Circuit and system fault tolerance techniques

Imran Wali

THÈSE

Pour obtenir le grade de

Docteur

Délivré par l'**Université de Montpellier**

Préparée au sein de l'école doctorale **I2S**
Et de l'unité de recherche **LIRMM**

Spécialité: **SYAM**

Présentée par **Imran WALI**

Circuit and System Fault Tolerance Techniques

Soutenue le 30 Mars 2016, devant le jury composé de:

M. Fabrice MONTEIRO, Professeur, Université de Lorraine	Rapporteur
Mme. Lirida NAVINER, Professeur, Telecom ParisTech	Rapporteur
M. Matteo SONZA REORDA, Professeur, Politecnico di Torino	Examinateur
M. Alberto BOSIO, MCF HDR, Université de Montpellier	Examinateur
M. Arnaud VIRAZEL, MCF HDR, Université de Montpellier	Directeur de thèse
M. Patrick GIRARD, DR CNRS, LIRMM	CoDirecteur de thèse

Acknowledgements

First and foremost, I would like to thank my Lord, the Almighty, for being close and responsive throughout the course of my thesis. Only due to His countless gifts of sense, intellect, patience, health, family and many more, I could complete this task.

I would like to express my deepest gratitude to my thesis director, Prof. Arnaud Virazel. With his encouraging and supporting attitude, meeting him has always been a work-stress reliever for me. His profound guidance and invaluable advises helped keeping my research well directed and my progress on schedule while maintaining my autonomy.

My sincere thanks also goes to my co-director, Prof Patrick Girard, for his immense knowledge, motivation and support throughout the research. Also for his thought-provoking ideas and helpful criticism in improving the quality of this manuscript and other publications. I also appreciate Prof. Alberto Bosio's insightful comments and advises which helped me solve various problems during my research.

My grateful thanks are also extended to Prof. Matteo Sonza Reorda for his generous support and effort to actively maintain the collaborative partnership, LAFISI, and his valuable and constructive suggestions that helped me enrich my ideas.

I am also thankful to him for carefully reviewing my work at different stages of my thesis. I would like to thank the rest of my thesis committee members, Prof. Lirida Naviner and Prof. Fabrice Monteiro for reviewing this manuscript and providing positive feedback.

I would also like to acknowledge all my master students who provided me an opportunity to enrich my understanding through sharing my knowledge about the subject and also for their valuable contribution to my work. I wish to acknowledge the support received form my friends at LIRMM for the various discussions and brain storming sessions. Also for making the last three years memorable.

Finally, none of this would have been possible without the love and patience of my family. Their constant support and strength has aided and encouraged me throughout this endeavor.

Abstract

Semiconductor is one of the most reliable inventions when engineered and used with longevity in mind. However, the increasing demand of fast and highly featured products has drastically changed the reliability realm in the recent years. The means of improving the reliability of nano-metric technology circuits encompass techniques that tackle reliability issues at the level of technology, design and manufacturing. Absolutely necessary but these techniques are almost inevitably imperfect. Therefore, it becomes essential to reduce the consequence of the "remaining" faults using fault tolerance techniques.

This thesis focuses on improving and developing new low-power fault tolerance techniques that combine the attractive features of different types of redundancies to tackle permanent and transient faults and addresses the problem of error detection and confinement in modern microprocessor cores. Our case study implementation results show that a power saving of up to 20% can be achieved in comparison with fault tolerance techniques that use only one type of redundancy, and offer lifetime reliability improvement.

With the objective to further improve the efficiency in terms of cost and fault tolerance capability we present a design space exploration and an efficient cost-reliability trade-off analysis methodology to selectively harden logic circuits using hybrid fault tolerant techniques. The outcome of the two studies establish that hybrid fault tolerant approaches provide a good foundation for building low-power reliable circuits and systems from future technologies, and our experimental results set a good starting point for further innovative research in this area.

List of Acronyms

Table of contents

List of figures

List of tables

Introduction

Technology scaling allows realization of more and more complex system on a single chip. This high level of integration leads to increased current and power densities and causes early device and interconnect wear-out. It leads to shifts in electrical characteristics of circuit elements or renders them permanently damaged. Moreover, a high integration density makes these complex systems difficult to test and some test-escaped manufacturing defects are encountered as errors only during infield operation. In addition, there are failures not caused by wear-out nor escaped manufacturing defects, but due to increased susceptibility of transistors to high energy particles from atmosphere or from within the packaging. Devices operating at reduced supply voltages are more prone to charge related phenomenon caused by high-energy particle strikes referred to as Single Event Effect (SEE). They experience particle-induced voltage transients called Single Event Transient (SET) or particle-induced bit-flips in memory elements also known as Single Event Upset (SEU).

High-performance microprocessors being at the forefront of technology are becoming increasingly vulnerable to hard and soft errors due to their growing complexity, high operating frequencies and fragility of future Very Large Scale Integration (VLSI) technology nodes. SEE in CL are playing important role in increasing Soft Error Rate (SER), which was historically considered as a cause of particle strikes in memory elements. In addition, it is estimated that the susceptibility of CL circuits to SET nearly doubles as the technology scales from 45nm to 16nm. Hence, it is inevitable for the industry to prevent reliability from becoming a bottleneck for the development of high-performance and low-power microprocessors.

The means of improving the reliability of nano-metric technology circuits encompass techniques that tackle reliability issues at the level of technology, design and manufacturing. Absolutely necessary but these techniques are almost inevitably imperfect. Therefore, it becomes essential to reduce the consequence of the "remaining" faults using fault tolerance techniques. These techniques employ information, timing and hardware redundancies to guarantee correct operations despite the presence of faults.

Various solutions using fault tolerant techniques for robustness improvement can be found in the literature, but a very few can address tolerance to both transient and permanent faults. These techniques generally rely on slow recovery mechanisms, thus are not suitable for highly interactive applications. For example, the method presented in [62] has little area overhead but runs Built-In Self-Test (BIST) during periodic time intervals to detect the presence of permanent faults and uses deep rollbacks that have a severe impact on performance. Fault-tolerant architectures like Razor [29], CPipe [89] and STEM [5] incorporate power saving and performance enhancement mechanisms like Dynamic Frequency Scaling (DFS) and Dynamic Voltage Scaling (DVS) to operate circuits beyond their worst-case limits. These architectures generally target timing errors and are not effective to deal with permanent faults. For instance, Razor only deals with timing faults and CPipe duplicates CL blocks in the pipeline to detect and correct transient and timing errors and can also detect permanent faults, but it does not offer provision for their correction.

Besides fault tolerance capability, power consumption is also a rising concern in the industry. In fact, as fault-tolerance becomes necessary in mass products, limiting power consumption of these techniques is one of the key factors in digital design. The classical fault tolerance techniques like TMR can effectively handle transient and permanent fault but cannot respect the low-power consumption demands.

Pipelining is a key technique to increase throughput in modern microprocessors by improving the resource utilization. But the complexity of interactions between pipeline stages make error detection and correction a major hurdle in designing high-performance reliable processing cores. To the best of our knowledge, none of the fault tolerance techniques in the literature address the problem of error detection and confinement in non-linear pipelined circuits nor in pipelines with memory interfaces.

Selecting the ideal trade-off between reliability and cost associated with a fault tolerant architecture generally involves an extensive design space exploration. Employing state-of-the-art reliability estimation methods makes this exploration unscalable with the design complexity.

This thesis focuses on improving and developing new low-power fault tolerance techniques that combine the attractive features of different types of redundancies for robustness improvement of future technology-scalable digital circuits and systems against transient and permanent faults. It addresses the problem of error correction in CL parts of complex pipeline circuits. Furthermore, it develops a fault tolerance capability assessment framework and low-cost reliability estimation techniques for use in cost-reliability trade-off analysis. Overall, this thesis establishes that hybrid fault tolerant approaches provide a good foundation for building low-power reliable circuits and systems from future technologies, and our experimental results set a good starting point for further innovative research in this area.

The following of this manuscript is divided in five chapters:

- Chapter 1 details the context and motivation of our research. It starts with a discussion of the trends in semiconductor technology scaling and their impact on the reliability of nano-metric technology circuits. Later in Chapter 1 we briefly discuss the different approaches to improve their reliability and finally we end the discussion by presenting the objectives of our work.

- Chapter 2 covers the state-of-the-art in the field of fault tolerant architectures and robustness assessment techniques. It discusses some basic concepts of error detection and correction, followed by an overview of a set of state-of-the-art fault tolerant architectures, with a special focus on a hybrid fault tolerant architecture. In Section 2.3 we first develop the context by discussing some state-of-the-art methods employed for robustness assessment of circuits and systems and then present a gate-level fault injection framework for fault tolerance capability assessment of digital circuits. The last part of Chapter 2 presents an experimental study that quantitatively compare different fault tolerant architectures on the basis of their area, power, performance and their fault tolerance capability.

- In Chapter 3 we present a hybrid fault tolerant architecture for reliability improvement of complex pipelined circuits. We start the discussion by highlighting some issues that hinder error detection and correction in complex pipelined circuits. Then we discuss the extension of hybrid fault tolerant architecture discussed in Chapter 2 to make it solve the aforementioned problems. An experimental case study of the application of the developed fault tolerant architecture and other state-of-the-art solutions on a pipelined microprocessor is presented and finally with a comparative analysis we conclude the chapter.

- Chapter 4 consist of a design space exploration study aimed to optimize the cost and fault tolerance capability of the proposed hybrid fault tolerant architecture. We experimentally assess three proposed design improvements and present the results. The chapter is concluded by identifying the best candidate.

- Chapter 5 develops the principles of a low-cost reliability estimation method and a selective hybrid fault tolerant architecture. It starts with an overview of previous work in the area of selective hardening. In the subsequent sections we propose an efficient susceptibility analysis method to identify the most vulnerable circuit nodes for hardening and use it yo selectively harden some benchmark circuits. Also based on the susceptibility analysis method we also present a low-cost reliability estimation technique for fault tolerant architectures that use logic replication. In Section 5.5 we present the experimental result that compare the merits of circuits hardened to different extents and validate the hypothesis of the low-cost reliability estimation methods.

The Conclusion section summarizes the contributions of this work and presents some future perspectives.

Chapter 1

Context and Motivation

1.1 Semiconductor Technology Evolution

The steady growth of the global semiconductor industry over the past four decades has been driven by the demand for enhancing performance and functionality at reduced cost. This growth has been primarily facilitated by the continuous evaluation of semiconductor manufacturing technology. Transistors are scaled in each successive technology generation to increase their speed, improve packing density [2], decrease their power consumption and reduce cost. Thus, semiconductor technology scaling optimizes circuit performance and power consumption and allows realization of more and more complex system.

In November 1971, Intel introduced the world's first single-chip microprocessor, the Intel 4004. It had 2,300 transistors, ran at a clock speed of up to 740 KHz, and delivered 60,000 instructions per second while dissipating 0.5 watts. The following four decades witnessed exponential growth in compute power, a trend that has enabled applications as diverse as climate modeling, protein folding, electronic games and autonomous soft landings on extraterrestrial bodies.

Today's microprocessor chips employ billions of transistors, include multiple processor cores on a single silicon die, run at clock speeds measured in gigahertz, and deliver more than 4 million times the performance of the original 4004 [22]. The plots of Figures 1.1 and 1.2 show the trend of some technologic advancements in microprocessors in the past four decades. In 2015, Oracle presented the processor with over 10 billion transistors on the die, the most denser microprocessor in terms of transistor count. The major contributions to this massive advancement in semiconductor technology come from the field of lithography, advanced materials in manufacturing [70] and Electronic Design Automation (EDA) tools.

To reduce power dissipation of semiconductor devices, the supply voltage V_{dd} is also scaled, because dynamic power is proportional to the square of V_{dd} while leakage power is proportional

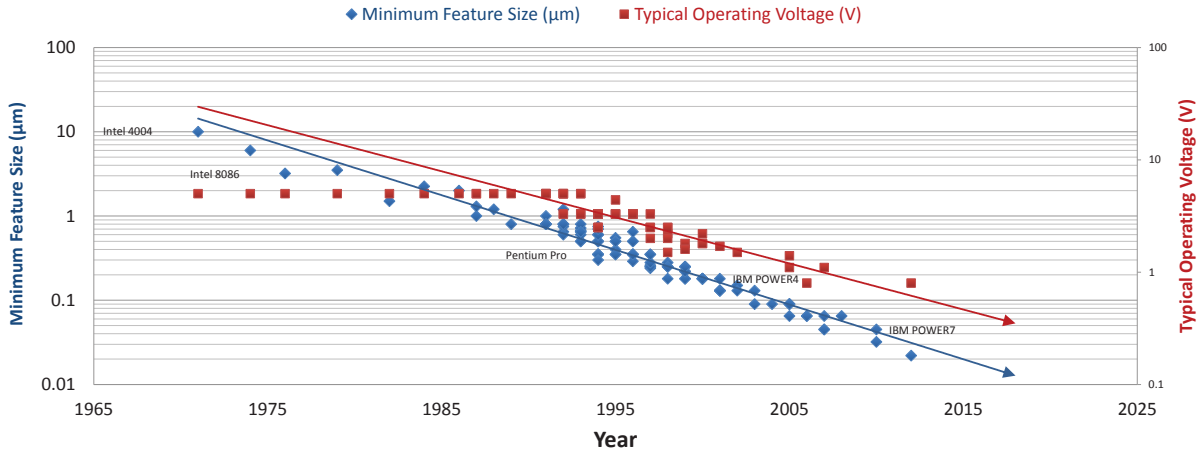


Fig. 1.1: Feature size and operating voltage scaling trend over years [22]

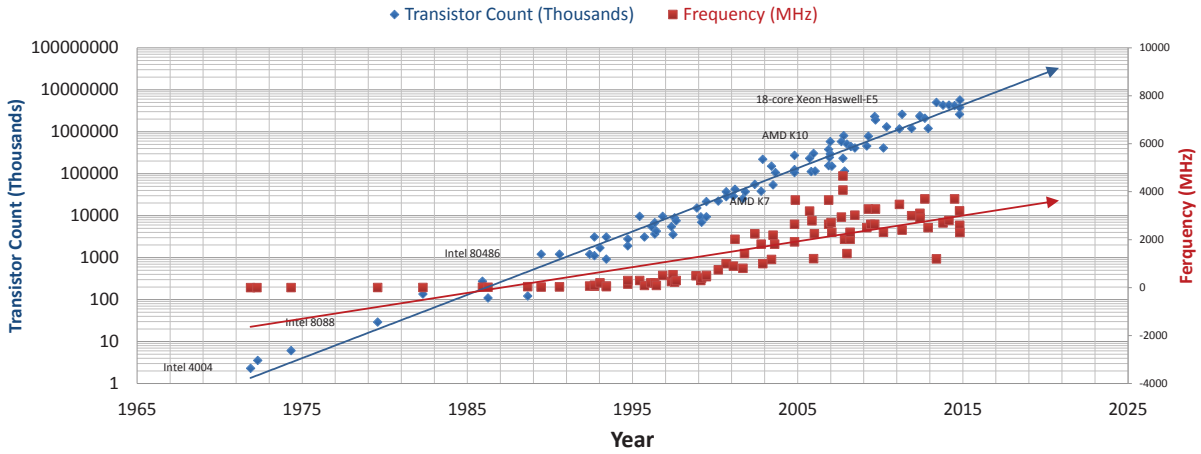


Fig. 1.2: Transistor count and clock frequency scaling trend over years [77]

to V_{dd}. However, voltage scaling only started in the late 80s because the industry had settled on 5V supplies in the early 70s to be compatible with bipolar Transistor-Transistor Logic (TTL) [21]. As power dissipation became unsustainable, this standard finally collapsed. V_{dd} was scaled within few years, first to 3.3V then to 2.5V, etc. In 2011, supply voltage of high-performance ICs was at 0.9V and predicted to be reduced to 0.66V in 2021 [46].

In Figure 1.2 we can see that in year 2005, the frequency scaling process reached the power wall limit at about 3 GHz. In fact, higher switching activity of transistors leads to higher power consumption. Even though these small transistors do not consume much, hundred millions of them are switching at the same time in less than five hundred millimeter square IC. This results in significant power density that must be limited to avoid breakdown of physical materials.

The manufacturing cost has also played an important role in the evolution of semiconductor technology. Building smaller physical structures demand more sophisticated thus expensive equipment. But this initial investment makes little difference in per wafer cost in view of the sheer production size of new high performance, enhanced featured and low-powered integrated circuits, that always remain in high demand.

1.2 Reliability threats in nano-metric technologies

While offering many advantages technological trends into the nanometer regime have lead to a host of reliability issues. The sources that affect the reliability of modern circuits and systems are wide spread including manufacturing process variations, manufacturing defects, wear-out and interference. In the following sections we discuss how these reliability impairment sources evolve with technology scaling.

1.2.1 Variability

Variability of transistor characteristics due to variations in Process, Voltage and Temperature (PVT) have always been an issue in integrated circuit design [105]. PVT variations prevent the circuit from functioning correctly even though each individual transistor behaves correctly [71]. As device scaling enters into the nano-metric regime the effect of PVT variations are becoming more and more pronounced. Manufacturing process variations are increased due to the lack of precise control on the fabrication processes at small-feature technologies. They causes deviation of transistor characteristics and the resulting wide distribution leads to lower parametric yield [11]. Supply voltage scaling brings it close to or even below the transistor threshold voltage (V_{th}) to achieve significant gains in energy efficiency. Once the supply voltage is lower than V_{th} the delay shows an exponential dependence on the supply voltage, and therefore the magnitude of supply voltage shifts can have a significant impact on circuit delay [16, 82]. Scaling also gives rise to chip power densities and inadequate heat sinking cause hot spots to appear. These fluctuations of ambient temperatures alter the timing characteristics of circuits [51].

A conventional approach to deal with the problem of variability is to introduce large voltage and frequency guard-bands which considerably impact the power consumption and performance, cannot respond to rapid environmental changes [23] and implies a sharp increase in chip cost. Effectively dealing with variability to maintain or improve the high performance and energy efficient systems while satisfying the historical standards for reliability is becoming increasingly challenging with the scaling of technology [80].

1.2.2 Manufacturing Defects

Semiconductor manufacturing process may induce permanent defects in a chip during one or more of hundreds of process steps involving implantation, etching, deposition, polarization, cleaning and lithography [46] due to imperfections. The device miniaturization is making advanced technology devices nodes increasing vulnerable to these manufacturing irregularities. It is important that the testing be thorough as possible to uncover defective chips before they are shipped out [73]. However, according to past microprocessor data, the die size remains relatively constant [44], whereas the number of transistors per chip double every 2 to 3 years. This means that defect densities continues to increase. Moreover technology scaling and increasing complexity give rise to defects more subtle and difficult to detect [81, 84]. All these factors when combined with aggressive time-to-market objectives cause test escapes and raise reliability concerns.

1.2.3 Wear-out

As we have seen in Figure 1.1, the area scaling has had an exponential rate but the supply voltage (V_{dd}) scaling has been quite slow. The two main reasons for slow V_{dd} scaling are to keep up with the competitive frequency growth and to retain the basic noise immunity and cell stability [88]. Hence the dissimilar area and supply voltage scaling rates result in high power densities and elevated temperatures. The four well known wear-out failure mechanisms namely Time-Dependent Dielectric Breakdown (TDDB), Electromigration, Thermal Cycling and stress migration are all adversely affected by increase in temperature. These age-related hard errors that appear in-field after a certain period of use are major factors impairing the life-time reliability of modern microprocessors [87] used in data centers and space equipment. These applications demand high throughput and better lifetime reliability. However, these wear-out failures limit their useful lifetime and performance.

1.2.4 Interference

Beyond manufacturing defects and wear-out related permanent faults, integrated systems of recent years are more susceptible to temporary effects like transient or intermittent faults. These effects may be due to electromagnetic influences, alpha-particle radiation or cosmic radiations. They are the major portion of digital system malfunctions, and have been found to account for more than 90% of the total maintenance expense [79]. Smaller geometries increase coupling capacitances amount interconnects and current densities and cause problems of crosstalk. In addition threshold for noise sensitivity also shrinks with (V_{dd}) scaling and make new technology

nodes more susceptible to transient faults due to high energy particle from environment or from within the packaging.

1.3 Errors in Integrated Circuits

A fault, when active, is the cause of an error. An error is that part of the system's state that may cause a subsequent failure, and a failure occurs when an error reaches the service interface and alters the service [7]. An error in integrated circuit can be classified according to its temporal characteristics [52], the product life-cycle stage of its induction, its severity etc. In the following subsections however we classify errors on the basis of their underlying faults, as this classification forms the basis of techniques to tolerate them.

1.3.1 Soft Errors

Soft errors are a growing concern of the reliability of circuits and systems fabricated in advanced in Complementary Metal-Oxide-Semiconductor (CMOS) technologies. They are defined as events in which data is corrupted, but the device itself is not permanently damaged [69]. Soft errors are a subset of SEE, which are caused by high energy neutrons from cosmic rays or by alpha particles that are generated from impurities in packaging materials, when such particles strike sensitive region in microelectronic device.

When a particle strikes a microelectronic device, the most sensitive regions are usually reverse-biased p/n junctions. The high field present in a reverse-biased junction depletion region can very efficiently collect the particle-induced charge through drift processes, leading to a transient current at the junction contact. Strikes near a depletion region can also result in significant transient currents as carriers diffuse into the vicinity of the depletion region field where they can be efficiently collected. Even for direct strikes, diffusion plays a role as carriers generated beyond the depletion region can diffuse back toward the junction.

Shortly following the discovery of SEU, researchers at IBM used numerical device simulators to compute the response of reverse-biased p/n junctions to alpha-particle strikes [43, 42]. An important insight gained from these early charge-collection simulations was the existence of a transient disturbance in the junction electrostatic potential, which was termed the "field funnel". Charge generated along the particle track can locally collapse the junction electric field due to the highly conductive nature of the charge track and separation of charge by the depletion region field, as shown in Figure 1.3a [9]. This funneling effect can increase charge collection at the struck node by extending the junction electric field away from the junction and deep into the substrate,

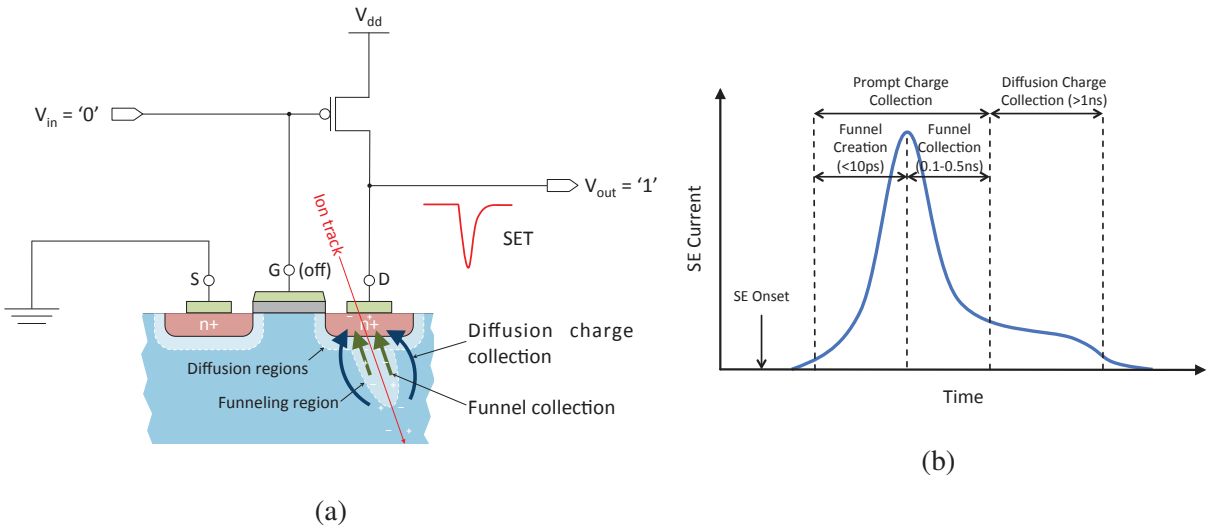


Fig. 1.3: Single Event Effect Mechanism [9]

such that charge deposited some distance from the junction can be collected through the efficient drift process [25] .

1.3.1.1 A brief history and nomenclature

In 1975, the first ever conclusively investigated encounters of soft errors in microelectronic devices, were found to be caused due to high-energy neutrons striking the active (sensitive) regions of memory cells [12]. Historically, soft errors have been of primary concern for memories because of their high active region per unit area than that of CL circuits and the fact that a particle strike in the sensitive nodes of memory cells immediately results in a soft error provided that it infuses sufficient amount of charge to flip the stored value [60], [110] resulting in a SEU, a terminology that was born in the context of memories [111].

However, in the 1980s, few studies addressed another emerging single-event related issue that is, soft errors due to single-events in combinational logic [35], [24], [61]. Hence, the terminology of SEU got a broader context. But the adaptation led to two different interpretations. One circuit-level definition of SEU found in [48], [64], [109] and [36] states that an SEU is a high-energy particle-induced disturbance in a logic circuit node causing a voltage transient that may propagate through the CL stages and eventually be latched by a sequential element [48]. Thus treats SEU as a direct cause of particle strike in CL elements and the voltage transient that propagates through the CL network as its direct effect, which can lead to a soft error. Whereas a system level interpretation that can be found in [25], [86] and [38] considers SEU as soft error in storage elements and defines it as: “Radiation-induced errors in microelectronic circuits caused when

charged particles (usually from the radiation belts or from cosmic rays) lose energy by ionizing the medium through which they pass, leaving behind a wake of electron-hole pairs" [1]. This definition of SEU is in accordance with the one traditionally used in the context of memories and considers SEUs as soft errors in storage elements caused by the voltage transient that propagates through CL from the point of single-event till that storage element.

In order to avoid ambiguity we use the system level interpretation of SEU throughout this manuscript irrespective of the abstraction level at which the issue is being discussed or resolved. We also use the terminology of SET to refer to the single-event induced transient voltage pulses generated in and propagating through CL stages as used unambiguously by many literary works [10, 32, 26].

In the late 1990s a new research interest towards developing techniques to limit SER due to SEE in CL emerged. This drive was in the wake of diminishing impact of the natural redundancies of CL circuits to prevent SET to result in soft errors and the perception that memory soft error situation was controllable with advanced technologies and effective error detection and correction techniques.

Lidén et al. in 1994 experimentally estimated that only 2% of bit flips in memory elements also known as SEU were caused by particle-induced transients or SET generated in and propagated through CL. The rest were due to direct particle strike in latches. Their experiments involved using a 1 μm CMOS process at 5MHz [53]. Since then physical gate-length has downscaled up to 50 times, supply voltages have dropped to 0.9 V and operating frequency has shown a thousand fold increase [46]. This massive change in technology has resulted in greater sensitivity of memory elements to high-energy particle, but the effects are more pronounced on CL networks [86]. A more recent work uses a probability model to estimate that the susceptibility to CL circuits to SET nearly doubles as the technology scales from 45 nm to 16 nm [94].

1.3.1.2 Increasing soft error rate in logic circuits

Early evidences of the problem of SEU in memories as discussed in the previous subsection, gathered prompt attention of the research community. However, in CL the concern of SET emerged much later but the inevitable drive of CMOS device scaling played a significant role in increasing soft errors due to SEE in CL circuits and raised certain new reliability issues. The major reason being the diminishing critical charge of future technology nodes, necessary to generate SET in CL, which is the result of reducing supply voltages levels [86] and error margins. Besides the adverse impact on CL node susceptibility to particle strikes, the technology scaling also lessened the impact of natural barriers posed by CL to SET propagation. There are three inherent properties of CL networks that have prevented SET from resulting into soft errors:

Electrical Masking: The electrical characteristics of the transversed gates may cause the SET pulse duration and amplitude to be attenuated. If a pulse looses strength while propagating through a sensitized path or completely disappear before reaching a memory element, then the SET is referred to be electrically masked [58].

Latching-window Masking: Consider a SET pulse that gets a sensitized path and reaches a Flip-Flops (FF) input with amplitude enough to be considered a valid logic level, but misses the latching-window of the FF. This SET pulse will not affect the stored data due to the latching-window masking effect [37].

Logical Masking: For a SET to propagate through CL and result in a soft error it is necessary that the path from the point of SEE generation to a memory element, should be functionally sensitized during the time of SEE propagation [37]. For example, as incorrect logic value at one input of an OR gate somewhere in the path, does not produce an error at its output if another of its inputs is at logic level 1. This depends on the input vector being applied at the time of the SEE propagation.

As technology scales these factors are diminishing. Electrical masking is supposed to decrease because the smaller transistor are faster thus have less attenuation effect on a SET. High operating frequencies mean that there are more latching-windows per unit time thereby increasing the probability of a SET being latched. Among the three masking effects logical masking seems least affected by the technology trends [86]. As a result research attention drawn towards developing techniques to limit SER in CL is becoming comparable to effort made in protecting state elements.

1.3.2 Hard Errors

Hard errors are caused by permanent silicon defects, which either exist due to manufacturing processes imperfections as discussed in Section 1.2.2 or are caused by in-field wear-out phenomenon as discussed in Section 1.2.3. The rapid development of silicon process has provided steady increase in processor performance and capabilities for the past three decades. However, the growth in the number of transistor per core, increases the chance of having more hard errors in a given core. In addition these high performance microprocessors mostly operate at higher clock frequencies and voltage, thus experience accelerated aging due to temperature and voltage stress [18]. Furthermore, the increasing complexity in connectivity between different stages of high-performance processing cores, to support advanced features (like hazard detection, branch prediction, data forwarding etc) and also having a large number of stages, makes error confinement a challenge [99].

Some well-known failure mechanisms such as time dependent dielectric breakdown (TDDB) in the gate oxides and electromigration (EM) on interconnects have increasing adverse effects due to shrinking feature size. On the other hand, degradation of device parameters over the circuit's lifetime has emerged as a major threat to system reliability. In particular, circuit wearout resulting from negative bias temperature instability (NBTI) and random telegraph noise (RTN) that cause electrical parameter shift (e.g., transistor threshold voltage increase) is of particular concern with technology scaling and it is shown that they could result in significant performance degradation of the circuit over its service life [59].

1.3.3 Timing Errors

Unlike hard and soft errors, components that suffer from timing error still provide correct logic outputs. However, they have higher delays between input and output signal establishments. Faults induced by the drift in the electrical characteristic of circuit elements caused by PVT variability, manufacturing defects and aging are responsible for this type of errors. With the continuous downscaling of transistor feature size, there is an increasing uncertainty for the timing behavior of today's ICs, often manifesting themselves as infrequent timing errors on speed-paths, i.e., critical or near-critical paths [59].

1.4 Reliability Improvement Approaches

In order to achieve reliability goals, reliability practices must begin early in the design process and must be well integrated into the overall product development cycle. Steps must be taken at design time, implementation time as well as during execution time. Similarly, achieving system reliability requires understanding reliability needs at each level of the design. With the increasing complexity of systems interactions, interfaces and stress profiles it is becoming more and more important to understand when, what and where to use and how to create a balance of reliability improvement efforts on each design level and development cycle to meet the overall system reliability targets. Heimerdinger et al. in [40] characterize the reliability improvement practices according to their chronology in the product development and life cycle as follows.

1.4.1 Fault Avoidance

Fault avoidance uses various tools and techniques to specify, design and manufacture systems in such a manner that introduction of faults is minimized [85] by targeting the source mechanisms that cause the failure as shown in Figure 1.4. Use of formal methods to express specification may

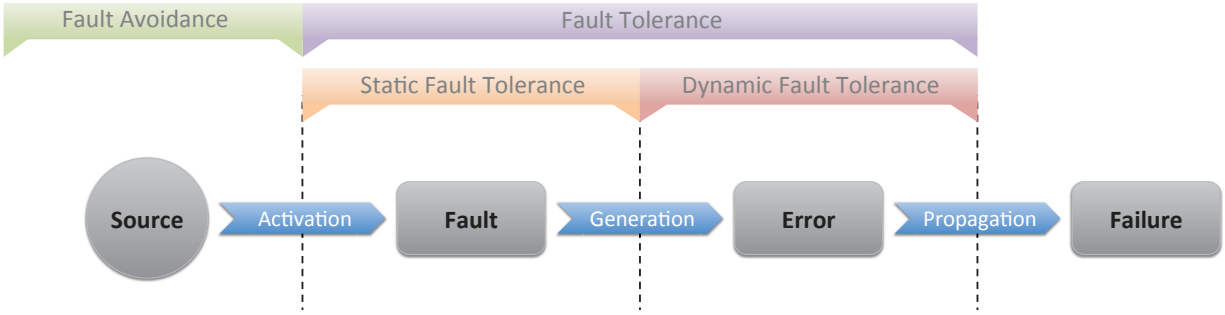


Fig. 1.4: Reliability improvement approaches across fault-failure life cycle [17]

reduce their analogousness and avoid faults during specification phase [78]. A typical example of fault avoidance at design phase is employing transistor resizing within critical gate to decrease the vulnerability to soft errors [109]. Fault avoidance may also include technology mitigation techniques that implicate modifications of conventional manufacturing processes, like modern chips designs use Silicon-On-Insulator (SOI) technology that provides many advantages including significantly reduced susceptibility to soft errors due to the smaller volume for charge collection [39]. Besides these, used of radiation hardened components is another example of fault avoidance at design phase. Whereas fault avoidance during manufacturing involves adaptation of suitable standards of quality, for instance, ensuring the cleanroom standards.

1.4.2 Fault Removal

Fault removal refers to a wide spectrum of approaches aimed at detecting and eliminating existing faults during specification and design, and remove faulty components during production and operational phases. Fault removal uses various methods including formal verification, design rule checking, signal integrity analysis, static timing analysis etc. during sign-off to locate faults in specification or design enabling the necessary changes to be made before tape-out. Burn-in is one of the fault removal techniques that weeds out defective chips after manufacturing time, so that chips actually used in systems have very low failure rate. Chips also typically include Design For Test (DFT) structures such as scan chains, online and offline tests etc to aid in fault removal during their operational life [85].

1.4.3 Fault Tolerance

In spite of the best efforts to avoid or remove them, there are bound to be faults in any operational system. Once a fault has been generated it can be prevented from activating an error using static

fault tolerant techniques (refer Figure 1.4) such as masking. Alternately, errors can be detected and recovered using dynamic fault tolerance techniques like Error-Correction Code (ECC) [17]. Fault tolerance aims at guaranteeing the service provided by the product despite the presence or appearance of faults [40].

There are various approaches to achieve fault tolerance in a system, but what they have in common is the use of certain amount of redundancy. John von Neumann in 1950's pioneered the idea of using redundancy to improve the reliability of systems in his work titled as "Probabilistic logic and synthesis of reliable organisms from unreliable components" [68]. According to the classification of redundancy done by Avižienis in [6], there are two basic types; spacial and temporal redundancies.

Spacial Redundancy: Refers to redundant components, functions or data units used either to verify original execution or to replace the defective or erroneous ones [27]. Spacial redundancy is further classified into hardware, software and information redundancy, depending on the type of redundant resources added to the system.

Temporal Redundancy: In temporal redundancy a computation or data transmission is repeated to compare with the original one [27]. Spacial redundancy impacts the system size and power in terms of area for extra hardware resources to hold and execute extra software and information. In some application it may be desirable to spend extra time rather than extra space to tolerate faults. Therefore, temporal redundancy generally has overhead in terms of performance in order recompute data using the same hardware resources.

Hybrid Redundancy: Hybrid redundancy approaches combine the spacial and temporal redundancies to optimize the use of redundant resources. Faults appear in system from time to time, the use of temporal resources to detect errors could be of an adverse impact on the throughput of the system. Similarly occupying hardware resources for redundant computations can have an overhead on power consumption. Hybrid redundancy makes efficient use of spacial and temporal resources to optimize the impact on area, power and performance. A simple example of fault tolerant architecture that uses hybrid redundancy is given in Section 2.2.1.

1.4.4 Fault Evasion

It is possible to observe the behavior of a system and use this information to take action to compensate for faults before they occur [40]. During the operational life of a product situations like perturbations coming from the non-operational environment can induce faults. To avoid such

situations external environment is monitored to detect these dangerous conditions and adequate shielding action are taken to protect faults from appearing. An example is the use of on-chip temperature sensors for power management to avoid wearout related faults [15, 31].

1.5 Research Objectives

The work in this thesis aims to improve the transient, permanent and timing error reliability of future technology circuits and modern microprocessor systems for their use in high-reliability applications. It places a special emphasis on the issue of error detection and confinement in complex pipeline circuits and addresses the reliability concerns arising from the Combinational Logic (CL) parts of logic circuits, a problem illustrated symbolically with the help of Figure 1.5. It shows the share and types of problems arising from sequential logic and CL parts of digital circuit. As discussed in Section 1.3.1.2, CL networks are becoming increasingly susceptible to SEEs. In addition, pronounced variability and power densities either causes the electrical characteristics of these node to change, resulting in timing errors or become permanently damaged giving rise to hard errors. As a result, the research attention drawn towards developing techniques to limit SER in CL is becoming comparable to effort made in protecting state elements.

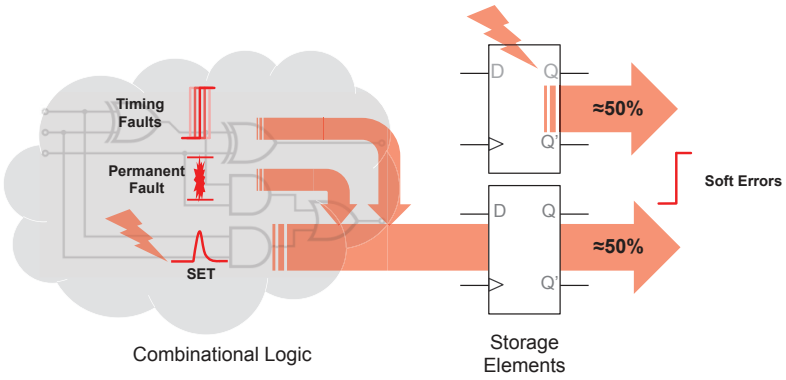


Fig. 1.5: Error occurrences in combinational logic and storage elements

- As the first step towards achieving this global objective, the thesis aims to perform an in-depth analysis of spacial, temporal and hybrid redundancy approached existing in the state-of-the-art and classifying them with respect to their area, power and performance overheads, and fault tolerance capabilities.
- It aims to develop an effective hybrid fault tolerant approach that can be applied to complex pipeline circuits.

- To compare and identify the shortcomings of the developed technique by its application and the application of state-of-the-art techniques to a specific core architecture.
- It also aims to lay down the framework for the quantitative comparative analysis of different fault tolerant architectures specially in terms of their fault tolerance capability.
- As a final objective, It intends to optimize the developed technique for cost-reliability and provide the means to selectively use the technique to be able to control the cost-reliability trade-off.

Chapter 2

Fault Tolerant Architectures and Assessment Techniques

In order to prevent reliability from becoming a bottleneck for the development of high-performance, low-power systems, design architects must address the concern of reliability through the use of fault-tolerant architectures. These architectures are commonly used to tolerate on-line faults, i.e. faults that appear during the normal functioning of the system, irrespective of their transient or permanent nature [50]. They use redundancy to tolerate faults in Combinational Logic (CL) and storage elements. These techniques as generally classified by the employed type of redundancy discussed in Section 1.4.3.

In this chapter we will first discuss some error detection and correction techniques generally employed in fault-tolerant architectures. In the second section of this chapter we will discuss some relevant fault-tolerant architectures. Among these architectures we will discuss a Hybrid Fault Tolerant (HyFT) architecture in detail because it serves as the starting point of the work in this thesis. In Section 2.3 we will briefly overview the robustness assessment techniques with special focus on a simulation based fault injection framework developed and used throughout the course of this thesis for fault tolerance capability assessment of different fault tolerant architectures. Later we will present an experimental study to compare different architectures on the basis of their implementation cost, performance and most importantly the fault tolerance capability and finally in the last section we will summarize the results of the experimental study and conclude the discussion.

2.1 Fault-Tolerance Techniques

Circuit-level fault-tolerance techniques are generally based on any one or a group of redundancies discussed in Section 1.4.3. These techniques change the original circuit by adding logic for error detection and correction or error masking. At system-level (for example microprocessor, System on Chip (SoC) etc.) these circuit-level fault tolerance techniques are generally categorized as hardware-based techniques because they mostly rely on hardware replication and additional hardware modules to protect system against faults. Moreover software-based fault-tolerance techniques are also widely implemented at system-level because of their non-intrusiveness, high flexibility, low development time and cost [50]. However, software-based fault tolerance techniques cannot achieve full system protection because of their inability to handle all the possible control flow errors. In this section we focus on some circuit-level fault-tolerance techniques commonly used as building block of fault-tolerant architectures.

2.1.1 Concurrent Error Detection

Exhaustive testing has been long ruled out because of the increasing complexity of chips. Thus most but not all manufacturing defects can be detected during *testing*. The manufacturing faults that escape testing can appear infield anytime. Similarly, the moment of transient fault occurrence are not predictable. Also the failures caused due to aging effects on digital circuits cannot be precisely forcasted. Thus the errors caused by these faults have to be detected during normal operation. Error detection during normal operation is called *concurrent error detection* or *on-line detection* [38].

2.1.1.1 Duplication with Comparison

Duplication with comparison is a commonly used, simple to implement error detection technique based on hardware redundancy. It uses two identical copies of a circuit to compare their outputs and flags error if there is an inequality in the results computed by the two copies as shown in Figure 2.1. One of the reasons of its popularity is its simple implementation, and also its ability to detect a wide variety of faults, which include permanent, transient and timing faults. An important design decision for schemes that use duplication with comparison is the placement of the comparator.

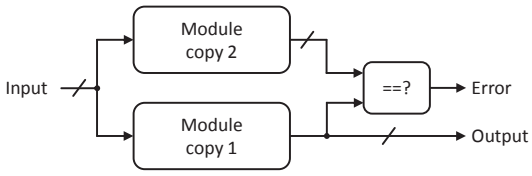


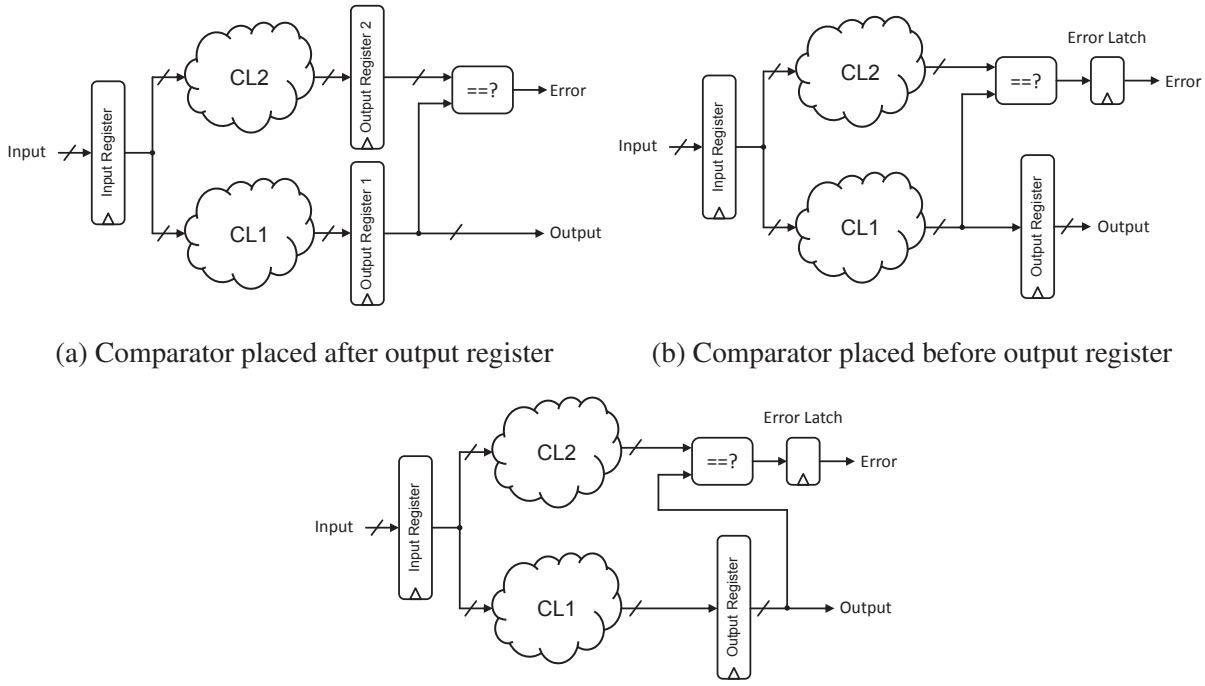
Fig. 2.1: Duplication with Comparison

Theoretically, any sequential circuit can be transformed into a pipelined circuit by grouping together all the FFs with same sequential depth[1] as individual pipeline registers. Although the resulting pipeline may contain a large number of feedback and feed-forward signals but essentially functions as a pipelined circuit [57]. Consider a sequential circuit with FFs having sequential depth not more than 2. Such a sequential circuit can be transformed into a single stage of a pipeline, by grouping all the FFs with sequential depth of 1 as input register and all the FFs with sequential depth of 2 as output register. If this circuit is to be added with a duplication with comparison based error detection capability, after duplicating the CL blocks, a common way of placing comparator is to insert it after the pipeline register as shown in Figure 2.2a. The block labeled as '==?' represents a static comparator and the clouds represent CL blocks.

Any discrepancy in CL1 of CL2 will cause erroneous data to be latched in one of the register and the comparator will signal error for at least one cycle without a need of latching the error signal. With this placement not only the errors latched in the output registers due to Single Event Transient (SET)s caused in the CL blocks are detected, but also the Single Event Upset (SEU)s due to direct high-energy particle strikes in the output register FFs are detected and indicated by a stable error signal. This scheme can also detect timing unless faults in both the CL copies manifest themselves the same way at the outputs of CL. Furthermore, the power dissipation in the comparator's OR-tree will be low because it will undergo at most one transition per input per cycle. However this technique requires duplication of not only the CL but also of the output register.

A commonly used comparator placement is to insert it before the pipeline register as shown in Figure 2.2b. Although this scheme does not require the duplication of pipeline register, it has a few inherent drawbacks. Since the comparator compares during the entire cycle, the time when the CL outputs are unstable due to the difference in circuit path lengths, the OR-tree experiences a increased switching activity thus higher power consumption.

[1] A flip-flop has a sequential dept of d_{seq} if its output is dependent on primary inputs and at least one flip-flop of depth d_{seq}-*1*. For example, if the output of a FF can be controlled by only primary inputs (and a clock pulse) it has sequential depth of 1



(a) Comparator placed after output register

(b) Comparator placed before output register

(c) Comparator placed across output register

Fig. 2.2: Duplication with Comparison in sequential logic circuits

Another possibility of comparator placement is to insert it across the output register as shown in Figure 2.2c. This scheme is similar to the architecture shown in Figure 2.2b except that, since in this case the comparator gets one synchronous input from the output register, half of the comparator inputs do not experience high activity during the CL outputs are unstable. Secondly, as the comparator gets to compare the output of the register is also marginally protects it against SEUs, which occur due to direct particle strikes in it.

In both the cases of Figure 2.2b and 2.2c the comparator output is an unstable signal which perturbs during the time CL outputs are unstable and needs a latching mechanism if a stable error signal is required. This in turn raises another question of, when to latch the error signal? If the error signal is latched with the clock edge, i.e. at the same time as data is latched in the register, a transient glitch or a delayed transition due to a timing fault may get latched in the output register but can possibly escape getting latched in error FF. These glitch escapes can happen because of the additional path it has to traverse in the coparator OR-tree before it can reach the error FF or can get electrically masked by the logic in OR-tree.

One solution to this problem is to move the error latching mechanism upstream in the OR-tree. This will not only reduce the difference in the lengths of paths a glitch or a delayed transition has to travel before reaching the FFs in the output register and the error latching mechanism but

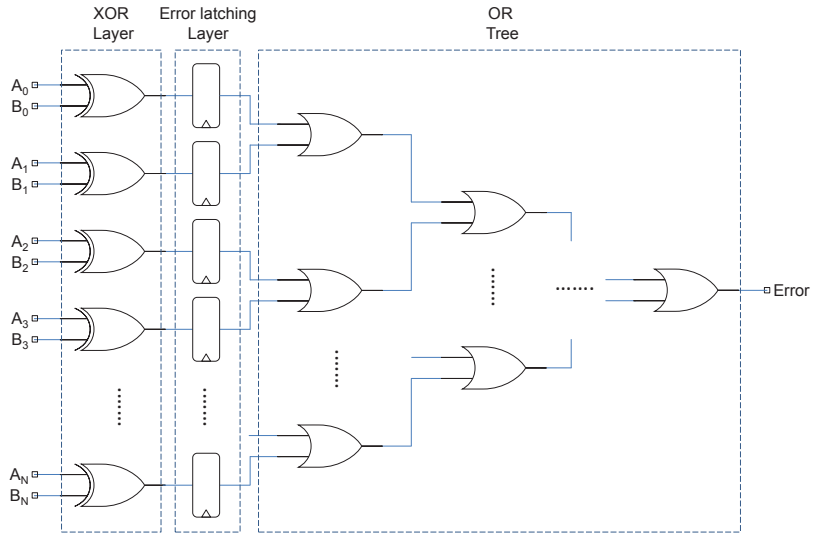


Fig. 2.3: An static comparator with Error Latching mechanism moved upstream

also blocks the unwanted signal perturbations from causing excessive power dissipation in the OR-tree. But even if the error latching mechanism is moved all the way up till the XOR gates as shown in Figure 2.3, the architecture will not meet the power consumption of the first scheme (of Figure 2.2a) because the XOR gates will still experience the unwanted activity and the increased number of FFs to latch error cause higher power consumption due to constant activity of clock network. Alternatively, a latching mechanism at the end of the OR-tree can be used with a delayed clock to latch the error. This clock offset should match the delay of the OR-tree to prevent any glitches from escaping detection. But due to process variations it is difficult to match and could not guarantee zero glitch escapes. On top of that the switching activity in the OR-tree will still be there.

To address some of the problems due to static comparator used in duplication with comparison, the work in [93] proposes a circuit-level implementation of a special comparator discussed in the following subsection.

Pseudo-dynamic Comparator: The main idea is to use an error latching mechanism upstream in the OR-tree. Secondly, instead of depending on a capture edge to latch the error signal it uses a user defined *comparison-window* for detection of permanent, transient and timing errors. Furthermore it reduces the power consumption in the error latching layer by the use of DOR gates shown in Figure 2.4.

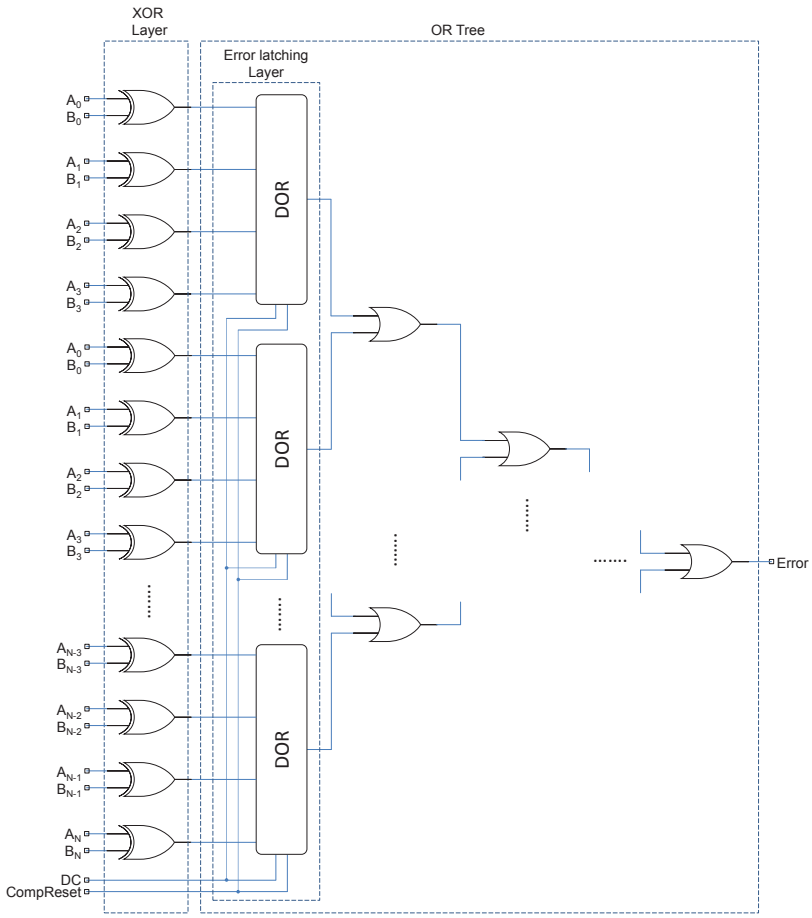


Fig. 2.4: Pseudo-dynamic Comparator [93]

Figure 2.5 shows the structure of the 4-input DOR gate. The *dynamic logic* part, which is controlled by *CompReset* and *DC* signals forms an OR gate and the *keeper* acts as an error latching mechanism. During the pre-charge phase, reset is at logic-0. Input capacitance of the *inverter* is pre-charged to V_{DD}, which puts the *Error* signal at logic-0. During the evaluation phase *CompReset* and *DC* are both at V_{DD}. If atleast one of the inputs (C_i) turns to logic-1, the discharge will occur. Consequently, *Error* switches to logic-1. Note that once the discharge happens, *Error* will remain at logic-1 until the next pre-charge phase. Besides, the keeper must be weak enough so that its input can be pulled down to logic-0 by the driving transistors.

Figure 2.6 present the complete error detection architecture using the pseudo-dynamic comparator represented by the block labeled as '==?*'. It places the pseudo-dynamic comparator across the output register in order to protect it against SEUs that are generated in it. The *CompReset* and *DC* and the control logic that generates them is not show for clarity. Experimental results in [93] show that the new comparator exhibits better glitch and delayed transition detection

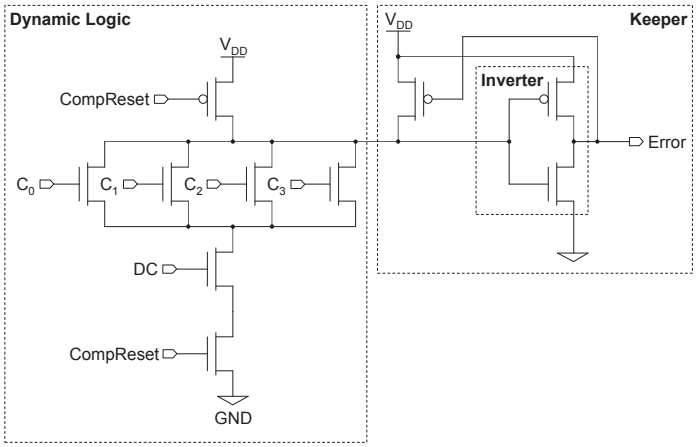


Fig. 2.5: 4-input DOR gate [93]

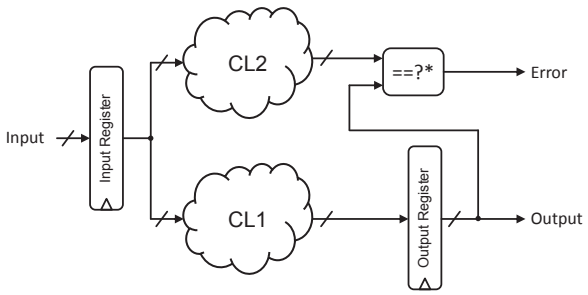


Fig. 2.6: Error detection architecture using the pseudo-dynamic comparator [93]

capability due to its error latching layer moved upstream in the OR-tree and with the use of dynamic logic it offers about 30% reduction in power compared with a static comparator, while having negligible area overhead in comparison with a static comparator.

2.1.1.2 Error Detecting Codes

Another widely used techniques of error detection in digital circuits is the use of error detecting codes. Error detecting codes introduce redundancy in information representation to detect possible errors in that representation [54]. Consider a logic circuit that performs a function f on the n-bit input data i and produces a m-bit output $f(i)$ as shown in Figure 2.7. Using error detecting codes to improve the data integrity of this logic block generally involves implementing a block which predicts some special characteristic $C(i)$ of the output $f(i)$ for every input sequence i. A checker circuit first computes this characteristic $C'(f(i))$ of output from the output itself and finally compares it with the predicted characteristic $C(i)$ to check for any data anomalies.

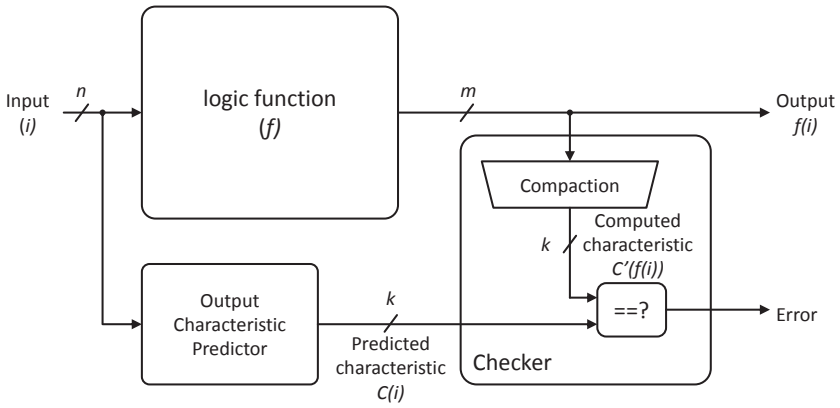


Fig. 2.7: General architecture of Error Detection with Codes [63]

Error detecting codes have been widely used to protect memories against SEUs and permanent faults [74]. The main reason of their widespread use in memories is that their regular structure allows efficient incorporation [28]. Use of error detecting codes to detect errors in logic circuit has been preliminarily based on the idea of reducing the implementation overhead of duplication with comparison. However reducing area overhead below the cost of duplication for random logic require redesigning the original circuits or compromising on fault coverage [38].

2.1.2 Error Recovery

Following an error detection an error recovery mechanism restores the error-free state of the system or prevents faults from being activated again or both [54]. Error recovery mechanism generally take two forms discussed in following subsections.

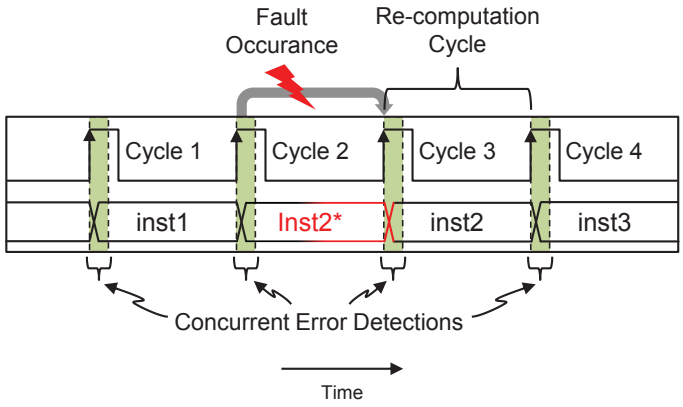
2.1.2.1 Rollback Error Recovery

In rollback recovery scheme errors are recovered by repeating the operation(s). Architectures based on rollback recovery generally use spatial redundancy to detect errors and correct them with temporal redundancy. Hardware and software use check-pointing to periodically or occasionally save system state, which is used to bring the system back to an error free state in case of error detection. These rollbacks can be as long as upto several thousands of cycles [62] or can be just one cycle deep [92].

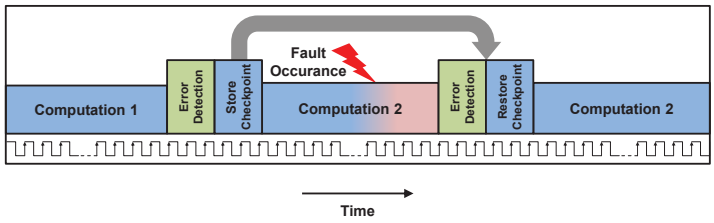
An example of a single cycle deep rollback recovery scheme is shown in Figure 2.8a in which a concurrent error detection mechanism checks for errors at the end of each cycle and if it detects an error the instruction re-executes during the next cycle. Figure 2.8b shows an example of a much deeper rollback recovery scheme. After each computation, which takes several cycles an

error detection mechanism checks for errors. If no error is detected, system state (which can include the contents of pipeline register, control registers and memory update information etc.) is stored as checkpoint data. In case an error is detected the previous saved checkpoint data is used to restore system error-free state and the effected computation is repeated.

A fine-grained rollback recovery schemes demands frequent error checks and checkpoint updates, thus incur high energy cost. However the amount of check-pointing data to be stored is minimal and are generally less intrusive. Coarse-grained rollback recovery schemes on the other hand generally rely on software intrusive recovery sequences and check-pointing procedures, and demand significant storage to store check-pointing data specially if memory updates are involved. Besides implementation cost, coarse-grained recovery schemes can have significant performance overhead particularly in high error rate environments [107].



(a) Fine-grained rollback recovery scheme



(b) Coarse-grained rollback recovery scheme

Fig. 2.8: Examples of Rollback recovery schemes

2.1.2.2 Forward Error Recovery

In forward error recovery techniques whenever an error is detected, the correction is made ahead of time such that corrected data is available at or before the time when it is normally needed. Error

masking is a typical example of forward error recovery scheme, where the recovery mechanism act as soon as an error is detected unlike rollback recovery in which the system steps back in time to recompute. Forward error recovery schemes are efficient to be used in tight deadline applications as they do not rely on re-computations [90].

Forward error recovery schemes generally uses spacial redundancy for both error detection and correction. Significant amount of hardware resource is dedicated all the time whether there are errors or not. On the other hand, rollback recovery schemes make use of temporal resources only when they are needed thus have better allocation of redundant resources [66].

2.2 Fault tolerant Architectures

Several fault-tolerant architectures have been proposed in the literature in the past to address the circuit reliability concerns. A few of these relevant solutions include; Partial-TMR, Full-TMR, DARA-TMR [108, 107], PaS [47], CPipe [89], STEM [5] and Razor [29], which are discussed in following subsections. We select this set of architectures because it includes a representative of each class of the broad spectrum of fault tolerant architectures available in the literature.

2.2.1 Pair-and-A-Spare

Pair-and-A-Spare (PaS) Redundancy was first introduced in [47] as an approach that combines Duplication with Comparison and standby-sparing. Figure 2.9 illustrated the principle of PaS redundancy. In this scheme each module copy is coupled with a Fault-Detection (FD) unit to detect hardware anomalies within the scope of individual module. A comparator is used to detect inequalities in the results from two active modules. In the case of inequality a switch decides which one of the two active modules is faulty by analyzing the reports from FD units and replaces it with a spare one [27]. This scheme was intended to prevent hardware faults from causing system failures. The scheme fundamentally lacks protection against transient faults and it incurs a large hardware overhead to accomplish the identification of faulty modules.

2.2.2 Razor

Razor is a well known solution to achieve timing error resilience by using the technique called *timing speculation* presented in [29]. The principle idea behind this architecture is to employ temporally separated double-sampling of input data using Razor FFs, shown in Figure 2.10a, placed on critical paths. The main FF takes input sample on the rising edge of the clock, while a time-shifted clock (*clk-del*) to the shadow latch is used to take a second sample. By comparing the

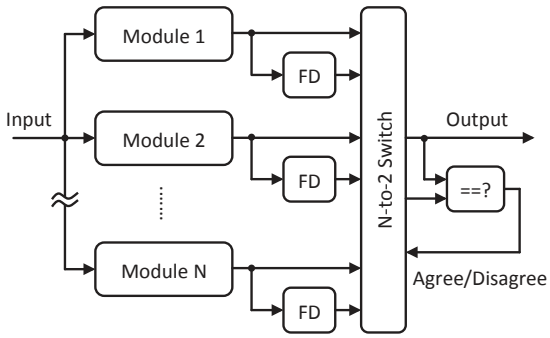
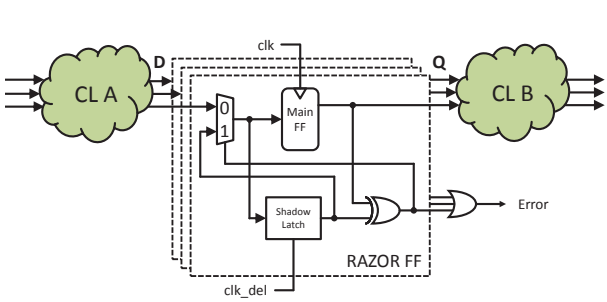
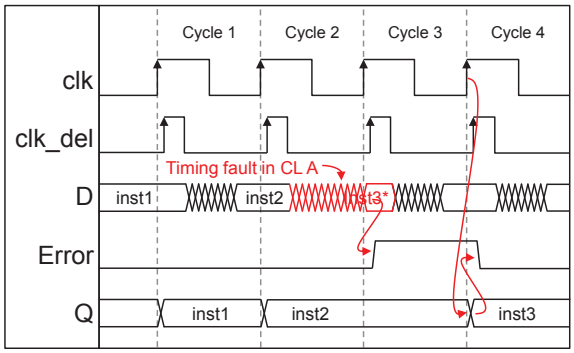


Fig. 2.9: Principle of PaS redundancy



(a) RAZOR FF



(b) Example timing diagram of error correction

Fig. 2.10: RAZOR Architecture [29]

data of the main FF and the shadow latch, an error signal is generated. The timing diagram of how the architecture detects timing errors is shown in Figure 2.10b. In this example a timing fault in CL A causes the data to arrive late enough that the main FF captures wrong data but the shadow always latches the input data correctly. The error is signaled by the XOR gate which propagates through the OR-tree for correction action to be taken. Error recovery in Razor is possible either by clock-gating or by rollback recovery. Razor also uses Dynamic Voltage Scaling (DVS) scheme to optimize the energy vs. error rate trade-off.

2.2.3 STEM

STEM cell architecture takes Razor a step further by incorporating capability to deal with transient faults as well. STEM cell architecture presented in [5] incorporates power saving and performance enhancement mechanisms like Dynamic Frequency Scaling (DFS) to operate circuits beyond their

worst-case limits. Similar to Razor FFs, a STEM cells (shown in Figure 2.11) replace the FF on the circuit critical paths, but instead of taking two temporally separated samples, A STEM cell takes three samples using two delayed clocks. Mismatches are detected by the comparators and the error signals is used to select a sample which is most likely to be correct for rollback.

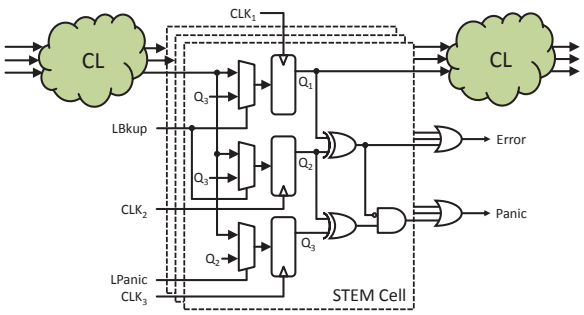


Fig. 2.11: STEM Architecture [5]

2.2.4 CPipe

The CPipe or Conjoined Pipeline architecture proposed in [89] uses spatial and temporal redundancy to detect and recover from timing and transient errors. It duplicates CL blocks and the FFs as well to from two pipelines interlinked together as shown in Figure 2.12. The primary or *leading* pipeline is overclocked to speedup execution while the replicated of *shadow* pipeline has sufficient speed margin to be free from timing errors. Comparators placed across the *leading* pipeline register in somewhat similar way as the scheme of Figure 2.2c, detects any metastable state of *leading* pipeline register and SETs reaching the registers during the latching window. The error recovery is achieved by stalling the pipelines and using data from the *shadow* pipeline registers for rollback and it takes 3 cycles to complete.

2.2.5 TMR

TMR is one of the most popular fault tolerant architectures. One of the very first use of TMR in computing systems can be found in [55]. In a basic TMR scheme called Partial-TMR and shown in Figure 2.13a, we have three implementation of same logic function and their outputs are voted by a voter circuit. This architecture can tolerate all the single-faults occurring in the CL block but faults in voter or pipeline registers cause the system to fail. Full-TMR on the other hand, triplicates the entire circuit including the FFs and can tolerate all single-faults in any part

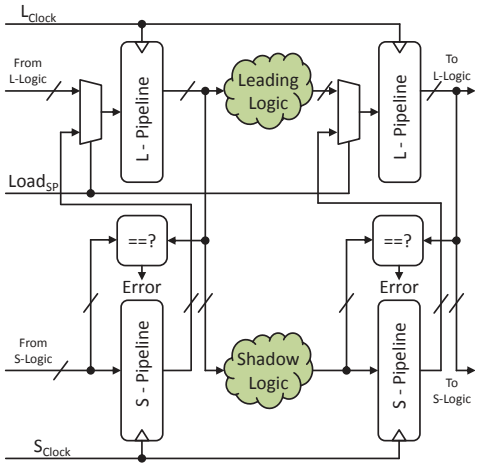
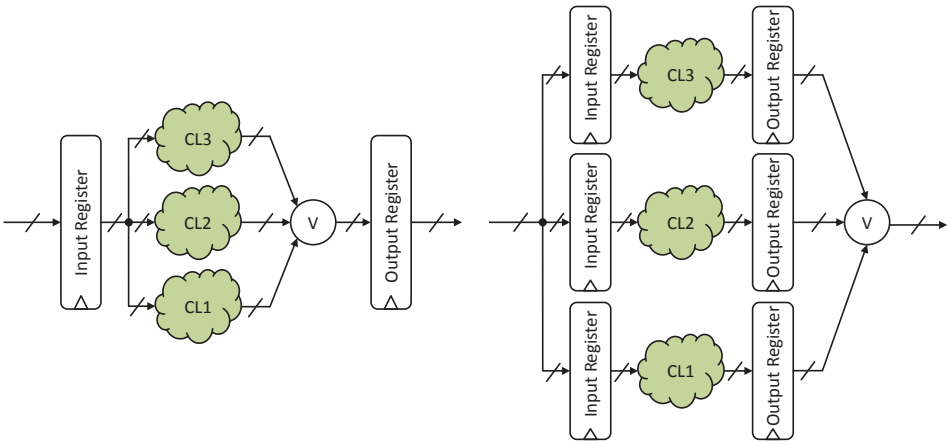


Fig. 2.12: CPipe Architecture [89]

of the circuit except voter and the signals to the input pipeline register, which may result in common-mode failure.



(a) Partial-TMR (b) Full-TMR

Fig. 2.13: Triple Modular Redundancy

2.2.6 DARA-TMR

DARA-TMR triplicates entire pipeline but uses only two pipeline copies to run identical process threads in Dual Modular Redundancy (DMR) mode. The third pipeline copy is disabled using power gating and is only engaged for diagnosis purposes in case of very frequent errors reported by the detection circuitry. Figure 2.14 shows a simplified representation of DARA-TMR scheme. Once the defective pipeline is identified the system returns back to DMR redundancy mode

by putting the defected pipeline in off mode. The error recovery follows the same mechanism as pipeline branch misprediction, making use of architectural components for error recovery. DARA-TMR treats permanent fault occurrence as a very rare phenomenon and undergo a lengthy reconfiguration mechanism to isolate them [108].

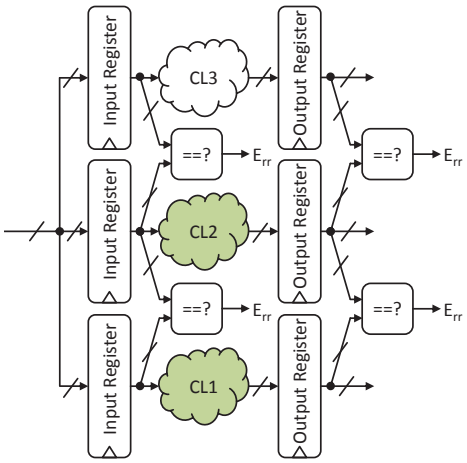
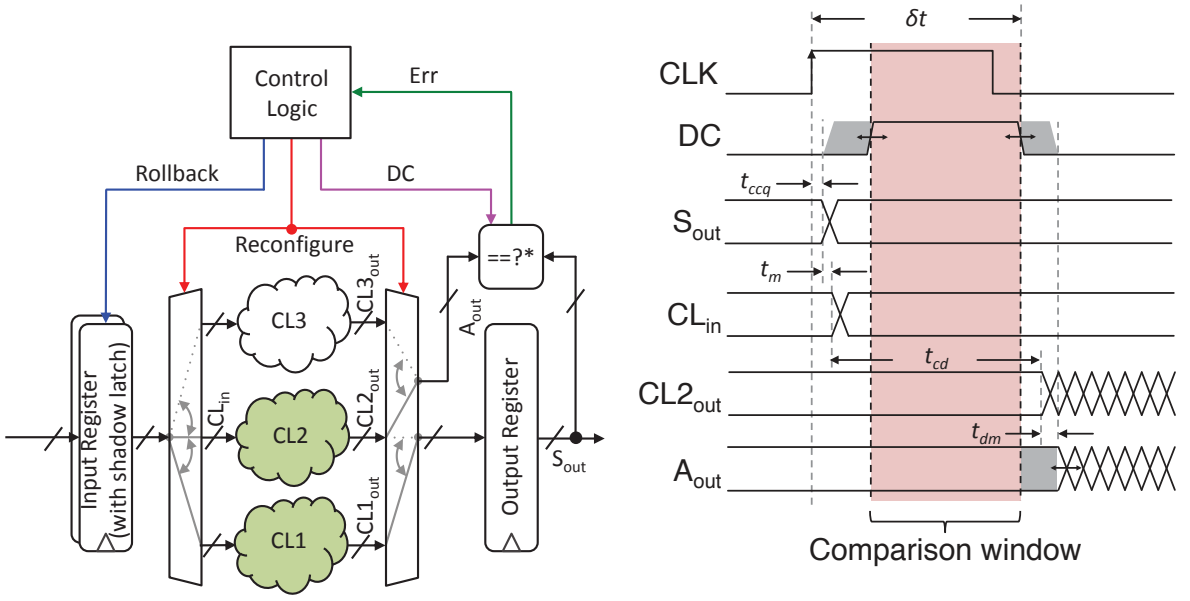


Fig. 2.14: DARA-TMR [108]

2.2.7 Hybrid Fault-Tolerant Architecture

HyFT architecture employs information redundancy (as duplication with comparison) for error detection , timing redundancy (in the form of re-computation/rollback) for transient and timing error correction and hardware redundancy (to support reconfiguration) for permanent error correction [102]. As shown in the simplified representation in Figure 2.15a, the HyFT architecture employs triplication of CL blocks. A set of multiplexer and demultiplexer is used to select two primary CL copies and to put the third CL copy in standby mode during normal operation. HyFT architecture is driven by a control logic module that generates the necessary control signals. The details of the HyFT control logic is provided in Appendix A. HyFT architecture uses the pseudo-dynamic comparator [93] for error detection to achieve better glitch detection capability and to reduce the power consumption as discussed in Section 2.1.1.1.

The HyFT architecture uses a concurrent error detection mechanism. A pseudo-dynamic comparator compares the outputs of two active CL copies. It can be seen in Figure 2.15a that the comparator is placed across the output register such that it gets to compare the output of the output register S_{out}, which is a synchronous signal with the output of the secondary running copy A_{out}, which is an asynchronous signal. As discussed in Section 2.1.1.1, this orientation of the pseudo-dynamic comparator also offers marginal protection against the errors due to faults in the



(a) Hybrid Fault Tolerant Architecture

(b) Timing constraints imposed on CL synthesis by pseudo-dynamic comparator

Fig. 2.15: HyFT Architecture and associated timing constraints

output register and allows it to remain off the critical path. Thus, it does not impact the temporal performance of the circuit [103, 95].

The error recovery scheme uses stage-level granularity reconfigurations and single-cycle deep rollbacks. The shadow latches incorporated in pipeline registers keep one clock cycle preceding state of the pipeline register FFs. The comparison takes place after every clock cycle. Thus, error detection can invoke a reconfiguration and a rollback cycle, confining the error and preventing it from effecting the computation in the following cycles.

The comparison takes place only during brief intervals of time referred to as *comparison-window*, represented in Figure 2.15b as regions shaded in red color. The timing of *comparison-window* is defined by the high phase of a delayed clock signal *DC*, which is generated from *CLK* using a delay element. These brief comparisons allow keeping the switching activity in OR-tree of the comparator to a minimum, offering a 30% power reduction compared with a static comparator. The functioning of the pseudo-dynamic comparator requires specific timing constraints to be applied during synthesis of CL blocks, as defined below.

Timing Constraints: In typical pipeline circuits the contamination delay of CL should respect hold-time of the pipeline register latches. However in the HyFT architecture, as CL also feeds to the pseudo-dynamic comparator, CL outputs need to remain stable during the comparison. And

since the comparison takes place just after a clock edge, any short paths in the CL can cause the input signals of the comparator to start changing before the lapse of the comparison-window. Thus the CL copies have to be synthesized with minimum delay constraints governed by:

$$t_{\mathrm{cd}} \geqslant \delta t - t_{\mathrm{ccq}} - t_{\mathrm{cdm}} - t_{\mathrm{cm}} \tag{2.1}$$

where:

t_{cd} = CL contamination delay
δt = the amount of time between *CLK* capture edge and the lapse of the *comparison-window* (as shown in Figure 2.15b)
t_{ccq} = FF clk-to-output delay
t_{cdm} = demultiplexer contamination delay
t_{cm} = multiplexer contamination delay

In Figure 2.15b, with the help of a timing diagram we explain the associated timing constraints. Besides *CLK* and *DC* the timing diagram shows the signals at the two inputs of the comparator labeled as A_{out} and S_{out} also indicated in Figure 2.15a. The remaining two signals are the inputs of CL labeled as CL_{in} and the outputs of CL labeled as CL_{out}. The grey shaded regions in Figure 2.15b represent the allowed margins of the corresponding signals. The timing allowance for the start of the *comparison-window* depends on the clk-to-output delay of the output register. This implies that the comparison should not begin until the output of the output register stabilizes. Whereas the end of the comparison window should not extend beyond the point when the outputs of CL block start to change. Conversely, the inputs of the comparator should not begin to change before the lapse of *comparison-window*.

In Figure 2.16 we explain the error detection and correction principle of HyFT Architecture shown in Figure 2.15a, with the help of three different fault scenarios among many other possible cases. Figure 2.16a is a timing diagram of system response to an occurrence of a permanent fault, Figure 2.16b shows the response in case of a SET occurrence and Figure 2.16c is in case of a timing fault. It can be noticed that either A_{out} or S_{out} can be affected by an error depending on the fault location. In both the cases the comparator detects the inequality and flags *Err* signal. In Figure 2.16, this is represented by a comparison window marked with an inequality ($\neq$) sign, which compares a correct result with a faulty one represented by an asterisk (*) sign. The *Err* signal remains active for two cycles until the system returns back to normal operation after a reconfiguration and a recompilation cycle.

Besides these three, many other fault scenarios are possible. For instance in case of permanent or timing fault it is possible that the system may not be able to isolate the fault by undergoing just one reconfiguration and a recompilation cycle. Since it is not possible to determine which of

the two working copies of CL in a stage exhibits a permanent fault, the reconfiguration choice is irrespective of that. There is 50% chance that the first reconfiguration eliminates the fault. In this case another error detection triggers a pair of reconfiguration and re-computation cycles, which will definitely be able to select two CL copies that provide good results. Thus the error recovery penalty in this case will be four cycles instead of two.

2.3 Robustness Assessment Techniques

Unlike area/power overheads and temporal performance degradation of any fault tolerant architecture, robustness improvement and error recovery overhead are two important design merits, which cannot be assessed using standard circuit analysis methods and tools. The methods to assess these design merits can generaly be divided into two groups.

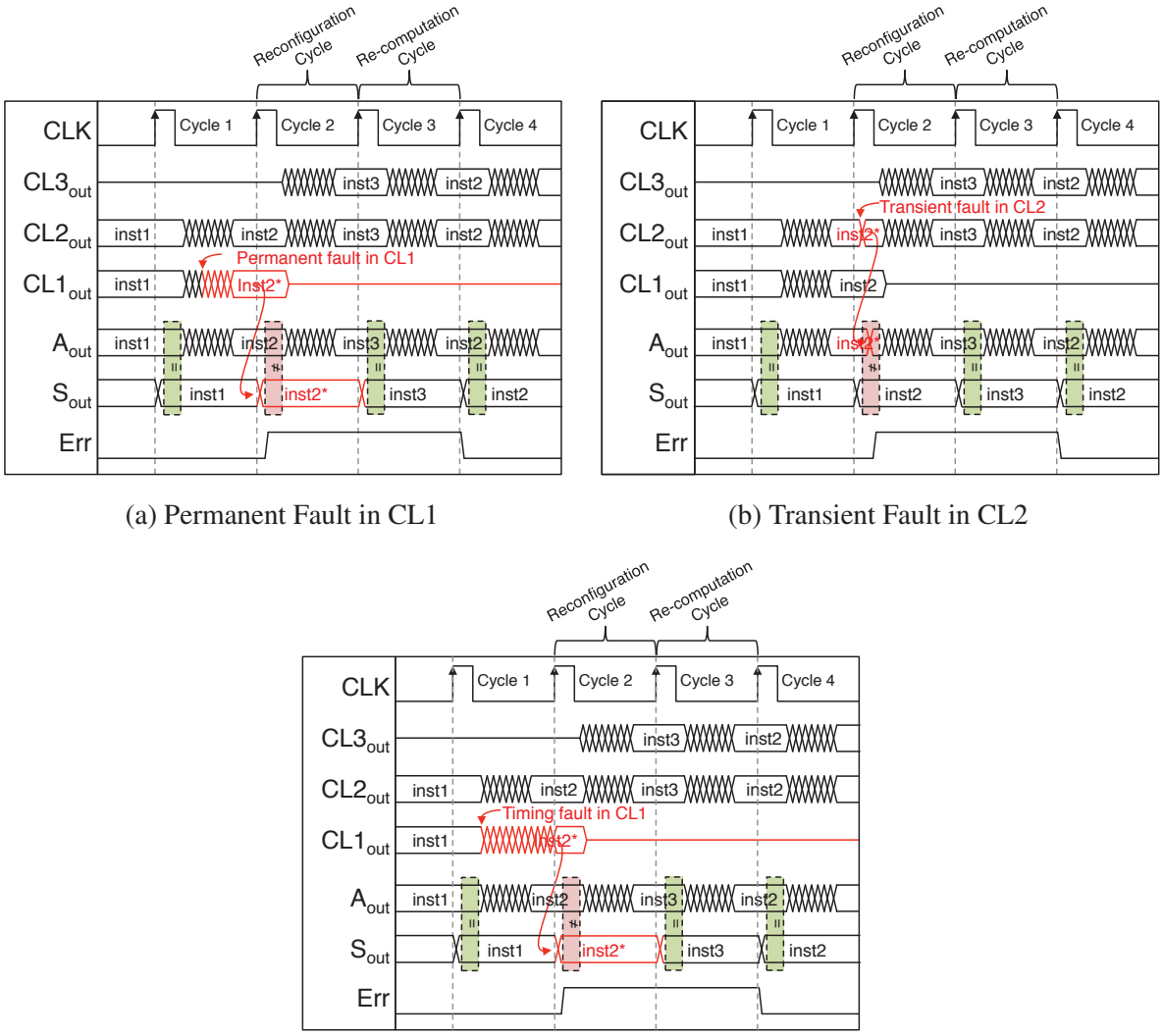
2.3.1 Axiomatic Methods

2.3.1.1 Analytic Methods

Analytic assessment of robustness requires abstract mathematical or logical models to specify the behavior of the design and the fault environment. It is then necessary to determine how well the fault tolerant mechanism works by producing analytic solution of the models [3]. Accurate analytic modeling of complex systems is extremely difficult since the models can get very large in practice, and the simplifying assumptions made in order to make analysis tractable reduces the usefulness of the results.

2.3.1.2 Simulation based

Simulation emerges as a reasonable solution to the problem of tractability versus usefulness in analytic methods of robustness assessment. It involves preparing a stochastic model of the fault tolerant system and the environment. Then simulating it over some relevant period of time and using the data gathered to characterize the model's behavior. Simulations allows speeding up fault occurrences to analyze the system response in terms of fault-tolerance capability and performance. Simulation-based fault injection environments need less time and effort to implement and offer better controllability and observability.



(a) Permanent Fault in CL1

(b) Transient Fault in CL2

(c) Timing Fault in CL1

Fig. 2.16: Error detection and correction in HyFT Architecture

2.3.1.3 Verification based

Formal verification techniques have been used in industry not only to ensure the correctness of hardware designs, but also to identify inherent robustness shortcomings in fault tolerant architectures for digital circuits. One of the earliest use of automated formal reasoning to verify fault tolerance of digital circuits in literature dates back to 1986 in [19], where Petri nets were used to validate the fault tolerance of a processor architecture. This work defines formal verification as "A vehicle for hierarchically structuring the verification process so that only few claims need to be proven and only a controllable amount of critical assumptions need to be generated". A modern interpretation and use of formal verification in robustness analysis of digital circuits can be found in [33]. This work addresses the problem of large state space and longer observation time needed in simulation based approaches for robustness assessment with the use of formal techniques such as boolean SATisfiablity (SAT)-based bounded sequential equivalence checking.

Besides these contributions many others have shown that, formal approaches to prove fault tolerance are relatively complete with respect to the whole input space in comparison with simulation based approaches. But still these methods do not scale well with the increasing complexity of digital electronic systems and suffer from run time limitations [3].

2.3.2 Empirical Methods

2.3.2.1 Field Experience based

Field experience based robustness methods rely on data collected form field to assess the robustness of designs. Generally a long history of field data in needed to make expert judgments about the reliability.

2.3.2.2 Fault Injection based

Besides analysis and field-experience based robustness assessment methods whose accuracy and applicability are significantly restricted, fault injection has emerged as a particularly attractive and viable solution for complex fault tolerant systems [49]. It allows speeding up fault occurrences to analyze the system response in terms of fault-tolerance capability and performance. Simulation-based fault injection environments need less time and effort to implement and offer better controllability and observability.

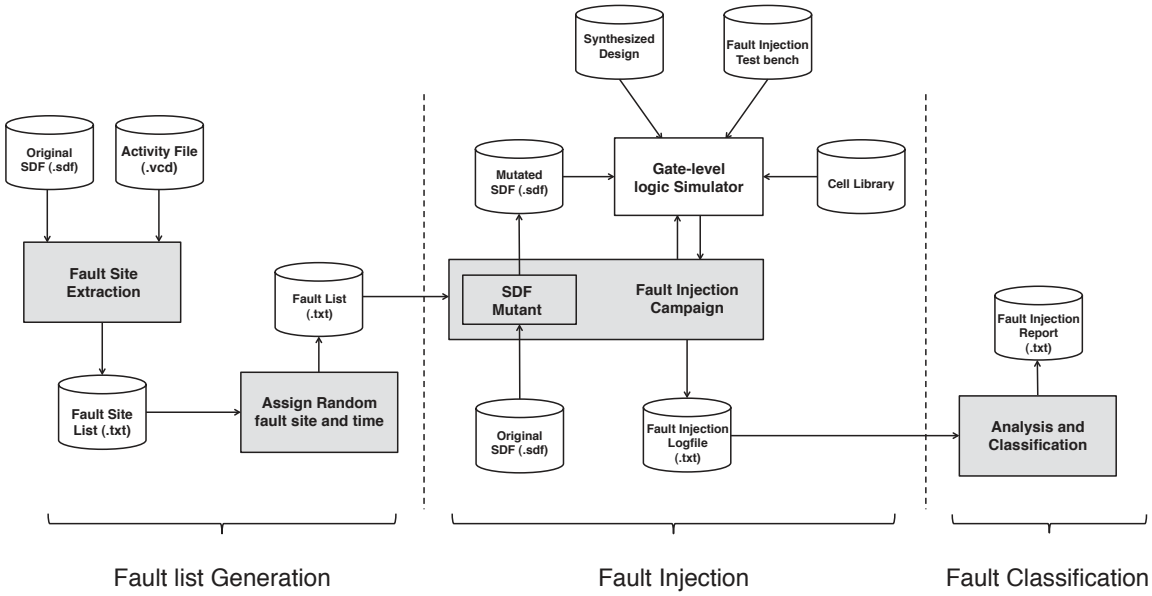


Fig. 2.17: Fault Injection Flow

2.3.3 A Gate-Level Fault Injection Framework

To assess and compare the fault-tolerance capability of the Fault Tolerant architecture, we devised a simulation-based gate-level fault-injection framework. Gate-level simulation provides a suitable paradigm to perform fault-injection experiments because unlike micro architectural-level simulation, it offers high fidelity to model most of the physical defects and transient faults, and is much faster than transistor-level simulation. This fully automated fault-injection framework uses a flow shown in Figure 2.17. The flow is partitioned into three sections, each explained in the following subsections.

2.3.3.1 Fault list generation

In the fault list generation part a parsing script extracts an exhaustive fault site list from either Standard Delay Format (SDF) file or form Value Change Dump (VCD) file depending on the type of faults. Transient and permanent fault sites are extracted form the VCD file and the timing fault sites are extracted from the SDF file. The choice of using the VCD and SDF files for fault site extraction is based on the fact that the extracted sites are good gate-level representations of fault location and are easily interpreted by the gate-level logic-simulator used for fault injection. Another script randomly selects fault sites from the exhaustive fault site list with a possibility of constraining the fault locations to be in the specified design modules. It also associates timings

Table 2.1: Fault injection parameters

Input Parameter	Description
Fault model	To specify to generate either permanent (stuck-at), transient (temporary stuck-at) or timing (interconnect delay) fault.
No. of injections	Specifies the number of faults in the list to be generated.
Injection time range	Specifies the range to constrain the random injection time generation process. Mainly used to ensure that none of the faults are injected during circuit initialization nor at the time too close to the end of the simulation.
Injection duration range	Used for transient faults to specify the pulse width of injected transient fault.
Timing error range	Specifies the range of additional random delay values to be used for timing errors.
Fault Location constraint	Constraints the fault list generation process to produce a faults list according to specified module(s).
Simulation duration	Gives the time of each fault simulation.
Injection type	Indicates whether single of multiple fault injections per simulation

with each selected fault site in a constrained-random fashion. The input parameters it uses to control the fault injection are listed in Table 2.1 with their description.

2.3.3.2 Fault injection

In the fault injection part of the flow the *fault injection script* recursively runs gate-level simulations in logic simulator and inject faults according to the fault list by controlling it through simulator commands. A test bench generates the required data to be logged for analysis. Depending on the type of design under test, either cycle-by-cycle information is stored during simulations or just the final computed result is logged. The test bench also monitors *Err* signal and this information is also logged. The SDF mutant script injects timing faults in the SDF file to be used of simulations which involve timing faults. The fault injection campaign script generates a fault injection report. Based on the possibilities offered by gate-level simulation to inject faults, we model permanent faults as stuck-at faults, SETs as temporary stuck-at faults and timing faults as interconnect delay faults. These fault models and their implications on the fault injection mechanism are discussed in the following subsections.

Permanent fault injection: We use standard stuck-at fault model to represent permanent faults. However instead of arbitrarily imposing stuck-at-0 or stuck-at-1 on the circuit nodes, we base this decision on the logic state of that node at the time of injection, i.e. if it were at logic-level 0 at the time of injection, a stuck-at-1 is forced and vice versa. This modification results in a large number of faults to manifest themselves as errors. A permanent fault is defined by fault location (l) and fault injection time (t). Figure 2.18a shows a timeline illustration of a permanent fault injection campaign. As shown, a fault injection campaign consist of a number of fault simulations specified at the fault list generation step. Each simulation is started and proceeded till time t. At

this time the logic value at the injection location *l* is read and an inverted logic is forced on *l*. The simulation is resumed till the end of the workload and the collected information is logged into the fault injection logfile.

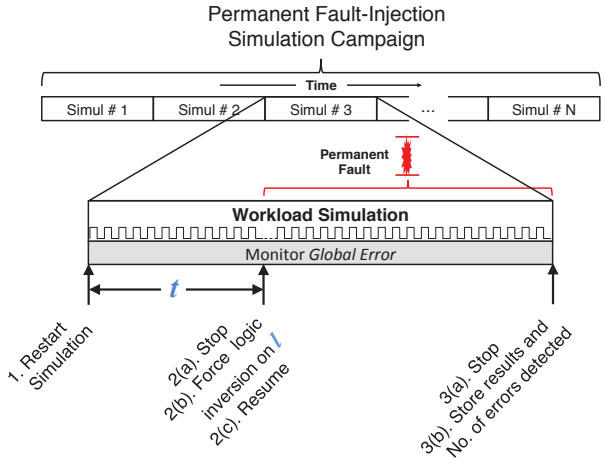
Transient fault injection: Transient faults are modeled as temporary stuck-at faults, which represent SETs as digital pulses using three parameters, fault location (*l*), fault injection time (*t*) and pulse duration (*d*). As shown in Figure 2.18b transient faults are injected in the same way as permanent except that the forced logic value on *l* is released after time *d* form the time it is forced i.e. *t*. At the end of each simulation the information gathered is saved in the fault injection logfile.

Timing fault injection: Timing fault injection uses SDF mutation technique, which models timing fault as an interconnect delay fault. In this technique the original SDF file generated by the synthesis tool is mutated such that the delay of a specific interconnect (*l*) is increased by an amount Δt. As shown in Figure 2.18c at the beginning of each simulation the original SDF file (*sdf1*) is mutated based on *l* and Δt. This mutated SDF file (*sdf2*) is used for gate-level simulation instead of the original one. Once the workload is finished running and the necessary information is logged and the mutated SDF file is deleted.

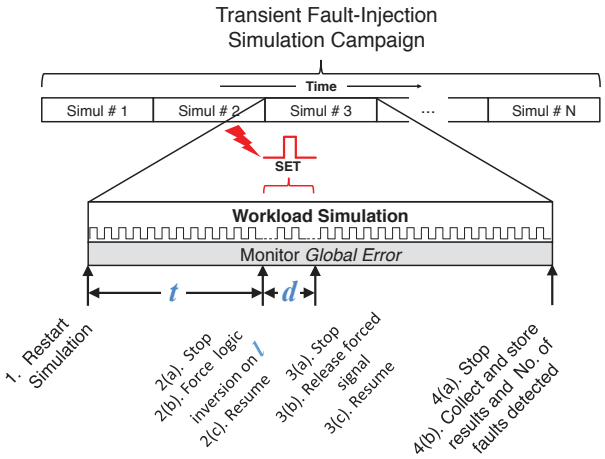
2.3.3.3 Fault analysis and classification

An analysis of the fault-injection report allows us to classify the injected faults into five categories based on the outcomes.

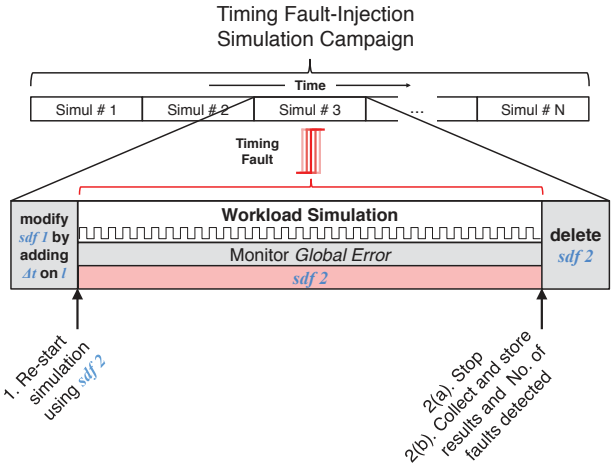
- **Silent Faults:** Faults that had no effect on the execution of the workload are classified as Silent. The workload terminates normally with no error detection, the result is correct and the contents of pipeline registers and other memory elements are the same as those of golden run.
- **Latent Faults:** Faults are classified as latent if the workload terminates normally, the result is correct, but the contents of pipeline registers, register-file or other memory elements are not the same as those of golden run. These stored errors can effect the computation at any later time moment when the workload makes use of data at the corrupt memory locations for computation. These types of faults effects are considered critical because they may result in erroneous computation without detection.
- **Fail-silent Faults:** The workload terminates normally with no error detection and the result computed is wrong. These faults are the most critical as the result computed are wrong without any error indication.



(a) Permanent Fault Injection Campaign



(b) Transient Fault Injection Campaign



(c) Timing Fault Injection Campaign

Fig. 2.18: Timeline illustration of Fault Injection Campaigns

- **Corrected Faults:** The workload terminates normally with at least one error detected, the result is correct and the content of pipeline registers and register-file are the same as that of the golden run.
- **Unclassifiable:** Some injected faults result in setup or hold violations and cause unknown logic value *X* to propagate. These *X*-propagations are due to limitation of gate-level simulation but in real circuits setup and hold violations may cause a faulty value to be stored in memory elements and this anomaly can be detected by the detection mechanism. Thus, in actual silicon test case these fault will either fall into corrected or silent fault categories, which are non-critical from the robustness point of view. Since using gate-level simulation we cannot make a distinction among them, we put them into the unclassifiable category.

Among these five categories of faults, we consider Latent and Fail-silent faults to be critical in our analysis of fault tolerant capability. These critical faults are the ones that escape the detection and lead to a failure. The ratio of the number of these critical faults w.r.t the number of total injected faults gives a figure of merit to compare the fault tolerant capability of different architectures.

2.4 Experimental Comparative Study

Many studies in literature like [62], [89] and [108], provide evaluation results within the scope of the architecture proposed therein. However, it is essential that these similar schemes be comprehensively compared using identical set of experiments and conditions in order to have a meaningful contrast. In this section, we present a comprehensive experimental comparative study of three fault-tolerant architectures with similar fault-tolerance capability in the context of spatial and temporal characteristics of faults and the architectural cost merits, which include area and power consumption. These architectures include partial-TMR and full-TMR [50] and the HyFT [92, 91].

2.4.1 Experimental Methodology

Experiments are performed to compare the merits of HyFT architectures with partial-TMR and full-TMR architectures. Each architecture is applied to a few of the ITC'99 benchmark circuits and are synthesized using NanGate 45nm Open Cell Library [67]. The area figures are obtained from the synthesized designs and power estimates are obtained by taking into account the switching activity generated by back-annotated gate-level simulations. The workload for simulation is a set of patterns optimized for stuck-at fault detection. The reason for using such a workload is to obtain switching activity distributed in all parts of the circuit. The performance overhead is

Table 2.2: Fault injection parameters

(a) Common parameters

Parameter	**Transient faults**	**Permanent faults**
Fault model	Temporary Stuck-at	Stuck-at
Injection duration range	0.25 ns - 1.25 ns (Constrained random)	-
Workload	Stuck-at fault detection patterns	
Fault location constraint	Allover the architecture (Random)	
Injection type	Single fault	

(b) Individual parameters

Benchmark	**No. of injections**		**Simulation duration** (per injection)	**Injection time range** (Constrained random)
	Transient faults	**Permanent faults**		
b01	6088	6088	240ns	45ns - 200ns
b02	5245	5245	140ns	45ns - 100ns
b03	25630	25630	340ns	45ns - 300ns
b05	25248	25248	1020ns	45ns - 980ns
b06	9601	9601	220ns	45ns - 180ns
b08	20771	20771	550ns	45ns - 510ns

evaluated in two different aspects. Firstly, in terms of temporal performance degradation, which is basically the additional delay in the data-path due to the fault-tolerant architecture (e.g., voter delay in TMR), and secondly in terms of error recovery penalty under a certain fault rate.

The fault-tolerance capability of the three schemes is estimated by performing transient and permanent fault injections in the combinational logic parts of the circuits, by using a gate-level simulation based fault-injection framework discussed in Section 2.3.3. The fault injection parameters used for this experiment are listed in Table 2.2.

During the fault injection campaign a fault-injection report is generated which contains the cycle-by-cycle outcome of each simulation. At the end of fault-injection campaign the fault-injection report is analyzed to classify the faults according to the fault effects into *silent*, *corrected* and *fail-silent* fault. The ratio of the number of fail-silent faults to the number of total injected faults gives us a figure to compare the fault tolerance capability of the four different schemes.

2.4.2 Comparative Analysis

2.4.2.1 Area and Power Overhead

Table 2.3 gives the average area and power for the BL circuits and the fault-tolerant implementations based on the results of their application on six ITC'99 benchmark circuits in column 2 and 4. In column 3 and 4 it gives their associated percentage overheads of area and power with the BL circuits as reference. The most obvious area and power overhead figures are those of Full-TMR. As it is based on triplicating the CL blocks and also the FF, it occupies a little more than three

times the area and consumes a few microwatts over the three times the power consumed by BL implementations. This extra area and power is due to the voter in the Full-TMR scheme.

Table 2.3: Average Area and Power estimation results

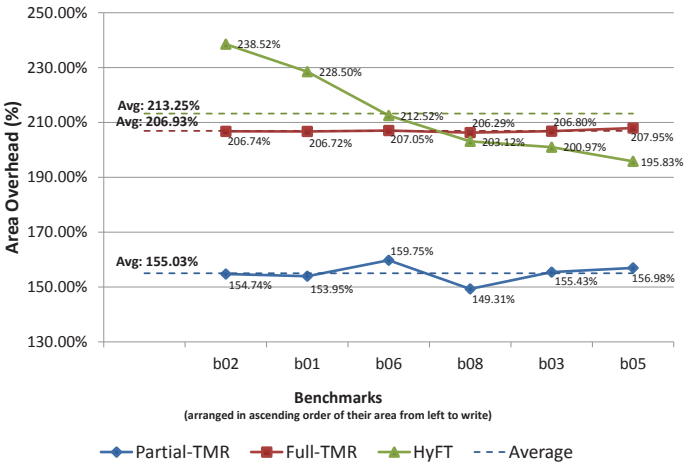
	Avg. Area (μm^2)	Avg. Area Overhead (%)	Avg. Power (μW)	Avg. Power Overhead (%)
BL	1231.00	-	351.50	-
Partial-TMR	3141.59	155.02	971.66	173.32
Full-TMR	3781.32	206.93	1077.74	206.09
HyFT	3739.43	213.24	859.67	157.36

The average percentage of area overhead values in Table 2.3 show that the partial-TMR implementation consumes less in terms of area that is about 155%. The most expensive architectures in terms of area is HyFT, with an average overhead of around 213%.

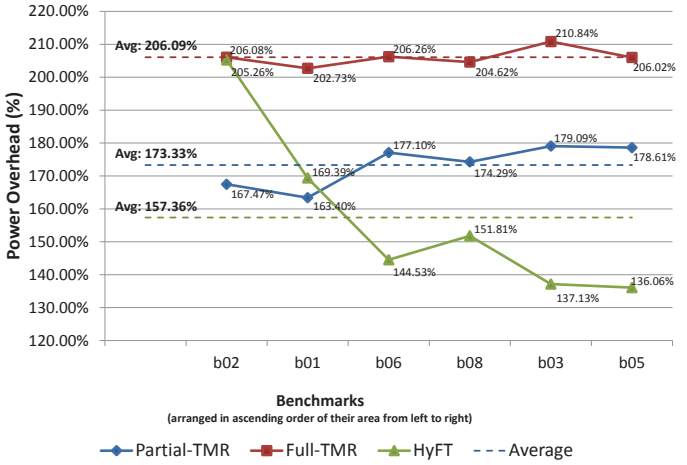
As far as the power consumption is concerned, HyFT architecture is most efficient based on the average power overhead figure of about 157% in Table 2.3. partial-TMR stands at 173%, making full-TMR the least power efficient scheme. This high power consumption of full-TMR is accounted to the triplication of FF. On the other hand HyFT saves power by having one CL copy in stand-by all the time.

The graphs in Figure 2.19 show the percentage increase in area (Figure 2.19a) and power (Figure 2.19b) of the BL circuits to implement the three considered fault-tolerant architectures. Note that the benchmark circuits are arranged in ascending order of their size from left to right on X-axis to illustrate the impact of the size of CL block on the area and power overheads. The dotted lines in Figure 2.19 represent the average percentage figures of area and power overheads for the corresponding fault-tolerant architecture implementation.

An important observation that can be made in the graphs of Figure 2.19 is that, the area and power overheads of both partial and full-TMR are relatively independent of the size of CL block to which they are applied. However, these overheads for HyFT change with different sizes of benchmarks such that the area and power overheads of HyFT decrease with the larger benchmarks. This observation also gives an idea of the anticipated impact on the area and power overheads for CL blocks larger than the benchmarks considered in this study. Although the average area overhead of HyFT is higher than other considered fault-tolerant architectures but with large CL blocks we can expect it to decrease. Whereas the power overhead of HyFT, which is already the minimum, tends to further reduce with larger CL blocks.



(a) Area overhead



(b) Power overhead

Fig. 2.19: Impact of CL blocks size on area and power overheads

2.4.2.2 Performance

The first evaluated measure of performance is the temporal performance degradation. In partial-TMR and full-TMR, it is defined by the delay of voter circuit in the data-path. In case of HyFT it is due to the delay of shadow latch multiplexers in input register responsible for rollback and the reconfiguration multiplexer and demultiplexer. The comparator being outside the critical path does not contribute to the temporal performance degradation. Using Static Timing Analysis (STA) the temporal performance degradation for partial-TMR and full-TMR was estimated to be 0.73% for a 100MHz operation. The same for HyFT was found to be 9.7% without any design optimization.

Table 2.4: Transient fault injection results summary

	Avg. % of Silent faults	Avg. % of Corrected faults	Avg. % of Fail-silent faults
BL	92.51%	0.00%	7.49%
Partial-TMR	99.97%	0.00%	0.03%
Full-TMR	100%	0.00%	0.00%
HyFT	92.46%	7.26%	0.28%

Table 2.5: Transient fault injection results

	Percentage of Fail-silent faults (%)			
	BL	**Partial-TMR**	**Full-TMR**	**HyFT**
b01	7.49	0.00	0.00	0.37
b02	8.11	0.11	0.00	0.28
b03	8.18	0.03	0.00	0.26
b05	7.11	0.02	0.00	0.21
b06	7.07	0.04	0.00	0.33
b08	7.45	0.05	0.00	0.33
Average	**7.56%**	**0.03%**	**0.00%**	**0.28%**

The figures that can give us a measure of the second considered performance aspect, i.e. the error recovery penalty, can be interpreted from the transient fault injection results presented in Table 2.4. These results are obtained by injecting transient faults at an average rate of 250K faults/second. It can be observed in Table 2.4 that for partial-TMR and full-TMR the percentage of corrected faults is zero. This is because TMR is an error masking technique rather than an error detection and correction one and does not indicate the presence of error. With no provision of identifying the corrected faults, they are kept within the category of silent faults in our analysis. It also indicates that the error recovery penalty for TMR is zero as it corrects errors by masking them instead of undergoing a reconfiguration and re-computation cycle. It can also be seen in Table 2.4 that HyFT corrected on average 7.26% of injected faults. For each detected and corrected SET the HyFT undergoes a recovery phase that takes 2 additional cycles [14]. According to these figures, HyFT spends around 14.52% of total computation time on recovering from potentially erroneous states.

2.4.2.3 Fault Tolerance Capability

Quantitative analysis To compare the fault-tolerance capability of different architectures we analyze them in terms of the percentage of faults that resulted in a fail-silent outcome among the total number of injected faults. Table 2.5 gives the transient fault-injection experiment results.

Table 2.6: Permanent fault injection results

	Percentage of Fail-silent faults (%)			
	BL	**Partial-TMR**	**Full-TMR**	**HyFT**
b01	98.37	2.37	0.00	0.15
b02	96.28	2.03	0.00	0.06
b03	98.15	1.38	0.00	0.06
b05	97.84	0.50	0.00	0.08
b06	97.23	0.66	0.00	0.13
b08	98.03	2.34	0.00	0.07
Average	**98.03%**	**1.36%**	**0.00%**	**0.08%**

Table 2.5 shows that the incorporation of each fault-tolerant architecture into the BL circuit reduces the percentage of fail-silent faults to a different extent. The average percentage of fail-silent faults that was originally 7.56% in BL is brought down to 0.03% by partial-TMR. A through analysis of the fault-injection report revealed that these 0.03% faults were among those which were injected at the inputs of CL blocks and affected all the three TMR copies in the same way, thus resulted in a common-mode failure. Full-TMR on the other hand did not encounter this problem because of it construction and turned out to be the most effective by tolerating the effects of all the injected transient faults. In case of HyFT 0.28% of injected faults escaped detection and affected the results. With further investigation we found out that these fail-silent outcomes were not linked to a specific location as in case of partial-TMR, but escaped detection due to their specific timing characteristics. Static timing analysis showed that these 0.28% fail-silent faults were among those that were injected at a time such that their effects appeared at the inputs of register during the clock setup-hold window. Since in HyFT the *comparison-window* does not overlap the setup-hold window, these transient faults managed to affect the data during captured but escaped detection by missing the *comparison-window*. This problem of non-overlapping setup-hold window and *comparison-window* is analyzed in Chapter 4 and an improved version of HyFT architecture is proposed that solves the problem.

Similar observations can be made from the permanent fault injection results shown in Table 2.6. An average 1.36% of faults injected in partial-TMR result in fail-silent outcome, mainly due to the common-mode effect. Full-TMR shows nearly complete tolerance against permanent faults and in HyFT 0.08% faults escaped detection mainly due to the setup-hold window and *comparison-window* separation.

Qualitative analysis Some aspects of fault-tolerance capability that have an implication on the lifetime reliability of the circuit cannot be inferred from the fault injection experiment result discussed in the previous subsection. Therefore, we analyze them qualitatively here.

When a circuit enters into the wear-out phase of it's lifetime, most of the wear-out mechanisms show early symptoms as increasing signal propagation latency prior to inducing permanent device failures [24]. The ability of the HyFT architectures to detect these early symptoms and act upon by causing reconfigurations reduces the aging effects on the system by distributing the stress on two of the three CL copies. The capability of selective sparing helps reduce the rate of failures and increase the life span of circuit parts that embed such fault-tolerant architecture. Another qualitative aspect of fault-tolerance capability is fault accumulation effect that distinguishes both considered versions of TMR from HyFT. TMR is an error masking architecture that does not indicate the presence of error, instead just corrects them until only one computational copy exhibits an error. When faults accumulate due to wear-out and multiple copies start getting affected, TMR fails to correct them and the lack of any provision of indicating error ends up in fail-silent outcomes. Whereas HyFT is able to correct errors until two faulty copies manifest the effect of fault at the output in a same way at the same time, which is very less likely. In all other possible scenarios HyFT, if cannot correct can at least indicate the presence of error and continue fail-safe operation.

2.5 Summary

Table 2.7 gives a summary of comparison of the architectures discussed in Section 2.2, also listed in column 2 of the table. Columns 3, 4 and 5 specify the type of fault that these architectures have the capability to detect and correct. Column 6 and 7 identify which of these architectures have or can possibly incorporate power conservation and performance improvement features of DVS and DFS. The architectural components that each scheme replicates is given in column 8. Columns 9, 10 and 11 give some figures of area, power and error recovery overheads respectively, associated with some of these architectures. Finally the last column determines which of these schemes also improve the lifetime reliability of the circuit.

We classified these architectures in two categories, those which provide protection against permanent, transient and timing faults as *Full Protection* solutions and those, which can handle a subset of these types of faults as *Partial Protection* solutions. Partial-TMR, Full-TMR, DARA-TMR and HyFT architectures fall in the first category. On the other hand, Razor is intended to detect and correct timing errors, and STEM and CPipe architectures are capable to deal with transient and timing faults, thus are considered partial protection solutions. However, some of these partial protection techniques have features like DVS and DFS, as can be seen in columns 6 and 7, which is not the case with full protection schemes considered here. On the other side, the

HyFT architecture, due to its unique error detection and micro-rollback capability, becomes an ideal candidate for the application of these performance and power optimization techniques.

Now if we focus our attention to the overheads associated with each of the architectures that offer protection against all the three type of faults, we can observe that HyFT incurs area overhead more that Partial-TMR and Full-TMR. Although there is no experimental data available about area cost of DARA-TMR in literature but it would be certainly more than HyFT because it uses CL and FF triplication, whereas HyFT triplicates CL only. In addition, DARA-TMR uses three times more comparators than HyFT. It can be observed in Table 2.7 that HyFT saves a significant amount of power in comparison with Partial-TMR and Full-TMR. The error recovery overhead for Partial-TMR and Full-TMR is zero because TMR is an error masking technique instead of an error detection and correction scheme. As its name suggests, DARA-TMR is also based on TMR but treats permanent fault occurrence as a very rare phenomenon and does not offer a fast reconfiguration mechanism. On the other hand, HyFT incurs an error recovery penalty of either 2 or 4 cycles to mitigate any type of fault. Although the partial protection schemes considered here have slightly lower error recovery penalties, they cannot be compared with that of HyFT because of the difference in their fault-tolerance capability. Among the eight fault-tolerant architectures considered in this comparison, HyFT is the only one that offers a life-time reliability improvement by selectively sparing the weakest CL block as discussed in Section 2.4.2.3. From this comparison it can be inferred that the HyFT architecture offers a fault tolerant capability almost equivalent to Full-TMR structures but with additional benefits of power saving and lifetime reliability improvement and also provides the opportunity to apply power conservation and performance enhancement techniques like DVS and DFS.

Table 2.7: Summary of comparison of different related fault-tolerant architectures

		Permanent fault tolerance	**Transient fault tolerance**	**Timing fault tolerance**	**DFS**	**DVS**	**Hardware redundancy**	**Area overhead**	**Power overhead**	**Error recovery overhead**	**Lifetime reliability improvement**
Partial	**PaS[47]**	✓					CL×3	No data in [47]	No data in [47]	No data in [47]	
	Razor [29]			✓	✓		FF×2	1%-3%	3.1%	1 cycle	
	STEM [5]		✓	✓		✓	FF×3	14%-15%	No data in [5]	1 or 3 cycles	
	CPipe [89]		✓	✓		✓	CL×2, FF×2	No data in [89]	No data in [89]	1 cycle	
Full	**Partial-TMR**	✓	✓	✓			CL×3	155%	173%	0 cycles	
	Full-TMR	✓	✓	✓			CL×3, FF×3	207%	206%	0 cycles	
	DARA-TMR [108, 107]	✓	✓	✓			CL×3, FF×3	No data in [107, 108]	No data in [107, 108]	No data in [107, 108]	
	HyFT	✓	✓	✓	Feasible	Feasible	CL×3, FF×2	213%	157%	2 or 4 cycles	✓

Chapter 3

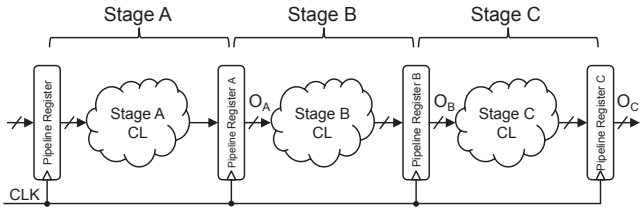
Pipelined Hybrid Fault Tolerant Architecture

In computer systems, pipelining is defined as *"an implementation technique whereby multiple instructions are overlapped in execution; it takes advantages of parallelism that exists among actions needed to execute an instruction"* [41]. Pipelining is one of the key methods used to increase the throughput of digital systems such as microprocessors by splitting up each instruction into a sequence of steps so that different steps can be executed concurrently in parallel, rather than processing each instruction one at a time.

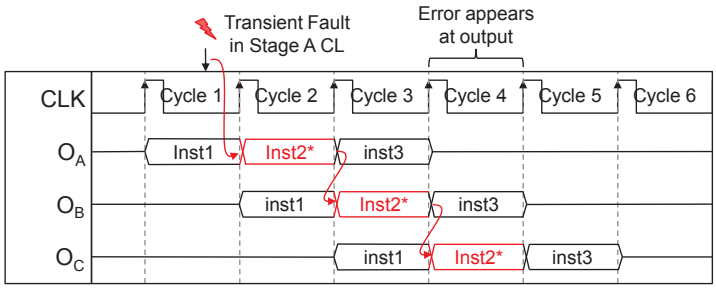
We start this chapter with a discussion on how errors can propagate in complex pipeline structures focusing on the specific pipeline properties that make error detection and correction difficult. We will extend the capabilities of HyFT architecture with some architectural modification that allow it to address these problems of error confinement, while doing that, we will formulate the principles of the new Pipelined Hybrid Fault Tolerant (PHyFT) architecture. We will also present the details of a comprehensive case study carried out to prove the effectiveness of PHyFT architecture to improve the robustness of practical pipeline structures and to compare it with other state-of-the-art techniques. Eventually, we will present the experimental results obtained form the case study and will conclude the chapter.

3.1 Error Propagation in Pipelined Circuits

Similar to stand-alone logic circuits, pipeline architecture may also suffer from transient, permanent and timing errors. Furthermore, in these structures, there are error propagations between pipeline stages which require special error detection and re-computation schemes for error correc-



(a) Linear Pipeline



(b) Time plot of transient error propagation in Linear Pipeline

Fig. 3.1: Example transient error propagation in Linear Pipeline

tion. In the following subsections we focus on a few types of pipelines and see what implications they have on the error propagating through them.

3.1.1 Linear Pipeline

A pipeline in its simplest form, in which the entire datapath traverse successively through each pipeline register without any signals bypassing any of the pipeline registers is called a linear pipeline. Figure 3.1a illustrates an example of a three stage linear pipeline. The graph of Figure 3.1b shows the timing plot of outputs of each of the three pipeline stages, namely O_A, O_B and O_C in case of a transient fault occurrence in the CL block of *Stage A* during *Cycle 1*. The executions in red with an asterisk represent erroneous instructions.

In this example, we can see that at the beginning of *Cycle 2*, the fault is captured by the *Pipeline Register A* and remains active as error at the output of *Stage A* while *Stage B* is executing the first instruction. At the next clock edge, the error appears at the output of *Stage B*. Consequently, the error appears at the pipeline output during *Cycle 4*. Eventually, the pipeline returns to normal operation at the fifth period when errors have reached the final register. In this example, we consider transient faults. A similar propagation behavior could be observed in case of hard permanent or a permanent delay fault except that in these two cases all the subsequent instructions would be effected by error because of the persistent nature of the faults.

We saw that the error latency depends on the number of stages the error traverses before becoming observable. In deep pipeline (with a greater number of dependent steps), and an error detection mechanism at the output of the long chain a deep rollback-recovery would be needed to tolerate the error effects. These deep rollbacks have a significant impact on circuit performance specially in high error rate environments and are considered an overkill solution for error recovery.

3.1.2 Nonlinear Pipeline

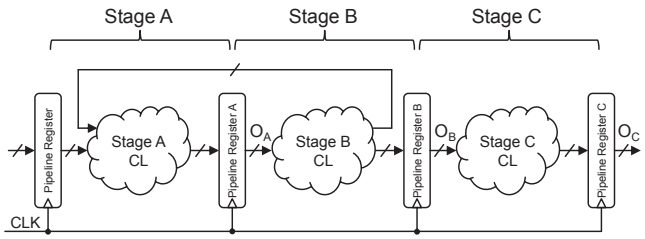
The presence of feedback and feed-forward signals give a pipeline its nonlinear nature. These "loosely-bounded" signals between stages are commonly found in microprocessor pipelines for functions like register write-back, program counter iterations, branch addressing, datapath control signaling etc. Furthermore, the number of these signals increase if the microprocessor also incorporates advanced features like hazard detection and data-forwarding [97]. These advanced features are intended to improve performance by increasing hardware resource utilization. However, they change the error propagation behavior of the pipeline.

Figure 3.1b gives an example of error propagation in a nonlinear pipeline shown in Figure 3.2a. Note that a feedback connection is shown that transverse form *Stage B* to *Stage A*. Consider a case in which a transient fault occurring in *Stage A* during *Cycle 1* causes erroneous data to be captured in *Pipeline Register A*, represented by *inst2** label during *Cycle 2*. This error not only effects the computation in *Stage B*, but it also propagates back to *Stage A* CL through the feedback connection. Thus both the pipeline registers hold erroneous data at the next capture edge, represented by *inst3** and *inst2** labels during *Cycle 3* in Figure 3.2b. Finally, during *Cycle 4* and *Cycle5* the error manifests at the output.

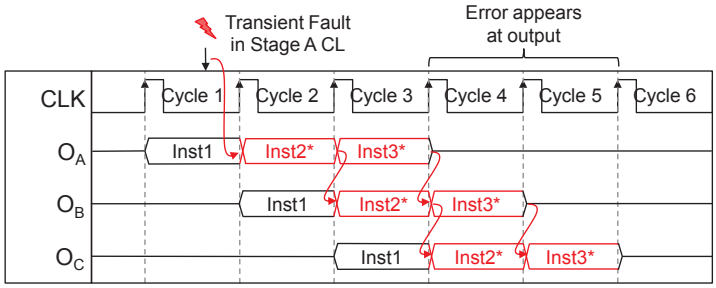
Here we saw that in non-linear pipelines the error effects may not remain confined within the faulty stage during one clock period but can propagate to other stages through pipeline feedback and feed-forward connection. This property of nonlinear pipelines makes error detection and making reconfiguration decisions a non trivial task.

3.1.3 Pipeline with Memory Interface

Memories in datapaths also change the error propagation dynamics of the pipeline. Figure 3.3a shows an example of pipeline containing an asynchronous memory and Figure 3.3b demonstrates the behavior of this pipeline in presence of a transient fault. This example considers a case in which a memory write instruction represented by *inst2(Wr)* label stores data in memory during *Cycle 3* and a read instruction reads the same data at during *Cycle 5* represented by *inst4(Rd)* label. Furthermore, it assumes a transient fault corrupts the data to be stored in memory in *Cycle 2* and



(a) Nonlinear Pipeline



(b) Time plot of transient error propagation in Nonlinear Pipeline

Fig. 3.2: Example transient error propagation in Nonlinear Pipeline

during *Cycle 3* the erroneous data becomes latent inside the memory and remains there until the read instruction reads the data and it appears at the output during *Cycle 6*.

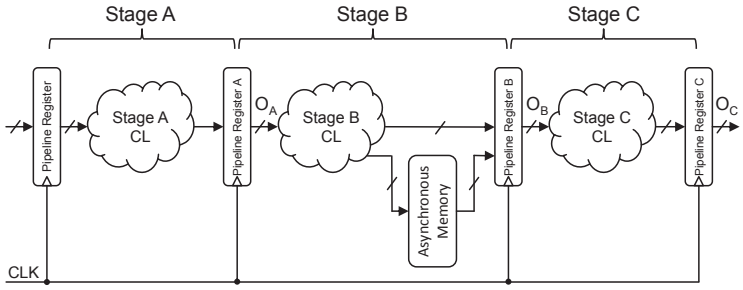
Memories in datapath make error detection difficult because of their inherent nature of making error latent inside them, if a data or address anomaly occurred during write operation. No matter how deep the rollback-recovery scheme, it cannot ensure to have a checkpoint to return back in execution where the write operation can be repeated to tolerant the error latent in memory unless intelligent checkpointing schemes are employed. Although these advanced checkpointing schemes are effective in increasing the robustness of pipelines with memories, they remain a burden on performance of such systems.

3.2 Extension of HyFT Architecture to Pipelined Structures

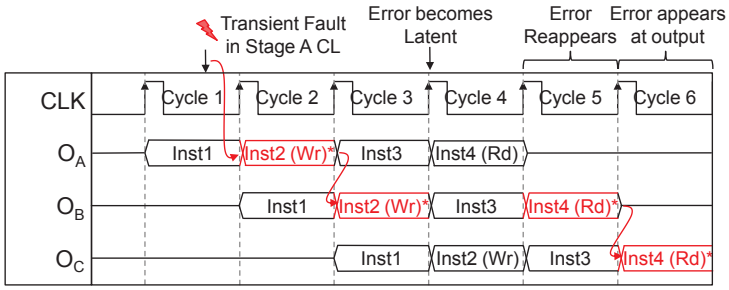
3.2.1 Basic Pipeline Cascading

In this section with the help of a simple example, we explain some necessary considerations and work through the problem of extending HyFT architecture to make it applicable to simple pipelined circuits.

In Section 2.2.7, we discussed the non-pipelined implementation of HyFT architecture for stand-alone circuits. Now consider two such hardened stages cascaded together to form a two-



(a) Pipeline with Memory Interface



(b) Time plot of transient error propagation in Pipeline with Memory Interface

Fig. 3.3: Example transient error propagation in Pipeline with Memory Interface

stage pipeline. In order to maintain synchronization, both the stages in thus formed pipeline should rollback in chorus when needed. This means that rollback is a centralized functionality of the entire pipeline and can be handled by a single rollback signal generated by the control logic. Unlike rollback, reconfiguration remains a distributed functionality because the reconfiguration state of each stage should be maintained individually according to the error profile of that stage. In order to incorporate a centralized rollback and distributed reconfiguration functionalities, the new control logic for PHyFT architecture is constructed to have a block that manages centralized functionalities like rollback and generating global error (*Err_global)* signal and distributed blocks for handling distributed functionalities like reconfiguration and delayed clock generation. The resulting structure of control logic is shown in Figure 3.4.

In the non-pipelined implementation of HyFT architecture shown in Figure 2.15a, the pseudo-dynamic comparator is placed across a register without any roll-back capability. Since the output register always holds a copy of data computed by the second running Combinational Logic (CL) block, the pseudo-dynamic comparator flagged error only when there are inequalities in the two data due to an error. In the case of cascaded stages as shown in Figure 3.4, the pipeline registers having pseudo-dynamic comparator across them also incorporate rollback capability. This gives rise to a problem of false error flagging during rollbacks. The problem is graphically illustrated in

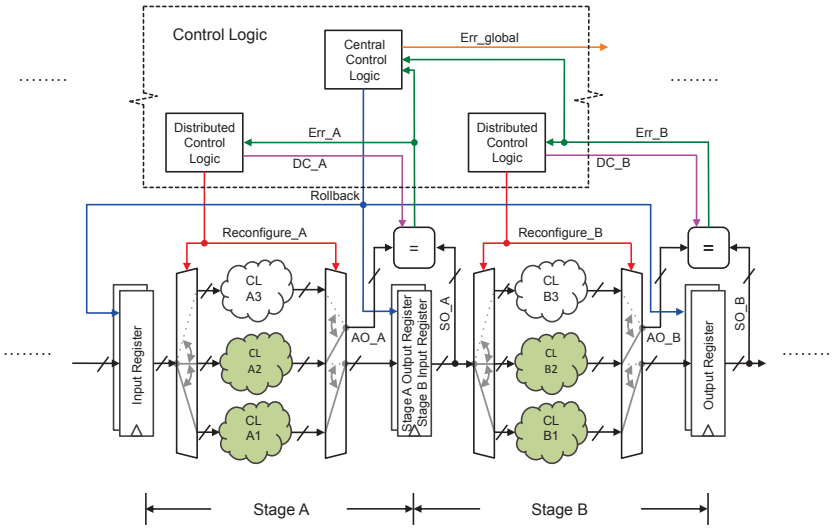


Fig. 3.4: Pipelined HyFT Architecture

Figure 3.5. It shows a case in which a SET in stage A causes an erroneous value to be captured in the sandwiched register. The error is detected and flagged by *Err_A* during *Cycle 2*. At the beginning of *Cycle 3* a rollback occurs in all three pipeline registers. Notice that this causes an unintended situation in which the pseudo-dynamic comparator across the output register compares the corrupted value corresponding to *inst3* with the value corresponding to *inst1*. This flags a false error during *Cycle 3*. This conversely trigger another rollback at the beginning of *Cycle 4* and the system enters an endless loop of rollbacks and false error flags.

The problem of false error flagging is solved by disabling the comparison in all the error free stages for one cycle that follows error detection in one of the stages. The disabling of comparison is realized by inserting additional logic, shown in Figure 3.6, in the distributed control logic blocks. This additional logic masks the delayed clock signal (*DC_B*) if the *Err_global* signal is asserted but the *Err_B* is not, as shown in the truth table given in Table 3.1. With this correction in place, the timing graph for transient error correction is shown in Figure 3.7. Notice that the disabled *comparison-window* of *Stage B* is represented by a non-shaded dotted line box.

3.2.2 Stage/CL Classification and Partitioning

Formulation of a stage-level error detection and recovery scheme for a complex pipeline needs to consider the presence of feedback and feed-forward signals and memories because these pipeline attributes have an impact on the error propagation behavior of the processor as seen in Section 3.1. In the next section we discuss the implications of this considerations on the design of the

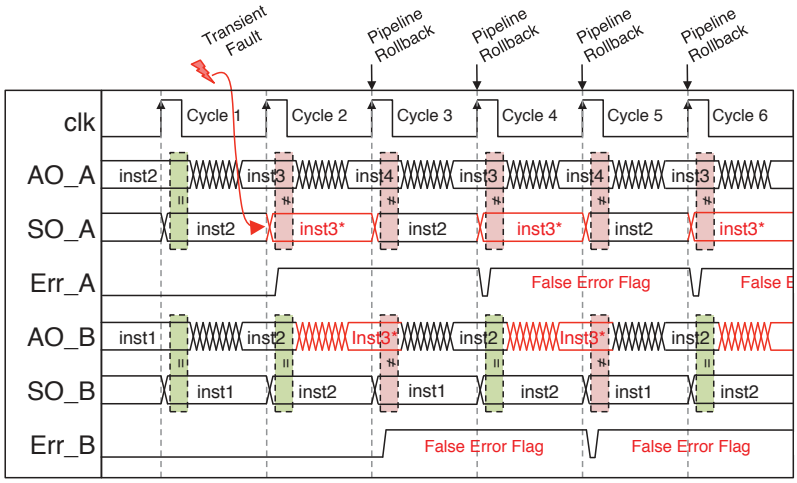


Fig. 3.5: False error flagging

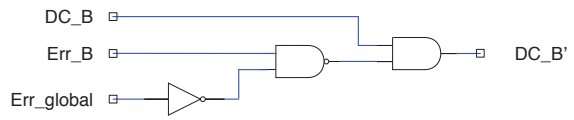


Fig. 3.6: Additional logic to prevent false error flagging

Table 3.1: Truth table

Inputs			Output	Comment
DC_B	Err_B	Err_global	DC_B'	
1	0	0	1	
1	0	1	0	*DC* masked
1	1	0	1	
1	1	1	1	

PHyFT architecture. But first, in order to simplify this discussion we give following definitions that classify the pipeline stages on the bases of aforementioned characteristics:

- **Bounded stages:** We classify the pipeline stages containing CL blocks that have all their inputs and outputs bounded by a single pair of consecutive pipeline registers as shown in Figure 3.8a, as *bounded* stages. A pipeline entirely made up of *bounded* stages forms a linear pipeline. These type of pipelines have a simple error propagation behavior as seen in Section 3.1.1, because an error in one stage remains confined in that stage for one clock period time and can be detected with an error detection mechanism placed at stage level granularity.

- **Loosely-Bounded stages:** The stages containing CL blocks that either generate or receive pipeline feedback or feed-forward signals are classified as *loosely-bounded* stages. We further divide them in two classes:

  – **Influencing stages:** The stages with CL blocks that supply feedback or feed-forward signals to CL blocks in other stages of the pipeline are classified as *influencing* stages.

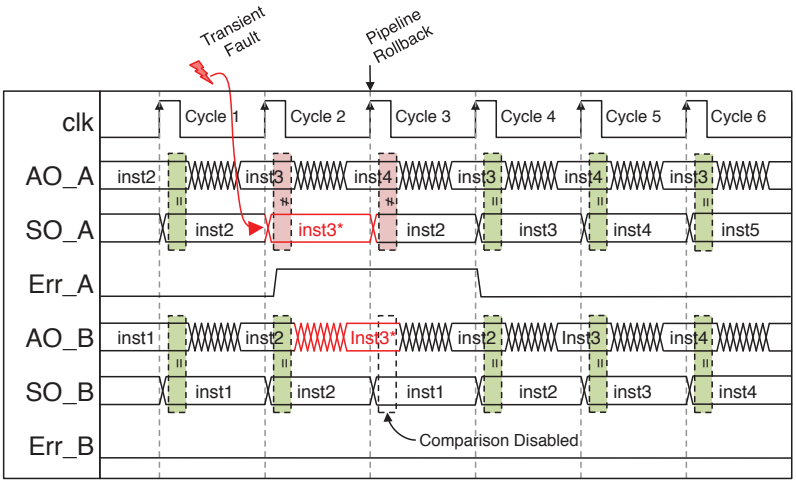
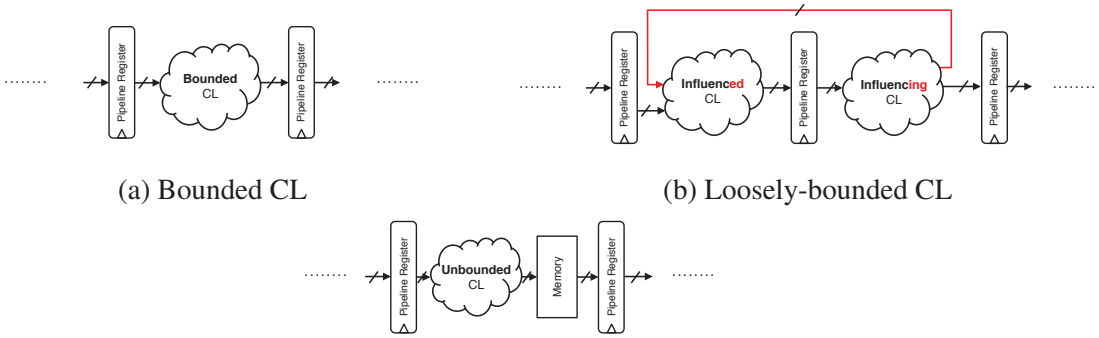


Fig. 3.7: False error flagging problem solved



(a) Bounded CL

(b) Loosely-bounded CL

(c) Unbounded CL

Fig. 3.8: CL Classification and Partitioning

An example of *influencing* stage is given in Figure 3.8b. These CL blocks not just feed signals to the following pipeline register but also to other stages.

– **Influenced stages:** We refer stages with CL blocks that receive feedback or feed-forward signals form CL blocks in other stages as *influenced* stages. Which means that these CL blocks besides getting inputs from their preceding pipeline register also get input signals from other stages. Figure 3.8b shows an example of *influenced* stage.

- **Unbounded stages:** This class of stages holds those with memory interface. The presence of memory interfaces in a pipeline stages make them venerable to latent faults, even if the memories are hardened using error detection and correction codes.

Implication of this classification on the implementation of the error detection and recovery scheme are discussed in next subsections.

3.2.3 Dealing with Error Propagation in Nonlinear Pipeline

Error manifestation due to faults in *bounded* and *influenced* stages remain confined within the stage itself during one clock period because of the absence of feedback or feed-forward connections that generate from these stages. Thus, error detection mechanism at the output of these stages is solely sufficient to detect these errors. Whereas in case of an *influencing* stage the fault effects may or may not remain confined in that stage during one clock cycle, because of the asynchronous feedback and feed-forward signals to other *influenced* stages. Figure 3.9 shows a hardened version of the non-linear pipeline of Figure 3.2 using PHyFT architecture. *Stage B* of the pipeline is an *influencing* stage indicated by the orange shaded region surrounding it. To be able to detect errors, which are due to faults in *Stage B* and have manifested themselves at the output of *Stage A*, the PHyFT Architecture duplicates the feedback and feed-forward signals represented by the pair of orange dotted lines in Figure 3.9. If there were only a single set of feedback and feed-forward signals, an error propagating through them would affect both running copies of CL block in the *influenced* stage in the similar way and may result in a common-mode failure.

The stage-level reconfiguration framework for recovering from permanent faults presented here is also based on the classification of CL discussed in Section 3.2.2. Let us assume that an error is detected at the output of *Stage A*. This error can be the result of fault occurring in itself or this error may have propagated to it through the pipeline feedback paths from *Stage B*. In such case, the PHyFT architecture cannot identify which stage is actually faulty, thus all the suspected stages (which includes *Stage A* and *Stage B*) undergo a reconfiguration. In PHyFT this is achieve by ORing the error signals of *Stage A* and *Stage B* and using this signal as the *error* input of the *Stage B centralized* and *distributed control logic* blocks as shown in control logic block of Figure 3.9. To generalize this principle of PHyFT we can say that the newly derived error signal for each *influencing* stage in the pipeline will be logical OR of the error signals of all the stages it *influences*.

3.2.4 Error detection in Pipeline stages with memory interface

Memories generally take a large amount of silicon area, which makes it impractical to duplicate or triplicate them in order to improve system reliability. On the other hand their regular structure makes error detection and correction codes like Parity and Hamming much feasible. These methods have been in use to efficiently and effectively protect them against SEUs and permanent faults. But that is not enough to ensure the reliability of overall system if an unprotected CL driving the memory inputs and/or if an unprotected CL processes its outputs before being fed to the next pipeline register. In such cases, data anomalies during write operations are especially

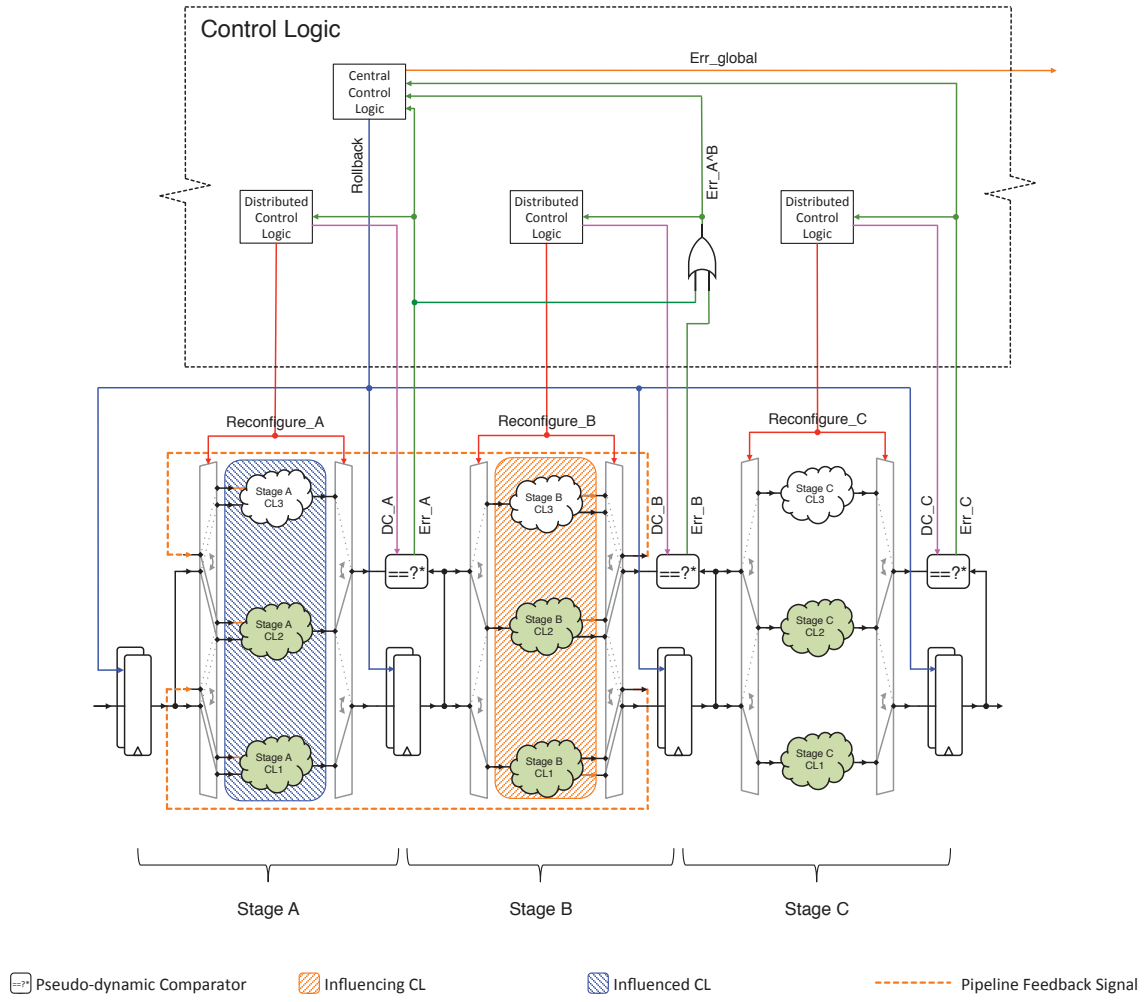


Fig. 3.9: Error detection and reconfiguration scheme in non-linear pipeline

difficult to detect because during write operations there is no data at the output to compare and detect errors and such situations give rise to the number of *latent* faults which are even more difficult to detect once they reappear, due to common mode effects. In the following subsections we will see what solutions the PHyFT architecture offers to improve the robustness of pipelines with memories.

3.2.4.1 Pipeline Stages with non-concurrent read-write access memories

Dual port memories have been previously used as a means of communication between copies of redundant CPU [8]. In our architecture we propose a similar use of dual-port memory. In addition

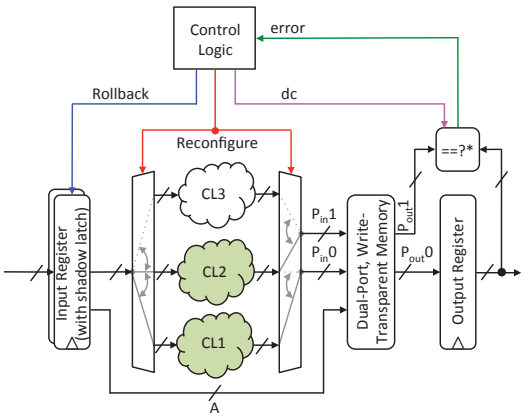


Fig. 3.10: HyFT hardening of pipeline stage with asynchronous non-concurrent read-write access memory.

to that, memories are described in VHDL to have write-transparency property. This property makes the memory latch to be transparent during write operations which means that if a memory location is being updated as a result of a write operation, the data that is being written into the memory location appears on the data output bus as well [8]. The use of dual-port write transparent memory allows propagating any error occurring in one of the two working copies of CL driving the memory, to the following pipeline register where they can be detected by the comparator. Read operations are by nature transparent so there is no need to take them into special consideration. An example of the use of dual-port write-transparent memory can be seen in Figure 3.10. The two input ports of the memory, which are labeled as $P_{in}0$ and $P_{in}1$ are supplied with memory data and control signals from the two running CL copies. The two output ports labeled as $P_{out}0$ and $P_{out}1$ feed the pipeline register and the pseudo-dynamic comparator respectively. However, the address bus comes directly from the preceding pipeline register and is assumed to be error-free because in the scope of this thesis we do not deal with the errors caused due to faults occurring in memory elements.

To evaluate the cost in terms of area, power and performance of the implementing write-transparency property and adding a port to a memory, we synthesized VHDL instances of a small memory block of 512 bytes using 45nm technology [67], with and without the write-transparency and dual-port features. The resulting area, average power and access time estimates are given in Table 3.2. The estimates show that the area overhead associated with these additional features is 3.94% of the original memory with a average power overhead of 13.9%. However, through STA we found that there is no impact of these additional features on the access time of the memory.

One limitation associated with this memory modification is that it can only tolerate errors on data and partially on the control buses of the memory. Anomalies on address bus can not be

Table 3.2: Area and power results of write-transparency dual-port memory

Memory	Area		Avg. Power		Access time
	absolute (μm^2)	%. increase	absolute (mW)	%. increase	absolute (ns)
Baseline	16461	-	0.244	-	1.12
Dual-port write-transparent	17110	3.94%	0.278	13.9%	1.12

detected because there is no possible way to make memory transparent for address data. Furthermore, it is important to note here that, since a transparent write operation can not concurrently coexist with a read operation, the applicability of this approach is only limited to memories without simultaneous read-write operations. Therefore, in the following subsection we detail the architectural modifications to be able to detect error in pipeline stages with concurrent read-write memories regardless of the error location.

3.2.4.2 Pipeline Stages with simultaneous read-write access memory

For the memories that do not allow write-transparency feature, we propose a simple solution to the problem of detecting errors at the memory inputs. It involves placing an additional set of pipeline register and a pseudo-dynamic comparator on the input ports of the memory. The scheme is illustrated by the example implementation in Figure 3.11, were a simultaneous read-write access synchronous memory is protected against error due to fault arising in the CL blocks that drive its inputs. The additional pipeline register is placed on the memory input coming from the first running CL block and the pseudo-dynamic comparator is anchored on to the signals from secondary running CL copy. These additional components are shaded in grey color in Figure 3.11 for easy identification. At each capture edge the additional pipeline register stores the data at the inputs of the memory and during the *comparison-window* the pseudo-dynamic comparator check for any inequalities in the stored data and the one generated by the secondary running copy. In the case of mismatch the pseudo-dynamic comparator reports error to the centralized control logic, which in turn initiates a rollback and the memory operation is repeated. Notice that there is no need of having a shadow latch array with the additional pipeline register because the additional register does not play any roll in the pipeline data path and thus does not need to rollback.

Unlike the previous solution, this method is based on detecting error right at the input of memory blocks instead of relying on comparison of data propagated through the memory. It can also detect error on the address bus of the memory.

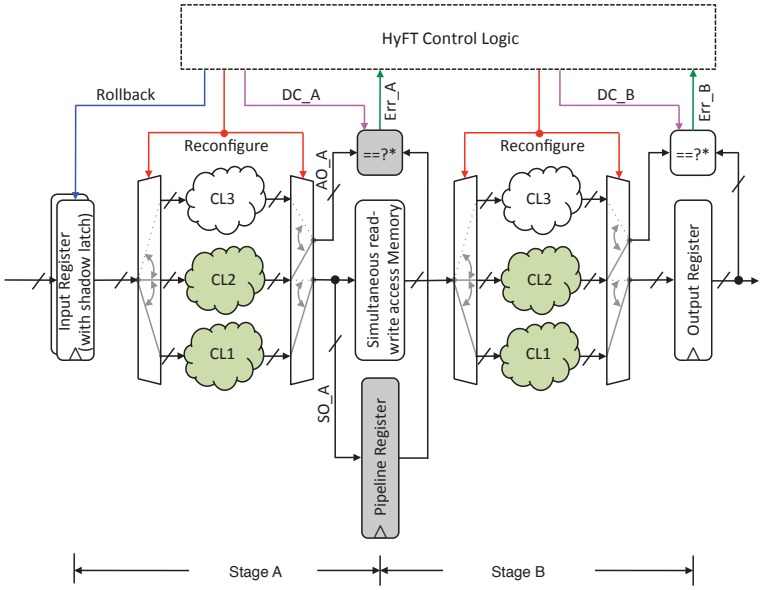


Fig. 3.11: HyFT hardening of pipeline stage with synchronous simultaneous read-write access memory.

3.3 Case Study: Fault Tolerant Microprocessor

The PHyFT architecture was implemented on a MIPS microprocessor with the objective to assess it with respect to area, power and performance overheads and its fault-tolerant capability and to compare it with a classical TMR scheme with bit-wise voter (TMR-b) and a TMR scheme that uses word-wise voter (TMR-w). Area estimations were obtained for five different versions of the microprocessor namely Baseline (BL), Hybrid Transient Fault Tolerant (HyTFT), Hybrid Transient and Permanent Fault Tolerant (HyTPFT), TMR-b and TMR-w by synthesizing them with a 45nm technology library [67]. A simple workload program (provided in Appendix A) that uses add-shift method to multiply two operands was used to obtain switching activity for each microprocessor version and using this activity, estimates of average power consumption were obtained. The fault-injection framework discussed in Section 2.3.3 was employed to comprehensively evaluate the fault-tolerant capability of the microprocessors against transient, permanent faults and timing faults. Details of the five different versions of microprocessor are given in subsequent subsections.

Table 3.3: Baseline microprocessor specifications

Parameter	Value
Data bus width	32-Bits
CPU register size	32-Bits
No. CPU registers	32
Max. data memory	16384 MB
Max. instruction memory	16384 MB
Clock frequency	100 MHz
No. of logic cells	110995
No. of flipflops	7380

3.3.1 BL Microprocessor

The microprocessor used as our case-study platform is a 5-stage MIPS processor. It incorporates hazard-detection and data-forwarding mechanisms. The microprocessor was developed as an academic learning exercise without considering any fault-tolerance capability. Some definite specifications of the baseline microprocessor are given in Table 3.3.

A simplified block diagram of the BL microprocessor is shown in Figure 3.12. The datapath is divided into five pipeline stages namely; Instruction Fetch (IF) stage, Instruction Decode (ID) stage, Execution (EXE) stage, Memory (MEM) stage and Write-back (WB) stage with their corresponding CL blocks, separated by five pipeline registers labeled as ①. The microprocessor incorporates three memory blocks, which include the asynchronous instruction memory labeled as ②, synchronous read-write access register-file with simultaneous read-write access labeled as ③ and synchronous data memory labeled as ④.

All the signals shown in Figure 3.12 are multiple bits signals and single bit signals like clock and other control signals are not shown for simplicity. There are some feedback signals like *next instruction address* signals labeled as ⑤, *branch address signals* labeled as ⑥, *wirte-back signals* labeled as ⑦ and *data-forwarding signals* labeled as ⑧. The datapath control logic is included in ID CL, which generates more feedback and feed-forward signals not shown in Figure 3.12 for the sake of simplicity.

It can also be noticed that the CL blocks are surrounded by regions of different colors. These colored regions represent the type of each CL according to the classification given in Section 3.2.2. The CL blocks surrounded by regions shaded in blue are classified as *influenced*, as they receive feedback signals from other *influencing* stages. For example, IF stage gets *branch address signals* from ID stage. This makes the IF stage an *influenced* stage and ID stage an *Influencing* stage represented by the orange shaded region around it. At the same time the ID stage is also an *influenced* stage because if recieves *Write-back* signals from the WB stage. EXE and MEM stages are also influenced by WB stage. Furthermore the MEM and WB stages are *unbounded*

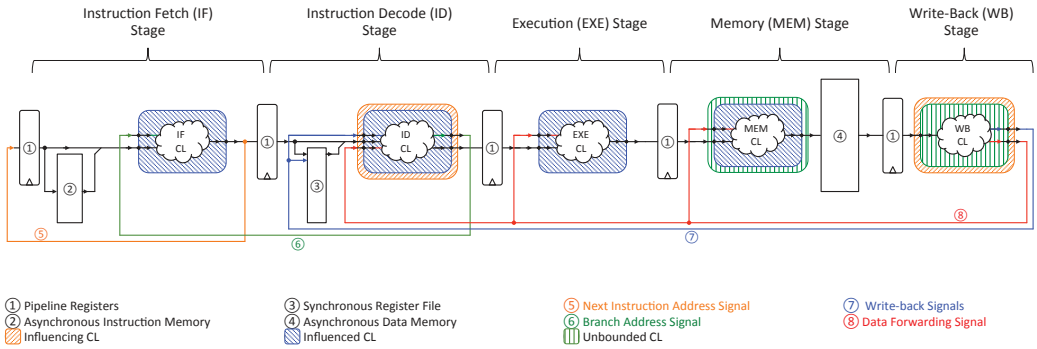


Fig. 3.12: Baseline Microprocessor

stages, because they generate signals that go to the asynchronous data memory and synchronous register-file respectively.

3.3.2 HyTFT Microprocessor

In Section 2.2.7 we discussed the duplication and comparison based HyTFT architecture capable of tolerating the effects of transient faults and detecting permanent and timing errors in stand-alone circuits. Later in Section 3.2 we formulated the principles of PHyFT architecture to extend the usability of HyFT architecture to complex pipeline circuits. Utilizing these principles we realized the HyTFT microprocessor, represented by a simplified block diagram in Figure 3.13.

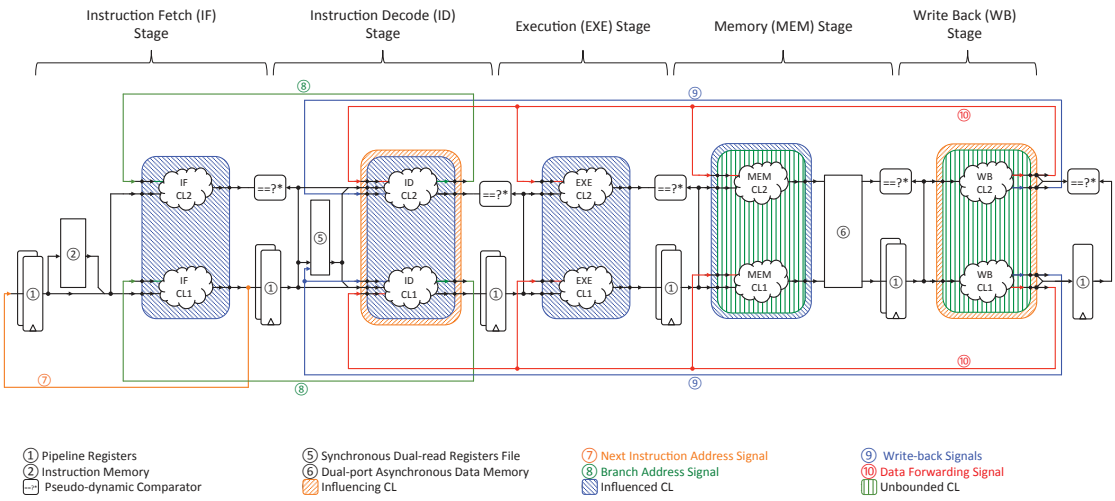


Fig. 3.13: HyTFT Microprocessor

It can be noticed that not only the CL blocks in each stage, but also the pipeline feedback signals are duplicated. The feedback connection duplication allows the detection of errors that propagate through these feedback connection from *influencing* stages to *influenced* stages as distinguished by orange and blue shaded regions respectively in Figure 3.13.

A practical utilization of a *dual-port write transparent* memory can be seen in the MEM stage of the HyTFT microprocessor. The block labeled as ⑥ represents the asynchronous data memory with two ports and incorporated write transparency property. Any transient, permanent or timing faults occurring in the *unbounded* MEM stage that could potentially become latent in memory can be detected by the following pseudo-dynamic comparator.

As discussed in Section 3.2.4, in order to detect errors on the synchronous memory interface we placed an additional set of pipeline register and a pseudo-dynamic comparator on the signals that go from the *unbounded* WB stage to the synchronous register-file, the block labeled as ⑤.

Unlike the asynchronous data memory and the synchronous register-file, the asynchronous instruction memory gets its inputs from a pipeline register and not form a CL block. Since the context of this thesis does not encompasses the problem of SEUs caused due to faults in flip-flops, we do not employ any special technique to protect the asynchronous instruction memory from such problems.

3.3.3 HyTPFT Microprocessor

Starting with the BL microprocessor, the Hybrid Transient and Permanent Fault Tolerant (HyTPFT) microprocessor was realized by triplicating CL blocks, duplicating feedback and feed-forward signals, inserting reconfiguration switches, modifying pipeline registers to incorporate rollback capability and by adding pseudo-dynamic comparators and acHyTPFT control logic blocks. An architectural overview of the resulting structure is shown in Figure 3.14. Clouds represent CL between pipeline registers. Three different memories are embedded in the microprocessor structure which are instruction memory, register-file and data memory. It can be seen that the data memory has two ports and thanks to its write transparency, any fault in the MEM stage CL block can be detected by the following comparator as elaborated in Section 3.2.4. It can also be noticed in Figure 3.14 that an additional set of pipeline register and comparator is placed after the WB stage. These additional components are there to detect errors on write-back signals feeding the simultaneous read-write access synchronous register file memory as we saw in Section 3.2.4.2.

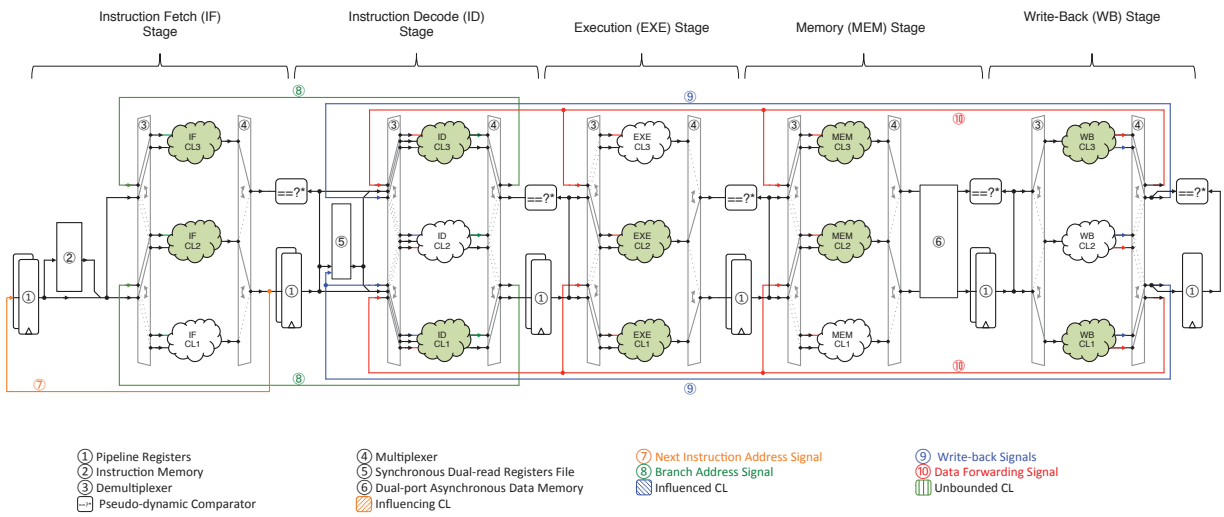


Fig. 3.14: HyTPFT Microprocessor

3.3.4 TMR-b Microprocessor

The TMR-b microprocessor is realized by triplicating the CL blocks in each stage of the pipeline and using a bit-wise voter that selects one CL output, which has at least one common equivalent, to be feed to the next pipeline register. The resulting scheme is represented by the simplified block diagram of Figure 3.15. In this way, each stage is dealt within its scope irrespective of its type (i.e. *bounded*, *influencing*, *influenced* or *unbounded*). In case of *unbounded* stages, since the voter masks all the types of error before they can potentially be stored in the memory, there is no need to use special memories to detect errors at the input of the memories.

3.3.5 TMR-w Microprocessor

The TMR-w microprocessor also has the same architecture as of TMR-b, shown in Figure 3.15. The only difference is the type of voter used. TMR-w microprocessor uses a word-wise voter that makes the voting decision based on the equivalence of entire words instead of individual bits. The word-wise voter has an additional output *error* which turns to logic-1 if the vote is impossible (i.e. no equivalent couple of input words exists).

3.3.6 Experimental Results

Experimental results in terms of area, power, fault tolerant capability and performance were obtained by first preparing Register Transfer Level (RTL) models of the five microprocessors

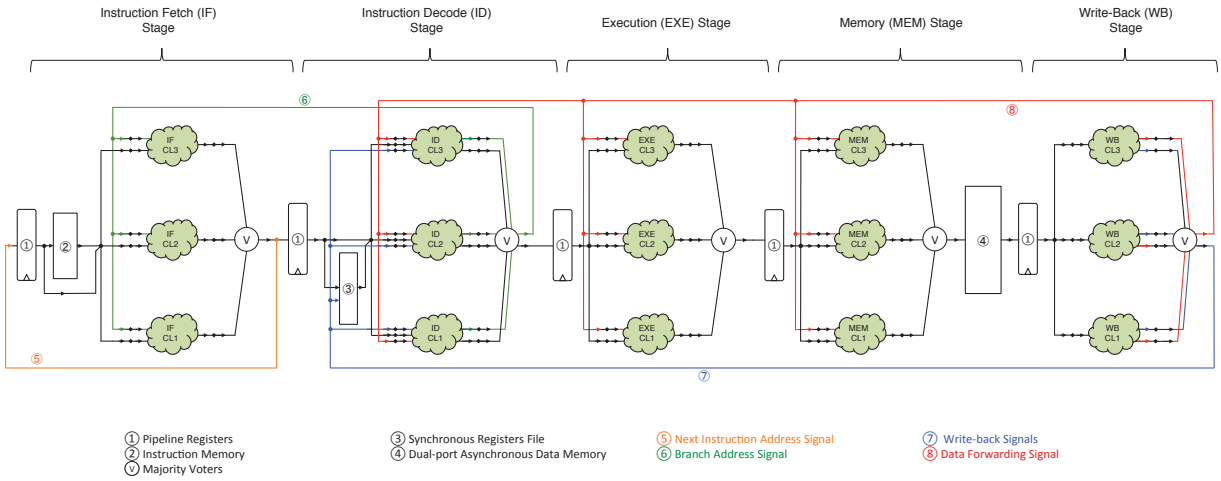


Fig. 3.15: TMR-b and TMR-w Microprocessor

Table 3.4: Fault injection parameters

Parameter	Transient faults	Permanent fault	Timing fault
Fault model	Temporary Stuck-at	Stuck-at	Interconnect-delay
Injection duration range	0.25 ns - 1.25 ns (Constrained random)	-	-
Timing error range	-	-	2 ns - 6 ns (Constrained random)
No. of injections	10,000		
Simulation duration	510 ns (per injection)		
Workload	Add-shift multiplication program (Provided in Appendix A)		
Fault location constraint	CL blocks (Constrained random)		
Injection time range	50 ns - 420 ns (Constrained random)		
Injection type	Single fault		

version discussed in previous subsections. These models were then synthesized hierarchically using Synopsys® Design Compiler with a 45nm technology library [67]. The area estimates were obtained from the synthesized gate-level design models and the average power estimates were obtained from back-annotated gate-level simulations of workload program (Provided in Appendix A) that multiply two 16-bit operands using shift-add method. The fault injection framework discussed in Section 2.3.3 was used to perform transient, permanent and timing fault injection experiments with parameters given in Table 4.5a. Fault tolerance capability and performance overhead results were deduced from the fault injection results.

3.3.6.1 Area Overhead

The area overhead figures obtained for the five microprocessors are given in Table 3.5 along with the individual share of each component making up the microprocessor. The first column lists

these design components and the subsequent columns give the absolute value of area and also the percentage share of each component in comparison with the corresponding component in the BL microprocessor. Most of the components show obvious amounts of increase when compared with BL for example, the CL blocks of HyTFT microprocessor are twice as large as those of the BL. The rest of the microprocessor versions employ triplication and thus, their CL blocks consume three times in comparison with BL.

Besides the CL blocks HyFT microprocessors have memories that consume 102% area of that of BL. This increased area is accounted to the dual-port write-transparency feature of the data memory in HyFT microprocessors. This area share is 100% for the two TMR microprocessors because they do not employ these features in their data memory. The HyFT microprocessors not only incorporate shadow latch arrays in their pipeline register but also use an additional pipeline register for error detection at the inputs of the synchronous simultaneous read-write access register file memory, therefore the area occupied by their pipeline registers is estimated to be 216% with respect to the pipeline registers in BL, TMR-b and TMR-w microprocessors.

Table 3.5: Area overhead results

	BL	**HyTFT**		**HyTPFT**		**TMR-b**		**TMR-w**	
	Area (μm^2)	*Area* (μm^2)	*%. of BL (%)*	*Area* (μm^2)	*%. of BL (%)*	*Area* (μm^2)	*%. of BL (%)*	*Area* (μm^2)	*%. of BL (%)*
Memory	26400	27039	102	27039	102	26333	100	26333	100
Pipeline Registers	2573	5548	216	5547	216	2574	100	2574	100
IF CL Blocks	4490	8980	200	13470	300	13470	300	13470	300
ID CL Blocks	6764	13528	200	20291	300	20291	300	20291	300
EXE CL Blocks	7434	14868	200	22203	300	22203	300	22203	300
MEM CL Blocks	1210	2420	200	3630	300	3630	300	3630	300
WB CL Blocks	1397	2794	200	4190	300	4190	300	4190	300
HyTFT Ctrl. Logic	-	237	-	-	-	-	-	-	-
HyTPFT Ctrl. Logic	-	-	-	727	-	-	-	-	-
Pseudo-dyn. Comparator	-	273	-	272	-	-	-	-	-
Voter-b	-	-	-	-	-	1538	-	-	-
Voter-w	-	-	-	-	-	-	-	3065	-
HyTPFT Mux. & Demux.	-	-	-	4758	-	-	-	-	-
Total	**50268 µm^2**	**75686 µm^2**	**151%**	**102127 µm^2**	**203%**	**94229 µm^2**	**187%**	**95756 µm^2**	**190%**

Table 3.5 also gives the area occupied by some components that are associated with one architecture only. Among these, an important figure to be compared between the four fault tolerant microprocessors is the area of their error detection hardware. We can see that the area of the pseudo-dynamic comparators is only 273 μm^2, which is significantly less than the area of bit-wise voters and word-wise voters, which are 1538 μm^2 and 3065 μm^2 respectively.

The last row of Table 3.5 gives the total area figures and their percentage with respect to the total area of the BL microprocessor. According to these figures the most efficient candidate in terms of area is the HyTFT microprocessor. But its fault tolerance capability is limited to transient faults only. TMR-b and TMR-w microprocessors have area that is 187% and 190% of the BL microprocessors respectively. The most area consuming candidate is HyTPFT microprocessor which stands at 203% of the area of our reference microprocessor.

Figure 3.16 gives a summary of overhead cost in terms of area. The height of each bar represent the total area of each microprocessor version. In each bar different colored regions represent the area occupied by individual components of the microprocessor. On the top, the total area in μm^2 is reported. The area overhead of HyTPFT microprocessor is 103% with respect to the BL microprocessor. This area is less than three times, since only the CL parts are triplicated in comparison with the basic approach [27], triplicating the entire microprocessor structure (including the memory). The TMR microprocessor versions incur lesser area overhead of 87% and 90% respectively for TMR-b and TMR-w. In comparison with HyTPFT this reduction is mainly due to the absence of reconfiguration multiplexers and demultiplexers and rollback shadow latches.

3.3.6.2 Power Overhead

Table 3.6 shows the power consumption results with a similar emplacement of area figures as in Table 3.5. Starting with the power consumption of memory, we see that the HyFT microprocessor versions consume slightly over than the memories in BL and TMR microprocessors, which is due to the write-transparency and secondary additional port of data memory. The HyFT pipeline registers, which on one hand occupy 216% area, consume 328% of power with respect of BL pipeline registers. This difference of increase rate is due to the fact that, incorporating shadow latches increase the switching activity in pipeline register with a larger ratio than the increase in area. Therefore we see a significant increase in power consumption of pipeline registers in HyFT microprocessors.

The power consumption in the TMR microprocessors CL blocks is close to three times of that of BL microprocessor. But in the case of HyTFT and HyTPFT this figure is slightly more that twice of that of BL CL blocks. The power consumption in HyFT CL blocks is due to the

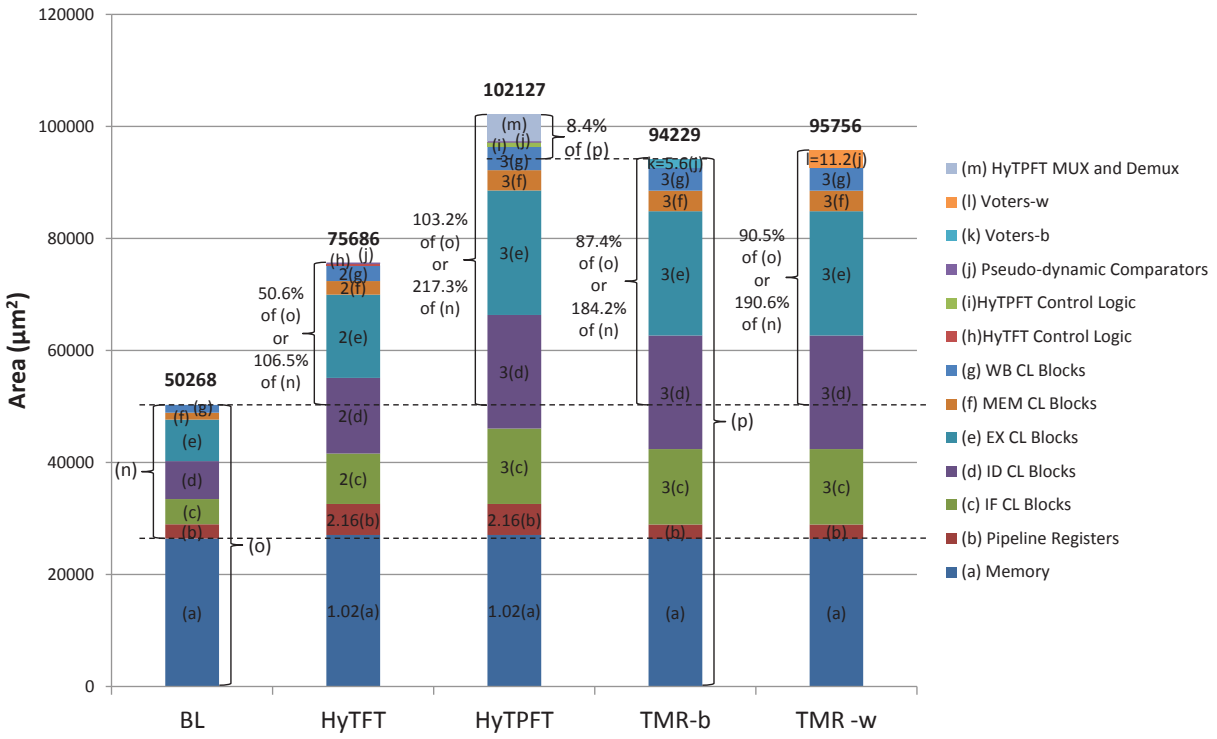


Fig. 3.16: Area overhead results summary

static and dynamic components of power in the two active CL blocks per stage plus the static component of power consumption of the stand-by CL block in each stage. The table shows that the pseudo-dynamic comparators consume significantly lower than the TMR-b and TMR-w voters.

Figure 3.17 gives the summary of estimations of average power consumption for different versions of the microprocessor. In each bar different shaded regions represent the power consumption share of different components of microprocessors. On the top the total average power in mW dissipated by each microprocessor version is reported. The power consumption overhead of HyTPFT microprocessor is 105% with respect to BL. This power is 11.6% and 11.8% less than that consumed by the TMR-b and TMR-w versions respectively.

Table 3.6: Power overhead results

	BL	**HyTFT**		**HyTPFT**		**TMR-b**		**TMR-w**	
	Avg. Power (mW)	*Avg. Power (mW)*	*%. of BL (%)*	*Avg. Power (mW)*	*%. of BL (%)*	*Avg. Power (mW)*	*%. of BL (%)*	*Avg. Power (mW)*	*%. of BL (%)*
Memory	1.319	1.329	101	1.329	101	1.324	100	1.324	100
Pipeline Registers	0.306	1.005	328	1.005	328	0.316	103	0.316	103
IF CL Blocks	1.382	2.719	197	2.935	212	4.145	300	4.145	300
ID CL Blocks	0.958	1.880	196	2.077	217	2.873	300	2.883	301
EXE CL Blocks	0.463	0.944	204	1.117	241	1.410	305	1.410	305
MEM CL Blocks	0.077	0.152	197	0.200	260	0.230	299	0.230	299
WB CL Blocks	0.192	0.422	220	0.449	234	0.579	302	0.582	303
HyTFT Ctrl. Logic	-	0.355	-	-	-	-	-	-	-
HyTPFT Ctrl. Logic	-	-	-	0.376	-	-	-	-	-
Pseudo-dyn. Comparator	-	0.026	-	0.026	-	-	-	-	-
Voter-b	-	-	-	-	-	0.037	-	-	-
Voter-w	-	-	-	-	-	-	-	0.040	-
HyTPFT Mux. & Demux.	-	-	-	0.117	-	-	-	-	-
Total	**4.697 mW**	**8.832 mW**	**188%**	**9.631 mW**	**205%**	**10.900 mW**	**232%**	**10.921 mW**	**233%**

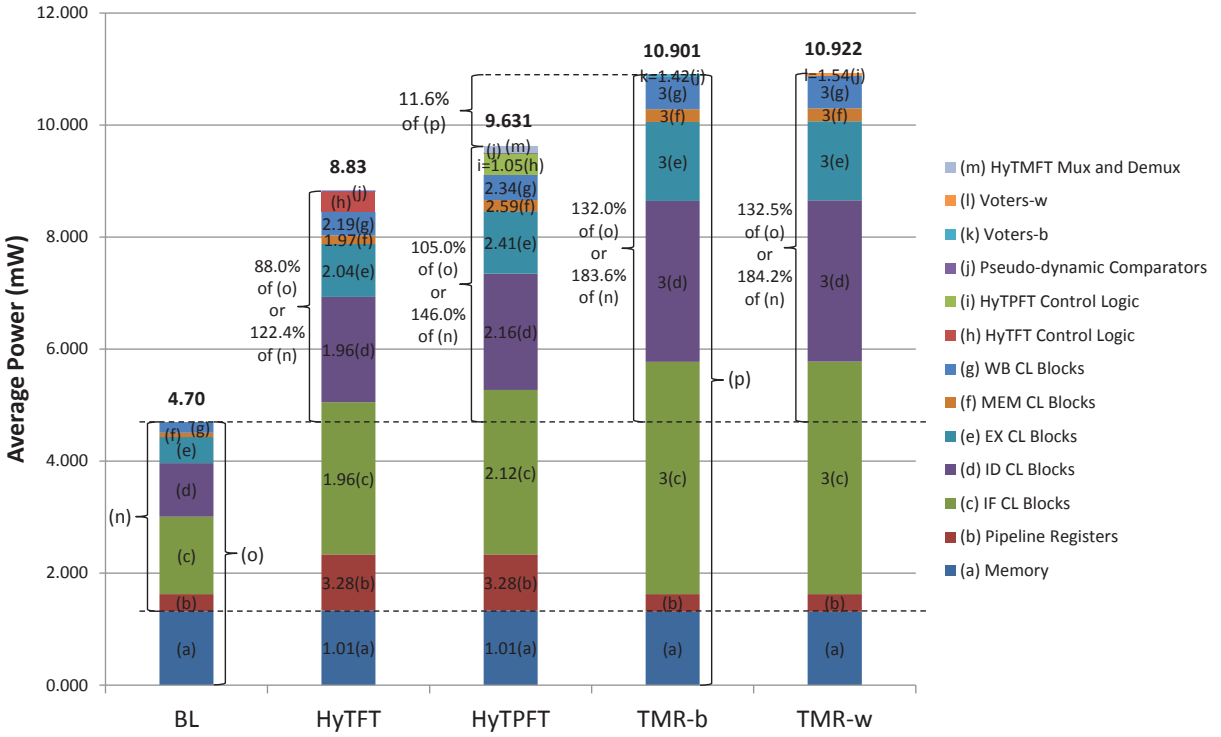
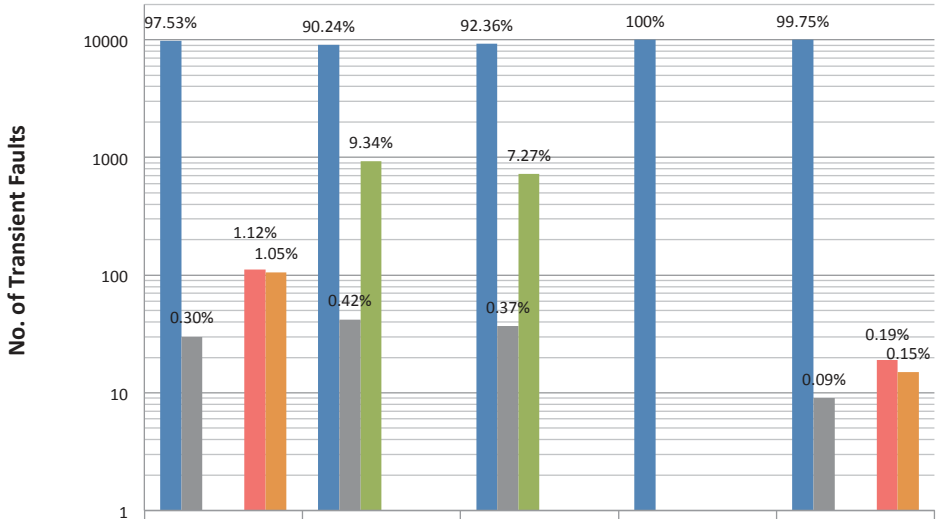


Fig. 3.17: Power overhead results summary

3.3.6.3 Fault Tolerant Capability

The results of fault tolerance capability, the key merit for comparison of the four considered fault tolerant architectures, are presented in this section. Figures 3.18, 3.19 and 3.20 show the transient, permanent and timing fault-injection results respectively for the five different versions of microprocessor. The bar charts in these figures represent the population distribution of injected faults according to the categories discussed in Section 3.3.6.3 on the basis of their outcome. The bars of each color represent a fault class and the their height represent the number of faults found to fall in that fault category on a logarithmic scale. On the top of each bar, the percentage share of the number of faults in the corresponding fault class in the total of 10,000 injected faults is given. The table attached at the bottom of the bar chart gives the exact number of faults found to fall in each category.

Now if we focus on the transient fault injection results shown in Figure 3.18, we can see that in the case of BL microprocessor out of 10,000 injected transient faults 112 and 105 faults resulted in *fault-silent* and *latent* outcomes respectively which collectively makes 2.17% of the total injected faults. These faults critically affected the computation of BL microprocessor. Whereas 0.3% of the faults could not be classified and 97.53 % of the injected faults had no effect on the program

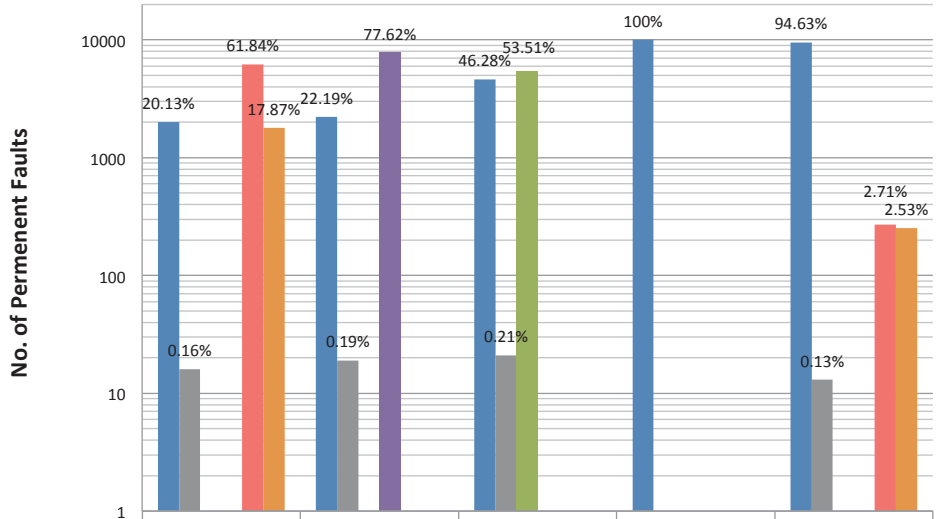


	BL	HyTFT	HyTPFT	TMR-b	TMR-w
Silent Faults	9753	9024	9236	10,000	9975
Undetermined Faults	30	42	37	0	9
Corrected Faults	0	934	727	0	0
Time-out Faults	0	0	0	0	0
Fail-Silent Faults	112	0	0	0	19
Latent Faults	105	0	0	0	15

Fig. 3.18: Transient fault injection results

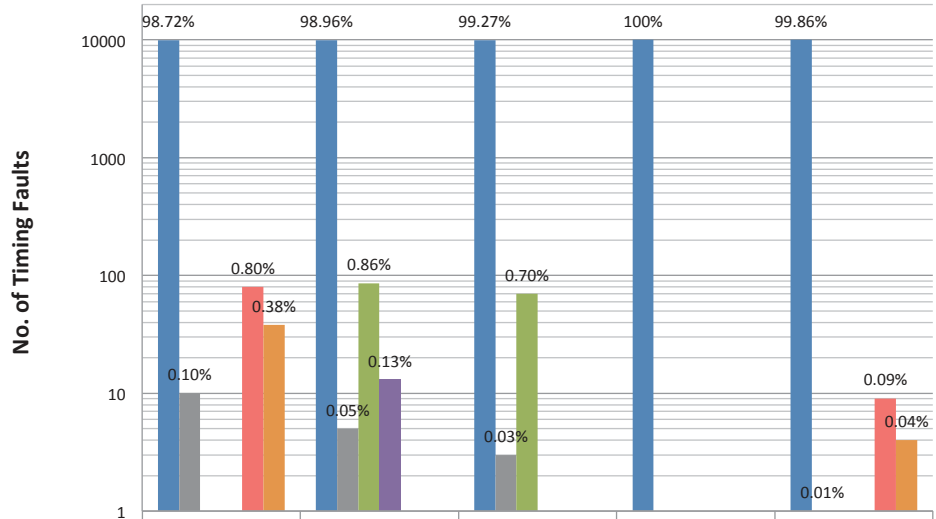
execution nor on the results. This percentage drops down to 90.24% and 92.36% in the case of HyTFT and HyTPFT microprocessors respectively but this drop is compensated by the rise in *corrected* faults to 9.34% and 7.27% in HyTFT and HyTPFT microprocessors respectively, as both the categories represent faults that do not cause an incorrect operation nor erroneous result. Unlike BL microprocessor, HyTFT and HyTPFT microprocessors did not undergo any critical failures. However, because of the absence of a discretely observable error flag that can be captured on periodic intervals equivalent to the time period of operation, in TMR-b and TMR-w voters, it is impossible to distinguish between the *silent* and *corrected* faults. Thus both of these type of faults are kept in the category of *silent* faults. An important observation that can be made from the results of TMR-w microprocessor is the presence of 0.19% and 0.15% of *fail-silent* and *latent* faults respectively. This suggests that among the four fault tolerant microprocessors TMR-w is least effective in dealing with transient faults.

The severity of the impact of permanent faults on the baseline microprocessor is evident from the high number of faults falling into the category of *fail-silent* and *latent* as shown in graph of Figure 3.19. HyTPFT and TMR-b seem to have dealt well with the situations of permanent faults by having none of the faults falling in the category of critical faults. The incorporation of TMR-w architecture in the BL microprocessor seems to have reduced the number of critical failure



	BL	HyTFT	HyTPFT	TMR-b	TMR-w
Silent Faults	2013	2219	4628	10,000	9463
Undetermined Faults	16	19	21	0	13
Corrected Faults	0	0	5351	0	0
Time-out Faults	0	7762	0	0	0
Fail-Silent Faults	6184	0	0	0	271
Latent Faults	1787	0	0	0	253

Fig. 3.19: Permanent fault injection results



	BL	HyTFT	HyTPFT	TMR-b	TMR-w
Silent Faults	9872	9896	9927	10,000	9986
Undetermined Faults	10	5	3	0	1
Corrected Faults	0	86	70	0	0
Time-out Faults	0	13	0	0	0
Fail-Silent Faults	80	0	0	0	9
Latent Faults	38	0	0	0	4

Fig. 3.20: Timing fault injection results

occurrences from 79.71% collectively to only 5.24% but still does not protect against permanent faults as fully as the HyTPFT and TMR-b microprocessors. As per its intended purpose, the HyTPFT microprocessor successfully detected 53.51% of the injected permanent faults, which makes the entire set of observable faults because all the rest were either masked and never showed up or resulted in an *unclassifiable* outcome. None of the injected permanent faults in HyTFT microprocessor escaped detection to cause either *fail-silent* or *latent* errors. This suggests that HyTFT microprocessor maintained fail-safe operation throughout the fault injection campaign.

The timing fault population distribution shown in Figure 3.20 shows a similar trend as for permanent faults but with lesser degree of effect on the computation. The BL microprocessor encountered 80 and 30 *fail-silent* and *latent* errors respectively, which is more or less the number of faults detected and/or corrected by HyTFT and HyTPFT microprocessor respectively. Since the timing faults injected were of permanent nature, the HyTFT repeatedly tried to recover form 13 *time-out* faults. However, TMR-w ended up in 13 critical outcomes.

From the results of three types of fault injection campaigns, we saw that the only architecture, which never resulted in an outcome that cannot be classified in TMR-b. As discussed in Section , these *unclassifiable* outcomes are due to the *X*-propagation that results because of setup or hold violations. The reason of not having any *unclassifiable* outcomes in TMR-b is associated with the property of TMR-b voter that it can mask these *X*-propagations and preventing it from spreading throughout the execution. However the small number of *unclassifiable* faults outcomes in other microprocessor versions does not have any impact on the conclusion that can be drawn from these results because *X*-propagations occur only in simulations. In real silicon implementation a setup or hold violation will result in either a correct computed value or a wrong one. In case a correct value is stored, there will be no impact on the execution nor on the data. Thus will result in *silent* outcome. But if the violation cause wrong data to be stored, the error detection and recovery mechanism in HyTFT and HyTPFT can deal with it and the fault will result in a corrected fault category.

3.3.6.4 Performance Degradation

There are two aspects in which the performance overhead of fault tolerant architecture can be assessed:

- Temporal performance degradation
- Error recovery overhead

The additional components inserted in the BL data-path to implement the HyTPFT architecture, which include reconfiguration multiplexers and demultiplexers and a level of multiplexers in the

pipeline registers for rollback capability, are accounted for the temporal performance degradation. Conversely, in the TMR schemes the voter circuit in the data-path is responsible for reducing circuit speed. STA showed that for HyTPFT microprocessor temporal performance degradation is 3.8%. This figure is obtained by comparing critical path delays of CL blocks with the maximum delay of aforementioned additional HyTPFT circuit components. In the same way the percentage of temporal performance degradation were obtained for TMR-b and TMR-w and were found to be 0.9% and 8.5% respectively. These figures show that the HyTPFT architecture incurs less temporal performance degradation then TMR-w but is more costly in comparison with TMR-b.

Performance overhead due to error recovery can be comprehensively measured by performing multiple faults injection campaign and observing the impact on performance due to different error rates. However our single fault injection experiments show that all the errors in HyTPFT microprocessor due injected SETs, 51.8% of the errors due to permanent faults and 77.1% of errors due to delay faults were recovered in a time equivalent to 2 clock cycles. The remaining errors due to permanent and delay faults were mitigated with a penalty of 4 cycles. On the other hand, TMR microprocessor versions have zero error recovery penalties, as TMR is an error masking scheme rather than an error detection and correction architecture.

3.3.7 Discussion

From the experimental results presented in this section we can infer that the PHyFT architecture offers a method to improve the robustness of CL parts of complex nonlinear pipeline processor cores, with little impact on performance and quite modest amount of overhead in terms of area with respect to TMR. Both the HyTFT and HyTPFT microprocessors subjected to a fault injection campaign of 30,000 faults in their CL parts sustained correct of fail-safe operation and offer similar fault tolerance capability as TMR-b scheme with the added advantage of 11.6% power saving.

3.4 Summary

In this chapter we have seen some issues in modern complex pipeline circuits that make error detection and confinement a difficult task. We briefly overviewed some state-of-the-art solutions to these problems and their limitations. In the second part of the chapter we went through the steps to develop a Pipelined Hybrid Fault Tolerant architecture based on the Hybrid Fault Tolerant architecture for stand-alone circuits by proposing solutions to the hurdles faced in doing so. In the last section of this chapter we showed how we practically applied the principles of PHyFT architecture formulated in preceding sections, on a case study instance of MIPS architecture

microprocessor. Finally we discussed the experiments performed to assess and compare PHyFT architecture with other solutions found in literature.

Chapter 4

Design Space Exploration and Optimization of HyFT Architecture

The case study outcomes in the previous chapter showed that the HyFT architecture can effectively tolerate errors due to transient, permanent and timing faults in the CL parts of complex pipeline circuits with an 11.6% average power saving compared with the TMR architecture. However, it is identified that there is further room for improvement in its efficiency in terms of area, power and performance, by optimizing the *comparison-window* timing. Furthermore, its applicability can also be improved by adapting it to work with symmetric (50%) duty-cycle clock.

In this chapter we will first analyze some properties of HyFT architecture, which limit its area and power saving, performance (in terms error recovery overhead) and applicability. Later, we will discuss some design and timing optimizations and based on these modifications we will describe four experimental design variants of HyFT architecture. We will analyze the impact of these modifications on the considered design merits. We will also verify that the proposed modifications do not have an adverse impact on the fault tolerance capability. Finally, based on the experimental results we will define the best candidate among the four design variants.

4.1 Limitations of HyFT Architecture

4.1.1 Contamination delay constraints

In Section 2.2.7 we studied the timing constraints needed to be imposed while synthesizing the CL blocks to be hardened with HyFT architecture. We saw that to prevent the asynchronous input of the pseudo-dynamic comparator from starting to change before the lapse of the *comparison-*

window it is necessary to make sure that there are no paths short enough to violate the condition of Equation 5.1.

Commercial synthesis tools generally provide provision to apply minimum delay constraints to avoid hold-time violations. These constraints are less stringent than those needed for HyFT CL synthesis. Synthesis tools use techniques like buffer insertion, circuitous routing, gate or wire re-sizing [83], [106] to increase the delay of short paths in the circuit. When these tools are used for tighter minimum delay constraints to considerably alter the path-delay distribution, as in the case of HyFT architecture, the resulting circuits cost significantly higher in terms of silicon space and power.

4.1.2 Asymmetric clock duty-cycle

In designs that use both, negative and positive edge triggered flip-flops, a symmetric clock duty-cycle is important to ensure easy timing closure. Even for designs that use just one polarity of clock to capture data, an asymmetric clock duty-cycle reduces the maximum clock frequency of operation because as the pulse becomes narrower the bandwidth of the signal spectrum increases. Furthermore, the high frequency components present in non-50% duty-cycle signal increase clock induced noise and interference.

As we have seen that HyFT architecture rely on an asymmetric clock duty-cycle to generate necessary control signals, which limits its applicability to circuits that work on both, positive and negative edges and those used for high performance applications. Thus, by removing the dependency of HyFT control signal generation on the system clock duty-cycle we can improve its applicability.

4.2 Design and Timing Optimization

The optimization efforts are aimed at reducing the area and power overhead due the minimum contamination delay requirements applied to the CL blocks. As shown in Equation 5.1, these timing requirements can be eased by reducing δt, which can be achieved by shortening and/or moving the *comparison-window* closer to the *CLK* edge. To study the impact of *comparison-window* timing on the four design metrics, i.e. area, power, fault tolerance capability and error recovery overhead, we selected two design parameters to be controlled, which include:

- *Comparison-window* position (with reference to *CLK* edge)
- *Comparison-window* width

Controlling these parameters we obtain four HyFT design variants including the original scheme (i.e. without any optimization). These HyFT design variants are detailed in the following subsections.

4.2.1 Original HyFT (HyFT-1a) Architecture

This is the original HyFT architecture which is designed without considering any timing optimization. We use this scheme as our reference to access the improvements. As discussed in Section 2.2.7, it uses a simple delay buffer (shown in Figure 4.1a) to generate *DC* from *CLK* signal as represented by the timing diagram in Figure 4.1b. The start and the end of the *comparison-window* are delayed by an amount of time equal to the delay of the buffer (d_{buf}), from the rising and falling edges of *CLK* respectively. Since the *DC* pulse width depends on the *CLK* duty-cycle, it is less controllable and also imposes limits on the *CLK* pulse width. SPICE-like simulation graph obtained from Synopsys®'s Nanosim tool for *DC* and *CLK* signals in HyFT-1a architecture is shown in Figure 4.2a.

4.2.2 Hybrid Fault Tolerant (HyFT) Architecture with short *DC* (HyFT-1b)

The second experimental design variant uses a *DC* pulse narrower than the one used in HyFT-1a as shown in Figure 4.1d. It uses a circuit shown in Figure 4.1c to generate *comparison-window* that ends with the falling edge of the *CLK*. The start of the *comparison-window* is delayed by an amount of time equal to the delay of the buffer and the delay of the 'AND' gate ($d_{\text{buf}} + d_{\text{and}}$). While the end of the *comparison-window* is delayed from the falling edge of *CLK* by time equal to the delay of 'AND' gate (d_{and}) only. The SPICE-like simulation plot of *DC* generation in HyFT-1b is shown in Figure 4.2b.

4.2.3 Hybrid Fault Tolerant (HyFT) Architecture with *CLK* enclosed *DC* (HyFT-2a)

In HyFT-1a and HyFT-1b architectures the pseudo-dynamic comparator gets a synchronous input form the output registers as shown in Figure 2.15a. It implies that the comparison should not start until the FF outputs are stable. In other words, the FF clk-to-output delay imposes a limit while moving the comparison-window close to the *CLK* as shown in Figure 2.15b. To go beyond this limit, placement of the comparator at RT-level needs to be changed, such that the comparator now compares the output of two CL copies directly from the multiplexer, as shown in Figure 4.3. In this orientation the comparator gets to compare two signals that are both asynchronous

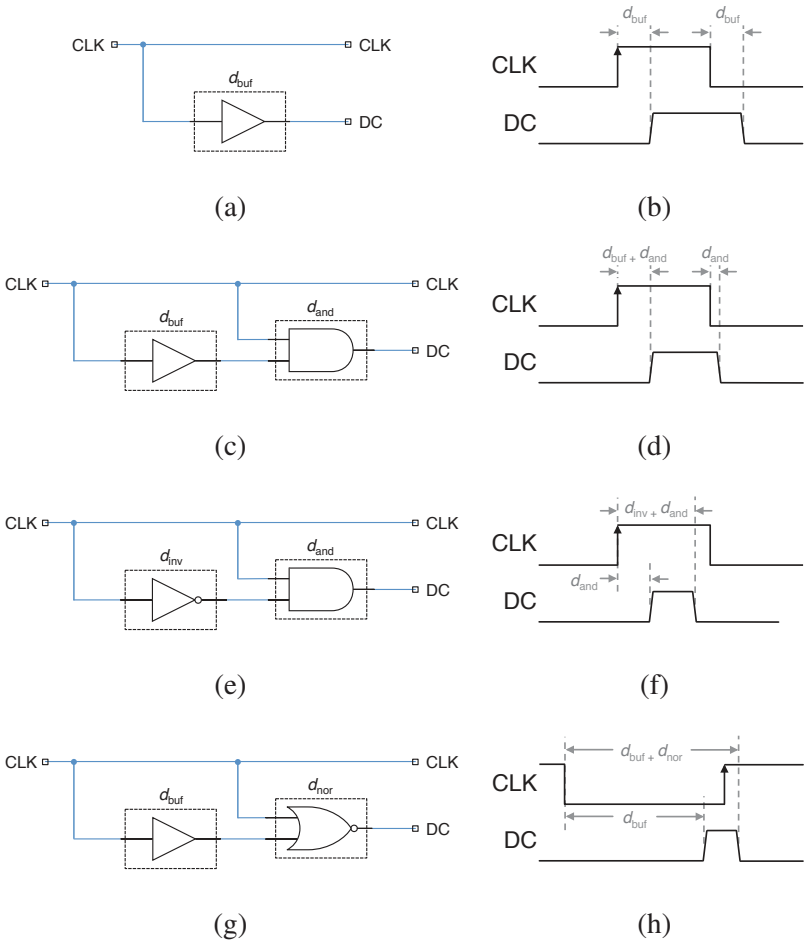
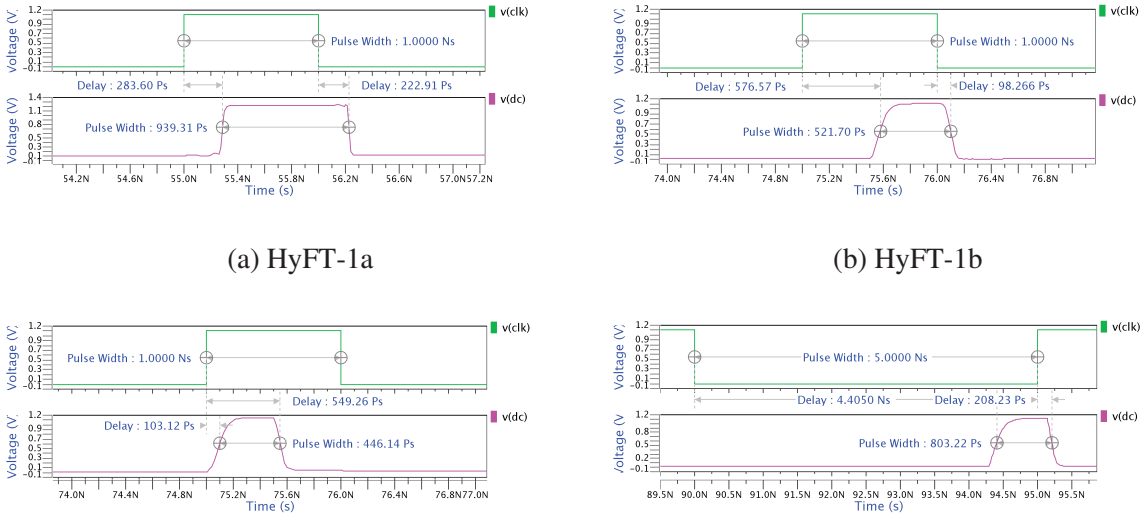


Fig. 4.1: *Comparison-window* Generation Circuits and delays

and allows it to use a *comparison-window* similar to one shown in Figure 4.1f. The circuit used for *DC* pulse generation is shown in Figure 4.1e. It can be seen in Figure 4.1f, the start of the *comparison-window* is delayed from the rising edge of *CLK* by an amount of time equivalent to the delay of the 'AND' gate (d_{and}) and the end of comparison-window is also delayed from the rising edge of *CLK* but by an amount equal to the cumulative delay of the inverter and the 'AND' gate ($d_{\text{inv}} + d_{\text{and}}$). The SPICE-like simulation graph of *DC* pulse in HyFT-2a is shown in Figure 4.1f. This scheme significantly reduces the amount of minimum required contamination delay as δt is now reduced.



(a) HyFT-1a

(b) HyFT-1b

(c) HyFT-2a

(d) HyFT-2b

Fig. 4.2: *Comparison-window* Timing Simulation Graph

4.2.4 Hybrid Fault Tolerant (HyFT) Architecture with *DC* across *CLK* edge (HyFT-2b)

This experimental design variant of HyFT is obtained by moving the comparison-window across the *CLK* capture edge as shown in Figure 4.2d. As in case of HyFT-2a, in this scheme also the comparator is placed before the pipeline register and gets its both inputs from the two running CL copies as shown in Figure 4.3. This scheme uses a 50% duty-cycle *CLK* with the logic circuit of Figure 4.1g, to generate *DC*. It can be seen in Figure 4.1h that the start of the *comparison-window* is a response of the falling edge of the *CLK* delayed by an amount of time equivalent to the delay of buffer (d_{buf}) and the end of *comparison-window* is also delayed form the falling edge of the *CLK* signal but with a delay equal to the cumulative delay of the buffer and the 'NOR' gate ($d_{\text{buf}} + d_{\text{nor}}$). The amount of δt for this variant is very minimal as it can be seen in the SPICE-like simulation graph of Figure 4.2d. When used with very narrow *comparison-window* completely renders the requirement of applying any minimum delay constraints because in that case the reconfiguration demultiplexer and multiplexer contribute sufficient contamination delay to satisfy the overall delay requirement.

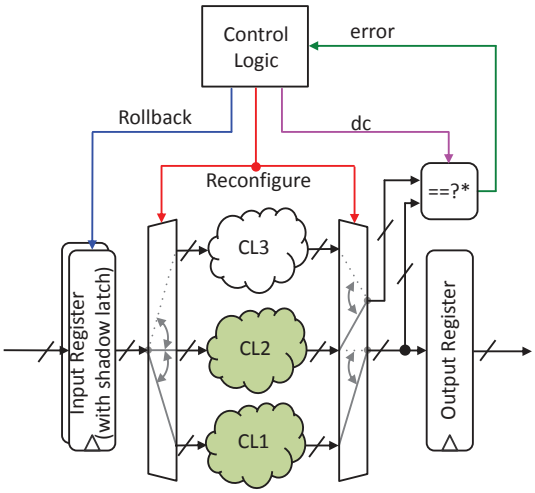


Fig. 4.3: Modified Hybrid Fault Tolerant (HyFT)

4.3 Experimental Assessment of Improvements

The four design variants discussed in previous subsections undergo a comparative experimental study in order to assess and compare their design merits. These merits include area, power, fault tolerance capability and error recovery overhead. This study gives us an idea about how the design merits change with *comparison-window* timing and comparator placement. The results of the comparative study allow us to select the best candidate.

4.3.1 Experimental Setup

Experiments are performed on a few of ITC'99 benchmark circuits [20] by hardening them with each of the four HyFT design variants and then assessing them for their design merits. The experimental methodology used can be broken down into the following steps:

4.3.1.1 CL Extraction

The first step in the experimental assessment is similar to the one explained in Section 2.4.1, in which we extract the CL from the original benchmark circuits. The step is also explained using an example in Appendix C.

4.3.1.2 CL Synthesis

In this step of the experimental assessment, the extracted CL blocks are synthesized using appropriate contamination delay constraints governed by the Equation 5.1, corresponding to

Table 4.1: Contamination delay constraints for CL synthesis

HyFT design variant	δt **(ns)**	$\delta t - t_{ccq} - t_{cdm} - t_{cm}$ **(ns)**	t_{cd} **(ns)**
HyFT-1a	1.2229	0.9976	1.00
HyFT-1b	1.0983	0.8730	0.88
HyFT-2a	0.5493	0.3240	0.33
HyFT-2b	0.2082	-0.0171	0.00

each HyFT design variant. As discussed in Section 4.2, δt is the design parameter that can be controlled to ease the contamination delay constraints. From the SPICE-like simulation graphs shown in Figure 4.2 we extract the value of δt for each HyFT design variant, which are mentioned in the second column of Table 4.1. The third column gives the lower limit of contamination delay constraints obtained by substituting the values of minimum clock-to-output delay of the FFs (t_{ccq}) to be 80 ps, minimum demultiplexer delay (t_{cdm}) to be 65.6 ps and minimum multiplexer delay (t_{cm}) to be 79.7 ps in Equation 5.1. These values are obtained from the technology library data and performing STA. The last column specifies the contamination delay constraints actually applied during the synthesis of CL blocks. It can be noticed in Table 4.1 that as we move the *comparision-window* back towards the *CLK* capture edge, it significantly reduces the minimum contamination delay requirements. Eventually, in the case of HyFT-2b the clock-to-output delay (t_{ccq}) of the flip-flops, the delay of reconfiguration demultiplexer (t_{cdm}) and multiplexer (t_{cm}) contribute enough that there is no need of applying any contamination delay constraints while synthesizing CL to be hardened using HyFT-2b architecture.

After synthesizing the CL blocks with contamination delay constraints corresponding to each of the HyFT design variant, we obtain area estimates which are given in Table 4.2. The first column lists the considered ITC benchmark circuits. The remaining columns are grouped according to the concerned HyFT variant. Since HyFT-1a serves as our reference the second column lists only the absolute area of CL blocks synthesized with corresponding contamination delay. While the other groups of columns, besides giving absolute area in μm^2 also gives the percentage area reduction with reference to the HyFT-1a CL blocks. From Table 4.1 and Table 4.2 we can deduce that with a 12% (from 1.00 ns to 0.88 ns), 67% (from 1.00 ns to 0.33 ns) and 100% (from 1.00 ns to 0 ns) reduction of contamination delay constraints there is a 11.3%, 82.1% and 90.3% reduction in CL area respectively. Although all the benchmark circuits show a significant gain in terms of area by easing the contamination delay but larger circuits seem to gain relatively less. The reason being that the larger circuits generally have a smaller percentage of very short paths, thus easing contamination delay has a less significant impact on their area.

Table 4.2: CL Synthesis Results

	BL CL	**HyFT-1a CL**	**HyFT-1b CL**		**HyFT-2a CL**		**HyFT-2b CL**	
	Area (μm^2)	Area (μm^2)	Area (μm^2)	Reduction (%)	Area (μm^2)	Reduction (%)	Area (μm^2)	Reduction (%)
b01	22.1	271.3	240.5	11.37	48.4	82.16	22.1	91.86
b02	14.9	210.4	186.7	11.25	35.9	82.93	14.9	92.92
b03	78.2	1052.8	923.5	12.28	162.8	84.54	78.2	92.57
b05	333.0	1863.3	1673.1	10.20	426.1	77.13	333.0	82.13
b06	33.2	471.3	416.8	11.57	71.5	84.82	33.2	92.95
b08	88.8	842.4	750.4	10.92	159.6	81.05	88.8	89.45
average	-	-	-	**11.268%**	-	**82.106%**	-	**90.314%**

4.3.1.3 Workload Generation

For power estimation and fault injection experiments the workloads used are Automatic Test Pattern Generator (ATPG) generated patterns optimized to detect stuck-at faults in the BL version of the CL blocks. The patterns contained in a *.dat* file are applied to the synthesized netlist by a test-bench also generated by the ATPG tool, in a gate-level simulation environment with back annotated delay information.

4.3.1.4 RTL Description

As we have seen in Section 4.2, the first considered HyFT design variant is in fact identical to the HyFT scheme discussed in Section 2.4.1. So, the RTL modules are reused to construct HyFT-1a circuits. The other three design variants bare two types of differences from HyFT-1a in their RTL descriptions. These differences are the comparator placement and the *comparison-window* timing. The former difference is easily made at RTL in the top-level wrapper modules of HyFT-2a and HyFT-2b architectures. The latter difference requires modifications to the control logic modules of HyFT-1b, HyFT-2a and HyFT-2b architectures. We have already seen a glimpse of these modifications in Section 4.2, where we discussed the *comparison-window* generator circuits. The remaining RTL components to these architectures are similar to those of HyFT. In this subsection we provide the details of the modified control logic modules to generate appropriate control signals for HyFT-1b, HyFT-2a and HyFT-2b architectures.

As architectures HyFT-1a, HyFT-1b and HyFT-2a are all driven by an asymmetric duty-cycle clock , they use a similar control logic *submudule1* with the only difference of their *comparison-window* generation components. Figure 4.4 shows the *submodule1* in the control logic of the HyFT-1b architecture. The labels on each logic element is its reference and the symbol on the top of each element represent its delay. These delays are critical and essential to be constrained during

the synthesis because a complete control of these delays is essential for the proper functionality of the HyFT architecture.

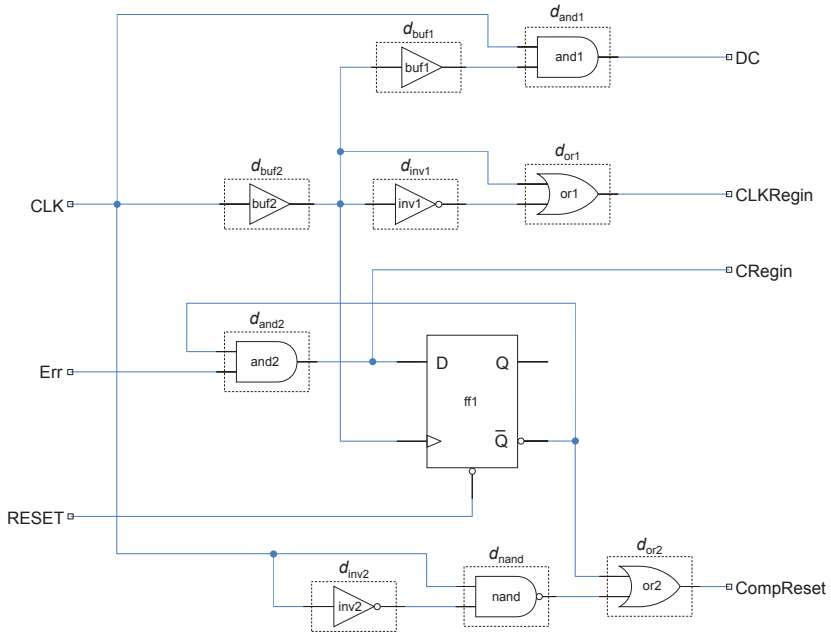


Fig. 4.4: HyFT-1b Control Logic *submodule1*

HyFT-1b control logic submudule1 is not much different from that of HyFT-1a. The only difference is the addition of an AND gate to end the *comparison-window* with the falling edge of the *CLK*. In Figure 4.4, *buf1*, *buf2* and *and1* form the *comparison-window* generation circuit also shown in Figure 4.1c. The simulation graph in Figure 4.5 shows the generated control signals with appropriate timing in the case of an error detection.

Similarly, circuit shown in Figure 4.6 and the simulation graph shown in Figure 4.7 represent the contol logic submodule and its timing characteristics respectively for the HyFT-2a scheme. It

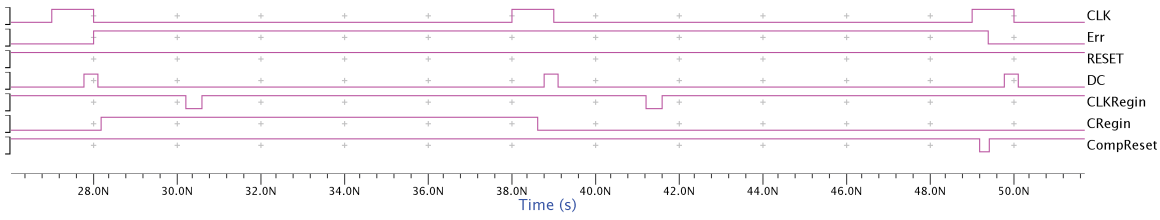


Fig. 4.5: HyFT-1b Control Logic *submodule1* simulation

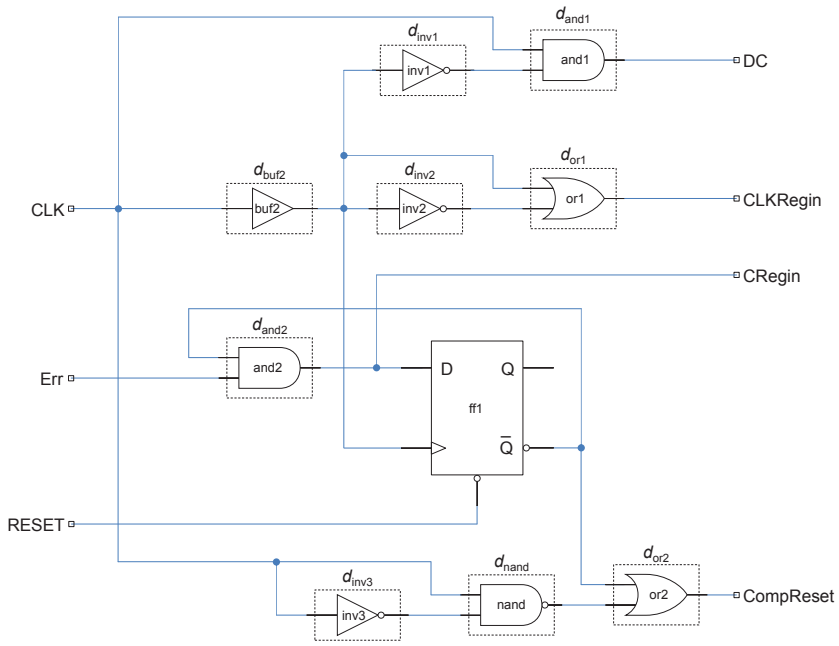


Fig. 4.6: HyFT-2a Control Logic *submodule1*

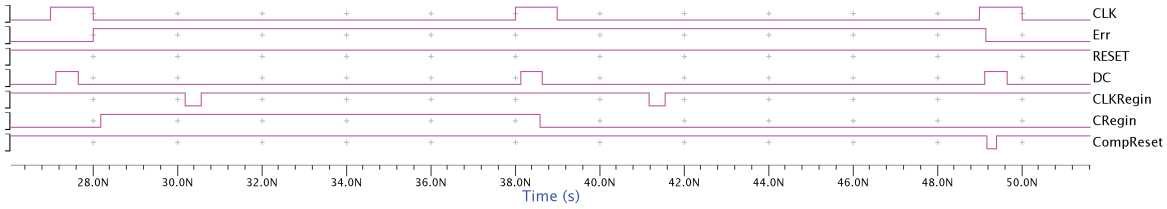


Fig. 4.7: HyFT-2a Control Logic *submodule1* simulation

can be noticed that the submodules of Figure 4.4 and 4.6 only differ in their *DC* pulse generation circuit.

The control logic *submodule1* shown in Figure 4.8 undergoes some significant modifications to allow the *comparison-window* overlap the setup-hold window. Its functionality is explained by the timing simulation graph of Figure 4.9.

4.3.1.5 Constrained Hierarchical Synthesis

The last step in the experimental setup is constrained hierarchical synthesis of the entire designs. A bottom-up synthesis approach is applied for the synthesis starting from the most time critical components like the control logic till the top-level HyFT wrappers. Comprehensive synthesis

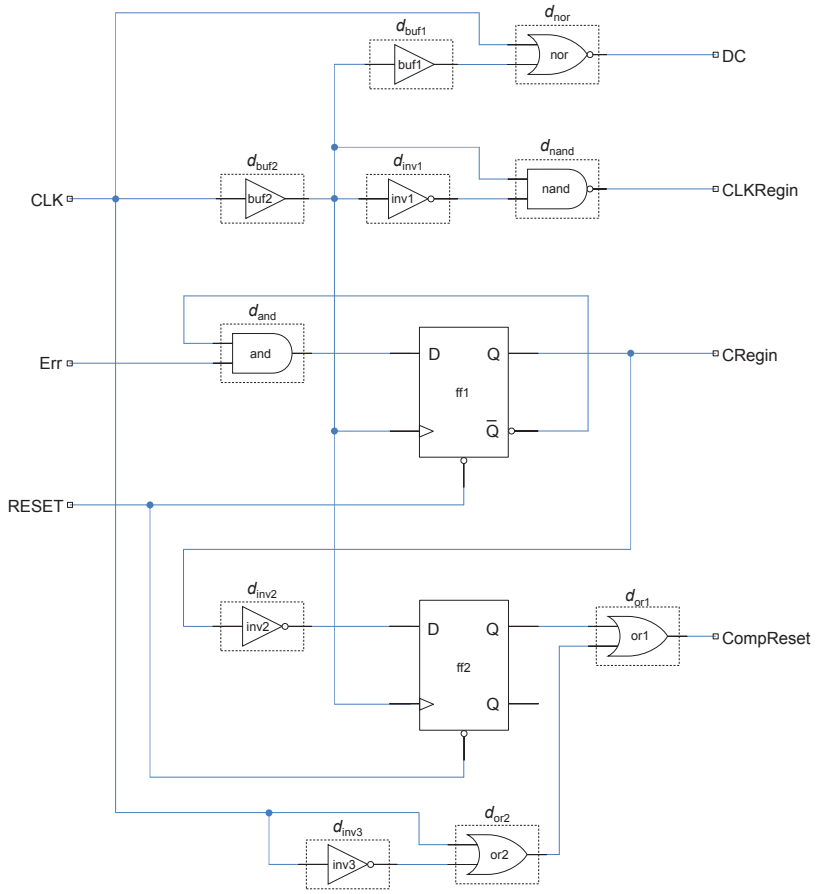


Fig. 4.8: HyFT-2b Control Logic *submodule1*

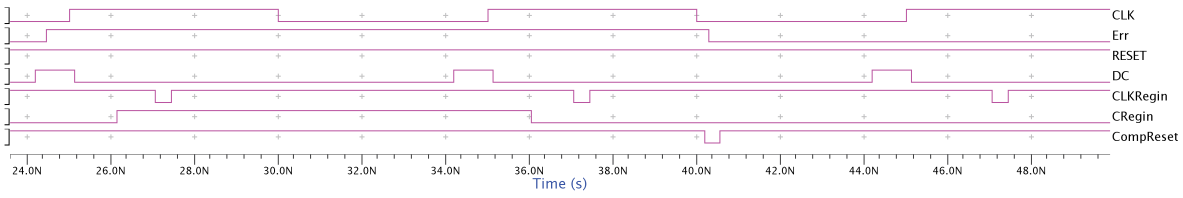


Fig. 4.9: HyFT-2b Control Logic *submodule1* simulation

Table 4.3: Area Savings

	HyFT-1a	**HyFT-1b**		**HyFT-2a**		**HyFT-2b**	
	Area (μm^2)	Area (μm^2)	Saving (%)	Area (μm^2)	Saving (%)	Area (μm^2)	Saving (%)
b01	1404.2	1176.2	16.23	613.9	56.28	627.0	55.35
b02	1066.7	884.7	17.06	510.2	52.17	530.4	50.27
b03	4942.8	3782.2	23.48	1787.2	63.84	1429.7	71.07
b05	8549.0	6634.3	22.40	3385.6	60.40	2720.6	68.18
b06	2263.7	1739.9	23.14	836.6	63.04	610.2	73.04
b08	4154.9	3217.8	22.55	1710.4	58.83	1195.4	71.23
average	-	-	**20.81%**	-	**59.09%**	-	**64.86%**

scripts are used to make sure that appropriate timing constraints are applied. Special attention is given to the synthesis of *submodule1* of the control logic as its timing is essential for proper operation.

4.3.2 Experimental Results

4.3.2.1 Area and Power Estimation

The area figures for the hardened circuits are estimated from the synthesized designs obtained using method discussed in Subsection 4.3.1.5. Each synthesized design contains a version of the six benchmark circuits, hardened by each of the HyFT design variant discussed in Section 4.2. The area estimates are presented in Table 4.3. The first column lists the considered benchmark circuits. The second, third, fifth and seventh columns give the absolute values of area occupied by each of the four HyFT design variants in μm^2. Where as the fourth, sixth and eighth columns give the percentage area savings in case of HyFT-1b, HyFT-2a and HyFT-2b respectively with reference to HyFT-1a architecture. The last row of the Table 4.3 gives the average area savings.

It can be seen in Table 4.3 that HyFT-1b saves on average 20.8% area. By further moving the *comparison-window* towards the *CLK* capture edge, the gain increases drastically as in the case of HyFT-2a and HyFT-2b architectures the percentage of average area saving reach to about 60%.

The power estimates are obtained by exercising the synthesized circuits with the workload generated as discussed in Subsection 4.3.1.4. Note that same workload is used with all the hardened version of each benchmark to have a fair comparison. The switching activity information from the back-annotated gate-level simulation is used in power estimation tool to obtain the power estimates. These estimates are presented in Table 4.4, which uses the same structure as of Table 4.3. Power savings show a similar trend as area savings. The average power savings are 6.11%,

Table 4.4: Power Savings

	HyFT-1a	**HyFT-1b**		**HyFT-2a**		**HyFT-2b**	
	Area (μm^2)	Power (mW)	Saving (%)	Power (mW)	Saving (%)	Power (mW)	Saving (%)
b01	318.7	300.0	5.84	178.5	43.98	164.0	48.55
b02	205.4	203.8	0.75	146.7	28.54	136.6	33.46
b03	954.4	873.2	8.51	432.0	54.74	346.6	63.68
b05	1856.6	1706.2	8.10	799.0	56.93	742.2	60.02
b06	441.9	407.3	7.85	206.8	53.21	174.5	60.50
b08	877.3	828.2	5.60	433.4	50.60	356.6	59.35
average	**-**	**-**	**6.11%**	**-**	**48.00%**	**-**	**54.26%**

48.0% and 54.3% for HyFT-1b, HyFT-2a and HyFT-2b respectively with HyFT-1a architecture as reference.

The area and power results show that easing the contamination delay constraints on the CL synthesis not only saves significant amount of area but also power. We can also deduce from the results that in terms of area and power the best candidate is HyFT-2b as it is the one with least lenient or even no contamination delay constraints at all. However the impact of HyFT-2a architecture on fault tolerance capability and performance is discussed in next subsection before the all rounder candidate can be selected.

4.3.2.2 Fault Tolerance Capability Assessment

The fault tolerance capability assessment framework detailed in Section 2.3.3 is employed to assess and compare the robustness improvements associated with each of the HyFT design variant. Synthesized circuits are subjected to fault injection campaign with fault injection parameters given in Table 4.5. Table 4.5a lists the parameters common to fault injection campaigns run on all the benchmark circuits, while Table 4.5b gives the parameters associated with fault injection campaigns involving to each individual benchmark circuit. An important parameter mentioned in Table 4.5a is the *fault location* and as specified, the locations for fault injection as chosen randomly all over the circuits but not only in the CL blocks. This means that the faults are also injected in the reconfiguration multiplexer and demultiplexer, pipeline registers, HyFT control logic and in the pseudo-dynamic comparator.

We recall from Section 2.3.3 that the fault injection reports are used to classify the injected faults on the basis of their impact on the circuit output. Table 4.6 gives the distribution of injected faults among three categories namely *Silent*, *Corrected* and *Fail-silent* (as discussed in Section 3.3.6.3) and their percentages in the total injected faults for each of the HyFT design variant hardened version of each benchmark circuit. The third column of Table 4.6 gives the total number

Table 4.5: Fault injection parameters

(a) Common parameters

Parameter	Value
Fault model	Temporary Stuck-at
Injection duration range	0.25 ns - 1.25 ns (Constrained random)
Workload	Stuck-at fault detection patterns
Fault location constraint	Allover the architecture (Random)
Injection type	Single fault

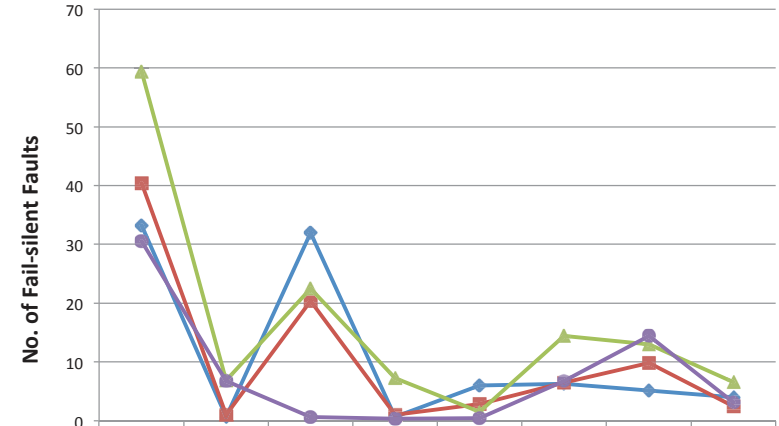
(b) Individual parameters

Benchmark	No. of injections	Simulation duration (per injection)	Injection time range (Constrained random)
b01	6088	240ns	45ns - 200ns
b02	5245	140ns	45ns - 100ns
b03	25630	340ns	45ns - 300ns
b05	25248	1020ns	45ns - 980ns
b06	9601	220ns	45ns - 180ns
b08	20771	550ns	45ns - 510ns

of injected faults in each benchmark circuit implementation. All the subsequent columns are grouped according to the associated HyFT variant, for example, forth, fifth and sixth columns give the number and percentages of faults with *Silent*, *Corrected* and *Fail-silent* outcomes respectively for HyFT-1a implementations of each benchmark circuit. Similarly the remaining three groups of columns give the same information for HyFT-1b, HyFT-2a and HyFT-2b architectures. Moreover, the last row gives the average percentages of faults that fall into the corresponding category.

Fault injection results in Table 4.6 show that on average 0.62% of total injected faults in HyFT-1a resulted in a fail-silent operation. Whereas, these percentages for HyFT-1b, HyFT-2a and HyFT-2b are 0.6%, 0.83% and 0.5%, respectively. This shows that HyFT-2a is the least effective against SETs. The reason for this robustness degradation in HyFT-2a can be explained with the help of the graph shown in Figure 4.10 that shows the average distribution of *Fail-silent* faults in different parts of the circuits protected by each HyFT design variant. The distribution associated with HyFT-2a architecture shows that four out of eight circuit regions remain relatively vulnerable. The input register with 59 *Fail-silent* faults, the CL copies with 22 *Fail-silent* faults, the HyFT-2a control logic with 14 *Fail-silent* faults and output register with 13 *Fail-silent* faults. This shows that HyFT-2a not only loses the marginal protection against SETs in the output register by having its comparator moved before the output register, but also misses detection of a significant number of faults injected in the CL blocks because of non-overlapping *comparison-window* and setup-hold window as shown in Figure 4.1f. Although, the HyFT-2b architecture also has no protection against SEUs due to direct strikes in the output registers, its ability to detect transients

which occur close to the *CLK* capture edge gives it an advantage that makes it the most effective technique in terms of fault tolerance capability.



	Input Register	Demux	CL	Mux	Comparator	Ctrl Logic	Output Register	Rest of the circuit
HyFT-1a	33	1	32	1	6	6	5	4
HyFT-1b	40	1	20	1	3	7	10	2
HyFT-2a	59	7	22	7	2	14	13	7
HyFT-2b	30	7	1	0	0	7	14	3

Fig. 4.10: Spatial distribution of *Fail-silent* faults

Table 4.6: Fault Injection Results

			HyFT-1a			HyFT-1b			HyFT-2a			HyFT-2b		
		Total	Silent	Corrected	Fail-silent	Silent	Corrected	Fail-silent	Silent	Corrected	Fail-silent	Silent	Corrected	Fail-silent
b01	#. of Faults	6088	5605	433	50	4713	338	50	2527	78	57	2589	80	49
	%. of Faults	-	92.07%	7.11%	0.82%	93.64%	5.54%	0.81%	97.78%	1.29%	0.93%	97.88%	1.31%	0.81%
b02	#. of Faults	5245	4784	427	34	3987	330	34	2352	96	62	2532	57	19
	%. of Faults	-	91.21%	8.14%	0.65%	93.07%	6.29%	0.65%	97.00%	1.82%	1.18%	98.56%	1.08%	0.36%
b03	#. of Faults	25630	23581	1918	131	18243	1232	137	8690	347	230	7058	246	110
	%. of Faults	-	92.01%	7.48%	0.51%	94.66%	4.81%	0.54%	97.75%	1.35%	0.90%	98.61%	0.96%	0.43%
b05	#. of Faults	25248	23259	1876	113	18236	1255	102	9550	297	151	7683	257	95
	%. of Faults	-	92.12%	7.43%	0.45%	94.63%	4.97%	0.40%	98.21%	1.18%	0.60%	98.61%	1.02%	0.38%
b06	#. of Faults	9601	8857	683	61	6871	449	59	3376	110	63	2456	74	58
	%. of Faults	-	92.25%	7.11%	0.64%	94.70%	4.68%	0.62%	98.21%	1.14%	0.65%	98.63%	0.77%	0.60%
b08	#. of Faults	20771	19163	1469	139	14892	1068	125	8161	241	149	5724	167	85
	%. of Faults	-	92.26%	7.07%	0.67%	94.25%	5.15%	0.60%	98.13%	1.16%	0.72%	98.79%	0.80%	0.41%
Avg.	**#. of Faults**	**15431**	**14208**	**1134**	**88**	**14567**	**779**	**85**	**15117**	**195**	**119**	**15215**	**147**	**69**
	%. of Faults	**-**	**91.99%**	**7.39%**	**0.62%**	**94.16%**	**5.24%**	**0.60%**	**97.85%**	**1.32%**	**0.83%**	**98.51%**	**0.99%**	**0.50%**

4.3.2.3 Performance Evaluation

The performance of the HyFT architecture can be interpreted in three aspects. First, the temporal overhead due to the delay added in the data path by the fault-tolerant architecture components. This aspect of performance remains the same for all the four HyFT variants because they all use the same components, which add a similar amount of delay to the data path. The second merit of performance is the error recovery penalty, i.e. the time needed to recover from a single error. Since all four variants take either 2 or 4 cycles to recover depending on the type and location of fault (as discussed in Section 2.2.7), even this aspect is irrelevant while comparing them. The third and the most important aspect of performance that differs among all the four variants is due to the effectiveness to target the transients that are most likely to result in an error. With this effectiveness, the performance can be improved by avoiding spending time to recover from conditions that are not even erroneous.

The figures that can give us a measure of this aspect of the performance overhead can be interpreted from the fault injection results presented in Table 4.6. It can be observed in Table 4.6 that HyFT-1a corrected on average 7.39% of injected faults. For each detected and corrected SET the HyFT undergoes a recovery phase that takes 2 additional cycles. According to these figures, HyFT-1a spends around 14.78% of the total computation time on recovering from potentially erroneous states. On the other side, HyFT-1b spends only 1.98% of the time in the recovery phase. Besides this, HyFT-2b also resulted in less fail-silent conditions. This clearly shows that the effectiveness to target only the fatal transient faults reduces the computation time and thus saves energy.

4.3.3 Discussion

Figure 4.11 graphically shows the way the four design metrics improve or degrade with each design modification. Each line graph corresponds to one of the four considered design metrics represented as a percentage in comparison with the corresponding metric of the initial design (HyFT-1a). Area, power and performance show very clear trends of improvement with each optimized design. Since the design evolution in moving from HyFT-1a to HyFT-2b causes the timing constraints to be relaxed, the improvements in terms of area and power are quite evident. Similarly, the performance improvement achieved with HyFT-2b signifies the advantage of having the comparison-window placed across the capture edge. The fault tolerant capability measured in terms of average number of fail-silent faults also shows an overall improvement with an exception in case of HyFT-2a. As discussed in subsection 4.3.2.2, the drop in robustness measure in the HyFT-2a variant is due to the lack of protection against SEUs caused by direct particle strikes in

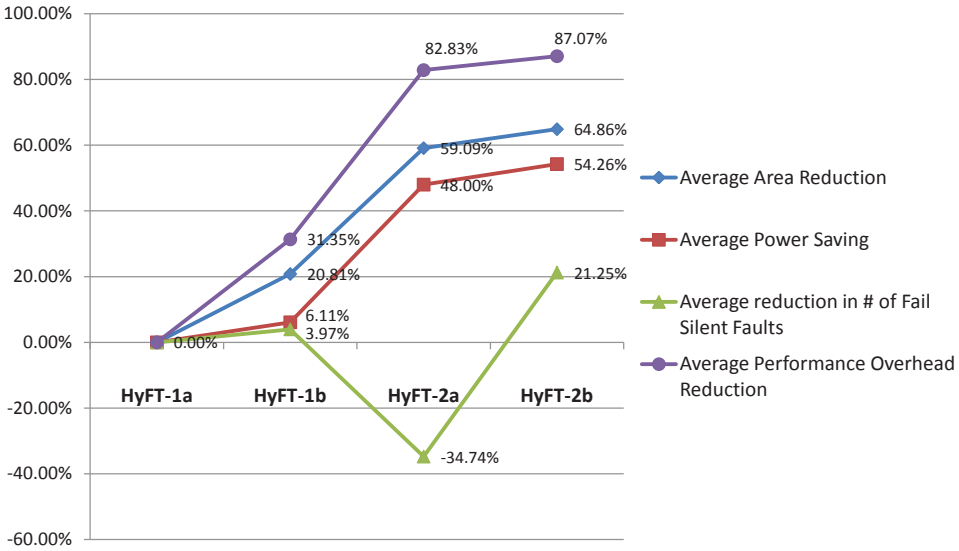


Fig. 4.11: Design metrics improvement summary

the output register but more pronounced because of the lack of ability to compare the outputs for inequalities at the most crucial time, i.e. during the *CLK* capture edge.

4.4 Impact of circuit size on the evaluated metrics

In this section we discuss how the evaluated metrics change with the size of the circuit to which the optimized HyFT architectures are applied. To illustrate this relationship we show in Figure 4.12 the percentage improvement of the evaluated metrics of the HyFT-2b design for each of the considered benchmarks arranged in the ascending order of their size from left to right. The graph shows an overall increasing trend for average area and power reduction. The proposed optimizations seem more effective for larger circuits in terms of area and power improvements because the area that is saved is due to the application of relaxed constraints on the CL blocks. Therefore, the larger the CL block, the larger the improvement in terms of area and power when compared with the initial HyFT architecture. Performance on the other hand shows a relatively regular trend for different sizes of circuits. We can expect that this design merit will remain constant when assessing the improvements for circuits of different sizes. On the other side, the fault tolerance capability improvement shows an irregular trend and seems not to depend on the circuit size.

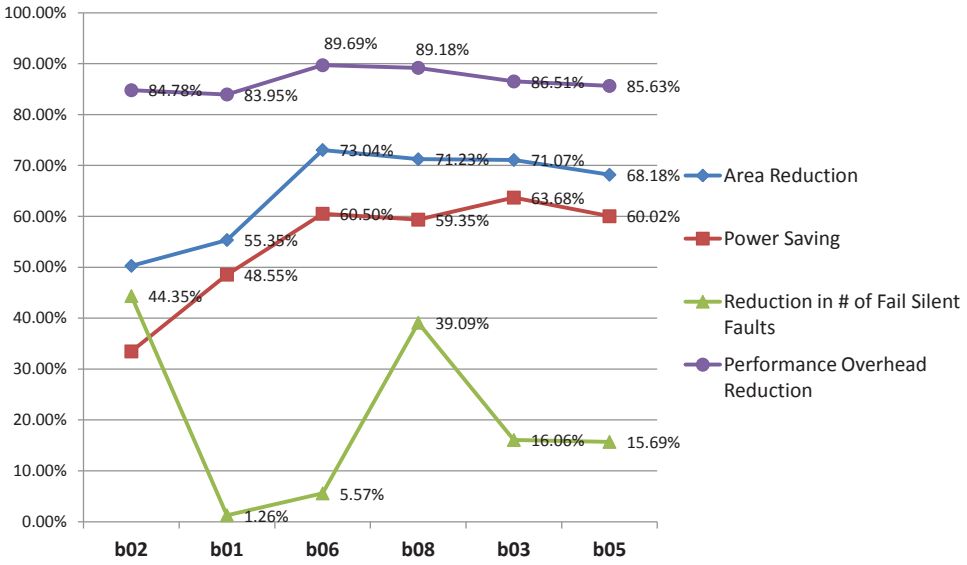


Fig. 4.12: Impact of circuit size on evaluated metrics

4.5 Summary

In this chapter we presented a design space exploration study aimed to optimize the HyFT architecture. A few limitations of HyFT architecture were also discussed. The prime design parameters were identified and controlled to limit these drawbacks. Three new design variants were obtained and their effect on the four important design merits were analyzed. The experimental results showed that the proposed optimizations save significant amounts of area and power by easing the synthesis constraints that were imposed by the initial design. Reducing surface intrinsically reduces the possibility of fault occurrence and the ability to act against only the potentially fatal SETs reduces the number of *Fail-silent* faults and improves the performance.

Chapter 5

Selective Hybrid Fault Tolerance

From the safety point of view, selecting the ideal trade-off between reliability improvement and cost associated with a fault tolerant architecture employed for hardening, mainly depends on the level of criticality of the application, the environment radiation levels and the technology used. It is important to create a balance that best suits the design cost-budget and the acceptable error rate constraint [72]. Selective hardening is a technique that creates this balance by allowing the design to move between the two edge-cases of no-hardening and fully hardened, and optimize it by selecting the most vulnerable circuit parts to be hardened. Selecting only the most vulnerable circuit parts for hardening and leaving others unprotected can provide improvement in error rate at an acceptable area and power overheads [75].

In this chapter we introduce an efficient cost (in terms of area and power) versus reliability improvement trade-off analysis methodology for CL circuits, which helps taking this key design decision. Based on this analysis we also propose a selective hardening technique using the Hybrid Transient Fault Tolerant (HyTFT) architecture. The selective hardening involves reducing the number of outputs to be compared for error detection. Reducing the comparison point not only reduces the size of the comparator but also significantly reduces the size of the duplicated CL copy since only the logic cones responsible for generating the selected outputs need to be synthesized to create the duplicated copy.

This chapter is organized such that, we first briefly review some selective hardening techniques from the literature in Section 5.1. In the subsequent section we discuss a low-cost method for identifying the most venerable outputs nodes of a logic circuit and validate the results by comparing them with those obtained from an accurate fault injection based method. In Section 5.4 we present the selective HyTFT architecture and a low-cost method to estimate its fault tolerance capability. In Section 5.5 we present the experimental results obtained by using the low-cost reliability estimation method to study the cost-reliability trade-off of a Selective HyTFT

implementation and we also validate them. Finally, we conclude the discussion with some perspectives in Section 5.6.

5.1 Previous work

Selective hardening is done in two steps. First the most vulnerable circuit parts are identified on the basis of their contribution to the overall circuit error rate. Second, a fault tolerance architecture is applied on the selected circuit parts [112]. Most of the approaches of selective hardening in the literature focus in improving the vulnerability analysis methodology and use existing fault tolerant architectures for hardening. Thus, are better classified on the basis of the former criteria. Table 5.1 give a summary of a few techniques used for selective hardening.

The three masking effects discussed in Section 1.3.1.2 that inherently prevent transient pulses from getting latched in FFs are generally employed in the circuit element vulnerability estimations. The use of accurate models that consider all three masking effects is impractical because of the immense amount of computational effort needed to simulate or to solve these models. Therefore, some techniques like [30], [65] and [14] rely on approximate abstract models while considering all three masking effects, whereas others like [72], [76], [56] and [112] resort to only one or two of them to identify circuit elements with highest impact on soft error rate.

One of the techniques that only take into account the effects of logical masking is proposed by Pagliarini et al. in [72]. It determines the a subset of standard cells when hardened, the circuit can meet the user defined reliability target in a power-effective manner. The vulnerability analysis is based on an algorithm in which reliability probability is attributed to each signal, then by propagating these signals probabilities through the logic block till the CL outputs, it reaches to an expression of the overall circuit reliability. Based on the extent of reliability contribution of each cell to theis circuit's reliability expression it ranks the cells. The work by Polian et al. in [76] also uses error probabilities of gates to estimate its contribution to the circuit soft error rate. But unlike [72], instead of propagating gate error probabilities, it expresses the circuit error rate as a conditional probability that an error at the gate output also have an observable effect on the circuit output. The gate observability metric is expressed as the detection probability of stuck-at fault on the output of that gate. This technique offer a scalable alternative to Binary Decision Diagram (BDD) based methods for soft error estimation. They use probabilistic models to account for possible unbalance in input values.

Table 5.1: Summary of selected previous work in the area of selective hardening

	Masking effects considered			Circuit element of interest	Targeted faults	Fault tolerance technique Employed	Vulnerability Analysis & Ranking		
	Logical	Latching-window	Electrical				Method	Error	Computational effort
Pagliarini et al. [72]	✓			standard cells	MTFs	Cell TMR	SPRA[34]	5 - 30% [45]	low
Polian et al. [76]	✓			gates	SETs	-	Probabilistic	-	low
Maniatakos et al. [56]	✓[1]			FFs	SEUs	-	Partial simulation based fault injection	1.4%	high
Bottoni et al. [14]	✓[1]	✓		FFs	SEUs & permanent	FF TMR	Simulation based fault injection	4.4%	high
Fazeli et al. [30]	✓	✓	✓	FFs	METs	-	MEPP[4]	2.5%	low
Mohanram et al. [65]	✓	✓	✓	gates	SEUs	CL partial duplication with comparison	Probabilistic & heuristic	-	-

[1] Workload-specific logical masking

The work by Maniatakos et al. in [56] focuses on identifying the most vulnerable parts of microprocessor for harden. The work takes into account the logical masking effects impact particular to the workload used. Bottoni et al. in [14] propose a fault injection vulnerability analysis method which inherently takes into account the workload specific logical masking and latching window masking. The computation effort for the method proposed is high because involves simulation based fault injection. Among the techniques which consider the impact of all three masking effects for identifying the soft spots in circuits some prominent pieces of work include by Fazeli et al. in [30] in which they use probabilistic models the three masking effects. Where as the work by Mohanram et al. in [65] besides using probabilistic models, uses a novel heuristic model to meet the reliability and cost trade-off for the proposed partial duplication architecture.

5.2 A low-cost output susceptibility analysis

The output susceptibility analysis methodology is based on the fact that not all the outputs of a CL block have the same susceptibility to SET effects and hypothesizes that their susceptibility is a function of the number of nodes in its fan-in logic cone. It exploits the structural properties of the output fan-in cone to get their relative susceptibility estimates. The outputs are ranked on the basis of their relative susceptibility and the best candidates are selected for error detection. The number of output candidates selected, defines the reliability improvement and cost trade-off and the vulnerability-aware selection allows us to optimize it.

As seen in Section 5.1, techniques in previous work mostly consider logical masking to select the most vulnerable circuit nodes for hardening. A few approaches also consider the impact of electrical and latching-window masking effects in the susceptibility analysis. In our approach, the impact of these factors is not taken into account mainly to reduce the computational costs associated with these methods and secondly to investigate the possibilities of improving the efficacy of our approximate method to be used with large complex designs.

Algorithm 1 shows the pseudo-code of the proposed method. The algorithm starts by reading the pre-place-and-route netlist of the design. Then it forms groups F_j of all fan-in cells for each CL output O_j. Once the groups are formed the weight W_j of each fan-in cone is calculated by adding together the weights of all the cells in the corresponding fan-in cone group. According to the hypothesis that forms the basis of this method, cell weight is the number of inputs and outputs of that cell. Ranks are assigned to each output on the basis of their fan-in cone weight using a sort function shown in line 15 of Algorithm 1.

```
1  read(netlist);
2  // Group all fan-in cone cells together for each output node
3  foreach O_j do
4  |  F_j ← O_j.get_fanin();
5  end
6  // Get weight of fanin cone of each output
7  foreach O_j do
8  |  foreach C_i do
9  |  |  if C_i ∈ F_j then
10 |  |  |  W_j ← W_j + C_i.get_pins();
11 |  |  end
12 |  end
13 end
14 // Sort outputs on the basis of their fanin cone weight
15 sort(O_j,F_j,W_j);
```

Algorithm 1: Output susceptibility analysis

The algorithm is further explained by its application to a simple example circuit shown in Figure 5.1. The shaded regions mark the boundaries of the two output fan-in cones. The weight parameter (W_i) is given on the top of each gate. The fan-in cones weight (S_j) given on the right of corresponding output is found to be 14 and 12 for O_1 and O_2 respectively. According to these figures we can infer that the output O_1 is more susceptible to SETs than output O_2 . In other words, having a SET detection mechanism placed on O_1 can improve the reliability of the circuit more than having it placed on O_2.

A Tcl script implements the susceptibility analysis algorithm, which is used with Synopsys® Design Compiler to perform the necessary operations. An experiment is performed by applying the proposed low-cost output susceptibility analysis method to a circuit obtained by extracting CL from an ITC'99 benchmark circuit (i.e. b05). We obtain the output susceptibility figures illustrated graphically by the red colored plot of Figure 5.2. It represents the normalized distribution of the output fan-in cone weight (S_j) with the output arranged on the horizontal axis. The trend of the graph suggests that there is a certain number of outputs driven by a notably large fan-in cones than the rest of the outputs. Presumably, these outputs should have a higher contribution to the circuit soft error rate.

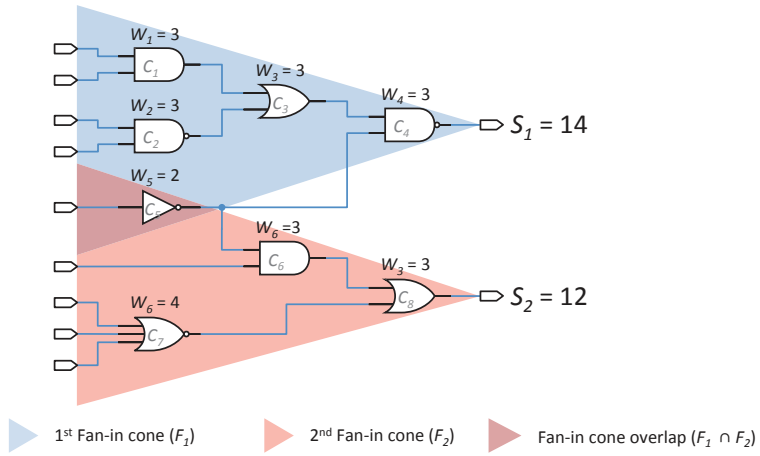


Fig. 5.1: Application of the output susceptibility analysis on an example circuit

5.2.1 Results

An experiment is performed by applying the proposed low-cost output susceptibility analysis method to a circuit obtained by extracting CL from an ITC'99 benchmark circuit (using the method explained in Appendix C) i.e. b05. We obtain the output susceptibility figures illustrated graphically in Figure 5.2. The graph represents the weight (S_j) distribution of the output fan-in cones. The trend of the graph suggests that there is a certain number of outputs driven by a notably large fan-in cones than the rest of the outputs. Presumably, these outputs should have a higher contribution to the circuit soft error rate.

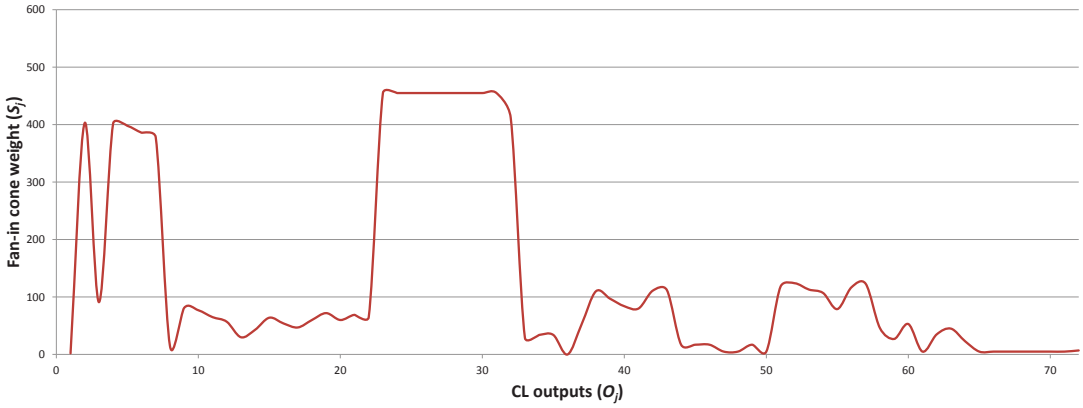


Fig. 5.2: Output susceptibility analysis results (of CL block extracted form b05)

5.3 Validation of the approach

The susceptibility analysis discussed in the previous section is based on the hypothesis that the susceptibility of CL outputs to SETs is a function of the number of signal nodes in the fan-in cone of that output. In order to test this hypothesis we compare the results of the low-cost susceptibility analysis method with fault injection based output susceptibility results.

5.3.1 Fault injection based output susceptibility analysis

The idea behind fault injection based output susceptibility analysis is to inject faults on randomly selected locations in an unhardened version of a CL circuit for which the output susceptibility is desired and observe which outputs fail more often. It involves running a number of fault injection campaigns using the fault injection framework detailed in Section 2.3.3. Running multiple fault injection campaigns allow us to obtain multiple distribution samples of the number of failures observed on CL outputs. The similitude among different distribution samples ensures the efficacy of this method.

Five fault injection campaigns each comprising of 2012 simulations with single SET injection are performed on same circuit used for the low-cost output susceptibility analysis experiment discussed in Section 5.2.1. The fault injection parameters for these experiments are given in Table 5.2, which are common for all the campaigns. It can be seen that, SETs with random duration within 0.25 ns and 1.25 ns are injected on locations distributed randomly within the CL block at random times during the simulation. The workload used during fault injection consist of patterns optimized for stuck-at fault detection for obtaining activity distributed allover the circuit and to reduce the impact of workload-specific logical masking effect on the susceptibility of outputs.

Table 5.2: Fault Injection Parameters for output susceptibility analysis

Parameter	**Transient faults**
Fault model	Temporary Stuck-at
Injection duration range	0.25 ns - 1.25 ns (Constrained random)
Workload	Stuck-at fault detection patterns
Fault location constraint	Allover the CL (Random)
Injection type	Single fault
No. of injections	2012

The graph of Figure 5.3 gives the distribution of average number of SEUs observed in the output register of the circuit with the output arranged on the horizontal axis. The vertical lines represent the standard deviation of five samples obtained from their average. Since the sample size of our experiment is low, we show the standard deviation figures to give an idea of the

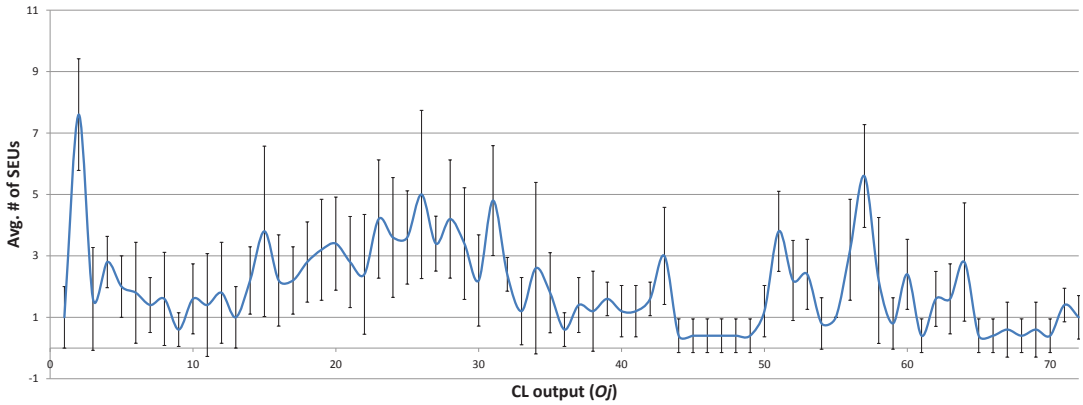


Fig. 5.3: Output failure distribution

confidence level, which is sufficient enough to infer that there exist a correlation between the number of failures observed on an output node during our random fault injection campaigns and the susceptibility of that output. The sample size can be increased by running more fault injection campaigns if a higher confidence level is desired.

5.3.2 Discussion

Figure 5.4 shows the normalized distribution of output susceptibility obtained from the low-cost susceptibility analysis method and the average normalized distribution of fault injection based susceptibility on the same graph. We can see that the distribution obtained from the low-cost approximate method bears some similarities with the accurate distribution.

The CPU runtime for the accurate fault injection based method is around 45.5 hours, which involves five fault injection campaigns each with 2012 simulation. Whereas, the low-cost method took only 2.7 seconds to generate the susceptibility estimates. This runtime also include the overhead associated with the use of Synopsys® Design Compiler. A standalone implementation of the algorithm can further reduce the runtime to make it feasible to be used with large industrial benchmarks. Low computational effort and the degree of accuracy of the low-cost output susceptibility analysis method gives good grounds to further study the short comings and improve its accuracy.

5.4 Selective HyTFT Architecture

We recall from Chapter 2 that the Hybrid Transient Fault Tolerant (HyTFT) architecture is capable of detecting transient, permanent and delay faults and correcting transient faults. It employs

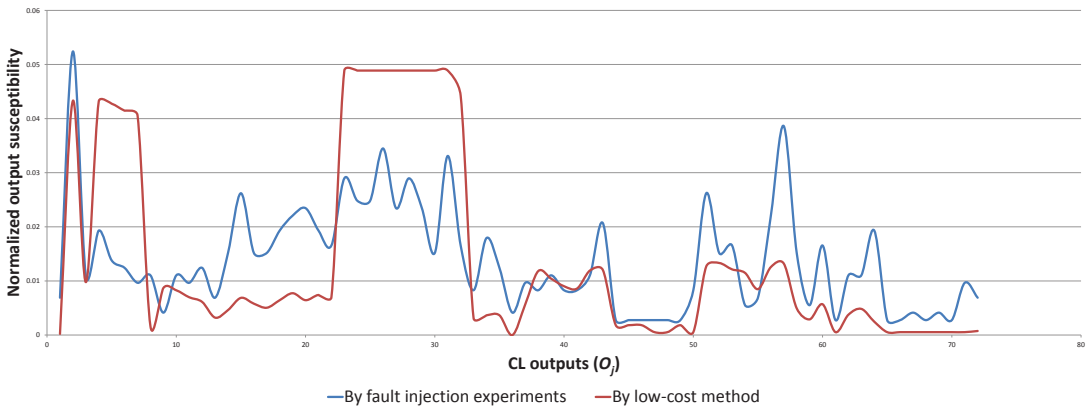


Fig. 5.4: Comparison of output susceptibility results (of CL block extracted form b05)

duplication and pseudo-dynamic comparison to detect faults and a single-cycle deep rollback scheme for correction. Since the architecture relies on duplication of CL block and a pseudo-dynamic comparator its implementation incurs an overhead of more than 100% in terms of area and power.

A practicable way of providing the designer the freedom to control the cost and reliability improvement of HyTFT implementation is to intelligently select a set of outputs for comparison. It allows reducing the overhead with duplication and comparison at a cost of the fault tolerance capability. Figure 5.5 shows a simplified scheme of selective HyTFT architecture. It can be seen that CL2 logic block only implements a part of the logic in CL1. CL2 only contains logic cones necessary to implement the logic function on the output pins selected of comparison. The method discussed in Section 5.2 provides us with a low-cost means of doing this selection.

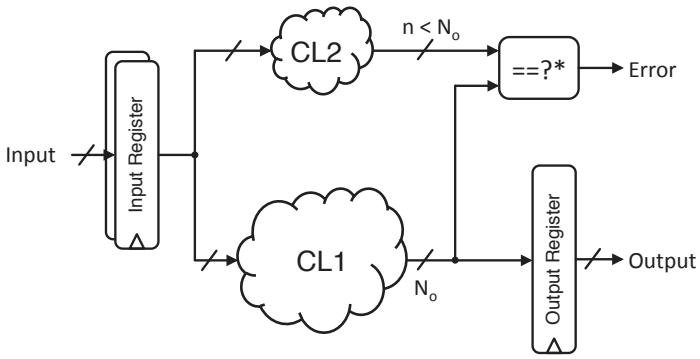


Fig. 5.5: Selective HyTFT Architecture

5.4.1 A low-cost HyTFT architectural reliability estimation

One method of estimating the reliability of a fault tolerant architecture is to perform fault injection experiments and analyze systems outcome, as we have seen in different experimental studies in this manuscript. To reach a statistical confidence level sufficient to infer the reliability improvement of a fault tolerant architecture, it requires a rigorous fault-injection campaign [56]. Furthermore, if fault-injection is employed to study the trade-off between cost and reliability improvement of various selectively hardened variants it multiplies the simulation effort as many times as the number of intermediate variants between no-hardening and full-hardened being analyzed.

To provide an analysis method that gives a reasonably accurate estimation of the reliability of the selective HyTFT architecture variants with minimum computational effort, we present a simple method that employs the same hypothesis used in Section 5.2 for output susceptibility analysis. According this methodology the ratio of cumulative weight ($\sum_{j=1}^{n} S_j$) of the duplicated CL cones in CL2 to that of the CL1 gives an estimate of the reliability degradation factor from the fully hardened edge-case.

In the output susceptibility analysis method presented in Section 5.2 we do not mind about the fan-in cone overlaps because the analysis is meant to produce output ranking on the basis of individual cone weight (S_j). But in the case of selective HyTFT architecture, where one or more output cones are duplicated for comparison, the cumulative weight of the selected outputs should be calculated by resolving the fan-in cone overlaps. In Algorithm 2 with the help of a pseudo-code we explain how we resolves the fan-in cone overlaps. It can be seen in line 1 of the pseudo-code that it calls Algorithm 1 to obtain a list of outputs with fan-in cone weights already assigned. Once the individual cone weights are assigned to the outputs, Algorithm 2 begins iterating each output O_j starting from the most susceptible one (i.e. one with highest weight) and continues in descending order ot their weights. During each iteration of O_j, each cell (C_i) is checked if lies in the current output fan-in cone. If it exists in that cone and is already claimed to lie in any other output fan-in cone during previous iterations, its weight is deducted from the cumulative weight. And, if the cell exist in the current fan-in cone but is not claimed by any previously iterated output fan-in cone, it is marked as claimed. In this way Algorithm 2 continues till it has iterated each output in the design.

Figure 5.6 explains how Algorithm 2 calculates the cumulative weight of outputs of an example circuit. It can be seen that in order to calculate the cumulative weight of two outputs (S_1 and S_2) the cells common to both the output fan-in cones (C_5 in this example) are counted only once.

```
1  Algorithm 1();
2  // Resolve fan-in cone overlaps by including common cells only in
       fan-in cone with larger weight
3  foreach O_j do
4      foreach C_i do
5          if C_i ∈ F_j then
6              if C_i ∉ claimed_cells then
7                  claimed_cells ← claimed_cells ∪ C_i
8              else
9                  S_j ← S_j − C_i.get_pins();
10             end
11         end
12     end
13 end
```

Algorithm 2: Cumulative weight caluclation

5.4.1.1 Results

Table 5.3 gives the results obtained by applying the low-cost selective HyTFT reliability estimation method to a CL block extracted from four ITC'99 benchmark circuit listed in Column 1 (*ckt.*). Each row in the table represent a selective HyTFT variant starting from the no-hardening to fully-hardened edge cases grouped together for each benchmark circuit. Column 3 (*#. of output nodes selected*) gives the number outputs selected for error detection corresponding to each variant listed in Column 2 (*Selective HyTFT variant*). So, for example the first row of each group corresponds to the case in which only one output that is most susceptible is selected for error detection and the second row of each group represents the case in which the two most susceptible outputs are selected and so on. The suffix in the name of each variant represent the percentage ratio of cumulative fan-in cone weight of the duplicated CL copy given in Column 4 (*Cumulative fan-in cone weight* $\sum_{j=1}^{n} S_j$), to that of the primary copy. This percentage is also given in Column 5 (*Cumulative fan-in cone weight %*). The last column gives the Vulnerability Factor (*VF*). *VF* gives us the estimate of the unreliability of the HyTFT variant on the scale from 0 to 1. It is obtained by the expression of Equation 5.1.

$$VF = 1 - \frac{\sum_{j=1}^{n} S_j}{\sum_{j=1}^{N_o} S_j} \tag{5.1}$$

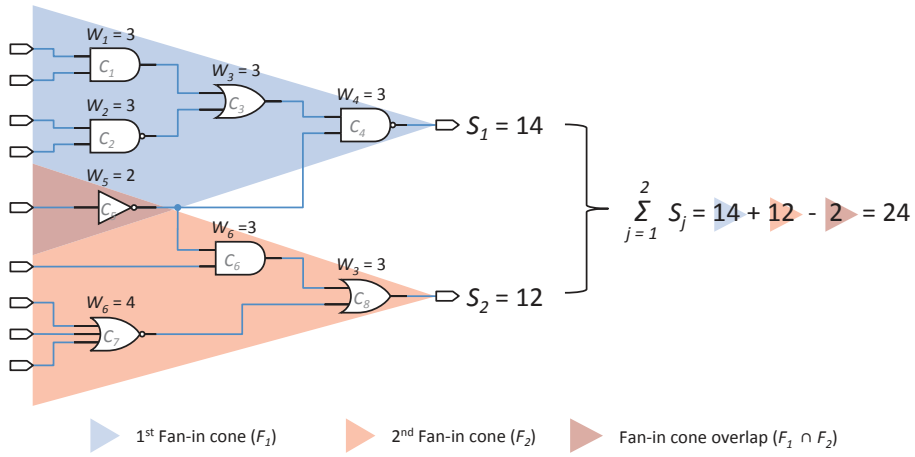


Fig. 5.6: Cumulative weight calculation of an example circuit

where:

VF = Vulnerability Factor
S_j = output node fan-in cone weight
n = number of output nodes selected for error detection
N_o = total number of primary CL block outputs

5.5 Experimental Validation and Results

The low cost vulnerability analysis method for selective HyTFT variants discussed in Section 5.4.1 provides us with fast way of analyzing the cost versus reliability improvement relationship. In order to validate the results obtained using this method we compare them with more accurate results obtained performing fault injection on the no-hardening version, the fully-hardened version and a number of intermediate selective HyTFT variants which lie between them.

5.5.1 Experimental Setup

The experimental setup for the validation of the results obtained from the low-cost selective HyTFT architecture vulnerability analysis is realized by applying the selective HyTFT architecture on the four considered benchmark circuits to obtain four sets of HyTFT variants each containing and the two edge-cases and a set of intermediate variants. The application involves steps described in following subsections.

Table 5.3: Area, power and reliability estimates of Selective HyTFT variants

ckt.	Selective HyTFT variant	#. of output nodes selected	Cumulative fan-in cone weight		CL Area		CL Power		Soft errors		Vulnerability Factor
			Absolute	Gain	Absolute	Gain	Absolute	Gain			
		[n]	$[\sum_{j=1}^{n} S_j]$	(%.)	(μm^2)	(%.)	(mW)	(%.)	(#.)	(%.)	[*VF*]
b03	b03_0%	0	0	0	75.01	50.00	14.66	58.83	15	Ref.	1
	b03_25%	1	54	25.35	93.10	37.94	17.13	51.91	12	80.00	0.75
	b03_30%	2	64	30.05	97.62	34.93	17.71	50.27	11	73.33	0.70
	b03_38%	4	82	38.50	101.88	32.09	19.32	45.76	11	73.33	0.62
	b03_46%	6	98	46.01	107.73	28.19	20.04	43.75	10	66.67	0.54
	b03_53%	8	113	53.05	111.72	25.53	21.36	40.03	10	66.67	0.47
	b03_60%	10	127	59.62	115.98	22.70	23.20	34.86	9	60.00	0.40
	b03_66%	12	141	66.20	120.23	19.86	25.05	29.68	9	60.00	0.34
	b03_72%	14	153	71.83	124.49	17.02	26.76	24.87	8	53.33	0.28
	b03_78%	17	167	78.40	128.48	14.36	28.53	19.90	6	40.00	0.22
	b03_86%	21	183	85.92	135.93	9.40	30.50	14.36	6	40.00	0.14
	b03_92%	24	195	91.55	142.04	5.32	32.13	9.78	5	33.33	0.08
	b03_100%	35	213	100	150.02	1	35.62	1	4	26.67	0
b05	b05_0%	0	0	0	392.62	50.00	119.75	52.22	134	Ref.	1
	b05_40%	1	455	40.37	530.40	32.45	159.40	36.74	105	78.36	0.60
	b05_51%	2	579	51.38	555.94	29.20	166.88	33.77	102	76.12	0.49
	b05_62%	3	696	61.76	576.42	26.59	170.70	32.25	102	76.12	0.38
	b05_70%	4	784	69.57	587.59	25.17	171.48	32.94	97	72.39	0.30
	b05_74%	6	831	73.74	620.05	21.04	180.31	28.44	91	67.91	0.26
	b05_80%	12	906	80.39	644.78	17.89	193.40	23.24	80	59.70	0.20
	b05_84%	17	951	84.38	663.14	15.55	198.14	21.36	66	49.25	0.16
	b05_88%	23	992	88.02	676.97	13.79	202.90	19.47	62	46.27	0.12
	b05_92%	32	1035	91.84	687.61	12.43	208.29	17.33	43	32.09	0.08
	b05_96%	44	1083	96.10	711.82	9.35	220.95	12.31	31	23.13	0.04
	b05_100%	72	1127	100	785.23	1	251.96	1	4	2.99	0
b06	b06_0%	0	0	0	33.78	50.00	4.10	57.53	10	Ref.	1
	b06_41%	1	47	40.87	44.16	34.64	5.48	43.27	6	60.00	0.59
	b06_58%	2	67	58.26	52.67	22.04	6.26	35.22	5	50.00	0.42
	b06_72%	3	83	72.17	57.99	14.17	7.04	27.13	4	40.00	0.28
	b06_82%	4	94	81.74	60.38	10.62	7.22	25.24	4	40.00	0.18
	b06_88%	5	101	87.83	63.04	6.69	8.44	12.65	4	40.00	0.12
	b06_94%	6	108	93.91	67.03	0.78	8.65	10.46	1	10.00	0.06
	b06_100%	16	115	100	67.56	1	9.66	1	0	0	0
b08	b08_0%	0	0	0	89.64	50.00	17.44	55.56	26	Ref.	1
	b08_58%	1	168	57.73	132.47	26.11	27.16	30.75	24	92.31	0.42
	b08_63%	2	187	64.26	137.79	23.15	27.11	30.88	23	88.46	0.36
	b08_68%	3	198	68.04	142.31	20.62	28.36	27.71	19	73.08	0.32
	b08_76%	6	222	76.29	151.62	15.43	31.17	20.56	17	65.38	0.24
	b08_81%	8	236	81.10	155.08	13.50	32.43	17.34	15	57.69	0.19
	b08_87%	12	253	86.94	163.06	9.05	35.30	10.02	11	42.31	0.13
	b08_92%	16	269	92.44	172.63	3.71	37.92	3.35	7	26.92	0.08
	b08_100%	27	291	100	179.28	1	39.22	1	1	96.15	0

5.5.1.1 CL Extraction and Reduction

We reuse the CL blocks extracted from the four benchmark circuit from the experiments discussed in Section 2.4.1 and 4.3.1.1. But the same can be obtained by the method explained in Appendix C.

In all the cases of duplication that we have seen in this thesis so far, all the CL copies are similar. But, in the case of selective HyTFT architecture the secondary copy is meant to have lesser logic than the primary copy, because it only contains the logic cones necessary to generate signals at the output selected for error detection. Such logic reduction can be realized by simply removing the CL outputs, which are not selected for error detection, from the input/output list of the HDL description of the extracted CL block. A re-synthesis of this edited CL block results in a CL block containing logic necessary for generating signals on the outputs remaining in the input/output list of the HDL description.

The HDL description of the extracted CL block is manually edited to obtain ten new HDL descriptions with only the required outputs remaining, each correspond to one of the intermediate selective HyTFT variants.

5.5.1.2 CL Synthesis

The control signal timing characteristics of the selective HyTFT architecture is similar to the HyFT_2b scheme which was found to be most effective in terms of area, power and performance by the study in Chapter 4. Since HyFT_2b scheme nullifies the need of applying minimum delay constraints during CL synthesis, as discussed in Section 4.3.1.2, both the primary and the set of reduced secondary copies are synthesized without any minimum delay constraints.

5.5.1.3 Workload Generation

The workload used for fault injection and power estimation experiments performed on selective HyTFT architecture is similar to the ones used in Section 2.4.1 and 4.3.1.1. We recall here that the workload is ATPG generated patterns optimized for stuck-at fault detection in the BL version of CL extracted form the four considered benchmark circuit.

5.5.1.4 RTL Description

In order maximize reuse, the architectural components for describing the ten HyTFT variants we modified the RTL blocks used in HyFT_2b scheme. The modifications we did to reuse the RTL models include; the size of the comparator, the control logic to exclude the reconfiguration

state machine and associated control signals and the wrapper to reduce the nets to account for the reduction of outputs of secondary CL copy.

5.5.1.5 Constrained Hierarchical Synthesis

We employed the same synthesis method for obtaining the selective HyTFT designs that is discussed in Section 4.3.1.5.

5.5.2 Experimental Results

The cost results of selective HyTFT variants are presented in Column 6 (*CL Area Absolute mW*) to Column 9 (*CL Power Gain %.*) of Table 5.3, in terms of area and power. These results are generated by analyzing the synthesized selective HyTFT variants with Synopsys® Design Compiler. For power estimation we used switching activity obtained by running the workload explained in previous section. We only focus on the CL parts of selective HyTFT variants instead of presenting the cost of entire implementation mainly because the non-CL part is relatively large in comparison with the CL parts and does not show significant change with the varying degree of duplication. Thus incorporating the relatively larger and fixed amounts in the presented data would make them obscure. In order to obtain comprehensive results, experimentation on larger circuits must be performed, but in this work with small benchmark circuits we intend to show the efficacy and cost-efficiency of the proposed method.

It is evident from Table 5.3 that reducing the number of outputs selected for error detection helps reducing the area and power of HyTFT variants. For instance, the selective HyTFT implementation of b05 benchmark circuit that uses only one output node for error detection (i.e. b05_40%) gains around 32.4% and 36.7% in terms of area and power, respectively.

To assess the fault tolerance capability of the selective HyTFT variants and to see how well our low-cost vulnerability analysis results coincide with the accurate results, we subjected the HyTFT variants to a fault injection campaigns with parameters listed in Table 5.2. A total of 442, 2333, 217 and 561 SETs are injected in each of the HyTFT variant of b03, b05, b06 and b08 benchmark circuits, respectively. The fault injection framework of [101] is used to perform the experiments.

Column 10 (*Soft error failure #.*) of Table 5.3 show the outcome of the fault injection experiments. It can be seen that with the reduction of error detection points, the number of soft error failures shows an increase. This increase is due to the fact that the variants with lesser logic in their secondary CL copy become more vulnerable to faults because of reduced coverage of the

error detection mechanism. Column 11 (*Soft error failure %.*) of the table gives the percentage of fail-silent faults with respect to the total number of faults injected.

5.5.3 Discussion

In order to compare the vulnerability factor obtained from the low-cost vulnerability analysis method with the fault tolerance capability results in terms of number of fail-silent faults obtained using fault injection experiments for the selective HyTFT variants we polt the two measures on the same graph on with a normalized vertical axis shown in Figure 5.7. The horizontal axis represents the percentage of cumulative weight of secondary CL copy to that of the primary CL copy such that the data points in the graph represent b05_0%, b05_40%, b05_51% till b05_100% from left to right. The straight line curve in purple represent the normalized venerability factor, where as the green curve represent the normalized number of fail-silent fault observed with the number marked on each data point on the graph. It can be seen that the vulnerability factor follows more or less a similar trend as the fault injection results with a positive error for the first two smallest HyTFT variants and a negative error for the rest of the variants.

The graph also shows in blue and red the trend of normalized area and power of the selective Hybrid Transient Fault Tolerant (HyTFT) variants respectively with absolute values of area and power mentioned on some data points. By analyzing the graph we can conclude that the first two selective HyTFT variants i.e. b05_40% and b05_51% seem most feasible because for these two cases the area and power overheads are minimum and the number of fail-silent faults are at reasonably low value, while the remaining cases either offer similar fault tolerant capability with increased cost (b05_62% and b05_70%) or cost too much compared to their impact on the fault tolerance capability (b05_70% till b05_100%).

5.6 Summary

The main objectives of the work presented in this chapter are, firstly, to propose a low cost method to analyze the cost versus reliability trade-off to avoid lengthy fault-injection campaigns at the early design phases. Secondly to show the feasibility of applying this method to selectively harden circuits using HyTFT architecture. We have seen in this chapter that the low-cost vulnerability analysis method is fast and the results obtained have a error within reasonable limits when compared with more accurate fault injection based methods. We have also shown that selective hardening circuits with HyTFT can improve the circuit reliability at a reasonable cost. Since the experiments are based on just one benchmark circuit and only one type of workload it is hard to make concrete conclusions. We intend to extend the work by adding more benchmarks and

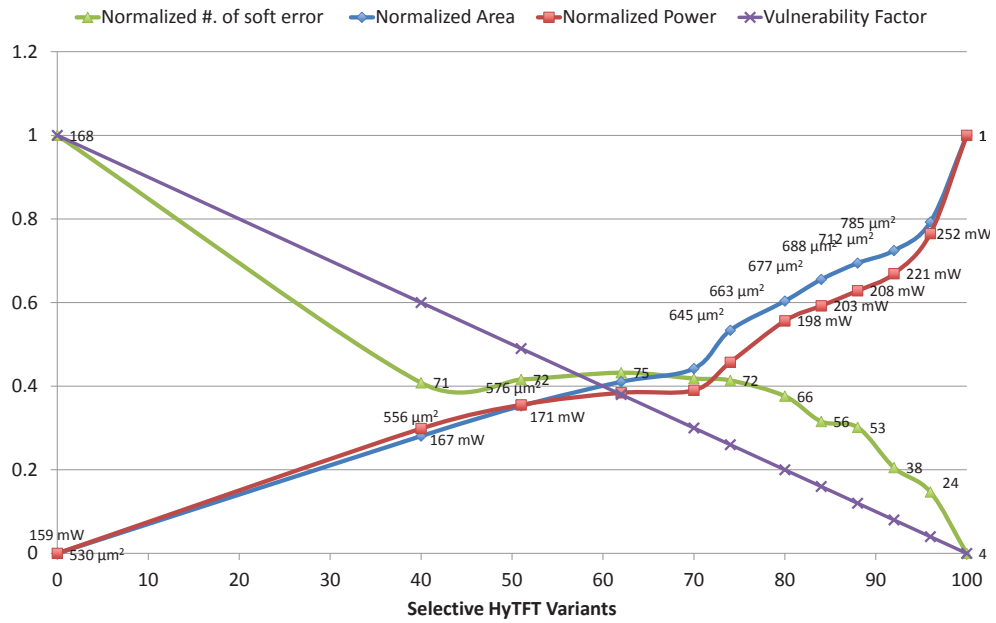


Fig. 5.7: Selective HyTFT cost versus soft-error (of CL block extracted form b05)

workloads to further explore the possibilities of using approximate methods for circuit reliability assessment method that can aid at early phases of design.

Conclusion

Whether it be, commerce, health-care, military, communication or transport, our increasing reliance on electronics to perform operations that demand high degree of reliability, availability and safety, and increasing vulnerability of transistors and interconnects are posing new challenges to the design of next generation of electronic systems. Moreover, the complexity of modern electronic systems make error detection and confinement a difficult task. On the other hand, the stringent low-power and high-performance demands has further shrinked the limits of industry's consensus about the permissible reliability enhancement overheads. In this work we present a low-power HyFT architecture for reliability improvement of modern complex pipeline circuits by protecting their combinational logic parts. The architecture can handle a broad spectrum of faults with little impact on performance. Moreover, it addresses the problem of error propagation behavior of nonlinear pipelines and error detection in pipeline stages with memory interface. We implement the proposed architecture onto a MIPS microprocessor as case study. We also present a gate-level fault-injection framework that offers high fidelity to model physical defects and transient faults. Our experimental results show the effectiveness of the proposed architecture in terms of fault-tolerant capability and power consumption.

To summarize, we state the major contributions of this work as follows:

- A comparative study based on experiments performed on four similar fault-tolerant architectures intended to reduce errors caused due to faults in combinational logic parts of microelectronic circuits and systems [100]. The experimental results show that the hybrid fault tolerant approach can handle transient faults as effectively as Partial-TMR and exhibits permanent fault tolerance capability similar to that of Full-TMR. Furthermore, it can handle the fault accumulation effect better than TMR.

- A Pipelined HyFT architecture that offers solutions to the problem of error detection and confinement in modern complex pipeline circuits [99]. A case-study of implementation of a HyFT microprocessor reveled that the implementation offers similar fault tolerance capability as TMR but with the added advantage of 11.6% power saving.

- A simulation based gate-level fault injection framework that offer high fidelity to model transient, permanent and timing faults [96]. Comprising of a comprehensive flow starting from fault site extraction till fault injection report analysis, it provides customizability to suit any application and the degree of automation allows running exhaustive fault injection campaigns without any manual intervention.

- A design space exploration study aimed at optimizing the HyFT architecture in terms of area, power and fault tolerance capability [101]. The experimental outcomes showed that by easing the synthesis constraints that were imposed by the initial design, one can save significant amounts of area and power, and the improved ability of the optimized architecture to act against only the potentially fatal SETs improves the performance.

- A selective HyTFT architecture based on a fast cost-reliability trade-off analysis methodology [98]. This fast trade-off analysis uses a low-cost reliability estimation method that is $2,500\times$ faster and produces results having error within acceptable limits when compared with a fault injection based method. The selective HyTFT architecture can improve circuit reliability at a reasonable cost.

This thesis can form a basis for various possible future work in the area of dependable circuits and systems design, including the following:

- There is a need to enhance the usability of the gate-level simulation based fault injection framework by restructuring its components to be able to control all the tasks form the common command line interface. The goal of this future task is to obtain a simulation based fault injection tool that uses the existing framework and offers a comprehensive and easy to use user-interface. As a long term perspective, we also intend to exploit an open source logic simulator like Verilator [104] or LIFTING [13] to provide a complete open source fault injection and effect analysis tool.

- Field-Programmable Gate Array (FPGA) based prototyping of HyFT architecture and fault injection to validate the fault tolerance capability results obtained from the simulation based fault injection experiments. A major anticipated obstacle during the FPGA prototyping phase is the implementation of HyFT control logic, which is the most time critical component of the architecture and the pseudo-dynamic comparator because of its non-standard logic implementation. Another major challenge that could be faced to accomplish the fault injection would be to implement the fault injection instrumentation with time resolution and accuracy sufficiently high to emulate the occurrence of SETs.

- We intend to extend the work in the area of low-cost circuit reliability assessment methods by adding more benchmarks and workloads, aiming to further explore the possibilities of using these approximate analytic methods of reliability estimation of fault tolerant architectures at their early implementation phase. Another area of future research is to use the proposed low-cost reliability estimation technique with heuristic methods of cost-reliability trade-off optimization.

Scientific Contributions

Journal

[96] I. Wali, A. Virazel, A. Bosio, P. Girard, S. Pravossoudovitch, and M. Sonza Reorda. "A Hybrid Fault-Tolerant Architecture for Highly Reliable Processing Cores". In: *Electronic Testing (JETTA), Journal of* (2015). To appear in the April 2016 issue.

International Conferences

[98] I. Wali, A. Virazel, A. Bosio, P. Girard, and M.S. Reorda. "A Low-cost Selective Hybrid Fault Tolerant Architecture". In: *Test Symposium (ETS), 2016 21st IEEE European*. To appear in the proceedings of ETS'2016.

[99] I. Wali, A. Virazel, A. Bosio, L. Dilillo, and P. Girard. "An effective hybrid fault-tolerant architecture for pipelined cores". In: *Test Symposium (ETS), 2015 20th IEEE European*. 2015, pp. 1–6. DOI: 10.1109/ETS.2015.7138733.

[100] I. Wali, A. Virazel, A. Bosio, and P. Girard. "An Experimental Comparative Study of Fault-Tolerant Architectures". In: *Advances in System Testing and Validation Lifecycle (VALID) , 2015, 7th International Conference on*. 2015, pp. 1–6. ISBN: 978-1-61208-441-1.

[101] I. Wali, A. Virazel, A. Bosio, P. Girard, and M.S. Reorda. "Design space exploration and optimization of a Hybrid Fault-Tolerant Architecture". In: *On-Line Testing Symposium (IOLTS), 2015 IEEE 21st International*. 2015, pp. 89–94. DOI: 10.1109/IOLTS.2015.7229838.

[102] I. Wali, A. Virazel, A. Bosio, L. Dilillo, P. Girard, and A. Todri. "Protecting combinational logic in pipelined microprocessor cores against transient and permanent faults". In: *Design and Diagnostics of Electronic Circuits Systems (DDECS), 17th International Symposium on*. Poster. 2014, pp. 223–225. DOI: 10.1109/DDECS.2014.6868794.

Seminars and Workshops

[95] I. Wali, A. Virazel, A. Bosio, L. Dilillo, and P. Girard. "A Fault-tolerant Architecture for Pipelined Microprocessor Cores". In: *Groupement de Recherche (GdR) SoC-SiP*. 2014.

[97] I. Wali, A. Virazel, A. Bosio, L. Dilillo, and P. Girard. "A Hybrid Fault-Tolerant Architecture for Non-Linear Pipelines". In: *Journées Nationales du Réseau Doctoral en Micro-nanoélectronique (JNRDM), 18ème*. 2015.

[103] I. Wali, A. Virazel, A. Bosio, L. Dilillo, and P. Girard. “Protecting combinational logic in pipelined microprocessor cores against transient and permanent faults”. In: *South European Test Seminar (SETS)*. 2014.

References

[1] United States. National Aeronautics, Space Administration. Scientific, and Technical Information Division. *NASA Thesaurus Aeronautics Vocabulary*. NASA technical memorandum. National Aeronautics, Space Administration, Office of Management, Scientific, and Technical Information Division, 2012. URL: http://www.sti.nasa.gov/thesvol1.pdf.

[2] M. Alam, B. Weir, and A. Silverman. "A future of function or failure? [CMOS gate oxide scaling]". In: *Circuits and Devices Magazine, IEEE* 18.2 (2002), pp. 42–48. ISSN: 8755-3996. DOI: 10.1109/101.994857.

[3] J Arlat. "Dependable Computing and Assessment of Dependability". In: *Zuverlässigkeit und Entwurf (ZuE 2011), Reliability and Design*. 2011.

[4] H. Asadi and M.B. Tahoori. "Soft Error Derating Computation in Sequential Circuits". In: *Computer-Aided Design, 2006. ICCAD '06. IEEE/ACM International Conference on*. 2006, pp. 497–501. DOI: 10.1109/ICCAD.2006.320164.

[5] N.D.P. Avirneni and A.K. Somani. "Low Overhead Soft Error Mitigation Techniques for High-Performance and Aggressive Designs". In: *Computers, IEEE Transactions on* 61.4 (2012), pp. 488–501. ISSN: 0018-9340. DOI: 10.1109/TC.2011.31.

[6] A. Avižiens. "Fault-Tolerant Systems". In: *Computers, IEEE Transactions on* C-25.12 (1976), pp. 1304–1312. ISSN: 0018-9340. DOI: 10.1109/TC.1976.1674598.

[7] A. Avižienis, J.C. Laprie, B. Randell, and University of Newcastle upon Tyne. Computing Science. *Fundamental Concepts of Dependability*. Technical report series. University of Newcastle upon Tyne, Computing Science, 2001. URL: https://books.google.fr/books?id=cDkmGwAACAAJ.

[8] E. Balaji and P. Krishnamurthy. "Modeling ASIC memories in VHDL". In: *Design Automation Conference, 1996, with EURO-VHDL '96 and Exhibition, Proceedings EURO-DAC '96, European*. 1996, pp. 502–508. DOI: 10.1109/EURDAC.1996.558250.

[9] R.C. Baumann. "Radiation-induced soft errors in advanced semiconductor technologies". In: *Device and Materials Reliability, IEEE Transactions on* 5.3 (2005), pp. 305–316. ISSN: 1530-4388. DOI: 10.1109/TDMR.2005.853449.

[10] J. Benedetto et al. "Heavy ion-induced digital single-event transients in deep submicron Processes". In: *Nuclear Science, IEEE Transactions on* 51.6 (2004), pp. 3480–3485. ISSN: 0018-9499. DOI: 10.1109/TNS.2004.839173.

[11] S. Bhunia, S. Mukhopadhyay, and K. Roy. "Process Variations and Process-Tolerant Design". In: *VLSI Design, 2007. Held jointly with 6th International Conference on Embedded Systems., 20th International Conference on*. 2007, pp. 699–704. DOI: 10.1109/VLSID.2007.131.

[12] D. Binder, E.C. Smith, and A.B. Holman. “Satellite Anomalies from Galactic Cosmic Rays”. In: *Nuclear Science, IEEE Transactions on* 22.6 (1975), pp. 2675–2680. ISSN: 0018-9499. DOI: 10.1109/TNS.1975.4328188.

[13] A. Bosio and G. Di Natale. “LIFTING: A Flexible Open-Source Fault Simulator”. In: *Asian Test Symposium, 2008. ATS ’08. 17th*. 2008, pp. 35–40. DOI: 10.1109/ATS.2008.17.

[14] C. Bottoni, B. Coeffic, J.-M. Daveau, L. Naviner, and P. Roche. “Partial triplication of a SPARC-V8 microprocessor using fault injection”. In: *Circuits Systems (LASCAS), 2015 IEEE 6th Latin American Symposium on*. 2015, pp. 1–4. DOI: 10.1109/LASCAS.2015.7250415.

[15] D. Brooks and M. Martonosi. “Dynamic thermal management for high-performance microprocessors”. In: *High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on*. 2001, pp. 171–182. DOI: 10.1109/HPCA.2001.903261.

[16] B.H. Calhoun, A. Wang, N. Verma, and A. Chandrakasan. “Sub-Threshold Design: The Challenges of Minimizing Circuit Energy”. In: *Low Power Electronics and Design, 2006. ISLPED’06. Proceedings of the 2006 International Symposium on*. 2006, pp. 366–368. DOI: 10.1109/LPE.2006.4271869.

[17] Victor Castano and Igor Schagaev. *Resilient Computer System Design*. Springer Publishing Company, Incorporated, 2015. ISBN: 3319150685, 9783319150680.

[18] Chang-Chih Chen and L. Milor. “Microprocessor Aging Analysis and Reliability Modeling Due to Back-End Wearout Mechanisms”. In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 23.10 (2015), pp. 2065–2076. ISSN: 1063-8210. DOI: 10.1109/TVLSI.2014.2357756.

[19] G H Chisholm, Joseph Kljaich, B T Smith, and A S Wojcik. “An approach to the verification of a fault-tolerant, computer-based reactor safety system: A case study using automated reasoning: Volume 2, Appendixes: Interim report”. In: 1986.

[20] F. Corno, M.S. Reorda, and G. Squillero. “RT-level ITC’99 benchmarks and first ATPG results”. In: *Design Test of Computers, IEEE* 17.3 (2000), pp. 44–53. ISSN: 0740-7475. DOI: 10.1109/54.867894.

[21] Dale L. Critchlow. “Recollections on MOSFET Scaling”. In: *Solid-State Circuits Society Newsletter, IEEE* 12.1 (2007), pp. 19–22. ISSN: 1098-4232. DOI: 10.1109/N-SSC.2007.4785536.

[22] Andrew Danowitz, Kyle Kelley, James Mao, John P. Stevenson, and Mark Horowitz. “CPU DB: Recording Microprocessor History”. In: *Queue* 10.4 (Apr. 2012), 10:10–10:27. ISSN: 1542-7730. DOI: 10.1145/2181796.2181798. URL: http://doi.acm.org/10.1145/2181796.2181798.

[23] S. Das et al. “RazorII: In Situ Error Detection and Correction for PVT and SER Tolerance”. In: *Solid-State Circuits, IEEE Journal of* 44.1 (2009), pp. 32–48. ISSN: 0018-9200. DOI: 10.1109/JSSC.2008.2007145.

[24] S.E. Diehl-Nagle, J.E. Vinson, and E.L. Peterson. “Single Event Upset Rate Predictions for Complex Logic Systems”. In: *Nuclear Science, IEEE Transactions on* 31.6 (1984), pp. 1132–1138. ISSN: 0018-9499. DOI: 10.1109/TNS.1984.4333470.

[25] P.E. Dodd and L.W. Massengill. “Basic mechanisms and modeling of single-event upset in digital microelectronics”. In: *Nuclear Science, IEEE Transactions on* 50.3 (2003), pp. 583–602. ISSN: 0018-9499. DOI: 10.1109/TNS.2003.813129.

[26] P.E. Dodd, M.R. Shaneyfelt, J.A. Felix, and J.R. Schwank. “Production and propagation of single-event transients in high-speed digital logic ICs”. In: *Nuclear Science, IEEE Transactions on* 51.6 (2004), pp. 3278–3284. ISSN: 0018-9499. DOI: 10.1109/TNS.2004.839172.

[27] E. Dubrova. *Fault-Tolerant Design*. Springer New York, 2013. ISBN: 9781461421139. URL: https://books.google.fr/books?id=FRs_AAAAQBAJ.

[28] A. Dutta and A. Jas. “Combinational Logic Circuit Protection Using Customized Error Detecting and Correcting Codes”. In: *Quality Electronic Design, 2008. ISQED 2008. 9th International Symposium on*. 2008, pp. 68–73. DOI: 10.1109/ISQED.2008.4479700.

[29] D. Ernst et al. “Razor: a low-power pipeline based on circuit-level timing speculation”. In: *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*. 2003, pp. 7–18. DOI: 10.1109/MICRO.2003.1253179.

[30] M. Fazeli, S.N. Ahmadian, S.G. Miremadi, H. Asadi, and M.B. Tahoori. “Soft error rate estimation of digital circuits in the presence of Multiple Event Transients (METs)”. In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*. 2011, pp. 1–6. DOI: 10.1109/DATE.2011.5763020.

[31] Shuguang Feng, Shantanu Gupta, Amin Ansari, and Scott A. Mahlke. “Maestro: Orchestrating Lifetime Reliability in Chip Multiprocessors.” In: *HiPEAC*. Ed. by Yale N. Patt, Pierfrancesco Foglia, Evelyn Duesterwald, Paolo Faraboschi, and Xavier Martorell. Vol. 5952. Lecture Notes in Computer Science. Springer, Feb. 24, 2010, pp. 186–200. ISBN: 978-3-642-11514-1. URL: http://dblp.uni-trier.de/db/conf/hipeac/hipeac2010.html#FengGAM10.

[32] V. Ferlet-Cavrois, L.W. Massengill, and P. Gouker. “Single Event Transients in Digital CMOS-A Review”. In: *Nuclear Science, IEEE Transactions on* 60.3 (2013), pp. 1767–1790. ISSN: 0018-9499. DOI: 10.1109/TNS.2013.2255624.

[33] G. Fey, A. Sulflow, S. Frehse, and R. Drechsler. “Effective Robustness Analysis Using Bounded Model Checking Techniques”. In: *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 30.8 (2011), pp. 1239–1252. ISSN: 0278-0070. DOI: 10.1109/TCAD.2011.2120950.

[34] D.T. Franco, M.C. Vasconcelos, L. Naviner, and J.-F. Naviner. “Reliability analysis of logic circuits based on signal probability”. In: *Electronics, Circuits and Systems, 2008. ICECS 2008. 15th IEEE International Conference on*. 2008, pp. 670–673. DOI: 10.1109/ICECS.2008.4674942.

[35] Arthur L. Friedman, Bruce Lawton, Kenneth R. Hotelling, J.C. Pickel, Virgil H. Strahan, and Keith Loree. “Single Event Upset in Combinatorial and Sequential Current Mode Logic”. In: *Nuclear Science, IEEE Transactions on* 32.6 (1985), pp. 4216–4218. ISSN: 0018-9499. DOI: 10.1109/TNS.1985.4334097.

[36] R. Garg, C. Nagpal, and S.P. Khatri. “A fast, analytical estimator for the SEU-induced pulse width in combinational designs”. In: *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*. 2008, pp. 918–923.

[37] N. George and J. Lach. "Characterization of logical masking and error propagation in combinational circuits and effects on system vulnerability". In: *Dependable Systems Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*. 2011, pp. 323–334. DOI: 10.1109/DSN.2011.5958246.

[38] M. Goessel. *New Methods of Concurrent Checking*. Frontiers in Electronic Testing. Springer, 2008. ISBN: 9781402084201. URL: https://books.google.fr/books?id=WvN8iOWu9sMC.

[39] S. Hareland, J. Maiz, M. Alavi, K. Mistry, S. Walsta, and Changhong Dai. "Impact of CMOS process scaling and SOI on the soft error rates of logic processes". In: *VLSI Technology, 2001. Digest of Technical Papers. 2001 Symposium on*. 2001, pp. 73–74. DOI: 10.1109/VLSIT.2001.934953.

[40] Walter L Heimerdinger and Charles B Weinstock. *A conceptual framework for system fault tolerance*. Tech. rep. DTIC Document, 1992.

[41] John L. Hennessy and David A. Patterson. *Computer Architecture, Fifth Edition: A Quantitative Approach*. 5th. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011. ISBN: 012383872X, 9780123838728.

[42] Chang-Ming Hsieh, P.C. Murley, and R.R. O'Brien. "Collection of charge from alpha-particle tracks in silicon devices". In: *Electron Devices, IEEE Transactions on* 30.6 (1983), pp. 686–693. ISSN: 0018-9383. DOI: 10.1109/T-ED.1983.21190.

[43] C.M. Hsieh, P.C. Murley, and R.R. O'Brien. "A field-funneling effect on the collection of alpha-particle-generated carriers in silicon devices". In: *Electron Device Letters, IEEE* 2.4 (1981), pp. 103–105. ISSN: 0741-3106. DOI: 10.1109/EDL.1981.25357.

[44] Wei Huang, M.R. Stan, S. Gurumurthi, R.J. Ribando, and K. Skadron. "Interaction of scaling trends in processor architecture and cooling". In: *Semiconductor Thermal Measurement and Management Symposium, 2010. SEMI-THERM 2010. 26th Annual IEEE*. 2010, pp. 198–204. DOI: 10.1109/STHERM.2010.5444290.

[45] K. Iniewski. *Advanced Circuits for Emerging Technologies*. Wiley, 2012. ISBN: 9781118181492. URL: https://books.google.fr/books?id=yuhbbtDFDtoC.

[46] *International Technology Roadmap for Semiconductors*. 2013. URL: http://www.itrs.net/ITRS%201999-2014%20Mtgs,%20Presentations%20&%20Links/2013ITRS/Summary2013.htm (visited on 12/06/2015).

[47] B.W. Johnson. *Design and Analysis of Fault-tolerant Digital Systems*. Addison-Wesley series in electrical and computer engineering. Addison-Wesley Publishing Company, 1989. ISBN: 9780201075700. URL: https://books.google.fr/books?id=DOVQAAAAMAAJ.

[48] T. Karnik and P. Hazucha. "Characterization of soft errors caused by single event upsets in CMOS processes". In: *Dependable and Secure Computing, IEEE Transactions on* 1.2 (2004), pp. 128–143. ISSN: 1545-5971. DOI: 10.1109/TDSC.2004.14.

[49] M. Kooli and G. Di Natale. "A survey on simulation-based fault injection tools for complex systems". In: *Design Technology of Integrated Systems In Nanoscale Era (DTIS), 2014 9th IEEE International Conference On*. 2014, pp. 1–6. DOI: 10.1109/DTIS.2014.6850649.

[50] I. Koren and C.M. Krishna. *Fault-Tolerant Systems*. Elsevier Science, 2010. ISBN: 9780080492681. URL: https://books.google.fr/books?id=o_Pjbo4Wvp8C.

[51] R. Kumar and The University of Wisconsin Madison. *Temperature Adaptive and Variation Tolerant CMOS Circuits*. University of Wisconsin–Madison, 2008. ISBN: 9780549635109. URL: https://books.google.fr/books?id=Up4SN4XcgZgC.

[52] T. Lehtonen, J. Plosila, and J. Isoaho. *On Fault Tolerance Techniques Towards Nanoscale Circuits and Systems*. TUCS technical report. Turku Centre for Computer Science, 2005. ISBN: 9789521215964. URL: https://books.google.fr/books?id=fVaUAwAACAAJ.

[53] P. Liden, P. Dahlgren, R. Johansson, and J. Karlsson. "On latching probability of particle induced transients in combinational networks". In: *Fault-Tolerant Computing, 1994. FTCS-24. Digest of Papers., Twenty-Fourth International Symposium on*. 1994, pp. 340–349. DOI: 10.1109/FTCS.1994.315626.

[54] Benjamin Lussier, Raja Chatila, Felix Ingrand, Marc-olivier Killijian, and David Powell. "On Fault Tolerance and Robustness in Autonomous Systems". In: *In proceedings of the third IARP/IEEE-RAS/EURON joint workshop on technical challange for dependable robots in human Environments*. Citeseer. 2004.

[55] Robert E Lyons and Wouter Vanderkulk. "The use of triple-modular redundancy to improve computer reliability". In: *IBM Journal of Research and Development* 6.2 (1962), pp. 200–209.

[56] M. Maniatakos and Y. Makris. "Workload-driven selective hardening of control state elements in modern microprocessors". In: *VLSI Test Symposium (VTS), 2010 28th*. 2010, pp. 159–164. DOI: 10.1109/VTS.2010.5469589.

[57] M.-C.V. Marinescu and M. Rinard. "High-level automatic pipelining for sequential circuits". In: *System Synthesis, 2001. Proceedings. The 14th International Symposium on*. 2001, pp. 215–220. DOI: 10.1109/ISSS.2001.156561.

[58] L.W. Massengill and P.W. Tuinenga. "Single-Event Transient Pulse Propagation in Digital CMOS". In: *Nuclear Science, IEEE Transactions on* 55.6 (2008), pp. 2861–2871. ISSN: 0018-9499. DOI: 10.1109/TNS.2008.2006749.

[59] J. Mathew, R.A. Shafik, and D.K. Pradhan. *Energy-Efficient Fault-Tolerant Systems*. Springer New York, 2013. ISBN: 9781461441922. URL: https://books.google.fr/books?id=lYmRMQEACAAJ.

[60] T.C. May and Murray H. Woods. "Alpha-particle-induced soft errors in dynamic memories". In: *Electron Devices, IEEE Transactions on* 26.1 (1979), pp. 2–9. ISSN: 0018-9383. DOI: 10.1109/T-ED.1979.19370.

[61] T.C. May, G.L. Scott, E.S. Meieran, P. Winer, and V.R. Rao. "Dynamic Fault Imaging of VLSI Random Logic Devices". In: *Reliability Physics Symposium, 1984. 22nd Annual*. 1984, pp. 95–108. DOI: 10.1109/IRPS.1984.362025.

[62] M. Mehrara, M. Attariyan, S. Shyam, K. Constantinides, V. Bertacco, and T. Austin. "Low-Cost Protection for SER Upsets and Silicon Defects". In: *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*. 2007, pp. 1–6. DOI: 10.1109/DATE.2007.364449.

[63] S. Mitra and E.J. McCluskey. "Which concurrent error detection scheme to choose ?" In: *Test Conference, 2000. Proceedings. International*. 2000, pp. 985–994. DOI: 10.1109/TEST.2000.894311.

[64] K. Mohanram. “Closed-form simulation and robustness models for SEU-tolerant design”. In: *VLSI Test Symposium, 2005. Proceedings. 23rd IEEE*. 2005, pp. 327–333. DOI: 10.1109/VTS.2005.35.

[65] K. Mohanram and N.A. Touba. “Cost-effective approach for reducing soft error failure rate in logic circuits”. In: *Test Conference, 2003. Proceedings. ITC 2003. International*. Vol. 1. 2003, pp. 893–901. DOI: 10.1109/TEST.2003.1271075.

[66] Helia Naeimi and André DeHon. “Fault-tolerant sub-lithographic design with rollback recovery”. In: *Nanotechnology* 19.11 (2008), p. 115708.

[67] NanGate. *NanGate 45nm Open Cell Library*. 2011. URL: http://www.nangate.com/?page_id=2325 (visited on 11/03/2015).

[68] John von Neumann. “Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components”. In: *Automata Studies* (1956). Ed. by C. Shannon, pp. 43–98.

[69] M. Nicolaidis. *Soft Errors in Modern Electronic Systems*. Frontiers in Electronic Testing. Springer US, 2010. ISBN: 9781441969934. URL: https://books.google.fr/books?id=WCqrOkMExu8C.

[70] Y. Nishi and R. Doering. *Handbook of Semiconductor Manufacturing Technology, Second Edition*. CRC Press, 2007. ISBN: 9781420017663. URL: https://books.google.fr/books?id=PsVVKz_hjBgC.

[71] S. Oda and D.K. Ferry. *Nanoscale Silicon Devices*. Taylor & Francis, 2015. ISBN: 9781482228670. URL: https://books.google.fr/books?id=LMg-rgEACAAJ.

[72] Samuel N. Pagliarini, Lirida A.de B. Naviner, and J.-F. Naviner. “Selective hardening methodology for combinational logic”. In: *Test Workshop (LATW), 2012 13th Latin American*. 2012, pp. 1–6. DOI: 10.1109/LATW.2012.6261262.

[73] Janak H. Patel. “Manufacturing Process Variations and Dependability - A Contrarian View”. In: *Dependable and Secure Nanocomputing (DSN), 2007 Workshop on*. 2007.

[74] W.W. Peterson and E.J. Weldon. *Error-correcting Codes*. MIT Press, 1972. ISBN: 9780262160391. URL: https://books.google.fr/books?id=5kfwlFeklx0C.

[75] I. Polian and J.P. Hayes. “Selective Hardening: Toward Cost-Effective Error Tolerance”. In: *Design Test of Computers, IEEE* 28.3 (2011), pp. 54–63. ISSN: 0740-7475. DOI: 10.1109/MDT.2010.120.

[76] I. Polian, S.M. Reddy, and B. Becker. “Scalable Calculation of Logical Masking Effects for Selective Hardening Against Soft Errors”. In: *Symposium on VLSI, 2008. ISVLSI ’08. IEEE Computer Society Annual*. 2008, pp. 257–262. DOI: 10.1109/ISVLSI.2008.22.

[77] Karl Rupp. *40 Years of Microprocessor Trend Data*. 2015. URL: https://www.karlrupp.net/2015/06/40-years-of-microprocessor-trend-data/.

[78] John Rushby. *Formal Methods and the Certification of Critical Systems*. Tech. rep. SRI-CSL-93-7. Also issued under the title *Formal Methods and Digital Systems Validation for Airborne Systems* as NASA Contractor Report 4551, December 1993. Menlo Park, CA: Computer Science Laboratory, SRI International, Dec. 1993.

[79] M. Sachdev. *Defect Oriented Testing for CMOS Analog and Digital Circuits*. Frontiers in Electronic Testing. Springer US, 2013. ISBN: 9781475749267. URL: https://books.google.fr/books?id=0bPaBwAAQBAJ.

[80] J. Samandari-Rad, M. Guthaus, and R. Hughey. “Confronting the Variability Issues Affecting the Performance of Next-Generation SRAM Design to Optimize and Predict the Speed and Yield”. In: *Access, IEEE* 2 (2014), pp. 577–601. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2014.2323233.

[81] M. Santoro. “New methodologies for eliminating No Trouble Found, No Fault Found and other non repeatable failures in depot settings”. In: *AUTOTESTCON, 2008 IEEE*. 2008, pp. 336–340. DOI: 10.1109/AUTEST.2008.4662636.

[82] S.S. Sapatnekar. “Overcoming Variations in Nanometer-Scale Technologies”. In: *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on* 1.1 (2011), pp. 5–18. ISSN: 2156-3357. DOI: 10.1109/JETCAS.2011.2138250.

[83] L. Scheffer, L. Lavagno, and G. Martin. *EDA for IC Implementation, Circuit Design, and Process Technology*. EDA for IC Implementation, Circuit Design, and Process Technology. Taylor & Francis, 2006. ISBN: 9780849379246. URL: https://books.google.fr/books?id=bHAeAQAAIAAJ.

[84] J. Segura and C.F. Hawkins. *CMOS Electronics: How It Works, How It Fails*. Wiley, 2004. ISBN: 9780471476696. URL: https://books.google.fr/books?id=Vl8naJHBTDIC.

[85] P. Shivakumar and The University of Texas at Austin. Computer Sciences. *Techniques to Improve the Hard and Soft Error Reliability of Distributed Architectures*. University of Texas at Austin, 2007. ISBN: 9780549172109. URL: https://books.google.fr/books?id=DYX5AVBYwKgC.

[86] 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, 2002. DSN 2002. Proceedings. International Conference on*. 2002, pp. 389–398. DOI: 10.1109/DSN.2002.1028924.

[87] J. Srinivasan, S.V. Adve, P. Bose, and J.A. Rivers. “The case for lifetime reliability-aware microprocessors”. In: *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*. 2004, pp. 276–287. DOI: 10.1109/ISCA.2004.1310781.

[88] J. Srinivasan, S.V. Adve, P. Bose, and J.A. Rivers. “The impact of technology scaling on lifetime reliability”. In: *Dependable Systems and Networks, 2004 International Conference on*. 2004, pp. 177–186. DOI: 10.1109/DSN.2004.1311888.

[89] V. Subramanian and A.K. Somani. “Conjoined Pipeline: Enhancing Hardware Reliability and Performance through Organized Pipeline Redundancy”. In: *Dependable Computing, 2008. PRDC ’08. 14th IEEE Pacific Rim International Symposium on*. 2008, pp. 9–16. DOI: 10.1109/PRDC.2008.54.

[90] Anjana Suresh and S. Sabi. “Advanced Error Recovery for TMR Systems”. In: *International Journal of Advanced Technology in Engineering and Science* 2.8 (2014), pp. 408–419. URL: http://ijates.com/images/short_pdf/1408988043_P408-419.pdf.

[91] D. A. Tran et al. “A New Hybrid Fault-Tolerant Architecture for Digital CMOS Circuits and Systems”. In: *J. Electron. Test.* 30.4 (Aug. 2014), pp. 401–413. ISSN: 0923-8174. DOI: 10.1007/s10836-014-5459-3. URL: http://dx.doi.org/10.1007/s10836-014-5459-3.

[92] D.A. Tran et al. “A Hybrid Fault Tolerant Architecture for Robustness Improvement of Digital Circuits”. In: *Test Symposium (ATS), 2011 20th Asian*. 2011, pp. 136–141. DOI: 10.1109/ATS.2011.89.

[93] D.A. Tran et al. "A pseudo-dynamic comparator for error detection in fault tolerant architectures". In: *VLSI Test Symposium (VTS), 2012 IEEE 30th*. 2012, pp. 50–55. DOI: 10.1109/VTS.2012.6231079.

[94] J. Velamala, R. LiVolsi, M. Torres, and Yu Cao. "Design sensitivity of Single Event Transients in scaled logic circuits". In: *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*. 2011, pp. 694–699.

[104] Duane Galbi Wilson Snyder and Paul Wasson. *Introduction to Verilator*. 2015. URL: http://www.veripool.org/wiki/verilator.

[105] M. Wirnshofer. *Variation-Aware Adaptive Voltage Scaling for Digital CMOS Circuits*. Springer Series in Advanced Microelectronics. Springer Netherlands, 2015. ISBN: 9789401783675. URL: https://books.google.fr/books?id=_UUyrgEACAAJ.

[106] Pei-Ci Wu, M.D.F. Wong, I. Nedelchev, S. Bhardwaj, and V. Parkhe. "On timing closure: Buffer insertion for hold-violation removal". In: *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*. 2014, pp. 1–6. DOI: 10.1145/2593069.2593171.

[107] Jun Yao, H. Shimada, and K. Kobayashi. "[2009] A Stage-Level Recovery Scheme in Scalable Pipeline Modules for High Dependability". In: *Innovative Architecture for Future Generation High Performance (IWIA), 2010 International Workshop on*. 2010, pp. 21–29. DOI: 10.1109/IWIA.2010.11.

[108] Jun Yao, S. Okada, M. Masuda, K. Kobayashi, and Y. Nakashima. "DARA: A Low-Cost Reliable Architecture Based on Unhardened Devices and Its Case Study of Radiation Stress Test". In: *Nuclear Science, IEEE Transactions on* 59.6 (2012), pp. 2852–2858. ISSN: 0018-9499. DOI: 10.1109/TNS.2012.2223715.

[109] Quming Zhou and K. Mohanram. "Gate sizing to radiation harden combinational logic". In: *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 25.1 (2006), pp. 155–166. ISSN: 0278-0070. DOI: 10.1109/TCAD.2005.853696.

[110] J. F. Ziegler and W. A. Lanford. "Effect of Cosmic Rays on Computer Memories". In: *Science*. Vol. 206. 4420. 1979, pp. 776–788. URL: http://www.sciencemag.org/cgi/content/abstract/206/4420/776.

[111] J. F. Ziegler, H. W. Curtis, H. P. Muhlfeld, C. J. Montrose, and B. Chin. "IBM Experiments in Soft Fails in Computer Electronics (1978&Ndash;1994)". In: *IBM J. Res. Dev.* 40.1 (Jan. 1996), pp. 3–18. ISSN: 0018-8646. DOI: 10.1147/rd.401.0003. URL: http://dx.doi.org/10.1147/rd.401.0003.

[112] C.G. Zoellin, H. Wunderlich, I. Polian, and B. Becker. "Selective Hardening in Early Design Steps". In: *Test Symposium, 2008 13th European*. 2008, pp. 185–190. DOI: 10.1109/ETS.2008.30.

Appendix A

HyFT Control Logic

The HyFT architecture is controlled by a control logic module having two submodules as shown in Figure A.1. The first submodule (named as *submodule1*) is responsible to generate time critical control signals for the input register, which include *CRegin* and *CLKRegin*, and for the pseudo-dynamic comparator, which include *DC* and *CompReset*. The functional description of these control signals is given in the second column of Table A.1.

The fixed logic shown in Figure A.2 when synthesized with appropriate delay constraints serves as *submodule1*. The circuit elements of *submodule1* that play part in the generation of each time critical control signal are listed in the third column of Table A.1 and the implication of *submodule1* circuit elements delays on the timing of these signals is given in the last column of Table A.1.

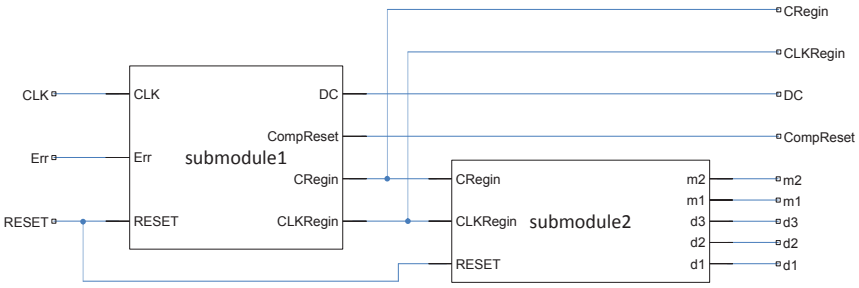


Fig. A.1: HyFT Control Logic

HyFT control logic *submodule2* holds the FSM that manages the reconfigurations in HyFT architecture by deciding which pair of CL copies to run and which one to keep in standby mode. A simplified illustration of *submodule2* is shown in Figure A.4. The label on each state represents the active CL pair, for example, state *1-2* is the one in which *CL1* and *CL2* run and

Table A.1: HyFT *submodule1* control signals

Control Signal	**Description**		**Generating circuit elements (of Figure A.2)**	**Timing (based on the circuit elements of Figure A.2)**
DC	A periodic signal that defines the time interval during which the pseudo-dynamic comparator compares for error detection (*comparison-window*) by being at logic level-1.		*buf1* & *buf2*	• $\Delta(CLK\uparrow, DC\uparrow) = d_{\text{buf1}} + d_{\text{buf2}}$ • $\Delta(CLK\downarrow, DC\downarrow) = d_{\text{buf1}} + d_{\text{buf2}}$
CRegin	This signal indicates a reconfiguration cycle by staying at logic level-1.	• If *CRegin* is at logic level-1 and *CLKRegin* undergoes a falling transition, the state machine reconfigures. • If *CRegin* is at logic level-1 and *CLKRegin* acquires low logic level, the shadow latch in the input register preserves the last state to be used for rollback. • If *CRegin* is at logic level-1 and *CLK* undergoes a rising edge, the input register captures data from shadow latch instead of capturing a new input sample.	*buf2*, *and* & *ff1*	• $\Delta(Err\uparrow, CRegin\uparrow) = d_{\text{and}}$ • $\Delta(CLK\uparrow, CRegin\downarrow) = d_{\text{ccq}} + d_{\text{and}} + d_{\text{buf2}}$
CLKRegin	A periodic signal used to trigger reconfiguration and rollback depending on the state of *CRegin*.		*buf2*, *inv1* & *or1*	• $\Delta(CLK\downarrow, CLKRegin\downarrow) = d_{\text{buf2}} + d_{\text{buf2}}$ • $\Delta(CLK\downarrow, CLKRegin\uparrow) = d_{\text{inv1}} + d_{\text{or1}}$
CompReset	An active-low reset to the pseudo-dynamic comparator. Resets the pseudo-dynamic comparator when the reconfiguration and recomputation cycles finish.		*inv2*, *nand*, *or2* & *ff1*	• $\Delta(CLK\uparrow, CompReset\downarrow) = d_{\text{nand}} + d_{\text{or2}}$ • $\Delta(CLK\uparrow, CompReset\downarrow) = d_{\text{nand}} + d_{\text{or2}}$

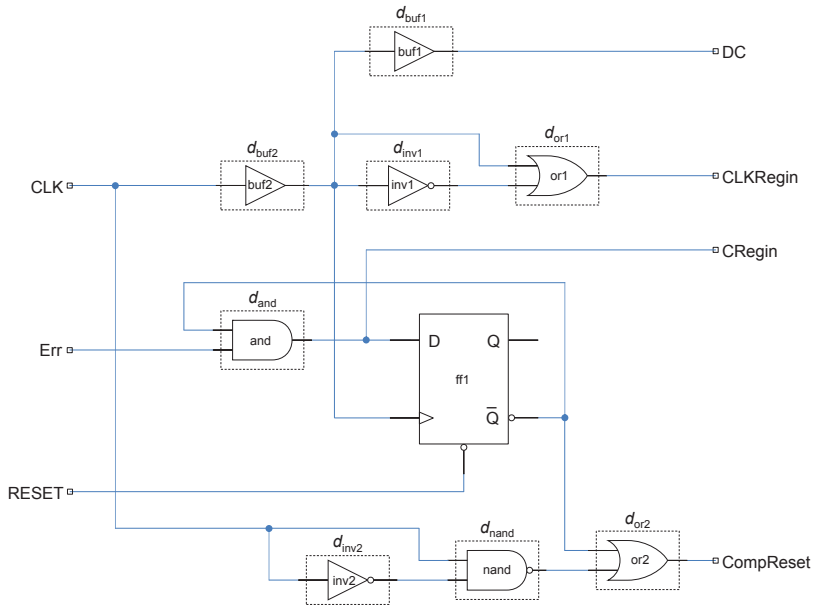


Fig. A.2: HyFT Control Logic *submodule1*

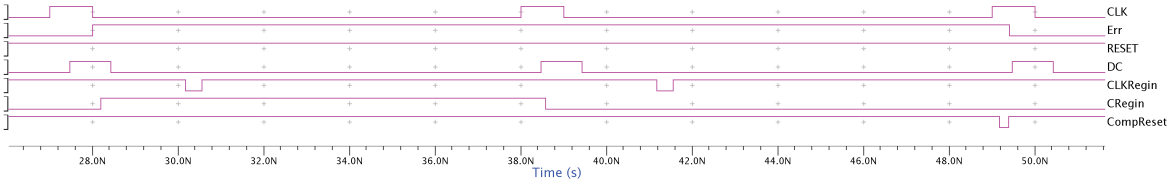


Fig. A.3: HyFT Control Logic *submodule1* simulation

CL3 stays in standby. It changes state if during that cycle an error is detected. Table A.2 gives the value control signals generated by *submodule2* FSM to control the reconfiguration multiplexer and demultiplexer. Possible customizations of the *submodule2* FSM aimed to optimize the performance by adapting to type of targeted faults, are presented in [92].

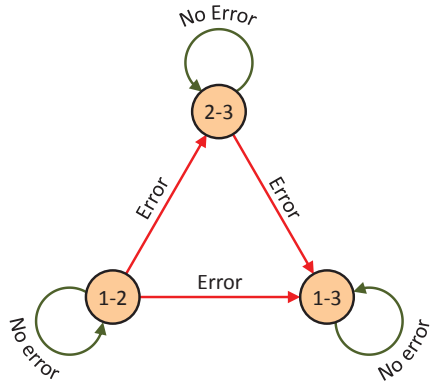


Fig. A.4: HyFT Control Logic *submodule2* FSM

Table A.2: HyFT *submodule2* control signals

State	m1	m2	d1	d2	d3	Active CLs
1-2	0	1	1	1	0	CL1, CL2
1-2	1	0	0	1	1	CL2, CL3
1-3	0	0	1	0	1	CL1, CL3

Appendix B

Workload Program

The assembly workload program used with different versions of MIPS microprocessor, for their power estimation and fault-injection experiments is given below:

```
; multiply 16 bits unsigned numbers r1 * r2
; input
addi r1, r0, #6 ; multiplicand
addi r2, r0, #4 ; multiplier

; output
add r3, r0, r0

; iteration counter
addi r4, r0, 0

; algorithm start
iteration:
sll  r5, r1, r4 ; shift left multiplicand #(iteration counter) amount
srl  r6, r2, r4 ; shift right multiplier #(iteration counter) amount
andi r7, r6, #1 ; mask multiplier rightmost bit
beqz r7, noadd ; if rightmostbit is 0 don't add
add r3, r3, r5 ; if rightmostbit is 1 add
noadd:
addi r4, r4, #1 ; increment iteration counter
sgei r8, r4, #16 ; set bit if iteration is >= 16
beqz r8, iteration
```

```
nop
nop
nop
; we have result in r3!
```

Appendix C

CL extraction

As we have seen in Chapter 2, combinational parts are extracted from ITC'99 benchmark circuits by removing all D flip-flops from the original netlist. For each flip-flop removed, a new primary output (nPO) and a new primary input (nPI) are added.

An example of combinational logic extraction is detailed for circuit b01 of ITC'99 benchmark in Table C.1. In this example, differences in the extracted netlist compared the orginal netlist are in bold.

Table C.1: An example of CL Extraction

Original netlist	Netlist of extracted CL part
module b01 (LINE1, LINE2, OUTP_REG, OVERFLW_REG, CLK);	module b01 (LINE1, LINE2, OUTP_REG, OVERFLW_REG, CLK, **nPI1, nPI2, nPI3, nPI4, nPI5, nPO1, nPO2, nPO3, nPO4, nPO5**);
input LINE1, LINE2, CLK;	input LINE1, LINE2, CLK;
output OUTP_REG, OVERFLW_REG;	output OUTP_REG, OVERFLW_REG;
	input nPI1, nPI2, nPI3, nPI4, nPI5;
	output nPO1, nPO2, nPO3, nPO4, nPO5;
//Begin sequential part	//Begin sequential part
dff dff_1 (STATO_REG_2_, U34, CLK);	**buf buf_1 (nPO1, U34);** **buf buf_2 (STATO_REG_2_, nPI1);**
dff dff_2 (STATO_REG_1_, U35, CLK);	**buf buf_3 (nPO2, U35);** **buf buf_4 (STATO_REG_1_, nPI2);**
dff dff_3 (STATO_REG_0_, U36, CLK);	**buf buf_5 (nPO3, U36);** **buf buf_6 (STATO_REG_0_, nPI3);**
dff dff_4 (OUTP_REG, U37, CLK);	**buf buf_7 (nPO4, U37);** **buf buf_8 (OUTP_REG, nPI4);**
dff dff_5 (OVERFLW_REG, U48, CLK);	**buf buf_9 (nPO5, U48);** **buf buf_10 (OVERFLW_REG, nPI5);**
// End sequential part	//End sequential part
//Begin combinational part .... //End combinational part	//Begin combinational part .... //End combinational part
endmodule	endmodule