^n}]]$$

$$= [1 - \frac{\sum_{(f_i,f_j)} k_{ij}}{M2^n}] \tag{5.6}$$

This work focuses only on stuck at faults, i.e., stuck at 0 or stuck 1. Thus, every fault point could be stuck at 1 or 0. For example, let us assume design D1 has F1 fault points, and D2 has F2 fault points. Each of these fault points could be affected by S-a-0 or S-a-1. Hence there exist 4 different fault pair combinations for one fault point, i.e., (D1-S-a-0, D2-S-a-0), (D1-S-a-0, D2-S-a-1), (D1-S-a-1, D2-S-a-0) and (D1-S-a-1, D2-S-a-1). Thus, the total number of fault pairs M = 4F1F2.

Finally, the complete flow iterates by updating the pragma mix, driving a cost function that maximises diversity. Every generated design is added to a database of designs against which the newly generated design is compared. The method returns the micro-architecture pair that leads to the highest possible diversity for all the three diversity methods used.

5.2.1 Experimental Result

The experiments to compare all three diversity metrics are carried out using different designs from the synthesisable SystemC benchmark suite (S2Cbench) with designs intended to work on different applications [74]. We choose on purpose smaller benchmarks such that an exhaustive Dmetric calculation could be done. The HLS tool

used is CyberWorkBench v.6.1 from NEC [57] and for logic synthesis Synopsys Design Compiler (DC) [83]. Nangate Opensource 45nm technology is the target technology used. The experiments are conducted on an Intel i7-6700 @3.50GHZ CPU and 16 GB memory, running CentOS 7. The 8 benchmarks used were explored without area and latency constraints to fully understand the strengths of raising the abstraction level.

Fig. 5.3 compares the Dmetric obtained for the gatenelitst ($Dmetric_{gln}$) vs the RTL description ($Dmetric_{rtl}$ for all of the 8 benchmark cases showing the average results at the last entry. The results show that the diversity metrics are very close (diversity range lies between 0.97 and 1.0). Thus, one could conclude that performing the Dmetric calculation at the RTL leads to similar results as compared to the gate netlist description. The main problem is the Dmetric values reported are not equivalent because the stuck-at-faults are not the same. Hence, these results do not completely show the entire picture of the accuracy of computing the Dmetric at the RTL vs.

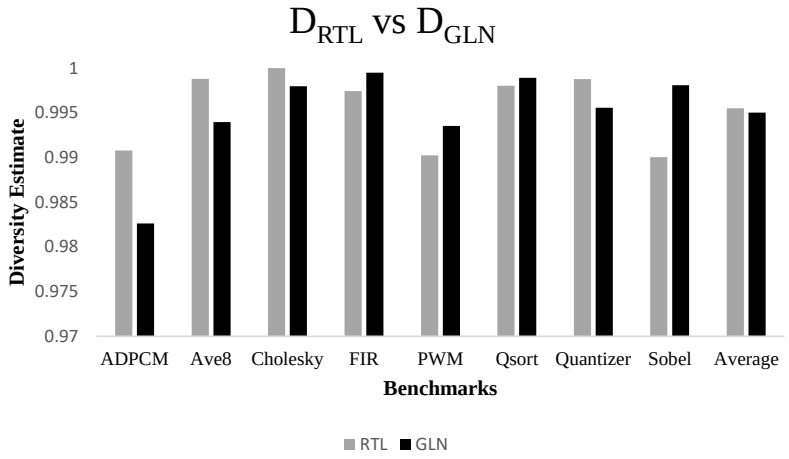

Figure 5.3: DRTL vs DGLN.

To better understand the difference between the results, Table 5.1 shows the most diverse design pairs generated by the two Dmetric cases ($Dmetric_{gln}$ and $Dmetric_{rtl}$). Example, for the ADPCM case, the designs D11 and D12 generated by the framework had the highest diversity when $Dmetric_{gln}$ was used; and design pairs D1 and D5 when

Table 5.1: Overview of design pairs with highest diversity found by the three different diversity estimation method ($Dmetric_{gln}$, $Dmetric_{rtl}$) and DIMP and comparison of the position of the most diverse design pairs compared to $Dmetric_{gln}$

Benchmark	Best Design Pair (GLN)	Best Design Pair (RTL)	Best Design Pair (DIMP)	GLN Ranks of RTL Design Pair	GLN Ranks of DIMP Design Pair
ADPCM	D11-D12	D1-D5	D1-D10	53	39
Ave8	D4-D9	D3-D8	D3-D12	15	41
Cholesky	D2-D10	D1-D2	D10-D3	16	23
FIR	D4-D5	D5-D9	D3-D11	6	58
PWM	D5-D6	D3-D6	D5-D11	7	26
Qsort	D3-D12	D11-D12	D1-D12	19	28
Quantizer	D8-D12	D2-D12	D12-D9	4	3
Sobel	D1-D5	D10-D11	D6-D1	66	41
Average				**23**	**32**

$Dmetric_{rtl}$ was used. When using DIMP design pair, D1-D10 was reported as the most diverse design pair. This clearly indicates that using different diversity metrics leads to different design pairs, where the pair reported by $Dmetric_{gln}$ is the actual best pair as it includes a detailed stuck-at-fault study at the gate netlist.

The last two entries show the position of the most diverse design pair found by using $Dmetric_{rtl}$ and DIMP when compared with $Dmetric_{gln}$ generated ranking order. It could be seen that on average, the most diverse design pairs found by $Dmetric_{rtl}$ and DIMP ranked 23^{rd} and 32^{nd} among the designs found by $Dmetric_{gln}$. Hence, the results reveal that $Dmetric_{rtl}$ leads to better results than using DIMP. However, it is not better than $Dmetric_{gln}$. Moreover, the results are not as accurate as the initial data revealed, and thus, $Dmetric_{rtl}$ and DIMP should *not* be used to find the most diverse system when extremely robust fault-tolerant systems are required. These metrics could nevertheless be used to prune the search space quickly and then perform a more detailed $Dmetric_{gln}$ estimation on the pruned search space.

In terms of runtime, it seems intuitive that the DIMP method has to be faster as no stuck-at-fault simulations are required. Fig. 5.4 shows the running time for each of the three diversity estimation methods in minutes. The last entry shows the geometric mean between all three methods to account for the size difference

between the different benchmarks. From the results, it can be observed that the DIMP method is the fastest, followed by $Dmetric_{rtl}$ and $Dmetric_{gln}$. The differences between $Dmetric_{rtl}$ and DIMP depend on the number of input vectors and stuck-at-fault pairs. If these are small, then the runtime is similar as DIMP requires a full logic synthesis, which $Dmetric_{rtl}$ does not. On average (geomean), DIMP is 8× faster than $Dmetric_{gln}$ and 4× faster than $Dmetric_{rtl}$, and $Dmetric_{rtl}$ is on average 4×faster than $Dmetric_{gln}$.

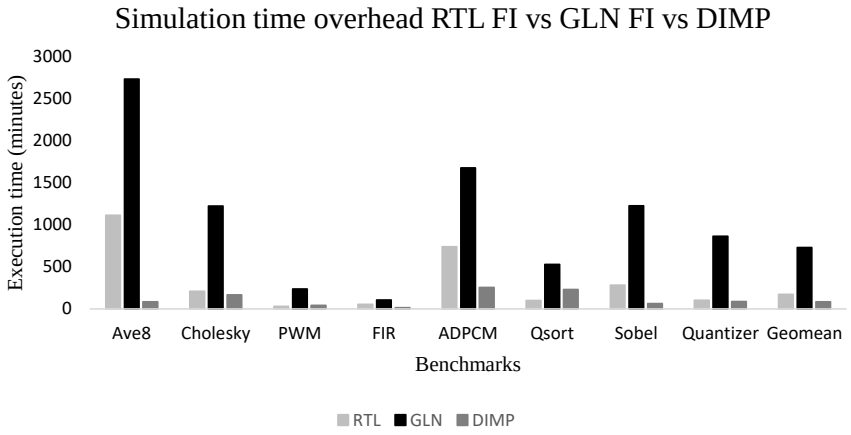

Figure 5.4: Simulation time overhead RTL FI vs GLN FI vs DIMP.

As shown the runtime is still significant, especially considering that the benchmarks used were small. For larger benchmarks, a faster as well as accurate method is required. Thus, the next section presents a predictive-model-based method to find diversity between designs right after HLS such that an HLS DSE can use this fast predictive model to guide it in finding the most diverse design pairs.

5.3 Learning Based Diversity estimation

Raising the abstraction level widens the design search space, which increases the demand for a quick diversity analysis technique as it is intractable to calculate Dmetric

at the gate netlist or RT-level for larger designs. Much work has been recently done in the use of predictive models for EDA, showing very good results. Thus, it seems to be a promising new path to quickly compute different micro-architectures' diversity, especially considering the ultimate goal of building an automated HLS design space explorer.

The subsequent sections explain in detail the two phases of the proposed method followed by a modified diversity driven design space explorer that prunes out the design pair with the highest diversity[85].

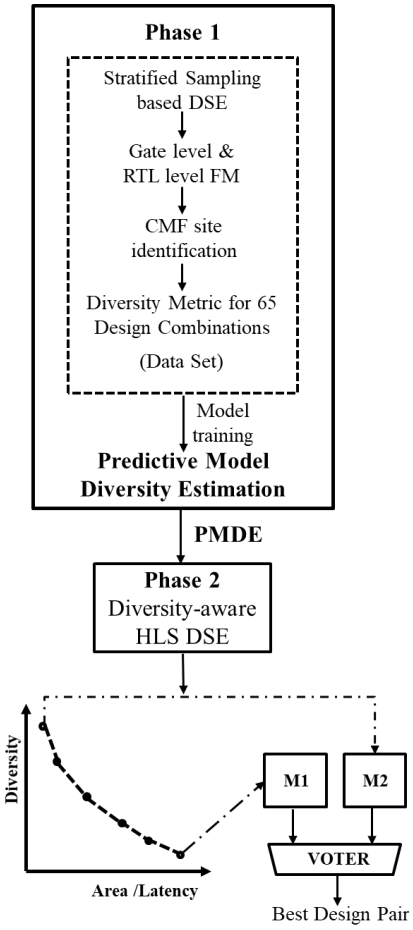

Figure 5.5: Complete proposed flow overview of the proposed scheme composed of two phases. Phase 1: Generation of predictive model for diversity estimation. Phase 2: Use of model for HLS design space exploration.

5.3.1 Methodology Overview

The proposed predictive model comprises of two phases. Fig. 5.5 shows an overview of the complete flow of the proposed method where phase 1 samples the search space and computes a full $Dmetric_{gln}$ for these initial designs while phase 2 generates a predictive model such that once it reaches a specific level of robustness (error), the explorer continues finding the two most diverse micro-architectures using this predictive model instead of having to compute $Dmetric_{gln}$. This process significantly speeds up the entire search process.

The model uses as predictors (labels) the different results reported after HLS, e.g., FSM states, registers, latency, area, muxes, FUs which have to predict the *Dmetric* value. Phase 1 takes in an input behavioural description and phase 2 outputs the design pair with the highest diversity index $DV_{i,j}$. The proposed method also outputs a trade-off curve with all the design pairs that possess unique diversity vs area and/or latency trade-offs that fall within the explorer's cost function.

Phase 1: Predictive Model Generation

Fig. 5.6 elaborates the work flow carried out in phase 1 of the methodology. Phase 1 can further be split into 4 steps where step 1 to step 3 is stratified random sampling, Synthesis and Diversity estimation.

This phase tries a library of predictive models called with predictors from the HLS synthesis report that holds information regarding the area of different logic resources, number of registers, number of states involved in the design and their corresponding *Dmetric* estimates. The end of this stage is the predictive model diversity estimator (PMDiversity) and a tool generated confidence interval. All the above steps are repeated until the model generator reports at least 95% confidence in its predictions.

Machine learning algorithm such as ExtraTree, GaussianProcesses, Ibk, LeastMedSq, LinearRegression, LWL, M5P, M5Rules, RandomTree, RBFNetwork, REP-

Tree and SMOreg were used. The training phase is repeated for every new behavioural description as different methods can work better for different designs. The Mean Absolute Error (MAE) and correlation coefficient that indicate the model's predictability are also reported for each predictive model, and the model that converges faster is finally chosen. MAE is defined as the sum of error values over the total number of samples under consideration. A low value on the MAE estimate means that a better correlation exists between the predicted and the golden outputs. In eq.(5.7), n represents the total number of samples while y_c and y_o are the predicted output and the golden output, respectively.

$$MeanAbsoluteError(MAE) = \frac{1}{n}\sum_{i=0}^{n} |{y_c}^i - {y_o}^i| \tag{5.7}$$

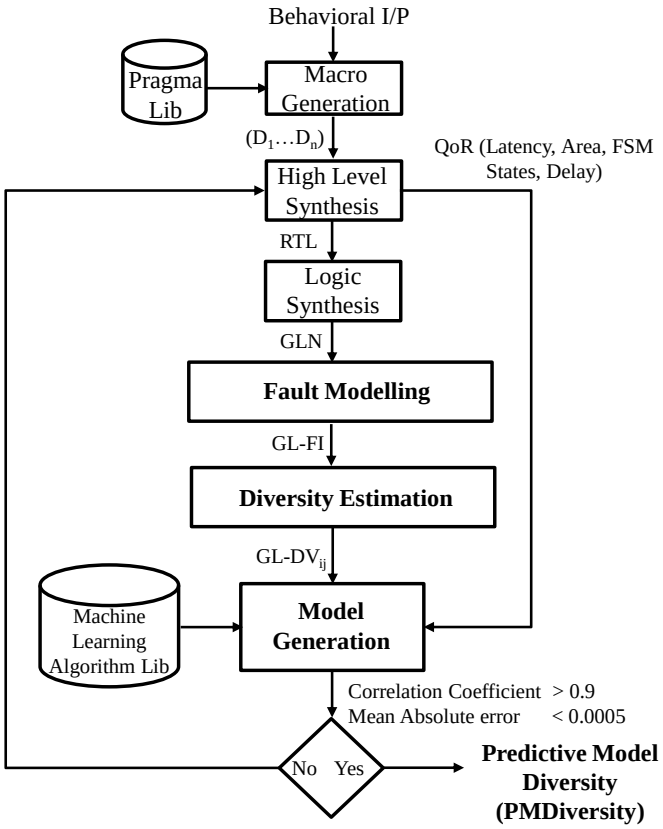

Figure 5.6: Training of machine learning for cost function generation.

Table 5.2: Comparison of Mean Absolute Error (MAE) and Correlation Coefficient of different algorithms on two benchmarks FIR and Quantizer.

Algorithms	FIR		Quantizer	
	Mean Error Rate	Correlation Coefficient	Mean Error Rate	Correlation Coefficient
Extra Tree	2.679	0.703	4.368	0.860
Gaussian Processes	2.661	0.765	5.744	0.863
Least Med Sq	2.554	0.734	4.084	0.883
Linear Regression	2.143	0.854	4.304	0.889
M5P	2.036	0.804	2.864	0.940
M5Rules	1.857	0.862	2.704	0.965
Random Tree	2.429	0.610	3.408	0.933
RBF Network	3.357	0.470	6.288	0.619
REP Tree	3.107	0.439	3.536	0.917
SMO reg	2.268	0.835	3.376	0.933
IBK	2.982	0.668	3.344	0.940
LWL	2.786	0.742	3.632	0.933

The predictive power for certain problem domains might be higher in spite of an unsatisfactory MAE value. Thus, it is appropriate to use MAE along with correlation coefficient measures. The correlation coefficient value tends to 1 if the predicted value correlates with the actual value, while it tends to 0 for predictions that deviate from the expected value.

Table 5.2 shows the MAE and correlation coefficients of two different benchmarks (FIR and Quantizer) for all the predictive methods used. It can be observed from the data that in most cases, M5Rules generates the best predictive model for these benchmarks. M5Rules is a supervised learning algorithm that uses a separate-and-conquer strategy wherein every iteration builds a model tree and sets a rule out of its best leaf. These are regression tree-based predictors where the predictions are derived from if-then-else conditions by segregating the data set into small groups with a simple regression model for each of them. The newly generated PMDiversity is used to drive the automated HLS design space exploration to find two micro-architectures that are highly diverse from one another.

5.3.2 Phase 2: Multi-objective HLS Design Space Exploration

Phase 2 of the proposed model uses samples from phase 1 to add diversity as another objective to area and latency. Fig. 5.7 shows an overview of the proposed diversity-aware HLS DSE. For exploration in this case, we make use of Genetic Algorithm (GA) as it has shown to lead to excellent results in multi-objective optimisation problems like this one. In an effort to minimise the cost between exploration runs, the cost function is set to maximising diversity, i.e., $C_{di,dj} = \frac{1}{DV_{i,j}}$, where $DV_{i,j}$ is the diversity estimate between two design D_i and D_j.

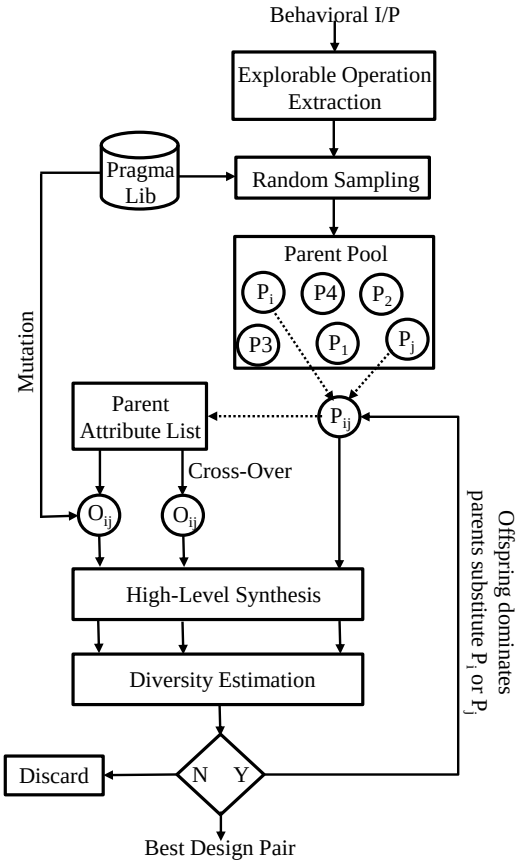

Figure 5.7: Overview of proposed Genetic Algorithm HLS Design Space Explorer for testing ML model.

The best design pair is said to satisfy area, latency and diversity constraints

thus for the above exploration, the cost function is set to minimise area and delay while maximising diversity, i.e $C = \alpha A + \beta L - \gamma DV$. The resulting design pairs are Pareto optimal design combinations that fall into this three-dimensional search space. The best design pair could then be easily chosen based on the requirements of its application. This phase is split into 3 steps as follows.

Step 1: Extraction of Explorable Operations: An input behavioural code has several operations that could be explored to generate unique micro-architectures. It is for this purpose that we parse a given behavioural C code to extract explorable operations. Synthesis directives used here control operations such as array, loops and functions implementation.

Step 2: Parent generation: The second step of phase 2 is identification of parents by randomly assigning each parent with a unique list of pragmas. Each assigned attribute is assumed to be a gene that could be inherited by their offspring. The pragmas in this work include arrays= register, expand, logic, RAM, ROM, loop= no, partial, all, fold and func= goto, inline. Meanwhile, the design points generated during phase 1 of the flow could also be used to pick parents from the pool.

Step 3: Genetic Algorithm based DSE: Two parents are randomly selected from the pool of parents. Let us assume P_i and P_j are the chosen parents. The list of attributes that corresponds to these parents are combined with a random cut-off point. The newly generated attribute list also possesses mutated genes that are attributes which are not acquired from their parents. They are selected from the attribute library. The rate at which this mutation occurs could be chosen and, in our case, is set to mr = 0.1. From the new attribute list, a mutated offspring (O_{ij}) is generated and synthesized using the HLS tool leading to a micro-architecture with unique features. The new design pair combinations are $DPair_{P_i,O_{ij}}$ and $DPair_{P_j,O_{ij}}$. The diversity of these design pairs is measured using the PMDiversity. An offspring substitutes one of its parents when the newly generated design pair has a lower cost function compared

to the parent combination. Each of the offsprings generated is synthesized using maximum functional units in order to parallelise the micro-architecture to a possible extent with the available pragma list. At the same time, it is also synthesized with minimum function unit implementation with maximum resources being shared. The algorithm continually tries to reduce the cost function until N number of offsprings fail to improve any of their parents. In this work, N had been set to 10.

5.4 Experimental Setup and Results

This section presents the experimental setup used to reproduce results for all the three methods being compared followed by an experimental results section that showcases the results retrieved from the comparison.

5.4.1 Experimental Setup

The efficiency of the proposed method is verified using 8 different benchmarks from the open source synthesizable S2C benchmark suite [74]. The benchmarks used are Ave8, ADPCM, Cholesky, FIR, PWM, Qsort, Quantizer and Sobel. The above-mentioned benchmarks are all small to allow finding the optimal solution exhaustively and hence be able to fully characterize our proposed approach.

The HLS tool used in this work is CyberWorkBench v.6.1 from NEC and logic synthesis is done using Synopsys Design Compiler (DC). Nangate Opensource 45nm technology is the targeted technology library. An Intel i7-6700@3.50GHZ CPU and 16 GB memory, running CentOS 7 were used to conduct the experiment. The gate-level netlist of all 8 benchmarks were injected with stuck-at-0 and stuck-at-1 faults and simulated using ModelSim. For implementing the machine learning phase, WEKA has been used.

To better characterize the strength of the proposed method ($PMDiversity$), we

compare it against the three other methods introduced previously ($Dmetric_{rtl}$,$Dmetric_{gln}$ and DIMP). All the methods being discussed are diversity estimation techniques used to calculate the degree of diversity between the design pairs generated by the proposed GA-based HLS DSE. In order to decrease the randomness induced by the GA based explorer, each exploration is repeated 5 times and the average results are reported.

All the above metrics guide our exploration process to generate the design pair with the highest diversity. To better understand the influence of diversity measurement technique on the design pair chosen, we generate 65 micro-architecture pairs for each benchmark and measure the runtime required by each method to generate the result.

5.4.2 Experimental Results

Fig. 5.8 shows the experimental results in terms of the diversity values reported by each of the four methods for the design pair with the highest diversity. As indicated previously, this value by itself does not mean anything as the design pairs can be very different to the design pair reported by $Dmetric_{gln}$, which is by definition the most accurate, but also slowest of all the methods.

Table 5.3 shows the *distance* between the design pair reported by $Dmetric_{gln}$ and the other three methods. It can be observed that our proposed PMDiversity method on average is only 1.5 design pairs away from the optimal design pair, while the other two methods are 20 and 32.4 design pairs away for $Dmetric_{rlt}$ and DIMP respectively. This indicates that our proposed method works very well.

Finally, Table 5.4 also highlights one of the big advantages of our proposed ML-based diversity estimation method. It is on an average 10.5× faster than using $Dmetric_{gln}$, 4× faster than $Dmetric_{gln}$ and 8.77× faster than using DIMP.

In summary, we have shown that our predictive machine learning based diversity

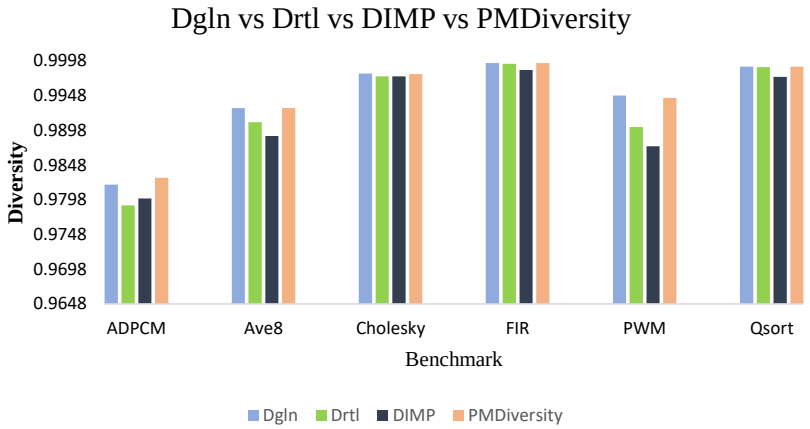

Figure 5.8: Diversity result comparison among all four diversity estimation methods: $Dmetric_{gln}(D_{gln})$, $Dmetric_{rtl}(D_{rtl})$, DIMP and PMDiversity.

Table 5.3: Best design choice selection ranks for three diversity estimation methods compared to baseline ($Dmetric_{gln}$).

Benchmark	**Ranking**		
	Dmetric$_{rtl}$	**DIMP**	**PMDiversity**
ADPCM	15	39	3
Ave8	51	41	1
Cholesky	7	23	2
FIR	6	58	1
PWM	7	26	2
Qsort	19	28	1
Quantizer	4	3	1
Sobel	51	41	1
Average	**20**	**32.4**	**1.5**

estimator outperforms the Dmetric estimation at the RT-level and DIMP, while being much faster in finding the design pair with the highest diversity estimate.

5.5 Summary and Conclusion

In this chapter, we have proposed methods that help mitigate CMFs when compared to the existing methodologies by leveraging the strengths of High-Level Synthesis. In particular we have proposed an automatic method based on HLS Design

Table 5.4: Runtime comparison between all methods compared to baseline ($Dmetric_{gln}$).

Benchmark	**Speedup**		
	Dmetric$_{rtl}$	**DIMP**	**PMDiversity**
ADPCM	2.46	3.69	12.58
Ave8	2.27	16.89	12.24
Cholesky	5.83	5.94	11.37
FIR	1.95	2.06	7.52
PWM	1.95	2.06	7.26
Qsort	5.35	2.44	10.54
Quantizer	8.38	13.92	10.67
Sobel	4.37	23.15	11.76
Average	**4.00**	**8.77**	**10.5**

Space Exploration to tactically prune out best design pairs to be used for fault tolerance applications. We have analysed two main diversity indicators used in literature (Dmetric and DIMP) and have evaluated a new version of Dmetric based on RTL stuck-at-fault simulations. We proceed by analysing the accuracy of DIMP copmared to Dmetric. Finally, we introduce a fast predictive model based diversity estimator that considerably accelerates the process of finding design pairs with high diversity.

Chapter 6

Functional Locking of Behavioural IPs

The previous chapters addressed the issue of fault-tolerance in heterogeneous MPSoC's against random SEU and CMFs. This chapter deals with protecting a dedicated hardware circuit from being illegally copied, reverse engineered or from any malicious alterations.

The protection of Intellectual Property (IP) has emerged as one of the most serious areas of concerns in the semiconductor industry. Logic locking has led to good results, with little overhead against IC over-production and IP piracy by untrusted end-users or foundries. Fig. 6.1 shows the complete IC design flow, including the insertion of a locking mechanism. Unfortunately, new attack techniques appear each time a new locking mechanism is presented to exploit its weakness. Some include path sensitisation attacks and more recently SAT-based attacks. This work proposes a functional locking mechanism for BIPs that exploits the internal structure of behavioural descriptions when synthesised using HLS, in particular the generation of a Finite State Machine (FSM) and a datapath. Our proposed locking mechanism breaks the key into multiple pieces, and each applied at a unique FSM state. Ap-

plying the incorrect key at any of the states leads to an invalid result. Breaking our proposed locking mechanism is much harder than previous work as the attacker would need to try all possible permutations of keys to find the correct key, thus making this virtually unbreakable. Moreover, we insert the key at the BIP itself, which implies that the locking logic is shared with the BIP logic, making it more robust against removal attacks. Experimental results show that the proposed locking mechanism works very well with minimal area and delay overheads for different key sizes.

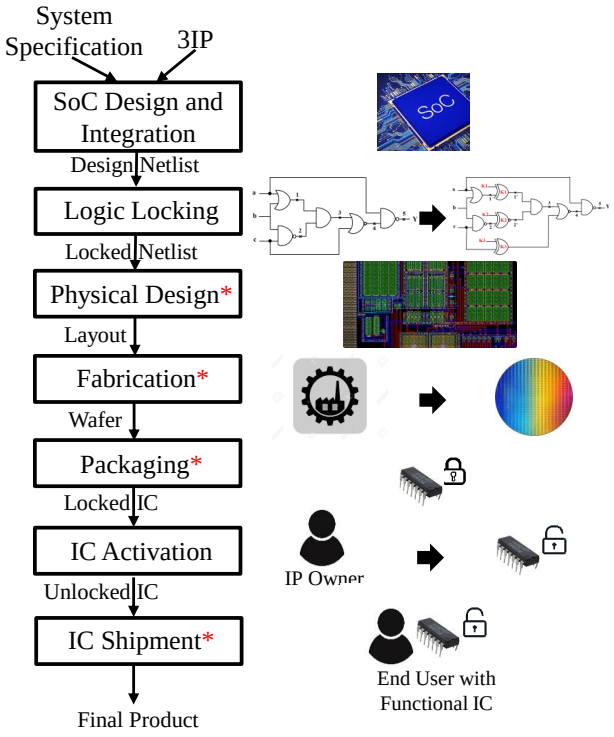

Figure 6.1: IC design flow that incorporates logic locking to protect an IC from supply chain vulnerabilities represented in *.

In addition, to reduce the complexity of designing these heterogeneous systems and these accelerators, companies have started to embrace HLS, also called C-based VLSI design. Finally, most IC design companies are now fabless, which implies that they have to expose the complete chip to a potentially untrusted foundry thousands of

miles away. These offshore foundries are located far away from the design companies that make it easy for attackers to insert malicious circuits into the original design.

Thus, it is important to protect these IPs created by these companies and in particular, the BIPs that lead to unique differentiating products in the form of hardware accelerators. Functional Locking is a promising technique to thwart supply chain attacks. The main idea behind logic locking is to add additional key enabled circuitry to the original circuit to be protected, thus preventing the correct functional execution of an IC if the correct key is not applied. The key is typically stored in a tamper-proof memory and cannot be retrieved, even when acquiring the IC lawfully from the market. This approach has shown to offer a certain level of protection against bruteforce attacks, yet has also been shown vulnerable to intelligent attacks, such as the clever use of Boolean Satisfiability (SAT) solvers [81]. This work proposes a new functional locking mechanism that exploits the fact that any circuit synthesised using HLS is built around a controller (FSM) and datapath to address the shortcomings of the existing techniques.

The following sections elaborate some of the existing logic locking techniques and attack strategies used by attackers to unlock any secure IP.

6.1 Logic Locking Overview

VLSI design companies often focus on developing these dedicated accelerators, dedicating a significant number of resources to optimise them fully. Therefore, it is extremely important for them to protect their Intellectual Property (IP) from being copied or illegally distributed. Because of the importance of this topic, a plethora of approaches have been proposed ranging from functional locking [73, 14] to split manufacturing [67] and diverse design obfuscation techniques [101]. Out of all the techniques, functional locking seems the most promising. In functional locking, a de-

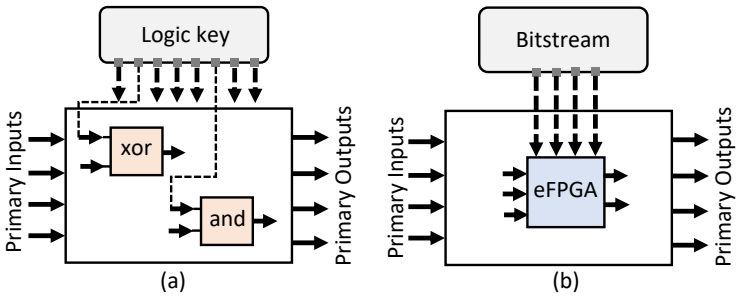

Figure 6.2: Overview of main functional locking mechanisms. (a) Traditional locking through additional gates and (b) locking through omission by mapping portions of the design onto an eFPGA.

sign is modified such that it does not comply with the original specification if a valid key is not applied to the design. The result of applying an incorrect key can range from the functional incorrectness of the output to the reduction of the performance. The main idea in functional locking is to make it harder for an attacker to find the correct key to unlock the correct functionality of the circuit. Fig. 6.2(a) shows a simple example where an XOR and an AND gate are added to the circuit. In this particular case, the logic key for these two gates needs to be set to 0 for the XOR gate and 1 for the AND gate to guarantee the correct behaviour of the circuit.

One relatively new approach to functional locking is, locking through *omission*. In this case, a portion of the design is mapped onto an embedded FPGA (eFPGA) as shown in Fig. 6.2(b). In this case, the key of the circuit is the actual bitstream that correctly configures the eFPGA [80]. This technique is more potent than traditional locking because a part of the circuit is entirely removed from the design as opposed to traditional techniques where the locking mechanism is still embedded in the circuit when sent to the fab. The main problem is the extremely large overheads associated with it in terms of area, power and timing degradation. Thus, traditional methods that are robust enough and lead to small overheads are necessary.

The design house, in this case, is assumed to be trusted while the offshore foundry, testing facilities and the end-users are considered to be stages where an IC is prone

to be misused. The secret key access is restricted to the IP owner alone. This makes the attack on an IC complicated as the correct functionality is restored only when the correct key is applied. For an attacker to identify a secret key, he/she has to critically analyse the structure of a locked netlist to single out a suitable heuristic to determine the hidden key. The time taken by an attacker to retrieve a key determines the strength of the locking mechanism. The IP owner activates the IC before it reaches the end-user.

6.2 Logic locking to Defend against Hardware Attacks

This section briefly describes the capabilities of logic locking in thwarting hardware attacks.

IP theft, Counterfeiting and reverse engineering: Foundries these days are individual entities that focus only on IC fabrication for several semiconductor companies, giving them access to sensitive layout data. It has thus become mandatory to protect an IC using logic locking. In cases such as IP theft or functionality identification by reverse engineering, the attacker is left with an altered design. Counterfeiting strongly depends on the attackers reverse engineering capability. However, the gates used for locking either replace a functional buffer or inverter or they are additionally added, thus making the attack process strenuous as the structure would not reveal much data regarding the lock implemented. One must find an effective method to identify the key in a short period of time, which itself is a tough task [88]. Thus, IP theft and counterfeiting post reverse engineering needs more probing to single out the secret key.

Malicious altercations: Intentional design changes made by attackers within the company act as hidden front doors that could potentially leak secret internal data.

Logic locking helps in mitigating the insertions of Trojan by altering the signal transition probabilities, making it difficult for an attacker to identify cloaked insertion points.

Overproduction: Offshore foundries or rogue employees in the foundry can fabricate more ICs than the client's requirement. These extra ICs, when sold in the black market, fetches the attackers an enormous amount of profit, disrupting the mainstream sales. By locking a design using logic locking techniques, the attacker fabricates a design with gate modifications embedded within. Thus, the manufactured IC is functional only after activation.

6.3 Lock Insertion Techniques

Logic locking techniques could either be combinational if XOR/XNOR gates, AND/OR gates or multiplexers are inserted [64, 66, 68, 73, 19]; or sequential if lookup tables (LUT) or finite state machines (FSM) are used [5, 33, 14, 36]. The wrath of a lock depends on the key gate insertion techniques. The three commonly used methods are (i) random logic locking (RLL), (ii) fault analysis-based logic locking (FLL), and (iii) strong interference-based logic locking.

RLL is an insertion technique where the key gates are inserted at random locations throughout the netlist. The key gates inserted may or may not be placed at critical positions. This ambiguity arises due to the randomness in the insertion point. Thus, the output corruption probability is reduced where the correct outputs are reflected for several wrong keys [73].

FLL was introduced to overcome the shortcomings of RLL. Fault injection-based analysis of a netlist helps identify pivotal insertion points that can increase the probability of output corruption. The corruption ability is measured by estimating the Hamming distance between the golden output and the corrupted output. A larger

Hamming distance indicates a higher degree of error. FLL does not survive the sensitization attack. Sensitization attack is an attack means by which an attacker applies a selected input that lets one or more bits of the secret key surpass the gates in its path and reach the primary output bits, thus revealing a bit of the secret key [68].

SLL was introduced to withstand sensitization attack. The key gate insertion is performed in such a way that two or more gates interfere with each other, thus not letting the attacker sensitize a path without assuming values for these keys bits at a time. This interference causes an inter-dependency between these key gates, thus making it difficult for an attacker to identify the secret key without exhaustively trying all permutation combinations (brute-force) [66].

Sensitization attacks can be thwarted by SLL but stronger attack like the Boolean satisfiability attack or SAT attacks can easily break all the above-mentioned locks and its improvised variants [82]. Attack methodologies that are being used these days are explained in detail in the next section.

6.4 Existing Attack Strategies

Attack strategies that exist in today's hardware security environment have increased the need to develop stronger locks to protect IC from being misused. Some of the strong attacks that exist are algorithmic, approximate, structural/removal and side-channel attacks.

Algorithmic Attacks: The vulnerability of a locked design depends on its algorithmic weakness. Extraction of the secret key by any means will help an adversary retrieve a complete design without any hindrance. The retrieved design will be an exact replication of a working IC, i.e., if key k and input i are applied to a locked circuit then L(k,i) = F(i), $\forall i \in I$. Sensitization attack [66], SAT attack [82] and circuit partitioning attack [43] are examples of algorithmic attack.

Approximate Attack: Approximate attacks aim at compound logic locking techniques where a low-corruptibility technique such as Anti-SAT is coupled with a high-corruptibility technique like SLL or FLL. Approximate attacks are capable of identifying the functionality of the locked circuit to an extent where only a few of the output cases fail. These retrieved designs are approximately equal to the original design with some missing bits that produces output variation for certain set of inputs. Examples of these attacks are AppSAT [32] and Double-DIP [78].

Structural/Removal Attack: An attacker who has easy access to a reverse engineered netlist can identify a locking unit or an error injection block by carefully examining its structure. One could bypass/remove this block that intends to protect the design. The result of such isolation can be represented as follows R:L(i, k) $\rightarrow H(I)$ such that H(i) = F(i), $\forall i \in I$. The recovered netlist produces the same outputs for every given input. Bypass attack is an example of structural attack [94].

Side-Channel Attacks: Side-channel attacks make use of a circuit's physical properties such as power and timing characteristics [97]. The timing graphs or power details reveal a great amount of information, giving way to an attacker's curiosity. Differential power analysis and desynthesis attacks are good examples of side-channel attack [34].

The ability of SAT attack makes it a strong attack strategy capable of breaking every known lock that existed before its release. Thus, in this work, we aim at thwarting all previously known attack while keeping SAT and removal attacks primarily in mind.

SAT Attack

Boolean satisfiability attack or SAT attacks when introduced were capable of breaking all the logic locks that then existed. The idea behind a satisfiability attack is to reduce the key search space iteratively using a SAT solver [82]. SAT attack holds good for

combinational locks. Given that an attacker has a reverse engineered netlist and a functional IC, an oracle-guided keyspace pruning retrieves the secret key within a short time span.

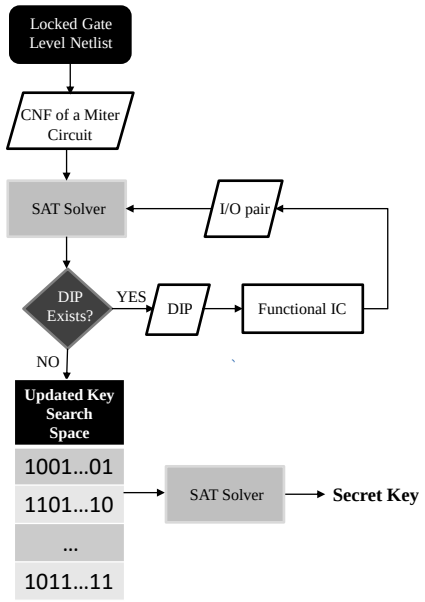

Figure 6.3: SAT attack flow.

A threat model assumed for SAT attack includes a reverse-engineered gate-level netlist of a locked IC, functional IC obtained lawfully from the market and a strong SAT solver like Lingeling [44] or Minisat [84]. Fig. 6.3 elaborates the flow of SAT attack that prunes out incorrect keys to unravel the hidden secret key. The unfolding of SAT attack could be explained step by step as follows.

Step1: Using the gate-level netlist of a locked IC, an adversary builds a miter circuit as shown in Fig. 6.4. Two copies of a gate-level netlist O_1 and O_2 are XORed, and all the XOR outputs are ORed to output a high. K_A and K_B correspond to the keys applied to modules O1 and O2 while the primary input is common to both the modules.

Step 2: SAT solvers take in inputs in the CNF format. CNF or Conjunctive Normal

Form means the entire functionality of a miter is expressed as a conjunction (AND) of clauses where each clause is a disjunction (OR) of literals.

Step 3: The given CNF is satisfiable (SAT) when output of the XOR gate shown in Fig. 6.4 evaluates to 1. Every iteration identifies a differentiating input pattern (DIP). A DIP is a primary input that when applied to O_1 and O_2 with two different keys K_A and K_B produce different outputs. This identifies that only one of the two keys could possibly be a secret key. With the help of a functional IC, the output corresponding to the identified DIP is retrieved to eliminate the wrong key.

Step 4: The key search space initially consists of 2n keys for an n-bit key. The SAT solver iterates if the CNF formula holds good.

Step 5: For each IO pair identified, the CNF formula is updated with a new clause to help identify other DIPs to eliminate furthermore keys. When the SAT solver reaches a point where the CNF formula fails to satisfy, the secret key could be recovered.

6.5 SAT attack resistant logic locking techniques

SARLock

SARLock was proposed to secure a locked logic from being attacked by SAT solvers. This lock makes use of additional circuitry such as a comparator, masking unit and

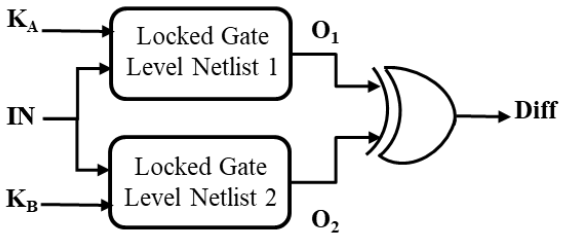

Figure 6.4: Miter circuit used to identify DIPs.

an XOR gate. Fig. 6.5 shows the structure of a SARLock. It could be understood that the strong point of a SAT attack is its ability to rule out keys that are not coinciding with the output of an oracle. Table 6.1 clearly indicates that the SAT attack has a best-case as well as a worst-case scenario. The distinguishing input keys in a best-case scenario will eliminate all the incorrect keys, or a number of incorrect keys in one go making it easier for the attacker to identify the correct key. A worst-case scenario is when the applied input pattern eliminates only one key or does not distinguish a wrong key. SARlock aims at increasing the complexity of identifying the correct key by letting the attacker eliminate only one key at a time. [98].

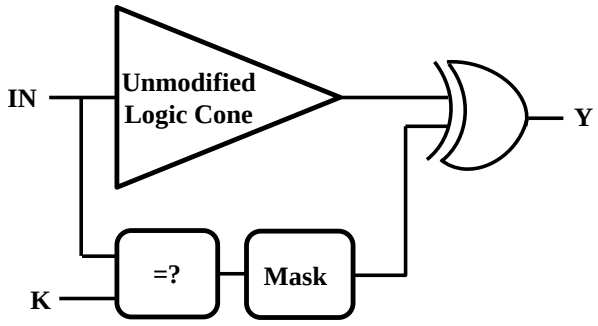

Figure 6.5: SARLock.

The comparator compares the input IN with the secret key from the memory. The masking unit produces a flip for every input-key combination except for the combination where the key and the input are the same. Table 6.1 shows the working of this lock where K2 and input 3 combination does not flip delivering the correct output. The masking unit maintains the functionality intact while letting an attacker identify only one wrong key at a time. This means that for each applied input pattern, only one wrong key could be ruled out. Thus, for a K bit key $2^K - 1$ DIPs are required.

Table 6.1: Input and output combinations of a SARlock for different keys [60]. The black entries represent a flip generated when the comparator shows a mismatch. The grey entries represent an input-key combination that produces the correct output value for the given circuit.

No.	Input	Output	Output Y for different keys							
	abc	Y	K0	K1	K2	K3	K4	K5	K6	K7
1	000	0	1	0	0	0	0	0	0	0
2	001	0	0	1	0	0	0	0	0	0
3	010	0	0	0	0	0	0	0	0	0
4	011	1	1	1	1	0	1	1	1	1
5	100	0	0	0	0	0	1	0	0	0
6	101	1	1	1	1	1	1	0	1	1
7	011	1	1	1	1	1	1	1	0	1
8	111	1	1	1	1	1	1	1	1	0

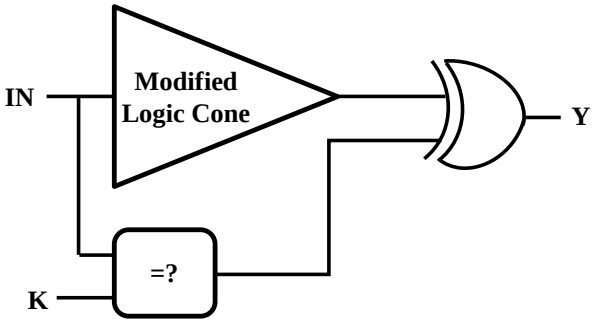

Figure 6.6: TTLock.

Vulnerabilities of SARLock This SAT resistant lock succumbs to a structural attack called the removal attack. The structural analysis of a reverse-engineered netlist would reveal the functionality of the original circuit as the logic cone implementation is unmodified. The comparator, mask and the XOR gates are all externally added to secure the circuit but do not protect the original unmodified circuit from being identified by an attacker.

Table 6.2: Input output combinations generated by a TTLock for different keys. The black entries represent a wrong value while the grey entries represent a input-key combination that generates the correct output. The grey entry is called the protected input pattern [99].

No.	Input	Output	Output Y for different keys							
	abc	Y	K0	K1	K2	K3	K4	K5	K6	K7
1	000	0	1	1	1	1	1	1	1	1
2	001	0	0	1	0	0	0	0	0	0
3	010	0	0	0	1	0	0	0	0	0
4	011	1	1	1	1	0	1	1	1	1
5	100	0	0	0	0	0	1	0	0	0
6	101	1	1	1	1	1	1	0	1	1
7	011	1	1	1	1	1	1	1	0	1
8	111	1	1	1	1	1	1	1	1	0

TTLock

TTLock is a modified SARLock that aims at overcoming the shortcoming of SARLock. In a TTLock, the comparator is disguised as a restoration unit which works to restore a wrong output bit. Fig. 6.6 represents the structure of a TTLock. Here an input pattern is protected, i.e., for a given key K and an input IN the output bit generated by the circuitry needs a restoration facility to withhold the correct functionality. The logic cone in a TTLock is by itself protected by minimal modification. Table 6.2 shows that for an input 000 and K0, the output generated is correct; while for the same input but the other keys, the output generated vary by two bits. The first bit flip generated as an effort to restore the output when 000 is applied while the other flip generated as an effort to mask the original logic cone.

Anti-SAT

The Anti-SAT logic locking technique mitigates SAT attack and suggests means by which one could overcome removal attack. The structure of this SAT mitigating logic locking technique is shown in Fig. 6.7 [91]. The functions implemented by logic

blocks G and $\overline{G}$ are complimentary. The inputs to these logic blocks are intermediate outputs from selected gates in the original circuitry and corresponding secret key. The AND gate at the end of the structure outputs 0 at all times given that the key applied is same the as the secret key. Integrating this block onto the original circuit via an XOR gate as shown in figure makes the block act as a buffer when the key is correct; while it acts as an inverter when the key is wrong, i.e., the blocks G and $\overline{G}$ are complimentary only for the correct key and for the other cases it may or may not preserve its functionality.

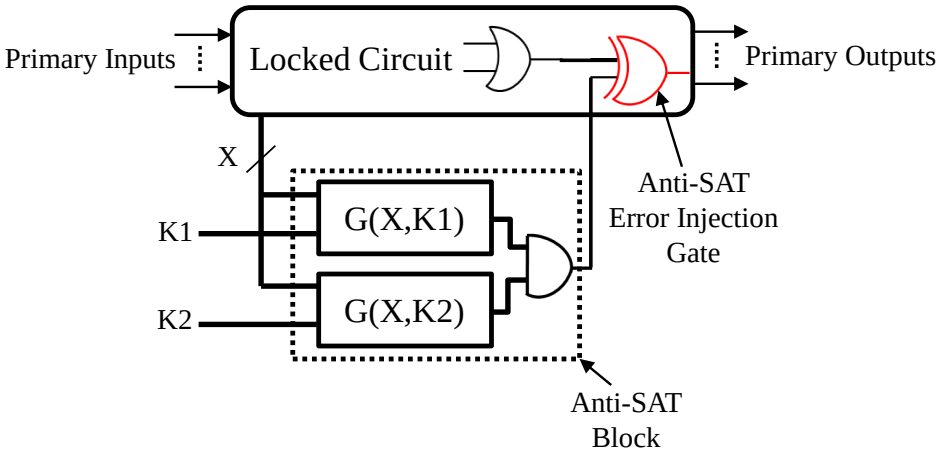

Figure 6.7: Anti-SAT Lock.

Effectiveness of Anti-SAT: Let us assume that for an input vector with n bits p of the inputs cause a type-0 Anti-SAT block to produce 1, making it flip a correct output. Thus, there are $2^n - p$ input vectors that still keep G and $\overline{G}$ complimentary. The sustainability of an Anti-SAT block depends on the p value, i.e., if p is close to 1 or $2^n - 1$ then the SAT iterations required to identify the correct key is higher. The lower bound for a successful SAT attack to decipher a 2^n-key bit can be written as follows $\lambda_l = \frac{2^{2n}-2^n}{p(2^n-p)}$ where $p\epsilon\{1 \rightarrow 2^n - 1\}$. If p=1 , λ_l = 2n. The strength of the system depends upon p but the exponential increase in the number of SAT iterations

based on the key bit size makes it resilient against SAT attack. It is also resilient against removal attack by functional and structural obfuscation [91].

SFLL-HD

The structure of SFLL-HD shown in Fig. 6.8 is being used [56] to better understand the state-of-the-art logic locking technique. The stripped functionality logic locking technique strips the original circuitry in such a way that the introduction of a restoration unit restores the original functionality. This retrieval procedure applies only to a set of protected inputs which are at a Hamming distance h from a secret key K stored in a tamper-proof memory. For given a h and k, $\binom{k}{h}$ number of protected input patterns exist. Let us assume an n-bit input and k-bit key. Stripped functionality logic locking structure shown in Fig. 6.8 has two restore units. The original circuit is modified by hardcoding selected keys that flip the output at Y. These hardcoded keys are the only keys that need a restoration in the latter half of the circuit, converting Y_{fs} back to Y. For these protected input cubes, Y_{fs} is wrong, i.e., even if the restoration logic is identified by structural analysis of the circuit the attacker still cannot recover the original circuit by removing the restoration units as it will leave them with a wrong output Y for all the protected input cubes. SFLL-HD are of three types $SFLL - HD^0$, $SFLL - HD^h$ and $SFLL - HD^{cxk}$. In this technique, when an input pattern applied to the circuit is at a certain predefined Hamming distance away from the secret key, then the unit acts to restore a probable wrong output coming into the final XOR gate. From Fig. 6.8 it could be noted that the logic cone output coming out of FSC is modified for a set of hardcoded key values. As long as the key is correct, the output at Y_{fs} is correct; but for a wrong key, extra errors pop at Y, i.e., when the key is wrong but the Hamming distance is h, then the unit tries to restore an output that does not need rectification. Every input pattern generating a wrong

output belongs to the protected input cube set P, but for every input cube there are 2^{n-k} input patterns. Thus, to identify a key, $P * 2^{n-k}$ iterations are required.

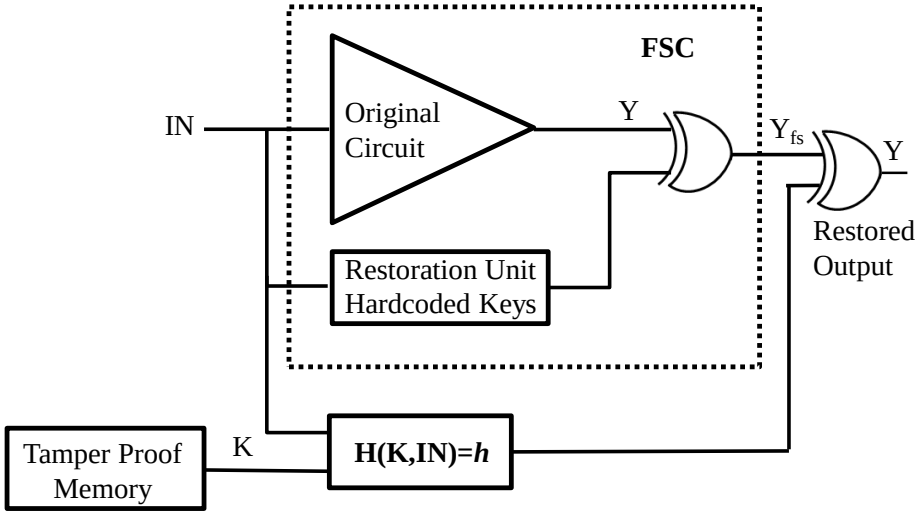

Figure 6.8: $SFLL - HD^h$ logic locking technique[56].

When the Hamming distance h=0, $SFLL - HD^0$ is the same as TTLock. The other versions of stripped functionality logic locking have a number of protected input cubes, making the set P a larger number. The SAT iterations depend on the P value and the Hamming distance h. Increasing the h value increases the number of protected input cubes. The $SFLL - HD^{cxf}$ version of this methodology could be used to protect sensitive parts of a circuitry or to protect an IP. This is flexible as the designer can protect c input cubes by specifying a k-bit key. In [56], it has been proven that the above method thwarts SAT attack, removal attack, sensitisation attack and all other derived attacks; but in [95], it has been proven that structural analysis of a re-synthesized netlist that was generated using design compiler leaves away traces of the changes made to the original circuit, letting the attacker track the protected patterns thus retrieving the original design.

6.6 Efficient Logic Locking for Behavioural IPs

To address the weaknesses of previous locking techniques, in this chapter we propose a new functional locking mechanism that exploits the fact that any circuit synthesized using HLS is built around a controller (FSM) and datapath. We thus split the logic key into *fragments*, each applied at a unique FSM state. This increases the robustness of the circuit to intelligent attacks. In summary, the contributions of this work are:

- Introduces a locking mechanism that increases the security to logic unlocking attacks and removal attacks.
- Extends this locking mechanism to reduce timing issues by making use of advanced HLS tool options.

6.7 Motivational Example

High-Level Synthesis is a process which takes as input an untimed behavioural description (*e.g.* ANSI-C, C++ or SystemC) and generates efficient RTL code (Verilog or VHDL) that can execute it. It performs three main steps: (1) resource allocation, (2) scheduling, and (3) binding. By default, the synthesizer extract as much parallelism as possible from the behavioural description by, for example, fully unrolling all loops. The main idea of creating custom hardware accelerators is to run these more efficiently than in software by processing as much data as possible in parallel at lower frequencies. This might not be the micro-architecture that the designer needs in terms of area, performance and power. Thus, HLS vendors make extensive use of synthesis directives in the form of pragmas to allow designers to guide the synthesis tool in order to generate the desired micro-architecture. Vendors also use dedicated hardware extensions to control the resultant micro-architecture. This work makes use of these type of extensions to check for a valid key during each unique FSM state.

Fig. 6.9 shows a motivational example for a code snippet that computes the average of 8 numbers. Depending on the target synthesis frequency and the delay of the adders, one possible schedule for this program is shown next to it. In this case, the loop that adds the 8 numbers is fully unrolled. The result is a circuit with latency equal to 3 clock cycles composed of a datapath with 4 adders (the adders in cycles 2 and 3 are re-used/shared) and an FSM that generates the control signals for the muxes that control the resource sharing of the 4 adders.

Fig. 6.10 shows our proposed work. It automatically updates the behavioural description based on the time report of the HLS tool that identifies which lines in the given code are scheduled in which clock cycle. This feature is key in our proposed method as it allows to insert a new locking checkpoint in each clock cycle, which in this case is the function 'key_check()'. This is automatically done by instrumenting the source code. There are two major advantages of inserting the locking mechanism at the behavioural source code instead of previous works where it was done at the RTL or gate-level. First, logic locking circuitry is shared and mixed with the rest of the logic, making the circuit resilient to removal attacks. Second, knowing the timing slack available in each clock cycle allows to insert logic locking primitives that do not lead to any timing violations. Hence, the generated circuit will meet the given timing constraint.

Checking a key every clock cycle requires the partition of the original N-bit size key into N/S, where S is the number of FSM states. One other option to further increase the security is to make the key $N \times S$ and apply an N-bit size key to the circuit every clock cycle, making it even harder to break. This makes it much harder to break the key as the attacker needs to find all the permutations of the keys applied each clock cycle. Based on this, the problem that we address in this work could be formulated as:

Problem Formulation: Given a BIP in any behavioural description language

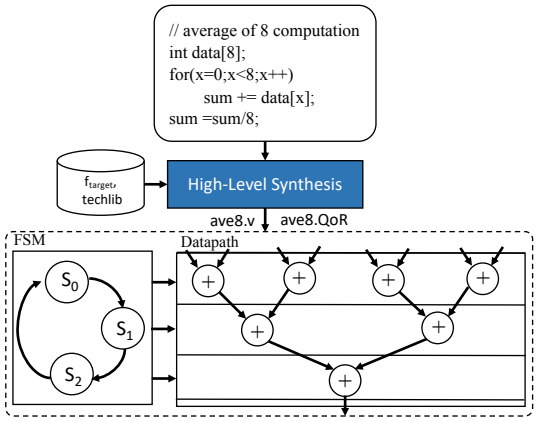

Figure 6.9: Behavioural code snippet (computes average of 8 numbers) and HLS scheduling result.

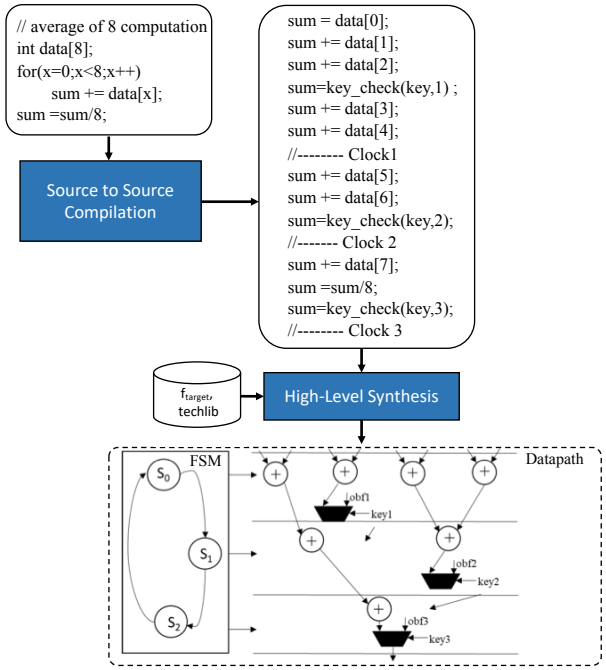

Figure 6.10: Modified C code with locking mechanism and new scheduling result for the newly enhanced C code.

(*e.g.* ANSI-C or SystemC), automatically insert a locking mechanism that prevents obtaining the correct result at the primary outputs, by checking at every state of the FSM the new key, with minimum area and timing overhead.

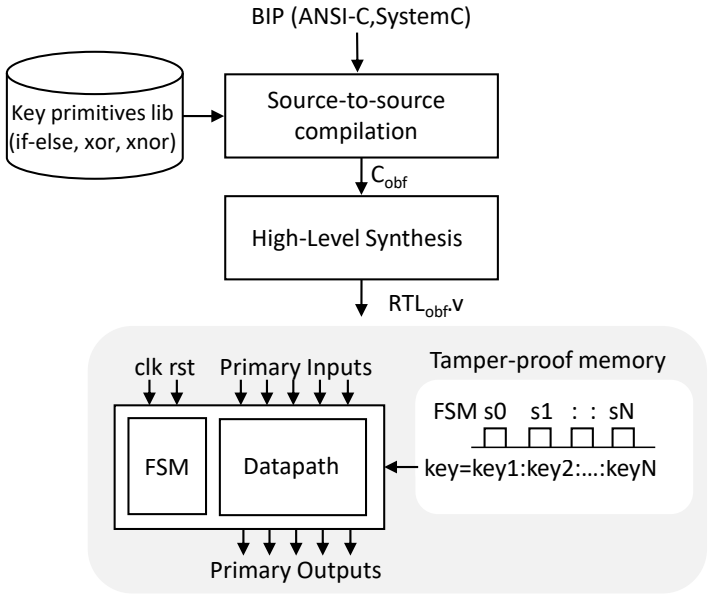

Figure 6.11: Flow overview of proposed method.

6.8 Threat Model

This work considers the same threat model used in most previous functional locking works and assumes that any party involved in the design and fabrication of the circuit is a threat and in particular the fab that can reverse engineer any traditional circuit given as GDSII netlist. We assume that the attacker has access to the layout and significant resources and technical knowledge in reverse engineering. The main goal of the attacker is to reverse engineer the IC to sell it as a pirate copy or to acquire the IP of the IC for its own profit. The attacker also has access to an oracle, a fully functional IC obtained legally from the market. The attacker can also apply input patterns to the locked IC and observe its response.

6.9 Proposed Method Overview

Fig. 6.11 depicts graphically the overall flow of the proposed method. The framework takes as input the untimed behavioural description to be locked and a set of locking primitives (xor, xnor, if-else clauses) and outputs the RTL code of the synthesized

description with the locking mechanism. The method is composed of three main steps described in detail below.

Step 1 Parameter Extraction: This first step performs HLS on the original behavioural description without any locking mechanism. Our method extracts the FSM states and the maximum delay of each of the states after the scheduling phase. This phase also reports the total area, which we will use to report the overhead introduced by inserting the locking mechanism.

Step 2 Source-to-Source Compilation: This second step takes as input the result from step 1, the behavioural description and the locking primitives. Based on the capabilities of the HLS tool used, this step either transforms the behavioural description into a manually scheduled description with the same clock cycles as the latency reported in step 1, or make use of the *allstate* construct mentioned in the motivational example. This second approach is much easier and hence, favoured in this work. Irrespective of the approach, our method has to identify one variable which is being written in each of the states so that the values of these variables can be obfuscated in case that the incorrect key is applied. This is done by parsing a signal access table generated by the HLS tool in step 1. By default, this step randomly chooses one of the signals from the signal table for each state and randomly chooses one of the locking primitives from the database. One of the problems with this approach is that it might lead to timing problems if the delay introduced by the locking mechanism in a particular scheduling step now exceeds the maximum frequency delay specified as input to the HLS process. Thus, to avoid this, our method can skip the states with little timing slack. In this case, any key passed to the circuit at that state is accepted as a correct one. This impacts the security, but guarantees that timing is maintained. Finally, it writes out the new behavioural description with the locking mechanism and adds the key port as primary input (C_{obf}).

Step 3 High-Level Synthesis: This last step synthesizes the new behavioural description (C_{obf}) and generates the new locked circuit as shown in Fig. 6.11. This circuit will only work if the correct key is applied in each of the states. It should be noted that this method only works if the controller generates the same sequence continuously, which is the typical case in many data-intensive applications targeted when using HLS, *i.e.*, DSP, image processing applications. In the case that the application has complex control dependencies, this approach would not work as the key sequence would only be known at runtime. A simple way to deal with this is to add a primary output to the BIP which outputs the current state. The correct key is then applied based on the current state.

6.10 Security Analysis

The proposed locking mechanism leads to a more robust functional locking in two main ways: firstly through the increase in the search space, thus making SAT attacks virtually impossible; and secondly through resource sharing of the locking logic, thus making removal attacks not possible.

Protection against SAT Attack: The proposed lock uses fewer IO pins to retrieve the secret key from a tamper-proof memory compared to the existing techniques. This lets the lock scale better while using much larger keys. The lock introduced here is tough to break using brute force attack while SAT attack would take less time than brute force. It has to be understood that SAT attacks capability will vary based on the obfuscation points. In this proposed technique, we have made random obfuscations, and hence the SAT iterations vary from one benchmark to the other. But in conclusion, it could be understood that when a design is obfuscated using variables with low corruptibility, the SAT iterations will increase and vice versa.

Thus, SAT attack mitigation strength could be enhanced by fault analysis based design obfuscation.

Protection against Removal Attack: In addition, most of the previous locking mechanisms are subject to removal attacks. These attacks identify the circuitry responsible for the logic locking and remove it at the foundry. In our case, because this logic is fully shared with the rest of the circuitry, it is impossible to remove. The nature of the HLS process leads to maximum resource sharing. In resource sharing, a single functional unit is shared among different operations in the source code by inserted multiplexers at its inputs and outputs [65]. The Finite State Machine (FSM) created by the HLS synchronizes all the multiplexers' control signals in order to steer the data through the datapath containing the shared FUs. Thus, the RTL code generated through HLS typically shared most of its resources when possible. This makes our approach extremely robust against removal attacks.

6.11 Experimental Results

Six computationally intensive applications from the open source S2Cbench SystemC benchmarks suite [74], were used to test our proposed flow. These benchmarks comply with the latest Accelera's Synthesizable SystemC subset and hence, could be synthesized with any commercial HLS tools [11, 47, 57] as these all support SystemC. Interp is a 3-stage interpolation filter, decim a 5-stage decimation filter, cholesky is the cholesky decomposition, jpeg a bmp-to-jpeg encoder and aes a block cipher. Table 6.3 shows the number of states of each of the benchmarks when synthesized in default mode, which in our work is equivalent to the total number of keys as there is one unique key per state (in the jpeg case the keys are inserted in the RLE module as it is composed of multiple blocks).

In this work, we use NEC's CyberWorkBench [57], which supports both manual

Table 6.3: Benchmark characteristics

	Benchmarks				
	interp	aes	decim	jpeg	cholesky
keys/FSM states	8	13	20	37	71

scheduling and C-extensions including *allstates*. This last mechanism is used in this work to check for the correct key in each cycle. The target synthesis frequency is set in all cases to 200MHz (clock period of 5ns), and Nangate's 45nm opencell technology library is targeted. All the experiments were executed in default mode, where a unique key is checked every FSM state. Synopsys Design compiler (DC) is used to measure the area and delay overheads of the locked circuit vs. the original unlocked one. Based on the suggestion by [72] which compares different SAT solvers to attack k-obfuscation problems like the one in this work, we use Lingeling solver [44], which has shown to lead to a good balance in terms of execution time and memory usage. We restricted the running time to 2 days.

Figs. 6.12 and 6.13 show the area and delay overheads introduced by our method

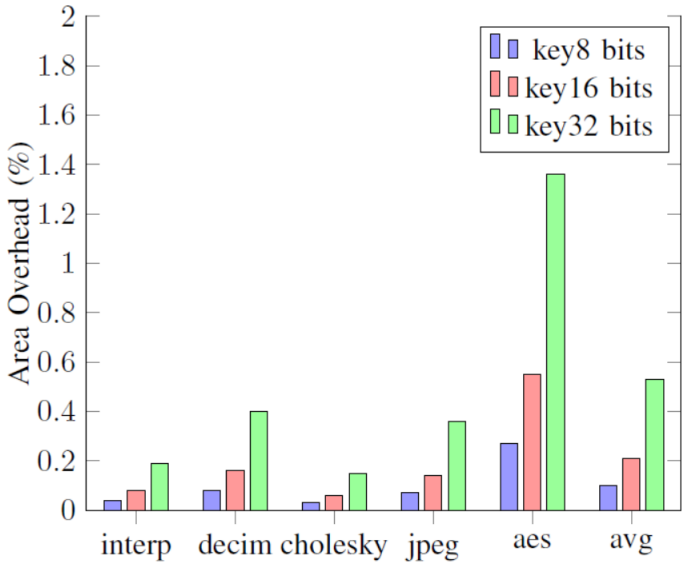

Figure 6.12: Area overhead results introduced by our proposed locking mechanism for keys of different bit sizes (8,16 and 32 bits).

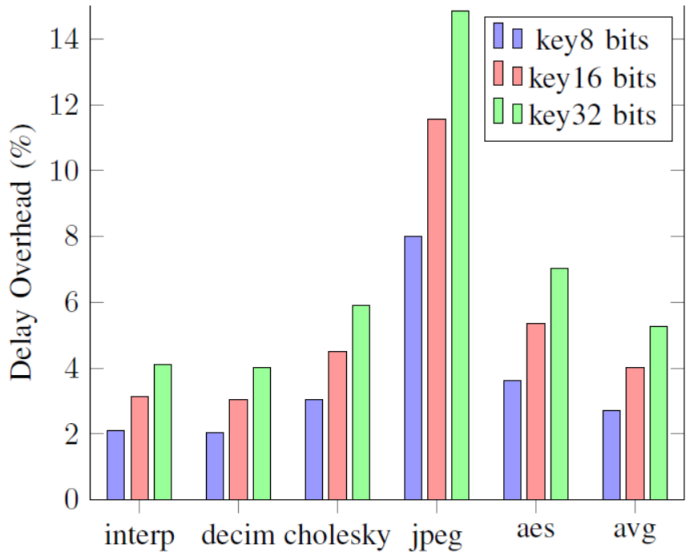

Figure 6.13: Delay overhead introduced by our proposed locking mechanism for keys of different bit sizes (8,16 and 32 bits).

for each of the benchmarks for different key sizes. As our method relies on randomly choosing the locking mechanisms at each state from a library of key primitives, the experiments are run five times, and the average values are reported. It could be observed that the overheads are very low and that they grow with the key size and the number of FSM states, as a new key is used for each state. On average the area overhead is 0.11%, 0.21% and 0.53% for the 8, 16 and 32-bit key cases, respectively. Regarding the delay overhead introduced by our method, it is on average 2.71%, 4.01% and 5.27% for the 8, 16 and 32-bit key cases, respectively, and up to 14.84% in the jpeg case. Although this maximum delay increase might seem large, in all cases the target maximum delay of 5ns was met, which means that all the locked circuits could still work at 200MHz. Thus, this delay increase does not affect the final circuit. It can also be observed that the area overhead grows linearly with the state count, as a new key primitive is inserted for each new state. Thus, further demonstrating the scalability of our approach.

With regard to the running time to break the lock, a brute force approach could

not break any of the benchmarks after allowing a maximum of 3 days to run, showing the strength of the proposed method.

6.12 Summary and Conclusion

In this chapter, we have proposed an automatic method to functionally lock behavioural IPs for HLS. The additional robustness compared to previous works comes from the fact that the circuit checks for a valid unique key at every state of FSM controller. In addition, the logic resources used for the locking mechanisms are shared with other operations, making a removal attack virtually impossible. Experimental results confirm that our proposed method is very effective.

Chapter 7

Conclusion

This chapter concludes the work done in this thesis as well as discusses future work and directions.

7.1 Conclusion

The primary focus of this work was to revisit both fault tolerance and hardware security using behavioural circuit design to reinstate reliability of integrated circuits. Reliability is a key factor that has to be kept in mind while designing an IC that is expected to be used in any safety-critical application. In this work here, we have viewed factors affecting system reliability to be multidimensional, which varies from radiation-induced faults to manual insertion of malicious circuits intending to cause system failure. We have proposed techniques that help mitigate such fault-induced system failures by leveraging some of the most salient advantages of using HLS.

In the first half of the thesis, we proposed techniques to reduce system failures due to transient errors followed by techniques to mitigate CMFs. The first proposed technique used SoC design capabilities of modern HLS tools to build a fault-tolerant MPSoC capable of recomputing during its slack time, induced due to bus congestion, making them robust against transient errors and thus preventing system failures. The

second contribution involved identifying techniques to mitigate CMFs. We proved the efficiency of using behavioural circuit design by using Design Space Exploration feature of HLS tools to generate a pool of micro-architectures, among which the best design pair with the highest diversity was chosen to mitigate CMFs. Time inefficiency of the existing technique in estimating diversity then paved way for an evaluation framework where we investigated if fault injection at the RT-level could accelerate the process of diversity estimation and replace the state of the art techniques. Since the results proved that they are not efficient enough to replace diversity estimation at the gate-level, we then introduced a predictive model called the PMDiversity capable of predicting the diversity between design pairs. Thus, in the first half of the thesis, we have been able to prove that revisiting fault tolerance using HLS could enhance the reliability of the existing systems by mitigating unintentional faults. The remainder of this thesis proposed a novel functional locking technique to protect behavioural IPs from being maliciously altered or from being stolen.

In this thesis, we have successfully bridged the gap between fault tolerance and hardware security by addressing their strengths together in enhancing the reliability of an IC. From this work, it could be concluded that in today's world of electronics, safety and security need to go hand in hand. At the same time, using HLS could improve modern day SoC design possibilities as well as pave the way for much advancements in future.

7.2 Future Work

The future directions of this work are multi-folds as we aim at bridging both mitigation of intentional and unintentional fault triggering factors. First, the two fault-tolerant techniques could be integrated to provide an even more robust system. Thus, it would be interesting to observe how robust such a system that exploits redundancy

in time and space against SEU and CMFs. Temporal redundancy could do more towards fault-tolerant systems. We will study the performance penalties introduced in a time redundant system when it is fully fault-tolerant, as opposed to the proposed method that re-computed only during available slack times and thus without any timing penalties. If the performance penalty lies within the permissible upper bound, we also aim at using this extra time to try and mitigate CMFs. Not all HWAccs need to extra cycles. But if one of the three outputs varies, it means that a fault has occurred. In case of a mismatch or lesser re-computation space, the slave could use extra cycles based on the upper bound to re-evaluate or do sufficient re-computations, respectively. Since from the proposed system it could be understood that bus congestion increases with the increase in complexity of the system, we aim at studying its effect on complex MPSoCs. Working in this direction will help develop a trade-off between timing overhead vs level of fault tolerance achieved.

Moreover, towards mitigating intentional alterations, we aim at strengthening the proposed system. The investigative study here tries to make use of behavioural circuit design to secure IPs, but it still needs more attention to be completely protected against SAT attacks. In the proposed method, brute force attack has been used for security evaluation because the variable to be obfuscated and the obfuscated bits are chosen at random. Using SAT attack on such obfuscation will have variations as the point of insertion is of great importance. Currently, due to its randomness, its ability to withstand SAT attack varies. Some benchmarks might take longer as the obfuscation point has low corruptibility; while some could be broken within seconds as a large number of keys get eliminated using a single DIP. With this in mind, we aim at performing a fault analysis based obfuscation to identify points that have the lowest output corruptibility. This could strengthen the lock from being broken by SAT attack.

With regards to protecting the BIP, future work would include developing a

stronger locking mechanism that makes use of circuits on-chip variations to try and protect an IP from being misused, while further reducing the area and delay overheads associated with them.

Publications

A. Balachandran and B. Carrion Schaefer, "Efficient Functional locking of Behavioral IPs", IEEE International Midwest Symposium on Circuits and Systems, pp.639-642, 2020.

F.N.Taher, **A. Balachandran** and B. Carrion Schafer, "Learning-based Diversity Estimation: Leveraging the Power of High-level Synthesis", *International Conference on Computer Design (ICCD)*, pp.460-467, 2019.

M.R. Babu, F. N. Taher, **A. Balachandran** and B. Carrion Schafer, "Efficient Hardware Acceleration for Design Diversity Calculation to mitigate Common Mode Failures", *IEEE International Symposium On Field-Programmable Custom Computing Machines (FCCM)*, pp.267-270, 2019.

F. N. Taher, M. Joslin, **A. Balachandran**, Z. Zhu and B. Carrion Schafer, "Common-Mode Failure Mitigation:Increasing Diversity through High-Level Synthesis", *Design, Automation, and Test in Europe (DATE)*, pp. 1563-1566, 2019.

A. Balachandran, N. Veeranna and B. Carrion Schafer,"On Time Redundancy of Fault Tolerant C-Based MPSoCs", *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp.631-636, 2016.

Bibliography

[1] S. Alcaide, C. Hernandez, A. Roca, and J. Abella. DIMP: A low-cost diversity metric based on circuit path analysis. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2017.

[2] A. Ayupov, S. Yesil, M. M. Ozdal, T. Kim, S. Burns, and O. Ozturk. A Template-Based Design Methodology for Graph-Parallel Hardware Accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(2):420–430, 2018.

[3] A. Balachandran, N. Veeranna, and B. C. Schafer. On time Redundancy of Fault Tolerant C-based MPSoCs. In *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 631–636, 2016.

[4] R. C. Baumann. Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Transactions on Device and Materials Reliability*, 5(3):305–316, 2005.

[5] A. Baumgarten, A. Tyagi, and J. Zambreno. Preventing IC Piracy Using Reconfigurable Logic Barriers. *IEEE Design Test of Computers*, 27(1):66–75, 2010.

[6] Hakem Beitollahi, Seyed Ghassem Miremadi, and Geert Deconinck. Fault-tolerant earliest-deadline-first scheduling algorithm. pages 1–6, 01 2007.

[7] S. Bhattacharya, S. Dey, and F. Brglez. Performance Analysis and Optimization of Schedules for Conditional and Loop-Intensive Specifications. In *31st Design Automation Conference*, pages 491–496, 1994.

[8] BJ Bonnett. Position paper on software safety and security critical systems. In *Proc. Compcon'84*, pages 191–198, 1984.

[9] N. S. Bowen and D. K. Pradham. Processor and memory-based checkpoint and rollback recovery. *Computer*, 26(2):22–31, 1993.

[10] T.-H. Lee C.-T. Hwang and Y.-C. Hsu. A Formal Approach to the Scheduling Problem in High Level Synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(4):464–475, 1991.

[11] Cadence. Stratus, October 2018.

[12] R. Camposano. Path-based scheduling for synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(1):85–93, 1991.

[13] Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Jason H. Anderson, Stephen Brown, and Tomasz Czajkowski. LegUp: High-Level Synthesis for FPGA-Based Processor/Accelerator Systems. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, page 33–36, 2011.

[14] R. S. Chakraborty and S. Bhunia. HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(10):1493–1502, 2009.

[15] Chia-Jeng Tseng and D. P. Siewiorek. Automated Synthesis of Data Paths in Digital Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 5(3):379–395, 1986.

[16] J. Cong and Zhiru Zhang. An efficient and versatile scheduling algorithm based on SDC formulation. In *2006 43rd ACM/IEEE Design Automation Conference*, pages 433–438, 2006.

[17] Morawiec Adam Coussy, Philippe. *High-Level Synthesis from Algorithm to Digital Circuit*, 38(8), April 2008.

[18] Philippe Coussy, Cyrille Chavet, Pierre Bomel, Heller Dominique, Eric Senn, and Eric Martin. GAUT: A High-Level Synthesis Tool for DSP applications. 2008.

[19] S. Dupuis, P. Ba, G. Di Natale, M. Flottes, and B. Rouzeyre. A novel hardware logic encryption technique for thwarting illegal overproduction and Hardware Trojans. In *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*, pages 49–54, 2014.

[20] K. Sankaralingam E.Blem Esmaeilzadeh, R. St. Amant and D. Burger. Dark silicon and the end of multicore scaling. In *International Symposium on Computer Architecture*, pages 365–376, 2011.

[21] Dubrova Elena. *Fault-Tolerant Design*. Springer, New York, 2013.

[22] Farah.N.Taher. *Fault Tolerance in HW Accelerators: Detection and Mitigation*. PhD thesis, University of Texas Dallas, 2019.

[23] K. Furutani, K. Arimoto, H. Miyamoto, T. Kobayashi, K. Yasuda, and K. Mashiko. A built-in Hamming code ECC circuit for DRAMs. *IEEE Journal of Solid-State Circuits*, 24(1):50–56, 1989.

[24] R. Gallo, M. Delvai, W. Elmenreich, and A. Steininger. Revision and verification of an enhanced UART. In *IEEE International Workshop on Factory Communication Systems, 2004. Proceedings.*, pages 315–318, 2004.

[25] C. H. Gebotys and M. I. Elmasry. Global Optimization Approach for Architectural Synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(9):1266–1278, 1993.

[26] Shalini Ghosh and Patrick Lincoln. Low-Density Parity Check Codes for Error Correction in Nanoscale Memory. 2007.

[27] S. Gupta, N. Savoiu, N. Dutt, R. Gupta, and A. Nicolau. Using global code motions to improve the quality of results for high-level synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(2):302–312, 2004.

[28] Stefan Hadjis, Andrew Canis, Jason Anderson, Jongsok Choi, Kevin Nam, Stephen Brown, and Tomasz Czajkowski. Impact of FPGA architecture on resource sharing in high-level synthesis. pages 111–114, 2012.

[29] Intel High-Level Synthesis Compiler, 2019.

[30] Rolf Isermann. Fault Diagnosis Systems An Introduction from Fault Detection to Fault Tolerance. *SERBIULA (sistema Librum 2.0)*, 01 2006.

[31] Rajiv Jain, Ashutosh Mujumdar, Alok Sharma, and Hueymin Wang. Empirical Evaluation of Some High-Level Synthesis Scheduling Heuristics. In *Proceedings of the 28th ACM/IEEE Design Automation Conference*, page 686–689. Association for Computing Machinery, 1991.

[32] T. Meade Z. Zhao D. Z. K. Shamsi, M. Li and Y. Jin. AppSAT: Approxi228 mately Deobfuscating Integrated Circuits. In *IEEE International Symposium on Hardware Oriented Security and Trust*, pages 95–100, 2017.

[33] S. Khaleghi, Kai Da Zhao, and Wenjing Rao. IC Piracy prevention via Design Withholding and Entanglement. In *The 20th Asia and South Pacific Design Automation Conference*, pages 821–826, 2015.

[34] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, page 388–397. Springer-Verlag, 1999.

[35] R. Koo and S. Toueg. Checkpointing and Rollback-Recovery for Distributed Systems. *IEEE Transactions on Software Engineering*, SE-13(1):23–31, 1987.

[36] F. Koushanfar. Provably Secure Active IC Metering Techniques for Piracy Avoidance and Digital Rights Management. *IEEE Transactions on Information Forensics and Security*, 7(1):51–63, 2012.

[37] D. C. Ku and G. De Mitcheli. Relative scheduling under timing constraints: algorithms for high-level synthesis of digital circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(6):696–718, 1992.

[38] Krzysztof Kuchcinski. Constraints-driven scheduling and resource assignment. *ACM Trans. Design Autom. Electr. Syst.*, 8:355–383, 07 2003.

[39] G. Lakshminarayana, K. S. Khouri, and N. K. Jha. Wavesched: a novel scheduling technique for control-flow intensive designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(5):505–523, 1999.

[40] J. H. Lala and R. E. Harper. Architectural principles for safety-critical real-time applications. *Proceedings of the IEEE*, 82(1):25–40, 1994.

[41] Parag Lala. Transient and permanent fault injection in vhdl description of digital circuits. *Circuits and Systems*, 03:192–199, 2012.

[42] P. A. Lee and T. Anderson. Fault Tolerance. In *Dependable Computing and Fault-Tolerant Systems*, 1981.

[43] Y. Lee and N. A. Touba. Improving logic obfuscation via logic cone analysis. In *2015 16th Latin-American Test Symposium (LATS)*, pages 1–6, 2015.

[44] A. Biere. Lingeling. Plingeling and Treengeling Entering the SAT Competition 2013. In *SAT Competition*, pages 51–52, 2013.

[45] Bev Littlewood. The impact of diversity upon common mode failures. *Reliability Engineering System Safety*, 51(1):101 – 113, 1996.

[46] M. C. McFarland. Using Bottom-Up Design Techniques in the Synthesis of Digital Hardware from Abstract Behavioral Descriptions. In *23rd ACM/IEEE Design Automation Conference*, pages 474–480, 1986.

[47] Mentor Graphics. Catapult high-level synthesis, October 2018.

[48] S. Mitra and E. J. McCluskey. Combinational logic synthesis for diversity in duplex systems. In *Proceedings International Test Conference 2000 (IEEE Cat. No.00CH37159)*, pages 179–188, 2000.

[49] S. Mitra and E. J. McCluskey. Which concurrent error detection scheme to choose ? In *Proceedings International Test Conference 2000 (IEEE Cat. No.00CH37159)*, pages 985–994, 2000.

[50] S. Mitra and E. J. McCluskey. Design of redundant systems protected against common-mode failures. In *Proceedings 19th IEEE VLSI Test Symposium. VTS 2001*, pages 190–195, 2001.

[51] S. Mitra, N. R. Saxena, and E. J. McCluskey. Common-mode failures in redundant VLSI systems: a survey. *IEEE Transactions on Reliability*, 49(3):285–295, 2000.

[52] S. Mitra, N. R. Saxena, and E. J. McCluskey. Common-mode failures in redundant VLSI systems: a survey. *IEEE Transactions on Reliability*, 49(3):285–295, 2000.

[53] S. Mitra, N. R. Saxena, and E. J. McCluskey. Techniques for estimation of design diversity for combinational logic circuits. In *2001 International Conference on Dependable Systems and Networks*, pages 25–34, 2001.

[54] N. Wehn M.Munch and M. Glesner. An Efficient ILP-Based Scheduling Algorithm for Control-Dominated VHDL Description. *ACM TODAES*, 2(4):344–364, 1997.

[55] Dr. Gordon Moore. Moore's law, 1965.

[56] M. T. Nabeel M. Ashraf J. Rajendran O. Sinanoglu M.Yasin, A. Sengupta. Provably-Secure Logic Locking: From Theory To Practice. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 1601–1618, 2017.

[57] NEC CyberWorkBench, 2015.

[58] M. Nicolaidis. Time redundancy based soft-error tolerance to rescue nanometer technologies. In *Proceedings 17th IEEE VLSI Test Symposium (Cat. No.PR00146)*, pages 86–94, 1999.

[59] Jaegeun Oh, Seok Joong Hwang, Huong Nguyen, Areum Kim, Seon Kim, Chulwoo Kim, and Jong-Kook Kim. Exploiting Thread-Level Parallelism in Lockstep Execution by Partially Duplicating a Single Pipeline. *Etri Journal - ETRI J*, 30:576–586, 08 2008.

[60] Ozgur Sinanoglu. Module 9 - SAT Attack on Logic Locking, 2017.

[61] B. M. Pangrle. Splicer: a heuristic approach to connectivity binding. In *25th ACM/IEEE, Design Automation Conference.Proceedings 1988.*, pages 536–541, 1988.

[62] A. C. Parker, J. Pizarro, and M. Mlinar. MAHA: A Program for Datapath Synthesis. In *23rd ACM/IEEE Design Automation Conference*, pages 461–466, 1986.

[63] Christian Pilato and Fabrizio Ferrandi. Bambu: A modular framework for the high level synthesis of memory-intensive applications. pages 1–4, 09 2013.

[64] S. M. Plaza and I. L. Markov. Solving the Third-Shift Problem in IC Piracy With Test-Aware Logic Locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(6):961–971, 2015.

[65] Raje and Bergamaschi. Generalized resource sharing. In *1997 Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*, pages 326–332, 1997.

[66] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri. Security analysis of logic obfuscation. In *DAC Design Automation Conference 2012*, pages 83–89, 2012.

[67] J. Rajendran, O. Sinanoglu, and R. Karri. Is split manufacturing secure? In *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1259–1264, 2013.

[68] J. Rajendran, H. Zhang, C. Zhang, G. S. Rose, Y. Pino, O. Sinanoglu, and R. Karri. Fault Analysis-Based Logic Encryption. *IEEE Transactions on Computers*, 64(2):410–424, 2015.

[69] Ganesan Ramalingam, J. Song, Leo Joskowicz, and R. Miller. Solving Systems of Difference Constraints Incrementally. *Algorithmica (New York)*, 23:261–275, 03 1999.

[70] S. K. Reinhardt and S. S. Mukherjee. Transient fault detection via simultaneous multithreading. In *Proceedings of 27th International Symposium on Computer Architecture (IEEE Cat. No.RS00201)*, pages 25–36, 2000.

[71] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. I. August. SWIFT: software implemented fault tolerance. In *International Symposium on Code Generation and Optimization*, pages 243–254, 2005.

[72] S. Roshanisefat, H. K. Thirumala, K. Gaj, H. Homayoun, and A. Sasan. Benchmarking the Capabilities and Limitations of SAT Solvers in Defeating Obfuscation Schemes. In *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, pages 275–280, 2018.

[73] J. A. Roy, F. Koushanfar, and I. L. Markov. EPIC: Ending Piracy of Integrated Circuits. In *2008 Design Automation and Test in Europe*, pages 1069–1074, 2008.

[74] B. C. Schafer and A. Mahapatra. S2CBench: Synthesizable SystemC Benchmark Suite for High-Level Synthesis. *IEEE Embedded Systems Letters*, 6(3):53–56, 2014.

[75] B. C. Schafer and Z. Wang. High-Level Synthesis Design Space Exploration: Past, Present, and Future. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2628–2639, 2020.

[76] A. Sengupta and D. Kachave. Spatial and Temporal Redundancy for Transient Fault-Tolerant Datapath. *IEEE Transactions on Aerospace and Electronic Systems*, 54(3):1168–1183, 2018.

[77] S.Haynal. *Automata-Based Symbolic Scheduling*. PhD thesis, University of California, Santa Barbara, 2000.

[78] Yuanqi Shen and Hai Zhou. Double DIP: Re-Evaluating Security of Logic Encryption Algorithms. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, page 179–184. Association for Computing Machinery, 2017.

[79] Shih-Chang Lai, Shih-Lien Lu, K. Lai, and Jih-Kwon Peir. Ditto processor. In *Proceedings International Conference on Dependable Systems and Networks*, pages 525–534, 2002.

[80] M. M. Shihab, J. Tian, G. R. Reddy, B. Hu, W. Swartz, B. Carrion Schaefer, C. Sechen, and Y. Makris. Design Obfuscation through Selective Post-Fabrication Transistor-Level Programming. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 528–533, 2019.

[81] P. Subramanyan, S. Ray, and S. Malik. Evaluating the security of logic encryption algorithms. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 137–143, 2015.

[82] P. Subramanyan, S. Ray, and S. Malik. Evaluating the security of logic encryption algorithms. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 137–143, 2015.

[83] Synopsys. Synopsys, "Design Compiler," Version 1.3.95, 2006.

[84] Niklas Sörensson and Niklas Een. Minisat v1.13-a SAT solver with conflict-clause minimization. *International Conference on Theory and Applications of Satisfiability Testing*, 01 2005.

[85] F. N. Taher, A. Balachandran, and B. Carrion Schafer. Learning-Based Diversity Estimation: Leveraging the Power of High-Level Synthesis to Mitigate Common-Mode Failure. In *2019 IEEE 37th International Conference on Computer Design (ICCD)*, pages 460–467, 2019.

[86] F. N. Taher, M. Kishani, and B. C. Schafer. Design and Optimization of Reliable Hardware Accelerators: Leveraging the Advantages of High-Level Synthesis. In *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, pages 232–235, 2018.

[87] Y. Tamir and C. Séquin. Reducing common mode failures in duplicate modules. 1984.

[88] D. DiMase J. M. Carulli M. Tehranipoor U. Guin, K. Huang and Y. Makris. Counterfeit Integrated Circuits: A Rising Threat in the Global Semiconductor Supply Chain. *IEEE*, 102(8):1207–1228, 2014.

[89] A. Vijayakumar and F. Brewer. Weighted Control Scheduling. In *International Conference on Computer Aided Design*, pages 777–783, 2005.

[90] J. Wei and K. Pattabiraman. BLOCKWATCH: Leveraging similarity in parallel programs for error detection. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, pages 1–12, 2012.

[91] Y. Xie and A. Srivastava. Anti-SAT: Mitigating SAT Attack on Logic Locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(2):199–207, 2019.

[92] Xilinx Vivado HLx, 2019.

[93] J. Xu and R. H. D. Netzer. Adaptive independent checkpointing for reducing rollback propagation. In *Proceedings of 1993 5th IEEE Symposium on Parallel and Distributed Processing*, pages 754–761, 1993.

[94] Xiaolin Xu, Bicky Shakya, Mark Tehranipoor, and Domenic Forte. Novel Bypass Attack and BDD-based Tradeoff Analysis Against All Known Logic Locking Attacks. pages 189–210, 08 2017.

[95] F. Yang, M. Tang, and O. Sinanoglu. Stripped Functionality Logic Locking With Hamming Distance-Based Restore Unit (SFLL-HD) – Unlocked. volume 14, pages 2778–2786, 2019.

[96] E. Yao, R. Wang, M. Chen, G. Tan, and N. Sun. A case study of designing efficient algorithm-based fault tolerant application for exascale parallelism. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, pages 438–448, 2012.

[97] M. Yasin, B. Mazumdar, S. S. Ali, and O. Sinanoglu. Security analysis of logic encryption against the most effective side-channel attack: DPA. In *2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, pages 97–102, 2015.

[98] M. Yasin, B. Mazumdar, J. J. V. Rajendran, and O. Sinanoglu. SARLock: SAT attack resistant logic locking. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 236–241, 2016.

[99] Muhammad Yasin, Abhrajit Sengupta, Benjamin Schäfer, Yiorgos Makris, Ozgur Sinanoglu, and Jeyavijayan Rajendran. What to lock?: Functional and parametric locking. pages 351–356, 05 2017.

[100] Y. C. Yeh. Triple-triple redundant 777 primary flight computer. In *1996 IEEE Aerospace Applications Conference. Proceedings*, volume 1, pages 293–307 vol.1, 1996.

[101] J. Zhang. A Practical Logic Obfuscation Technique for Hardware Security. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(3):1193–1197, 2016.