FAULT TOLERANCE IN HARDWARE ACCELERATORS: DETECTION AND MITIGATION

by

Farah Naz Taher

To my family

FAULT TOLERANCE IN HARDWARE ACCELERATORS:
DETECTION AND MITIGATION

by

FARAH NAZ TAHER, BS, MS

DISSERTATION

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY IN

ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT DALLAS

December 2019

ACKNOWLEDGMENTS

"The more I learn, the more I realize how little I know" - continues to be prevalent in my thought process. Whatever little I have achieved, would not have been possible without the amazing people I have in my life.

First, I want to thank my advisor, Dr. Benjamin Carrion Schaefer, for his support, guidance, encouragement and for being a great mentor. His supervision not only has widened my knowledge but also enlightened my problem-solving skills. I express my gratefulness to him for taking me as his apprentice and having faith in my work.

I want to thank Dr. Mehrdad Nourani, Dr. Dinesh Bhatia and Dr. Yang Hu for agreeing to be in my dissertation committee and their valuable feedback. I want to take this opportunity to thank all my teachers from my school and North South University, to Auburn University and UT Dallas.

I express my sincere appreciation to Matthew, Zhiqi, Mahesh and Anjana for being such wonderful co-authors and friends. My heartfelt thanks to Siyuan and Jianqi for helping me out with their expertise whenever I needed. I want to express my gratitude to my co-Darclab members and UTD friends: Bo, Nandesh, Zi, Mihir, Sushmitha, Himanshu, Monica V, Shuangnan, Vinay, Monica J, Hao, Monir and Taufique. I would like to thank Sharmin, Nadia and Tamanna family for all the love and affection they showered me with. I want to thank all my family and friends in both the USA and Bangladesh, for the constant support and encouragement. A special heartfelt gratitude to the teachers and caregivers of my daughter's daycare.

I thank Mustafa Munawar Shihab for being my brother and my best friend. Thank you for all the support, conversations, fighting and laughing together throughout NSU, AU and UTD.

No words are adequate to express my gratitude to my family for their unconditional love and support. I am more than lucky to have such a family who appreciates and supports my goal. I thank my mother, Shahnaz Sultana, and my father, Abu Taher Chowdhury, for all the sacrifices they have made. All their encouragement, hard work and prayers have made me what I am today. I thank my sister Mayesha Naz Taher, for all the unconditional love and courage she gave me.

I thank my husband Asad Muhammad Zaman for all his love, patience, and support. I am blessed to have such a supportive and patient life partner. I am fortunate to have my little jaan, Aairah, who has been understanding and affectionate throughout the journey.

I dedicate my work to my awesome family.

Finally, I thank the Almighty Allah for this beautiful life, and for the wonderful people He has filled it up with.

August 2019

FAULT TOLERANCE IN HARDWARE ACCELERATORS:

DETECTION AND MITIGATION

Farah Naz Taher, PhD
The University of Texas at Dallas, 2019

Supervising Professor: Benjamin Carrion Schaefer, Chair

In the age of self-driving cars and space adventures, fault tolerance has become a first-order design metric. Thus, it is vital to incorporate fault tolerance coherently into the Very Large Scale Integrated (VLSI) design process. This is especially the case in state-of-the-art complex heterogeneous Systems-on-Chip (SoC), which typically contain a variety of dedicated hardware accelerators. These SoCs have to be taped out at shorter and shorter periods while their complexity keeps increasing. This is driving designers to finally embrace the use of C-based VLSI design called as High-Level Synthesis (HLS). HLS has shown to significantly reduce the design and verification time compare to the use of low-level hardware descriptions languages. Moreover, one significant advantage of raising the level of abstraction is that C-based VLSI design allows to generate a variety of micro-architectures with different trade-offs from the same untimed behavioral description.

Fault-tolerance at the hardware level has so far has been mainly based around building N-modular redundant (NMR) systems like duplication and Triple Modular Redundancy (TMR), where the hardware channel is identical.

In this work, we exploit HLS's advantage to generate micro-architectures with different characteristics from the same behavioral description for automatically generating fault-tolerant systems.

In particular, we first propose an automated framework that given a single behavioral description generates a set of NMR systems with different area and performance trade-offs by choosing different mixes of micro-architectures.

Secondly, we leverage this advantage to generate redundant systems that minimize common-mode failure (CMF). CMFs imply that multiple modules in the redundant system are affected at the same time by a fault. Hence, it has been shown that adding diversity in the hardware channels can make the system more tolerant to these type of faults. We also leverage the power of machine learning to estimate the diversity through fast and efficient predictive methods, thus, significantly speeding up the redundant system generation. A previously reported design diversity metric called DIversity Metric based on circuit Path analysis (DIMP) or RT-level fault injection based method is investigated to check if they can achieve similar results compared to the gate-netlist fault injection based diversity calculation.

Lastly, a low-cost, universal fault-recovery/repair method that utilizes supervised machine learning techniques to ameliorate the effect of permanent fault(s) in hardware accelerators that can tolerate inexact outputs is proposed.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ALAP	As Late As Possible
ASAP	As Soon As Possible
ASIC	Application-Specific Integrated Circuit
BDL	Behavioral Descriptive Language
CC	Correlation Coefficient
CMF	Common-mode Failures
CMOS	Complementary Metal–Oxide–Semiconductor
CPU	Central Processing Unit
CWB	CyberWorkBench
DC	Design Compiler
DFG	Data Flow Graph
DP	Design Pool
DSE	Design Space Exploration
DSP	Digital Signal Processing
DV	Diversity
DWC	Duplication with Compare
ED	Error Distance
EMI	Electromagnetic Interfere
FPGA	Field-Programmable Gate Array
FU	Functional Unit
GA	Genetic Algorithm

HDL	Hardware Description Languages
HLS	High Level Synthesis
HWacc	Hardware Accelerators
IC	Integrated Circuits
IP	Intellectual property
ITRS	International Technology Roadmap for Semiconductors
MAE	Mean Absolute Error
ML	Machine Learning
MPSoC	Heterogeneous Multiprocessor Systems-on-Chips
NMR	N-modular redundancy
NVP	N-version Programming
QoR	Quality of Result
RTL	Register-Transfer Level
S2CBench	Synthesizable SystemC Benchmark Suite
SER	Soft Error Rates
SEU	Single-event upsets
SoC	Systems-on-Chip
TDDB	Time-dependent Dielectric Breakdown
TMR	Triple Modular Redundancy
TV	Test Vector
VLSI	Very Large Scale Integration

CHAPTER 1
INTRODUCTION

1.1 Motivation

One of the solutions being proposed to deal with the slowdown of Moore's law (2) and the breakdown of Dennard's scaling (3) is to customize the computing platforms to the application domain, also known as domain specific computing. At the computer level, by incorporating CPUs with FPGA and at the chip level through integrating heterogeneous Multiprocessor Systems on Chips (MPSoCs) with dedicated Hardware Accelerators (HWacc).

The ITRS roadmap 2009 edition (4) predicted: *"The ultimate nanoscale device will have a high degree of variation and a high percentage of non-functional devices right from the start"* and *"Circuits that can dynamically reconfigure itself to avoid failing and failed devices (or to change/improve functionality) will be needed"*.

In these modern complex heterogeneous SoCs, large portions of the computations are offloaded to dedicated hardware accelerators that can process these applications more efficiently by exploiting the inherent parallelism of the accelerators. Although more complex, these accelerators can perform computations faster at a much lower frequency and thus, lower power.

Our work mainly targets hardware accelerators because they are typically the main differentiating components between different IC vendors (e.g., Apples A11 bionic SoC with its dedicated neural network accelerator). At the same time, the pressure to deliver these new or updated ICs at shorter time periods and the use of new technologies makes these accelerators more prone to hardware errors compared to more mature and tested IPs.

Fault tolerance is the capability of a system to continue functioning without interruption in the occurrence of failure when one or more of its components fail. In an era of self-driving cars and space adventures, fault tolerance has become a first-order design metric. Thus, it is essential to incorporate fault tolerance coherently into the Very Large Scale Integrated (VLSI) design process.

N-modular redundancy (NMR) techniques such as Duplication and Triple Modular Redundancy (TMR) or additional logic are extensively popular for designing dependable systems with fault tolerance and data integrity. Data integrity is maintained if a system either produces correct outputs or generates an error signal when incorrect outputs are produced. This is especially the case in state-of-the-art complex Systems-on-Chip (SoC) containing a variety of dedicated hardware accelerators.

Occurring faults in hardware are either transient or permanent. The first, cause no permanent damage to the circuit, while the other result in permanent destruction or degradation of the circuit. Transient faults also known as Single-Event Upsets (SEUs) are errors due to electrical noise or external radiation rather than manufacturing or design defects (5). These effects were first predicted in 1962 by Wallmark et al. (6) and in 1975 the first anomaly in a spacecraft system was encountered attributed to energetic heavy-ion passage (7). Besides, process scaling has led to increasingly critical challenges in manufacturing and lifetime reliability of integrated circuits (ICs). Common-Mode Failures (CMFs) is also a major concern for redundant systems. CMF implies that multiple modules in the redundant system are affected at the same time. CMF can occur due to external events, e.g., Electromagnetic Interfere (EMI), power-supply disturbances, and radiation, internal or design mistakes (8). Traditional same hardware module redundancy systems cannot protect against CMFs because the replicated hardware modules will produce the same erroneous output and hence the voter will interpret the output as correct (9).

For larger electronic systems these channels are being built by sourcing parts from different vendors. However, in VLSI design, the underlying hardware channel is exactly the same since it is too expensive to design multiple hardware versions with the same specifications. The Triple Module Redundancy (TMR) is one of the most successful implementations of NMR systems as it can fully mask errors.

Most prior work on VLSI hardware reliability utilize NMR, assuming that each module is the same. A voter is in turn used to determine if a fault has happened by comparing the results. Although useful for independent component failures, rudimentary replication cannot fully protect against soft-errors occurring in a redundant system and most importantly for Common Mode failures (CMFs).

One of the proposed solutions is to use asymmetric redundancy (10), where multiple functional equivalent modules are instantiated in parallel with a voter. The main idea is to make faults occurring in both modules at the same time, visible at the outputs such that each module generates a different value. The worst case scenario in any fault tolerant system occurs if a system has a fault and the majority of the NMR modules produce the same erroneous output. Asymmetric redundancy aims at specifically avoiding this case.

The main problem with asymmetric redundancy is that it involves creating multiple different micro-architectures for each hardware channel. The most rudimentary form of design diversity is to have multiple parallel teams building the same circuit using different tools. This is obviously extremely expensive and inefficient. Thus, automated methods to obtain different implementations of the same circuit are required.

We make use of one of the main advantages of C-based VLSI design, a.k.a. High Level Synthesis (HLS) to generate fault tolerant systems. In particular, the main advantage that we leverage is the ability to generate micro-architectures with unique area vs. performance trade-offs without having to modify the input description. This is generally done by setting different synthesis options, typically specified as comments or pragmas at the source code. This allows to control how to synthesize loops (e.g., unroll or fold), arrays (e.g., memories or registers with a different number of ports) and functions (inlined or not). This distinct advantage is leveraged to generate more robust NMR systems. We also utilize the power of Machine Learning, a model based on training data in order to make predictions to perform a specific task, to build adaptive learning based systems.

In the majority of this dissertation, ways to tackle faults that may happen after the circuit is in use are examined, but in the last part, one method is proposed to recuperate faults which are already present when a chip is manufactured. We proposed a machine learning based permanent fault recuperation method to save the circuits, which would have been discarded without any fault tolerant methods.

1.2 Dissertation Contribution

The work done in this dissertation leverages the power of High-level Synthesis and/or machine learning to generate more robust hardware systems mainly against soft errors, common mode failure and permanent faults. The overview of this dissertation is depicted in Fig. 1.1. This dissertation makes the following contributions:

1. **Optimizing TMR System Design:** An automatic method to generate optimized redundant hardware accelerator with maximum reliability given a single behavioral description for High-Level Synthesis (HLS) is proposed

2. **Diverse Design Generation:** An automatic design space exploration (DSE) method to generate optimized redundant hardware accelerator with maximum diversity is advocated given a single behavioral description utilizing High-Level Synthesis(HLS) to protect against Common-mode Failure (CMF).

3. **Learning-based Diversity Estimation:** A fast machine learning based method that facilitates the design space exploration (DSE) of single behavioral description is proposed in order to generate optimized redundant hardware accelerator system with maximum diversity to protect against CMFs is proposed.

4. **Permanent Fault Compensation:** A low-cost, universal fault recuperation method that utilizes supervised machine learning techniques to ameliorate the effect of permanent fault(s) in hardware accelerators that can tolerate inexact outputs is proposed.

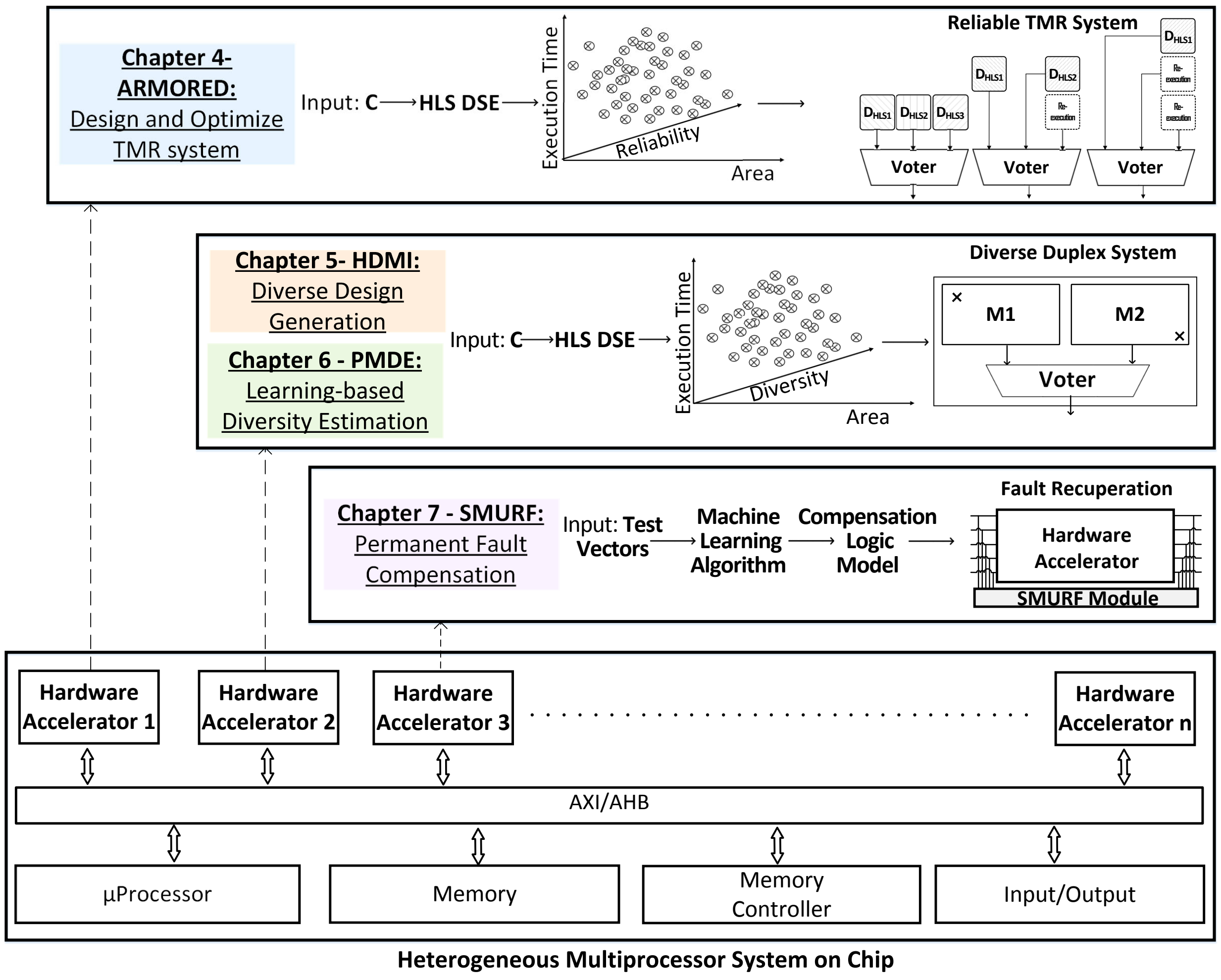


Figure 1.1: Overview: Targeted Heterogeneous SoC example and fault-tolerant techniques proposed in this dissertation.

1.3 Dissertation Organization

This dissertation focused mainly on fault tolerance in hardware accelerators are divided into eight chapters. The first chapter, the research work is motivated and the overall structure is incised. The next two chapter highlights the previous and related works as well as the background information in the area of this dissertation. As the boundaries of the technologies are pushed towards the atomic level, the probability of the faults introduced in the circuits or system is becoming significant due to factors like sub-atomic particle radiation. Chapter

4 proposes a method to generate an optimized reliable triple modular redundant system to protect against soft errors. Chapter 5 and 6, proposed methods to elevate the diversity of a design pair and also a method to estimate the diversity in an adaptive manner to protect a duplex system against common mode failure. Next, in Chapter 7, a machine learning based fault compensation method is proposed to recuperate the permanent faults in faulty hardware accelerators. In the final chapter, we conclude this dissertation and also suggest possible future works.

CHAPTER 2
BACKGROUND

This chapter introduces some important concepts required to fully appreciate the main contributions of this dissertation. In particular, HLS, HLS design space exploration, and machine learning.

2.1 High-level Synthesis

Specialized heterogeneous architecture design is a recent trend to deal with the power wall while increasing the performance of current chips. Manual RT-level hardware design is challenging as it is extremely time consuming and requires complex coding.

In a traditional VLSI design flow, the hardware designer has to analyze a given behavioral description and based on a set of constraints manually creates an RTL description that can execute the given functionality efficiently. Raising the level of abstraction approach has shown to shorten the time to market period (11). C-based VLSI design, has the additional advantage of allowing the generation of micro-architecture with different characteristics from the same behavioral description. This is typically done by setting different synthesis options to e.g., control how to synthesize arrays, loops and functions.

Moreover, HLS increases the re-usability of the code by allowing to easily re-target it to an FPGA or ASIC, by merely modifying a library & constraints files.

2.1.1 Design Flow of High Level Synthesis

Design cost and time-to-market are two primary issues that all hardware designers must overcome. Manual hardware design is more expensive and time consuming, whereas HLS decreases these issues significantly. HLS also expedite the DSE for effective hardware/software co-design. Although traditionally, design objectives for HLS were developing small and fast

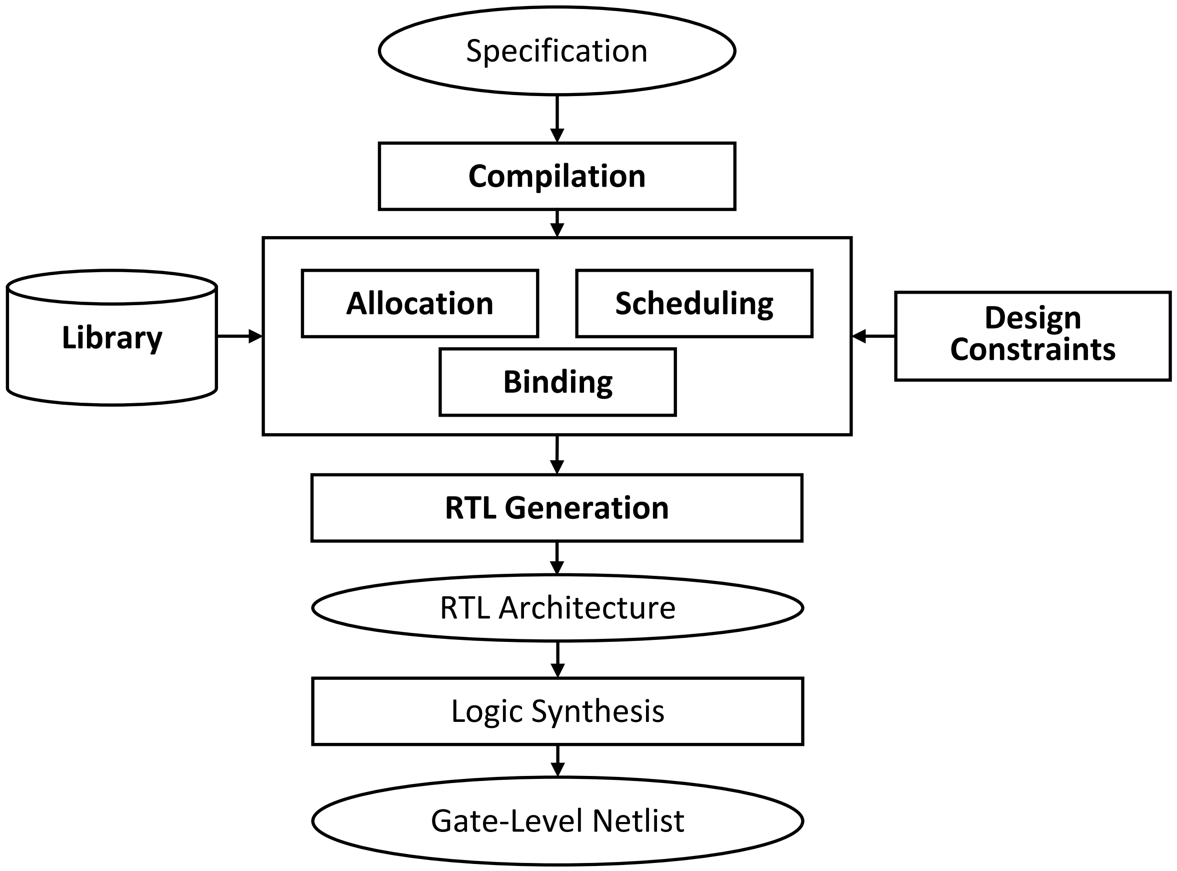


Figure 2.1: High-level Synthesis design flow.

designs. However, nowadays, designing power efficient design has gained more concerns as a more critical design constraint.

High level synthesis simplifies evaluation of architectural choices such as hardware and software designs, synthesis and verification, memory organization, and power management. HLS starts with the algorithmic specification of an application (standard synthesizable C/C++ source code or SystemC description), RTL design library (including component characteristics e.g., area, delay, power, etc.), and design constraints (cost, performance, power consumption, resources, testability criteria, pin-count, etc.) and which in turn would later automatically generates RTL design of datapath and control logic (12; 13).

Specification and Compilation

The standard description needs to be made synthesizable to make compatible with defining a hardware description. The first step typically requires to refine the original behavioral description to make it synthesizable. This synthesizable description can, in turn, be explored, creating designs with a unique area, time and/or other constraints. The functional specification defines the input data, output data, and computational delays. HLS defines the variable and data types at the software level. For a realistic hardware operation, we require alteration from floating-point and integer data types into bit-accurate data types of a specific length. Bit-accurate specification is critical in generating an optimized architecture. Designers can implement a specification of an application like custom processor or custom hardware unit using HLS.

The process starts with the compilation of the specification, allocates hardware resources like functional units, memory units etc, schedules the operation by clock cycles, binds the operations to functional units, variables to memory units, and generates the RTL micro-architecture (11). HLS begins with the compilation of functional specification; it translates the input specification to formal representation based on this formal representation CDFG is generated. CDFG is a graph that describes the control and data dependency between the operations (14). CDFG exhibits data dependencies in the basic blocks and captures the control flow between those basic blocks. It is used to generate the actual hardware. The following are the three main algorithms involved in High-Level synthesis: Allocation, Binding, and Scheduling.

Scheduling

Scheduling determines the sequence in which the operations are executed in each control step or each clock cycle. Let b be the operation to be scheduled, b= a op c, first variables a and c will be read from their sources which could be either a storage component or a functional unit component and brought it to the input of a functional unit that can execute operation op, and the result a will be sent to its destination. In this execution, the data dependency of the variables a and c is analyzed before it is brought to its input. If there is a dependency of operation preceding this constraint, the dependent operation is scheduled in the same way before it proceeds to schedule the above operation. For any HLS platform, module library comprising of circuits for various functionality like an adder, multiplier, registers etc will exist. Based on the functional components to which the operation is mapped, the operation can be scheduled within one clock cycle or several clock cycle. If there are sufficient resources available, the operations can be scheduled in parallel (11).

Scheduling algorithms can be roughly categorized into data-flow based (DF-based) scheduling and control-flow-based (CF-based) scheduling. DF-based scheduling concentrates on data-flow-intensive applications (e.g., digital signal processing and image processing) which based on the optimization goals can be divided further into two classes: timing-constrained and resource-constrained. Force directed scheduling and list scheduling are popular methods to solve timing and resource constraints scheduling problem. Path-based scheduling and Loop-directed scheduling are some of the CF-based scheduling that aims control-flow-intensive applications (e.g., controllers and network protocol processors) (15). The most basic scheduling techniques are i. As Soon As Possible (ASAP) scheduling and ii. As Late As Possible (ALAP) scheduling. ASAP schedules operations in the earliest control step one at a time whereas ALAP schedules it from the last control step towards the first (16).

Allocation

Allocation is defined by the number of hardware resources needed to meet the design constraints. Once the behavioral description is parsed, a functional unit constraint file containing the type and the number of functional units required to map the particular input description is generated. Based on the HLS tool, components like buses or interconnections among the components may be added during scheduling and binding tasks. RTL component library is the source of the tool to select the components for allocation. At least one component is a default requirement to operate a specific model. The library must also include component characteristics such as area, power, and delay and its metrics for the synthesis tasks. To achieve a high level of parallelism, the HLS tool will try to maximize the utilization of functional units as much as possible. The user has the leverage to overwrite this constraint file manually, setting the maximum number of functional units that the synthesizer can instantiate.

Binding

Binding is the last stage in HLS. After all the operations are scheduled and allocated, we conceive the exact information of the type of circuit models to be used and their numbers. Binding maps each operation to a functional unit and each variable to a register. However, we need more information to generate the RTL. During allocation, the operation in a control step is handled by different modules; however, modules are shared across the operations that are in different control steps. The binding task (also called resource-sharing step) assigns the operations and variables to hardware modules. A resource such as an operational module or register can be shared by different operations, data accesses, or data transfers if they are mutually exclusive. Storage and functional unit binding also rely on connectivity binding, which requires each handover from a component to a component be bound to a connection unit (e.g., a multiplexer or a bus) (14). Binding is a critical stage in HLS since it affects the routability of the final circuit and hence the wire length and critical path.

RTL Generation

The final step in the HLS process is the RTL generation. The RTL architecture generation step aims to apply all the design decisions made and generate an RTL model of the synthesized design. Generated RTL model includes a control path and data path. Data path consists of components like registers, functional units like ALU, shifters, multipliers, interconnect elements like tristate drivers, multiplexers, buses etc. Data inputs and outputs are connected to the data path, and control inputs and outputs are connected to the controller. The controller is a finite state machine; it manages the flow of data in the data path by setting the values of the control signal. The controller's input may come from primary inputs or the data path components like a comparator. Every component may consume one or more clock cycles to execute, can be pipelined, and can have input or output registers. We can pipeline the entire data path and controller into several stages. (14).

2.1.2 Design Space Exploration

High Level synthesis is an efficient approach to reduce the design time and cost, optimize all possible design constraint like area, execution time, functional unit constraints, memory unit constraints etc, to generate multiple efficient architectures.

The activity to leverage design possibilities prior to the implementation by exploring design alternatives of a system by varying the design metrics is denoted as Design Space Exploration (DSE). Major tasks like rapid prototyping, optimization, and system integration can be conducted on an ample design space of potential candidates by utilizing the power of DSE. The main challenge of DSE appears when the size of a problem is immense and complex making the design space enormous(17).

In this process, the specification is defined at the behavioral level and it is converted to Register transfer level by exploring the design space efficiently. Architectural choices have a significant impact on the performance and area of design. They also have a high impact on power, though often in less predictable ways than for area and performance. Exploring different choices and measuring their impact is greatly facilitated by HLS (12).

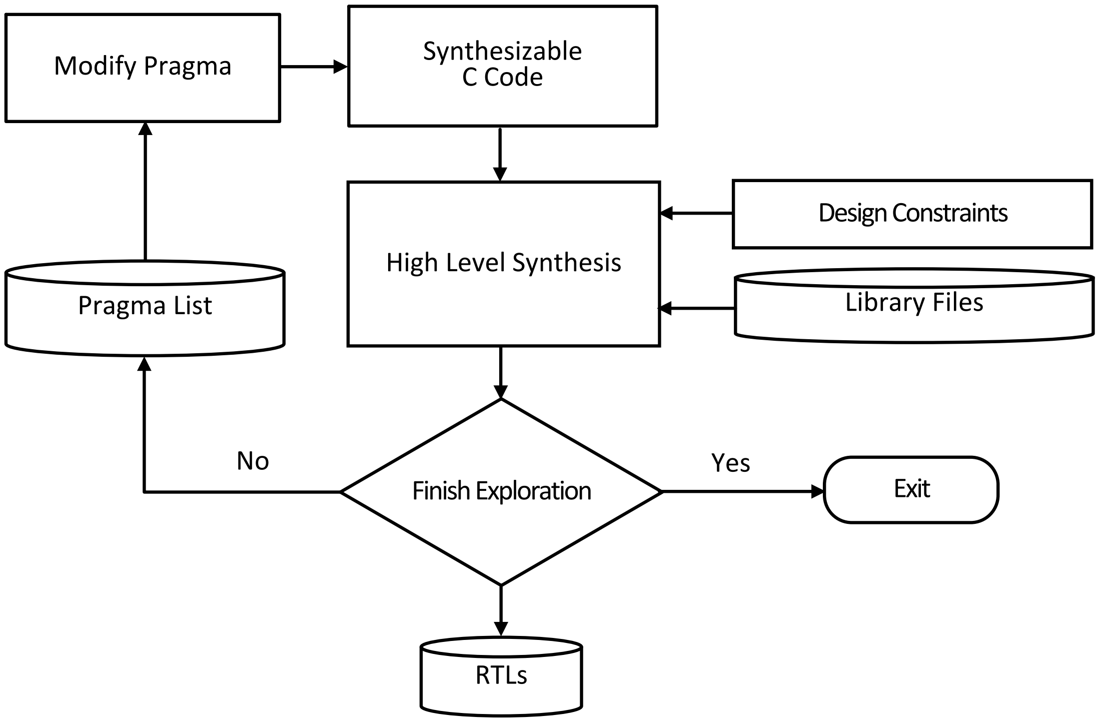


Figure 2.2: High-level Synthesis Design Space Exploration.

The HLS process is mostly considered as a black box where three leading families of synthesis knobs are set beforehand the process is executed. The knobs are (i) Synthesis Directives, (ii) Global Synthesis and (iii) FU Exploration.

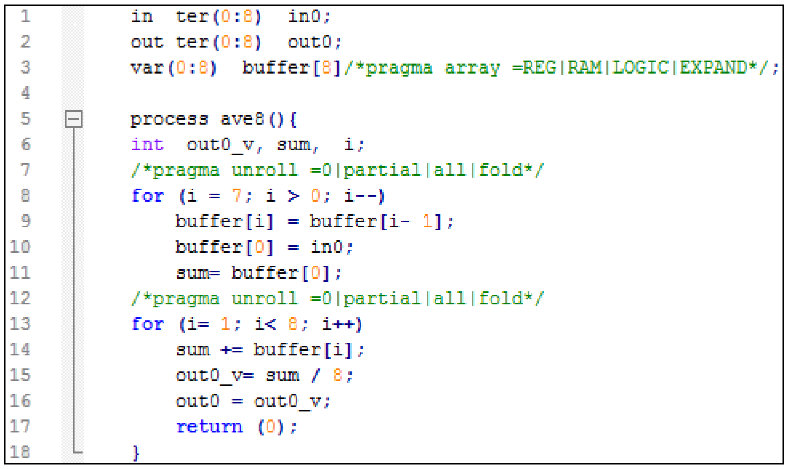
Table 2.1: Forms of Pragmas(1).

Operation	Attribute	Description
Loops	unroll = 0	do not unroll loop
	unroll = x	partial loop unroll
	unroll = all	unroll loop completely
	folding = N	fold loop N times
Functions	func = inline	inline each function call
	func = goto	single function instantiation
	func = seq_{opr}	function inst as sequential opr
	func = pipeline	function inst as pipeline opr
Arrays	array = RAM	array synthesized as memory
	array = logic	constant arrays synthesized as logic
	array = expand	expand array
	array = reg	synthesize array as registers

Knob 1: The knob that is extensively used is synthesis directives in the form of pragmas inserted directly into the synthesizable source code. It is also the most powerful *knob*, as it fixed the overall micro-architecture. The primary constructs in the behavioral description which have the highest footprint on the final micro-architecture in HLS are loops, arrays and functions. Table 2.1 highlights an example of different types of pragmas that can be inserted in the behavioral description.

With these pragmas, it is possible to synthesize arrays as memories or registers. Loops can be unrolled or pipelined and functions inlined or synthesized as single Hardware (HW) blocks.

Figure 2.3: Example for Pragma Insertion as Comments in ave8 code.



```
in  ter(0:8)  in0;
out ter(0:8)  out0;
var(0:8)  buffer[8]/*pragma array =REG|RAM|LOGIC|EXPAND*/;

process ave8(){
int  out0_v, sum,  i;
/*pragma unroll =0|partial|all|fold*/
for (i = 7; i > 0; i--)
    buffer[i] = buffer[i- 1];
    buffer[0] = in0;
    sum= buffer[0];
/*pragma unroll =0|partial|all|fold*/
for (i= 1; i< 8; i++)
    sum += buffer[i];
    out0_v= sum / 8;
    out0 = out0_v;
    return (0);
}
```

The example in Fig. 2.3 highlights how local synthesis directives in an ave8 code can be leveraged to control different levels of variations in pragma settings that can lead to a unique micro-architecture that might or might not be Pareto-optimal.

Knob 2: The second exploration knob is global synthesis options that can only be applied to the entire behavioral description to be synthesized. Clock constraint, scheduling type, data initiation interval for pipelined designs, encoding scheme for the finite state machine (FSM) controller, etc are some of the types of global synthesize options. These options if set, cannot be applied to specific portions. For example, the global synthesize frequency can be set to 500MHz/1GHz or the encoding of the controller FSM can be set to on hot encoding scheme.

Knob 3: This third exploration knob enables different levels of resource sharing, by controlling the number and type of Functional Units (FUs) that the synthesizer can use. Resource sharing is a well-known optimization technique that can be used to reduce the area while increasing the latency of the circuit where a single functional unit (FU) is re-used among various computational operations.

Authors of (18) presented a paper to simultaneously perform scheduling, allocation and module selection using a problem-space genetic algorithm to optimize the hardware resources (i.e., functional units, registers, and interconnection cost), clock period and control steps. In (19), an efficient and accurate DSE technique using HLS for applications consisting of multiple nested loops with or without data dependencies is presented.

Different types of DSE can be performed of a design in order to generate micro-architecture that better meets the project needs. Brute force DSE can be time-consuming and expensive, so several heuristic methods were proposed. Heuristics DSE techniques have been proposed based on Genetic Algorithms (20) or Simulating Annealing (21) as exhaustive search is too expensive for large solution space. Learning based methods are also proposed by several researchers to speed up the exploration process. The authors of (1) present a Machine learning predictive modeling high-level synthesis DSE. The method generates a predictive model for a training set until a given error threshold is reached. Next, it continues with the exploration using the predictive model, avoiding time-consuming synthesis and simulations of new configurations. A learning-based method based on random forest algorithm and Transductive Experimental Design is proposed to explore Pareto optimal set (22) efficiently.

2.1.3 Commercial HLS Tool

There are various HLS tools in the market like NEC's Cyberworkbench, Xilinx Vivado HLS, Mentor's Catapult, etc. which takes a behavioral description as input instead of structural description (Table 2.2). A couple of the tools are shortly described below.

CyberWorkBench (CWB) is an HLS tool of NEC Corporation. CWB supports C, Behavioral Description Language (BDL) and SystemC. CWB supports two types of scheduling automatic and manual. CWB supports verification at different levels: behavioral, source code, RTL and cycle accurate. Various simulation scenarios can be generated automatically or by user provided test benches. CWB also includes a system explorer that can generate design implementations for different combinations of compiler settings and represents them with an area-latency diagram (23). We have used CWB for our research.

Stratus is the HLS tool of cadence which accepts ANSI C and C++ as input languages and generate a SystemC wrapper. The wrapper generated by CtoS does not comply with OSCI SystemC standards mainly used for ASIC design (23).

Vivado HLS tool by Xilinx accepts C, SystemC and C++ as input, and hardware modules are generated in SystemC, Verilog and VHDL. During the compilation process, different optimizations, such as operation chaining, loop unrolling and loop pipelining is applied to perform micro architectural exploration (24).

Catapult C supports C, SystemC and C++ as input language. The tool does not optimize memory accesses and array elements that are re-used in subsequent iterations are fetched from memory on every use. The designer has to modify the source code to enable local buffering (23).

2.2 Machine Learning

Current technology has seen an outburst in machine learning with an analogous necessity to produce custom hardware for best area/performance, power efficiency and fault tolerance.

Table 2.2: Various HLS tools.

Tool	Company	Input	Output
Catapult (25)	Mentor	C,SystemC,C++	Verilog,VHDL
CyberWorkBench (26)	NEC corporation	BDL, C,SystemC, C++	Verilog,VHDL
Intel HLS Compiler (27)	Intel	C,C++	Verilog
LegUp HLS (28)	LegUp Computing	C	Verilog
MATLAB HDL Coder (29)	Mathworks	Matlab Code	Verilog,VHDL
Stratus (30)	Cadence	C,SystemC,C++	Verilog,VHDL
Synphony C Compiler (31)	Synopsys	C,C++	Verilog,VHDL,SystemC
Vivado HLS (32)	Xilinx	C,SystemC,C++	Verilog,VHDL,SystemC

Machine learning educates computers to learn from experience which comes naturally to humans and animals. Machine learning algorithms employ computational methods to "learn" directly from given data without depending on a preset equation as a model. The algorithms adaptively improve with the number of samples available for learning grows.

Machine learning algorithms discover inherent patterns in data that generate perception to produce better decisions and predictions. It is used in today's world in medical diagnosis, computational finances, image processing, face/motion detection, automotive, aerospace, language processing etc.

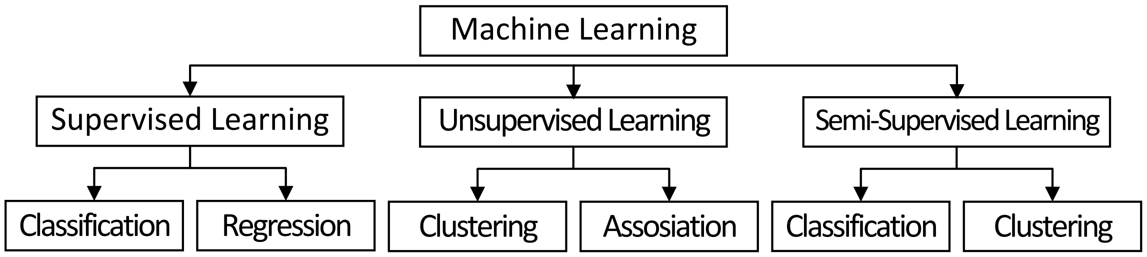


Figure 2.4: Machine Learning Classification.

i. Supervised learning and ii. Unsupervised learning iii. Semi-supervised learning are three different kinds of machine learning algorithms (Fig. 2.4).

i. Supervised learning: This task-driven method uses known input and output response data/observations to train a model to predict future outputs.

ii. Unsupervised learning: From input data this method finds intrinsic structures or hidden patterns in this data-driven method.

iii. Semi-supervised learning: In cases where some output observations are presents and some are missing, semi-supervised learning algorithms is the best candidate.

2.2.1 Supervised Learning

In this dissertation works, supervised learning algorithm is used. Supervised learning is the most crucial methodology in machine learning, which allows for a flexible, fast and scalable way to generate accurate compensation routines, which are specific to the particular set of application inputs and outputs. In supervised learning, a map between a set of input attributes and an output variable is used to predict the unseen data or in other words to build a brief model of the distribution of class labels according to predictor features (33).

The objective for supervised learning is to construct a concise model of the distribution of class labels in terms of predictor features. The subsequent classifier is used to allocate class labels to the testing instances where the values of the predictor features are known, but the value of the class label is unknown.

Fig. 2.5 shows the steps in a supervised learning method. The first step is setting the problem settings and collect data set. Next, data preparation, data pre-processing and training data selection are done. Finding the right algorithm is partially based on trial and error but a very critical step for the overall success. The next step is the training set where a model is generated using known input and output responses to generate a model. For the third step, unknown input data is used to validate the model. If the error rate is non-tolerable, previous steps need to be re-run with parameter tweaks.

Types of supervised learning

A supervised learning algorithm intakes a known set of input variables (x) and known responses (y) to the data and trains a model y=f(x) to generate reasonable predictions for the

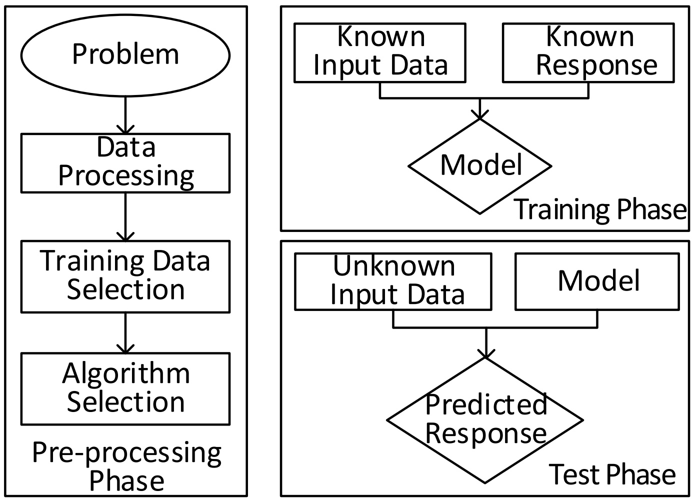


Figure 2.5: Supervised Learning.

response to new data. Supervised learning based problems can be grouped into two kinds: Classification and regression model. The main difference between the two tasks is the fact that the dependent attribute is numerical for regression and categorical for classification.

Classification Algorithms: Classification algorithms build predictive models using features learned from training data which have features and class labels. This model is used to predict class labels on previously unseen new data. The output class for classification is discrete, which attempts to estimate function (f) from the input variables (x) to discrete or categorical output variables (y). Decision trees, random forests, J48, support vector machines, random tree etc. are a couple of classification algorithm.

Regression Algorithms: Regression algorithms utilize input features and output values obtained from the data to build a model to predict unknown output values based on the training data. The output values are not discrete but continuous. These kinds of algorithms attempt to assess the mapping function (f) from the input variables (x) to numerical or continuous output variables (y). Linear regression, pace regression, regression trees, and lasso regression, simple linear regression etc. are a couple of different types of regression algorithm.

CHAPTER 3

FAULT TOLERANCE

The capability of a system to continue to perform its specified tasks after the occurrence of faults is termed as fault tolerance (34). Fault tolerance is associated with successful operation, reliability, and the absence of breakdowns. A fault-tolerant system ought to have the ability to handle faults in individual software or hardware components, power failures, or other types of unexpected problems while still meeting its specification.

Murphy's law: "If anything can go wrong, it will."(35)

Even though designers ensure the best to cleanse all the hardware defects and software bugs out of the system in advance to the hardware going on the market, history displays that such a goal is not achievable. Inevitably, either unanticipated environmental factor is not taken into account, or some prospective user mistakes are not foreseen. Thus, even though a system is designed and implemented correctly, faults are likely to be caused by situations outside the control of the designers (36).

First, some important terms are explained below:

3.1 Faults, Errors, and Failures

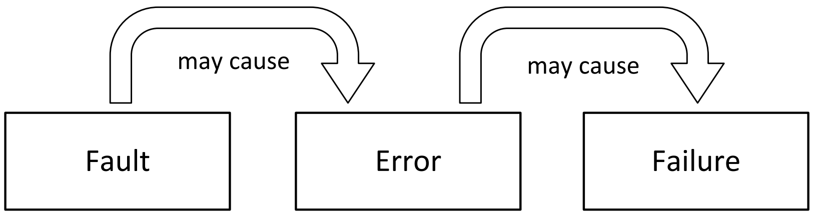


Figure 3.1: Fault, Error, Failure.

Fault: A fault is an imperfection, physical flaw or defect that occurs in hardware or software component; creates one or several latent errors in the component of occurrence. A physical fault can directly affect the physical layer components only, whereas human-made faults may affect any component, e.g., are a short circuit between two adjacent interconnects, a broken pin, or a software bug (36; 37).

We can categorize the origins fault into four domains. They are: Specification mistakes (incorrect requirement, incorrect specification, incorrect algorithms, etc.), component defects (manufacturing defect, random defect, wear out), external causes (electromagnetic interference, noise, radiation, overheating etc.) and implementation mistakes (bad design, coding mistakes etc.).

Error: An error is a deviation from perfection or accuracy in computation, which happens as a result of a fault. Errors are typically associated with incorrect values in the system state. A given error in a given component may be subsequent to different kinds of faults. For example, an error in a physical component (e.g., stuck at ground voltage) may result from: a physical fault acting at the physical layer comprising the component, an information error, caused by a design fault (37). A fault may stay dormant for a long time before it manifests itself as an error.

Failure: A failure is nonperformance of some action which is projected or expected. A system has a failure if the service provided to the user deviates from the compliance with system specification for a specified period. A system may fail either because it does not perform according to the specification, or because the specification did not satisfactorily describe its intended function. A component failure occurs when an error affects the service delivered (as a response to a request) by the component (36; 37). As the manifestation of an error might cause a whole system to deviate from its required operation; one of the primary goals of safety-critical systems is that error should not result in system failure.

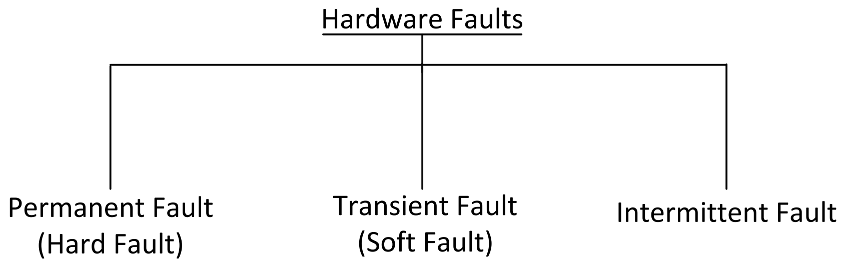


Figure 3.2: Hardware Fault.

3.2 Hardware Faults

Hardware faults can be triggered by three types of failure events that affect the hardware of the system: permanent failures of hardware components, temporary malfunctions of components and external interference through computer operation (38).

The faults can be classified depending on the duration, the extent and/or the value. The duration of a fault can be temporary or permanent, whereas the extent refers to several logic instantaneously affected by the fault. When there is an occurrence of a fault, the value express whether the fault assumes a constant value (e.g., "stuck at 0" or "stuck at 1") throughout the whole duration or it varies.

Permanent Faults/Hard Fault: Permanent faults are caused by permanent (also called "hard") failures or physical changes in components of a logic circuit which permanently modify the logic function defined by the designer. A permanent fault will remain active until corrective action is taken, such as the addition of a duplicate (spare) component, or provisions to continue operation without the failed part. Some physical defects usually cause these faults in the hardware, such as broken interconnections, shorts in a circuit or stuck cells in memory.

Transient Faults/Soft Errors: Though in late seventies single event upsets (SEU) occurred in semiconductor memories by radioactive contamination and the cosmic ray flux, nowadays logic circuitry, both sequential and combinational, are also getting increasingly susceptible to SEU (39). So, a single particle can either hit the combinational logic or the sequential logic part in the silicon (40).

Due to lower supply voltages and higher VLSI complexity; the occurrence rate of particle-induced transient faults, also known as soft faults, have increased. A soft error occurs when a radiation event causes enough charge disturbance to flip or reverse the data state of a latch, memory cell, register, or flip-flop. The error is called "soft" because the radiation does not permanently damage the circuit/device itself— the device will store it correctly if a new data is written to the bit (41).

Lower cell capacity, critical charge and supply voltage lead to higher soft error rates (SERs) (42). It was presumed that combinational circuits were not as susceptible to SEU in most previous works. However, authors of (5) showed that combinational circuits could be as susceptible as memories in new circuits predominately due to the continuous feature size reduction and number of pipeline stages increment, combined with escalated clock frequencies.

Transient faults mostly caused by environmental factors such as alpha particles, beta particles, atmospheric neutrons, electrostatic discharge, cosmic radiation, electrical power drops or overheating are faults with specific duration that is detected through the errors resulting in their propagation logic (5; 38; 39). Alpha particles and beta particles are the nuclei of helium atoms and electrons enhanced to relativistic speed respectively whereas cosmic radiation is a combination of primarily protons and a smaller quantity of heavy ions speeded up to relativistic speeds (43; 44). External causes (electromagnetic interference, noise in power supply, etc. or by temporary circuit malfunctions, e.g., overheating, overload conditions, etc.) may simultaneously produce various transient faults throughout the system. Therefore independent occurrence cannot be assumed for all transient faults (45).

Intermittent Faults: Intermittent faults are results of implementation flaws, aging, and wear-out, and to unexpected operating conditions. For example, a loose solder joint in combination with vibration can cause an intermittent fault. Upon fault activation, this fault creates errors to occur in bursts, at the same location, so replacement of the malfunctioning circuit removes the intermittent fault. The fault may oscillate between being quiescent and active. Intermittent faults often times precede the occurrence of permanent faults.

Common Mode Failure (CMF): CMF can be described as a fault that occurs at multiple modules at the same time. Based on the fault, these faults have the potential to propagate to the output manifesting as wrong, but identical outputs in two or more modules. Thus, the voter will not be able to detect this fault, leading to a potentially dangerous situation. The root cause of this kind of fault is the dependencies among the redundant units that lead to simultaneous failure. Such as:

- common environmental factors,
- common buses within a system,
– a common source of power,
– design mistake.

3.3 Fault Tolerance and Redundancy

Fault prevention, fault removal, fault forecasting and fault tolerance are four techniques to develop a dependable system. Fault prevention is to avoid the occurrence of any faults which can be done by quality control though it is not possible to achieve 100% prevention. Fault removal can be done during both development and operation stage of a system employing verification, diagnosis and correction. Fault forecasting is done to qualitatively and quantitatively estimate the fault that can happen in the future and the consequences.

A system will fail if it ceases to perform its intended functions. Fault tolerance is necessary because it is basically not possible to build a perfectly flawless system. With the increasing complexity of a system, the reliability will decrease along with it if no tolerance measures are taken. The interest in the topic of fault tolerance increased substantially after world war II with the space race being another catalyst (36).

The most common and reliable approach to achieve fault tolerance is redundancy. Redundancy can be attained through hardware replication, check bit with a string of data, an additional line of the program, among others. In 1950s, in the work called "Probabilistic logic and synthesis of reliable organisms from unreliable components" John von Neumann established the idea of incorporating redundancy in order to improve the reliability of a system (46).

3.3.1 Types of redundancy

There are four kinds of redundancy that are studied: hardware, information, time and software. Hardware faults are typically dealt with by using hardware, information, or time redundancy, whilst software faults are protected against by software redundancy.

Software Redundancy: Software redundancy techniques provide structures to the software system to prevent system failure from happening in case of a fault occurrence.

Single-version techniques include mechanisms for fault detection, containment, and recovery to increase fault tolerance. This helps to prevent fault propagation for fault containment and recovery. Redundant software components, majorly, recovery blocks, N-version programming, and N self-checking programming following diversity rules are used in multi-version software-based techniques. The secondary versions can be based on more straightforward and less accurate algorithms to be used only when the failure of the primary software produce unacceptable results. Just like hardware redundancy, the multiple versions of the program can be executed either concurrently (requiring redundant hardware as well) or sequentially (requiring extra time, i.e., time redundancy) upon a failure detection (47).

Time Redundancy: Time redundancy is reiterating the execution of some of the program critical functions, computation or data transmission multiple times to compare and reach a consensus. Hardware redundancy impacts the size, performance, power consumption, and cost of a system. It is more suitable to use extra time rather than extra hardware to tolerate faults in some applications.

Fig. 3.3, depicts time redundancy in case of a TMR system. In (48), the authors, investigates time redundancy method to increase the reliability of a fully automated behavioral C-Based MPSoCs on a parameterizable architecture.

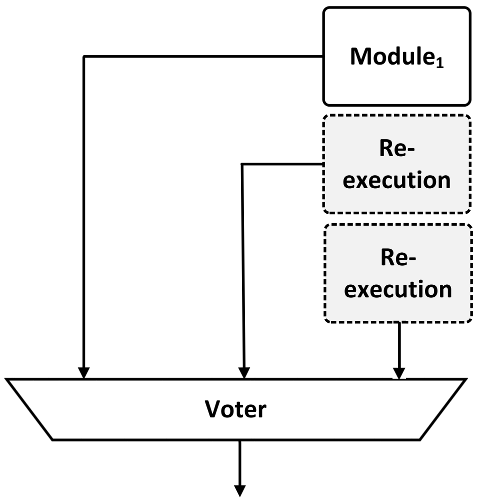


Figure 3.3: Time Redundancy: Example.

Information Redundancy:

Information errors can be tolerated by leveraging coding techniques such as parity codes, linear codes, arithmetic codes, cyclic codes, and unordered codes which will evade unwanted information deviations during data storage or transmission (36).

Claude Shannon (49) and Richard Hamming (50) were the two pioneers to originate coding theory using information redundancy changing the solitary use code triplication. The goal for Shanon's work was to compress a message in order to minimize the total number of transmitted symbols while allowing the receiver to recover the message. Hamming introduced the concept of hamming distance where a set of error-correction codes that can be used to detect and correct the errors.

Hardware Redundancy:

Redundancy can also be attained by repeating all or in parts of the functioning design units, components, or data items, with the same or diverse designs, which is called hardware space redundancy.

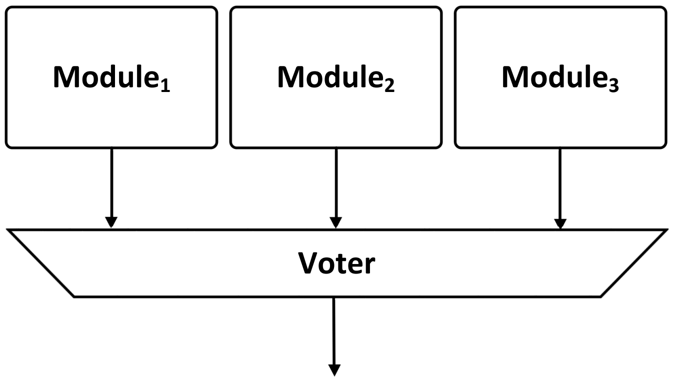


Figure 3.4: Space Redundancy: Example.

The principal idea behind hardware redundancy is to utilize NMR. N-modular redundancy (NMR) techniques like duplication with compare (DWC) and Triple Modular Redundancy (TMR) are widely used for designing dependable systems with data integrity. Data integrity is maintained if a system either produces correct outputs or generates an error signal when incorrect outputs are produced. NMR is a majority voting system with N identical hardware modules connected in parallel (51). There are two general cases: Duplication with Compare (DWC) when N=2 and Triple-Modular Redundancy (TMR) where N=3. If case of N=3, the structure can have the ability of fault tolerance. In case of N=2, errors can only be detected but not corrected and a full re-computation is needed. The evident advantage of DWC is that it requires less area than TMR systems. Fig. 3.4, depicts hardware redundancy in case of a TMR system. Redundant systems can be very effective in masking errors. Either transient due to, e.g., radiation single event upsets (SEU) and also permanent faults due to, e.g., electromigration or time-dependent dielectric breakdown (TDDB).

In (52), authors introduce an N-modular redundancy method that takes a single behavioral description as input for HLS and performs a source-to-source transformation to directly annotate the N-modular redundancy on the behavioral descriptions to provide tolerance against soft errors.

Adding redundant components to the circuit can help in tolerating manufacturing defects and thus increase the yield. However, too much redundancy may reduce the yield since a larger-area circuit is expected to have a more significant number of defects. Moreover, the increased area of the individual chip will result in a reduction in the number of chips that can fit in a fixed-area wafer. Successful designs of defect-tolerant chips must, therefore, rely on accurate yield projections to determine the optimal amount of redundancy to be added.

Hardware redundancy can be employed in three different forms: Passive, Active and Hybrid (36; 47).

Passive/Static: Passive or static redundancy approach uses the concept of fault masking where the system depends on the voting mechanism to achieve fault tolerance. TMR and extending to N-modular redundancy (NMR) are passive redundancy.

Active/Dynamic: Active or dynamic techniques achieve fault tolerance by first detecting the occurrence of faults and performing necessary action to remove the fault and return to the operation state of the system. These techniques use fault detection, location, and recovery in an effort to attain fault tolerance. Duplication with Comparison, Standby Redundancy, Pair-And-A-Spare are some examples of active redundancy.

Hybrid: Hybrid techniques combine the smart features of both passive and active approaches. Hybrid redundancy combines the advantages of passive and active approaches. Fault masking is used to prevent the system from producing momentary erroneous results. Self-Purging Redundancy and N-Modular Redundancy with Spares are examples of hybrid redundancy techniques.

In the next section, the need and importance for fault recuperation in hardware accelerators are explained.

3.4 Why Fault Recuperation in Hardware Accelerator is important?

Aggressive scaling of transistor sizes has driven these remarkable improvements in computational performance. Even though transistor count and chip density keep growing, ever stricter technology constraints are compelling a revisit to a whole new dynamic in VLSI design.

One promising resolution to deal with the breakdown of Dennard scaling is the customization of the computing platforms to the application domain, also known as domain specific computing. Customization is achieved by using CPUs with FPGA at the computer level and through heterogeneous Multiprocessor Systems-on-Chips (MPSoCs) with dedicated Hardware Accelerators (HWacc) at the chip level.

This dissertation mainly targets hardware accelerators because these often tolerate inaccuracies in their computations and because they are typically the main differentiating components between different IC vendors. Due to the evolution of technology constraints, especially energy/power constraints; modern complex heterogeneous SoCs, large portions of the computations are offloaded to dedicated hardware accelerators that can process these applications more efficiently. At the same time, reduced turn-around-time with the pressure to deliver these new or updated ICs at shorter time periods and the use of new technologies make these accelerators more prone to hardware errors compared to more mature and tested IPs.

The novel architecture of heterogeneous SoCs can provide sufficient application coverage, energy efficiency and performance. More attention needs to be given to find a balance between energy efficiency and fault tolerance while designing hardware accelerators. In safety critical systems as health monitoring system, autonomous vehicles or space crafts, lack of fault tolerance can have catastrophic consequences.

The accelerators consist of combinational and sequential circuits are susceptible to the impact of permanent and transient faults. To achieve fault tolerant hardware accelerators, the inherit circuits should be tolerant against these faults. The balance between area, performance and fault tolerance is essential to design an efficient modern heterogeneous SoC.

The Soft Error Rate (SER) of advanced CMOS devices is higher than all other reliability mechanisms combined. Soft errors have recently become a massive apprehension in cutting-edge CMOS products because if left uncorrected; they have the power to induce a failure rate that is higher than all the other reliability concerns altogether. A soft error occurrence in a circuit though does not permanently damage the circuit but when enough charge from a radiation event is collected in a node, it will corrupt the data state of the circuit(41).

Transient faults or soft faults are the predominant faults in modern technologies, can be caused by environmental conditions like temperature, pressure, humidity, voltage, power supply, vibrations, fluctuations and electromagnetic interferences, crosstalk. (53). Due to lower supply voltages and higher VLSI complexity; the occurrence rate of particle-induced soft faults have increased. Lower cell capacity, critical charge and supply voltage lead to higher soft error rates (SERs) (42). In most previous works, it was assumed that combinational circuits were not as susceptible to SEU as historically, soft errors were a concern in the design of memory elements, but the susceptibility of the combinational blocks to transient faults increases as a side effect of technological scaling. In addition, authors of (5) has shown that combinational circuits can be as susceptible as memories in new circuits mainly due to the continuous feature size reduction and the increase in the number of pipeline stages, combined with higher clock frequencies. There is a lot of existing research done for memories to protect against soft errors, but mitigation techniques for SER in logic circuits are not researched enough.

Manufacturing defects or hard faults can coarsely be classified into spot defects and global defects, both contributing to yield loss. Wafer mishandling scratches, over etching, under etching, mask misalignments etc. cause large defects known as global defects.

Spot defects are random small and local defects generated during the various steps of fabrication. It is mostly resulted from the materials employed in the process and from environmental causes, generally the consequence of undesired chemical and airborne particles deposited on the chip/wafer. In a well-controlled mature fabrication line, gross/global defects can be minimized to a meager percentage. However, controlling spot defects are more challenging to manage as a result of yield loss due to spot defect is inevitable and much higher than the global defect (47).

3.5 Related Work

In this section, we review the main prior work in the area of fault-tolerance in HLS, soft errors, CMF tolerance, automated diverse system generation and permanent fault compensation.

3.5.1 Fault tolerance using HLS

With regard to fault-tolerant HLS, Karri and Orailoglu developed the first HLS based fault tolerance methodology to maximize performance in the presence of fault-tolerance constraints for ASICs (54; 55). Their primary approach was to minimize the area overhead by sharing functional units. Authors of (56) extended this work by proposing a hybrid time and hardware redundancy system. Guerra et al. (57) proposed a technique for Built-in Self Repair (BISR) in HLS to achieve fault tolerant circuits which exploits different resource allocation, scheduling and bindings combined with transformations . Antola et al.(58) while keeping the error aliasing small extended this work to exploit resource sharing to minimize the redundancy. Later in (59), the authors pre-characterized different functional units' implementations, e.g., ripple carry adder, Brent-Kung adder and Kogge-stone adders, and annotated the reliability into the area and delay library used for HLS, leaving it up to the logic synthesis tools to implement these accordingly to meet the delay constraint. The approach utilizes high level synthesis and characterize library components in terms of reliability maximization by choosing the most efficient resource allocation, binding and scheduling. This work studies Soft Error Rate (SER) in combinational components, as a the function of component area, while the components relative SER is obtained by setting the reliability of ripple carry adder to 0.999.

3.5.2 Fault tolerance to protect against soft-error

The authors in (60) propose increasing the reliability using redundant operations and multiple binding of resources by estimating the reliability of different Data Flow Graphs (DFGs). Both works consider the reliability of combinational logic as a function of the component area. In (61), the authors investigate a comparative analysis of the area and delay constraints in synthesis tools on reliability. They use Error Propagation Probability (EPP), defined by (62), in a component that manifests in the output, as a metric to compare the reliability of different components.

A HLS method of fault-tolerant multi-cycle datapaths is proposed in (63). They propose a heuristics method for soft error detection and correction utilizing the TMR (triple-module redundant) system for k-cycle fault tolerance. Recently in (64), the authors integrate vulnerabilities into the register binding phase of HLS along with a selective register protection scheme. Also, the authors in (65) introduced scheduling and binding heuristic for HLS that explores tradeoffs between resource usage, latency, and the amount of redundancy.

More recently, the authors in (48) proposed a reliable system that makes use of the slack available in shared-bus architectures to re-compute the outputs of the slaves synthesized using HLS. The authors in (66) follow a similar approach. Finally, in (67) the authors introduce a compiler pass called StitchUp, which identifies the control critical parts of the design in HLS, then automatically replicates only those parts.

Our proposed methodology allows choosing from a pool of micro-architectures with different levels of fault-tolerance to build the best TMR system under a given set of constraints (i.e., area and latency). In addition, it presents a method to generate this pool automatically.

3.5.3 Diversity automation for fault tolerance

With regard to asymmetric, diverse fault tolerant redundant systems, in the software domain, much work has been done since the 1970's (68). This concept has recently also started to being applied to security (69). This previous work mainly focused on developing cost-effective ways to add diversity to the software automatically. N-version programming (NVP) was defined by Avizienis et al. (68) as the independent generation of $N \geq 2$ functionally equivalent programs from the same initial specifications. Initially, *independent generation* meant that different programming teams, without any interaction, would create these programs separately. Since then, automated techniques for NVP have been developed. Diverse compiling is achieved by using different compilers (e.g., gcc, LLVM or icc) and setting a different level of compiler optimizations (e.g., gcc -01, 02, -03). The main objective is to minimize software faults in two or more versions (70).

The hardware community also adopted the idea of diversity in order to protect against CMFs by perturbating the logic gate-netlist (71; 72; 73; 74). The core concept of diversity depends on *independent* generation of *different* implementations. In order to measure the effectiveness against CMFs, in (71), the authors proposed a metric to calculate *diversity*, which requires the fault injection at individual points of the netlist of the different netlists being evaluated.

Although accurate in measuring against CMFs, the Dmetric method has been shown to be an NP-complete problem(75). To address this, recently, the authors of (76) proposed a new diversity metric called DIversity Metric based on circuit Path analysis (DIMP) based on the structural path analysis of the gate netlist. They nevertheless do not compare it against the diversity metric proposed by (72) and hence, do not prove that maximizing the DIMP between two designs also maximizes the fault-tolerance against CMFs. The Dmetric calculation is explained more in Section: 6.1.1.

This previous works mostly concentrated on generating a diverse design pair at the gate-level granularity. In (77), the authors proposed a gate level exploration technique using the Dmetric as a cost function and maximize the diversity of a duplex system. With a given truth table of a combinational circuit N1 to be implemented, the objective was to synthesize another implementation N2 where the diversity of N2 with respect to N1 is maximized. The work is restricted to two-level synthesis in AND-OR form, generating multiple-output-prime-implicants (MOPIs) (78) and constructing MOPI covering table.

In order to generate different gate implementations of each circuit, in (76), the authors used synthesis optimization for different delay targets, from most demanding to most relaxed.

Our work is different from this previous work in multiple dimensions. First, it expands the search of diverse systems by raising the level of abstraction and starting from a unique behavioral description for HLS. Secondly, it proposes a new diversity metric based on a predictive model that makes use of the synthesis results reported right after HLS in order to find micro-architectures with the highest diversity. In this work, we also analyze if the DIMP diversity metric is useful to protect against CMFs.

3.5.4 Permanent fault Compensation

There has been significant work on detecting and correcting errors in hardware. Many of these techniques focus on using structural or information redundancy (i.e., error correcting codes) to make a particular circuit or subsystem reliable (79).

For test-related applications, parity trees have been used extensively (80). While most prior works focus on the detection of permanent faults, this work focuses on compensating for the impact of the permanent faults on the circuit output.

Traditional circuit and redundancy based solutions, such as dual modular redundancy for tolerating hardware errors, have become too expensive in terms of area, power and performance overheads; inspiring new low-cost reliability solution (81).

The increasing unreliability of hardware has generated a strong stimulus for the investigation of hardware fault detection, tolerance and recovery. In the context of timing errors in DSP and multimedia systems, researchers have shown that statistical methods can be used to build compensation models and improve overall accuracy (82).

Some of the previous works have addressed the use of error compensation in the context of digital signal processing (DSP) systems . Previous work in (82; 83), has proposed algorithmic noise-tolerance (ANT) techniques for DSP systems where timing errors are permitted to occur and then corrected by a statistical error control block.

In (84; 85), an approximate implementation of the main DSP block is used to estimate the output and provide reliability in the presence of timing errors introduced by voltage over-scaling. In (83), errors are corrected by using a composite error probability mass function to compute the probability ratio for each output bit.

Authors of (86) developed a measure to quantify code's vulnerability to permanent or intermittent faults, and suggested a transformation technique to reduce the worst case fault rate. On the other hand, (87) suggested a rollback and re-execute approach by comparing the faulty circuit with a fault-free core.

Our approach is focused on basic benchmarks circuits and uses techniques focused on predicting the entire output. The disadvantage of such a method is as the number of prediction feature directly relates to the number of samples, which creates a significant area overhead for larger circuits.

The accuracy of the machine learning algorithms decreases as the number of features for prediction increases. As the number of unique values in the target feature increases, it provides a negative effect on the accuracy of the prediction, because the probability of a predicted sample to end up in a wrong boundary increases with the size of the unique value.

We investigate an approach for building compensation logic for dedicated hardware accelerators amenable to approximate computing using supervised learning algorithms (i.e., inferring a compensation function based on the set of circuit behavior training data acquired during testing). For the proposed method, we do not need to generate bit-wise results for each output, which provides the approach more flexibility, as opposed to using statistical or Bayesian methods.

CHAPTER 4
DESIGN AND OPTIMIZATION OF RELIABLE HARDWARE ACCELERATORS: LEVERAGING THE ADVANTAGES OF HIGH-LEVEL SYNTHESIS

The need for high performance and energy efficiency has led to a trend shift towards heterogeneous System-on-Chip (SoC), which includes multiple dedicated hardware accelerators. At the same time, these heterogeneous SoCs have to be taped out at shorter and shorter time frames. This is forcing designers to raise the level of abstraction by using High-Level Synthesis. Shrinking geometries, lower operating voltages, higher operating frequencies and higher density in these accelerators have led to an increased sensitivity towards soft errors. These occur when a radiation event causes a disturbance significant enough to reverse a bit in the circuit. When the bit is flipped in a critical control register or configuration memory, it can cause the circuit to malfunction (88). This phenomenon raises the need for fault-tolerant hardware accelerator design.

Fault-tolerance can be described as the ability of a system to continue to work after the occurrence of an error. Fault-tolerant electronic systems are becoming a standard requirement in many industries ranging from aeronautics to automotive as electronics have become pervasive in many industries.

Most previous work on VLSI design reliability is based on time or space redundancy. In space redundancy, the main idea is to replicate the same channel N times, also called N-Modular Redundancy (NMR). For larger electronic systems these channels are being built by sourcing parts from different vendors. However, in VLSI design, the underlying hardware channel is exactly the same since it is too expensive to design multiple hardware versions with the same specifications. The Triple Module Redundancy (TMR) is one of the most popular implementations of NMR systems as it can fully mask errors. Fig. 4.1 shows the different TMR configurations using space and time redundancy.

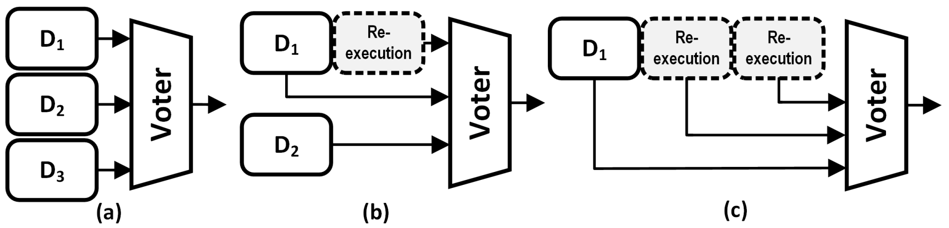


Figure 4.1: TMR Accelerator Variation (a) 3 same/different design in parallel. (b) 2 different/same design in parallel and one of them re-executed (c) One design re-executed 3 times.

There are multiple problems when implementing TMR system at the RT-level: (1) Designers tend to over-design the circuit by instantiating the same module thrice as shown in Fig. 4.1(a), because it is easier to verify than using time redundancy. (2) The underlying hardware is always the same, thus, in Fig. 4.1(a) $D_1 = D_2 = D_3$. This implies that if there is a fault/bug in the design, it will be present in all of them, and hence it will never be detected, as the voter will always see the exact same output from the different modules. (3) It is extremely hard to re-target the system when moving to another platform/system, with different constraints (*i.e.*, area, execution time, power) as this involves having to re-design and re-verify the entire system, which is time- consuming and error-prone.

Reliability is tackled very differently in the software domain. In contrast to most hardware faults, software faults (bugs) exist in every instance of the software. 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 (69). A classic approach to add diversity is to set up multiple teams working in parallel and independently on the same design (89). This obviously expensive method is 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 (70).

Thus, this work applies similar techniques to C-based VLSI design to find multiple different micro-architectures from a single behavioral description and then given an area and timing constraint, find the most reliable TMR accelerator system design from the three options shown in Fig. 4.1. In this work, we build the most reliable TMR accelerator with respect to space and time within the given constraints by leveraging the unique advantage of C-based VLSI design of allowing the generation of different micro-architectures from the same behavioral description.

4.1 Motivation and Flow Overview

Fig. 4.2 compares the traditional fault tolerant flow with our proposed method. Tradi-tional VLSI design flow of an accelerator starts with a behavioral description and some constraints, e.g., real-time or power, that needs to be mapped onto hardware. In the tra-ditional flow, Fig. 4.2(a), the hardware designer has to analyze the behavioral description and create a suitable RTL description that can be executed manually. This leads to a single micro-architecture with specific execution time, area and reliability. This module is in turn, instantiated thrice with a voter to mask faults. As mentioned previously, this implies that the system might be over-designed as time redundancy could be used to reduce the area and that the reliability of the system is compromised if there is a bug in the hardware mod-ule. Thus, this work advocates a new approach to fault-tolerance, based on leveraging one of the main advantages of C-based VLSI design: the ability to generate micro-architecture with different characteristics from the same behavioral description. Fig. 4.2(b) highlights the proposed flow. In C-based VLSI design, the first step typically requires to refining the original behavioral description to make it synthesizable. This synthesizable description can, in turn, be explored creating designs with a unique area, time and reliability. Finally, the proposed method creates the system with the highest reliability by creating a space, time or mixed redundant TMR system as shown in Fig. 4.1 based on the constraints given (area and timing).

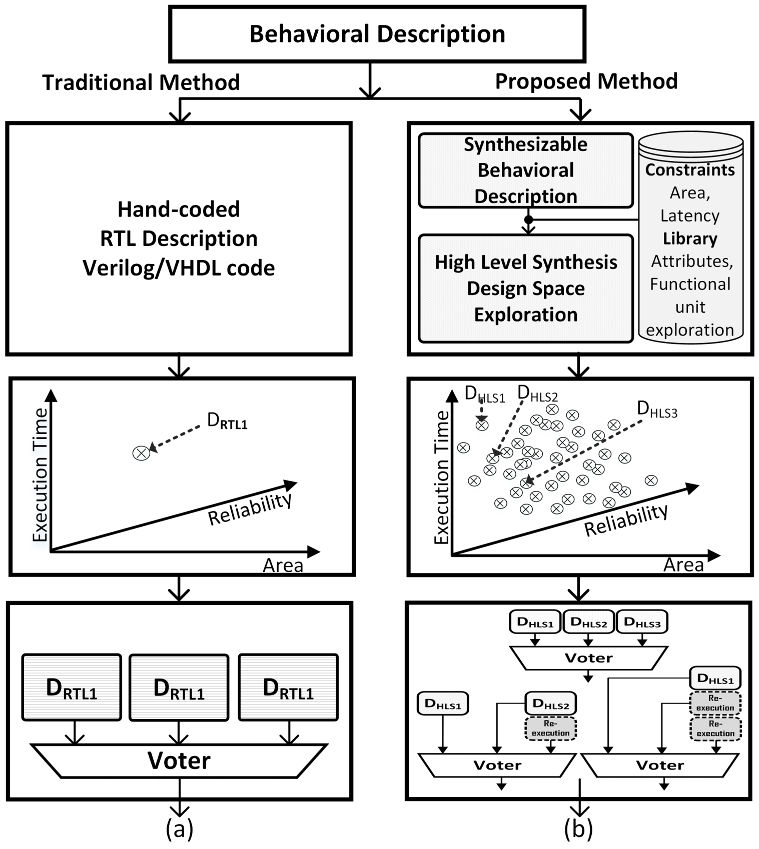

Behavioral Description
Traditional Method
Proposed Method
Hand-coded RTL Description Verilog/VHDL code
Synthesizable Behavioral Description
Constraints Area, Latency
Library Attributes, Functional unit exploration
High Level Synthesis Design Space Exploration
Execution Time
Reliability
Area
D_{RTL1}
D_{HLS1}
D_{HLS2}
D_{HLS3}
Voter
Re-execution
(a)
(b)

Figure 4.2: Design flow of traditional and proposed method.

Thus, C-based VLSI design seems a promising technology for automatic generation of reliable systems. The problem addressed in this work can be formulated as follow:

Problem Formulation: Given a behavioral description (C), area ($AreaBound$) and execution time ($TimeBound$) constraints, automatically generate a design pool $DP = \{D_1, D_2, \ldots, D_n\}$, from C with each design $D_i = \{A_i, T_i, R_i\}$ having a unique combination of area (A), time (T) and reliability (R), optimize a triple modular redundant (TMR) accelerator system with maximum reliability by choosing the best mix of designs from the DP.

4.2 ARMORED: Proposed Method

The proposed method, called ARMORED (A Reliable triple-MOdular REdundant accelerator Design), is composed of two main phases summarized in algorithm 1. The first phase performs the HLS designs space exploration (DSE) to generate the pool of designs (DP) to choose from, while the second phase determines which designs to choose to form the TMR system, given an area and execution time constraint. These two phases are described in detail below.

4.2.1 Phase 1: Reliability-aware Design Space Exploration

Although much work has been done in the past in the area of HLS DSE (19; 22), most work only target area and latency or throughput as the exploration target, as the HLS tools report these two metrics after the synthesis. In this work, we have included reliability as a third exploration metric in order to explore the three-dimensional space of area, time and reliability, but because commercial HLS tools do not provide reliability information, it is required to estimate the reliability of each new designs after HLS during the exploration.

Reliability Estimation: For estimating the reliability of each newly generated design, this work considers soft-errors caused by high-energy neutrons. Reliability is calculated as the negative exponential of failure rate multiplied by total execution time as given in (90), as follows:

$$R = e^{-\lambda . t} \tag{4.1}$$

In this case, the design failure rate (λ) is given as:

$$\lambda = SER \times AVF \tag{4.2}$$

where, AVF stands for Architectural Vulnerability Factor, defined as the probability that an error (such as soft-error) results in failure, i.e., changes the sensible output of the architecture

(91). In this work, the AVF value is considered as 1 which means a failure will always be interpreted as an error. We obtain the Soft-Error Rate (SER) for combinational logic, latches, and SRAMs from the model by authors of (5) as 0.00003, 0.00001 and 0.000009 respectively. For obtaining the total SER of a specific design, this method multiplies the number of logic elements to the SER per logic and adds them up. Here, t is the design execution time, obtained as:

$$t = Latency * Critical\ Path\ Delay * No\ of\ Runs \tag{4.3}$$

where, Latency is the number of cycles the design takes to generate a new output and Critical Path Delay is the longest combinational delay in the design. Both are obtained from the HLS tool. No of Runs is the number of times the module computes data. This work considers 10 million runs for the calculation.

Algorithm 1: Proposed flow overview.

```
input  : $\{C, AreaBound, TimeBound, \Delta\}$
1   $C$ : Input Behavioral Description
2   $AreaBound$: Area constraint of $TMR$ system
3   $TimeBound$: Execution Time constraint of $TMR$ system
4   $\Delta$: Threshold value to decided which non-Pareto-optimal designs to use in
    $TMR$ system
output: $\{TMR = \{D_1, D_2, D_3\}\}$
5   $TMR$: $TMR$ system with the highest reliability for given constraints
6   $D_i$: Micro-architecture of TMR system
7     /* Phase 1: Reliability-aware HLS DSE */
8     $DP_{pragma} = \{D_1, D_2, ...D_i\} = pragma\_explore(C)$;
9     $DP_{FUs} = \{D_1, D_2, ...D_i\} = fu\_explore(DP)$;
10    $DP = DP_{pragma} \cup DP_{FUs}$;
11    $DP_{prune} = prune\_DP(DP, \Delta)$;
12    /* Phase 2: TMR System Design */
13    $TMR = ILP(DP_{prune}, AreaBound, TimeBound)$;
14    return(TMR);
```

Design Space Exploration: The HLS explorer proposed in this work is based on a genetic algorithm (GA) which has shown to lead to good results in multi-objective optimization problems like this one (92). In this case, the two objectives that need to be minimized are area and execution time, while reliability needs to be maximized. Thus, the explorer uses a cost function ($C = \alpha A + \beta L - \gamma R$), that is adaptively modified during the exploration process in order to explore the complete trade-off plane. It should be noted that the reliability term of the cost function is negative because the objective is to minimize area and execution time and maximize reliability.

The explorer is divided into two steps. The first step explores the synthesis attributes in the form of pragmas, while the second stage explores the number of functional units (FUs).

Step 1: Local Synthesis Attributes Exploration. This first step explores synthesis directives in the form of pragmas inserted directly into the source code. It is also the most powerful *knob*, as it fixed the overall micro-architecture. With these pragmas, it is possible to synthesize arrays as memories or registers. Loops can be unrolled or pipelined and functions inlined or synthesized as single Hardware (HW) blocks (gotos) (line 2).

Each explorable operation $OP \in OP_{exp}$ is represented as a gene to which a synthesis attribute (pragma) AT is assigned. The list of all genes builds a chromosome Cr. Therefore a gene $\Leftrightarrow OP$ and a $Cr \Leftrightarrow OP_{exp}$. In this work OP_{exp}={arrays, loops, func}, where arrays={register, expand, logic, RAM, ROM}, loop={no, partial, all, fold}, and func={goto, inline}. The genetic algorithm first step is the random generation of an initial population of N chromosomes. Then, each member of the population is paired with another randomly selected member of the population and is given a chance to produce an offspring using the crossover operator. The crossover operator selects a cut-point at random and combines the left half of one parent with the right half of the other. The crossover operator is only a certain percentage of the time, according to the specified crossover rate specified beforehand by the user, r_c. The offspring is then mutated, which involves randomly selecting one gene within the offspring chromosome and changing it to another random value. Only a certain percentage of the offspring produced are mutated, the proportion of which is determined by the mutation rate again passed by the user as an exploration parameter, r_m. In this work r_c and r_m are set to 0.8 and 0.1 respectively.

At this point, the mutated offspring is synthesized calling the HLS tool and the area, critical path delay and latency of the synthesized design back-annotated. The reliability is then computed using the reliability model described previously, hence leading to a new design $D_{new} = \{A_{new}, L_{new}, R_{new}\}$.

The newly generated offspring will replace one of the parents if one of the several conditions is met: (i) If one of the parents is dominated by the offspring (i.e., is inferior to the offspring across all of the objectives). (ii) If the offspring improves on one or more of the best-so-far values, then the offspring replaces one of its parents (the parent to be replaced is randomly chosen). Moreover, if an identical copy of the offspring already exists within the population, then the offspring is discarded. The algorithm will continue until N number of child designs do not improve any of the parents.

Step 2: FUs Exploration. This second exploration knob enables different levels of resource sharing by controlling the number of Functional Units (FUs) that the synthesizer can use. Resource sharing is a well-known optimization technique that can be used to reduce the area while increasing the latency of the circuit. In resource sharing, a single functional unit (FU) is re-used among different computational operations in the behavioral description. Thus, once step 1 finishes, this next step explores the number of FUs of each micro-architecture. The explorer retrieves the number of FUs required in step 1, which uses the default option of allowing the synthesizer to use as many as required to parallelize the behavioral description fully and reduces all FUs automatically by a fixed percentage (in this case 10% was used) (line 3).

The result of these two steps is a 3-dimensional plane of dominating designs. Below is the algorithm utilized to sort out the pareto-optimal design set.

Algorithm 2: Pareto Optimal Design according to Area, Execution Time and Reliability

```
   input  : Design Pool
1    $D_{total}\{D_1(A_1, T_1, R_1), D_2(A_2, T_2, R_2), ...D_i(A_i, T_i, R_i)\}$
2    $A$: Area
3    $T$: Execution Time
4    $R$: Reliability
5    $i$: Number of Designs
6    $j$: Number of Pareto Designs
   output: $Pareto\ Optimal\ Design$ :
           $D_{PO}\{D_{PO_1}(A_{PO_1}, T_{PO_1}, R_{PO_1}), D_{PO_2}(A_{PO_2}, T_{PO_2}, R_{PO_2}), ...D_{PO_j}(A_{PO_j}, T_{PO_j}, R_{PO_j})\}$
7    /* Sort according to Area */
8    $D_{sort}\{D_1(A_1, T_1, R_1), D_2(A_2, T_2, R_2), ..., D_i(A_i, T_i, R_i)\} = sort(D_{total})$
9    /* Optimal Design Search */
10   for $(i = 1;\ i < Number\ of\ Designs;\ i++)$ do
11       if $(T[i] < temp\_T \ || \ R[i] > temp\_R)$ then
12           if $(T[i] < temp\_T)$ then
13               $temp\_T = T[i]$;
14           end
15           if $(R[i] < temp\_R)$ then
16               $temp\_R = R[i]$;
17           end
18           return($ParetoOptimalDesign : D_{PO}[j]$);
19           $j++$;
20       end
21   end
22   return($Pareto\ Optimal\ Design : D_{PO}$);
```

Intuitively one would prune away all non-optimal designs and pass only the dominating designs to the next phase in order to find the most reliable TMR system. As we show in the experimental results section, this often leads to sub-optimal systems or in some cases, does not even find a solution. Hence, we also have to pass non-dominating designs to the next phase to generate optimal systems. The experimental results study the trade-off of quality of results vs. runtime when passing non-dominating designs to the next phase (Pareto designs + Δ threshold). The un-used designs are pruned away (line 5).

4.2.2 Phase 2: TMR System Design

The input to this second phase is the design pool (DP_{prune}) of designed passed from phase 1. As shown in Fig. 4.1, three possible systems can now be created: (a) Choose 3 design of same/different architecture in parallel and run them simultaneously (Space redundancy). (b) Choose 1 design and re-execute 3 times (Time redundancy). (c) Choose 2 different/same design and re-executed one of them (Space-Time Mixed redundancy).

For this, the overall system reliability is calculated as:

$$R_{total} = 1 - \prod_{i=1}^{3}(1 - R_i)^n \tag{4.4}$$

Here, R_i is the micro-architecture reliability and n is the number of times it is executed (used). Based on this, we formulate the problem as an ILP.

ILP Formulation: Linear programming one of the most popular and efficient method to formulate a mathematical model to calculate maximum profit with minimum cost. The ILP formulation is used in this work is described below:

$$\text{maximize} \quad 1 - \prod_{i=1}^{3}(1 - x_i R_i)^{y_i}$$

$$\text{subject to:}$$

$$\mathrm{A}_1 m_1 x_1 + A_2 m_2 x_2 \cdots + A_i m_i x_i \leq AreaBound$$

$$T_1 n_1 x_1 + T_2 n_2 x_2 \cdots + T_i n_i x_i \leq TimeBound$$

$$x_1 + x_2 \cdots + x_i = 3$$

$$x_i = \begin{cases} 1, & \text{if } m_i \geq 1 \\ 1, & \text{if } n_i \geq 1 \\ 0, & \text{otherwise} \end{cases} \qquad \mathrm{y}_i = \begin{cases} m_i, & \text{if } m_i \geq 1 \\ n_i, & \text{if } n_i \geq 1 \\ 0, & \text{otherwise} \end{cases}$$

In the constraints of the formulation, the $AreaBound$ has to be less than or equal to the total area of the micro-architectures used and the $TimeBound$ has to be less than or equal to the total execution time of the micro-architectures used. Here, m_i and n_i represents how many times a single micro-architecture is placed and executed, respectively. The objective of the ILP formulation, is to maximize the TMR system reliability expressed as $1 - \prod_{i=1}^{3}(1 - x_i R_i)^{y_i}$, where x_i represents if a design is used or not and y_i represents how many times a design is used. To design a TMR system, the total maximum number that a design can be used in the system is set to 3.

4.3 Experimental Results

Six different benchmarks from the open source synthesizable SystemC benchmark suite (S2CBench) (93) are used to evaluate the efficiency of our proposed method. The HLS tool used is CyberWorkBench from NEC (26) and the target technology is Nangate Opensource 45nm. The HLS target frequency is fixed in all cases to 100MHz. The experiments are conducted on an Intel i7-6700 @3.50GHZ CPU and 16 GB memory, running CentOS 7. The ILP formulation was done using Solver (94) by FrontlineSolver.

In order to fully characterize our proposed method, different area ($AreaBound$) and time ($TimeBound$) constraints are set. The smallest $AreaBound$ is set to three times the smallest Pareto-optimal design area ($3 \times A_{min}$) found during the HLS DSE, to enable a standard space redundant TMR solution and incremented until three times the largest Pareto-optimal design area ($3 \times A_{max}$). The same is done with the $TimeBound$, and the constraints combined for practical reasons.

Three methods are compared. The first is a naïve method, which selects the smallest design which instantiated three times meets the $AreaBound$ and $TimeBound$ constraints from all the design pool composed of all the designs created during the exploration phase. The second method is based on the ARMORED (ARMORED:Pareto), but only uses the

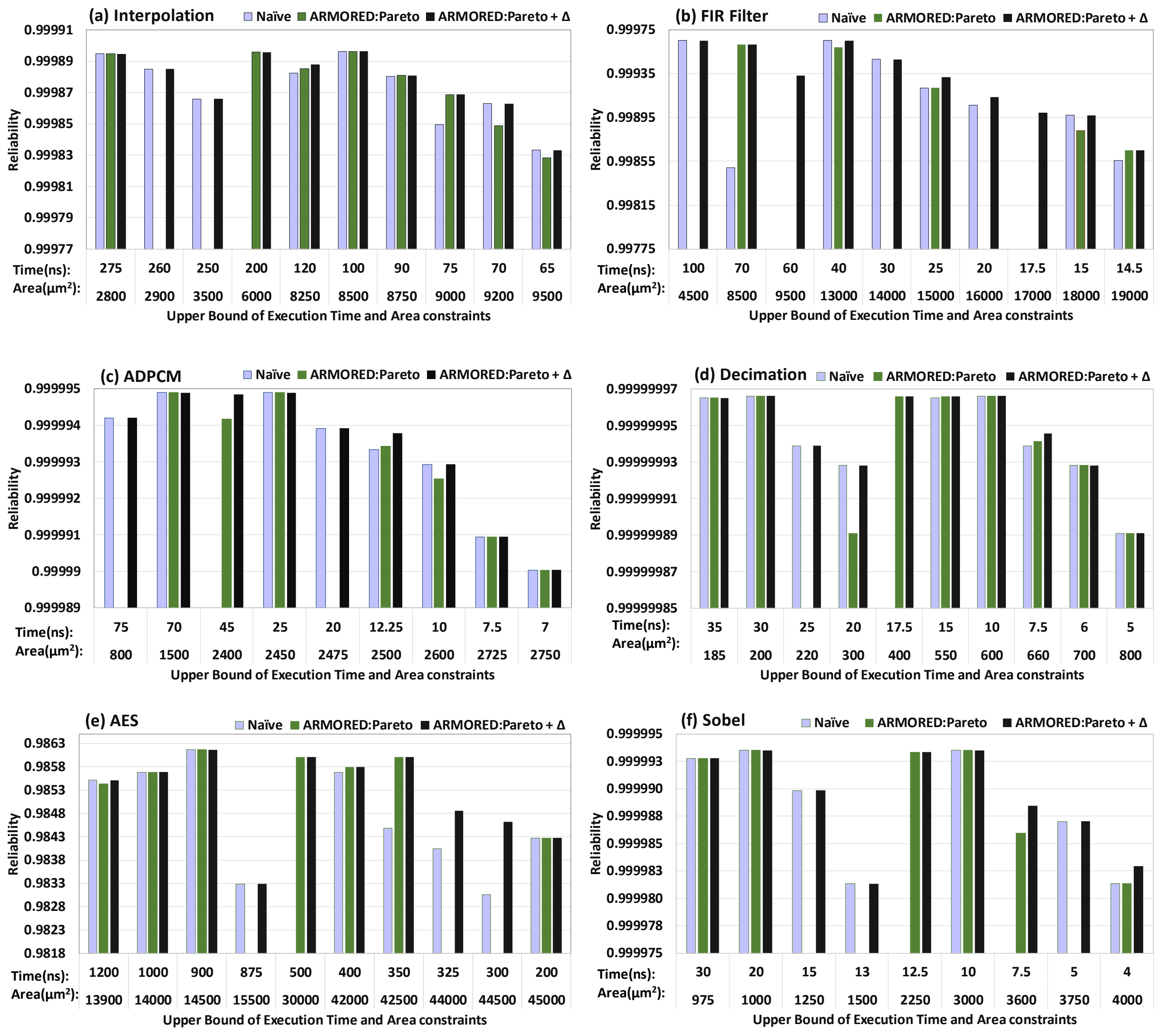


Figure 4.3: Sytem reliability of TMR system for 6 different benchmarks.

Pareto-optimal designs found during the exploration to build the systems. Finally, the last method is again based on ARMORED, but uses the Pareto-optimal designs $+\Delta$ Area or $+\Delta$ Latency, where in this case, $\Delta = 5\%$ (ARMORED:Pareto+Δ). We have also observed that additional designs selected through choosing Δ>5% do not contribute to improving the results in any benchmark. This second method is crucial because intuitively one would only use the Pareto-optimal designs found during the HLS DSE to build reliable systems. Nevertheless, the results show that in this case, often sub-optimal solutions are found as well as no solution at all, as in many cases using non-Pareto-optimal designs lead to an optimal system.

Fig. 6.7 shows the results obtained for the 3 methods, where the x-axis shows the $TimeBound$ constraint in ns and the $AreaBound$ constraint in μm^2. Several observations can be made. First, the naïve and ARMORED:Pareto method does not find any solutions under some constraints. This involves that either the area and/or time constraints have to be relaxed, leading to larger and/or slower circuits. Secondly, the ARMORED:Pareto+Δ, always finds a system that meets the constraints and is consistently better than the other two methods. Fig. 4.4 highlights the reliability average overall constraints boundary of all benchmarks for the three methods. It can be observed that the systems created by only using the Pareto-optimal results lead to poor average reliability, mainly because it often cannot find a solution, similar to the naïve method. Table 4.1, shows the reliability percentage increase of utilizing ARMORED:Pareto+Δ method over traditional method.

Table 4.1: Average Improvement of Reliability after utilizing ARMORED

Design#	Interpolation	FIR	ADPCM	Decimation	AES	Sobel
Reliability	20% +	50% +	20% +	20% +	20% +	50% +

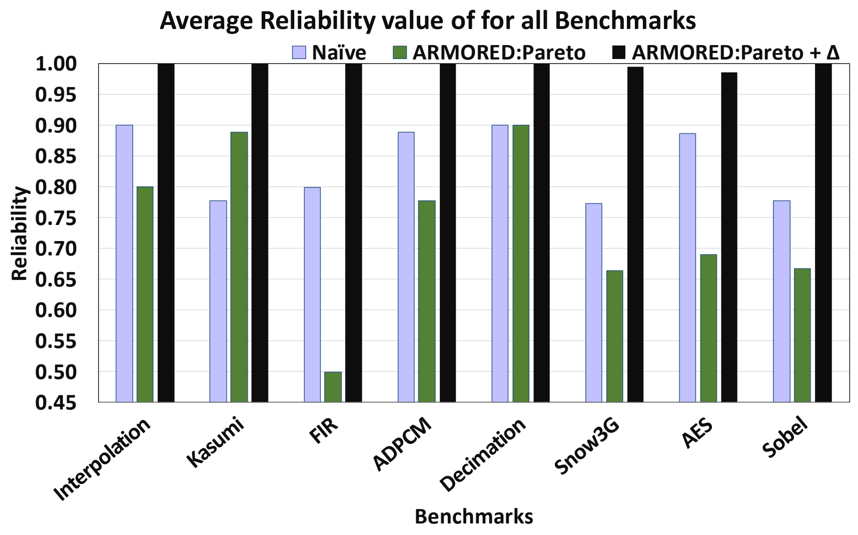


Figure 4.4: Average reliability over all constraints.

4.3.1 Study of Area and Timing Overhead

From Table 4.2, we can observe, for the constraints where both naïve and ARMORED: Pareto+Δ provides a solution, there is an area overhead of ∼ 1% and time overhead of ∼ 2%. The only exception is FIR, where there is an area reduction of 6% for 30% time increase.

Table 4.2: Overhead for ARMORED over naïve method

Design#	Interpolation	FIR	ADPCM	Decimation	AES	Sobel
Area(μm^2)	0.4% +	6.6% -	0.8% +	1.3% +	0.01% +	1.1% +
Time(ns)	3.9% +	30% +	0%	0%	1.7% +	4.1% +

4.3.2 Scalability Study

As well known, ILP formulations do not scale well to substantial problem sizes. This is why it is essential to prune the search space using the Δ threshold value. Fig. 4.5 compare the running time of the two ARMORED versions (ARMORED:PARETO and ARMORED:Pareto+Δ). The naïve method is not included because it is extremely fast and thus, the running time is negligible. It can be observed that for 5%, the running time is still very low, as the design pool is still relatively small. On average, for ARMORED:Pareto it is $36s$ and for ARMORED:Pareto+Δ it increases to $62s$. This being a one time process, the overhead is negligible in regards to the increase in system reliability achieved.

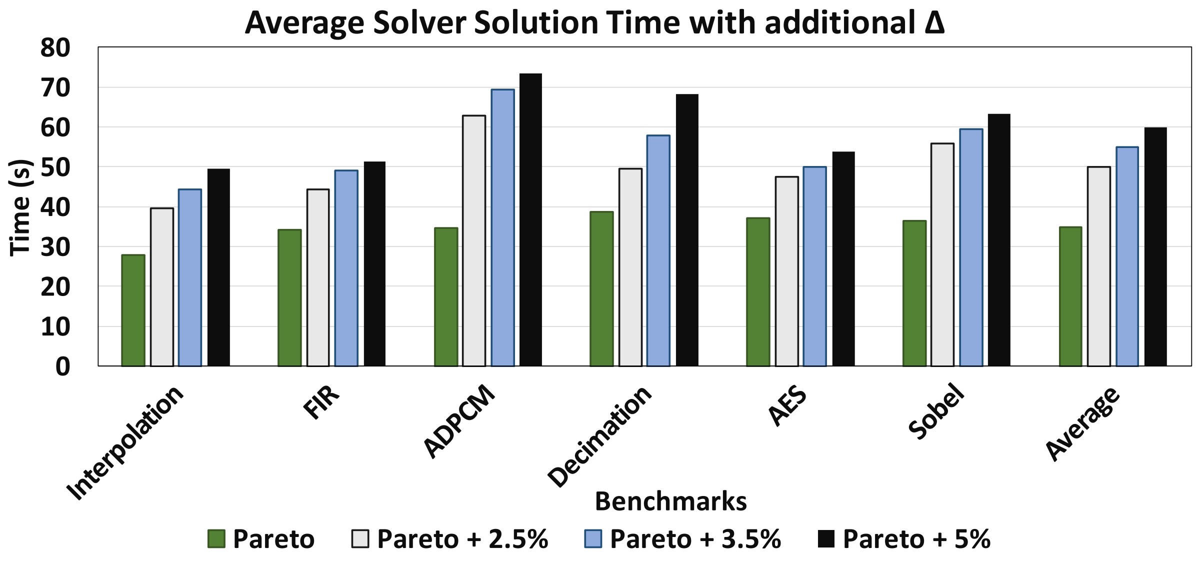


Figure 4.5: ILP Run-time comparison.

In summary, based on the results, it can be concluded that our proposed method works well and can find TMR accelerator systems with maximum reliability subject to different area and time constraints.

4.4 Summary and Conclusions

This work exploits one of the main benefits of C-based VLSI design over traditional RT-level design: The ability to generated micro-architectures with different characteristics from the same behavioral description, to automatically generate reliable TMR systems, by either performing space and/or time redundancy. In the RT-level, TMR systems are often over-designs in order to facilitate the verification of the system by merely instantiating the same design three times.

To avoid this, this work proposed a High-level synthesis design space explorer that estimate the reliability of each new micro-architecture and formulates the problem as an ILP to find the optimal system under a set of a given area and timing constraints. Results show the effectiveness of our proposed method, where a reliability improvement of at least 20% is achieved.

CHAPTER 5

COMMON-MODE FAILURE MITIGATION: INCREASING DIVERSITY THROUGH HIGH-LEVEL SYNTHESIS

Redundancy and diversity have been widely used throughout disciplines and mankind to tolerate design faults. E.g., airlines require a pilot and a co-pilot in each cabin and for complicated surgeries, two or more surgeons need to be present. In most of the safety-critical systems which require mechanisms to tolerate faults and the typical way to address these is through redundant systems. These systems are built by replicating the same hardware N times. This is typically called N-Modular Redundancy (NMR). Although effective, this approach cannot protect against Common Mode failures (CMFs). CMFs can result from failures that affect more than one module at the same time, generally due to a common cause in a redundant system. CMF can occur due to external (e.g., EMI, power-supply disturbances, and radiation), internal or design mistakes. Although redundancy can be adequate for physical fault detection and system recovery; but simple redundant systems cannot protect against CMFs because the replicated hardware modules will produce the same erroneous output and hence the voter will interpret the output as correct (10).

Design diversity has been proposed to address CMF detection. While for larger electronic systems these channels have been built by sourcing parts from different vendors. The most rudimentary form of design diversity is to have multiple parallel teams building the same circuit using different tools. Nevertheless, the reliability of these redundant systems does not only depend on the reliability of each version, but also on the differences (dissimilarities) between them. This technique adds a considerable cost overhead to the overall design proportional to the level of redundancy. Thus, automated methods to obtain different implementations of the same circuit which output different values in the presence of CMFs are required.

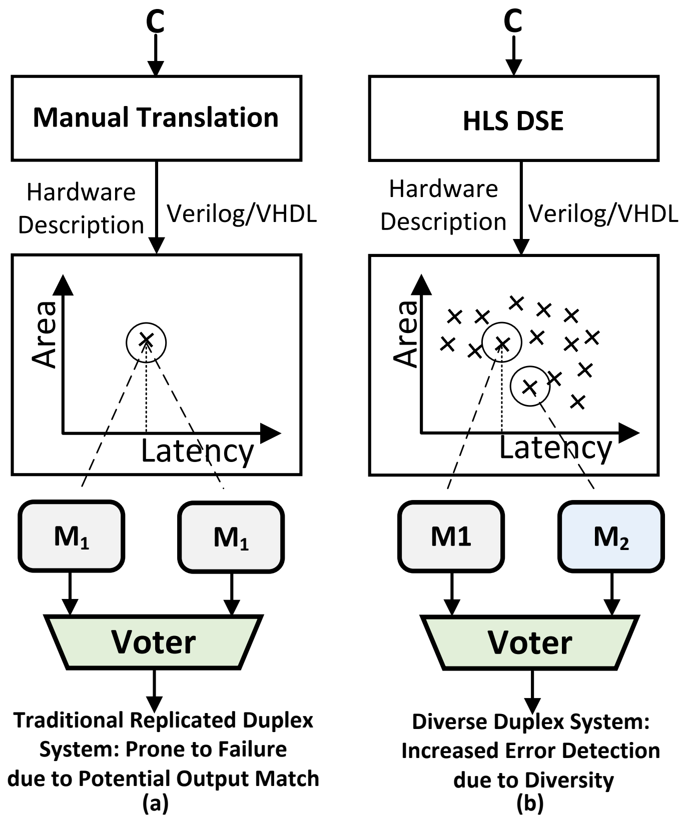

C
Manual Translation
Hardware Description
Verilog/VHDL
Area
Latency
M_1
M_1
Voter
Traditional Replicated Duplex System: Prone to Failure due to Potential Output Match
(a)
C
HLS DSE
Hardware Description
Verilog/VHDL
Area
Latency
M1
M_2
Voter
Diverse Duplex System: Increased Error Detection due to Diversity
(b)

Figure 5.1: Duplex System.

5.1 Diversity

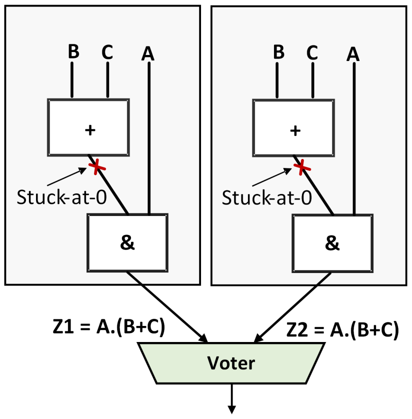
The need for high performance and energy efficiency has led to a trend shift towards heterogeneous System-on-Chip (SoC), which includes multiple dedicated hardware accelerators. At the same time, these heterogeneous SoCs have to be taped out at shorter and shorter time frames. This is forcing designers to raise the level of abstraction by using High-Level Synthesis, especially to design the dedicated hardware accelerators.

In a traditional VLSI design flow, the hardware designer has to analyze a given behavioral description and based on a set of constraints manually creates an RTL description that can execute the given functionality efficiently. To design redundant systems at the RT-level, designers tend to over-design the circuit by instantiating the same module multiple times. This approach is easier as only one module has to be fully verified. This implies that if there is a fault/bug in the design, it will be present in all of them, and hence it will never be detected, as the voter will always see the exact same output from the different modules. Fig. 5.1(a) shows this traditional flow. One additional problem with this approach is that it is extremely hard to re-target the system when moving to another platform/system, with different constraints (i.e., area, execution time, power) as this involves having to re-design and re-verify the entire system, which is time-consuming and error-prone. Besides, as mentioned previously, replicating identical modules does not protect against CMF. Thus, we raise the level of abstraction to create a diverse design pool to choose from to increase the error detection probability and preserve data integrity. Fig. 5.1(b) shows this alternative new approach to generate NMR systems that can also protect against CMFs automatically.

In C-based VLSI design, the first step typically requires to refine the original behavioral description to make it synthesizable. This synthesizable description can, in turn, be explored creating designs with a unique area, time and reliability. C-based VLSI design seems a very promising technology for automatic generation of reliable systems, including CMFs.

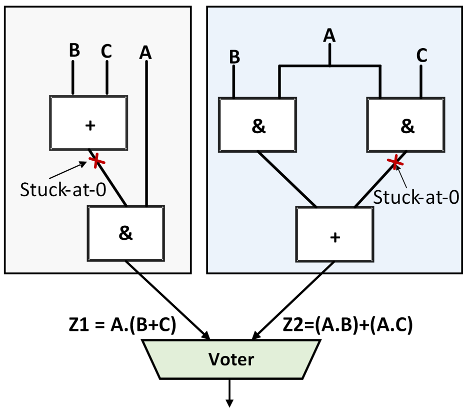
Common Mode Failure (CMF): CMF can be described as a fault that occurs at multiple modules at the same time. Based on the fault, these faults have the potential to propagate to the output manifesting as wrong, but identical outputs in two or more modules. Thus, the voter will not be able to detect this fault, leading to a potentially dangerous situation. The root cause of this kind of fault is the dependencies among the redundant units that lead to simultaneous failure (e.g., common environmental factors or buses within a system).

Fig. 5.2 shows an example. The two modules (M1) represent the same identical designs, where, $Z1 = Z2 = A \cdot (B + C)$. If a CMF, modeled as a stack-at-fault in the figure, leads to the output of the OR gate that produces $B + C$ to be a logic 0 when the inputs are either ABC=101,110,111, then Z1=Z2=0, whereas the correct output should have been Z1=Z2=1.



Inputs ABC	Correct Outputs	Faulty Outputs	
		Z1	Z2
000	0	0	0
001	0	0	0
010	0	0	0
011	0	0	0
100	0	0	0
101	1	0	0
110	1	0	0
111	1	0	0

Figure 5.2: Common-mode failure in Identical Design Instantiation.



Inputs ABC	Correct Outputs	Faulty Outputs	
		Z1	Z2
000	0	0	0
001	0	0	0
010	0	0	0
011	0	0	0
100	0	0	0
101	**1**	**0**	**0**
110	1	0	1
111	1	0	1

Figure 5.3: Common-mode failure in Different design Instantiation.

Because both modules produce the same result, the voter will assume that the wrong output is correct and continue normal operation.

Fig. 5.3 shows a diverse redundant system composed of two modules (M1 and M2) that are functional equivalent, but have different gate-netlists. In this case $Z1 = A \cdot (B + C)$ and $Z2 = A \cdot B + A \cdot C$. In this case, if a CMF happens, as shown in the figure, the only combination that will now lead to the same erroneous output is for the case of ABC=101. Thus, the risk against CMF has decreased significantly by increasing the diversity of the redundant system.

Based on this, intuitively, the larger the design pool from which to choose the micro-architectures for each module in the NMR system, the more robust a system against CMF will be. Nevertheless, sheer usage of dissimilar implementations does not always assure higher diversity compared to redundant systems with identical implementations, so it is essential to analyze and measure the diversity of redundant systems using a dependable metric.

5.2 Diversity Calculation Techniques

To the best of our best knowledge, there are two published existing methods to quantify diversity.

Dmetric

The authors of (71; 73; 74) proposed a diversity metric (D_{metric}) based on gate-level fault injection, given as:

$$DV = \frac{1}{m} \sum_{f_i, f_j} d_{i,j} \tag{5.1}$$

where, di, j is the probability of both designs not producing identical faulty output in response to a given input sequence with respect equally probable m fault pairs (f_i, f_j) in two designs implementing the same function. di, j's generate a diversity profile for the two implementations in a duplex system with respect to a fault model. The implementations can produce one of the following cases at their outputs In response to any input combination; (a) Both of them produce correct outputs, (b) One of them produces correct output and the other produces incorrect output, (c) Both of them produce the same incorrect value.

This motivates the concept of joint detectability, ki, j which is defined as the number of input patterns that produce the same erroneous output pattern in both implementations. For all the input patterns are equally likely, di, j can be specified as:

$$d_{i,j} = 1 - \frac{k_{i,j}}{2^n} \tag{5.2}$$

The main problem with this method is that it is an NP-complete problem, as it requires to simulate all possible fault-pairs in order to observe when the output of both netlists are identical and erroneous. The diversity value is between 0 to 1. This method nevertheless gives an accurate picture of how tolerant a design pair is against CMFs.

DIMP

To speed up the diversity computation process, the authors in (76) proposed a fast method called DIversity Metric based on circuit Path analysis (DIMP) based on the static analysis of the different gate netlists. The idea is based on the concept that lack of diversity happens when a given input signal I_i propagates to a given output signal O_j through similar gates in both circuit instances. So, to quantify diversity, DIMP takes into account whether the same netlists are traversed in the same order from I_i to O_j in both circuit instances. To achieve this, all paths of a given circuit are taken as inputs and circuit paths timing reports are obtained as output after logic synthesis listing all paths.

$$\begin{aligned} DIMP &= \sum weight_{(p^1_{i,j},p^2_{i,j})}.(1 - overlap_{(p^2_{i,j},p^2_{i,j})}) \\ MaxDIMP &= \sum weight_{(p^1_{i,j},p^2_{i,j})} \\ DV &= \frac{DIMP}{maxDIMP} \end{aligned} \tag{5.3}$$

In this case, the overlap is the number of gates repeated across paths in the same order, and weight is the maximum gate count of the paths. Two identical circuit instances will lead to a diversity of 0 since all the path are identical. The closer the diversity value is to 1, the more diverse the designs are and intuitively less susceptible to common-mode failure.

This method is much faster than the previous method, but still requires the logic synthesis and the path calculation for every micro-architecture, which for larger sequential circuits is also very time-consuming. Besides, the authors did not prove that two most diverse micro-architectures based on DIMP are at the same time the most efficient against CMFs, as the primary goal of asymmetric redundancy is to have circuits behave differently in the presence of a given fault (basically the outputs should be different).

5.3 Motivation

Fig. 5.4 shows a motivational example for this work. Fig. 5.4(a) shows a traditional fault-tolerant flow which does not take into account hardware diversity.

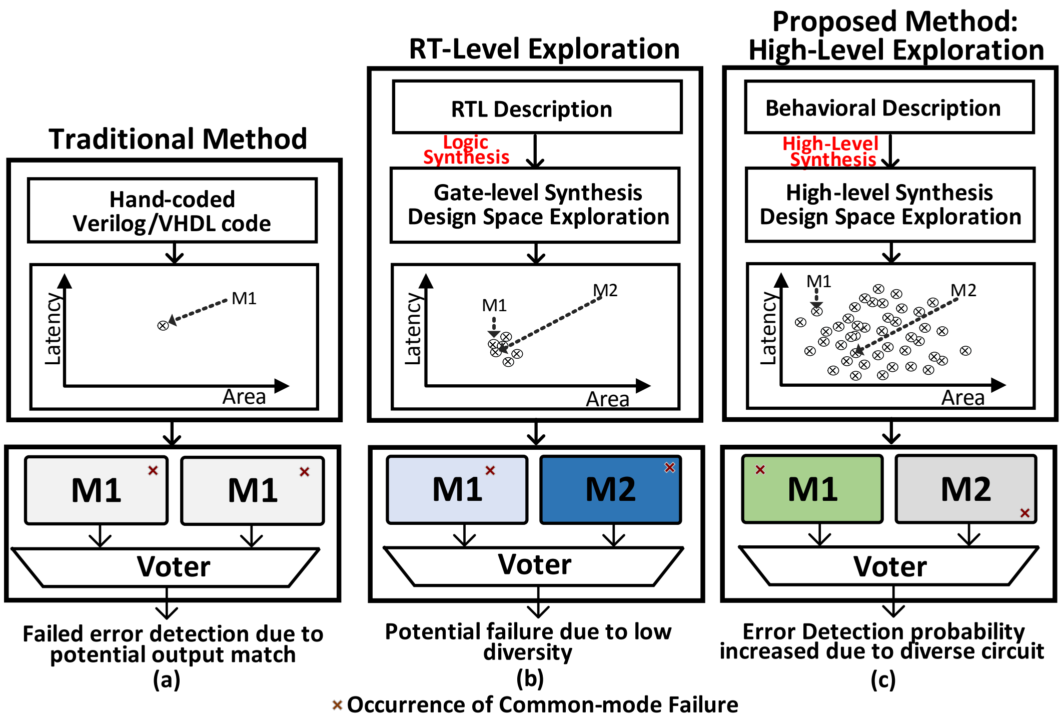

Traditional Method
Hand-coded Verilog/VHDL code
Latency
Area
M1
M1
M1
Voter
Failed error detection due to potential output match
(a)
RT-Level Exploration
RTL Description
Logic Synthesis
Gate-level Synthesis Design Space Exploration
Latency
Area
M1
M2
M1
M2
Voter
Potential failure due to low diversity
(b)
Proposed Method: High-Level Exploration
Behavioral Description
High-Level Synthesis
High-level Synthesis Design Space Exploration
Latency
Area
M1
M2
M1
M2
Voter
Error Detection probability increased due to diverse circuit
(c)
× Occurrence of Common-mode Failure

Figure 5.4: Design flow for Common-mode failure avoidance in Traditional and Proposed method.

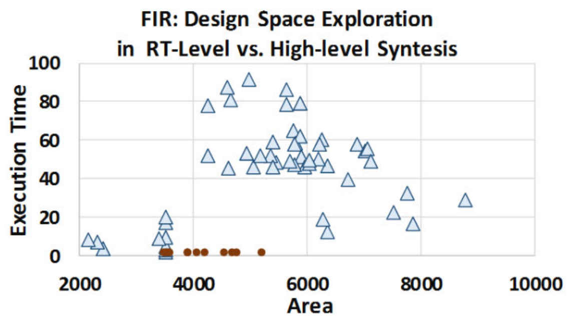
In this case, the exact same hardware channel is replicated twice and a voter checks if the outputs match or not, also called Duplication with compare (DWC). This configuration cannot detect CMF, which lead to the same erroneous outputs. To address this, previous work achieved diversity by synthesizing the RTL description with different constraints, such as maximum fanout, target synthesis frequency, etc (74; 76). As shown in Fig. 5.4(b), this leads to a family of micro-architectures with different performance and area, which in some cases, can detect CMFs.

Currently, VLSI design methodology is transitioning from the use of low-level Hardware Description Languages (HDLs) to higher levels of abstraction. This work advocates a new approach to fault-tolerance for CMF avoidance, based on leveraging one of the main advantages of C-based VLSI design: the ability to generate micro-architecture with different characteristics from the same behavioral description. Fig. 5.4(c) highlights the proposed flow. As seen, the design space is much larger than in Fig. 5.4(b) because completely different micro-architectures can be obtained from the behavioral level by setting different synthesis options as opposed to modifying the RTL synthesis constraints. The main synthesis directives used in HLS include the specification of how to synthesize arrays, loops and functions. E.g., arrays can be synthesized as RAMs (with a different number of ports) or registers, loops unrolled, partially unrolled or pipelined and functions inlined or not. Setting different combinations of these synthesis attributes leads to a completely new micro-architecture with unique area, power, performance and reliability tradeoffs. The main goal of this work is to investigate if the unique ability of behavioral VLSI design of diverse synthesis can be used to create diverse redundant hardware systems.

As we observe in Fig. 5.5, for both instances, design space exploration in High-level synthesis creates a larger design pool compared to the exploration result at the RT-level. In summary, the main contributions of this work are:

- Present an HLS design space explorer to maximize the diversity of hardware accelerators to protect against CMFs.

- Study the increase in diversity and hence CMF protection of traditional RT-level design diversity compared behavioral-level diversity.



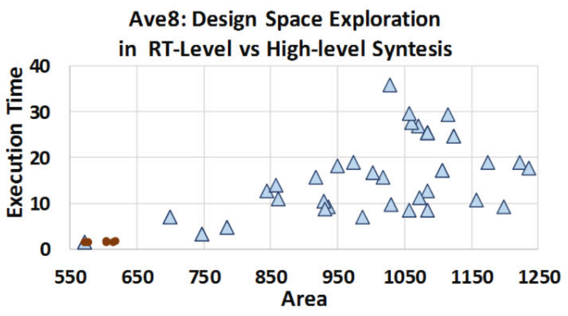


Figure 5.5: Design Space Exploration in Traditional and Proposed method.

5.4 Proposed Method

The proposed method is called High-Level Synthesis Design Space Exploration Method for Diverse Design Implementation (HDMI). Algorithm:3 summarizes the flow of our proposed method. The input to HDMI is a single behavioral description (C_{in}) and the output a set of gate netlist pairs with the highest diversity. The proposed method is based on a traditional HLS design space exploration (DSE) with modified cost function to maximize diversity. Thus, before we explain in detail how HDMI works, we define how to measure diversity.

5.4.1 Diversity Calculation

In this work, we utilize DIMP(76) method to quantify Diversity. In the next chapter, we will compare all different diversity estimation method and propose our noble technique.

Algorithm 3: Proposed Flow Overview.

```
input  : $C$: Input Behavioral Description,
         $M$: Exploration mode; $mr$: mutation rate
output: $DMR = \{D_1, D_2\}$
         $DMR$: Dual Modular Redundant System
1   /* Genetic Algorithm */
2   $population(D_1, D_2, ..., D_n) = gen\_population(C)$;
3   do
4       $p_i$,$p_j$= $gen\_parents(population)$;
5       $C_{pi,pj} = compute\_cost(p_i, p_j, M)$;
6       do
7           $o_{pi,pj} = gen\_offspring(p_i, p_j, mr)$;
8           $C_{pi,oij} = compute\_cost(p_i, o_{i,j}, M)$;
9           $C_{pj,oij} = compute\_cost(p_j, o_{i,j}, M)$;
10          $substitute\_parent(p_i, p_j, o_{ij}, C_{i,j}, C_{i,ij}, C_{j,ij})$;
11      while (N number of child doesn't improve result);
12  while (No smaller cost function obtained);
13  return (DMR with $C_{min}$);
```

5.4.2 Diversity-aware HLS Design Space Exploration:

The proposed HLS explorer is based on a Genetic Algorithm (GA), which has shown to lead to good results in multi-objective optimization problems (92). The explorer has three objectives. Two objectives need to be minimized, area and execution time and one maximized, the diversity. Because the final goal is to find design pairs to implement the DMR system shown in Fig. 5.4, a cost function that considers these three metrics for each new design pairs is needed.

The cost function nevertheless depends on the execution mode of the explorer, which can run in two modes: The first mode is the unconstraint case, in which the only objective is to find the design pairs with the highest diversity (highest DIMP value). In this case the cost function is defined as ($C_{di,dj} = 1/DV_{i,j}$), with $DV_{i,j}$ being the DIMP value between the two designs (D_i and D_j) being considered, with each design characterized by its area and latency, $D_i = \{A_i, L_i\}$ and $D_j = \{A_j, L_j\}$. In this case, a high diversity value is better as the objective is to maximize the diversity, which in turns reduces the cost function. This cost function is subject to the given total area (A_{max}) and execution time (L_{max}) constraint, and thus $A_{DMR} = \{A_i + A_j\} \leq A_{max}$ is the area of the complete DMR system composed of the two modules (A_i and $A_{j]}$)(the voter is excluded as it is the same in all of the solutions) and $L_{DMR} = max(L_i, L_j) \leq L_{max}$ the longest latency in clock cycles of the two designs, as these operate in parallel. In the unconstraint case, the explorer has to *sweep* the complete design space finding Pareto-optimal pairs of unique area vs. latency vs. diversity. Hence, the cost function used is defined as $C = \alpha A + \beta L - \gamma D$, where α, β and γ represents the weights representing the importance to minimize either the area, the latency or to maximize the diversity (thus, the negative sign in the diversity term). In this second unconstraint case, the explorer adaptively modifies these weights to sweep across the entire search space. The next subsections describe in detail how the explorer works.

HLS provides multiple *knobs* to generate different micro-architectures. They include global synthesis option that allows to e.g., control the FSM encoding and the target synthesis frequency, function unit constraints (limit of functional units to be instantiated) and local synthesis attributes in the form of pragmas or comments to control how to synthesize mainly arrays, loops and functions. These are basically comments inserted directly in the source code before the explorable operation. Our proposed explorer explores these last two knobs as they are also the most important knobs as they lead to completely different micro-architectures. Algorithm:1 illustrates an overview of the proposed GA-based diversity aware HLS DSE. The first step parses the behavioral description and extracts all the explorable operations in the code. These are operations that can be controlled through synthesis directives (in this work we consider arrays, loops and functions). It then continues by generating a pool of parents by randomly assigning each parent a unique list of pragmas. Each explorable operation is represented as a gene to which a synthesis attribute (pragma) is assigned. In this work, the pragmas we have used are: arrays={register, expand, logic, RAM, ROM}, loop={no, partial, all, fold}, and func={goto, inline}. With these pragmas, it is possible to synthesize arrays as memories or registers. Loops can be unrolled, partially unrolled or pipelined and functions inlined or synthesized as single hardware blocks (gotos). Two parents are in turn randomly selected (p_i and p_j) and their cost function computed depending on the type of exploration that is being run (unconstrained or constrained), thus leading to DMR system characterized by these three metrics $DMR_{pi,pj} = \{A_{DMR}, L_{DMR}, DV_{i,j}\}$. The list of attributes is then combined based on a randomly chosen cut-off point and some of the attributes mutated by choosing a different one from the attribute library. The mutation rate depends on the mutation rate mr specified by the user, which by default is set to $mr = 0.1$. The mutated offspring is then synthesized by calling the HLS tool and performing a logic synthesis on the generated RTL code (required to compute the DV) and the area and latency annotated. The diversity value is then computed using the DIMP value described previously, for the offspring and each of the parents leading to two new design pairs (DMR systems) $DMR_{pi,oij} = \{A_{DMR}, L_{DMR}, D_{i,ij}\}$ and $DMR_{pj,oij} = \{A_{DMR}, L_{DMR}, D_{j,ij}\}$. The offspring substitutes one of the parents if the cost of the new design pairs is lower than the cost of the two parents. Each new offspring is synthesized using the maximum number of functional units to fully parallelized the micro-architecture based on the pragma list and then again with a single functional unit of each time to maximize resource sharing. The algorithm will continue until N number of child designs do not improve any of the parents.

5.4.3 Predictive HLS DSE Method for Diverse Design Implementation (pHDMI)

One of the problems with the previously described GA-based diversity explorer is that in order to compute the DIMP value, it has to perform a logic synthesis for each newly generated offspring. We have observed that the HLS process is relatively fast (in the order of seconds), while the logic synthesis process is much slower (in the order of minutes). To speed up the exploration, we, therefore, investigated the use of predictive models that can be used to predict the diversity of a design pair with the information obtained right after the HLS process. For this, we used a well-known machine learning tool (WEKA (95)) which contains a library of predictive methods and run it on the results obtained from the GA-based explorer. The predictors were the area, segregated based on type (i.e., sequential vs. combinational, functional unit area, muxes area, decoder area, etc.), latency and delay values reported by the HLS tool and the predicted value the DIMP value calculated from the gate netlists. It was found that the differences in sequential logic $\Delta S_{i,j} = S_i - S_j$ (basically registers) could be used as a simple predictor for diversity. We call this method predictive HDMI (pHDMI) The experimental section compares the results for the GA-based exploration with gate netlist DIMP calculation and using $\Delta S_{i,j}$ as DIMP estimation.

5.4.4 RT-Level Design Space Exploration

Previous work, create diversity by generating functional equivalent gate netlists from a given RTL code. This can be achieved by setting the mapping effort to different levels, power effort, maximum fanout and target clock frequency. As shown in the motivational example, this leads to different netlists, but the actual micro-architecture does not change, thus, leading to a smaller search space and hence, diversity range. To compare our proposed method, we have implemented an RTL explorer which takes as inputs an RTL description in Verilog or VHDL and synthesizes it (logic synthesis) using the following *knobs*: mapping effort={medium, high}, area effort={medium, high}, and power effort={medium, high}, fanout={2,4,6}, clock = {n,n/2,n/4,2n}, where n is the original target synthesis frequency. Because an exhaustive enumeration of all the exploration knobs is not large, we perform a brute force search.

5.5 Experimental Results

We have chosen six different benchmarks with different complexities from the open source synthesizable SystemC benchmark suite (S2CBench)(93) as hardware accelerators to test

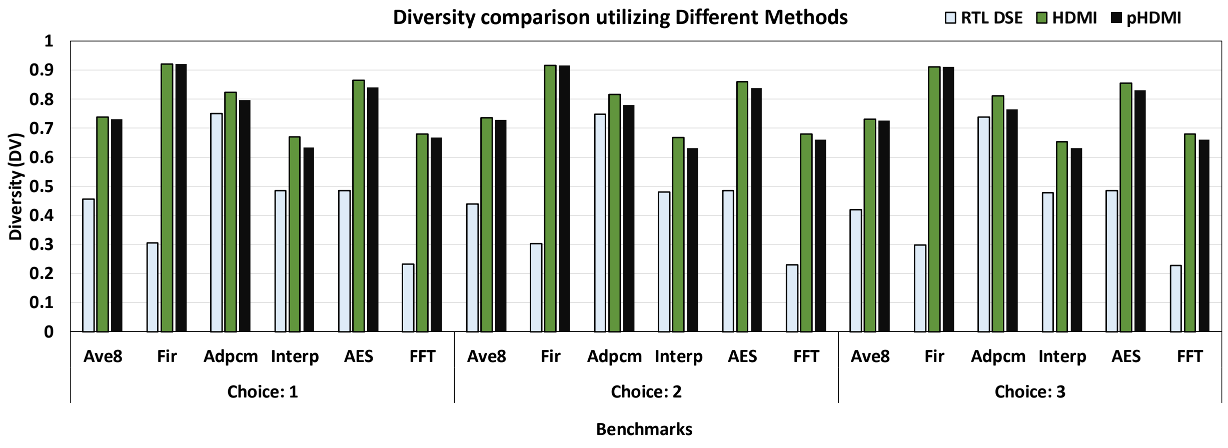


Figure 5.6: Top 3 design pair choices using High-level synthesis and RT-level Design Space Exploration methods.

the effectiveness of our proposed method. The tool used for the HLS exploration is CyberWorkBench v.6.1 from NEC (26) and Synopsys Design Compiler (DC)(96) as logic synthesis tool. The target technology is Nangate Opensource 45nm. The experiments are conducted on an Intel i7-6700 @3.50GHZ CPU and 16 GB memory, running CentOS 7.

The 6 benchmarks were explored in an unconstrained mode to fully observed the benefits of raising the level of abstraction. Fig. 5.7 shows the results of the three methods highlighting the three design pairs found by each of the methods with the highest diversity.

For the traditional RTL-based method, we synthesized the behavioral description using the default options from the HLS tools and fed the RTL code generated to the RTL-based explorer. HDMI is the full GA-based explorer performing a logic synthesis on each newly generated offspring and pHDMI is the predictive-based HDMI version that only uses the difference in sequential logic as diversity predictor and then fully computing the DIMP value for the selected design pairs in order to report the actual diversity values. For all the benchmarks, we observe that raising the level of abstraction to C leads to more diverse designs.

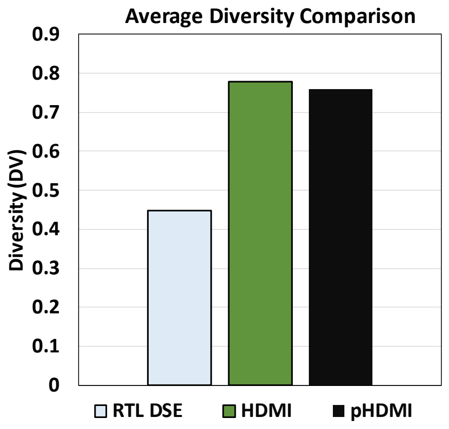


Figure 5.7: Average Diversity using High-level synthesis and Gate-level Netlist Design Space Exploration Methods.

The average diversity value increased by 1.75× and 1.70× for the HDMI and pHDMI method, respectively. This proves that using HLS leads to more robust designs against CMFs.

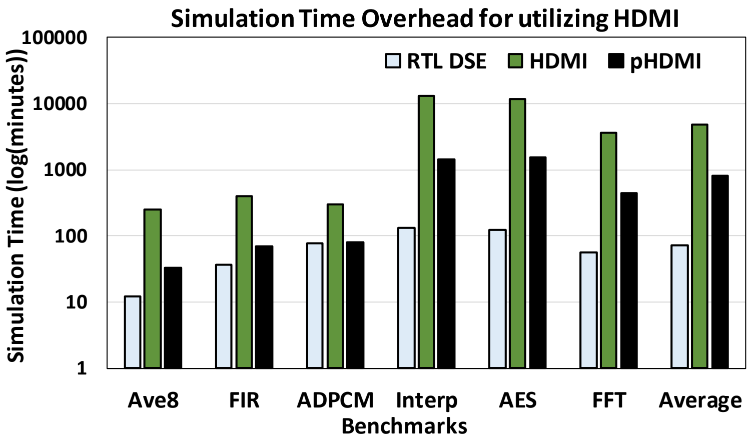


Figure 5.8: Comparison between simulation times for unconstrained and search space reduction methods.

Fig. 5.8 compares the three methods in terms of the runtime to obtain these results. The full HDMI method takes on average 6× longer than the predictive HDMI (pHDMI) and on average 66× longer than the RTL-based exploration. The main reason for the long running time for the HDMI method is that it requires a full logic synthesis for every new offspring. Although the runtime is high, we believe that it is still acceptable, especially considering the extra fault tolerance achieved and that this exploration process only requires to be executed once. Based on these results, we can conclude, raising the level of abstraction can significantly increase the design diversity and hence protect against CMFs.

5.6 Conclusion

In this work, we have shown that raising the level of abstraction from RT-level to C-level has the additional superiority of being able to increase the design diversity to detect common mode failures. Experimental results have shown that the diversity achieved at the behavioral level of abstraction is on average 2× higher, thus, proving the benefit of using HLS in fault tolerance designs.

CHAPTER 6
LEARNING-BASED DIVERSITY ESTIMATION: LEVERAGING THE POWER OF HIGH-LEVEL SYNTHESIS TO MITIGATE COMMON-MODE FAILURE

In an era of self-driving cars and space adventures, fault tolerance has become a first-order design metric. Thus, it is incredibly crucial to integrate fault tolerance seamlessly into the Very Large Scale Integrated (VLSI) design process. This is especially the case in state-of-the-art complex Systems-on-Chip (SoC), which typically contain a variety of dedicated hardware accelerators. At the same time, these SoCs have to be taped out at shorter and shorter time frames. This is forcing designers to embrace the use of High-Level Synthesis finally, also called C-based VLSI design, mainly to design the dedicated hardware accelerators, which are often the main differentiating parts of these SoCs (e.g., Apples A11 bionic SoC with its dedicated neural network accelerator).

Occurring faults are either transient or permanent. The first, cause no permanent damage to the circuit, while the latter result in permanent degradation or destruction of the circuit. Transient faults also called as single-event upsets (SEUs) are errors due to electrical noise or external radiation rather than design or manufacturing defects (5). These effects were first predicted in 1962 by Wallmark et al. (6) and in 1975 the first anomaly in a spacecraft system was encountered attributed to energetic heavy-ion passage (7). In addition, process scaling has led to increasingly critical challenges in manufacturing and lifetime reliability of integrated circuits (ICs). N-modular redundancy (NMR) techniques like duplication and compare (DWC) and Triple Modular Redundancy (TMR) are widely used for designing dependable systems with data integrity. Data integrity is maintained if a system either produces correct outputs or generates an error signal when incorrect outputs are produced. Most prior work on VLSI hardware reliability makes use of NMR, assuming that each module is the same. A voter is in turn used to determine if a fault has happened by comparing the results. Although effective for independent component failures, this approach cannot fully protect against Common Mode failures (CMFs). A CMF implies that multiple modules in the redundant system are affected at the same time. CMF can occur due to external events, e.g., Electromagnetic Interfere (EMI), power-supply disturbances, and radiation, internal or design mistakes (8). Traditional same hardware module redundancy systems cannot protect against CMFs because the replicated hardware modules will produce the same erroneous output and hence the voter will interpret the output as correct (9).

One of the proposed solutions to deal with CMF is to use asymmetric redundancy (10). In asymmetric redundant systems, multiple functional equivalent modules are instantiated in parallel with a voter. The main idea is to make faults occurring in both modules at the same time, visible at the outputs such that each module generates a different value. The worst case scenario in any fault tolerant system occurs if a system has a fault and the majority of the NMR modules produce the same erroneous output. Asymmetric redundancy aims at specifically avoiding this case.

The main problem with asymmetric redundancy is that it involves creating multiple different micro-architectures for each hardware channel. The most rudimentary form of design diversity is to have multiple parallel teams building the same circuit using different tools. This is obviously extremely expensive and inefficient. Thus, automated methods to obtain different implementations of the same circuit that output different values in the presence of CMFs are required.

In this work, we make use of one of the main advantages of C-based VLSI design, *a.k.a.* High Level Synthesis (HLS) to generate these asymmetric fault tolerant systems. In particular, the main advantage that we leverage is the ability to generate micro-architectures with unique area vs. performance trade-offs without having to modify the input description. This is typically done by setting different synthesis options, typically specified as comments or pragmas at the source code. This allows to control how to synthesize loops (e.g., unroll or fold), arrays (e.g., memories or registers with a different number of ports) and functions (inlined or not). This distinct advantage is leveraged to generate more robust NMR systems.

In (97), we already showed that HLS could be used to increase the diversity of hardware accelerators to protect against CMFs. In this work, we extend this work and propose a fast and efficient predictive learning method to estimate the *diversity* between different micro-architectures right after HLS such that no fault injection is required, thus, significantly speeding up the redundant system generation.

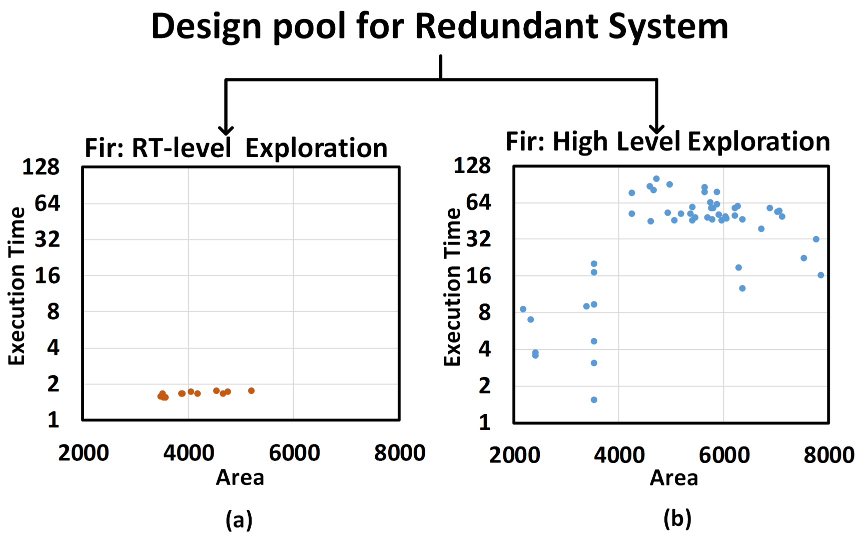


Figure 6.1: Design Pool generation at a different level of abstraction for FIR filter (a) RT-level Exploration (b) High-level synthesis Exploration.

Fig. 6.1 depicts the design pool generated at the Register-Transfer Level (RT-level) vs. High-level for the same FIR filter. In the first case, Fig. 6.1(a), the RTL Code (Verilog) is synthesized (logic synthesis) with different constraints (i.e., timing constraints). It can be observed that a much larger set of unique micro-architectures can be generated by raising the level of abstraction from RTL to the behavioral level as the pragmas allow generating completely different micro-architectures (Fig. 6.1(b)). Thus, it is tempting to investigate if raising the level of abstraction can help protect against CMFs. In summary, the main contributions of this work are:

- Investigate if a previously reported design diversity metric called DIversity Metric based on circuit Path analysis (DIMP) (76) based on the structural analysis of the gate netlist is effective against CMFs.
- Study if performing fault injections at the RT-level as compared to the gate-netlist can accelerate the diversity computation and achieve similar results.
- Present a new machine learning based method to estimate the diversity of design pairs right after HLS quickly.
- Make use of the new diversity metric to perform quick HLS design space exploration in order to find the design pairs that protect best against CMF.
- Study and compare different diversity estimation methods for CMF protection.

6.1 Proposed Method

The proposed flow is composed of two main phases. Phase 1 is the training phase that creates the predictive model to quickly estimate the diversity (DV) value between two micro-architectures. Phase 2 then, in turn, uses this predictive model to explore the search space of hardware accelerator specified as a behavioral description for HLS that needs to be made fault-tolerant against CMFs. The input of the proposed flow is a single behavioral description and the output is the design pair (two micro-architectures) with the highest diversity $DV_{i,j}$. Our proposed method, as shown in Fig. 6.2, can also output a trade-off curve with complete DWC systems with unique diversity vs. area and/or latency trade-offs, depending on the explorer's cost function (every point in the trade-off curve is a design pair). The next subsections describe these two phases in detail.

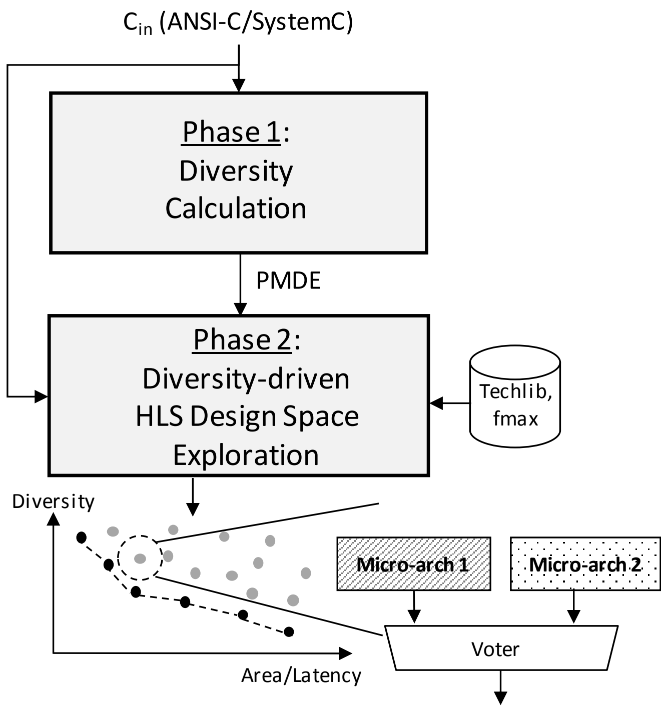


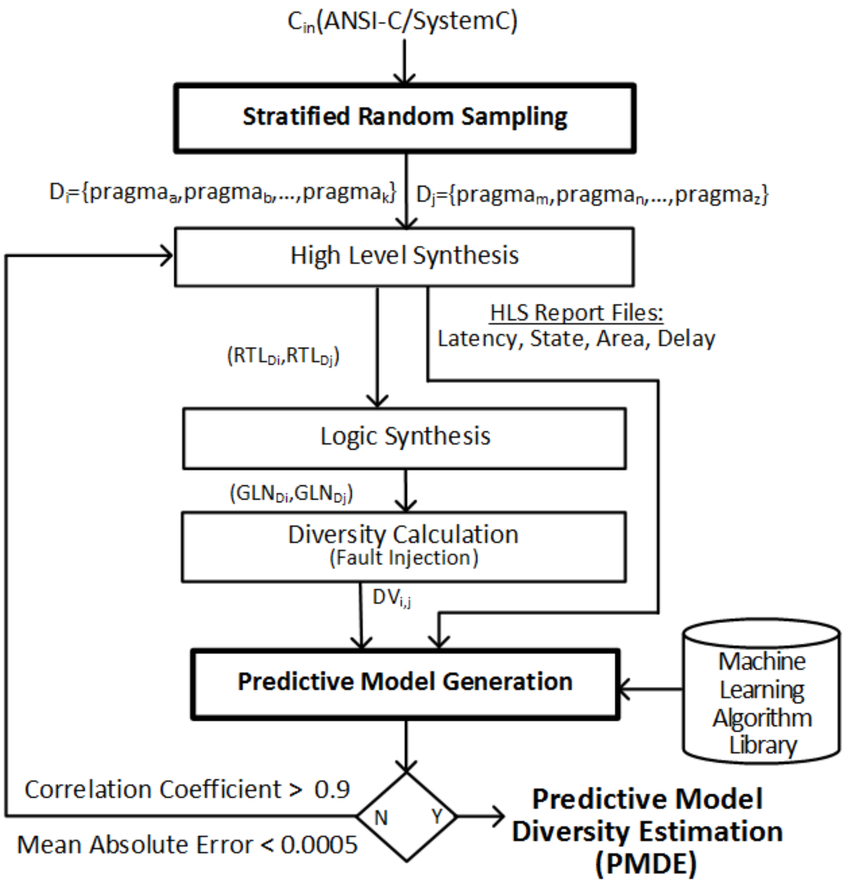
Figure 6.2: Complete proposed flow overview of the Proposed scheme composed of two phases. Phase 1: diversity calculation. Phase 2: Diversity-driven HLS design space exploration.

6.1.1 Phase 1: Diversity Calculation

To the best of our best knowledge, there are two published existing methods to quantify diversity, D_{metric} and $DIMP$.

The main problem with this method is that it is an NP-complete problem, as it requires to simulate all possible fault-pairs in order to observe when the output of both netlists are identical and erroneous. The diversity value is between 0 to 1. This method nevertheless gives an accurate picture of how tolerant a design pair is against CMFs.

This method is much faster than the previous method, but still requires the logic synthesis and the path calculation for every micro-architecture, which for larger sequential circuits is also very time-consuming. In addition, the authors did not prove that two most diverse



C_{in}(ANSI-C/SystemC)
Stratified Random Sampling
D_i={$pragma_a$,$pragma_b$,...,$pragma_k$}
D_j={$pragma_m$,$pragma_n$,...,$pragma_z$}
High Level Synthesis
HLS Report Files:
Latency, State, Area, Delay
(RTL_{Di},RTL_{Dj})
Logic Synthesis
(GLN_{Di},GLN_{Dj})
Diversity Calculation
(Fault Injection)
$DV_{i,j}$
Predictive Model Generation
Machine Learning Algorithm Library
Correlation Coefficient > 0.9
Mean Absolute Error < 0.0005
N
Y
Predictive Model Diversity Estimation (PMDE)

Figure 6.3: Training of Machine learning for Cost Function generation.

micro-architectures based on DIMP are at the same time the most efficient against CMFs, as the primary goal of asymmetric redundancy is to have circuits behave differently in the presence of a given fault (basically the outputs should be different).

To address the shortcomings of these two diversity metrics, we propose a new metric based on a predictive model that we call the Predictive Model Diversity Estimator (PMDE). Fig. 6.3 outlines the training phase. The original goal was to find a unique predictive model that could be used for any new behavioral description. Unfortunately, this did not lead to good results and hence, we had to base our approach on training each new design in order to generate a unique predictive model for that particular design. The predictive model generation is based on 4 steps:

Step 1: Stratified Random Sampling: This first step generated random design pairs $DPair_{i,j} = (D_i, D_j)$, by creating a unique set of synthesis directives (pragmas) for each of Design, such that $D_i = \{pragma_a, pragma_b, \ldots, pragma_k\}$ and design two $D_j = \{pragma_m, pragma_n, \ldots, pragma_z\}$, wherein this work we consider all the pragmas to control how to synthesize loops, arrays and functions.

Step 2: Synthesis: In this second step, the behavioral description annotated with the two different sets of pragmas is synthesized (HLS) leading to two unique functional equivalent micro-architectures specified in RTL (Verilog or VHDL), each with a unique area and performance. A logic synthesis is in turn performed on these two micro-architectures in order to obtain their gate-netlist.

Step 3: Diversity Estimation: This step computes the diversity of the $DPair_{i,j}$. As mentioned previously, there are two available metrics D_{metric} and $DIMP$. In this work, we use D_{metric} as this metric actually measures fault-tolerance and as we will show in the experimental results section using $DIMP$ leads to sub-optimal results. We follow the instructions of (10) and shown in equation (1) and inject fault pairs in the two gate-netlists to calculate the D_{metric}.

Step 4: Predictive Model Generation: This last step calls a predictive model generator that contains a library of well known predictive models (95). The predictors of the model are the HLS synthesis report outputs in terms of area of different logic resources (area of muxes, decoders and functional units), number of registers and the number of states of the micro-architecture and the predictive value is the D_{metric}. The output of this step is the predicted model diversity estimator ($PMDE$) and a confidence interval reported by the predictive modeling tool. These 4 steps are repeated until the model generator reports a confidence in the prediction of at least 95%.

Machine Learning Algorithm: We explore various machine learning algorithms i.e., ExtraTree, GuassianProcesses, Ibk, LeastMedSq, LinearRegression, LWL, M5P, M5Rules, Ran-

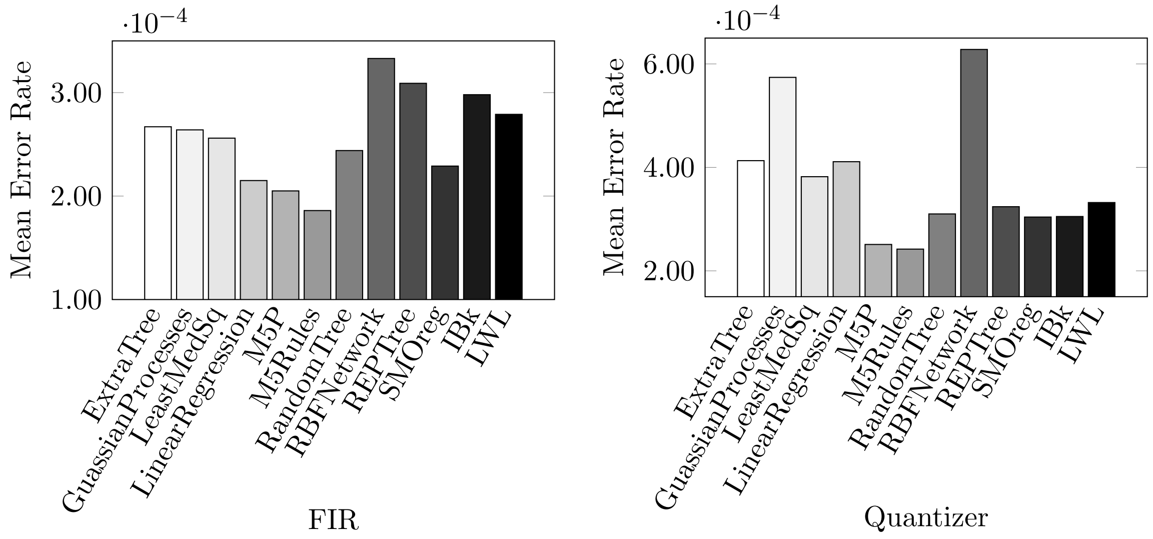


Figure 6.4: Mean Absolute Error (MAE) comparison among different algorithms. Lesser the value reflects more Accurate the result.

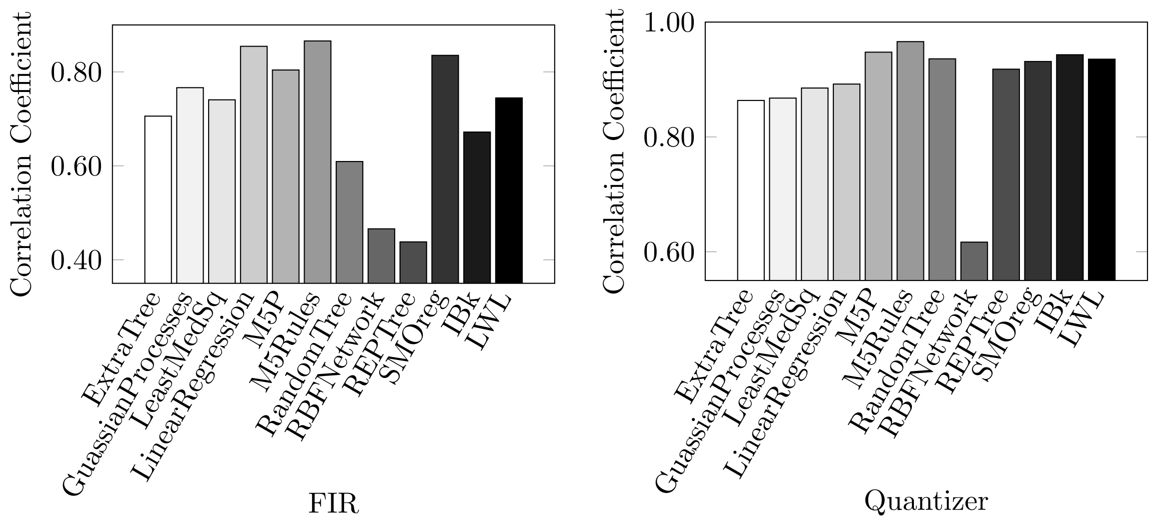


Figure 6.5: Correlation Coefficient (CC) comparison among different algorithms. Higher the value reflects more precision.

domTree, RBFNetwork, REPTree and SMOreg, which are suitable for our specific dataset. For this, we run the training phase for multiple benchmarks and compute the Mean Absolute Error (MAE) and the correlation coefficient.

MAE is the summation of the error values over the given number of samples. The lower the value of MAE is, the better the results are considered.

$$\text{Mean Absolute Error (MAE)} = \frac{1}{n}\sum_{i=1}^{n}|y_c^i - y_o^i|$$

where y_o^i and y_c^i are the golden output and the output being compared. 'n' represents sample size.

Nevertheless, only using MAE can sometimes be misleading as some model with lower accuracy might have higher predictive power for a specific problem domain. The correlation shows how close the prediction is to the actual value where a value 0 means the strongest possible disagreement and value 1 means total correlation.

In Fig. 6.4 & 6.5, we show the MAE and correlation of the different predictive models for two different benchmarks: FIR and Quantizer. From all the explored machine learning algorithms, *M5rules* consistently proved most efficient in respect to accuracy and precision. M5Rules is a supervised machine learning algorithm that generates a decision list for regression problems using separate-and-conquer wherein every iteration it builds a model tree using M5 and makes a rule out of the best leaf. This algorithm is basically a regression tree based method to predict a linear relationship between a predictor and the continuous response variable. Regression trees derive predictions from if-then-else conditions by portioning the data set into small groups and then fit a simple regression model for each subgroup. Basic tree models can become highly unstable and predict poorly. Nevertheless, by combining the power of regression and decision tree, the amalgamated model becomes effective and powerful.

Once the $PMDE$ is generated, the HLS design space explorer can start in order to find the two micro-architectures with highest $PMDE$.

6.1.2 Phase 2: Diversity Driven HLS Design Space Exploration

The predictive model generated in phase 1 is used in our modified HLS design space explorer that finds the micro-architecture pairs with the highest diversity. Fig. 6.6 gives an overview of

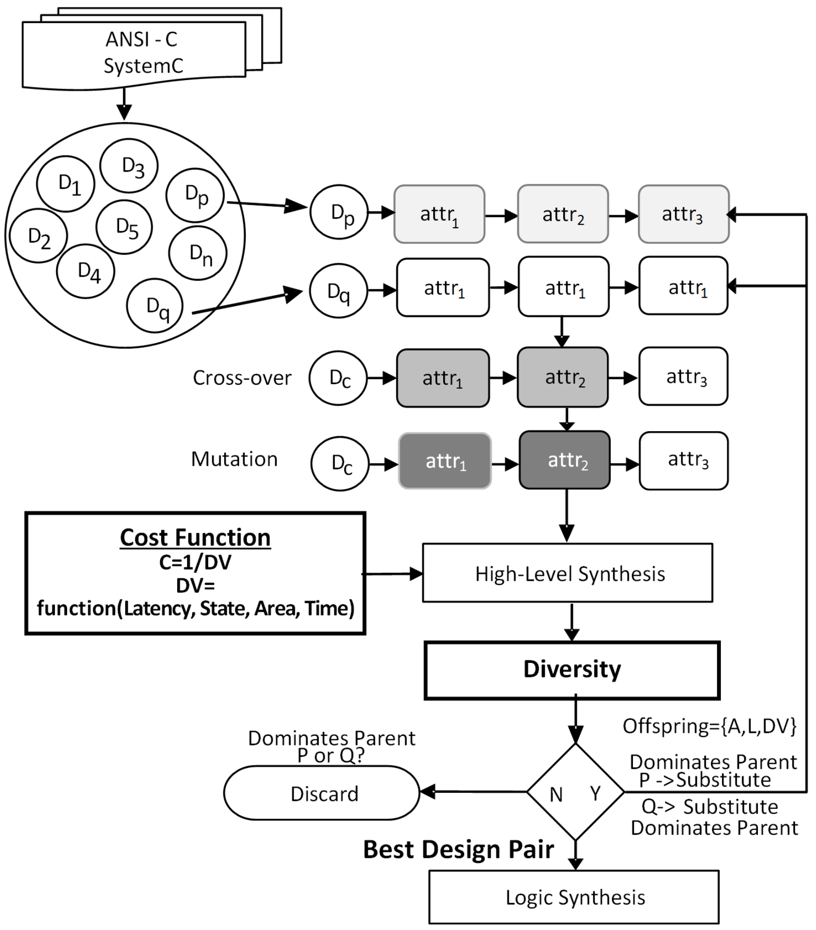


Figure 6.6: Overview of proposed Genetic Algorithm HLS Design Space Explorer for testing ML model.

the proposed diversity-aware Genetic algorithm-based HLS DSE. Although many heuristics for HLS DSE have been proposed (e.g.,(19; 22)) we choose Genetic Algorithm (GA), because it has proven to lead to effective results in multi-objective optimization problems (98). It should be noted that the main contribution of this work is not on the actual exploration method.

For the sake of simplification, we have set the cost function to maximize diversity. Thus, the cost function is set to ($C_{di,dj} = 1/DV_{i,j}$), (to minimize the cost between exploration runs), with $DV_{i,j}$ being the diversity value between the two designs (D_i and D_j).

A more realistic case would involve minimizing multiple objectives at the same time, like area, delay, while maximizing diversity. This would lead to a three-dimensional search space result from which the user can then choose the design pair that better meets the projects' needs. This phase can be further subdivided into 3 steps as follows:

Step 1: Explorable Operations Extraction: The first step of the exploration is to extract all the explorable operations in the code by parsing from the behavioral description. Synthesis directives (in this work, we consider arrays, loops and functions) can control these operations.

Step 2: Parents Generation: This second step generates a pool of parents by randomly assigning each parent a unique list of pragmas. A synthesis attribute (pragma) is assigned to each exploration operation, which is represented as a gene. The pragmas in this work are: arrays={register, expand, logic, RAM, ROM}, loop={no, partial, all, fold}, and func={goto, inline}.

These two steps can be nevertheless skipped as they are also required to generate the predictive model in phase 1. Thus, our method uses the designs generated during the sampling stage of phase 1 as the parent pool.

Step 3: Genetic Algorithm DSE: Two parents are in turn randomly selected (P_i and P_j). The list of attributes is then combined based on a randomly chosen cut-off point and some of the attributes mutated by choosing a different one from the attribute library. The mutation rate mr is specified by the user, which in our case is set to $mr = 0.1$. The mutated offspring ($o_{i,j}$) is then synthesized calling the HLS tool leading to micro-architecture with a unique area and latency $O_{i,j}$.

Next, the diversity value is evaluated using the predictive model diversity estimator ($PMDE$) generated in phase 1, for the offspring and each of the parents leading to two new design pairs. Thus, the following designs pairs are created: $DPair_{Pi,Oij}$ and $DPair_{Pj,Oij}$. The offspring substitutes one of the parents if the cost of the new design pair is lower than the cost of the two parents. Each new offspring is synthesized using the maximum number of functional units to fully parallelize the micro-architecture based on the pragma list and then again with a single functional unit each time to maximize resource sharing. The algorithm will continue until N number of offspring designs do not improve any of the parents. In this work, we set N=10.

6.2 Experiments

This next section presents the experimental results. It first introduced the experimental setup in order to allow the reproducibility of the results and the continues comparing the different methods.

6.2.1 Experimental Setup

We have chosen eight different benchmarks from different domains from the open source synthesizable benchmark suite (S2CBench) (93) as hardware accelerators to evaluate the effectiveness of our proposed method. In particular, Ave8, ADPCM, Cholesky, FIR, PWM, Qsort, Quantizer, and Sobel. Most of these benchmarks are small in order to allow us to find the optimal solution using an exhaustive enumeration of all the CMF cases, modeled as stack-at-faults.

CyberWorkBench v.6.1 from NEC (26) is used for HLS and Synopsys Design Compiler (DC)(96) used for logic synthesis. The target technology is Nangate Opensource 45nm. The experiments are conducted on an Intel i7-6700@3.50GHZ CPU and 16 GB memory, running CentOS 7. In order to evaluate *Dmetric*, comprehensive stuck-at-0 or stuck-at-1 faults were inserted in the gate-netlists of the 8 benchmark circuits using Modelsim (99). WEKA (95) is used to perform the machine learning phase.

Four different diversity calculation methods are compared to characterize our proposed method fully. These four methods are used to calculate the diversity of the design pairs generated by the proposed GA-based HLS DSE. Due to the randomness in the GA method, the GA-based explorer is executed five times and the best results reported.

$\mathbf{Dmetric_{gln}}$: The first method ($Dmetric_{gln}$) is based on the original Dmetric method proposed by Mitra et al.(10; 71; 73) computed at the gate netlist level. This method has proven to be the most accurate method to estimate fault-tolerance against CMF but is also the most time consuming one.

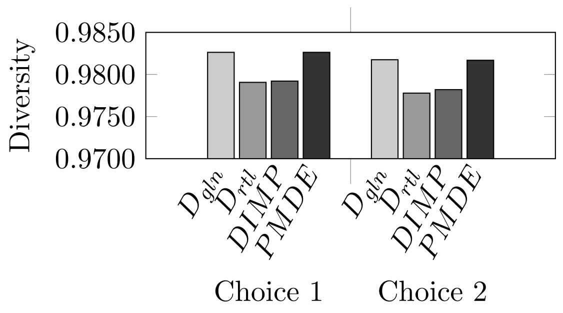
$\mathbf{Dmetric_{rtl}}$: The second method is based on the Dmetric, but applies at the RT-level ($Dmetric_{rtl}$). This reduces the search space as the RTL code has not expanded arithmetic operations like additions and multiplications.

DIMP: The third method is the DIMP method (76) based on path-based static analysis of the design pairs' gate-netlists.

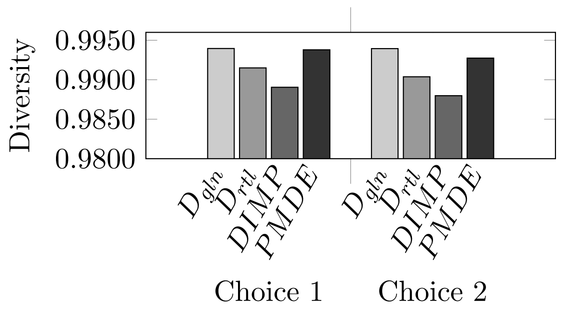
PMDE: The last method is our predictive model based method ($PMDE$).

These four metrics will be used to guide our proposed method in generating the most diverse system with highest CMF reliability, where using $Dmetric_{gln}$ is used as the baseline method as this more accurately measures the resilience against CMFs. Moreover, to better comprehend the impact that the diversity metric used has on the final system reliability, we generated 65 micro-architecture pairs for each benchmark and calculate diversity using all methods for each micro-architecture pair. We again use $Dmetric_{gln}$ as our reference model for ranking the design pairs with respect to diversity.

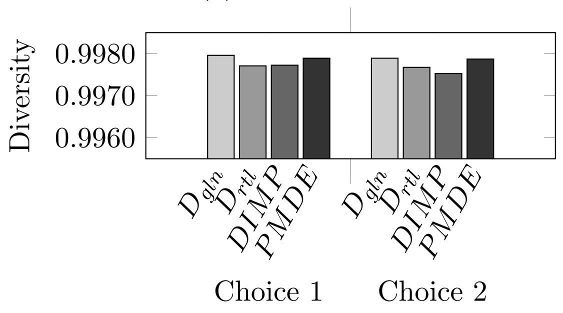
Thus, the experimental results try to answer the following questions? Can we accelerate the Dmetric calculation by doing it at the RT-level instead of at the gate-netlist level? Is DIMP a good metric to predict resilience against CMF? Can our predictive model diversity estimation accurately predict the Dmetric?



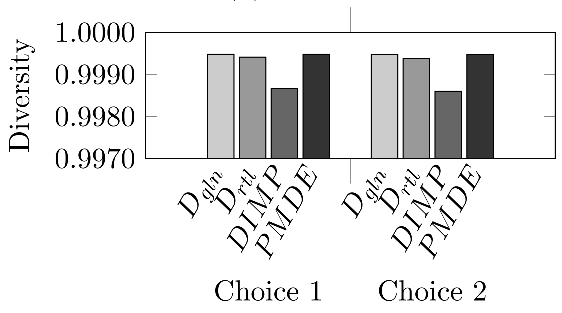
(a) ADPCM



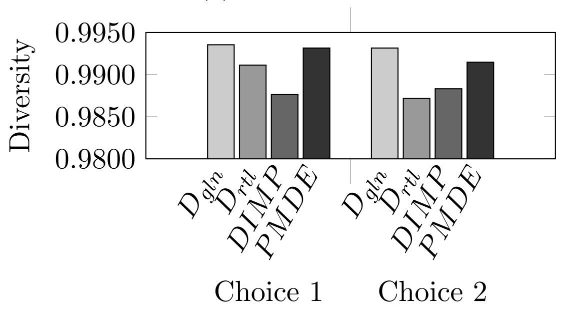
(b) ave8



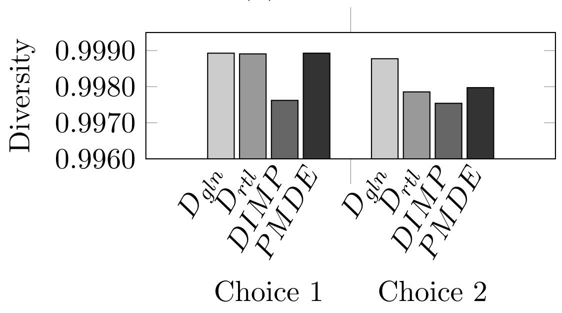
(c) Cholesky



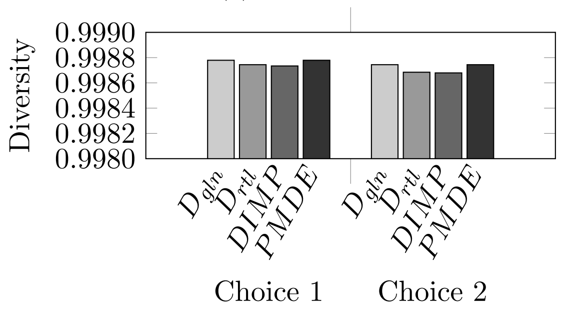
(d) fir



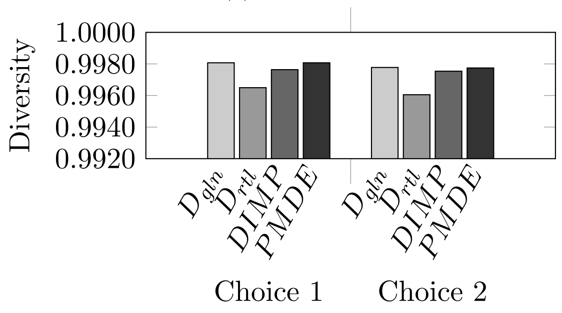
(e) PWM



(f) Qsort



(g) Quantizer



(h) Sobel

Figure 6.7: Diversity result comparison among all four Diversity Estimation Methods: $Dmetric_{gln}(D_{gln})$, $Dmetric_{rtl}(D_{rtl})$, $DIMP$ and $PMDE$.

6.3 Experimental Results

This section answers the questions posed previously.

Is* Dmetric$_{\mathbf{rtl}}$ *Efficient?

Intuitively there are fewer fault points in the RTL description generated after HLS compared to the gate-netlist. This is mainly because the RTL code contains '+' and '*' symbols for adders and multipliers etc, while the gate-netlist fully flattens these out. Here, the questions that we try to answer in this work is, if the Dmetric calculation can be speed up by doing the fault injection at the RT-Level.

Fig. 6.7 shows the experimental results when comparing the Dmetric calculated at the gate-level and RT-level. We can observe that for any 8 benchmarks the RT-level Dmetric does not produce the same result as the gate-level metric. Table 6.1 further highlights the difference showing design pair that $Dmetric_{rtl}$ chose as best ranked against the actual value using $Dmetric_{gln}$. The results imply for all the experiments; on average, $Dmetric_{rtl}$ chooses the 20th ranked design as it is the top diverse choice. One obvious advantage of doing the Dmetric calculation at the RT-Level is that it is $4\times$ faster as the search space is dramatically reduced.

Is* DIMP *Efficient?

The DIMP method was proposed as an alternative to the computationally intensive $Dmetric_{gln}$ method to calculate diversity. Although it is efficient to determine how different the designs are from each other, it has not also been proven to find design pairs resilient to CMFs. Fig. 6.7 also shows the comparison between $Dmetric_gln$ and DIMP results for all 8 benchmarks.

It can also be observed, from Table 6.1 that the DIMP method selects in 7 out of 8 cases a design-pair as the best choice which is not even in top 20 reference design choices. The advantage again is that it is very fast, achieving an average speedup across all test cases of $8.77\times$.

Table 6.1: Best design choice selection Ranks and Speed up for three Diversity Estimation methods compared to Baseline ($Dmetric_{gln}$).

Benchmark	Ranking			Time Saving		
	$Dmetric_{rtl}$	$DIMP$	$PMDE$	$Dmetric_{rtl}$	$DIMP$	$PMDE$
ADPCM	15	39	3	2.46	3.69	12.58
Ave8	51	41	1	2.27	16.89	12.24
Cholesky	7	23	2	5.83	5.94	11.37
FIR	6	58	1	1.95	2.06	7.52
PWM	7	26	2	1.95	2.06	7.26
Qsort	19	28	1	5.35	2.44	10.54
Quantizer	4	3	1	8.38	13.92	10.67
Sobel	51	41	1	4.37	23.15	11.76
Average	20	32.4	1.5	4.00	8.77	10.5

Effectiveness of PMDE

Finally, we study the effectiveness of our proposed machine learning based method. Fig. 6.7 and Table 6.1 summarize the results. From the results, we observe that our method ($PMDE$) can in most cases (6 out of 8) find the same design pairs as the $Dmetric_{gln}$. The average diversity is 0.99542 for $Dmetric_{gln}$ and 0.99534 for $PMDE$, which proves the efficiency of our method. Our prediction-based method, on the contrary, can find design pairs as the best choice from the top 3 baseline design choices for all benchmarks. In addition, our method is the fastest of all methods with an average speedup of 10× compared to $Dmetric_{gln}$, reducing the search time from hours to minutes.

In summary, we can conclude that $Dmetric_{rtl}$ and $DIMP$ lead to similar results, with $Dmetric_{rtl}$ leading to slightly better results on average. It should be noted that although in all cases the diversity values are over 0.99, in fault-tolerance, the main objective is to achieve 0% faults as a single fault could have catastrophic consequences.

6.4 Conclusion

In this work, we have extensively studied different diversity metrics and compared them qualitatively. We have also presented a learning-based diversity estimation method to facilitate the generation of redundant systems that mitigate the effect of common mode failures. This diversity estimation method has been used in a genetic algorithm-based HLS design space explorer to find design pairs with the highest diversity. Experimental results show that our method leads to better result compared to one state of the method and to almost the same results as the most accurate method while being much faster.

CHAPTER 7

A MACHINE LEARNING BASED HARD FAULT RECUPERATION MODEL FOR APPROXIMATE HARDWARE ACCELERATORS

The necessity for fault tolerance is increasing as a result of the persistent acceleration in the density and the total number of devices in VLSI. Imperfections caused by manufacturing defects are inevitable in VLSI chips containing Millions of submicron devices triggering a reduction in yield. Yield is the percentage of operational chips out of the total number fabricated.

Transient and permanent are two types of faults typically occurring in hardware. The first, cause no permanent damage to the circuit, while the latter result in permanent degradation or destruction of the circuit (87). Process scaling has led to increasingly critical challenges in manufacturing and lifetime reliability of integrated circuits (ICs). Future VLSI design is likely to lead to a compromise between lower reliability due to process manufacturing issues and cost, mainly due to fundamental limitations in device physics and the manufacturing process (100; 101).

In the case of permanent faults, the fault persists during the entire lifetime of the device. Thus, in order to continue benefiting from Moore's law, it is paramount to develop methods that can compensate for these permanent faults with minimal overheads.

The conventional approach is to test extensively for permanent faults. As ICs increase in complexity, this involves increasingly more significant testing time and thus cost. After testing, depending on the severity of the defects found, faulty circuits are typically either discarded (providing reliability at the cost of reducing yield), or in some cases, partially disabled circuits (providing reliability at the cost of performance) (102; 103).

However, as the rate of permanent faults increases, due to the increase in transistor count and increasing variability, there is an increasingly significant cost associated with yield and performance loss. Thus, it makes sense to tackle this issue at the design stage in order to build compensation circuits that can mitigate the effect of hard faults into the ICs.

If there is an active permanent fault in the circuit (which is not masked by the circuit propagation) the fault will cause the output to deviate from the golden expected output. In parallel, emerging applications implemented in hardware accelerators have been shown to have output solution spaces that contain a range of valid outputs rather than a single golden output (104).

Much work has already begun to explore energy/accuracy trade-offs in the context of approximate applications (105; 106). These applications are inherently resilient to inexactness or approximations and for that reason are tolerant to some loss of quality or optimality in their computed results (e.g., multimedia processing, DSP and machine learning) (107).

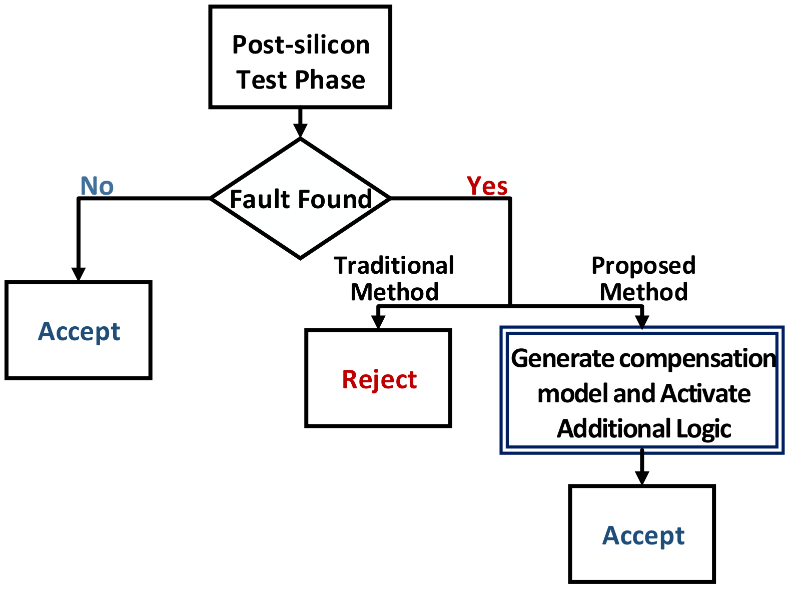


Figure 7.1: Fault Recuperation.

In modern complex heterogeneous SoCs, large portions of the computations are offloaded to dedicated hardware accelerators that can process these applications more efficiently by exploiting the inherent parallelism of the accelerators.

In this work, we propose, *SMURF: A S̲mart M̲odel for U̲niversal R̲ecuperation of Hard F̲aults in Approximate Computing Applications*, an error compensation approach that can compensate circuits of any complexities like the ones commonly found in dedicated hardware accelerators.

This work mainly targets hardware accelerators because these often tolerate inaccuracies in their computations and because they are typically the main differentiating components between different IC vendors. At the same time, the pressure to deliver these new or updated ICs at shorter time periods and the use of new technologies make these accelerators more prone to hardware errors compared to more mature and tested IPs.

As shown in, Fig. 7.1, the hardware accelerators will be manufactured with a SMURF module (Fig. 7.2) which can be activated if a permanent fault is found in the testing phase. The proposed SMURF module compensates the impact of a fault by predicting the relative difference of the correct and faulty outputs, rather than predicting the whole output.

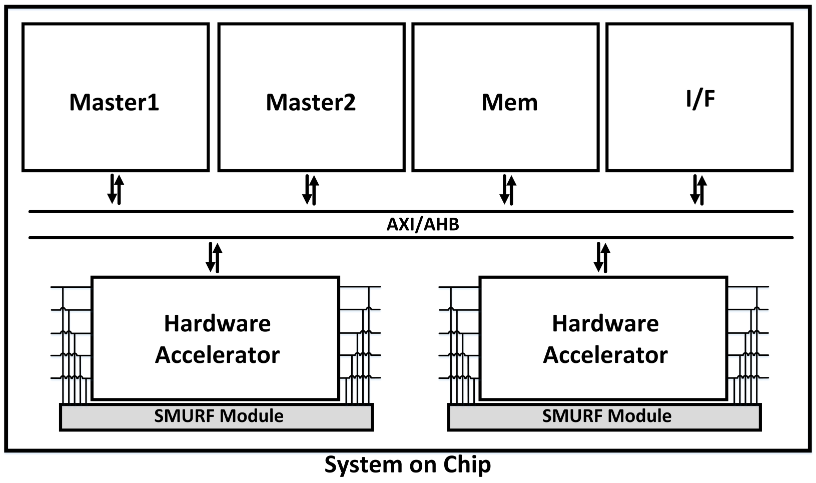


Figure 7.2: Abstract diagram proposed compensation logic integrated in Heterogeneous Multi-Processor System-on-Chip.

The compensation of the output is done by learning the difference distribution. This allows the method to scale much better with significantly less area overhead. This is because, we have observed that permanent faults commonly result in a small set of unique differences between the correct and faulty outputs, compared to a large number of unique outputs generated by the accelerator.

In summary, this work makes the following contributions:

- We propose a scalable approach for building error compensation modules by focusing on the difference in the result of the faulty and golden output for hardware accelerators that can tolerate inaccuracies in their outputs.

- We perform gate-level experiments figfor different accelerators of different complexities (purely combinational and sequential) and evaluate different supervised learning predictive methods for building the compensation logic.

7.1 Hard Fault Recuperation Model for Approximate Hardware Accelerators

In this work, we investigate an approach for building compensation logic for dedicated hardware accelerators amenable to approximate computing using supervised learning algorithms (i.e., inferring a compensation function based on the set of circuit behavior training data acquired during testing).

For the proposed method, we do not need to generate bit-wise results for each output, which provides the approach more flexibility, as opposed to using statistical or Bayesian methods. We believe our method is orthogonal to much of the previous work.

Most prior works focusing on error compensation is highly application dependent. Our proposed approach is application independent and scales very well with the number of faulty outputs. Thus, it is particularly well suited in the context of permanent fault models, as the activation of the fault is solely dependent on the inputs, and not the previous circuit state or environmental conditions (83).

Our proposed approach leverages the wrong outputs as an asset for fault compensation, as the number of erroneous outputs made visible at the accelerator's outputs is relatively low compared to the total number of input test-vectors.

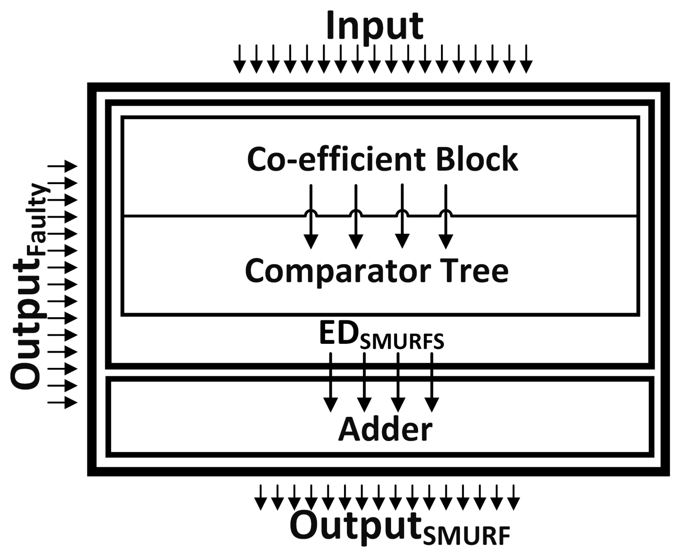


Figure 7.3: SMURF Module for Hard Fault Recuperation.

The proposed compensation logic is composed of a tunable module making it flexible enough to compensate for different permanent faults. Although the compensation logic does not fully generate the correct output, it mitigates the overall error. Thus, it is specifically well suited for hardware accelerators that can tolerate inaccuracies in their outputs (approximate computing). The next subsections describe in detailed the proposed compensation logic and how it is tuned to compensate for errors.

7.1.1 Architecture

In order to perform the correction of a permanent fault, typically, the location of the intermediate node that contains the fault needs to be located. The main problem is that the presence and the location of permanent faults can only be determined post-silicon. This implies that any compensation logic would require as many additional signals as any potentially faulty nodes in the original circuit. This is nevertheless not practical.

For approximate computing applications, like the ones targeted in this work, the compensation logic only needs connections from the global inputs and global outputs, as shown in Fig. 7.2, thus, reducing and simplifying the overall circuit design. Therefore the IOs to the SMURF module is only the global input and output signals of the accelerator.

The proposed fault compensation architecture is instantiated along all of the hardware accelerators that are amiable to approximate computing. Fig. 7.3 shows an overview of the SMURF module, which is divided into two main parts: A programmable input tree module and an output adder module.

The programmable tree itself is composed of two main modules: a coefficient block and a comparator tree. The coefficient block contains unique vectors that result in wrong outputs. We call these vectors, Error Distance values ED, and are described in detail in the next subsection.

The programmable tree module is fixed and needs to be tuned by choosing a set of coefficients to compensate for different faults in the accelerator of each unique chip. The size of this tree is independent of the complexity of the accelerator and depends only on the number of inputs. Thus, the area grows $log(inputs)$, which is an additional advantage of the proposed compensation logic.

The last module is a n-bit adder where n is the number of outputs of the accelerator. The adder compensates the faulty outputs by adding a compensation vector (ED_{SMURF}) to the faulty output value $Output_{Faulty}$ as shown:

$$\text{Output}_{SMURF} = \text{Output}_{Faulty} + \text{ED}_{SMURF}$$

This reduces the output error (E_{SMURF}) compared to the faulty error without compensation (E_{Faulty}), such that:

$$E_{SMURF} < E_{Faulty}$$

This output stage also grows linearly with the number of outputs and hence further contributes to the scalability of the proposed compensation logic.

Given the trend toward heterogeneous System-on-Chip (SoC) architectures containing numerous hardware accelerators, another important usage scenario for the compensation module would be as a shared component among different accelerators. The area costs could then be amortized by sharing the compensation module among these accelerators. The next subsection describes in detail how this tuning is done using learning-based methods, also summarized in Algorithm 1.

7.1.2 SMURF: Methodology

The SMURF method is divided into four main steps. The first step finds the test-vectors that lead to observable erroneous outputs, while the second step tunes the compensation by generating the error distance. The third step generates the model by leveraging supervised machine learning while the last step activates the recuperation model.

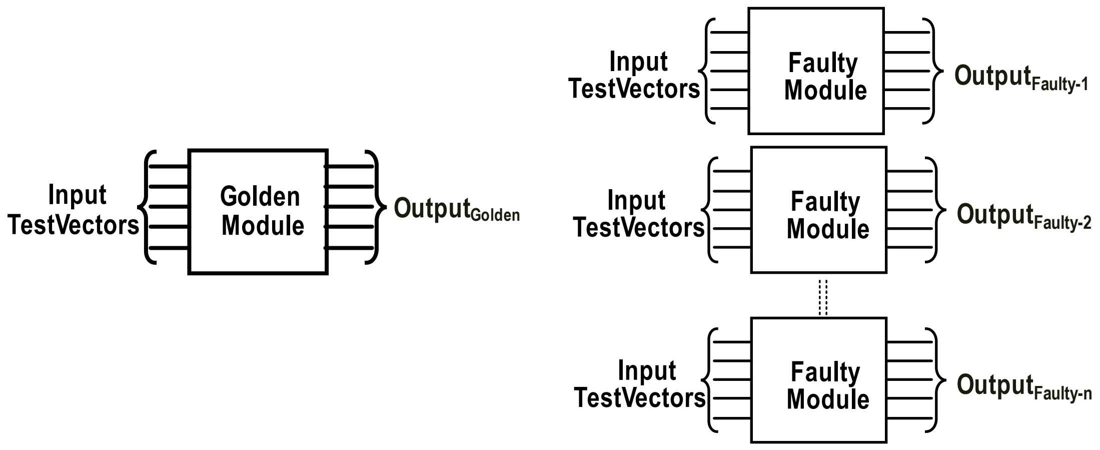


Figure 7.4: SMURF: Data Preparation.

Step 1: Data Preparation: The very first step in our proposed flow is to find which input test-vectors (TVs) lead to observable errors in the outputs. Thus, all of the ICs are analyzed using the same input patterns and their outputs compared against the golden, fault-free, outputs. Faulty circuits are generated by injecting single stuck at fault with gate-level simulation. Input test vectors (TV), $Output_{Golden}$ and $Output_{Faulty}$ are collected.

Step 2: Compensation Tuning -Error Distance (ED) Vectors:

In order to tune the compensation logic, a measurement unit called Error Distance (ED) vectors is introduced, were ED can be defined as the difference between the golden output and the faulty output.

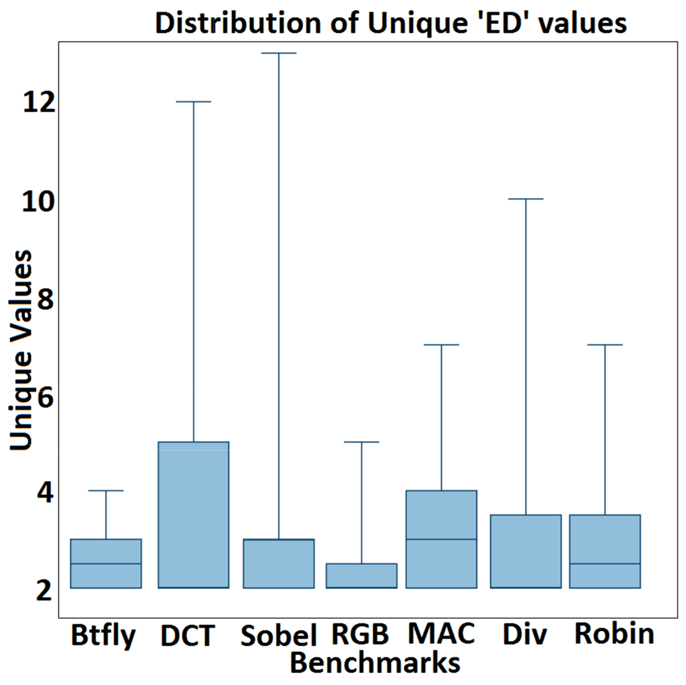


Figure 7.5: Number of Unique ED value distribution in various benchmarks.

$$\text{ED} = |\text{Output}_{Golden} - \text{Output}_{Faulty}|$$

This phase stores all the TVs and outputs. A database of all these TV is stored in the comparator tree of the SMURF module to detect when n given input will generate wrong output.

Although it might seem that the number of TVs leading to an observable erroneous output might be large, in the case of permanent faults, the unique number of ED, is typically very low.

Fig. 7.5 shows the unique number of *ED* for 7 accelerators for 100,000 input test vectors when a single random permanent fault is inserted in them (modeled as a stuck-at 0 or 1). It can be observed that the number of *EDs* is very low, typically less than 5.

Algorithm 4: Proposed flow overview.

```
     Input  : $\{TVs, Output_{Golden}, Output_{Fault}\}$
1       TVs : Input test vectors
2       $Ouput_{Golden}$: Error free outputs
3       $Output_{Faulty}$: Faulty outputs when IC has permanent fault
     Output: $\{Output_{SMURF}\}$
4       $Output_{SMURF}$: Corrected approximated outputs
5    /* Compensation Tuning */
6    $ED = ErrorDistCalc(TVs, Output_{Golden}, Output_{Faulty})$
7    /* Supervised Learning */
8    $ED_{SMURF} = HoeffdingTree(Input, Output_{Golden}, Output_{Faulty})$
9    /* SMURF Module Activation: Load Co-efficient */
10   $Output_{SMURF} = ActiveSMURF(Output_{Faulty}, ED_{SMURF})$
11   return($Output_{SMURF}$);
```

Step 3: Supervised learning- Compensation Coefficient: Our method then continues by analyzing the input vectors and faulty outputs pairs in order to tune the proposed compensation logic, and in particular the input coefficient block. A supervised machine learning method is used for this tuning.

Supervised learning allows for a fast, flexible, and scalable way to generate accurate compensation routines that are specific to the particular set of application inputs and permanent faults. It allows automated data analysis with a set of methods which not only detect patterns in the given data set but also predict future data. In supervised learning, a map between a set of input attributes and an output variable is used to predict the unseen data or in other words to build a model of the distribution of class labels according to predictor features(33; 108).

Both linear and non-linear classification/regression algorithms can be used for the compensation block. Different algorithms typically have different trade-offs (e.g., accuracy vs.

size). In this case, Hoeffding Tree was found to provide the best accuracy results as the compensation of fault is highly non-linear.

A Hoeffding tree is an incremental, anytime decision tree induction algorithm that is capable of learning from massive data streams, assuming that the distribution generating examples does not change over time, like in our case. It is a fast decision tree classifier, that grows a decision tree incrementally based on the theoretical guarantees of the Hoeffding bound (or additive Chernoff bound). A node is expanded as soon as there is sufficient statistical evidence that an optimal splitting feature exists, a decision based on the distribution-independent Hoeffding bound (109). This leads to a compact and precise tree model.

To train the tree, the inputs (TVs), golden outputs (Output_{Golden}), tested outputs (Output_{Faulty}) and ED values collected in the previous phase are used.

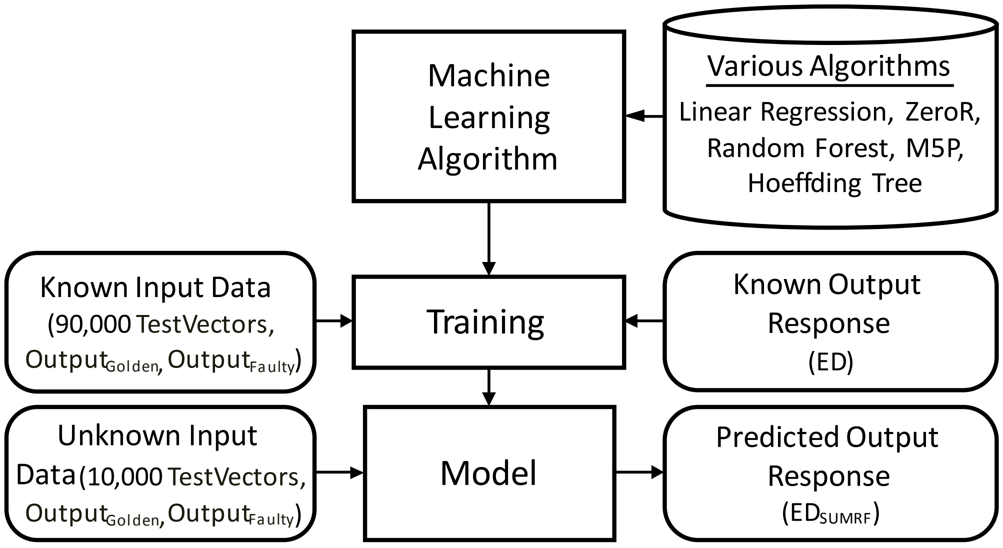

Machine Learning Algorithm
Various Algorithms
Linear Regression, ZeroR, Random Forest, M5P, Hoeffding Tree
Known Input Data (90,000 TestVectors, Output_Golden, Output_Faulty)
Training
Known Output Response (ED)
Unknown Input Data (10,000 TestVectors, Output_Golden, Output_Faulty)
Model
Predicted Output Response (ED_SUMRF)

Figure 7.6: SMURF:Supervised Learning.

Feature selection is an important measure to get accurate results during the prediction. Thus, the supervised learning method selects the attributes from the training data provided to build an accurate and precise model.

The target variables that need to be predicted have a tremendous impact on the performance of a machine learning scheme. The accuracy of the scheme improves if the unique number of classes that need to be predicted decrease, as the probability of a prediction sample falling into a wrong target variable boundary decreases as the number of unique target variable decreases.

For typical DSP, image processing and mining accelerators, n unique input pattern can reciprocate to n different output pattern. Thus, predicting a large number of unique outputs will lead to an extremely large circuit, making this impractical.

Therefore, the proposed method predicts the ED values as the target values. Again from Fig. 7.5, it can be observed that the number of predicted values is much smaller than if the total number of outputs would need to be predicted. This significantly reduces the size of the compensation logic. Fig. 7.6 overview the steps for the supervised learning process to generate the compensation coefficient to be added to the faulty output.

Step 4: SMURF Module Activation: When a module is found faulty, the SMURF module will be activated with the correct model for the particular fault. The outputs of this training phase are the ED_{SMURF} vectors that need to be added to the fault outputs generated by the module when an input combination that triggers the permanent fault is passed to the module.

7.2 Experimental Setup

We have chosen multiple accelerators for evaluating our proposed methodology. In particular: Butterfly, DCT, Sobel, RGB, MAC, Division, and Round-Robin Arbiter taken from various open source benchmark suite (110; 111). These have been synthesized using Yosys (112) in order to obtain their gate-netlists.

In order to simulate permanent faults, random stuck-at-0 or stuck-at-1 faults were inserted in the gate-netlists of the 7 benchmark circuits. These were simulated using Modelsim

(99) before and after the fault was inserted using 100,000 random inputs. WEKA (95), is used to perform the machine learning phase. For the benchmarks described above, more than 100,000 random inputs did not improve the quality of the machine learning accuracy any further.

A more extensive test set may be needed for more massive accelerators to generate accurate recuperation models. With large accelerators, Automatic Test Pattern Generation (ATPG) and fault collapsing method (113), can also be utilized for further improvement of model accuracy and reduction of the number of faults.

7.3 Experimental Results

This section presents the experimental results obtained for the 7 benchmarks circuits in terms of accuracy and precision. Finally, we compare the quality of results (QoR) when using the Hoeffding tree, vs. other predictive methods, e.g., linear regression and RepTree.

7.3.1 Accuracy

The accuracy of the proposed method is shown in Fig. 7.7, when using our SMURF compensation logic compared to not using any. Fig. 7.7 shows the percentage of time when the module is producing the correct results (same as the golden outputs). The closer the value is to 100%, the better the model is considered.

The accuracy of the faulty modules is on average 57%, while it increases to 96% on average. The exact output generation is significant, as this indicates that our proposed method might also be useful for non-approximate computing applications.

Here we also observe, that the accuracy of the faulty module has a broader distribution ranging from 30% to 90%, while the accuracy after using SMURF, reduces the overall distribution significantly.

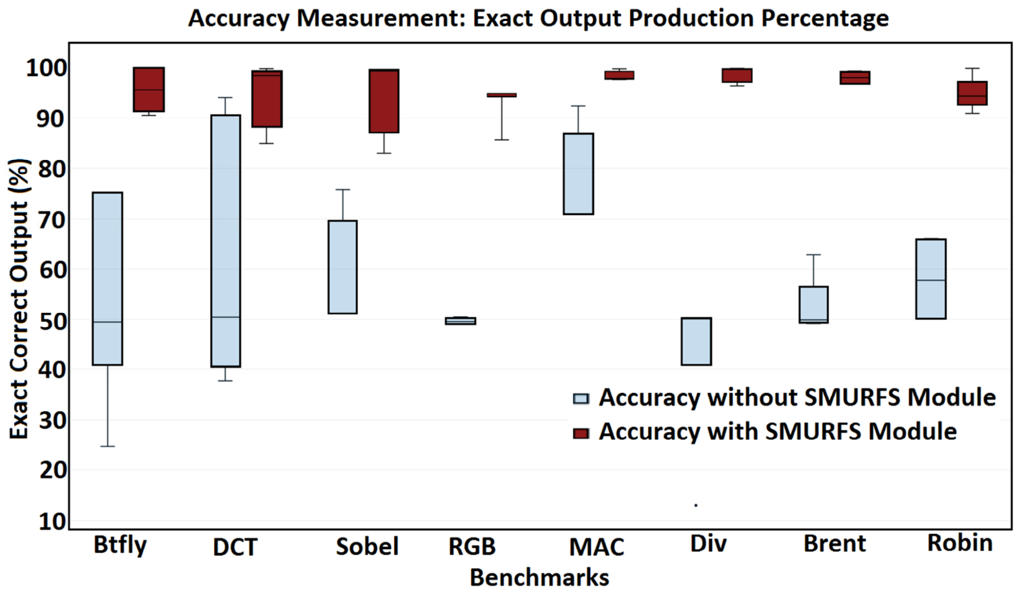


Figure 7.7: Accuracy Measurement: Distribution of exact output generation before and after the use of SMURF.

Next, the overall error in the benchmarks is analyzed using Mean Absolute Error. Mean Absolute error is the summation of the error values over the number of samples.

$$\text{Mean Absolute Error (MAE)} = \frac{1}{n}\sum_{i=1}^{n} |y_c^i - y_o^i|$$

where y_o^i and y_c^i are the golden output and the output being compared. 'n' represents sample size.

To normalize the value of MAE, we have calculated the percentage decrease in MAE in other words, improvement in accuracy. Here, larger values mean more decrease in error, thus, an increment in accuracy.

$$\text{Decrease in MAE} =$$

$$(\text{MAE}_{Faulty} \text{ - } \text{MAE}_{SMURF}) \ / \ \text{MAE}_{Faulty} \ \ \text{X } 100\%$$

Fig. 7.8 shows the decrease in MAE after using SMURF model in the faulty modules. For benchmarks, Division, MAC, RGB and Sobel, we observe above 95% improvement in

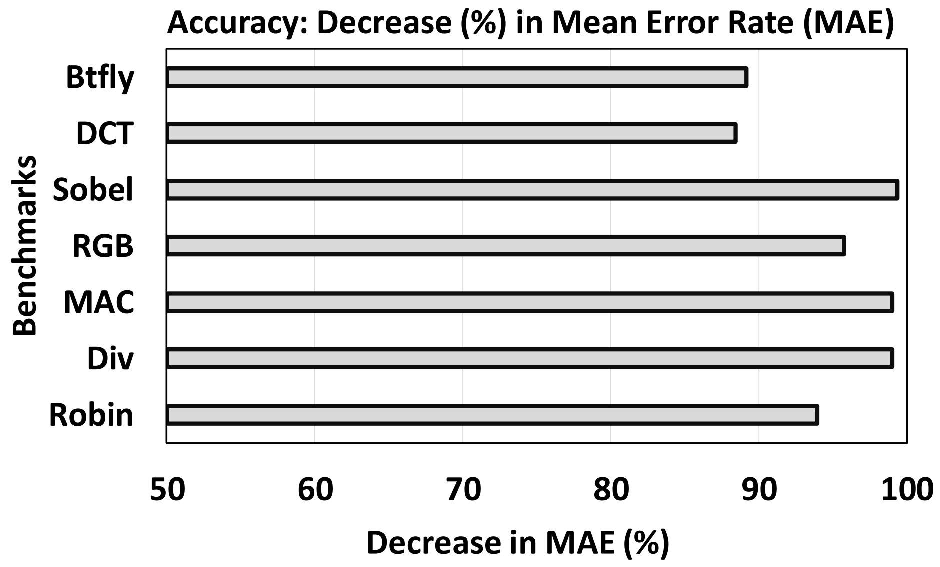


Figure 7.8: Improvement of Mean Absolute Error after using SMURF model.

accuracy. We also observe around 90% improvement in accuracy for Butterfly, DCT and Robin benchmarks. The utilization of the SMURF model provides a significant increase in accuracy for hard fault compensation.

7.3.2 Precision

Kappa statistics are widely used to analyze a machine learning's reliability. Kappa is intended to produce a quantitative measure of the magnitude of agreement between observations; in other words, represent the precision of prediction (114). A value closer to 1 represent better precision.

In our simulation results, we observe kappa value closer to 1 for all the benchmarks, which signifies the reliability of the machine learning output (Table 7.1).

Table 7.1: Prediction Precision Analysis: Kappa Statistic

Benchmark	**Btfly**	**DCT**	**Sobel**	**RGB**	**MAC**	**Div**	**Robin**
Kappa Statistics	0.9122	0.8906	0.8157	0.8604	0.9676	0.9635	0.8904

7.3.3 Comparison

In these last set of experiments, we compare different predictive methods. In particular, the Hoeffding tree explained previously vs. Linear Regression and RepTree machine learning algorithms. In all cases, the same benchmarks and the same faults were used as described previously. For Butterfly, DCT, MAC and Robin, we clearly observe that SMURF model has a lower MAE value. In the case of Sobel and RGB, the MAE values are close.

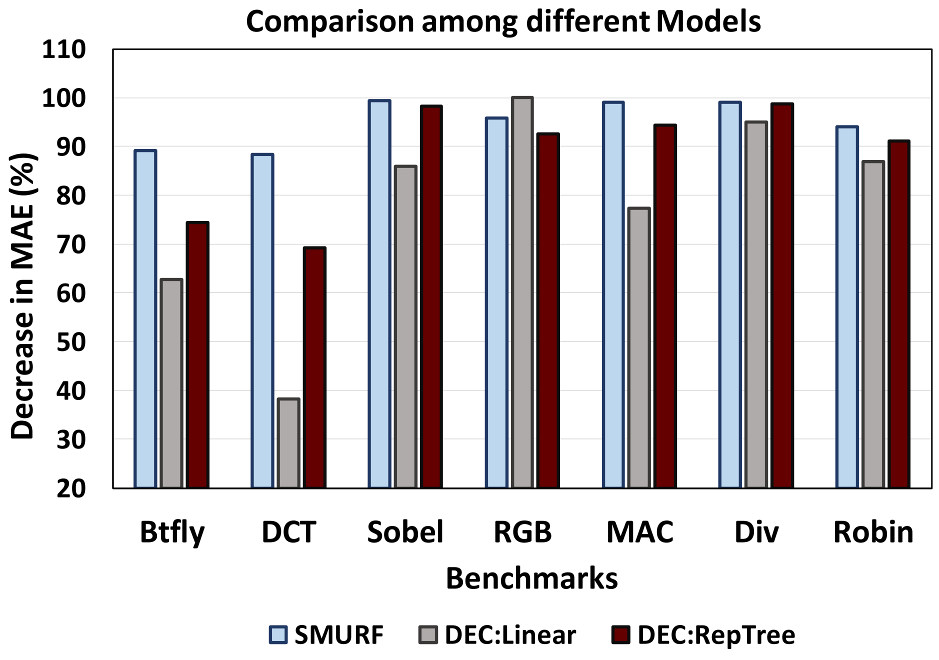


Figure 7.9: Comparison of Mean Absolute Error among different models.

Nevertheless, one significant observation is that the proposed Hoeffding tree model is consistently very accurate for all the benchmarks, 95% of the time, thus suggesting a significantly robust method. In contrast, the regression method leads only to 90% of the times to accurate results, while the RepTree in 93% of the times on average.

In addition, the size of the decision tree produced by the RepTree model is much larger compared to SMURF model. In order to highlight this, we inject the same stuck at fault in a specific node of the butterfly circuit and simulate using RepTree and the Hoeffding tree method. Fig. 7.10, shows the two trees produced by the two models. The tree generated by the Hoeffding model is significantly smaller than the RepTree, and therefore uses less area and consumes less power.

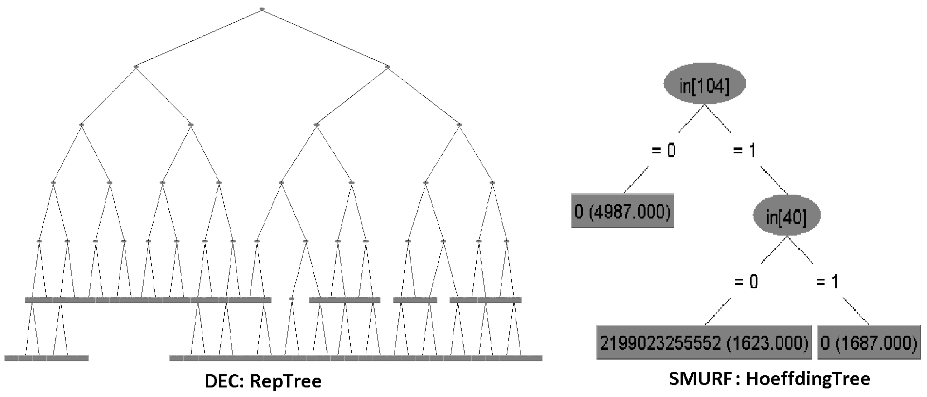


Figure 7.10: Tree size comparison.

7.3.4 Overhead

Finally, Table 7.2 shows the area overhead in percentage for various applications of our proposed compensation logic methodology. For most of the applications, the area overhead is on average 5%. We believe that this overhead is acceptable, considering the improvement in error correction achieved.

Table 7.2: Area overhead

Benchmarks	**Btfly**	**DCT**	**Sobel**	**RGB**	**MAC**	**Div**	**Robin**
Area Overhead (%)	7.26	6.41	9.33	3.88	3.37	1.53	8.29

7.4 Conclusion and Future Work

In this work, we have proposed a low-cost low-level recovery/repair method for permanent hardware faults to reduce the impact of hard faults in hardware accelerators that tolerate a certain level of errors by using supervised learning algorithms. We have performed gate-level experiments for a variety of complex, multi-level circuits and provided comprehensive experimental results for diverse complex hardware accelerators, showing significant error reduction.

Our proposed method is based on compensating for the effect of faults on the output by predicting the relative difference of the output rather than predicting the actual full output. The proposed method is shown to improve the exactly accurate result by 50% and decrease the overall mean error rate by 90% with an area overhead of 5%.

The proposed method has the additional advantages that it considers the accelerator as a black box. The complexity of the solution does not depend on the size or complexity of the application. It only depends on the number of inputs and outputs. As a result, our proposed method is scalable as the size of the SMURF module will not increase along with the size and complexity of the application.

CHAPTER 8
CONCLUSION AND FUTURE WORK

This section concludes the work completed in this dissertation and addresses possible future work directions.

8.1 Conclusion

One of the main benefits of C-based VLSI design over traditional RT-level design: The ability to generated micro-architectures with different characteristics from the same behavioral description, is leveraged to automatically generate reliable TMR systems, by either performing space and/or time redundancy. Experimental results show, the proposed method provides a 20% reliability increase compared to the most common approach of simply using a single micro-architecture and instantiating it multiple times with no significant area or timing overhead.

Next, we have shown that raising the level of abstraction from RT-level to C-level has the additional advantage of being able to increase the design diversity to detect common mode failures. Experimental results show that the proposed method provides a significant diversity increment compared to using RTL-based exploration for diverse design generation.

Furthermore, we have extensively studied different diversity metrics and compared them qualitatively. We have also presented a learning-based diversity estimation method to facilitate the generation of redundant systems that mitigate the effect of common-mode failures. This diversity estimation method has been used in a genetic algorithm-based HLS design space explorer to find design pairs with the highest diversity.Experimental results show, our proposed method is a fast and efficient way to generate diverse designs to protect the system against CMFs.

A low-cost low-level recovery/repair method for permanent hardware faults to reduce the impact of hard faults in hardware accelerators that tolerate a certain level of errors by using supervised learning algorithms is also proposed. Experimental results show that the proposed method improves the accuracy by 50% and decreases the overall mean error rate by 90% with an area overhead of 5% compared to execution without fault compensation.

8.2 Future Works

Future work could extend this work to additional components of the MPSoCs other than only the HWAcc. In addition, the methods, ARMORED, PMDE and SMURF can be combined to produce a more robust system. Furthermore, these methods can be further studied in more complex applications such as neural network, face recognition, etc.

REFERENCES

[1] B. C. Schafer and K. Wakabayashi, “Machine Learning Predictive Modelling High-level Synthesis Design Space Exploration,” *IET computers & digital techniques*, 2012.

[2] G. E. Moore, “Cramming More Components onto Integrated Circuits, Reprinted from Electronics,” *IEEE Solid-state Circuits Society Newsletter*, 2006.

[3] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc, “Design of Ion-implanted MOSFET’s with very Small Physical Dimensions,” *IEEE Journal of Solid-State Circuits*, 1974.

[4] “International Technology Roadmap for Semiconductors,” 2009.

[5] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi, “Modeling the Effect of Technology trends on the Soft Error Rate of Combinational Logic,” in *Dependable Systems and Networks (DSN)*, 2002.

[6] J. T. Wallmark and S. Marcus, “Minimum Size and Maximum Packing Density of Non-redundant Semiconductor Devices,” *Proceedings of the IRE*, 1962.

[7] D. Binder, E. C. Smith, and A. Holman, “Satellite Anomalies from Galactic Cosmic Rays,” *Transactions on Nuclear Science*, 1975.

[8] R. Ubar, J. Raik, and H. T. Vierhaus, “Design and Test Technology for Dependable Systems-on-chip,” 2011.

[9] A. Aitken, “Quantification of the Occurrence of Common-mode Faults in Highly Reliable Protective Systems,” International Atomic Energy Agency, Tech. Rep., 1978.

[10] S. Mitra, N. R. Saxena, and E. J. McCluskey, “Common-mode Failures in Redundant VLSI systems: A Survey,” *IEEE Transactions on Reliability*, 2000.

[11] P. Coussy, D. D. Gajski, M. Meredith, and A. Takach, “An Introduction to High-level Synthesis,” *Design & Test of Computers*, 2009.

[12] A. Takach, “Design and Verification using High-level Synthesis,” in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2016.

[13] M. R. Alam, M. E. S. Nasab, and S. M. Fakhraie, “Power Efficient High-level Synthesis by Centralized and Fine-grained Clock Gating,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2015.

[14] H. Ren, “A Brief Introduction on Contemporary High-level Synthesis,” in *International Conference on IC Design & Technology (ICICDT),*, 2014.

[15] J. Cong and Z. Zhang, "An Efficient and Versatile Scheduling Algorithm based on SDC Formulation," in *Design Automation Conference (DAC)*, 2006.

[16] A. M. Sllame and V. Drabek, "An Efficient List-based Scheduling Algorithm for High-level Synthesis," in *Proceedings Euromicro Symposium on Digital System Design. Architectures, Methods and Tools*, 2002.

[17] E. Kang, E. Jackson, and W. Schulte, "An Approach for Effective Design Space Exploration," in *Monterey Workshop*, 2010.

[18] I. Ahmad, M. K. Dhodhi, and F. H. Hielscher, "Design-space Exploration for High-level Synthesis," in *IEEE Annual International Phoenix Conference on Computers and Communications*, 1994.

[19] G. Zhong, V. Venkataramani, Y. Liang, T. Mitra, and S. Niar, "Design Space Exploration of Multiple Loops on FPGAs using High Level Synthesis," in *International Conference on Computer Design (ICCD)*, 2014.

[20] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, 2002.

[21] B. C. Schafer, T. Takenaka, and K. Wakabayashi, "Adaptive Simulated Annealer for High Level Synthesis Design Space Exploration," in *International Symposium on VLSI Design, Automation and Test*, 2009.

[22] H.-Y. Liu and L. P. Carloni, "On Learning-based Methods for Design-space Exploration with High-level Synthesis," in *Design Automation Conference (DAC)*, 2013.

[23] W. Meeus, K. Van Beeck, T. Goedemé, J. Meel, and D. Stroobandt, "An Overview of Today's High-level Synthesis Tools," *Design Automation for Embedded Systems*, 2012.

[24] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi *et al.*, "A Survey and Evaluation of FPGA High-level Synthesis Tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016.

[25] Mentor, "Catapult® High-Level Synthesis," URL: https://www.mentor.com/.

[26] NEC, "CyberWorkBench," Version 6.1, URL: https://www.cyberworkbench.com, 2018.

[27] Intel, "Intel® HLS Compiler," URL: https://www.intel.com/.

[28] The University of Toronto, "LegUp High-Level Synthesis," URL: http://legup.eecg.utoronto.ca/.

[29] MathWorks, “HDL Coder.”

[30] Cadence, “Stratus High-Level Synthesis.”

[31] Synopsys, “Synphony HLS,” URL: https://www.synopsys.com/.

[32] Xilinx, “Vivado High-Level Synthesis,” uRL: https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html.

[33] P. Cunningham, M. Cord, and S. J. Delany, “Supervised Learning,” in *Machine Learning Techniques for Multimedia*, 2008.

[34] B. Johnson, “Fault-tolerant Microprocessor-based Systems,” *IEEE Micro*, 1984.

[35] “Murphy’s laws site all the laws of murphy in one place.” [Online]. Available: http://www.murphys-laws.com/murphy/murphy-true.html

[36] E. Dubrova, *Fault-tolerant Design.* Springer, 2013.

[37] J.-C. Laprie, “Dependable Computing and Fault-tolerance,” *Digest of Papers FTCS-15*, 1985.

[38] A. Aviziens, “Fault-tolerant Systems,” *IEEE Transactions on Computers*, 1976.

[39] C. Constantinescu, “Intermittent Faults in VLSI Circuits,” in *Workshop on Silicon Errors in Logic-System Effects*, 2007.

[40] F. L. Kastensmidt, L. Carro, and R. A. da Luz Reis, *Fault-tolerance Techniques for SRAM-based FPGAs.* Springer, 2006.

[41] R. Baumann, “The Impact of Technology Scaling on Soft Error Rate Performance and Limits to the Efficacy of Error Correction,” in *Digest. International Electron Devices Meeting,*, 2002.

[42] C. Constantinescu, “Trends and Challenges in VLSI Circuit Reliability,” *IEEE Micro*, 2003.

[43] J. H. Adams Jr, R. Silberberg, and C. Tsao, “Cosmic Ray Effects on Microelectronics. Part 1. The Near-Earth Particle Environment,” Naval Research Lab Washington DC, Tech. Rep., 1981.

[44] R. J. Weber *et al.*, “Reconfigurable Hardware Accelerators for High Performance Radiation Tolerant Computers,” Ph.D. dissertation, Montana State University-Bozeman, College of Engineering, 2014.

[45] A. Avizienis, “Design of Fault-tolerant Computers,” in *AFIPS Fall Joint Computing Conference*, 1967.

[46] J. Von Neumann, "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," *Automata Studies*, 1956.

[47] I. Koren and C. M. Krishna, *Fault-tolerant Systems.* Elsevier, 2010.

[48] A. Balachandran, N. Veeranna, and B. C. Schafer, "On Time Redundancy of Fault Tolerant C-Based MPSoCs," in *Computer Society Annual Symposium on VLSI (ISVLSI)*, 2016.

[49] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, 1948.

[50] R. W. Hamming, "Error Detecting and Error Correcting Codes," *The Bell System Technical Journal*, 1950.

[51] R. Karri and A. Orailoglu, "High-level Synthesis of Fault-tolerant ASICs," in *International Symposium on Circuits and Systems*, 1992.

[52] Z. Zhu, F. N. Taher, and B. Carrion Schafer, "Exploring Design Trade-offs in Fault-Tolerant Behavioral Hardware Accelerators," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, 2019.

[53] V. Castano and I. Schagaev, *Resilient Computer System Design.* Springer, 2015.

[54] A. Orailoglu and R. Karri, "A Design Methodology for the High-level Synthesis of Fault-tolerant ASICs," in *Workshop on VLSI Signal Processing*, 1992.

[55] ——, "Automatic Synthesis of Self-recovering VLSI Systems," *Transactions on Computers*, 1996.

[56] K. Wu and R. Karri, "Fault Secure Datapath Synthesis using Hybrid Time and Hardware Redundancy," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2004.

[57] L. Guerra, M. Potkonjak, and J. Rabaey, "High Level Synthesis for Reconfigurable Datapath Structures," in *International Conference on Computer-aided Design (ICCAD)*, 1993.

[58] A. Antola, V. Piuri, and M. Sami, "High-level Synthesis of Data Paths with Concurrent Error Detection," in *International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, 1998.

[59] S. Tosun, N. Mansouri, E. Arvas, M. Kandemir, and Y. Xie, "Reliability-centric High-level Synthesis," in *Design, Automation and Test in Europe (DATE)*, 2005.

[60] M. Glaß, M. Lukasiewycz, T. Streichert, C. Haubelt, and J. Teich, "Reliability-aware System Synthesis," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2007.

[61] D. B. Limbrick, S. Yue, W. H. Robinson, and B. L. Bhuva, "Impact of Synthesis Constraints on Error Propagation Probability of Digital Circuits," in *International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2011.

[62] G. Asadi and M. B. Tahoori, "An Analytical Approach for Soft Error Rate Estimation in Digital Circuits," in *International Symposium on Circuits and Systems (ISCAS)*, 2005.

[63] T. Inoue, H. Henmi, Y. Yoshikawa, and H. Ichihara, "High-level Synthesis for Multi-cycle Transient Fault Tolerant Datapaths," in *International On-Line Testing Symposium (IOLTS)*, 2011.

[64] L. Chen and M. Tahoori, "Reliability-aware Register Binding for Control-flow Intensive Designs," in *Design Automation Conference (DAC)*, 2014.

[65] A. Shastri, G. Stitt, and E. Riccio, "A Scheduling and Binding Heuristic for High-level Synthesis of Fault-tolerant FPGA Applications," in *International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2015.

[66] G. Lee, D. Agiakatsikas, T. Wu, E. Cetin, and O. Diessel, "TLegUp: A TMR Code Generation Tool for SRAM-Based FPGA Applications Using HLS," in *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017.

[67] S. T. Fleming and D. B. Thomas, "StitchUp: Automatic Control Flow Protection for High Level Synthesis Circuits," in *Design Automation Conference (DAC)*, 2016.

[68] A. Avizienis, "On the Implementation of N-version Programming for Software Fault Tolerance during Execution," *Proc. COMPSAC*, 1977.

[69] F. B. Cohen, "Operating System Protection Through Program Evolution," *Computers & Security*, 1993.

[70] A. Höller, N. Kajtazovic, K. Römer, and C. Kreiner, "Evaluation of Diverse Compiling for Software Fault Tolerance," in *Design Automation and Test in Europe (DATE)*, 2015.

[71] S. Mitra, N. R. Saxena, and E. J. McCluskey, "A Design Diversity Metric and Reliability Analysis for Redundant Systems," in *International Test Conference (ITC)*, 1999.

[72] S. Mitra, "Globally Optimized Robust Systems to Overcome Scaled CMOS Reliability Challenges," in *Design Automation and Test in Europe (DATE)*, 2008.

[73] S. Mitra, N. R. Saxena, and E. J. McCluskey, "A Design Diversity Metric and Analysis of Redundant Systems," *IEEE Transactions on Computers*, 2002.

[74] ——, "Efficient Design Diversity Estimation for Combinational Circuits," *IEEE Transactions on Computers*, 2004.

[75] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., 1979.

[76] S. Alcaide, C. Hernandez, A. Roca, and J. Abella, "DIMP: A Low-Cost Diversity Metric Based on Circuit Path Analysis," in *Design Automation Conference (DAC)*, 2017.

[77] S. Mitra and E. J. McCluskey, "Combinational Logic Synthesis for Diversity in Duplex Systems," in *International Test Conference (ITC).* IEEE, 2000.

[78] E. J. McCluskey, "Logic Design Principles with Emphasis on Testable Semicustom Circuits," 1986.

[79] K. Huang and J. Abraham, "Algorithm-Based Fault Tolerance for Matrix Operations," *IEEE Transactions on Computers*, 1984.

[80] S. Almukhaizim, P. Drineas, and Y. Makris, "Cost-driven Selection of Parity Trees," in *VLSI Test Symposium*, 2004.

[81] S. K. S. Hari, S. V. Adve, and H. Naeimi, "Low-cost Program-level Detectors for Reducing Silent Data Corruptions," in *International Conference on Dependable Systems and Networks (DSN)*, 2012.

[82] N. Shanbhag, "Reliable and Energy-Efficient Digital Signal Processing," in *Design Automation Conference (DAC)*, 2002.

[83] R. A. Abdallah and N. R. Shanbhag, "Robust and Energy Efficient Multimedia Systems via Likelihood Processing," *IEEE Transactions on Multimedia*, 2013.

[84] B. Shim, N. R. Shanbhag, and S.-J. Lee, "Energy-Eefficient Soft Error-Tolerant Digital Signal Processing," in *Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, 2003.

[85] G. V. Varatkar and N. R. Shanbhag, "Energy-efficient Motion Estimation using Error-tolerance," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2006.

[86] A. Sharma, J. Sloan, L. F. Wanner, S. H. Elmalaki, M. B. Srivastava, and P. Gupta, "Towards Analyzing and Improving Robustness of Software Applications to Intermittent and Permanent Faults in Hardware," in *International Conference on Computer Design (ICCD)*, 2013.

[87] M.-L. Li, P. Ramachandran, S. K. Sahoo, S. V. Adve, V. S. Adve, and Y. Zhou, "Trace-based Microarchitecture-level Diagnosis of Permanent Hardware Faults," in *International Conference on Dependable Systems and Networks (DSN)*, 2008.

[88] C. Carmichael and C. W. Tseng, "Correcting Single-event Upsets in Virtex-4 FPGA Configuration Memory," *Xilinx Corporation*, 2009.

[89] P. Traverse, "AIRBUS and ATR system Architecture and Specification," in *Software diversity in computerized control systems.* Springer, 1988.

[90] M. J. Flynn and W. Luk, *Computer System Design: System-on-Chip.* John Wiley & Sons, 2011.

[91] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-performance Microprocessor," in *Symposium on Microarchitecture (MICRO)*, 2003.

[92] A. E. Eiben, P.-E. Raue, and Z. Ruttkay, "Genetic Algorithms with Multi-parent Recombination," in *International Conference on Parallel Problem Solving from Nature*, 1994.

[93] B. Carrion Schafer and A. Mahapatra, "S2CBench: Synthesizable SystemC Benchmark Suite for High-level Synthesis," *IEEE Embedded Systems Letters*, 2014.

[94] "Analytic Solver Platform," Frontline Systems, Inc., http://www.solver.com, 2017.

[95] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software: An Update," *ACM SIGKDD Explorations Newsletter*, 2009.

[96] Synopsys, "Design Compiler L-2016.03-SP3," URL: https://www.synopsys.com, 2016.

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

[98] F. Ferrandi, P. L. Lanzi, D. Loiacono, C. Pilato, and D. Sciuto, "A Multi-objective Genetic Algorithm for Design Space Exploration in High-Level Synthesis," in *IEEE Computer Society Annual Symposium on VLSI*, 2008.

[99] Mentor Graphics, "ModelSim SE 10.1b," URL: https://www.mentor.com, 2011.

[100] S. Borkar, "Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation," *IEEE Micro*, 2005.

[101] M. Stanisavljević, A. Schmid, and Y. Leblebici, "Reliability, Faults, and Fault Tolerance," in *Reliability of Nanoscale Circuits and Systems*, 2011.

[102] I. Koren and Z. Koren, "Defect Tolerance in VLSI circuits: Techniques and Yield Analysis," *Proceedings of the IEEE*, 1998.

[103] I. Koren and A. D. Singh, "Fault Tolerance in VLSI Circuits," *Computer*, 1990.

[104] M. A. Breuer, S. K. Gupta, and T. Mak, "Defect and Error Tolerance in the Presence of Massive Numbers of Defects," *Design & Test of Computers*, 2004.

[105] K. Asanovic *et al.*, "The Landscape of parallel Computing Research: A View from Berkeley," Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Tech. Rep., 2006.

[106] S. Xu and B. C. Schafer, "Exposing Approximate Computing Optimizations at Different Levels: From Behavioral to Gate-Level," *IEEE Transactions on Very Large Scale Integration (TVLSI) Systems*, 2017.

[107] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-Simplify: A Unified Design Paradigm for Approximate and Quality Configurable Circuits," in *Conference on Design, Automation and Test in Europe (DATE)*, 2013.

[108] S. Kotsiantis, "Supervised Machine Learning: A Review of Classification Techniques," *Informatica*, 2007.

[109] P. Domingos and G. Hulten, "Mining High-speed Data Streams," in *International conference on Knowledge Discovery and Data Mining (KDD)*, 2000.

[110] OpenCores, "Free Open Source IP Cores and Chip Design," 2017. [Online]. Available: https://opencores.org/projects

[111] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The EPFL Combinational Benchmark Suite," in *International Workshop on Logic & Synthesis (IWLS)*, 2015.

[112] C. Wolf, "Yosys Open Synthesis Suite," 2014.

[113] V. D. Agrawal, A. Prasad, and M. V. Atre, "Fault Collapsing via Functional Dominance," in *International Test Conference (ITC)*, 2003.

[114] A. J. Viera, J. M. Garrett *et al.*, "Understanding Interobserver Agreement: The Kappa Statistic," *Fam Med*, 2005.

BIOGRAPHICAL SKETCH

Farah Naz Taher is a PhD candidate in the Department of Electrical & Computer Engineering, The University of Texas at Dallas. Her research interests include Digital Design, Fault Tolerance, Machine Learning, and High-level Synthesis. Farah received her MS degree in Electrical Engineering from Auburn University, Alabama and BS degree in Electronics and Telecommunication Engineering from North South University, Bangladesh.

CURRICULUM VITAE

Education

PhD in Electrical Engineering Fall 2019

University of Texas at Dallas – Richardson, TX

PhD Dissertation: Fault Tolerance in Hardware Accelerators: Detection and Mitigation

Master of Science in Electrical Engineering Summer 2013

Auburn University – Auburn, AL GPA: 4.00/4.00

Thesis: Analog Bus for Low Power On-Chip Digital Application

Bachelor of Science in Electronic and Telecommunication Engineering Fall 2008

North South University – Dhaka, Bangladesh GPA: 3.76/4.00

Project: FinFET - The Promising Successor of Conventional MOSFET in Next Generation Nano Technology Era

Publications

Farah N. Taher, A. Balachandran, and B. Carrion Schafer, "Learning-based Diversity Estimation: Leveraging the Power of High-level Synthesis to mitigate Common-mode Failure" (Under Review).

Z. Zhu, **Farah N. Taher**, and B. Carrion Schafer," Exploring Design Trade-offs in Fault Tolerant Behavioral Hardware Accelerator", in the 29th Great Lakes Symposium on VLSI (GLSVLSI), May 2019.

M. R. Babu, **Farah N. Taher**, A. Balachandran, B. Carrion Schafer, “Efficient Hardware Acceleration for Design Diversity Calculation to Mitigate Common Mode Failures” in the 27th IEEE International Symposium On Field-Programmable Custom Computing Machines (FCCM), April 2019.

Farah N. Taher, M. Joslin, A. Balachandran, Z. Zhu, and B. Carrion Schafer, “Common-Mode Failure Mitigation: Increasing Diversity through High-Level Synthesis”, in the 22nd Design, Automation and Test in Europe Conference (DATE), March 2019.

Farah N. Taher, M. Kishani and B. Carrion Schafer, “Design and Optimization of Reliable Hardware Accelerators: Leveraging the Advantages of High-Level Synthesis”, in the 24th IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS), July 2018.

Farah N. Taher, J. Callenes-Sloan and B. Carrion Schafer, “A Machine Learning based hard fault Recuperation model for Approximate Hardware Accelerators”, in the 55th Design Automation Conference (DAC), June 2018

Farah N. Taher and J. Callenes-Sloan, "Hardware Fault Compensation Using Discriminative Learning", in the 21th Pacific Rim International Symposium of Dependable Computing (PRDC), November 2015.

Farah N. Taher and J, Callenes-Sloan, "Hardware Fault Compensation Using Supervised Learning", in the 11th Workshop on Silicon Errors in Logic -System Effects (SELSE), March 2015.

Farah N. Taher, S. Sindia, V. D. Agrawal, “An Analog Bus for Low Power On-Chip Digital Communication”, in Poster Session of 50th Design Automation Conference (DAC), June 2013.

Research Experience

Department of Electrical Engineering - The University of Texas at Dallas

PhD Candidate and Graduate Research Assistant Aug 2014 – Aug 2019

Responsibility: Develop and evaluate novel fault tolerance mechanisms for emerging hardware and software designs. Explore algorithmic and architectural techniques of harnessing system-faults for better yield and performance.

Department of Electrical and Computer Engineering **-** Auburn University

Graduate Research Assistant Aug 2012 – Jul 2013

Responsibility: Explored various open problems in low power VLSI Designs that resulted in the thesis on “Analog Bus for Low Power On-Chip Digital Application”.

Alabama Microelectronics Science and Technology Center - Auburn University

Graduate Research Assistant Aug 2011– May 2012

Responsibility: Assisted the lab director in maintaining the class 100-clean room. Conducted experiments and maintained the inventory management system.

Teaching Experience

Department of Electrical Engineering - The University of Texas at Dallas

Graduate Teaching Assistant Jan 2014 – Dec 2017

Courses: Computer Architecture, Electronic Device Laboratory, Introduction to Digital Systems.

Responsibility: Assisted the instructor in updating the course content, design assignments, and proctoring exams. Evaluated quizzes, assignments, projects and lab work. Engaged with the students to help them achieve insightful understanding of the course material.

Department of Electrical Engineering & Computer Science - North South University, Dhaka, Bangladesh

Teaching Assistant Sep 2007 – Apr 2009

Courses: Theory of Electromagnetic Fields, Data Communications and Computer Networks, Fiber Optic Communication System, VLSI Chip Design

Related Coursework

Advanced VLSI Design, VLSI Testing, Computer Architecture, Trusted and Secure Integrated Circuits and Systems, Low-Power Electronic Circuit, Microelectronic Fabrication

Technical Skills

- Programming Languages: Verilog, Python, C, MATLAB
- Machine Learning: Weka, R
- High-level Synthesis: CyberWorkBench
- Circuit Simulation and Layout Tools: Design Compiler, HSPICE, LTspice, Lasi7
- VLSI Design: ModelSim, Xilinx ISE, Yosys, LabView
- Information Processing: Microsoft Office