CUSTOM HARDWARE OPTIMIZATIONS FOR RELIABLE AND HIGH PERFORMANCE COMPUTER ARCHITECTURES

A DISSERTATION SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

COMPUTER ENGINEERING

By

Hamzeh Ahangari

September 2020

Custom Hardware Optimizations for Reliable and High Performance Computer Architectures
By Hamzeh Ahangari
September 2020

We certify that we have read this dissertation and that in our opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Özcan Öztürk(Advisor)

Uğur Güdükbay

Fazlı Can

Şenan Ece Schmidt

Süleyman Tosun

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

Copyright Information

ABSTRACT

CUSTOM HARDWARE OPTIMIZATIONS FOR RELIABLE AND HIGH PERFORMANCE COMPUTER ARCHITECTURES

Hamzeh Ahangari
Ph.D. in Computer Engineering
Advisor: Özcan Öztürk
September 2020

In recent years, we have witnessed a huge wave of innovations, such as in Artificial Intelligence (AI) and Internet-of-Things (IoT). In this trend, software tools are constantly and increasingly demanding more processing power, which can no longer be met by processors traditionally. In response to this need, a diverse range of hardware, including GPUs, FPGAs, and AI accelerators, are coming to the market every day. On the other hand, while hardware platforms are becoming more power-hungry due to higher performance demand, concurrent reduction in the size of transistors, and placing high emphasis on reducing the voltage, altogether have always been sources of reliability concerns in circuits. This particularly is applicable to error-sensitive applications, such as transportation and aviation industries where an error can be catastrophic. The reliability issues may have other reasons too, like harsh environmental conditions. These two problems of modern electronic circuits, meaning the need for higher performance and reliability at the same time, require appropriate solutions. In order to satisfy both the performance and the reliability constraints either designs based on reconfigurable circuits, such as FPGAs, or designs based on Commercial-Off-The-Shelf (COTS) components like general-purpose processors, can be an appropriate approach because the platforms can be used in a wide variety of applications. In this regard, three solutions have been proposed in this thesis. These solutions target 1) safety and reliability at the system-level using redundant processors, 2) performance at the architecture-level using multiple accelerators, and 3) reliability at the circuit-level through the use of redundant transistors. Specifically, in the first work, the contribution of some prevalent parameters in the design of safety-critical computers, using COTS processors, is discussed. Redundant architectures are modeled by the Markov chains, and sensitivity of system safety to parameters has been analyzed. Most importantly, the significant presence of Common Cause

Failures (CCFs) has been investigated. In the second work, the design, and implementation of an HLS-based, FPGA-accelerated, high-throughput/work-efficient, synthesizable template-based graph processing framework has been presented. The template framework is simplified for easy mapping to FPGA, even for software programmers. The framework is particularly experimented on Intel state-of-the-art Xeon+FPGA platform to implement iterative graph algorithms. Beside high-throughput pipeline, work-efficient mode significantly reduces total graph processing run-time with a novel active-list design. In the third work, Joint SRAM (JSRAM) cell, a novel circuit-level technique to exploit the trade-off between reliability and memory size, is introduced. This idea is applicable to any SRAM structure like cache memory, register file, FPGA block RAM, or FPGA look-up table (LUT), and even latches and Flip-Flops. In fault-prone conditions, the structure can be configured in such a way that four cells are combined together at the circuit level to form one large and robust memory bit. Unlike prevalent hardware redundancy techniques, like Triple Modular Redundancy (TMR), there is no explicit majority voter at the output. The proposed solution mainly focuses on transient faults, where the reliable mode can provide auto-correction and full immunity against single faults.

Keywords: Graph Processing, Hardened SRAM, Hardware Accelerator, High-Level Synthesis, IEC 61508, Markov Modeling, Reliability, Safety-Critical Computer, Redundancy, Xeon+FPGA.

ÖZET

GÜVENİLİR VE YÜKSEK PERFORMANSLI BİLGİSAYAR MİMARİLERİ İÇİN ÖZEL DONANIM OPTİMİZASYONLARI

Hamzeh Ahangari
Bilgisayar Mühendisliği, Doktora
Tez Danışmanı: Özcan Öztürk
Eylül 2020

Son yıllarda, Yapay Zeka (AI) ve Nesnelerin İnterneti (IoT) gibi büyük bir yenilik dalgasına tanık olduk. Bu akımda, yazılım araçları sürekli artan işlem gücü talep ediyor ve bu artık geleneksel işlemciler tarafından karşılanamıyor. Bu ihtiyaca yanıt olarak, Grafik İşleme Üniteler (GPU'lar), Alanda Programlanabilir Kapı Diziler (FPGA'lar) ve Yapay Zeka (AI) hızlandırıcılar dahil olmak üzere çok çeşitli donanımlar her gün piyasaya sürülüyor. Öte yandan, donanım platformları daha yüksek performans talebi nedeniyle daha fazla güce aç hale gelirken, eşzamanlı olarak transistör boyutunun küçülmesi ve voltajın azaltılması, devrelerde her zaman güvenilirlik endişelerini arttırmıştır. Bu, özellikle bir hatanın felaketle sonuçlanabileceği ulaşım ve havacılık endüstrileri gibi hataya duyarlı uygulamalar için geçerlidir. Güvenilirlik konularının, sert çevre koşulları gibi başka nedenleri de olabilir. Modern elektronik devrelerin bu iki sorunu, yani aynı anda daha yüksek performans ve güvenilirlik ihtiyacı, uygun çözümler gerektirir. FPGA'lar gibi yeniden yapılandırılabilir devrelere dayalı tasarım veya genel amaçlı işlemciler gibi Piyasadan Hazır Temin Edilebilen (COTS) bileşenlere dayalı tasarım uygun bir yaklaşım olabilir, çünkü bu platformlar çok çeşitli uygulamalarda kullanılabilir. Bu bağlamda, bu tezde üç çözüm önerilmiştir. Bu çözümler, 1) yedekli işlemciler kullanarak sistem düzeyinde güvenlik ve güvenilirliği, 2) birden çok hızlandırıcı kullanarak mimari düzeyinde performansı ve 3) yedekli transistörlerin kullanımıyla devre düzeyinde güvenilirliği hedefler. Özel olarak, ilk çalışmada, Piyasadan Hazır Temin Edilebilen (COTS) işlemcileri kullanarak güvenlik açısından kritik bilgisayarların tasarımında bazı yaygın parametrelerin katkısı tartışılmıştır. Yedekli mimariler Markov zinciri kullanılarak modellenmiştir ve sistem güvenliğinin parametrelere duyarlılığı analiz edilmiştir. En önemlisi, bu tür sistemlerde Yaygın Neden Arızalarının (CCF'ler) önemli varlığı araştırılmıştır. İkinci çalışmada, Yüksek Seviyeli Sentez (HLS) tabanlı, FPGA

hızlandırmalı, yüksek verimli / iş verimli, sentezlenebilir şablon tabanlı grafik işleme ünitesinin tasarımı ve uygulaması sunulmuştur. Bu ünite yazılım programcıları için bile FPGA ile kolay programlayabilme için basitleştirilmiştir. Sunulan yapı, yinelemeli grafik algoritmalarını uygulamak için özel olarak Intel'in son teknoloji ürünü Xeon + FPGA platformunda denenmiştir. Yüksek verimli boru hattının yanı sıra, iş verimli mod, yeni bir etkin liste tasarımıyla toplam grafik işleme süresini önemli ölçüde azaltır. Üçüncü çalışmada, güvenilirlik ve bellek boyutu arasında tercih yapmak için devre düzeyinde yeni bir teknik olan Ortak SRAM hücresi tanıtılmıştır. Bu fikir, ön-bellek, kayıt dosyası, FPGA BRAM veya FPGA arama tablosu (LUT) ve hatta mandallar ve Flip-Floplar gibi herhangi bir SRAM yapısı için de uygulanabilir. Hataya eğilimli koşullarda, yapı, dört hücrenin bir büyük ve sağlam bellek biti oluşturmak üzere devre seviyesinde birleştirileceği şekilde yapılandırılabilir. Üçlü Modüler Yedeklilik (TMR) gibi yaygın donanım yedekliliği tekniklerinin aksine, belirgin bir seçim ünitesi yoktur. Çözüm, temel olarak, güvenilir modun otomatik düzeltme ve tek hatalara karşı tam bağışıklık sağlayabildiği geçici hatalara odaklanır.

Anahtar sözcükler: Çizge İşleme, Güçlendirilmiş SRAM, Donanım Hızlandırıcı, Üst Düzey Sentez, IEC 61508, Markov Modelleme, Güvenilirlik, Güvenlik Açısından Kritik Bilgisayar, Yedeklilik, Xeon+FPGA.

Acknowledgement

The past years at Bilkent University will be a part of the most memorable, pleasant, and fruitful years of my life. The productive academic activities and peaceful life on the campus, which will always be like home to me, will forever be part of my memories. After endless thanks to God Almighty, for all his graces and mercies in all stages of my life, I would like to thank all people, who with their valuable support and assistance, completion of this thesis was made possible.

First and foremost, I would like to express my sincere and deep gratitude to my advisor Prof. Özcan Öztürk for his invaluable and continuous support, understanding, and guidance during my Ph.D. study and research. It has always been a pleasure for me to work with him. Second, I would like to thank the members of my thesis committee, Prof. Mustafa Özdal, Prof. Süleyman Tosun, and Prof. Uğur Güdükbay for spending valuable time on meticulously reading each and every progress report. Also, I convey my thanks to Prof. Fazlı Can and Prof. Şenan Ece Schmidt for accepting to read and review this thesis, and their insightful comments and suggestions.

I am grateful to the Bilkent CS department for the facilities they provided, especially financial support during my study. It was an honor and a privilege to be a member of the Bilkent CS family, and I would like to thank each member of the department. I would like to thank all of my office mates, particularly Naveed UI Mustafa, for the friendly and constructive times we spent together. I would like to thank the Scientific and Technological Research Council of Turkey (TÜBİTAK) 1001 program for supporting me during my Ph.D. studies in the EEEAG-115E835 project.

Above all, I would like to thank my family, especially my dear wife, Hajar, for her everlasting love, endless support, and understanding.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In modern times, computers have been invented to help humans. The computational power of these calculating machines is millions of times faster than the power of the human brain. Traditionally, designed software algorithms run upon these machines. However, with the rapid development of information technology, artificial intelligence and numerous branches of science, this traditional approach has changed. Software running on general-purpose processors is no longer the only solution that meets the growing needs for higher computation power, less power consumption, more accuracy, and more reliability. In many applications, customized hardware, designed specifically for a specific purpose, are a great help to processors. Widely used multimedia encoders and decoders, graphic accelerators, cryptographers, artificial intelligence accelerators, and many more are examples of such hardware, designed to provide higher computational power beyond the conventional processors.

Processors and other hardware, particularly designed for error-sensitive applications such as passenger planes, trains, satellites, and military equipment are other examples of the need for special, non-generic, and usually expensive hardware. On the other hand, in recent years, long time-to-market and costly design processes have made customized hardware not to be affordable for designs with limited production volume. Instead, using Commercial Off-The-Shelf (COTS)

components is the prevalent approach, where hardware systems based on commercial devices such as generic processors, or field-programmable gate arrays (FPGAs) are widely used in the industry. Therefore, having reliability solutions based on COTS components can be economically beneficial. In this dissertation, we consider the design of specific hardware to help increase the safety, reliability, or performance. Our solutions are targeting these objectives on three different levels, namely, system-level, architecture-level, and circuit-level. In all of the proposed techniques, we utilize resource replication, including data, processors, or accelerators.

1.1 Research Problems and Motivations

Nowadays, safety-critical computers are obligatory constituents of many electronic systems that affect human life safety. Several areas of the transportation industry like railways, avionics, and automotive, increasingly use such systems. To design a computer for safety-critical applications, industrial safety levels such as international safety standard IEC 61508 (shown in Table 1.1) have been set. In this domain, safe microcontrollers with limited processing capabilities are available in the market for mostly control purposes. However, as systems become more and more complex and versatile, having safe processors with intensive processing capabilities becomes an essential need. According to IEC 61508-2 standard, a single processor can achieve at most Safety Integrity Level 3 (SIL3). In most cases, safety-critical applications in civil domains require a higher level of safety, such as SIL4. Hence, to answer this eminent need, a computing platform needs to be architected in system-level with safety in mind. In order to achieve such high standards, it is necessary to make improvements in numerous aspects of a general-purpose system. The reliability of electronic components is the most obvious factor that needs to be satisfied when building a robust system. Besides, clever system design by means of available electronic components is as important as the quality of components themselves. Even with reliable and robust parts, safety goals may not be achieved without a safety aware design process. Prevalent design issues like a perfect printed circuit board, EMC/EMI isolation, power

10^{-9}	$\leqslant$	PFH of SIL4	$<$	10^{-8}
10^{-8}	$\leqslant$	PFH of SIL3	$<$	10^{-7}
10^{-7}	$\leqslant$	PFH of SIL2	$<$	10^{-6}
10^{-6}	$\leqslant$	PFH of SIL1	$<$	10^{-5}

Table 1.1: Safety levels in IEC 61508 standard for high demand/continuous systems (PFH: average frequency of a dangerous failure of safety function per hour, SIL: Safety Integrity Level) [2].

circuitry, failure rates of equipment, etc., are examples of common quality considerations. However, in critical systems, in addition to these, some other less obvious issues have to be observed. Chapter 2 is dedicated to elaborate on design parameters of safety-critical computers.

Recent years have seen a massive interest in Artificial Intelligence (AI), due to a sharp increase in demand from industry, such as security, finance, healthcare, and military. In many applications, AI helps substantially by providing higher levels of precision and improving productivity. In this enormous wave of evolution, many companies, including giants of the computer industry, invest considerable resources in AI technology. As we see, new software and hardware products are released daily, including FPGA accelerated CPUs, deep-learning models, neural network accelerators, and AI SDKs targeting a various range of hardware platforms, from CPU and GPU to FPGA and VPU. Meanwhile, graph theory, as one of the most fundamental modeling techniques in data science, contributes to many of the current AI applications. Specifically, graphs provide a better context for machine learning algorithms due to the way data is organized, enabling relationships of numerous degrees to be analyzed quickly. Problems that can be modeled as communicating entities, such as the Internet of Things (IoT), social networks, web search, transportation, health-care systems [5], and even biology [6] are examples for which graph data models are an excellent fit. Many of these AI algorithms, such as deep-learning-based computer vision, are not only compute-intensive but also operate on enormous data-sets. According to official statistics, as of the first quarter of 2019, Twitter had more than 330 million monthly active users, with more than 500 million total number of tweets per day [7]. Having strong support from the underlying hardware to accelerate

software algorithms has become an indispensable requirement in many applications due to the widespread usage of AI on huge data-sets. FPGAs, GPUs, and emerging novel hardware devices, such as neural network accelerators, have started to play a shining role in AI execution. Graph applications are among the important algorithms used in AI and ML implementations since iterative model training is one of the greatest challenges. Concurrent nature of graph models -as vertices/edges represent concurrent entities/links- provides large parallelism potentials. However, efficiently implementing these applications on existing systems is not trivial. Therefore, having an efficient co-processor-based graph processing is considered as a promising solution for the growing need for graph applications within the AI domain. Graph processing acceleration on FPGA is discussed in Chapter 3.

Today Static-RAM (SRAM) memory is the building block for the most critical processor structures such as cache and register file. Different levels of cache altogether occupy a large portion of die area in modern processors, where decreasing area and power consumption is highly desirable. However, scaling leads to stability problems more often in cache cells than in other parts of the processor [8]. In the case of Register-File (RF) story is quite different as registers are the fastest memory component with the smallest capacity in the memory hierarchy. Because of performance requirements, RF cells are larger, faster, and more reliable than those found in the cache. But, because of the higher access rate, chances of having transient errors and subsequent fault propagation are high [9]. In caches, ECC is an effective technique. However, unlike cache, due to timing and power overheads, ECC is not an appropriate solution for register file reliability. In RF, the activity rate per address is higher than cache memories, making power consumption more important. Additionally, RF is in the processor's critical path and priority of performance is an essential necessity. Consequently, finding a suitable technique for RF reliability enhancement is a new kind of challenge when compared to cache memories. Latch and Flip-Flop (FF) are other essential digital storage elements that are used for sequential logic implementation. Although their access circuitry is somewhat different than SRAMs, nevertheless, the main data retention core is identical. All are made by two cross-coupled inverters to form static memory

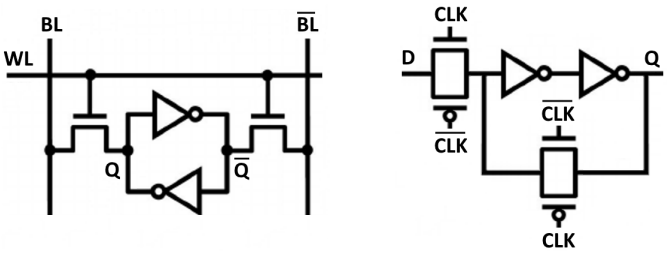

Figure 1.1: Conventional static memory bit made by two cross-coupled inverters in: standard single-port 6T SRAM cell(left), and D-latch(right).

bit as shown in Figure 1.1. In Chapter 4, we aim at enhancing the reliability of static memories, to be used in a wide set of structures.

1.2 Contributions

Our solutions target 1) safety and reliability at the system-level using redundant processors, 2) performance at the architecture-level using multiple accelerators, and 3) reliability at the circuit-level through the use of redundant transistors. Emphasis on improving the performance of digital circuits to meet the growing needs, at the same time, requires attention to reducing power consumption. Because the improvement in performance is usually achieved by increasing the frequency or increasing the area, such as the number of parallel cores. In such cases, the increase in power consumption is inevitable which can draw a decisive limit. Due to the quadratic relationship between voltage and power, voltage scaling is one of the most effective ways to reduce power. However, as discussed in Chapter 4, at lower voltage levels, the stability of circuits is adversely affected, leading to a higher number of fault rates. Therefore, techniques to increase the reliability of circuits, particularly, at low voltage levels, can provide better opportunities to increase the performance as well. The contributions in this thesis are proposed in different design levels. This provides the possibility of applying all of them in a hierarchical way on the same system, to improve the safety, reliability, and performance at the same time. The circuit-level reliable SRAM cell (Chapter 4), which is applicable to FPGAs to improve the reliability, can be

employed in a graph accelerator on FPGA (Chapter 3). In this way, the graph accelerator can be used in critical applications. Similarly, the graph processing platform (Chapter 3), can be used as the processing element in safety-critical processor (Chapter 2). In this way, the graph processing platform can be used in safety-critical applications, such as avionics.

Resource replication is the key idea in all of the approaches presented in this thesis. More specifically, it has been shown how resource replication can provide higher levels of reliability, safety, and performance in computer systems. In the safety-critical processor (Chapter 2), multiple identical processors redundantly run in parallel. However, these processors run the same software to provide multiple order of magnitude increase in the safety and reliability levels, based on redundancy in processing. In graph accelerator for FPGA (Chapter 3), multiple identical accelerators run in parallel similarly. However, they divide and run the same task to enhance the processing power up to a saturation point. In reliable SRAM cell (Chapter 4), identical copies of a memory bit are preserved to build a fault-tolerant structure, using data redundancy.

Our redundancy-based contributions in this thesis, regarding the design of customized hardware for enhancing safety, reliability, and performance of computer systems, are listed more specifically in the following order: 1) system-level, 2) architecture-level, and 3) circuit-level.

- **System-level:** A sensitivity analysis of functional system safety to some critical design parameters in a system-level replication-based design of a safety-critical computer is introduced. Analysis is given for two fundamental and widely used multi-channel safe configurations, 1oo2 and 2oo3, where a basic 1oo1 system is used as a baseline. All configurations have been modeled by Markov chains to examine at which safety integrity level (SIL) they stand, and how distant they are from the target level. In this way, each safety parameter's contribution to safety can be understood. Through these measurements, instead of blindly improving an unsafe system, designers can make an informed decision to select the most appropriate parameter for improvement. A significant presence of Common Cause Failures (CCFs), is

an important factor for analyses in this study.

- **Architecture-level:** A high-performance/work-efficient, SystemC based, synchronous, deeply pipelined, and fully synthesizable graph processing architecture, ready to be implemented on FPGA is proposed. The template-based framework is usable by non-hardware experts, such as IT engineers, to generate FPGA programs automatically, using only C/C++. The proposed framework is specifically tested on, and prepared for the Intel state-of-the-art Xeon+FPGA platform. However, it can be extended and adapted to other similar FPGA-based architectures. In work-efficient mode, a novel fast *bit-vector* to keep an active vertex list is introduced. The alternative HLS language option for FPGA, OpenCL, is investigated and contrasted to show limitations and difficulties of implementing high-performance pipelined graph algorithms by high-level languages.

- **Circuit-level:** A novel static memory cell is implemented to mitigate transient faults. In the case of SRAM memory, we improve both the soft-error rate and the V_{dd}-induced fault rate at the same time. In the case of latch and Flip-Flop, the soft-error rate is improved. This approach brings reconfigurability of reliability to static memory structures (SRAM, latch, and Flip-Flop) with fine granularity, where it is particularly useful for reconfigurable fabrics like commercial-grade FPGAs to make them more suitable for critical applications. Additionally, the reliable mode can provide auto-correction and full immunity against single faults. Moreover, it efficiently can cope with Multiple Bit Upsets (MBUs). Unlike prevalent hardware redundancy techniques, like Triple Modular Redundancy (TMR), there is no explicit majority voter at the output, in this technique. Consequently, voter failure and latency are of less concern. Moreover, the adjacent cell immediately recovers the faulty bit value to avoid degradation in redundancy. Based on a similar idea and as a typical application, we propose a new approach to exploit vacant spaces in a Register-File (RF) to keep redundant data for better robustness.

1.3 Outline

The remainder of this dissertation is organized as follows. In the next chapter, analysis of design parameters in safety-critical computers is presented. The motivation for having sensitivity analysis, a list of considered design parameters, Markov modelings, and sensitivity simulation results are discussed in this chapter. In Chapter 3, the architecture of the HLS-based high-throughput and work-efficient synthesizable graph processing template framework is proposed. Background of graph processing, Intel Xeon+FPGA platform, and HARP program are presented in this chapter. Furthermore, different high level synthesis (HLS) language options for FPGAs (OpenCL and SystemC), proposed architectures for high-throughput and work-efficient modes, and evaluation results are also given here. In Chapter 4, the joining technique to build reliable and reconfigurable static memory elements is proposed. Common fault sources in static memories (such as voltage scaling, process variation, noise, and radiation), the structure of joining technique, improvements, and evaluation results are presented in this chapter. Finally, the thesis is concluded in Chapter 5.

Chapter 2

Analysis of Design Parameters in Safety-Critical Computers

2.1 Introduction

Redundancy, meaning having multiple processors (channels) doing the same task, is required by safety standards in many cases. The ratio of Common Cause Failures (CCFs), defined as the ratio of concurrent failure rate (simultaneous failures among redundant channels) over total failure rate, has great impact on system safety. The percentage of failures the system is able to detect by means of fault detection techniques also has a direct effect on safety. This is because undetected failures are potential dangers. To remove such undetected failures, an important factor is the frequency and the quality of system maintenance. Frequency and comprehension of the system tests (automatically or by technicians) to repair or replace the impaired components, can guarantee the required safety level by removing transient failures or refreshing worn-out parts.

As safety is a very wide subject, the main objective of this work is to investigate the sensitivity of system safety, affected by random hardware failures, to

See [10, 11] for the original work.

some crucial design parameters [10, 11]. Three widespread configurations; 1oo1, 1oo2, and 2oo3; with known values of parameters, are assumed as base systems. For these systems, we evaluated individual parameters that contribute to safety. In this work, we target high demand/continuous systems, where the frequency of demand to run the safety function is more than one per year, unlike low demand systems where it is less than one per year [2]. Average frequency of a dangerous failure of safety function per hour (PFH), is the safety measure for high demand/continuous systems, while probability of failure on demand (PFD), is the measure for low demand systems. PFH is defined as average rate of entering into unsafe state, while PFD is defined as probability of being in unsafe state.

This chapter is organized as follows: In Section 2.2, some of the recent or relevant works are reviewed and our motivation is given in more detail. In Section 2.3, definition and modeling for considered design parameters are described. In Section 2.4, base systems and their Markov modeling are proposed. In Section 2.5, experimental results are discussed, while Section 2.6 discusses a simplified Markov modeling in safety calculations. Finally, a summary is given in Section 2.7.

2.2 Related Works and Motivation

During the design process, concentrating on multiple aspects of design altogether for the purpose of improvement can be complicated. Normally, if the prototype design does not meet the requirements, it is rational to find the system's bottleneck and focus on it. In safety-related designs, by knowing the share that each parameter provides to safety, the designer can decide where to put more emphasis to improve the outcome with the least amount of effort. Here, we discuss some of the safety-critical computer system designs in literature, which considered a subset of safety design parameters due to complexity.

In [12], authors designed a redundant computer system for critical aircraft control applications, and an acceptable level of fault tolerance is claimed to be

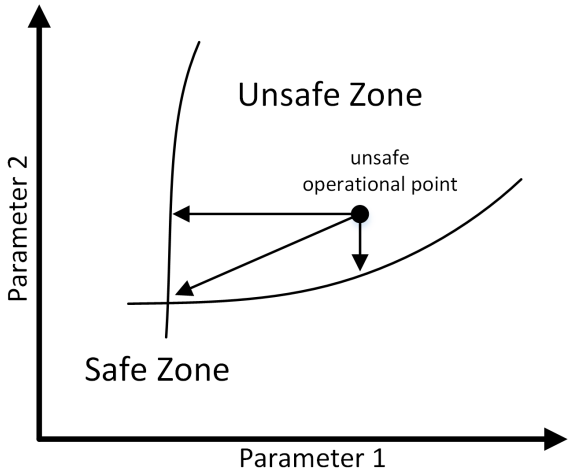

Figure 2.1: Safety goal should be achieved by the most economical improvement.

achieved with using five redundant standard processors, extensive error detection software and fault isolation mechanism. In [13], dual-duplex and Triple Modular Redundancy (TMR) synchronous (with common clock signal) computer systems have been built using military and commercial electronic parts. While authors tried to improve the system safety, the effect of CCFs is not assessed, where this effect can be significant in synchronous systems. Besides, the achieved safety level is not compared to any standard level. In microcontroller-based SIL4 software voter [14], SIL4 level is claimed to be obtained with a duplex architecture. Nevertheless, neither failure coverage, nor CCFs are assessed in sufficient details. Similarly, in [15], authors target safe computer system for a train, which is not compared to standard levels, and does not consider CCFs or diagnostic coverage.

These approaches, either lack in considering some of the most influential safety design parameters or methodology to assess the system safety level with respect to standards. Thus, these studies are incomplete to be considered for real safety-critical applications due to complexity of taking all parameters into account. This stimulated us to have an analysis on a few safety architectures also used in above studies. By showing the sensitivity of safety to each such parameter, we aim to provide a comparative understanding of these occasionally ignored parameters. This can help practitioners to select the most appropriate parameter for improving the safety. Depending on the constraints, the most appropriate parameter can be

translated to the one that leads to cheapest, fastest, or easiest system modification (as shown in Figure 2.1).

In [16], authors model a safety-related system in low demand mode using Markov chain to calculate PFD measure, in a way that is explained in the respective standard [3]. Several parameters such as CCF, imperfect proof testing, etc. are integrated into the model to investigate their influence over safety. However, in our work, we focus on PFH, where its calculation is not as straightforward as PFD. Moreover, we include additional parameters such as frequency of online testing, self-testing, etc., with sensitivity analysis for each parameter.

There have been many efforts related to generalized and simplified PFH formula for M-out-of-N (MooN) architectures. The works proposed in [17, 18] develop a set of analytical expressions with some assumptions and parameters different than ours, like considering partial proof test, slightly taking the CCF contributions into account or dealing with dangerous detected failures differently. In [19], probabilistic analysis of safety for MooN architectures is proposed when considering different degrees of uncertainty in some safety parameters such as failure rate, CCFs, and diagnostic coverage, by combining Monte Carlo sampling and fuzzy sets. Emphasizing the significance of CCF impact over safety in redundant systems, in [20], authors explore the criticality of beta-factor on safety calculations. Specifically, they address PFD measure for a typical 1oo2 system. Influence of diversity in redundancy (i.e. implementing redundancy with components technologically diverse) over CCF is assessed in [21] by a design optimization approach for low demand systems.

2.3 Safety Parameters In Our Analysis

In this section, we review the definition and modeling of design parameters that effect safety. It is assumed that the safety-critical computer system is composed of multiple redundant processors since such replication is recommended by safety standards.

2.3.1 Processor Failure Rate

A safety-critical computer system is composed of one or more redundant processors, connected to each other by communication links. We may also call them as channels or programmable electronics (PEs) according to the safety standards terminology. Generally, there is no extraordinary requirement regarding reliability of PEs. Due to low quantity and high cost of these systems, components are not necessarily designed for reliability purposes. Most often, a PE is a standard processor module, built from available Commercial-Off-The-Shelf (COTS) electronic parts including processor, memory, power circuitry, etc. In this study, we take a PE as a black box, assuming it comes with a single overall failure rate, λ_{PE}.

2.3.2 Common Cause Failure (CCF)

According to IEC 61508-4 standard [2], Common Cause Failure (CCF, or dependent failure) is defined as concurrent occurrence of hardware failures in multiple channels (PEs) caused by one or more events, leading to system failure. If it does not lead to a system failure, it is then called common cause fault. The β factor represents the fraction of system failures that is due to CCF. Typically, for a duplicated system, β value is around a few percent, normally less than 20%. In safety standards, two β values are defined for detected and undetected failures (β_D and β), while here we only assume a single β value for both. Assuming that the CCF ratio between two PEs is taken as β, by using the extended modeling and notations shown in [1], we make the following observations for all systems in this work:

- 1oo1 configuration: Since there is no redundant PE, $\beta_{1oo1} = 0$.
- 1oo2 configuration: As depicted in Figure 2.2, the β of system is only related to CCFs between two PEs. Therefore, $\beta_{1oo2} = \beta$.
- 2oo3 configuration: As depicted in Figure 2.2, the overall β of 2oo3 system

is related to mutual CCFs, plus CCFs shared among all three PEs. Note that by definition, the CCF ratio between every two PEs is taken as β. The β_2 is defined as a number in $[0, 1]$ range, expressing part of β which is shared among all three PEs [1]. For a typical 2oo3 system, we assume $\beta_2 = 0.3$ of β, making $\beta_{2oo3} = 2.4\beta$ (see Figure 2.2).

The two parameters, β and β_2, indicators of mutual and trilateral PEs isolation, are evaluated in our analysis.

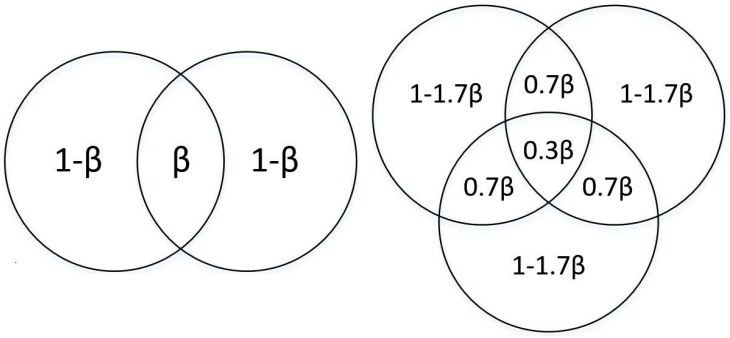

Figure 2.2: β models for duplicated and triplicated systems [1]. Here, $\beta_2 = 0.3$ of β, while β is split into $0.3\beta + 0.7\beta$.

2.3.3 Failure Diagnostic Coverage

According to IEC 61508-4 [2], Diagnostic Coverage (C or DC) is defined as the fraction of failures detected by automatic online testing. Generally two complementary techniques are employed to detect failures, self-testing and comparison. Self-testing routines run upon each PE to diagnose occasional failures autonomously and they usually detect absolute majority of failures, normally around 90%. Second diagnostic technique is data comparison among redundant PEs for detecting the rest of the undetected failures. Hence, generally we can express C as:

$$C = C_{selftest} + C_{compare} \approx 1$$

As formulated in [22], we use the following expressions to describe the system's

C rate. According to the referred formulation, the total C is expressed as:

$$C = C_{selftest} + (1 - C_{selftest}) \cdot k$$

More specifically, k is the efficiency of comparison test. Since the comparison method is more effective against independent failures (non-CCFs), it is reasonable to differentiate between C rate of CCF and independent failures. Therefore, two variants of former expression can be derived:

$$C^i = C_{selftest} + (1 - C_{selftest}) \cdot k^i$$

$$C^c = C_{selftest} + (1 - C_{selftest}) \cdot k^c$$

Here k^i and k^c are two constants, $0 \leqslant k^i, k^c \leqslant 1$, describing the efficiency of comparison for either of two classes of failures. Since comparison is less effective against CCFs, the k^c value is low, generally less than 0.4, while k^i can be close to 1 [22]. Therefore, normally $C^c \leqslant C^i$. Three representative parameters; $C_{selftest}$, k^c and k^i; are used in our analysis.

2.3.4 Test and Repair

Based on the IEC 61508 standard, two forms of test and repair have to be accessible for safety systems: online test and proof test. In online test (or automatic test), diagnostic routines run on each PE periodically, while system is available. As soon as a failure is detected, the faulty PE (or in some configurations the whole system) is supervised to go into fail-safe mode to avoid dangerous output. Thereafter, system tries to resolve the failure with immediate call for personnel intervention or a self commanded restart without human intervention. For transient failures, a system restart can be a fast solution, while for persistent failures switching to a spare PE or system, provides a faster recovery. In any case, online repairing is supposed to last from a few minutes to a few days. Repair rate is denoted by μ_{OT} which is defined as $1/MRT_{OT}$ (MRT: mean repair time). The t_D parameter is the time to detect a failure in online testing. There is no direct reference to this parameter in the standard, probably because it is assumed to be negligible with respect to repair time. However, it has been considered in

literature [23]. $MTTR_{OT}$ (mean time to restoration) is mean total time to detect and repair a failure (see Figure 2.3). Some systems may support partial recovery which means repairing a faulty PE, while the whole system is operational. Restarting only the faulty PE (triggered by operational PEs), can make it operational again. However, if the fault is persistent, such recovery is not guaranteed. Three parameters t_D, μ_{OT} and availability of partial recovery are also considered in our analysis.

Proof test (or offline test or functional test) is the second and less frequent form of testing, whereby the periodic system maintenance process is performed by technicians. During such a maintenance, system is turned off and deeply examined to discover any undetected failure (not detected by online diagnostics) followed by a repair or replacement of defective parts. Test Interval (TI) defines the time interval at which this thorough system checking is performed and is typically from a few weeks to a few years. In such a scenario, repair time is negligible relatively. $MTTR_{PT}$ is the mean time to restoration (detect and repair as shown in Figure 2.3) from an undetected failure, and on average is taken as $TI/2$ [3]. Proof test and repair rate is denoted by $\mu_{PT} = 1/MTTR_{PT}$. The μ_{PT} is another parameter considered in our analysis.

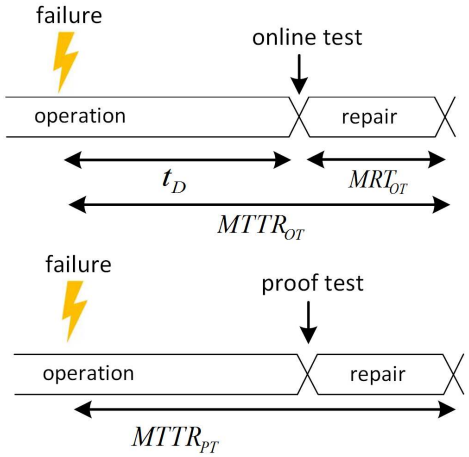

Figure 2.3: Illustration of test-repair abbreviations in safety standards [2, 3]. Top: online test, bottom: proof test.

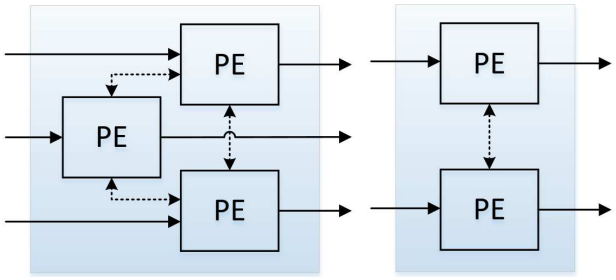

Figure 2.4: High level view of triplicated and duplicated systems.

2.4 Base Systems

In this section, we define two prevalent safety configurations, 1oo2 and 2oo3 plus simple 1oo1 as a reference, for our analyses. Configurations are modeled by Markov chain with employing all the aforementioned parameters. The assigned set of default values for parameters specify the initial safety point for each system.

2.4.1 Assumptions

In this study, we make the following assumptions: Typically, a safe computer is responsible for running user computations. At the same time, it is in charge of checking the results for possible failures and taking necessary measures (in other words, running safety function). Specifically, the safety function of the system detects and prevents any erroneous calculation result on PEs. All PEs are asynchronous and identical (homogeneous) and connected to each other by in-system links, whereby software voting and comparison mechanisms operate (Figure 2.4). In this work, our focus is on processors (PEs), while I/O ports and communication links are assumed to be black-channel, by which safety is not affected. This assumption can be realized by obeying standards applied for safe communication over unsafe mediums (e.g. EN-50159). These systems are assumed to be single-board computers (SBCs), meaning all redundant PEs reside on one board. Online repair of a PE with detected failures makes it operational again, but a PE with undetected failures can be repaired only by proof test and repair. Moreover, for biasing a high demand/continuous system toward safety,

degraded operation is not allowed. It means when a failure is detected, the faulty PE activates its fail-safe output and contributes to voting (alternatives are 1)to report the failure but keep the PE's output silent, leading to degradation of system, for example from 1oo2 to 1oo1, or 2)not to tolerate any faulty PE [17]). A PE with undetected failure is assumed to be seemingly operational and able to run diagnostic routines. Another simplifying assumption is that a CCF occurs in a symmetric way across PEs in a way that it is detectable on all PEs or on none of them.

2.4.2 Default Parameters

For safety parameters which we intend to investigate in this study, we assign a set of default values to define an initial safety point for each system (shown in Table 2.1). In our experiments, we sweep each parameter around the default value and illustrate how safety is affected. This way, the sensitivity of system safety with respect to that parameter will be revealed. Although in implementing a system some parameters such as β and β_2, or k^c and k^i, may not be practically independent, we disregard this dependency which is due to implementation.

Parameter	Meaning	Default value
λ_{PE}	PE failure rate	1.0E-5/hour
$C_{selftest}$	Diagnostic coverage of self-testing [22]	0.90
k^i	Comparison efficiency for independent failures [22]	0.90
k^c	Comparison efficiency for CCFs [22]	0.40
β	CCF ratio between each two PEs	0.02
β_2	Part of β shared among three PEs [1]	0.3
t_D	Time to detect failure by online test (inverse of online test rate)	0 (negligible)
μ_{OT}	Online repair rate	1/hour
μ_{PT}	Proof test and repair rate	0.0001/hour ($\approx$ 1/year)

Table 2.1: Default values for safety design parameters.

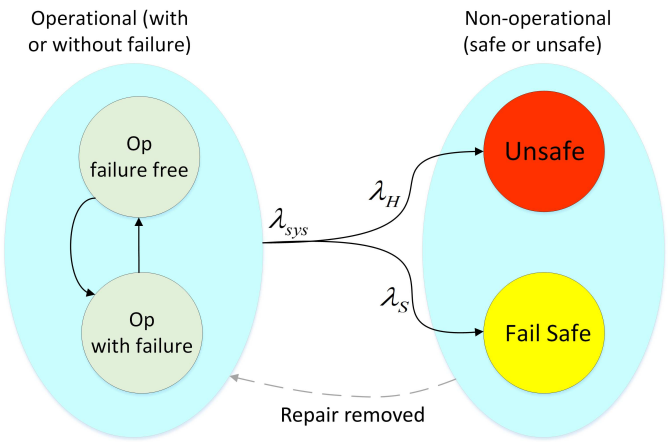

Figure 2.5: Unsafe (hazardous) and safe failure rate.

2.4.3 Markov Models

In this section, we give our safety configurations modeled by Markov chain. RAMS (Reliability, Availability, Maintainability and Safety) measures are calculated according to guidelines suggested in ISA-TR84.00.02 [3] and IEC 61165 [24] standards, along with previous studies [25]. States are divided into two main categories; 'up' (or operational) and 'down' (or non-operational). In up states, the system is able to correctly run the safety functions. Up state is either all-OK initial state or any state with some tolerable failures. Down states are those in which system is not able to correctly run safety functions, either intentionally as in the fail-safe state or unintentionally as in unsafe (hazardous) state. System moves into fail-safe/unsafe state if there are intolerable number of dangerous detected/undetected failures present.

As soon as a dangerous failure is detected, system may either tolerate it (like the first detected failure in 2oo3 system) or enter into fail-safe state. On the other hand, if the failure is left undetected, system may inadvertently tolerate it (like first undetected failure in 1oo2 or 2oo3 systems) or enter into unsafe (hazardous) state.

PFH is defined as the average rate of entering into an unsafe state. For safety calculation purposes, repair transitions from unsafe states toward up states should

not be considered [24]. Additionally, as in the case of reliability calculation, we also remove repairs from down fail-safe states to account only the effective safety when system is operational. Note that the repairs inside up states are not removed. All of the following Markov models are in the full form before repair removal. From the total failure rate of each system, λ_{sys}, only the hazardous part, λ_H, should be considered (as shown in Figure 2.5). By definition, PFH is calculated as the average of λ_H. The details of the following formula, required for decomposing λ_{sys} into λ_H and λ_S, is explained in the literature [26](P_H: Probability of being in hazardous state, P_S: Probability of being in fail-safe state, $P_{HS} = P_H + P_S$, P'_{HS}: derivative of P_{HS} with respect to time):

$$\lambda_H = \frac{P'_{HS}}{1 - P_{HS}} \cdot \frac{P_H}{P_{HS}}$$

Generally, probabilities of the system over time is described with the following set of differential equations:

$$\boldsymbol{P}'_{1\times n} = \boldsymbol{P}_{1\times n} \cdot \boldsymbol{A}_{n\times n}$$

where $\boldsymbol{P}$ is vector of state probabilities over time, $\boldsymbol{P}'$ is derivative of $\boldsymbol{P}$ with respect to time, n is number of states, and A is transition rate matrix. Abbreviations used in Markov chains are listed in Table 2.2.

1oo1 Configuration: The single system is composed of a single PE without any redundancy. Hence, all failures are independent (non-CCF) and can only be detected by self-testing. If failure is detected, next state is fail-safe, otherwise it is unsafe (Figure 2.6). Transition terms for 1oo1 system are ($k^i = 0$, $C^i = C_{selftest}$):

$$\lambda_{DD}s = C^i \cdot \lambda_{PE}$$
$$\lambda_{DU}s = (1 - C^i) \cdot \lambda_{PE}$$
$$\lambda_D s = \lambda_{DD}s + \lambda_{DU}s = \lambda_{PE}$$

Transition rate matrix for illustrated Markov is as follows:

D	Dangerous failure, a failure which has potential to put system at risk (only such failures are considered in this work).
C	Diagnostic coverage factor is the fraction of failures detected by online testing.
DD	Dangerous detected failure, a dangerous failure detected by online testing.
DU	Dangerous undetected failure, a dangerous failure not found by online testing.
CCF	Common cause failure (dependent failure).
λ	Total failure rate of component or system.
λ^i	Independent failure rate of component or system.
λ^c	CCF failure rate of component or system.
λ_{DD}	DD failure rate of component or system.
λ_{DU}	DU failure rate of component or system.
s, d, t	Number of redundancies (single, dual or triple).

Table 2.2: Other abbreviations and symbols.

$$A = \begin{bmatrix} -\lambda_D s & \lambda_{DU} s & \lambda_{DD} s \\ \mu_{PT} & -\mu_{PT} & 0 \\ \mu_{OT} & 0 & -\mu_{OT} \end{bmatrix}$$

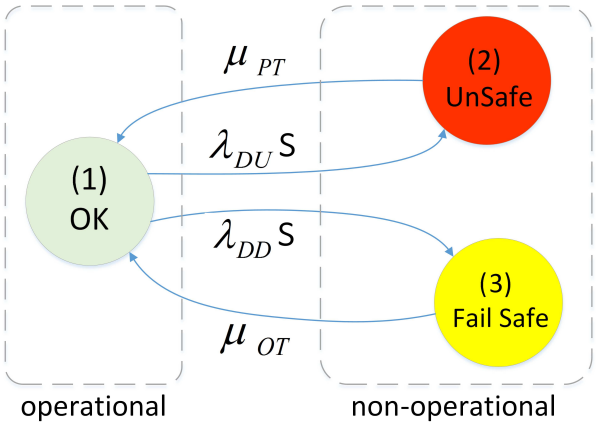

Figure 2.6: Markov model for 1oo1 (single) system.

1oo2 Configuration: According to IEC 61508-6, 1oo2 system consists of two parallel channels which can both run the safety functions. However, only one dangerous-failure free channel is sufficient to keep the system safe. In this configuration, a single DU is tolerable which means that hardware fault tolerance (HFT) is equal to one. Since it is assumed that DD failures contribute to voting, then no DD failure is tolerated and system immediately enters into fail-safe mode (Figure 2.7). Generally, 1oo2 system has high safety against DU failures and low reliability against safe failures. Note that in Figure 2.7, since the failure in state (2) is undetected or hidden, the system seemingly works with two operational channels. Therefore, the system has similar behavior against DD failures in both state (1) and state (2). However, in fact the faulty channel is not counted as operational. Because state (2) is one step closer to unsafe state than state (1). Transition terms used for 1oo2 system are:

$$\lambda^i_{DU}d = 2[(1 - C^i) \cdot (1 - \beta_{1oo2})]\lambda_{PE}$$
$$\lambda^c_{DU}d = [(1 - C^c) \cdot \beta_{1oo2}]\lambda_{PE}$$
$$\lambda_{DD}d = \lambda^i_{DD}d + \lambda^c_{DD}d =$$
$$(2 \cdot C^i \cdot (1 - \beta_{1oo2}) + C^c \cdot \beta_{1oo2})\lambda_{PE}$$
$$\lambda_D d = \lambda_{DD}d + \lambda_{DU}d =$$
$$\lambda^i_{DD}d + \lambda^c_{DD}d + \lambda^i_{DU}d + \lambda^c_{DU}d$$

Transition rate matrix for illustrated Markov is as follows:

$$A = \begin{bmatrix} -\lambda_D d & \lambda^i_{DU} d & \lambda_{DD} d & 0 & \lambda^c_{DU} d \\ \mu_{PT} & -(\lambda_{DD} d + \mu_{PT} + \lambda_{DU} d/2) & 0 & \lambda_{DD} d & \lambda_{DU} d/2 \\ \mu_{OT} & 0 & -\mu_{OT} & 0 & 0 \\ \mu_{PT} & \mu_{OT} & 0 & -(\mu_{OT} + \mu_{PT}) & 0 \\ \mu_{PT} & 0 & 0 & 0 & -\mu_{PT} \end{bmatrix}$$

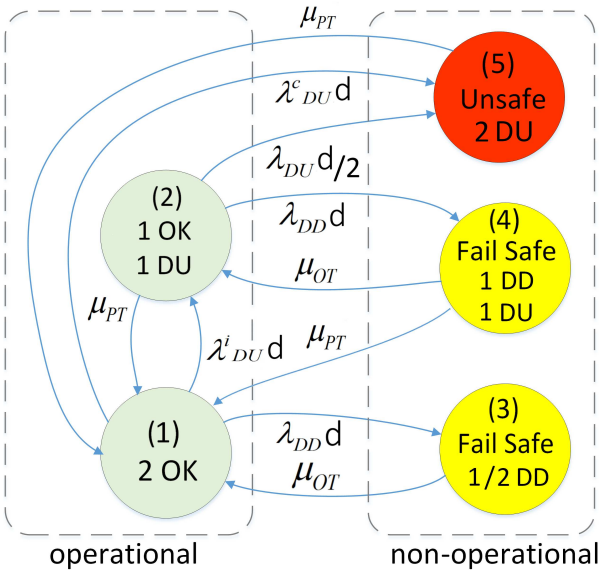

Figure 2.7: Markov model for 1oo2 system.

2oo3 Configuration: Similar to 1oo2, 2oo3 is also capable of tolerating one DU failure, meaning hardware fault tolerance (HFT) is equal to one. Besides, it has higher reliability (continuity of operation) due to being able to tolerate a single DD failure, similar to 2oo2 system (note that 2oo2 is not discussed here). Therefore, in literature, 2oo3 is known to have benefits of both 1oo2 and 2oo2 at the same time (as shown in Figure 2.8). However, due to higher number of vulnerable channels (since total failure rate of all channels increases as the number of channels increases), 2oo3 is neither as safe as 1oo2, nor as reliable as 2oo2. Note that, in such a system, we assume online repairing does not remove undetected failures. In Figure 2.8, the μ^*_{OT} transitions represent partial recovery (explained in Section 2.3.4). Transition terms for 2oo3 system are ($\beta_2 = 0.3$):

$$\lambda^i_{DU}t = 3 \cdot [(1 - C^i) \cdot (1 - 1.7\beta)]\lambda_{PE}$$
$$\lambda^c_{DU}t = [(1 - C^c) \cdot 2.4\beta]\lambda_{PE}$$
$$\lambda_{DD}t = \lambda^i_{DD}t + \lambda^c_{DD}t =$$
$$[3C^i \cdot (1 - 1.7\beta) + C^c \cdot 2.4\beta]\lambda_{PE}$$
$$\lambda_D t = \lambda_{DD}t + \lambda_{DU}t = \lambda^i_{DD}t + \lambda^c_{DD}t + \lambda^i_{DU}t + \lambda^c_{DU}t$$

Transition rate matrix for illustrated Markov is as follows:

$A =$

$$\begin{bmatrix}
-\lambda_D t & \lambda^i_{DU}t & \lambda^i_{DD}t & 0 & 0 & \lambda^c_{DD}t & \lambda^c_{DU}t \\
\mu_{PT} & -\mu_{PT}-\lambda_{DD}t \\ & -2/3\lambda_{DU}t & 0 & \lambda^i_{DD}t & \lambda^c_{DD}t & 0 & 2/3\lambda_{DU}t \\
\mu^*_{OT} & 0 & -\lambda_D d \\ & & -\mu^*_{OT} & \lambda^i_{DU}d & 0 & \lambda_{DD}d & \lambda^c_{DU}d \\
\mu_{PT} & \mu^*_{OT} & 0 & -\mu_{PT}-\lambda_{DD}d \\ & & & -\mu^*_{OT}-\lambda_{DU}d/2 & \lambda_{DD}d & 0 & \lambda_{DU}d/2 \\
\mu_{PT} & \mu_{OT} & 0 & 0 & -(\mu_{OT} \\ & & & & +\mu_{PT}) & 0 & 0 \\
\mu_{OT} & 0 & 0 & 0 & 0 & -\mu_{OT} & 0 \\
\mu_{PT} & 0 & 0 & 0 & 0 & 0 & -\mu_{PT}
\end{bmatrix}$$

2.5 Experimental Results and Discussion

In this section, we investigate the influence of aforementioned parameters over PFH measure through solving Markov models of four configurations: 1oo1, 1oo2, and 2oo3 without/with partial recovery (2oo3 and 2oo3-PR). First, we give the initial state of these configurations corresponding to default parameter values for other RAMS measures - reliability and availability.

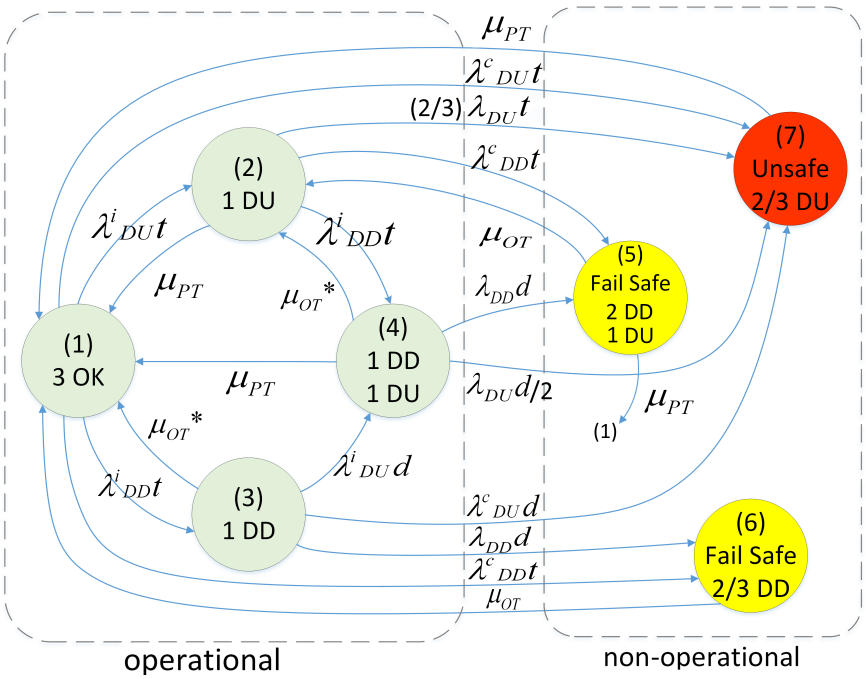

Figure 2.8: Markov model for 2oo3 system.

2.5.1 Reliability and Availability

Reliability function, which is defined as probability of continuously staying operational, is depicted in Figure 2.9 (refer to Table 2.1 for fixed parameter values). Despite high safety level, the 1oo2 suffers from high rate of false-trips (transitions into fail-safe state), even more than the simple 1oo1. This follows from the fact that total failure rate of 1oo2 is around 2λ, and any single DD failure brings whole system into fail-safe state. This is the cost paid for having high safety with a simple architecture. Moreover, note that if partial recovery is not provided, more complex 2oo3 system is not much better than the others. After first DD failure, 2oo3 degrades to 1oo2 where reliability drops sharply even less than 1oo1. With partial recovery, the faulty PE with DD failure is quickly recovered largely reducing the probability of having two consecutive DD failures. Superiority of systems for operational availability at time=∞ (steady state availability), at very low β value (which is not practically achievable), is as expected (see Figure 2.10, refer to Table 2.1 for fixed parameter values). However, as CCF rate increases, their order is swapped. This is due to the fact that staying more in operational

states means higher probability of being exposed to DU CCFs and having a direct jump into unsafe state which takes considerable time to be recovered from. Nevertheless, availability values are almost same, except 1oo1 which is by far the lowest (1oo1 is not shown).

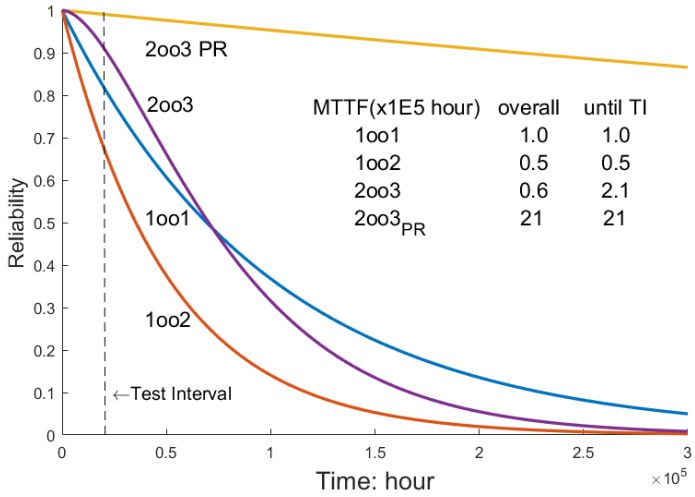

Figure 2.9: Reliability functions for base systems.

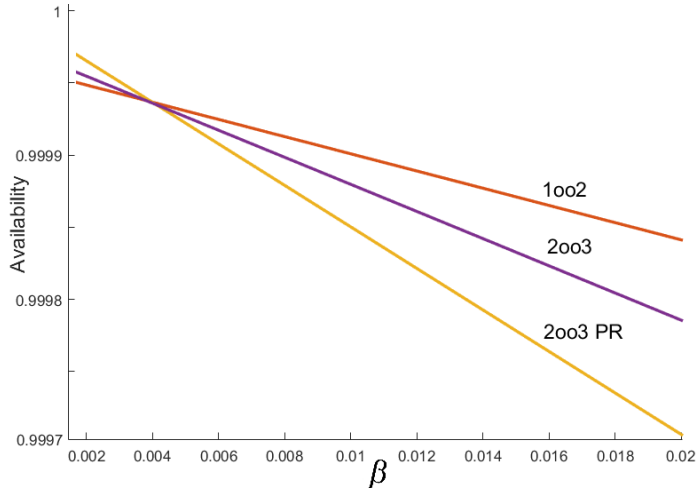

Figure 2.10: Steady state operational availability value for base systems with β variation (1oo1 is not shown).

2.5.2 Safety Sensitivity Analysis

In this section, we show the effect of variation in each of aforementioned parameters around defined default value, over PFH value. Mathematically, the following experiments show the partial derivations, $\partial PFH/\partial p$, where p is one of the safety parameters. SIL1-SIL4 safety levels are plotted by horizontal lines to show relative safety position. By such illustration of safety, designer perceives the distance of current design state from desired safety level. Besides, we also show a few pairs of relevant parameters in 2D-space. At initial states of configurations specified by default parameters, 1oo1 marginally could not achieve SIL2, while the rest are in SIL3 region. As explained before, generally 1oo2 is the safer configuration, while 2oo3 has higher reliability.

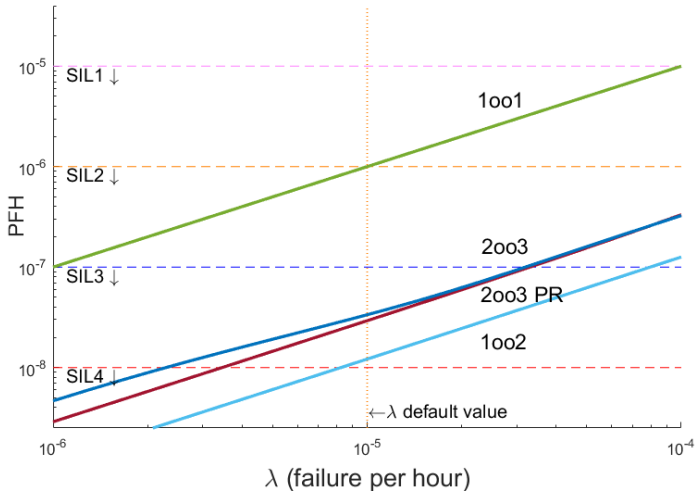

Figure 2.11: Effect of λ_{PE} variation over safety.

Sensitivity to λ_{PE}:

Figure 2.11 shows how safety is affected by different λ_{PE} values. Plots are almost linear in logy-logx plane with slope equal to one, which describe linear functions in y-x plane passing through the origin. In other words, $PFH = K \cdot \lambda_{PE}$. This is also understandable from Markov models, where most of the transition rates are linear function of λ_{PE}. Linearity implies that by just knowing the line slope which is achievable by having a single (λ_{PE}, PFH) point, and without solving the complicated Markov models or other techniques every time, safety of system

can be tuned. For example, an order of magnitude (10X) improvement in λ_{PE}, results in shifting up one safety level (e.g. from SIL3 to SIL4).

Sensitivity to β and β_2:
β and β_2 are the indicators of mutual and trilateral isolation among PEs. It is a well-known fact that CCF failures have strong adverse effect on safety-critical systems. Figure 2.12 depicts how systems' safety is affected by β variation (refer to Table 2.1 for fixed parameter values). 1oo1 is independent from β as expected. One can observe from this figure that, for default parameters, it is quite difficult to get SIL4 through β improvement. Because by questionnaire method for β estimation (described in IEC 61508-6 [2]), β can hardly be estimated to be below 1%. A noticeable observation here is that similar to λ_{PE}, plots are almost linear in logy-logx plane. This linearity makes adjustment of safety by tuning β parameter easily, without needing to solve complicated mathematical models every time.

The β_2 is defined as a number in $[0, 1]$ range, expressing part of β which is shared among all three PEs [1], where it is typically in 0.2-0.5 range. It is clear that 1oo1 and 1oo2 are independent from β_2. For a fixed β value, increase in β_2 leads to a decrease in β_{2oo3} and an obvious improvement in PFH. Therefore, to have a more meaningful analysis, instead of β, we fix the β_{2oo3}. Figure 2.13 shows that the variation in β_2 has almost no effect over PFH (refer to Table 2.1 for fixed parameter values). The reason is that in a 2oo3 system, any mutual or trilateral CCF failure leads to the same situation, fail-safe or unsafe state. This assumption has to be reminded from Section 2.4.1 that CCFs are symmetric. In fact, this parameter affects the systems such as 1oo3 (not discussed here) in which the mutual CCFs can not defeat the redundancy, but trilateral CCFs can. One good design practice in such systems is to avoid sharing common resources among all PEs, such as communication links or power supply lines.

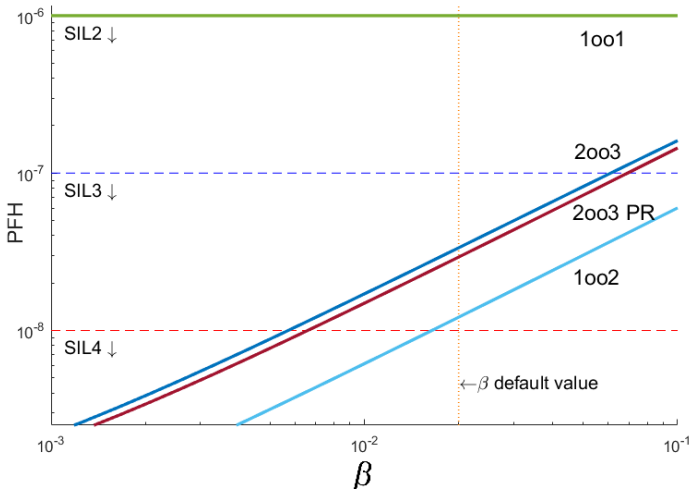

Figure 2.12: Effect of β variation over safety.

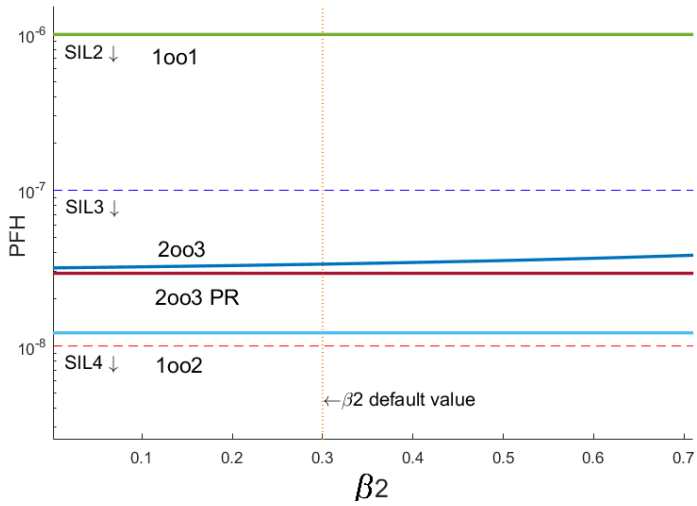

Figure 2.13: Effect of $\beta 2$ variation over safety.

Sensitivity to $C_{selftest}$:

According to formulas in section 2.3.3, self-testing is assumed to be equally effective for both CCFs and non-CCFs. As shown in Figure 2.14, variation in this parameter can significantly affect the safety (refer to Table 2.1 for fixed parameter values). For achieving SIL4 in the 1oo2 system, the $C_{selftest}$ has to be increased by 2%, while in 2oo3, it is more difficult, where at least 6% improvement is required (default of $C_{selftest} = 0.9$).

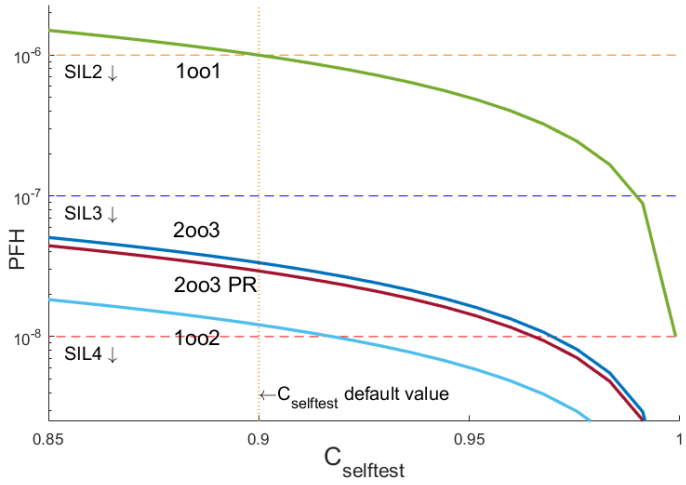

Figure 2.14: Effect of $C_{selftest}$ variation over safety.

Sensitivity to k^i:

k^i is a constant which specifies the efficiency of comparison among PEs for detecting independent failures. Comparison is expected to be more efficient against non-CCFs than CCFs ($k^i = 0$ for 1oo1). In Figure 2.15, there is an unexpected behavior as k^i has almost no sensible (or very small) influence on safety (refer to Table 2.1 for fixed parameter values). The main reason for such observation is the absolute dominance of CCFs in above systems. More precisely, any DU CCF takes the whole system into unsafe state. However, two consecutive DU independent failures have to occur to cause the same situation which is far less probable. This is translated to an order of magnitude less influence of non-CCFs over safety. As a result, these systems seem to be rather insensitive to k^i.

One possible incorrect conclusion from this observation is to give up comparison for independent failures. But the fallacy is that whether a failure is dependent or not is not distinguishable before detection. As we will see, k^c still has considerable effect on safety and as a result, comparison cannot be ignored. Since k^c is usually as low as 0.1-0.4, a relaxed comparison mechanism that leads to k^i value as low as k^c is completely acceptable. Because it is enough to just have a reasonable value for k^c.

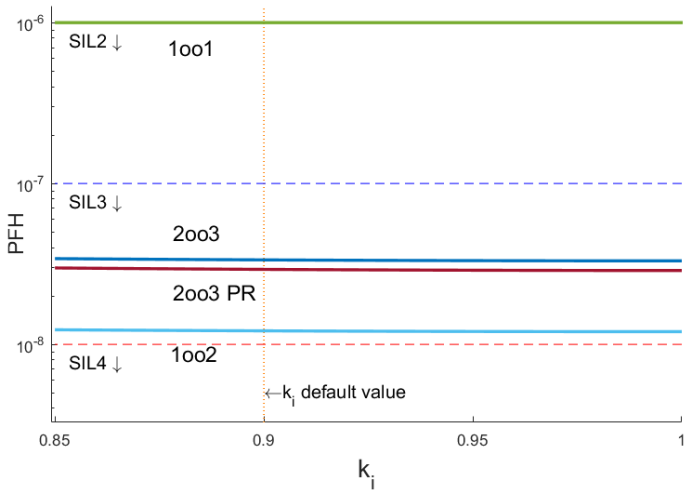

Figure 2.15: Effect of k^i variation over safety.

Sensitivity to k^c:
k^c is a constant which specifies the efficiency of comparison among PEs for detecting CCFs. In both 1oo2 and 2oo3 configurations, CCFs mostly have a larger negative influence when compared to independent failures. Because a single independent failure is tolerable in both cases. By the definition of CCF, comparison is not expected to be very efficient against CCFs ($k^c = 0$ for 1oo1). Nevertheless, experiments (as shown in Figure 2.16) show that k^c still has a considerable effect over safety.

In case when one parameter is not sufficient to achieve the required safety level, simultaneous improvements on multiple parameters can be tried. Figures 2.17 and 2.18 show SIL regions in 2-D space while target safety is possible with values on SIL4 border lines (refer to Table 2.1 for fixed parameter values).

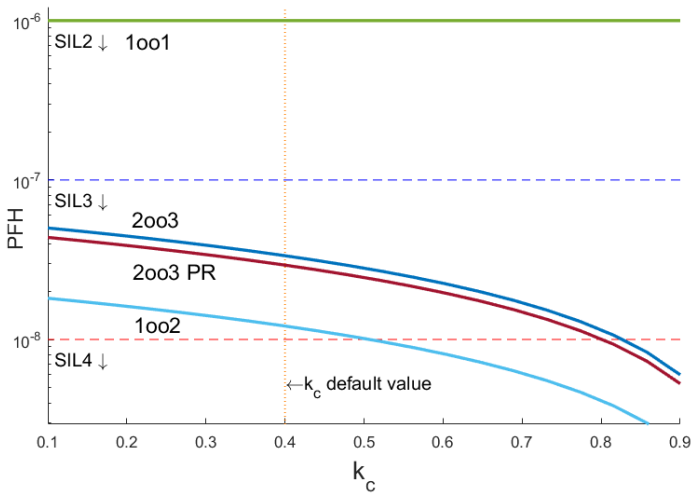

Figure 2.16: Effect of k^c variation over safety.

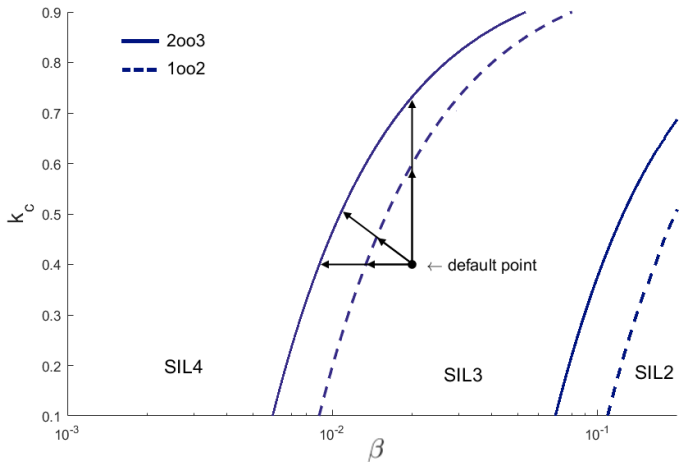

Figure 2.17: Simultaneous improvement of β-factor and k^c to reach SIL4 level.

Sensitivity to μ_{OT}:

Online repair which is invoked after online failure detection, is either employed when a single PE is not operational due to a DD failure (provided that partial recovery is available) or when DD failures are tolerated until whole system is in fail-safe state (if partial recovery is not provided). Effect of repair rate in the former (only applicable to 2oo3 with partial recovery) is negligible. Since such repair does not reduce the number of DU failures. In the latter, effect is

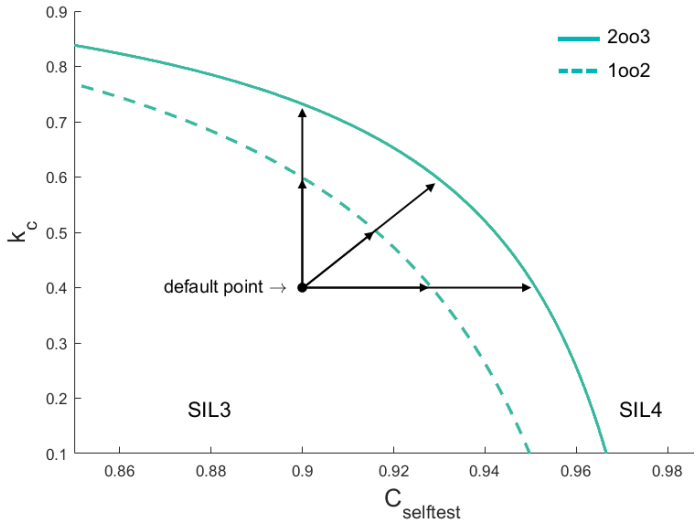

Figure 2.18: Simultaneous improvement of $C_{selftest}$ and k^c to reach SIL4 level.

zero as expected (see Figure 2.19, refer to Table 2.1 for fixed parameter values.). Note that repairs from down states toward up states are not considered in PFH calculation (refer to Figure 2.5). In practice, this parameter is useful for adjusting reliability and availability.

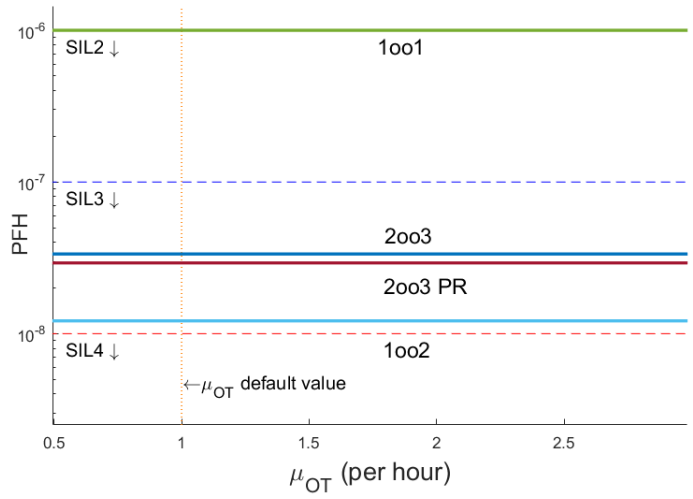

Figure 2.19: Effect of online repair rate variation over safety.

Sensitivity to μ_{PT}:

Proof test and repair occurs periodically in long periods of time (at TI - test interval) to remove DU failures. It is either employed when whole system is in

unsafe state or while a DU failure is tolerated (as in both safe configurations: 1oo2 and 2oo3). In the former, its effect on PFH is zero, similar to online testing, since repairs from down states are removed in PFH calculation. In the latter, although number of DU failures are reduced, due to dominance of CCF rate, such improvement is not observed in safety (see Figure 2.20, refer to Table 2.1 for fixed parameter values.). A single DU CCF failure can defeat safety in both 1oo2 and 2oo3, while two consecutive DU independent failures have to occur for leading to the same situation.

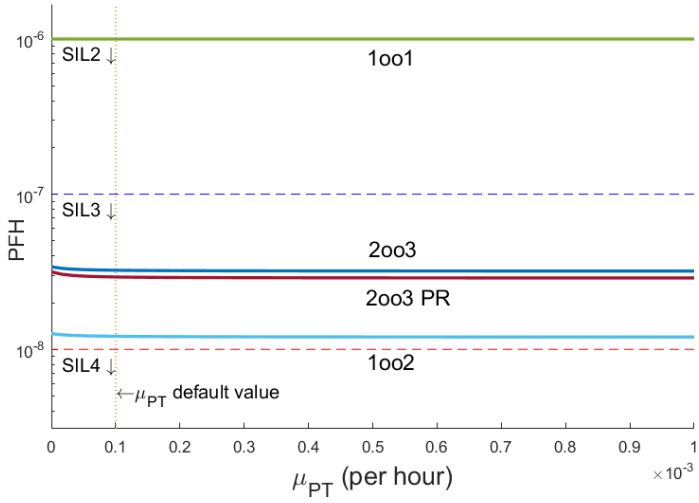

Figure 2.20: Effect of proof-test rate variation over safety.

Sensitivity to partial recovery:
As illustrated earlier in Figure 2.9, if partial recovery (repair) is not provided for 2oo3 system, gain in reliability which is the main advantage of 2oo3 over 1oo2 is not significant. Therefore, in this case, usage of the more complex 2oo3 configuration is not logical. Unlike reliability, effect of partial recovery over safety (PFH) is negligible (shown in Figures 2.11 to 2.20). Note that, partial recovery removes DD failures affecting reliability due to less number of transitions into fail-safe state. On the other hand, PFH is mainly a function of DU failure rate.

Sensitivity to t_D:
In online testing, time to detect a detectable failure (t_D), is the time between occurrence and detection of a failure. Equivalently, $\delta = 1/t_D$ is the frequency of online testing per hour. There is no direct reference to this parameter in IEC

61508 standard (except briefly for β_D estimation), probably because in comparison to component failure rates, it is assumed to be negligible. In order to capture this parameter, an intermediate state is added to Markov chain for every DD failure transition (shown in Figure 2.21), in which the DD failure is temporarily considered as a DU failure. This additional state makes the Markov chain more complex. Therefore, due to Markov chain solution complexity, we only execute this for 1oo1 and 1oo2 configurations. Intermediate states can be one of three types, operational, fail-safe or unsafe, as other normal states. But, for PFH calculation, they are not absorbing, meaning the transition of online testing is not removed. They also contribute to decrease the safety by causing more possibility of transition into unsafe states. Based on experimental results shown in Figure 22, we can observe that if $\delta < 0.1$ per hour, safety level is slightly affected, while if $\delta < 0.01$ per hour, the effect is significant.

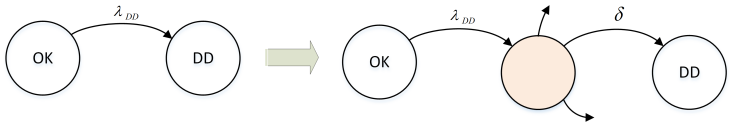

Figure 2.21: An intermediate state is added to Markov chain, in which a DD failure has not been detected yet.

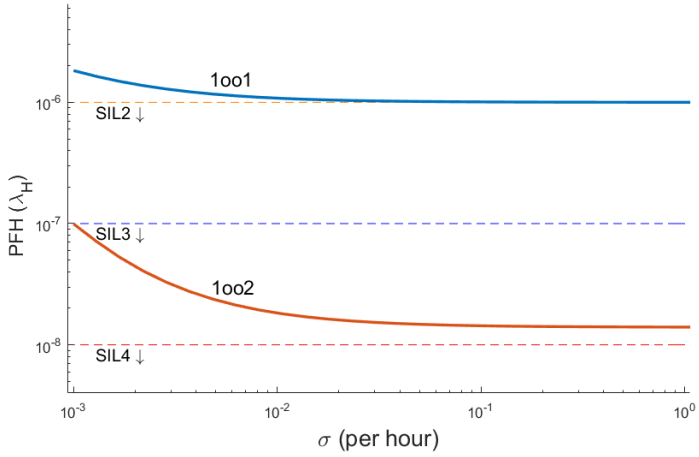

Figure 2.22: Effect of δ $(= 1/t_D)$ variation over safety.

2.6 Approximate Rate Calculation on Markov Chain

Our experimental results show the significance of β-factor in safety systems. Considering this fact and the $\Omega(n^2)$ runtime required for Markov transition (matrix multiplication formula given in Section 2.4.3, where n is the number of states), we propose a method for simplifying complex Markov chains into simpler ones. In this way, a quick and approximate failure rate can be calculated. Although CCF transitions have smaller rate (multiplied by β), but due to jumping over several states, their impact on the system failure rate is decisive.

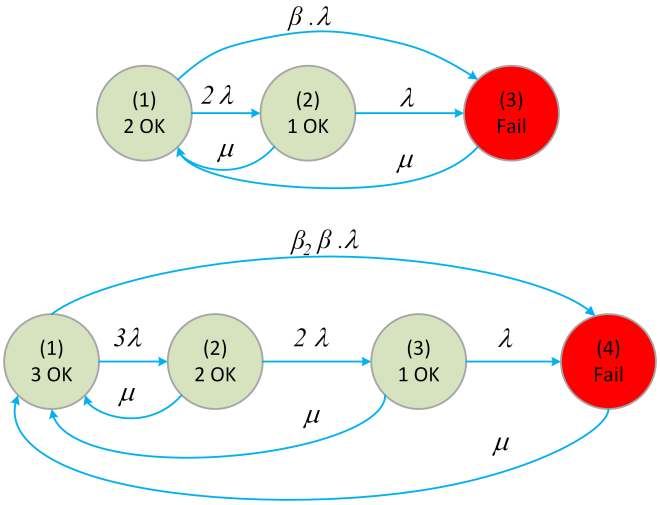

Figure 2.23: Simple models for measuring CCF2 and CCF3 influences. $\lambda = \lambda_{PE}$, $\mu = \mu_{PT}$.

In the simple models depicted in Figure 2.23 (without online testing), two and three consecutive component failures move the system into fail state, while a single CCF has the same consequence (they are called as CCF2 and CCF3, depending on the number of jumps). In such a setting, it is desirable to compare the share of CCF and non-CCF transitions in the total failure rate. By using the conventional reliability formulations, failure rate is calculated for three scenarios: 1) full model, 2) keeping only the CCF transition, 3) keeping only non-CCF transitions. In the results shown in Table 2.3, as β increases, share of CCF transitions also increases. However, even at a smaller β value, order of the CCF rate is comparable to the

accurate result. While even solving such simple Markov chains requires computer applications, scenario 2 gives an immediate result in both models, where system failure rate is equal to CCF transition rate.

	$\beta = 0.02$	$\beta = 0.1$
Full model (Figure 2.23:top)	8.7E-7	16.1E-7
Only CCF2 transition	2.0E-7	10.0E-7
non-CCF2 transitions	6.7E-7	6.7E-7
Full model (Figure 2.23:bottom)	1.08E-7	3.24E-7
Only CCF3 transition	0.6E-7	3.0E-7
non-CCF3 transitions	0.54E-7	0.54E-7

Table 2.3: Failure rates of systems in Figure 2.23 for three scenarios.

Following this observation, we propose a simplification for complex models. This way, only the paths from start to fail state which include a CCF transition are preserved, while transitions or states not included in these paths are removed. The idea is explained on a simple Markov chain for a 1oo3 system shown in Figure 2.24 (refer Table 2.1 for parameters). Using the same model as shown in Figure 2.2, transition terms are:

$$\lambda^i s = \lambda_{PE} \qquad \lambda^c d = \beta \cdot \lambda_{PE}$$
$$\lambda^i d = 2(1-\beta) \cdot \lambda_{PE} \qquad \lambda^{c2} t = 2.1\beta \cdot \lambda_{PE}$$
$$\lambda^i t = 3(1-1.7\beta) \cdot \lambda_{PE} \qquad \lambda^{c3} t = 0.3\beta \cdot \lambda_{PE}$$

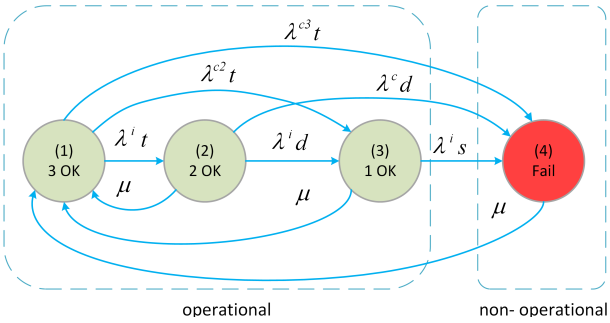

Figure 2.24: A simple model for 1oo3 configuration.

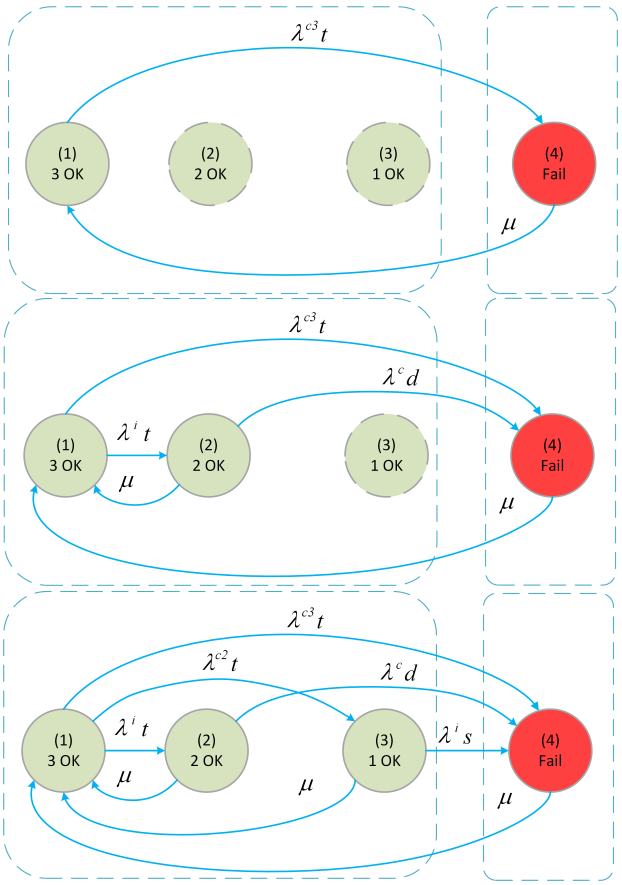

Figure 2.25: Transition paths with longer CCF jumps are added one by one from top to bottom.

Starting from state (1), there are four different paths that lead to the fail state, namely, 1-4, 1-2-4, 1-3-4, and 1-2-3-4, where shorter paths have longer CCF jumps and more share in PFH value. To model this, we start with an empty Markov without any transition, and we add these paths one by one as shown in Figure 2.25. Solutions to these three approximate Markov chains and the complete one are illustrated in Figure 2.26 (refer Table 2.1 for fixed parameters). As can be seen, even the simplest chain which includes only a single edge, provides a suitable approximation to estimate PFH and SIL level. In the simplest case, there is no need to solve the Markov as it is obvious that $PFH = \lambda^{c3}t$.

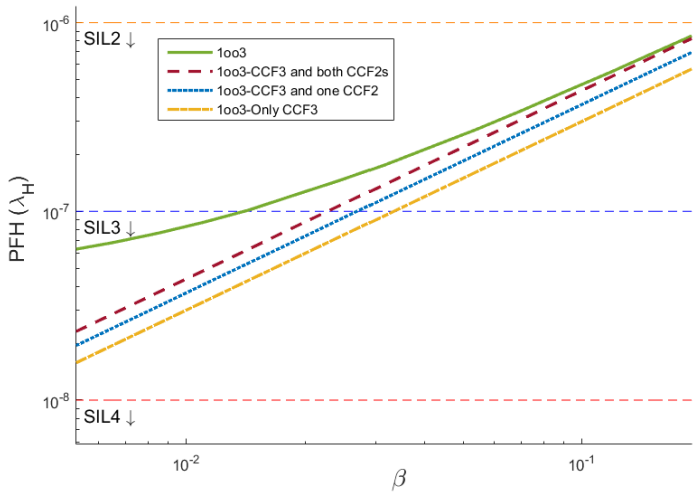

Figure 2.26: Comparison between complete and simplified Markov chains of 1oo3 system.

2.7 Summary

In this work, we analyzed the sensitivity of system safety to some critical design parameters in two basic multi-channel safe configurations, 1oo2 and 2oo3, where a 1oo1 system is used as baseline. All configurations have been modeled by Markov chains to examine at which safety integrity level (SIL) they stand, and how distant they are from the target. Hence, each safety parameter's contribution on safety can be understood. Through these measurements, instead of blindly improving an unsafe system, designers can make an informed decision to select the most appropriate parameter for improvement. A significant presence of Common Cause Failures (CCFs), is an important assumption for analyses in this study. Through experiments, we showed that there is a linear relationship between safety (PFH) and two parameters: λ_{PE} and β, in logy-logx plane. We also observed that parameters which have considerable effect on CCF rate are more appropriate candidates for safety level enhancement. These include λ_{PE}, β, $C_{selftest}$, and k^c. Additionally, we propose a method for simplifying Markov chains in PFH calculation which can largely reduce the complexity to get an approximate result.

Chapter 3

HLS-based High-Throughput and Work-Efficient Synthesizable Graph Processing Template Framework

3.1 Introduction

As the complexity of digital systems increases, preferring productive High-Level-Synthesis (HLS) languages, such as SystemC, over conventional low-level ones, like RTL-level Verilog and VHDL, in high-performance Computing (HPC) applications becomes inevitable. Besides, as CPU-FPGA hybrid platforms are becoming omnipresent, particularly in IT domain like in data centers to run analytics applications, writing FPGA programs for these platforms by software programmers will be a dilemma. This mandates having even higher level HLS languages, like Intel OpenCL for FPGA, where even most basic digital design concepts like clock signal are absent in the syntax. Currently, mainstream FPGA-makers are

See [27] for the original work.

following this approach to provide better programmability. However, when abstraction level of design increases, optimality of implementation, particularly in complex designs, degrades. This issue specifically applies to graph processing applications, in which, as we will discuss, the shrewd system architecture is necessary to efficiently exploit the valuable memory bandwidth. Considering these two opposing requirements, namely, easier programmability and implementation efficiency, our goal is to have a hybrid graph processing platform, mixing both cycle-accurate implementations to provide efficiency, with high-level programmability to support productivity.

In this chapter, we propose an HLS-based graph processing template framework with the architecture of a host processor, connected to the multiple hardware accelerators on the FPGA [27]. A software program on the host processor is only responsible for initiating the execution of FPGA, with no further intervention afterward. FPGA's user codes are composed of two kinds of modules. The first one is the collection of fixed user modules, provided by our framework, which is written in SystemC and in a cycle-accurate way, providing efficient/high-throughput vertex processing pipeline. The second one is a single customizable module, to be written by the user in pure C/C++, in which he/she writes intended iterative graph algorithm, merely inside this specific module of the template. In this architecture, parallelism is provided in two dimensions, across multiple accelerators, and also inside a manually designed pipeline of each accelerator. While not limited to, our tool is specifically usable on the emerging generation of Intel platforms, Xeon+FPGA, where our experiments have been executed. For comparison purposes, candidate graph algorithms are also implemented in Intel OpenCL for FPGA, the alternative HLS platform presented for Xeon+FPGA. In very high-level Intel OpenCL, the user program is written in a straightforward way, rather similar to software programming, where a pipelined architecture is automatically generated by the architecture compiler tool. We show that our reusable template can be used to implement a high-throughput or work-efficient graph algorithm by only writing pure software code. This way, the programmer does not need to engage in the complicated HLS techniques or transformations, typically required to generate an efficient architecture out of an HLS design. The

main contributions of this work are as follows:

- We propose a high-performance/work-efficient, SystemC based, synchronous, deeply pipelined, and fully synthesizable graph processing framework, ready to be implemented on FPGA.
- The template-based framework is usable by non-hardware experts, such as IT engineers, to generate FPGA programs using only C/C++.
- The proposed framework is specially tested on, and prepared for the Intel state-of-the-art Xeon+FPGA platform. However, it can be extended and adapted to other similar architectures.
- We implement a novel fast *bit-vector* to keep an active vertex list, which is mapped to FPGA Block RAMs. This enables work-efficient graph processing.
- We compare and contrast alternative HLS platforms for FPGA, including OpenCL, to show limitations and difficulties in implementing high-performance pipelined graph algorithms.

This chapter is organized as follows. The next section describes the background on graph processing as well as the Intel Xeon+FPGA platform. Section 3.3 gives the relevant studies on the topic. In Section 3.4, baseline implementations of graph algorithms in Intel OpenCL, various structures and drawbacks are discussed. In Section 3.5, our HLS-based pipelined graph processing architecture is introduced with details. In Section 3.6, the experimental setup and results are discussed. Finally, the chapter is concluded in Section 3.7.

3.2 Background

3.2.1 FPGA or GPU?

Currently, utilizing the computational power of GPUs in user applications such as multimedia processing tools, deep-learning algorithms, or scientific computations is a well-known approach for high-performance Computing (HPC). This General Purpose GPU (GPGPU) computing can potentially be limited due to control divergence and memory divergence [28]. In contrast, graph-based applications typically have data-dependent behavior, both in control flow and memory references, mainly due to the diversity of graph topology.

In a more recent trend, High-Performance Reconfigurable Computing (HPRC), which is integrating FPGA-accelerators into general-purpose processors, has attracted the attention of dominant market vendors. Recently, Intel Corp. has bought Altera, one of the leading FPGA-makers, in a significant investment valued at 16.7$ billion, which was recorded as the largest deal in chip-makers history. Despite volatile architecture and lower operational frequency of FPGAs, due to complex routing interconnects, there are many applications where the flexibility of FPGA architecture to design a deep and customizable pipeline, outperforms computational power of GPGPUs, for equal available off-chip memory bandwidth [29, 30]. In this trend, as FPGA-accelerators are becoming serious competitors for GPUs, many cloud service providers extensively use them in their data centers to provide massive parallelism [31]. Currently, Amazon offers FPGA nodes on EC2 platforms [32]. Microsoft has integrated over 100 million FPGAs into its data centers aiming at having FPGA-powered high-performance real-time AI systems, including enhancing the performance of Bing web search engine [33, 34].

3.2.2 Hardware Accelerator Research Program (HARP) Platform

We implemented and tested our approach on a version of the state-of-the-art Intel Xeon+FPGA platform, where a Xeon processor is connected to an Arria10-GX1150 FPGA, via two PCIe and one QPI channels (see Figure 3.1). On this platform, a DDR memory is available on the processor side, and FPGA can access it through three serial channels and processor memory controller. The maximum total read/write bandwidth between processor and FPGA is ∼20 GB/s, while according to our measurements, read and write can go up to ∼17 and ∼15 GB/s, respectively. Part of the FPGA bitstream, which includes implementations of memory and host CPU communication controllers, to be used by FPGA, has already been provided to users by Intel SDK, as so-called fixed *blue-stream*. In addition to the blue-stream, with FPGA partial reconfigurability, user develops his/her own hardware design as Accelerator Function Unit (AFU). The customizable user code, called *green-stream*, attaches to the existing blue-stream to make a full bitstream. In this setting, user logic implementation can work at a maximum frequency of 400 MHz. After programming bitstream to FPGA, a host-side software initiates FPGA run and thereafter, FPGA continues operating without host CPU intervention until indicating a done signal.

We have implemented our approach on a remote system, under Intel Hardware Accelerator Research Program (HARP), at Paderborn University in Germany.

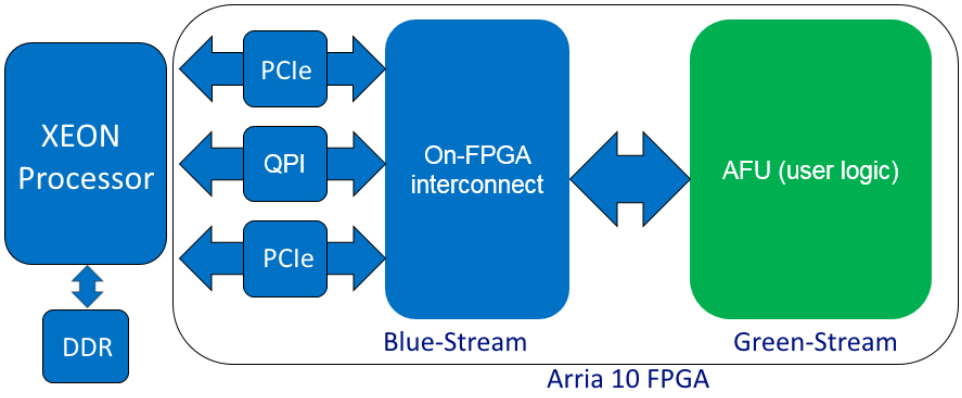

Figure 3.1: Intel Xeon+FPGA platform.

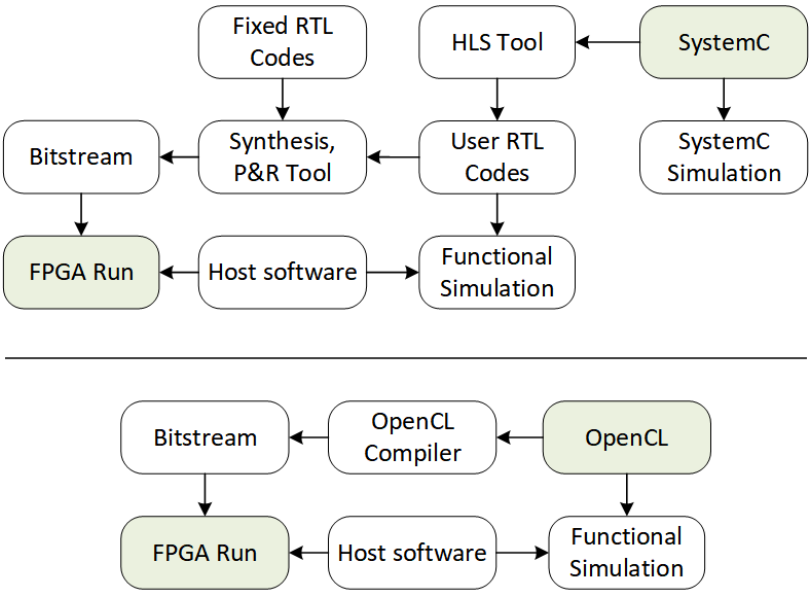

Figure 3.2: FPGA design flow using SystemC (top) and OpenCL (bottom) for Xeon+FPGA platform.

At the time of writing this chapter, two SDKs that we used are only available under the HARP program, with some limitations. The two applied design flows, for baseline and the template, are illustrated in Figure 3.2. As can be seen, OpenCL flow (bottom) is simpler in terms of execution, whereas SystemC flow (top) involves HLS and placement and routing tools. From a user point of view, OpenCL code is directly converted to bitstream, but in SystemC, intermediate RTL code is generated. To convert SystemC to RTL, we followed the literature [35], which requires some additional third-party tools. In both cases, a host software initializes the FPGA program.

3.2.3 Graph Processing

Iterative Graph Processing Model: A graph with a starting initial value per vertex/edge is iteratively processed in a defined algorithm. After every iteration, values are expected to improve, until they reach the final desired result. In order to complete the execution, all vertices must converge.

Memory Access Bottleneck: In large-scale graph applications, low cache utilization, high memory access latency, and low bandwidth utilization in accessing graph data on off-chip memory are well-known bottlenecks. This is because of the inevitable random (irregular) memory access pattern. Diverse references to graph data (e.g. neighbor vertices/edges of a vertex) on a wide range of memory addresses lead to poor data locality. This, in turn, causes dramatically low cache utilization and low memory bandwidth utilization, in addition to large (typically around 100 clock cycle) data access latency. Hence, graph applications are typically memory bandwidth-bound.

Gather-Apply-Scatter (GAS) model: Represents three conceptual phases of a vertex-centric graph program. In the gather phase, data of adjacent vertices/edges is collected. Then, in the apply phase, after the necessary calculations, the new vertex value is calculated and updated. Finally, in the scatter phase, the neighbor vertices are informed about the value change in the current vertex. In the GAS model, data access is limited to neighbor vertices.

Convergence: A vertex is said to be converged when its corresponding data value reaches to its final value (or gets close enough, if the data type supports). This value will not change in the next iterations. A graph is said to be converged if all vertices are converged.

Pull-based vs Push-based: There are two general strategies to implement graph algorithms. In pull-based, vertex execution is composed of first, some data reads from neighbor vertices/edges, then some computation, and finally write a new value to current vertex, without informing neighbors (in other words, there is no explicit scatter phase). In contrast, in push-based, after doing calculations, the vertex may write data to neighbor vertices too [36]. Comparatively, pull-based has more number of reads and higher edge processing throughput, while push-based has more writes. Depending on the algorithm, one may be a better fit than the other. For example, in the case of straightforward and non-work efficient Breadth-First Search (BFS), push-based can be by far more efficient than pull-based. Because in pull-based, deep vertices such as leaves of a tree graph, unnecessarily iterate over all parents for many iterations waiting for the

parent to change, and waste valuable memory bandwidth by vain data reads. Even though push-based is more efficient in general, it lacks in terms of parallel execution due to race conditions and false-sharing. Consistent solutions, such as using atomic accesses, have high hardware cost, or performance degradation because of serialization. Our work is based on a pull-based design, but as will be explained, work-efficient mode obviates unnecessary vertex processing.

Synchronous vs Asynchronous Execution: In an iterative graph algorithm, in each iteration, all or part of vertices are processed, for which, next data values are calculated. There are two strategies about when to update vertices with their calculated next value. In asynchronous execution, vertex value is immediately updated, then neighbor vertices can read new value immediately, in the same processing iteration. In contrast, in synchronous execution, the next value is kept in a temporary variable and is updated only at the end of the iteration. Therefore, there are two copies of data, one for reading older values, and the other for writing new values. New values become available for reading starting from the next iteration. Synchronous execution converges more slowly, but it has an easier implementation, as there is no intra-iteration dependency. On the other hand, in asynchronous execution, some complexities may raise, such as the possibility of race-conditions (because data is both read and written), and ensuring sequential consistency needed for correctness, where consistent solutions can be costly [37].

Workload Imbalance: In many systems, irrespective of being implemented at the software or hardware level, balancing the workload becomes a significant problem for graph applications because of the power-law distribution of vertex-degrees. Techniques like dynamic load balancing or vertex-degree aware scheduling, as mentioned in Section 3.3 try to tackle this problem.

High-Throughput vs Work-Efficiency: In executing graph applications, one can opt for high-throughput execution, where all vertices are sequentially loaded to the pipeline in every iteration for possible processing. On the other hand, in work-efficient execution, only a part of the vertices are active. The active vertices that need to be processed are called *active set*. The former achieves a higher number of processed edges per second, whereas the latter converges faster.

Setting the number of edges processed per second as the only performance metric may not be a smart decision since it is inefficient in terms of work. However, work-efficiency support can have significant complexity and overhead [37].

3.3 Related Work

While a wide variety of approaches in graph processing exist both at the hardware and software level, to focus on FPGAs, we do not consider complete software frameworks developed for providing users with high-level and easy to use software modeling on CPU and GPU. We've already explained in Section 3.2.1 that why GPUs are not serious options for graph processing. Moreover, comparisons with CPU have been provided by FPGA works we cover below.

In a series of works, with focus on optimizing off-chip memory bandwidth efficiency, techniques such as data layout and compression are proposed. For example, in [38], significant potential of locality in real-world graphs is explored in an online fashion. A locality-aware online scheduler, tries to improve data reuse by exploiting the community structure of real-world graphs, and predict the well-connected regions ahead of time. In [39], the inherent graph property, 'vertex degree', is considered for optimizing a software/hardware co-design architecture. Since high-degree vertices can be the bottleneck in graph algorithms, authors propose degree-aware adjacency list reordering. In [40], using common CSR graph data format, first graph workloads are analyzed to show higher performance sensitivity to L3 cache size, rather than L2 private cache size. Based on profiling insights, like data reuse distances, an application specific and data-aware data prefetcher is proposed to increase inherent data reuse. In [41], different types of irregularities in graph analytics are classified. Then, to alleviate them with a co-design approach, data-aware dynamic scheduling to schedule the program on-the-fly is suggested, which has microarchitecture support to extract data dependencies at run-time. Such techniques are generally orthogonal to our work and can be applied simultaneously.

A set of implementations due to very low-level hardware modification or implementation complexity are mainly suitable for ASICs (application specific integrated circuit) rather than FPGAs. Graphicionado [42] proposes a set of data type and memory subsystem specializations to reduce memory access latency. Processing-In-Memory (PIM) based accelerators like [43], reduce memory access cost by integrating accelerators inside the memory. In [44, 45], authors offer a configurable, work-efficient, asynchronous and template-based graph processing accelerator architectural model. After all, due to excessive complexity devoted to ensuring advanced features listed in [37], such as strict sequential consistency property in asynchronous execution, the design is not practically usable on FPGA platforms. FPGA can not afford the intricacy, area size, and burden of interconnect network of architecture to give a practically efficient implementation. In our work, architecture is simplified enough for synchronous execution to better fit to an FPGA. In addition to the high-throughput mode, the work-efficient feature is added with a novel and efficient bit-vector.

There have been frameworks particularly for large-scale graph processing on FPGA. ForeGraph by Microsoft [46] introduces a scalable multi-FPGA architecture. The graph is partitioned among FPGAs, inside dedicated off-chip memories, while an optimized inter-communication mechanism among FPGAs exists. In the proposed scheduling scheme and data compression technique, the graph is loaded into fast on-chip Block RAM memory used as cache, and dedicated off-chip memories provide higher bandwidth. The main idea is to have more Block RAM by using multiple FPGAs, hoping for more data locality. ForeGraph also has edge reordering for edges with potential data write conflict. Despite having rather a large cache (16MB per FPGA), multiple data optimization techniques for more locality, a high number of parallel processing elements with shallow pipelines (with handling data conflicts), and after all, a complex multi-FPGA full framework, the improvement over our work, which only focuses on efficient pipeline design, with no cache, no locality assumption, no dependability on graph size, and no data optimization techniques, is not largely better (worst case: 364 vs. 225 M-Edge/s, BFS application, per FPGA, similar memory bandwidth). This comparison is with our high-throughput (HT) mode, while as we will see, our

main idea is work-efficient (WE) mode which decreases run-time largely. In [47], a data layout technique with architectural support is proposed to minimize the number of random accesses to external memory, which also reduces the power consumption of on-chip Block RAMs. In HitGraph [48], a design automation tool is proposed to generate a synthesizable RTL code for graph accelerator. Besides, to improve performance, several algorithmic optimizations, such as graph partitioning, optimized data layout, and inactive partition skipping are introduced. In the aforementioned studies, processing pipelines are usually simple and shallow, where side techniques are the main novelty for coping with memory bottleneck problems. However, our work without conflicting with these techniques focuses directly on pipeline architecture, where we intentionally remove possible locality from graph data with initial shuffling. In [49], a parallel accumulator is proposed to remove serialization in atomic operations for conflicting vertex updates, applicable to specific graph algorithms. On the other hand, WaveScheduler [50], proposes a scheduler for Sparse Matrix-Vector Multiplication (SpMV) based multi-accelerator graph processing on FPGA. Besides two data re-ordering optimizations, the key insight is the appropriate tiling of the underlying adjacency matrix to eliminate all read/write conflicts in on-chip BRAM. Again, most of these works generally focus on increasing the locality and are orthogonal to our work. Because we do not consider locality in our implementation, we rather focus on designing an efficient pipeline.

3.4 OpenCL-based Design

The simplest way to utilize the underlying FPGA architecture is through OpenCL, where an HLS tool for Xeon+FPGA by Intel (formerly Altera) is used. Unlike SystemC, in this environment, the programmer does not deal with clock signals explicitly. This is very convenient as the programming style can be rather similar to conventional C/C++ software programming. Hence, with the aim of making FPGAs attractive for all programmers, this C-to-Gate tool is usable by non-experts too. The OpenCL architecture compiler converts the software-like HLS code to logic, in a bitstream file to be programmed to FPGA. Since OpenCL

is mainly a pipeline generator, it is selected as our baseline.

3.4.1 High-performance OpenCL for FPGA

As widely accepted, automated and high-performance output generation is more difficult for a programming language at a higher abstraction level. Moreover, the shortage of inherent parallelism in a programming language (such as C/C++), makes extracting a parallel execution model and the required resource management, a challenging task for compiler [51, 52]. Generally, users are required to follow some specific coding style or use defined transformations to help compiler [53] extract the parallelism. In the case of OpenCL, the key point to have a high-throughput implementation for an iterative algorithm is writing the code in such a way that the compiler is able to automatically generate efficient pipelined hardware. In other words, having a smaller loop Initiation Interval (II) or pipeline stall rate as much possible is desirable. Initiation interval is the number of clock cycles between consecutive launches of an iterative task which ideally is one clock cycle.

There are two choices to structure kernels in Intel OpenCL: 1)ND-Range kernel and 2)task kernel (or single work-item kernel). While the former is mainly intended for GPUs, the latter is recommended for FPGAs [54, 55], as it benefits from FPGA-specific techniques. A task kernel is coded almost similar to C/C++ software program, and at the end, is converted into a single logic module. In this kind of kernel, the compiler implicitly generates pipelined hardware for the program, where for/while loops are used to provide concurrency in the execution of consecutive loop iterations. Moreover, additional parallelism can be obtained by having multiple instances of the same kernel (multi-accelerator). On the other hand, the ND-Range kernel is structured according to standard GPU-optimized OpenCL style. The kernel is converted into a small work-item (like a software thread), where a large number of these threads run concurrently on a Compute-Unit (CU) hardware core. Unlike task kernel, program loops are not pipelined inside an ND-Range kernel. However, pipelining is supplied at the work-item

level. A compute-unit pipeline can run many work-items concurrently. Similarly, for having additional parallelism on hardware, multiple compute-units can be instantiated. Work-items are grouped in so-called work-groups that are assigned to a specific compute-unit. A run-time scheduler is responsible for dynamically distributing work-items and work-groups in/along compute-units. If the kernel body is not work-item dependent (which is the case for us), single instruction multiple data (SIMD) can add further parallelism inside the compute-unit. In light of these OpenCL features, we will discuss generating high-performance OpenCL code for our applications.

3.4.2 OpenCL implementation options

In order to have an efficient OpenCL implementation for vertex-centric graph algorithms as a baseline, we executed several scenarios.

1. Starting with the recommended task kernel structure, we use a straightforward doubly nested loop, where an outer loop to go over all vertices, and an inner one to go over all edges of each vertex (see the top part of Figure 3.3). Since vertex degree is variable, the inner loop has variable length too. Therefore, in general, this style can not generate efficient pipelines [54, 55].

2. To overcome the abovementioned variable length loop pipelining problem, we examined the ND-Range structure, where loops are not pipelined statically. This way dynamic scheduling at run-time among work-items with the variable workload is a possible solution. Therefore, the same doubly nested loops were examined on an ND-Range kernel, with different configurations, including different work-item numbers, work-group sizes, compute-unit numbers, etc. In this case, vertices are divided to a large number of small chunks, and each chunk is processed by a work-item. Since the execution order of work-items is unknown, accessing the large shared global data on DDR memory (e.g. edge info) is not guaranteed to be consecutive. Due to this, low cache utilization even for sequential data is inevitable, which wastes the limited global memory bandwidth. Through a

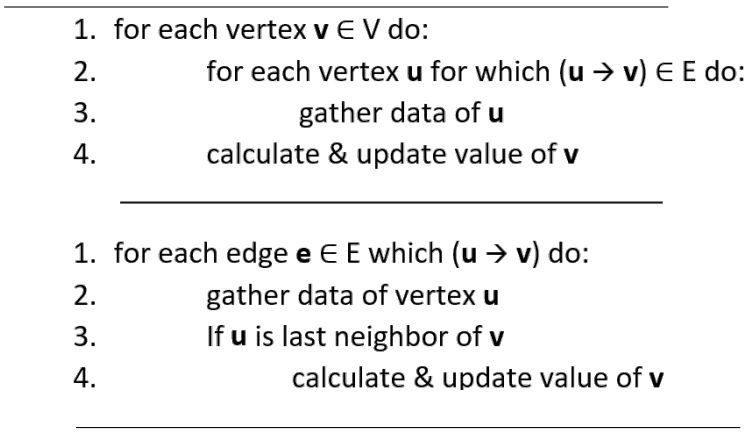

Figure 3.3: Pseudo-code for doubly nested loops (vertex-centric) at the top versus a single flat loop (edge-centric) at the bottom used in OpenCL implementation.

complicated manual implementation, a user-level tiny local cache was added to resolve this issue. This technique is called local banking [55] and comes with an additional area and Block-RAM usage cost. Even with these custom changes, pipeline utilization was not high enough, due to memory contention and high pipeline stall rate. This was especially the case for rather large graphs.

3. In a task kernel style, it is also possible to use a so-called edge-centric approach, where edges are traversed sequentially in a single flat loop (see the bottom part of Figure 3.3). This way edges are divided statically among kernels in large sequential chunks, thereby leading to better workload balancing when compared to vertex-centric processing. However, this also does not come for free. More specifically, a high loop body has high latency due to floating-point operations used in applications like PageRank. This prevented the generation of a high-throughput pipeline, where II$\geqslant$4 clock cycles. As a possible solution, one can replace floating-point data types with fixed-point. But fixed-point format can not support a wide range of real numbers, leading to precision problems in some cases. Another problem is with sequential global inputs that are not accessed in every loop iteration. For example, some of the vertex data are only requested when the edge counter transitions from one vertex to another. Since the compiler cannot handle this problem, as a manual solution, a helper kernel implements a FIFO, which repeatedly is filled by global data (from DDR) on one side, and

consumed by the pipeline on the other side. Eventually, the compiler can handle this case efficiently by achieving II=1. Therefore, one edge can be processed in every clock cycle by each kernel, provided that global memory bandwidth is sufficient. For more parallelism, a few kernels (accelerators) are instantiated, where each kernel is assigned a large chunk of graph edges.

3.4.3 OpenCL for FPGA: conclusion

Our extensive implementation trials with many different OpenCL styles to achieve a high-performance baseline on FPGA accentuates again on the difficulty of writing efficient code by such languages for realistic HPC applications [51, 53]. One evident reason in the case of OpenCL for FPGA is having less control over the final implementation, particularly, in the absence of clock signal in the language syntax. Besides, another noteworthy drawback is a very long compilation time for converting even a few lines of code to hardware. Furthermore, some custom requirements due to application or structure demanded excessive time to write programs and debug.

3.5 Template-based accelerator architecture

In this section, we describe the pull-based template pipeline architecture used for vertex-centric graph processing.

In each iteration, large chunks of vertices are dynamically assigned to, and processed in parallel running accelerators. For each vertex, all connected edges are processed in a loop. To optimally utilize memory bandwidth, enabling spatial locality and cache memory, vertex, and edge lists are fetched in-order. Graphs are stored in a common Compressed Sparse Row (CSR) format, which facilitates straightforward streaming memory access. Since vertex degrees can be rather

different, vertex processing times may differ considerably. Hence, vertices are executed (processed) out-of-order. Similar to load, vertices are also committed and written back to memory in-order, for efficient usage of memory bandwidth, and avoiding potential false-sharing problems. This way, the majority of bandwidth (more than 90% in our experiments) can be dedicated to inevitable random (irregular) memory accesses, reading data of a neighbor vertex.

As mentioned earlier, large-scale graph applications have a well-known inherent bottleneck in accessing off-chip memory, which leads to high latency and low bandwidth efficiency. Carefully utilizing this limited resource, requires precise execution, which can be done in a cycle-accurate level. But, this at the same time is tedious and complicated. In our template-based design, we implement the majority of common modules only once, except for a single user specific module. In a deep-pipeline vertex processing architecture, multiple vertices are in processing phase, in different stages of the pipeline, with many concurrent outstanding memory requests to tolerate high latency of main memory access. Architecture can be configured in two execution modes. In lighter high-throughput mode, all vertices are loaded to the pipeline in each iteration. However, in work-efficient mode, using a novel and fast bit-vector design to implement the active-list, only active vertices are loaded and processed in each iteration. We explain both of these options in the following subsections.

3.5.1 High-throughput mode

Figure 3.4 gives the different modules in our high-throughput architecture which we explain in detail below.

Data and Control Tables: There are a few tables to keep track of vertices being processed. They keep vertex states, such as vertex degree and value, number of remaining unprocessed edges, and temporary data in gather phase. Table length is the maximum number of vertices in execution at the same time (pipeline depth is set to 128 because of 100 clock cycle memory latency). Different pipeline stages of design may have simultaneous read/write accesses to these tables. Their

implementation which usually has resource contention and timing constraints has to be efficient too. For this purpose, on-chip multi-port memory resources of FPGA, including Block RAMs, and even memories made by LUTs and Flip-Flops are utilized.

Table Allocator: is responsible for allocating a vacant row to the next incoming vertex-ID in control and data tables. After that, the assigned row-id which points to a specific vertex flows through the next pipeline stages until the end of processing.

Vertex Initiator: sequentially reads a free row-id from 'Table Allocator' queue, and an unprocessed vertex from a streaming memory port, to fill some table entries with initial vertex data, such as vertex value and degree. This module internally is composed of multiple pipeline stages for higher throughput.

Edge Loop Setup: sequentially, reads a row-id of an initialized vertex from the 'Vertex Initiator' queue, and information of all connected edges from a streaming memory port. Data of a connected edge contains vertex-ID of the other (neighbor) vertex. After that, in a so-called edge loop, for each edge of the current vertex, a random access memory request is sent to read data of a neighbor vertex. Row-id of the current vertex under processing is also attached to the request to be returned back with the response (supported by memory controller). Since responses may come out-of-order, this ensures that the owner vertex of response will be known. This is the only irregular memory access in the overall design. When all requests for neighbor vertices of the current vertex are sent, the module moves to the next vertex (row-id) from the 'Vertex Initiator' queue. Multiple vertices being processed (up to 128) may have many pending vertex info requests for their own neighbors. Having many in progress outstanding memory requests hides large off-chip memory latency (over 100 cycles). Ideally, requests are sent successively one per clock cycle.

Edge Loop Execution: Responses of neighbor vertex info requests, sent by the previous module, are received out-of-order, ideally one per clock cycle. Responses

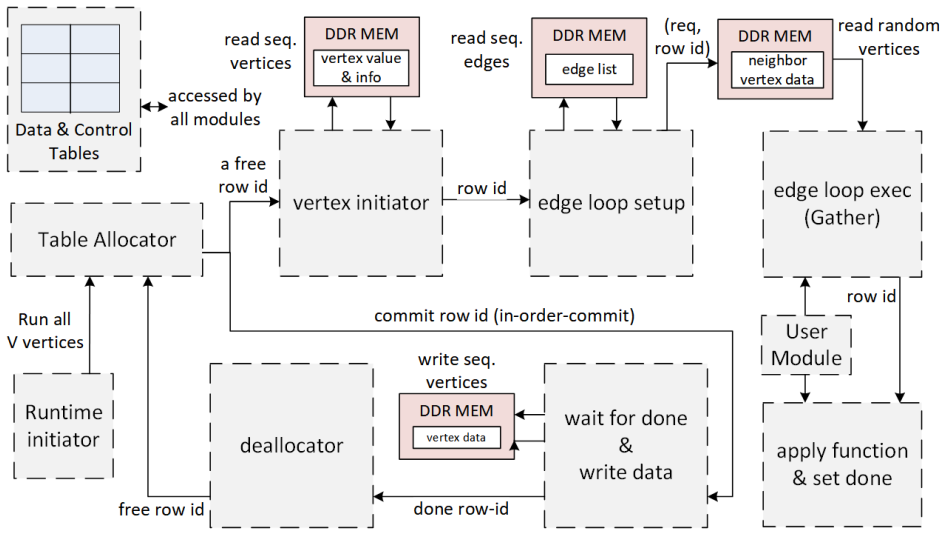

Figure 3.4: Simplified diagram of one high-throughput deeply pipelined vertex processing accelerator.

may also belong to various vertices. Because of poor locality, it is highly probable that each read response consumes one full cache-line of memory bandwidth. Data of each neighbor vertex is 'gathered' (accumulated, or so, depending on the specific graph algorithm) in temporary variable(s), in the table row of the related vertex. When all neighbors of any of under processing vertices are processed, its row-id is passed to the 'apply' stage.

Apply: In this module, after having a final complementary calculation on the result of the 'gather' stage, the vertex value is updated again in the related table. Then, a 'done' bit is set for this vertex in the related table. Similar to other modules, vertices may finish this step out-of-order.

User Module: All customizable user codes are inside this module. It includes functions applicable to vertex data in gather and apply phases.

Write-Data: sequentially reads a vertex row-id from commit queue, and waits for its completion by monitoring the 'done' bit. When the done signal is detected, vertex's new value is written back to the DDR memory. As row-ids in commit queue are in-order, writings are also done in-order. This way, one iteration of the iterative algorithm, for a specific vertex is accomplished. For realizing a

synchronous update strategy, the read and write vertex-info data structures on the off-chip DDR are different.

Deallocator: finally releases all table entries for row-id of a finished vertex, to be reused again by the next vertices.

3.5.2 Work-efficient mode

As explained before, Work-efficient architecture does not execute for all the vertices in every iteration. Therefore, it has some additional modules, plus some modifications to previous ones. They are shown in Figure 3.5 and explained in detail in the below paragraphs.

Pre-Fetch: Sequential vertex access in the 'Vertex Initiator' module of high-throughput mode, makes it possible to have a single read/write request for a large vertex chunk (containing whole vertices dedicated to each accelerator). But in work-efficient mode, only some of the vertices are processed in every iteration. Hence, instead of a big chunk, there should be multiple disjoint and smaller ones. To reduce access overhead, especially for small and near-by-address memory requests, a grouping mechanism is provided to create such smaller chunks at run-time. The pre-fetch module is responsible for batching consecutive individual vertices (consecutive vertex-ID), into one large batch. Therefore, a single memory request is sent for the whole batch (chunk).

Read-CL: If successive disjoint vertex read requests have overlapping cache-line data (end of one request and beginning of the next one), the memory request is responded by a local single size read cache-line. This can be helpful for small and near-by-address requests. This module also provides the value of unmodified vertices for the next write operation.

Write-CL: Similar to its read counterpart, this module is responsible for handling a cache-line for the write operation. The cache-line is filled one-by-one and then flushed into the memory.

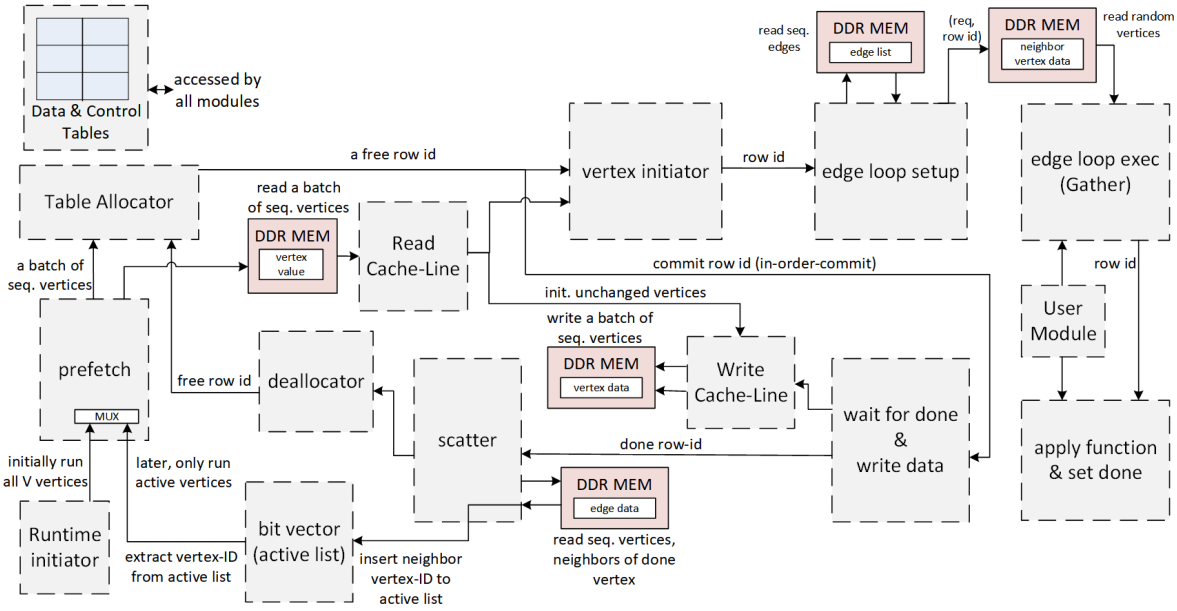

Figure 3.5: Simplified diagram of one work-efficient deeply pipelined vertex processing accelerator.

Scatter: Finally, after updating a vertex with its new value, provided that the change is beyond a specified threshold, the scatter module informs all neighbors about its value update. This module iterates over all neighbors, and send their vertex-ID to bit-vector module to construct active-list of next iteration.

Bit-Vector: This key module is responsible for keeping a list of active vertices in work-efficient mode. This way, in each iteration, instead of processing all of the vertices, only those tagged as active are processed. Efficient implementation of active-list can be challenging [37]. In our pipelined and fast solution, active-list members can both be quickly inserted to and extracted from the bit-vector module as they are mapped to fast on-chip Block RAMs. As depicted in Figure 3.6, the Bit-Vector is composed of two sets of ping-pong swappable memories. One keeps the active-list of current iteration which is read from, and the other stores the active-list of next iteration which is written on to. Ping-pong memories are swapped after each iteration. In the basic bit-vector implementation, a single memory bit is assigned to a vertex, where being '1' indicates being in the active-list. While inserting a vertex-ID into on-chip memory is naturally fast, by just setting a single bit, on a specific address (vertex-ID) to '1' in a single clock

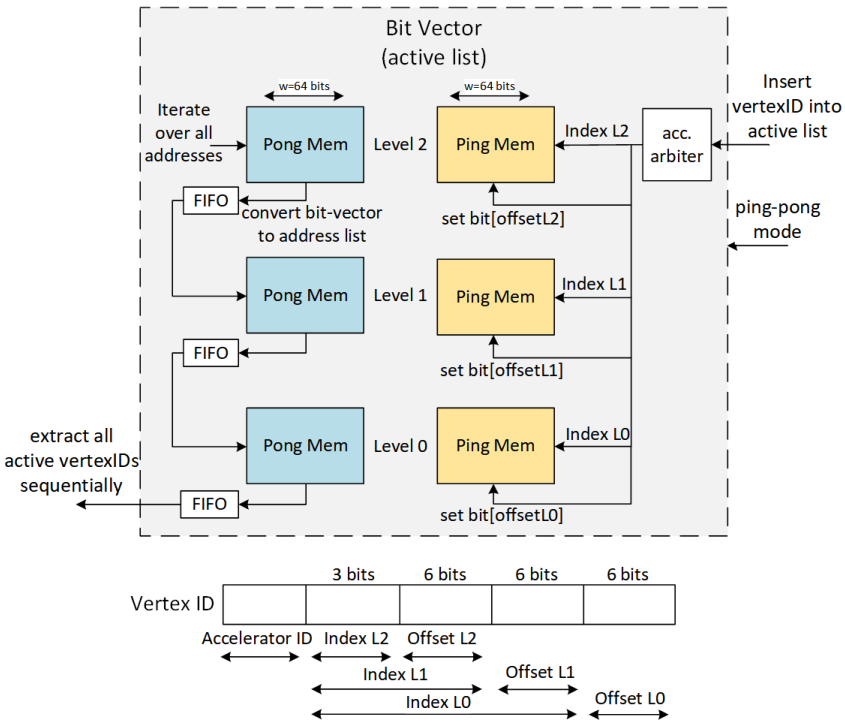

Figure 3.6: Multi-level bit-vector architecture in the basic version, where list resolution is one vertex.

cycle, retrieving them can be rather slow, since the whole bit-vector should be scanned. To tackle this problem, besides internal pipelining, a novel multi-level design is proposed for the Bit-Vector. In the advanced implementation, vertices are addressed in three different levels. Higher-level active-set is responsible for coarser groups of vertices, meaning larger chunks. When a vertex is added, it is added to all hierarchy levels in parallel as shown in the right part of Figure 3.6. However, the benefit is realized in the reduction of active-list retrieval latency as shown in the left part of Figure 3.6. Using higher-level indicator bits, non-active chunks of vertices are quickly skipped without deeply searching in lower level addresses. At the start of each iteration, the small top-level list is fully searched for any active chunk. Then, only active addresses are collected in a FIFO and passed to the second-level list. The same is done for the third list level. Finally, the list of all active vertex-IDs is extracted monotonically and sequentially.

Scheduler: For alleviating possible load imbalances, a run-time scheduler module (not shown in Figures) is responsible for dynamically assigning the next vertex

chunk to the first idle accelerator.

Note that, the available block memory size of the underlying FPGA, determines the maximum graph size supported in the work-efficient mode. In this version of Xeon+FPGA, and in the basic version of the work-efficient active-set, where a single memory bit is assigned to each vertex, graph size of 8 million vertices is supported (2 million per accelerator as shown in Figure 3.6). By reducing the resolution it is possible to support larger graphs in exchange for efficiency. For example, if a single bit of a bit-vector is assigned to a group of 32 vertices, graphs of size 256 million vertices can be handled. However, in this case, all 32 vertices have to be loaded, even if only one of them is really active.

3.5.3 User programming interface

As mentioned earlier in Section 3.2.3, Gather-Apply-Scatter (GAS) is the common processing model for vertex-centric programs. According to this model, the user defines three following functions in the discussed *User Module*, to specify three conceptual phases of an iterative algorithm. In every iteration, and for every vertex under processing:

Gather: is called whenever a neighbor is accessed and its data is read. Accumulation or finding the maximum value are examples of common operations in this function. For example, in PageRank, the function includes:

$$accum \mathrel{+}= VertexValue[NeighbourID]$$

Apply: is called once, after processing the last neighbour, to finalize the calculation and update the vertex value. Division after the accumulation in the average operation can be an example of this. For example, in PageRank, we have:

$$VertexValue[ThisVertexID] = (A + (B * accum))/C$$

Scatter: is called once, after 'Apply' to inform the neighbors about value

update. This function is not used in the pull-based implementation, such as our framework.

Vertex data which are large arrays indexed by vertex ID (stored in global off-chip DDR memory) and any other temporary variable for a vertex in the pipeline (stored on on-chip *Data Tables*) are defined by user in a specific class. The user can implement any complex function, dependent on data, iteration number, etc. The three C/C++ functions are converted to logic, the same as SystemC units. For more relaxed timing in high-latency functions, the function latency can be defined by the user as the number of clock cycles. This value is passed to the underlying SystemC implementation during logic synthesis.

3.6 Experimental Evaluation

3.6.1 Experimental Setup

A. Graph Applications:

We implement and execute some conventional iterative graph algorithms in our template. For comparison purposes, they are also implemented using OpenCL. Some traditional algorithms, such as Breadth-First-Search (BFS) are conceivably parallelizable, while some others like Depth-First-Search (DFS) are inherently difficult to parallelize [56]. The three common parallelizable algorithms chosen for our experiments are listed below. As discussed earlier, due to FPGA hardware, and also Xeon+FPGA platform limitations to handle the complexity of the push-based mechanism, these algorithms are implemented in a pull-based fashion.

- PageRank (PR): is a widely used algorithm in web search engines to rank web-pages, based on reference counts from other pages. The score value assigned to rank each node has a real data type. In our baseline OpenCL implementation, it was not possible to implement this in an efficient pipelining due to

floating-point limitations in OpenCL. Therefore, to have a fair comparison, we tested with a fixed-point version instead.

- Breadth-First-Search (BFS): In the iterative execution of conventional BFS, starting from a randomized root vertex (same in all experiments), all reachable vertices are visited and labeled with their depth value.

- Maximal Independent Set (MIS): Independent set or anti-clique is a set of vertices, where no two are adjacent. Maximal Independent Set (MIS) is the one that is not a subset of any others. MIS is used as part of different applications such as graph coloring. A fully parallelizable algorithm to find one MIS (more than one solution can exist) is chosen [57]. During the execution, vertices are labeled as undecided-yet, in-list, and out-of-list. Implementing the exact form of the cited algorithm requires two different 'gather' functions to run in even/odd iterations.

B. Datasets:

Selected applications are evaluated using graph datasets provided by a widely accepted repository [58]. Datasets include cases from real-world applications like social networks (Com-Orkut, Com-LiveJournal) and those generated by simulation (Adaptive, Delaunay, Hugetric). Graph sizes vary from a few to tens of millions of vertices. As explained earlier, due to memory limitations in the compiler of the baseline OpenCL, we could not experiment on larger graphs. While graph sizes are large enough for small FPGA platform cache, graph vertices are also shuffled properly to remove any possible locality that can affect the experimental results. The profiler tool confirms that, in all graphs, the cache hit-rate of accessing neighbor vertices is minimal.

C. Performance Estimation Metrics:

We have two kinds of performance metrics: throughput (TP), and work-efficiency (WE). In TP measurement, the number of processed edges (irregular access to neighbor vertices) per unit of time is counted. While in WE, task completion

Graph	# of Vertices	# of Edges
Adaptive	6.8M	27M
Com-LiveJournal	4M	69M
Com-Orkut	3.1M	234M
Delaunay	16M	100M
Hugetric	6.6M	20M

Table 3.1: Graph datasets used in experiments.

time is counted. The TP metric represents the efficiency of the pipeline design. However, relying only on the TP metric can be misleading, as WE mode can converge faster using the active-list.

Theoretically, an efficient pipeline finishes one task per clock cycle. In our case, processing one edge (neighbor vertex) per cycle is the primary goal of high-throughput (HT) design, per pipeline (accelerator). Recall that the cache memory is not in effect. Then, only the data of the requested neighbor vertex is used per memory request, with high memory access latency. We tried to achieve this objective in both OpenCL-based and template implementations.

Since the off-chip memory bandwidth is the decisive bottleneck, when it is saturated, no more data can be provided to the pipeline. Beyond this point, performance can not increase any more, and adding additional parallel processing power, in any kind, such as more accelerators, does not improve TP. Due to this, approaching the theoretical maximum memory bandwidth of the system efficiently, with the minimum number of accelerators, is our primary goal to achieve peak TP from the architecture. For this purpose, we increase the number of accelerators until no improvement is seen in the performance. With using profiler tools, bandwidth saturation is measured and the design is tested accordingly. Assuming $\sim$10% of bandwidth is consumed by other sequential graph data, the maximum TP of system, under no locality assumption is:

$$\text{Maximum TP} \approx 0.9 * \text{Memory Bandwidth/Cache-Line Size}$$

For OpenCL and template with the cache-line size of 64B, the available bandwidth is $\sim$14 GB/s and, $\sim$17 GB/s respectively, and maximum throughput is $\sim$200 M-Edge/s and, $\sim$240 M-Edge/s, respectively. Please note that if locality exists and

the cache is enabled, with the exact same setup, throughput can ideally increase up to 16X (cache-line size/size of data type), i.e. ∼4 G-Edge/s. Equivalently, this can be calculated as bandwidth/size of data type (17GBps/4B).

3.6.2 Results

We show some of the results based on baseline HT OpenCL. Thereafter, improvements by mainline SystemC are presented.

3.6.2.1 Baseline OpenCL

First of all, we illustrate the discussed memory bottleneck issue. In Figure 3.7, peak edge throughput and off-chip memory bandwidth usage of different applications (for baseline HT OpenCL on Delaunay dataset) are depicted. As can be seen from this figure, when the maximum available memory bandwidth is saturated the throughput is also saturated. More specifically, in all applications, because of the high-throughput pipeline design (II=1 clock cycle), two accelerators use up the whole available data read bandwidth, while even one accelerator could use more than 75% of it. As mentioned earlier, irregular memory accesses are responsible for around 90% of the bandwidth usage. Since the sequential write consumes comparatively negligible bandwidth, only the read bandwidth is shown.

In Figure 3.8, the average throughput of different graph datasets with different numbers of accelerators is shown for the BFS application. The figure illustrates the effect of graph topology on throughput. According to Table 3.1, the Com-LiveJournal graph is denser than the other three (Com-Orkut is too large for the OpenCL platform). Moreover, as can be seen from Table 3.2, this very same graph converges quickly in BFS application (15 iterations), causing many early wasted clock cycles, due to unnecessary loading of converged vertices. This leads to a comparatively poor average throughput for this graph. Since in this case, performance is limited by the pipeline (not the memory), having more accelerators is potentially effective. Other graphs converge far more slowly, for which average

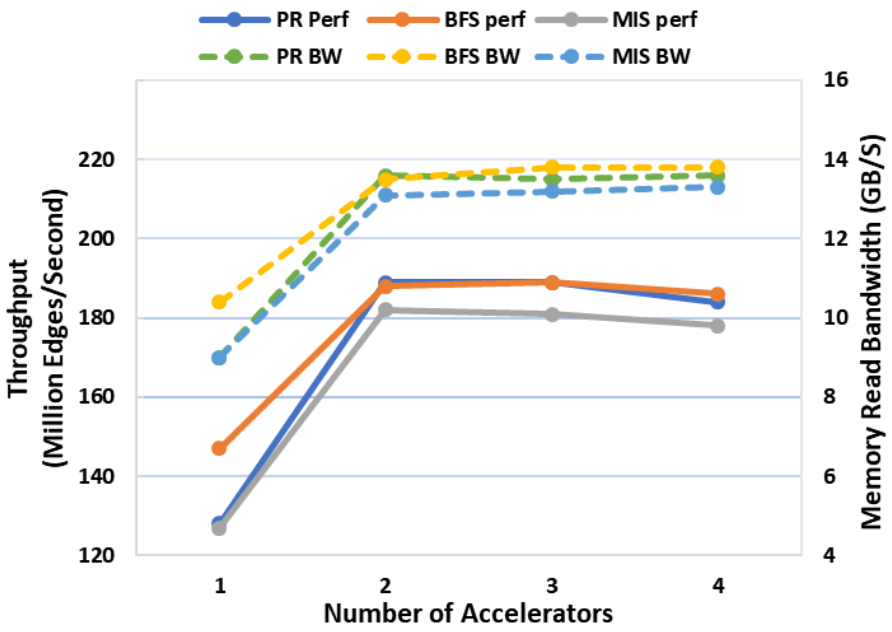

Figure 3.7: Saturation of off-chip memory bandwidth and throughput, with the number of accelerators (Delaunay).

throughput is measured in the first 200 iterations.

Graph	iter. 1	iter. 2	iter. 3	iter. 4
Adaptive	4	12	24	40
Com-LiveJournal	1	171	25,309	307,515
Delaunay	5	14	26	52
Hugetric	3	8	16	27

Table 3.2: The number of vertices converged in the first four iterations of BFS.

Figure 3.9 shows the average throughput of different applications, for a high number of iterations unless early convergence happens (specifically in the case of MIS). As can be seen, the OpenCL-based implementation, in some cases, can achieve rather close to maximum feasible (saturated) throughput. Recall that a small portion of bandwidth is dedicated to sequential data accesses. Since MIS generally converges very quickly, it leads to low pipeline utilization and low average throughput after initial iterations.

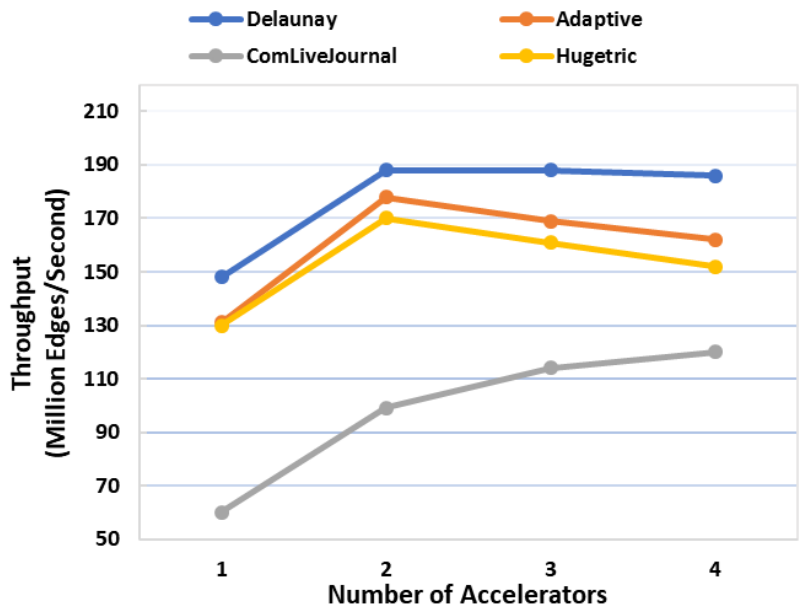

Figure 3.8: Average throughput for the OpenCL implementation of BFS with different datasets and different number of accelerators.

3.6.2.2 Template-based

As explained earlier, the SystemC template has two modes, namely, HT and WE. While the WE is the desired one for users, we use the HT mode for comparison with the baseline OpenCL which has only HT mode. In Figure 3.10, it can be seen that even HT outperforms the automatically generated OpenCL pipeline, by up to 50% in peak throughput. This is mainly due to cycle-accurate and more efficient pipeline design, better memory management (both off-chip DDR and on-chip BRAM), and higher frequency of memory controller which provides more bandwidth (17 vs. 14 GB/s) in template-based SystemC, altogether allowing more parallelism. In Figure 3.11, we have almost similar results, where average performance is depicted by application run-time per iteration, for higher iteration numbers. Results show that template-based implementation provides up to 33% shorter run-time. As expected, larger graphs with higher edge count, have longer run-time.

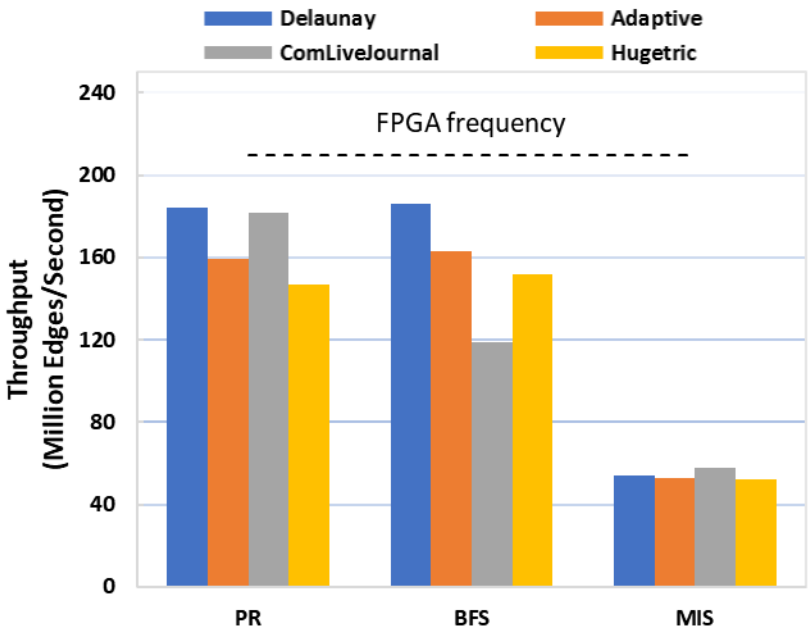

Figure 3.9: Average throughput for the OpenCL implementation with different datasets on 4 accelerators.

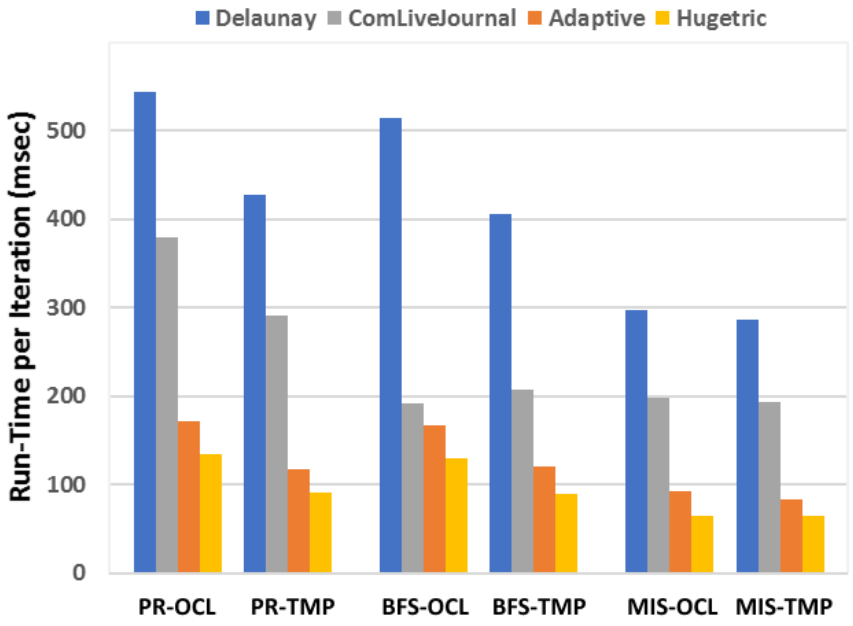

Figure 3.11: Average run-time per iteration for OpenCL vs. template HT.

WE mode provides even better results by only working on active vertices. Vertices that really needs to be processed when the wave of changes reaches to

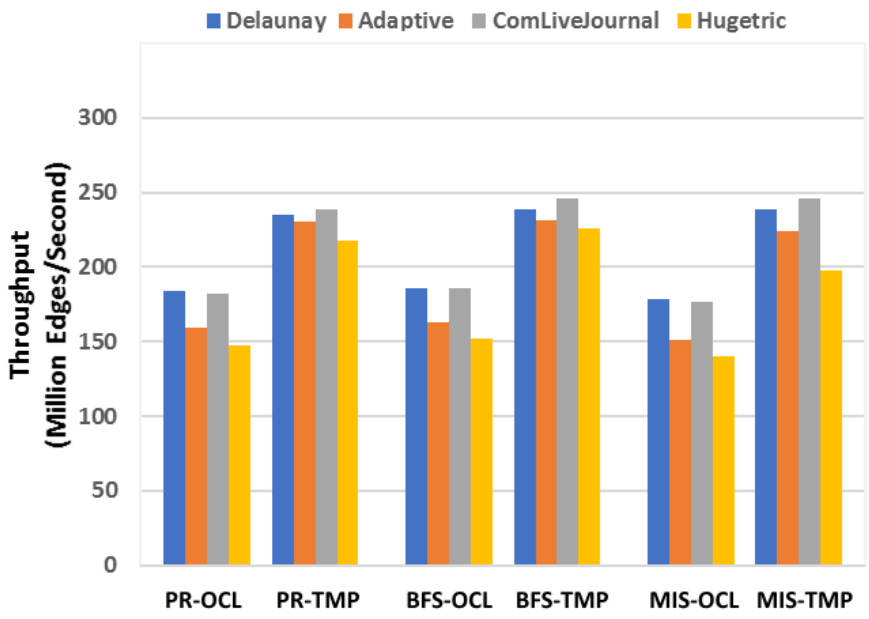

Figure 3.10: Peak throughput of OpenCL vs. Template-based HT.

them are loaded into the pipeline in each iteration. As can be seen from Figure 3.12, MIS application [57] usually converges very quickly, for any dataset (in all of our datasets, <20 iterations). Because of the greedy approach, almost all vertices are processed and converged early, in practice. Then, vertices are most of the time engaged in processing and then included in active-list, until full graph convergence. As a result, the benefits of WE mode is generally limited for MIS, which is a 15% improvement on the average. In contrast, BFS usually converges slowly, especially for sparse graphs. Since the algorithm starts from a specified root and goes one level deeper after each iteration, only frontier vertices are contained in the active-list. Results show around 5X improvement for two rather denser graphs (converge in <30 iterations), and at least 100X for two sparse graphs (for which measurements are done in the first 1000 iterations). On the other hand, PR converges faster than BFS, as all vertices are engaged from the start. But no vertex is decided as converged individually until the whole graph converges. For all datasets, and damper parameter β=0.99, convergence happens in <1000 iterations. Results show that from 2.5X, up to 23X improvement is achieved in WE mode.

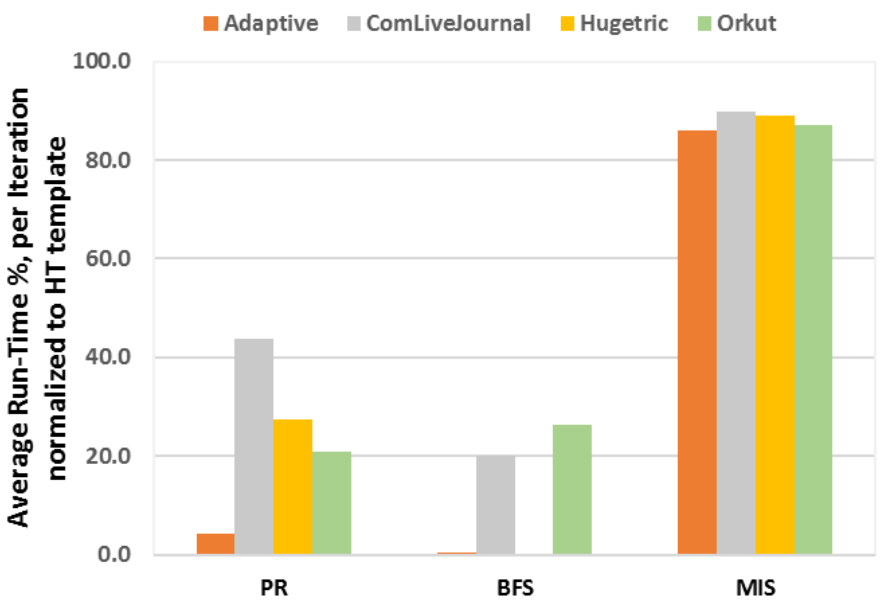

Figure 3.12: Average run-time per iteration of WE template normalized with respect to HT template.

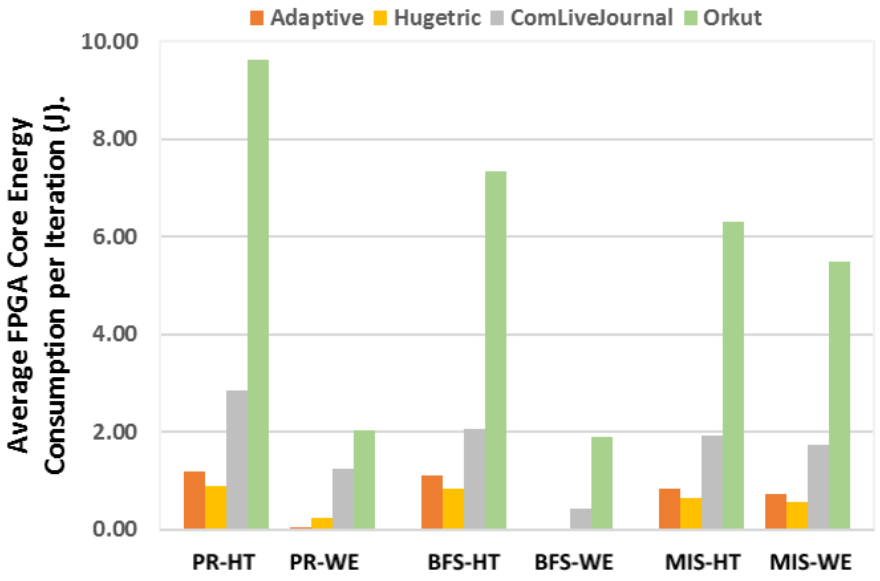

Figure 3.13: Average energy consumption per iteration, for HT vs. WE.

The improvement of total energy consumption of WE mode versus HT mode is shown in Figure 3.13. The measurements are done by reading the current sensors embedded on to the FPGA, which are available to the users on some special

status register addresses, during the run-time. Only the power of FPGA core (not I/O ports, etc.) is considered here. The core voltage is 0.95V. Results are rather similar to run-time improvements since the energy saving is mainly due to shorter run-time, provided by WE mode.

3.6.2.3 Resource utilization and clock rate

Here, we give a summary of low-level implementation details about resource utilization and the clock rates. In Table 3.3, area utilization is summarized, where all applications have close values. While the OpenCL compiler relies more on Block RAM usage, the SystemC and the RTL tools prefer using more logic. The WE mode uses more resources to implement the bit-vector and other extra units. On the other hand, the OpenCL compiler tries to sustain the highest possible clock frequency, where ∼210 MHz is achieved by all of the applications. In OpenCL (not in our template), this frequency directly affects bandwidth and throughput, because the memory controller also works with this clock frequency (210MHzx64B≈14GB/s). On the other hand, in SystemC flow, while bandwidth is fixed (∼17GB/s, read bandwidth), the user can set the frequency. This allows reducing the clock rate as much as needed if performance is already saturated. In Figure 3.10, while OpenCL works with ∼210 MHz, the template works with 100 MHz, because four cores are enough for saturating the available bandwidth.

Design\Resource Type	**Logic**	**Block RAM**
Blue-stream (fixed)	20%	12%
Green-stream (user):		
OpenCL-HT, per core	3%	7%
Template-HT, per core	7%	2%
Template-WE, per core	13%	14%

Table 3.3: FPGA resource utilization.

3.6.2.4 Observations

First, the well-known memory bottleneck problem in graph applications, which throttles the performance, is experimentally illustrated in Figure 3.7. The performance achieved by baseline OpenCL and the effect of different datasets and applications is shown in Figure 3.8-3.9. Note that, theoretically, the maximum achievable throughput, in the worst case without locality is memory bandwidth/cache-line size (Section 3.6.1-C). These Figures depict the reasonable quality of the baseline, which can be close to the maximum throughput. Recall from Section 3.4 that the main disadvantage of OpenCL is not the implementation, but it is the difficulty of generating a high-performance pipeline, which only allows HT mode in our work. The advantage of our high-throughput SystemC implementation is shown in Figure 3.10, which is mainly due to efficient deep pipeline design and higher bandwidth. Through Figure 3.11-3.13, the main advantage of our template, the WE mode is presented. Besides high throughout processing, the active-list significantly reduces the total run-time and power consumption, particularly for sparse graphs, up to 100X. While it is shown that a few accelerators can maximize throughput, the low area allows convenient fitting to FPGA.

3.7 Summary

In this chapter, we designed and implemented a SystemC HLS-based graph processing template with a clock-wise precisely designed deep-pipeline architecture. The template framework is simplified for easy mapping to FPGA, even for software programmers. The template can be customized using a single module in C/C++, combining the high-level programmability with the efficiency of SystemC hardware language. In high-performance mode, the high-throughput pipeline achieves maximum edge throughput with the minimum number of accelerators. In addition, the work-efficient mode significantly reduces total run-time with a novel active-list design. Through experiments on the Intel Xeon+FPGA platform, we showed the benefits of the proposed template compared with respect to

OpenCL based implementation. Based on our results, this template outperforms the OpenCL version considerably, by providing higher edge processing throughput, lower run-time, and lower power consumption.

Chapter 4

Reliable and Reconfigurable SRAM Cell and Flip-Flop

4.1 Introduction

Beside processors, static memory structure is also widely used within all modern Field-Programmable-Gate-Array (FPGA) chips. Although flexibility of reconfigurability is the key benefit of FPGAs, volatility is not free and comes with some costs, such as more silicon area, delay, and power consumption [62]. Another significant shortage is being less reliable when compared to ASICs. This becomes important particularly for applications with high reliability requirements which usually are produced in low quantities, such as avionics, safety critical systems, satellites, etc. Therefore, usage of FPGAs in these domains is significantly limited and due to this, enhancing reliability levels for FPGAs is always under spotlight [63]. Generally, FPGA architectures are island-style where a wide variety of heterogeneous resources, including logic blocks, on-chip block memories, I/O blocks and DSPs are connected via routing switch boxes (Figure 4.1). Configuration memory bits which normally are not directly visible to user, specify how reconfigurable fabric operates. An example for this invisible configuration is

See [59–61] for the original work.

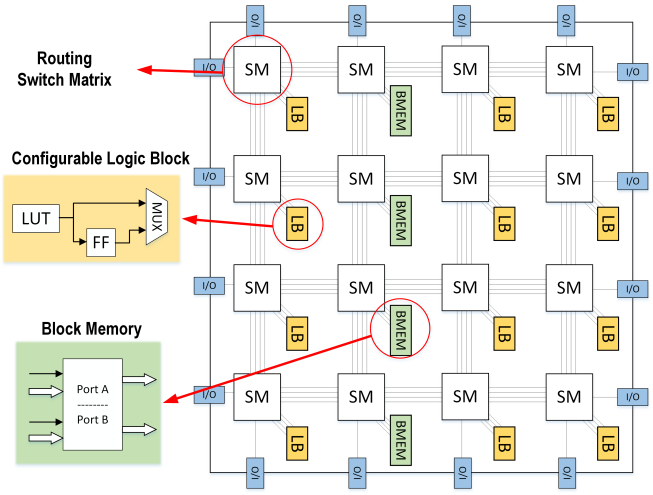

Figure 4.1: Distribution of logic blocks, on-chip memory blocks, routing switch matrices, and I/O blocks inside a generic FPGA architecture.

the operation of switch boxes, where they need to be set up to provide required connectivity among utilized resources. SRAM-based FPGAs preserve configuration bits inside volatile static memory bits (which are loaded to FPGA every time at power up from an off-chip flash memory). Block RAMs (BRAMs), Look-Up Tables (LUTs) and Flip-Flops (FFs) are essential building blocks for user design. BRAMs are small SRAM blocks which tens or hundreds are distributed throughout the FPGA [64, 65]. User design is mapped to several configurable logic blocks, each contains a few read-only Look-Up Tables (LUTs) to implement the combinational logic (function generator), and a set of user flip-flops/latches to implement sequential logic such as registers and state machines. In this setup, user FFs/latches keep the state of user circuit [66]. All these resources are based on static memory structure.

In this chapter, we propose Joint SRAM (JSRAM) cell, Joint Latch (JLatch) and Joint Flip-Flop (JFF), all based on a novel hardening solution for static memory structures [59–61]. Specifically, we implement a configurable static storage element that takes advantage of the trade-off between resource count and fault resilience at a tunable granularity level. Joint storage bit is a simple low-overhead circuit idea which joins together four selected bit cells to make a single but highly robust cell when reliability becomes a concern (e.g. when applications

require high reliability or when the voltage is decreased). The idea is analogous to multiple modular redundancy techniques like Double or Triple Modular Redundancy (DMR, TMR). The main contributions of this study are as follows:

- We implement a novel static memory cell to mitigate transient faults. In the case of SRAM, we improve both soft-error rate and V_{dd}-induced fault (usually happens during memory access) rate at the same time. Existing solutions usually address only one of these. In the cause of latch and FF, soft-error rate is improved.
- Our approach brings reconfigurability of reliability to static memory structures (SRAM, latch, and Flip-Flop) with fine granularity, where it is particularly useful for reconfigurable fabrics like commercial-grade FPGAs to make them more suitable for critical applications.
- The reliable mode provides auto-correction and full immunity against single faults, provided that the standard SRAM transistor sizing is applied. Moreover, it efficiently can cope with Multiple Bit Upsets (MBUs).
- Unlike prevalent hardware redundancy techniques, like TMR, there is no explicit majority voter at output, in this technique. Consequently, voter failure and latency are of less concern. Moreover, joint cell immediately recovers the faulty bit value to avoid degradation in redundancy.
- Based on a similar idea and as a typical application, we propose a novel approach to exploit vacant spaces in Register-File (RF) to keep redundant data to achieve more robustness in RF.

4.2 Background

In this section, we present the metrics used in stability measurement and nomenclature of soft errors in SRAMs.

4.2.1 Static Noise Margin and Bit Error Rate

Alongside area and performance, stability is another important factor in SRAM cell design. Static Noise Margin (SNM) is the traditional gauge for measuring stability in SRAM cell. By definition, in hold state or during read operation, SNM is defined as maximum tolerable simultaneous DC noise on inputs of both inverters. Any stronger noise flips cell value. The graphical maximum square technique and its formulation is the prevalent way of calculating SNM in circuit simulators [67]. Main adverse factors which degrade SNM are process variations and voltage scaling [4, 68].

SNM at read time (Read-SNM) is considered as worst case SNM and it is the main parameter considered for stability improvement. On the other hand, SNM at hold time (Hold-SNM) is less critical [68]. Writability of SRAM cell is measured by Write SNM (WSNM), which is calculated in a different way. By definition, WSNM is the maximum acceptable voltage for '0' value on bit line to push and flip the state of cell in write operation [69].

Bit Error Rate (BER) is a metric indicating SRAM failure rate. Static noise margins, particularly in read operation, highly influence this metric [70].

4.2.2 SEU, MBU, and Soft Error Rate

A single event upset (SEU) is an unwanted upset in electronic circuits caused by high energy particle strikes and leads to a change in data. These non-destructive and non-persistent errors are also called soft errors [69,71]. Not only circuits work above ground level (like satellites) are exposed to, but also they have become a real concern in terrestrial devices. Cosmic radiations and impurities inside chip package are two main sources of SEU [72, 73].

By technology feature size shrinking, node capacitance or the charge stored at each transistor node also shrinks. Therefore, the minimum charge of hitting particle that can upset the cell value (critical charge [74]) is also decreased. This

tends to increase sensitivity against single events. If compared to past generations of technology, particles with lower energy are able to cause an SEU [72]. On the other hand, smaller dimensions mean smaller area being exposed to radiation and, thereby less upset. It has been shown that these opposing factors almost cancel each other and soft error rate (SER) per bit does not change significantly, or even decreases, in newer technologies [69, 73]. However, dense integration achieved by silicon geometry scaling means more cells to fit in a unit area and, thereby, more SER per system. One key factor for increasing importance of soft errors in smaller technologies is that, in smaller dimensions, systems are more susceptible to multiple bit upsets (MBU) by a single strike [72].

4.2.3 Fault sources in SRAMs

Although widely used, static cells are prone to myriad types of faults such as those caused by voltage scaling, process variation [68], noise during read/write operation [4], or soft errors [71]. Thus, improving the stability, which in SRAM cells conventionally is measured by static noise margin (SNM) or soft error rate, is always under the spotlight.

Due to power consumption concerns in today's electronic devices, a variety of power saving techniques like dynamic voltage scaling (DVS) and dynamic frequency scaling (DFS) are applied. Main goal in such techniques is avoiding excessive heating or improving the battery lifetime, especially in ubiquitous mobile devices. Quadratic relation between power and voltage makes supply voltage scaling decisively effective in power reduction by trading off performance when low-power is preferred over high-performance. However, voltage reduction; by silicon geometry scaling or DVS; causes reduction in static noise margin (SNM). Reducing SNM directly affects stability, which means increased susceptibility to noise as presented in Figure 4.2(a) [4]. In Figure 4.2, input/output diagrams of two inverters intersect at two stable points which are two possible values that a cell can hold. Size of largest square fitted between two functions expresses noise

margin. Therefore, SRAM cells become less stable and more vulnerable to adverse disturbing factors in lower V_{dd} voltages. Fault rate, which in memories is measured by Bit Error Rate (BER) metric, increases severely when the voltage is scaled down This effect prevents further opportunities for voltage reduction and consequently hinders additional power savings. Hence, providing stability solutions to increase SNM while operating at lower voltages in SRAM memories has attracted a lot of attention in recent years in order to provide substantial power savings [4, 75].

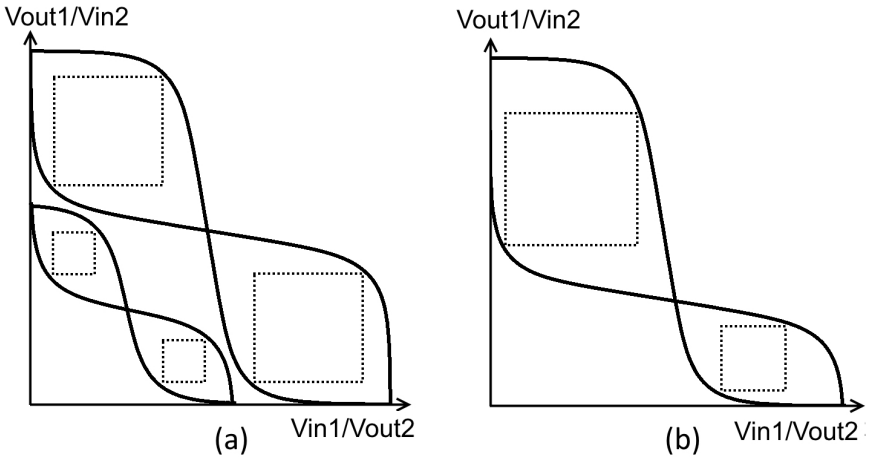

Figure 4.2: SNM degradation due to (a) V_{dd} scaling (b) process variation [4].

A different source of reliability concern in SRAM cells is random manufacturing process variation which raises instabilities in smaller dimensions. In the ideal scenario, transistor parameters should be the precise nominal values and SRAM cell should have a symmetric SNM butterfly diagram (like the ones in Figure 4.2(a)). However, random manufacturing variations lead to fluctuations in parameters around nominal values, like transistor threshold voltage (V_{th}). This process variation contributes to asymmetric SNM distribution, and consequently less stable cells [21] as shown in Figure 4.2(b). Manufacturing deficiencies are more accentuated in lower V_{dd} levels and put limitation on minimum reachable reliable V_{dd} voltage. Due to the severity of variation, some of the manufactured cells are permanently faulty, and thereby causing production yield loss.

Even in the absence of process variation or voltage scaling, reliability may

decrease because of adverse extraneous factors including noise and high energy particle strikes. Particles may originate from impurities inside chip package or from cosmic radiations. Thus, they are inevitable, especially in high altitudes. Such particle strikes can corrupt data stored in memory and induce so-called soft errors. Moreover, the likelihood of a single particle strike causing Multiple-Bit Upsets (MBU) has increased for recent technologies [72, 73]. This recent change blunts the effectiveness of some prominent previous solutions such as DICE [76] and reinforces the call for new approaches for soft error mitigation [71,77]. Due to increasing importance of soft errors, decreasing the sensitivity against radiation is desirable for both terrestrial and space applications [77].

4.3 Related works and discussion

There have been many efforts devoted to improve reliability and stability of SRAMs and static storage elements since they are widely used in processors and FPGAs. Solutions can be classified into some categories; such as device (process) level, circuit design level, architectural level and system level [77, 78]. Device level techniques are fundamental ones applied in manufacturing process such as error-aware transistor positioning [79]. In Radiation Hardening by Process (RHBP) hardening is achieved by none-conventional and expensive foundry processes with using special materials. On the other hand, circuit design level techniques usually rely on using extra transistors or gates and circuit level redundancy strategies (i.e., comes with area penalty) for hardening against soft errors (Radiation Hardening by Design, RHBD), or increasing SNM for making more stable cells. In such designs, considerable overheads of area, power and performance may be tolerated to achieve necessary reliability which make them improper for commercial applications.

Among the studies on hardening by design, DICE [76] is one of the most important techniques. Although many of the efforts were designed based on (or compared to) DICE in recent years like [80, 81], today it has been shown that this idea is no longer as effective as before [82]. Many of the previous

techniques, like DICE, aimed to tolerate just one upset. However, in today's deep sub-micron technology, the probability of having multiple bit upsets (MBU) is increasingly intensified [72]. Although our solution is also inspired by DICE when joining SRAM cells together, it is completely different in multiple ways. First, DICE is a fixed structure while JSRAM is configurable in joining four SRAM cells. In normal mode, which is high performance and less reliable, area and power overhead of DICE is almost double the JSRAM. Second, DICE uses double transistors which provide single point of failure immunity, however it is still vulnerable to MBU [82]. On the other hand, JSRAM is capable of coping with MBU while it is immune against single faults too. Third, besides soft error, JSRAM concurrently addresses cell stability by increasing SNM. Consequently, it decreases BER and provides more opportunity for voltage and power reduction. Fourth, DICE is only based on feedback loops [76], whereas in JSRAM, traditional SRAM cell transistor sizing also plays a role.

As other examples, a series of studies proposed cross-layer hybrid memories [8, 83]. In these techniques, memory is implemented by different types of SRAM cells from big and robust cells to small and weaker cells. For example, in [8], 6T cells with six different sizes are used, whereas in [83], 6T and 10T cells are used to implement cache. Considering the fact that, in low power mode, less performance and capacity is required, weak part of the cache is turned off gradually to preserve reliability. However, above architectures unnecessarily use big and power hungry cells in high-performance mode, which makes them only appropriate for applications spending most of their time in low power mode [83]. In 7T/14T [84], however, it is suggested to combine two cells similar to JSRAM, either to achieve more reliability or performance. While JSRAM provides slightly better SNM than 7T/14T (i.e., less BER), it offers three additional features: 1) providing full immunity against single faults by a novel self-correcting technique 2) achieving more resistance against radiation (i.e., much better SER reduction). 3) capability to cope with MBU by joining far enough cells. However, 7T/14T focused on performance improvement by joining adjacent cells.

A variety of architecture or system level solutions have been proposed. In the case of radiation, they are also called Radiation Hardening by Architecture

(RHBA) techniques [78]. While area and performance penalty in these techniques can be excessive and non-negligible, sometimes they are the only solutions available. We survey some of these solutions in the next paragraphs.

Dense and regular pattern of SRAM memories, provides possibility of using Error Correcting Code (ECC). But generally the reliability gain achieved by devoting more bits to ECC grows slowly, thereby limiting the effectiveness of ECC against multiple errors in highly critical applications like spacecraft and satellite electronics. Particularly, this is true in the case of multiple bit upsets (MBU) which are prevalent in today's extremely dense circuits [85]. Simple ECC like parity or Hamming code, sometimes are used in innovative ways to be more effective against multiple faults, like 2D ECC [86] and bit interleaving [85]. Another drawback of ECC is that the faulty bit stays uncorrected at least for a while until the next write operation which overwrites errors. This can be problematic in predominant read-only memories where the faults are accumulated and become irreversible. Moreover, ECC is inappropriate for time-critical components due to its encode/decode time and power. For instance, while ECC is effectively employed in cache memories, due to unbearable timing and power overheads, it is inappropriate for register file (RF) [9]. More specifically, RF has higher activity rate and it is in processor's critical path. ECCs are also employed in FPGAs such as Xilinx 7 Series where, on-chip BRAM memories are also equipped with single-bit correction, double-bit detection (SECDED) ECC [63].

Replication techniques like Triple Modular Redundancy (TMR) are another important class of solutions. Data replication in vacant part of RF has been the focus of some studies [61]. While effective and simple, area overhead may be significant in such approaches. Flexicache [87] proposes flexibility in cache replication to achieve more reliability in low operating voltage level, while keeping cache capacity intact in higher voltage levels. The drawback in this solution is abrupt transition of cache from full size to half and then to one third. In Flexicache, all cache lines have to be replicated in reliable mode, but in JSRAM, some parts of memory can be individually selected for replication. In Flexicache, similar to many other replication schemes, majority voters are used to decide the correct data value. However, these majority voters can potentially become single

point of failure. Bit-fix [88] combines two physically consecutive cache lines in register-transfer level (RTL) to make one logical line. If faulty bits in two lines do not fall into same index, this technique creates one healthy line from two partially faulty lines. Positions of faulty bits are stored in a quarter of cache ways. Bit-fix only targets permanent faults aimed to reach lower supply voltage and improve yield. Since our solution combines cells (or lines) too, it is in a sense similar to bit-fix idea but at the circuit level. JSRAM makes unstable cells more robust, rather than filtering them out. It also mitigates soft errors and imposes no limitation on higher layer solutions like bit-fix or ECC. Therefore, it is orthogonal to these architecture level techniques.

In the context of FPGAs, configuration memory bits, internal block memories, and the state of user circuit inside sequential logic are typical vulnerable parts of an FPGA. Any fault in configuration results in corruption of FPGA operation (corruption of routings or combinational functions) which mandates correction, for example by reloading the configuration. In the case of sequential logic, however, fault leads to a change in the state of circuit where a circuit restart may become necessary. Some reliability solutions are specifically applicable to configuration memories, such as storing them on on-chip none-volatile flash memory bits as in the flash-based FPGAs; or scrubbing which means reloading the configuration memory periodically or on demand to overwrite occasional errors [89]. In the case of internal block memories, ECC is a common solution as it can be effectively employed within any regular memory structure. Because user FFs are individual storage cells with changing value, above solutions are not applicable to them. Studies about protecting FFs are mainly based on redundancy techniques such as TMR. Some radiation-hardened FPGAs use three FFs plus a dedicated voter to implement TMR to have a single hardened FF [90]. Xilinx V5QV FPGA [91] is a modified version of commercial-grade Virtex-5 FPGA that incorporates RHBD technology to protect FPGA against SEUs and achieve space qualification. As one can expect, area overhead in RHBD techniques is significant. Xilinx's TMRTOOL [92], is a tool for automatic conversion of user design into TMR version for protection against radiation in commercial-grade FPGAs. The tool automatically triplicates inputs, logics, and outputs and also inserts majority voters with

feedback loops to correct failures without programmer intervention.

As explained before, in the case of register file (RF), story is a bit different. As has been discussed, ECC is not an efficient technique for RF reliability improvement. Most of studies for RF aim at utilizing information redundancy techniques. In some studies, register duplication is proposed. For example, in [9] by means of register renaming unit, unused registers are detected and exploited to preserve redundant copies of other registers. In-Register Duplication (IRD) is proposed in [93, 94] in which, by an opportunistic idea, dummy sign bits of narrow-width register values are replaced with replication of meaningful bits during RF write operation. For a 64-bit RF, based on distribution of length of operands, extracted from benchmarks, registers are divided into three classes. Those by length of less than 32, between 32 and 34, and more than 34. For the first two classes which have dummy sign bits, IRD is applied. ALU detects such sign bits and replaces them with meaningful bits of data. Later, in read operation, replicated and original bits are bitwise compared to find mismatch as error indication. Additionally, two parity bits are embedded for each half. By means of both error detection mechanisms, which together are similar to a 2D parity system, they added error detection/recovery for narrow width values stored in RF. Extra circuit for detecting effective length, applying replication and comparison are all collected in the execution stage with ALU. Duplication is done on the output of ALU, whereas communication path from ALU output to RF input is also protected against transient faults. Nevertheless, long operands are not protected by IRD. If applied to 32-bit RF, this disadvantage is more serious, because long operands are frequent. In [95], authors proposed an extension to previous work [93]. In addition to short operands, long operands are also protected by this approach. In 32-bit RF, for long operands, values are replicated in other unused registers, similar to [9]. For avoiding negative effect on performance, two stages are added to pipeline for detecting efficient length first, and later performing sign extension in read operation. All of the above-mentioned works are architectural level ideas based on information redundancy and explicit comparison operation. Our solution is rather similar to some of them, however, with a difference where our duplication idea is based on a circuit techinque, very similar to JSRAM.

4.4 JSRAM structure

4.4.1 Circuit Description

The standard single-port six-transistor (6T) SRAM cell shown earlier in Figure 1.1, has two internal nodes that hold strong opposite values by feedback loops, '0'-'1' or '1'-'0'. JSRAM (Joint SRAM) is constructed by connecting four cells like a ring by means of four joiner switches as depicted in Figure 4.3(bit-lines and access transistors are not shown). Each joiner connects two internal nodes of two distinct cells to each other, here PA to PB, $\overline{PB}$ to $\overline{PC}$, PC to PD and $\overline{PD}$ to $\overline{PA}$. Provided that the joiner type and size is appropriately selected, this structure has two stable states: all four cells store '1' (PA=PB=PC=PD='1') or all four cells store '0' (PA=PB=PC=PD='0'). Any other state immediately goes to one of these two stable states. The logic behind this idea is that the coupled nodes which hold same value, electrically support each other if any instability threatens either one of them. Each cell receives a support from two adjacent ones. For example, cell A is supported by cell B and cell D.

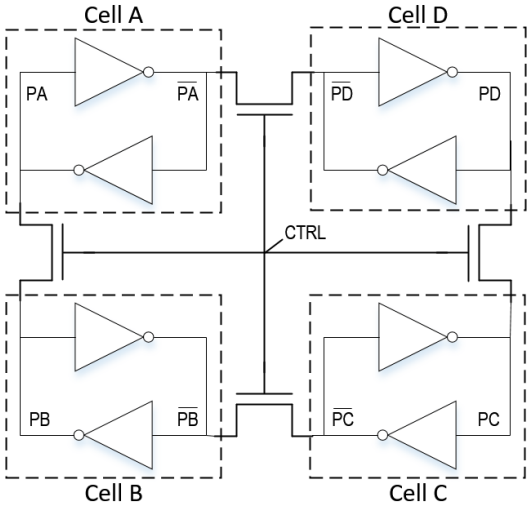

Figure 4.3: JSRAM is a robust cell implemented by combining four SRAM cells like a ring.

In normal mode, switches (joiners) are off. Therefore, cells A, B, C, and D are four separate cells which store independent values without interfering with each

other. When reliability becomes a concern due to reasons stated previously, the joiner switches can be turned on (by making CTRL=1) to join the cells together. In this mode, ring of four bits constructs a single but more robust SRAM bit. All four cells keep identical data values similar to prevalent hardware redundancy techniques like Triple Modular Redundancy (TMR). In TMR, unless the majority voter is also replicated (which imposes large overhead), reliability is still a concern due to the voter. However, in this work, there is no explicit single point of failure gate-level hardware voter.

4.4.2 Joiner Switch

Type of joiner switches, that connect the four cells, significantly depend on the electrical resistance. If the resistance of switch is low, like a zero-ohm ideal switch, then the four cells support each other strongly. However, error propagation happens easily as well. Our simulations show that, in such a scenario, a single and durable fault can corrupt all other copies. On the other hand, if the resistance is too high, the error propagates hardly but cells receive less support from each other too. In this case, a fault may remain in the ring without being corrected by other redundant copies. In our simulations, we investigated NMOS and PMOS transistors with different W/L sizes to find the proper switch. Simulation results show that NMOS has superior performance over PMOS. Except minimum size PMOS (PMOS1) which has high resistance, other sizes are acceptable from following tested sizes: NMOSX and PMOSX, X=1,..,6 (NMOSX is X times wider than access transistor, PMOSX is X times wider than pull-up transistor). Therefore, joiner switches are not very sensitive to manufacturing process variation.

4.4.3 Various Layouts and Applications

Key feature of JSRAM is to provide reconfigurable levels of reliability. For example, four cache lines (or four registers) can be combined vertically upon request to create a single highly reliable cache line (or register) as shown in Figure 4.4(a).

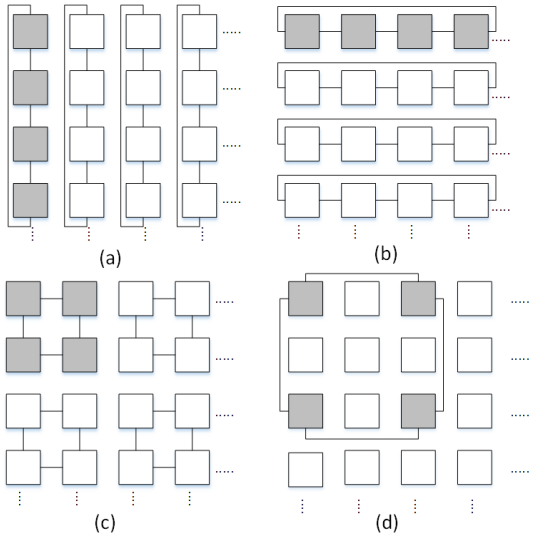

Figure 4.4: Variants of JSRAM layout.

Or, four cache ways of a 4-way cache, can join together horizontally to make a single robust cache way. Four candidate cells do not necessarily have to be adjacent. Joining distant bits has the benefit of better resistance against MBU by observing a safe interleaving distance [96]. However, it obviously requires higher routing overhead as well (Figure 4.4(d)). While applicable to cache and RF, another suitable use for JSRAM is FPGAs, where reconfigurability of reliability particularly matches with the nature of reconfigurable fabrics. Two types of SRAMs constitute a large portion of available resources in modern FPGAs: the on-chip BRAMs and Look-Up-Tables (LUTs).

A BRAM is a few Kbyte SRAM memory block [64] which can be found in modern FPGAs, distributed over the chip in tens or hundreds. For instance, in Xilinx Kintex-7 XC7K480T, 995 BRAM blocks, each with a size of 32Kbit are available [65]. BRAMs have a variety of use such as data buffer, instruction or data memory for on-chip processor (like Xilinx MicroBlaze™) or even being as cache memory for off-chip DDR memory. BRAM blocks can be subdivided or cascaded by FPGA development tool to make smaller or larger memory blocks with customizable width and depth. JSRAM adds the reconfigurability of reliability in the expense of memory size to such existing structures.

Besides BRAM, Look-Up Tables (LUTs) are another main resource with SRAM cells. LUTs are tiny tables which exist in thousands in FPGAs. They are encapsulated in standard logic blocks (in Xilinx terminology are called Configurable Logic Block or CLB). LUTs routinely are employed as function generators to implement combinational logic [66]. Function values are stored inside LUT and accessed like a read-only memory (ROM). Additionally, in a less frequent application, LUTs from different logic blocks can join together to create a so-called *distributed memory* with similar goals like BRAMs. Since LUTs are mostly used in a read-only fashion to implement combinational logic, accumulated bit errors can potentially defeat ECC in a high-reliable application. This is because there is no occasional write to overwrite faulty bits. However, automatic error correction mechanism in JSRAM can prevent such errors.

JSRAM can be applied with finer granularity on some part of memory address space. Therefore, part of memory that keeps critical data becomes extremely reliable. For instance, processor's instruction memory can be a good candidate. In the context of FPGAs, internal BRAMs can be cascaded to make a large instruction memory for on-chip soft processor. Highly critical parts of FPGA such as the main processor (which may be responsible for diagnostics of other parts of system) may operate on the hardened LUTs.

4.4.4 Static Noise Margin Improvement

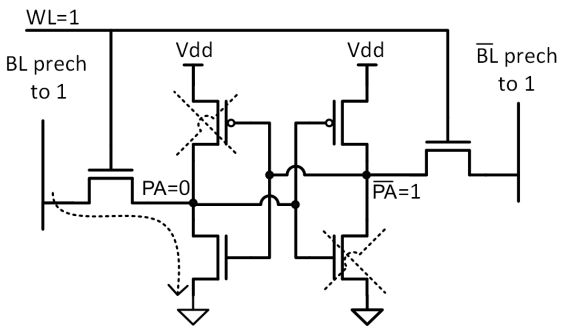

Figure 4.5: Standard read operation in SRAM.

Bit Error Rate (BER) is a metric for measuring failure rate in communication

and in memories. In memories, such failures frequently occur during read/write operations. The value of Static Noise Margin (SNM)—the maximum tolerable noise inside feedback loop of SRAM bitcell—highly influences BER [97]. Degradation of SNM during read access (Read-SNM) determines a critical limitation on stability of SRAM [68]. During read operation both bitlines (BLs) are precharged to V_{dd} and the word-line (WL) becomes '1' to connect bitlines to internal nodes. Drop of voltage on bitlines is detected by sense amplifiers and cell value is read. In practice, read operation is performed only on half of the cell that keeps '0' value (Figure 4.5). The path formed by access and pull-down transistors from V_{dd} to ground and voltage division on elements on this path pulls the voltage of internal node upward from ideal zero (see dotted line in Figure 4.5). Such undesirable increase in node voltage, decreases the maximum tolerable noise on it. This phenomenon makes Read-SNM a critical stability metric. One common technique to improve Read-SNM is to increase β ratio (or cell ratio), the ratio of pull-down transistor over access transistor. However, this increases cell area.

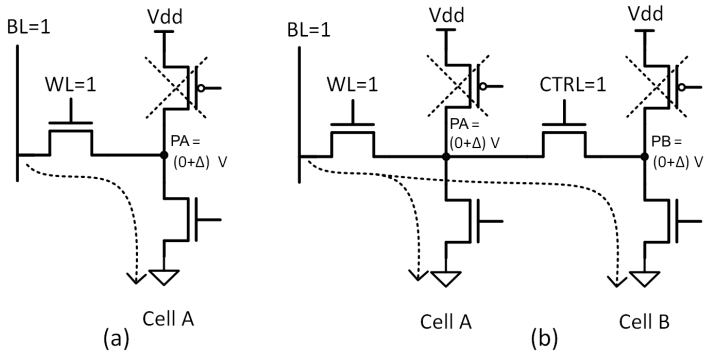

Figure 4.6: a) Voltage division on V_{dd}-gnd path in normal read operation increases '0' voltage in node PA. b) Parallelizing pull-down transistors of two cells increases β ratio of JSRAM.

JSRAM technique provides a smart way to indirectly increase the β ratio to improve Read-SNM. Let us assume internal node PA stores '0' and a read operation is ongoing on cell A. As Figure 4.6 illustrates, by turning on the joiners, a different path from PA to ground is formed via pull-down of PB node, besides the PA node pull-down NMOS. This parallelization is equivalent to decreasing the resistance between node PA and ground or equivalently increasing size of pull-down transistor of node PA. In addition, other three cells that surround

cell A tend to hold their static value creating a strong resistance against a value change in cell A. Our simulations show that JSRAM has hysteresis behavior which enhances SNM greatly.

As will be shown, although JSRAM has self error correction for single faults, the improved SNM is still important. This is mainly due to the fact that while the error will be corrected immediately, it causes the current read operation to fail and return an incorrect value.

4.4.5 Soft-Error Rate Improvement

Collision of retroactive particles with silicon, injects electrical charge into circuit nodes. Effect of collected charge is a short current pulse [74]. If the collected charge is more than the so-called critical charge (Q_{crit}), then bit value flips. Q_{crit} is a measure of soft-error sensitivity of nodes. Node capacitance and its charging/discharging time constants (τ) determine value of Q_{crit}. Time constants specify how quickly the node's capacitor is charged or discharged. The traditional method for measuring Q_{crit} in simulation, is through injection of current pulse into circuit nodes.

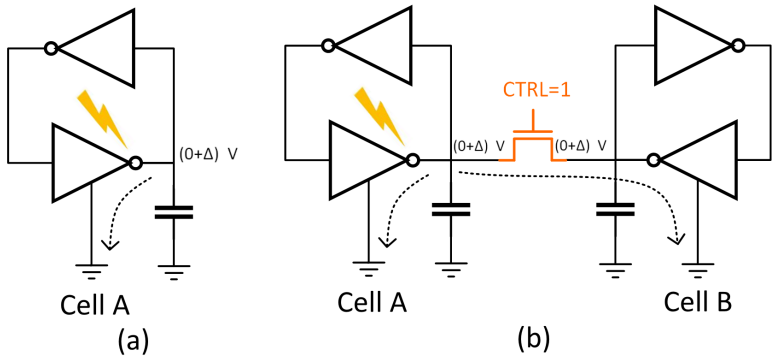

Figure 4.7: a) Hitting particle charges node capacitor and then, node voltage rises. b) Node capacitor and discharging path of cell B is added to cell A to discharge faster.

By hitting particle, node capacitor starts charging. The 'ON' transistor tries to discharge it in the opposite direction (Figure 4.7(a)). If charging rate is larger,

node voltage may reach to cell toggle point. While we only show the scenario for a node with value '0', same applies to a node with value '1'. Larger node capacitance and smaller resistance in pull-down transistor, both contribute to the stability of cell. But as technology shrinks, node capacitance also decreases.

In JSRAM, when joiners are active, capacitance and discharging path of another node is added to the node suffering from a radiation hit (Figure 4.7(b)), whereby capacitance is enhanced and pull-down resistance is decreased. Therefore, a stronger particle is needed to flip the cell A. Additionally, if just cell A flips, it will be immediately recovered by automatic self error correction mechanism explained before. Simulation shows that nodes of JSRAM keeping '1' value (let us assume $\overline{PA}$, $\overline{PB}$, $\overline{PC}$, and $\overline{PD}$) are fully immune. No particle strike can flip the whole JSRAM (all four bit values) by hitting '1' node. In the case of '0' nodes (let us assume PA, PB, PC, and PD), although not fully immune, the increase in Q_{crit} is still drastic. It is around three orders of magnitude (1000X). Therefore, considering exponential relation between critical charge and SER [98], we can practically say JSRAM is fully immune against SEU.

4.4.6 Auto-Correction and Fault Immunity

Although JSRAM structure shows enhanced robustness to assist lower operating voltages, having more stable accesses, or working in harsh environmental condition; faults are still inevitable. An energetic particle or a noisy read may toggle bits of SRAM. In this case, JSRAM proposes two solutions.

First, to avoid or to decrease the corruption probability of more than one bit inside JSRAM at the same time (two or more of the four cells) by a single particle strike (MBU effect), four cells can be *interleaved* far enough to guarantee a safe distance. For a possible layout see Figure 4.4(d). However, more distance implies more wiring overhead.

Second, JSRAM provides full immunity against any single fault (one of four bits). This novel technique is based on conventional transistor sizing of standard

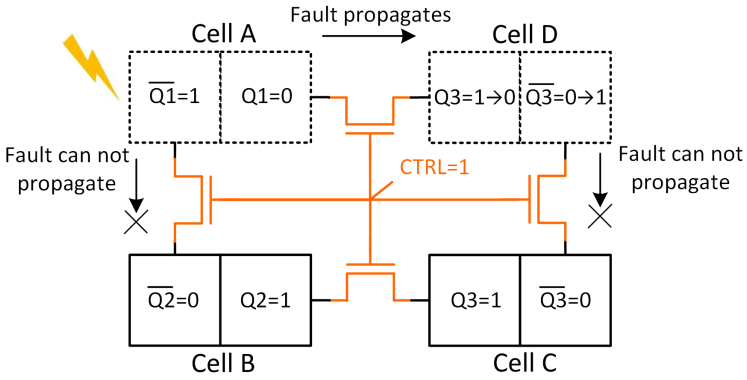

Figure 4.8: Durable fault propagates from cell A to D. Cell B and C are not affected and later will recover cell A and D.

SRAM cell. We assume such sizing holds, otherwise the SRAM does not work correctly. In standard CMOS SRAM design [69], pull-down transistors (which provide '0') are multiple times stronger than pull-ups (which provide '1'). Equivalently, logical '0' is more potent than logical '1'. When bitcells are connected like a ring (Figure 4.8), strong intact '0' values build two barriers around the faulty bit to prevent fault propagation. Hence, the fault can propagate at most to a neighbor cell. In Figure 4.8, fault happened in cell A propagates to cell D, but cells B and C are not affected. After disappearance of the noise, the two potent fault-free '0' values, here in cells B and C, will recover those (at most two) faulty cells. As mentioned previously, selecting proper joiner is necessary and have been investigated as part of this study. As a conclusion, circular connections among four cells, constitute a symmetric and single fault immune structure.

It should be noted that the circuit level JSRAM has no conflict with other conventional higher level techniques. Benefits of solutions like simple parity or in-cache replication can effectively be exploited alongside with JSRAM.

4.4.7 Read/Write Operations, Decoder Modification

When joiners are inactive (CTRL=0), SRAM structure behaves like a typical 6T SRAM. Thus, address decoding and basic read/write operations are performed

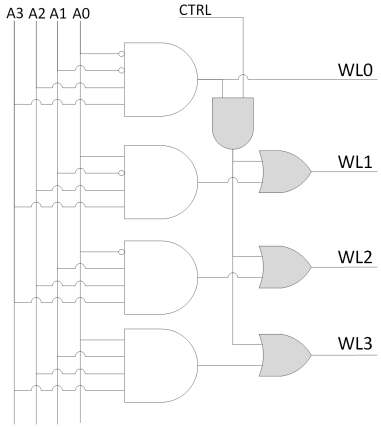

Figure 4.9: Reconfigurable decoder for layout of Figure 4.4(a). Gray gates have been added.

normally. But, when the joiners are active they should be handled differently. Since all four joined cells keep identical redundant value, read can be done arbitrarily through any of the cells. However, read can also be executed on two, three, or four cells concurrently for some other reasons like simpler decoder or having faster read (concurrent read discharges bitline more quickly and provides faster read). But such concurrent read decreases Read-SNM and nullify one of the main benefits of JSRAM since more cells are exposed to noise in concurrent read.

Unlike read, write is performed on all four cells concurrently to be able to change all. If four cells are not located in a row (like Figures 4.4(a), 4.4(c) and 4.4(d)), a modification to decoder is necessary to activate multiple lines simultaneously. A simpler but slower write scenario which can be handled at a higher layer is performing four consecutive normal writes to each cell separately. In the meantime, joiners are disabled and cells are temporarily separated. After four writes, joiners are activated again. Such slow scenario makes sense if write is not frequent as in LUTs where SRAM is used as ROM and write is only needed initially to load the data, or can be done at the background in parallel with other operations. In this case, no modification is necessary for memory decoder.

Depending on JSRAM layout, address decoder has to be modified. Since this is a straightforward logic design, here we only discuss the case that is shown in

Figure 4.4(a), where four lines (addresses) are combined without interleaving to make a single reliable line. Read is possible using any of the four addresses, but write is done on all four. Moreover, decoder should be reconfigurable between normal decoding and reliable decoding. In Figure 4.9, one OR gate is added to decoder output for each line. If CTRL=0, decoder is in normal mode. If CTRL=1, then first address of every four lines (A1A0=00) activates all four lines for concurrent write. Reading from any other three addresses provides a reliable single read. Applications at higher layer should be aware of such mechanisms.

4.5 JLatch and JFF

Structure of a static latch is rather similar to SRAM cell, where two cross-coupled inverters hold stable opposite values by feedback loops, '0'-'1' or '1'-'0'. However, access circuitry and transistor sizing in some implementations are quite different than SRAM cell. Three typical latch implementations are shown in Figure 4.10: (1) SR latch similar to SRAM cell with special transistor sizing, (2) D latch based on SR NAND latch, (3) D latch by pass transistor logic. To store a value into a latch, in (1), the value is pushed inside which requires proper transistor sizing, similar to SRAM cell. But, in (2) and (3), to write a new value, the feedback loop is broken by setting the enable input. JLatch technique is applicable to all cases in Figure 4.10.

Similar to JSRAM, JLatch (Joint Latch) is constructed by connecting outputs of four static latches like a ring (Figure 4.11). Provided that the joiner type and size, and the size of latch transistors are appropriately selected, this structure has two stable states: all four latches store '1' (Qi='1', $\overline{Qi}$='0') or all store '0' (Qi='0', $\overline{Qi}$='1'). Any other state immediately goes to one of these two stable states. In some latch/FF designs a buffer is placed at output. In those cases, the joiner switches should be placed before output buffers to connect internal nodes of feedback loops to each other. As typically master-slave FFs are built from two level-sensitive latches, JFF (Joint Flip-Flop) is also built from two JLatches to form one robust FF from four normal FFs. In Figure 4.12-left, one JDFF (Joint

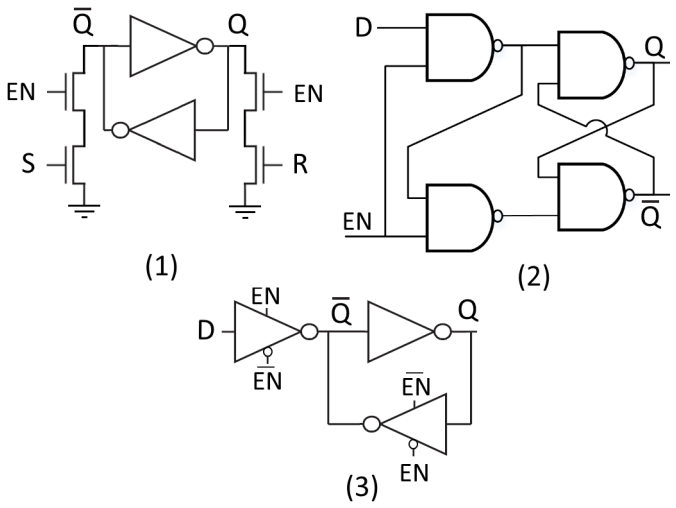

Figure 4.10: Three typical implementations for static latch.

D Flip-Flop) is built from two JDLatches (Joint D latches), where four normal DFFs can be traded-off for having one hardened DFF. This is somehow similar to prevalent hardware redundancy techniques like Triple Modular Redundancy (TMR) as shown in Figure 4.12-right, but without explicit voter. It is good to note that we assume that the latches and FFs (sequential logics) are in hold state. We do not consider Single Event Transients (SETs) inside combinational logic which may generate a glitch and then propagate into sequential logic components. SET is also a research topic in the literature, but it is not relevant to this work.

Transistor sizing can be a bit different in latch than SRAM. Traditionally in standard CMOS gate design, pull-up and pull-down networks are sized unbiasedly to have equal high-to-low and low-to-high worst-case delay. On the other hand, SRAM cells require special transistor sizing for proper functionality as discussed earlier. Then our solution requires a biased transistor sizing for the inverters or gates inside feedback loop of static latch. The latch (1) in Figure 4.10 has already such necessary sizing, but for the other two, size of transistors inside data retention feedback loop should be adjusted accordingly.

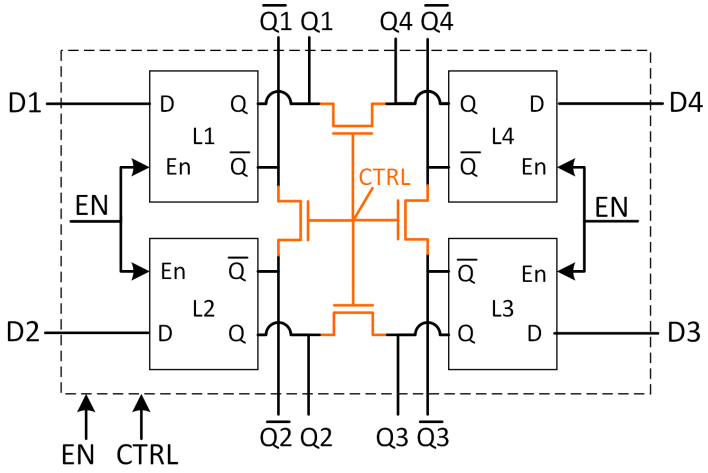

Figure 4.11: JLatch is a reconfigurable hardened static latch implemented by joining outputs of four latches like a ring.

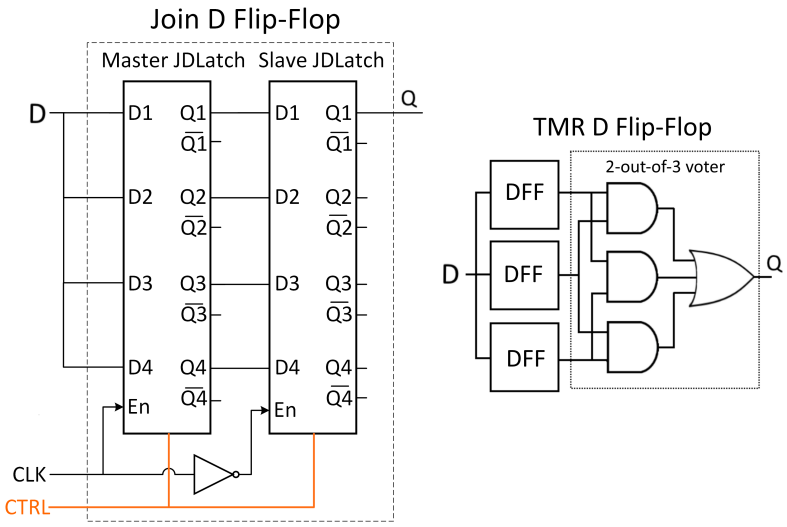

Figure 4.12: JDFF (Joint D Flip-Flop) is built from two JDLatches (Joint D Latches) on the left. This structure provides four separate normal DFF or a single radiation hardened DFF. Conventional TMR technique on the right.

If the four latches to join are in a safe interleaving distance to avoid the MBU effect [96], JLatch can withstand against MBUs effectively as well, but obviously with additional wiring cost. Information about dies of existing FPGAs is commercial secrets and not publicly available. However, considering the size of circuits

inside a typical FPGA logic block which includes multiple LUTs and multiplexers [66], we expect that a few latches or FFs can easily be interleaved (if are not already) among other resources for having such a safe distance.

4.6 Overheads

We expect that timing and area overheads of additional wirings for this technique to be negligible in most of situations. Unlike conventional voters (e.g. the voter in Figure 4.12-right), the extra circuitries are not in series with output. For example, in the case of FPGA, where JLatch/JFF are limited to inside logic blocks, due to excising long routing lines inside FPGA's complex interconnection network, low operational frequency of reconfigurable fabrics, and relatively small size of logic blocks we expect overhead to be negligible. This idea requires one (two) extra transistor(s) per latch (FF) as shown in Figure 4.11. For instance, Xilinx Virtex UltraScale XCVU440 contains more than 20 billion transistors [99] and around 5 million CLB FFs. Therefore, increase in total transistor count is trivial. Moreover, equipping only some FPGA logic blocks with JLatch/JFF to protect most critical parts of design, such as the processor running diagnostics routines, can reduce the costs. As it had been depicted in Figure 4.12, JDFF requires an identical data input for all four FFs. This is done by normal FPGA routing resources, similar to what is done for a typical TMR design.

4.7 Register File Reliability Enhancement

The joining idea is similarly applicable to the processor's Register File (RF). The RF unit is usually implemented using SRAM. But, this leads to a reduction in the RF size (fewer number of usable registers) which is undesirable. In such a case, software code should be generated by a customized reliability-aware compiler. As an example application of the joining technique, we propose a relatively different approach that is transparent to the compiler. As discussed in the related work

section, exploiting vacant spaces in RF is a possible use case. More specifically, redundant data is kept in the dummy extended sign bits of a 2's complement integer number in the processor's integer RF. However, this requires additional read/write operations which does not come for free. As a novel approach, we apply the joining technique to the existing idea in literature for the RF's reliability enhancement. Most importantly, with this approach, the system's higher layers like the operating system or the compiler are oblivious to the existence of a circuit-level mechanism. We refer readers to our earlier work [61] for more details on the joining technique used for RF.

4.8 Experimental Evaluation

4.8.1 Setup

Simulations were performed in Keysight Technologies' Advanced Design System (ADS) with 22nm predictive technology model library [100]. Transistor sizes for typical 22nm SRAM cell have been chosen from literature [101], where SRAM ratio values are: cell ratio = $W_{PD}/W_{PG} = 2.02$ and, pull-up ratio = $W_{PU}/W_{PG} = 1.18$. Transistor sizes for all latches of Figure 4.10 are selected as follows (in all cases $L = L_{min}$): for latch 1, $(W/L)_{PD} = (W/L)_{PU} = (W/L)_{joiner} = 1$ and, $(W/L)_{PG} = 2$. For latch 2, $(W/L)_{PD} = 2$ and $(W/L)_{PU} = 1$ for NAND gate, and $(W/L)_{joiner} = 1$. For latch 3, $(W/L)_{PD} = (W/L)_{PU} = 1$ for inverter, $(W/L)_{PD} = (W/L)_{PU} = 2$ for tri-state inverter and, $(W/L)_{joiner} = 2$.

4.8.2 SNM Improvement

SNM is simulated through a conventional method, where two identical noise sources are inserted into the victim SRAM cell. Then, noise voltage is increased to destabilize the cell until bit value flips [67]. In simulation, bit flipping happens when voltages of two internal nodes of a cell (e.g., nodes PA and $\overline{PA}$) pass

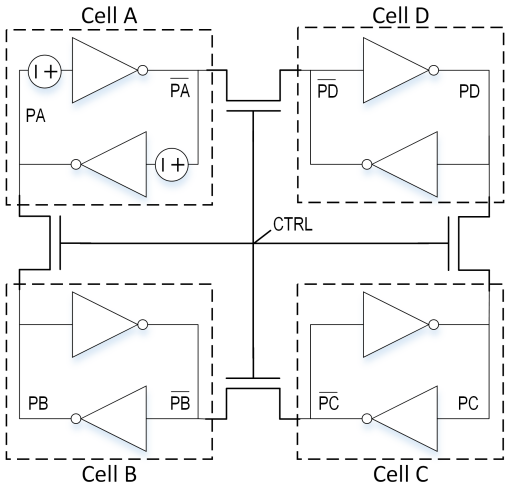

Figure 4.13: JSRAM with identical static noise sources inserted into cell A.

each other. In JSRAM, assuming cell A is suffering from noise, noise sources are inserted inside cell A (see Figure 4.13). We assess different joiners, NMOS and PMOS transistors with two widths: NMOS1, NMOS2, PMOS1, and PMOS2. To understand the maximum feasible SNM improvement by JSRAM, an ideal 0-ohm switch is also simulated as joiner. To experiment on a range of voltages, V_{dd} is swept from 0.3V to 0.8V. Improvements of Read-SNM and Hold-SNM are depicted in Figures 4.14 and 4.15, respectively.

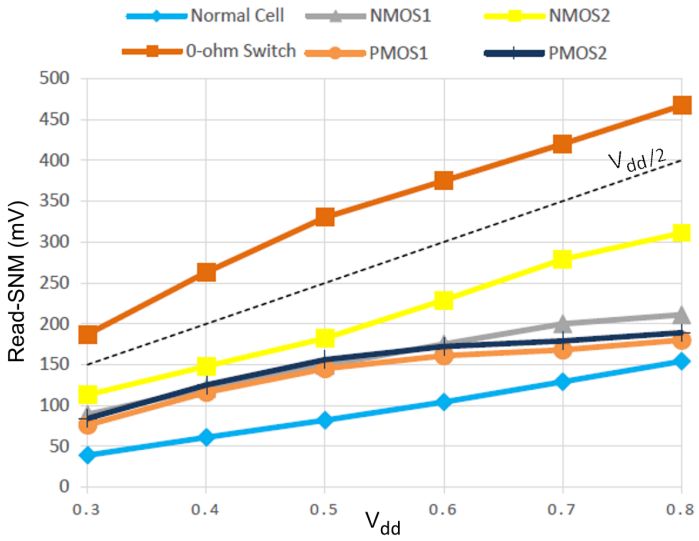

Figure 4.14: Improvement in JSRAM's Read-SNM with different joiner types and sizes over a typical 6T SRAM.

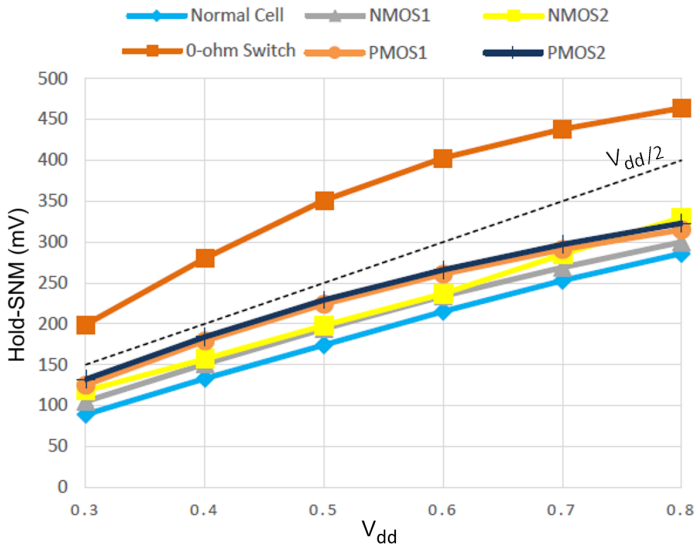

Figure 4.15: Improvement of JSRAM's Hold-SNM with different joiner types and sizes over a typical 6T SRAM.

In the case of Read-SNM, which is a critical stability metric, we see a significant improvement in all joiner types. For example, when NMOS2 is used, 102% and 190% improvement in Read-SNM is achieved for V_{dd}=0.8V and 0.3V, respectively. As one can expect, wider joiner (e.g., NMOS2 over NMOS1) provides a

better connectivity and a better result. However, it requires more area as well. Overall, NMOS shows superiority over PMOS. This is mainly due to the fact that NMOS transistor intrinsically passes '0' value better than PMOS. Moreover, read operation is practically done only on node storing '0'. Less critical Hold-SNM generally has higher value than Read-SNM, whereas improvements over Hold-SNM are less.

Our results show that JSRAM has hysteresis behavior which enhances SNM greatly. Three cells surrounding cell A tend to hold their value, creating a strong resistance against a value change in cell A. This is the reason why SNM values of JSRAM with 0-ohm joiner in Figures 4.14 and 4.15 go beyond $V_{dd}/2$. Conventional butterfly diagram is shown for one case in Figure 4.16. Voltage transfer characteristic (VTC) diagram of each half of cell A in JSRAM is dependent on stored value. Because of this, we ran two DC-simulations for each case (green and red plots in Figure 4.16(b)). The diagonal dotted lines show maximum tolerable noise.

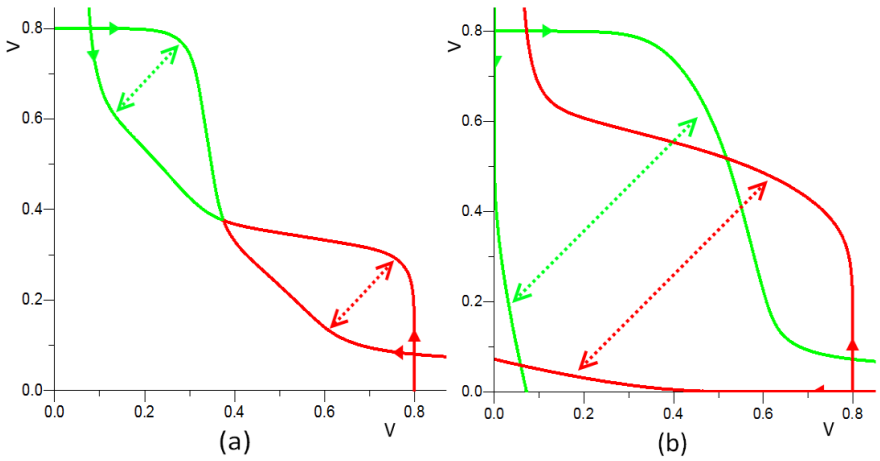

Figure 4.16: Graphical demonstration of Read-SNM (Butterfly diagram). (a) 6T SRAM cell (b) JSRAM with imaginary 0-ohm joiner. Hysteresis behavior causes SNM to go beyond $V_{dd}/2$.

4.8.3 Automatic Error Correction

To illustrate how JSRAM immediately recovers any single faulty bit, a fault is injected into cell A as it was presented in Figure 4.13. Originally we assume that PA=PB=PC=PD='0' and $\overline{PA}$=$\overline{PB}$=$\overline{PC}$=$\overline{PD}$='1'. Noise increases linearly until it passes SNM value and then value of cell A toggles. As depicted in Figure 4.17, fault in cell A propagates to cell D and corrupts it too. Because faulty $\overline{PA}$ is '0' which dominates weaker '1' in $\overline{PD}$. However, since '0' values in nodes PB and PC can not be dominated, cells B and C are not affected. Later, when noise disappears, cells B and C push and recover genuine values of cell A and D in around one nanosecond. Since JSRAM structure is symmetric, same story applies to other cells and inverse cell values.

Our simulation results show that self-correction does not always execute correctly, as in the case of PMOS1 joiner type. Faulty cell may remain uncorrected due to two reasons: 1) it has too much electrical resistance and 2) PMOS type generally does not pass '0' voltage well. Overall, excessive electrical isolation makes PMOS1 an inappropriate choice as joiner circuit. Hence, we conclude that NMOS transistor is a more appropriate implementation for our JSRAM technique.

4.8.4 Soft-Error Rate Improvement

Injecting current pulse into circuit nodes is the conventional method of simulating soft-error effect. Various waveform shapes including exponential, triangular, and rectangular pulse shapes are commonly used for such modeling [102]. Here, a rectangular pulse with variable amplitude and width is generated by current source and injected into victim nodes of cell A (see Figure 4.18). Two current sources shown in the figure either inject a pulse into '0' node (here PA) or out of '1' node ($\overline{PA}$). We run both cases and consider the worst one. For every width, pulse amplitude is increased up to the point such that the whole joint cell toggles. Unlike SNM simulation, where noise inside cell A was not able to corrupt JSRAM,

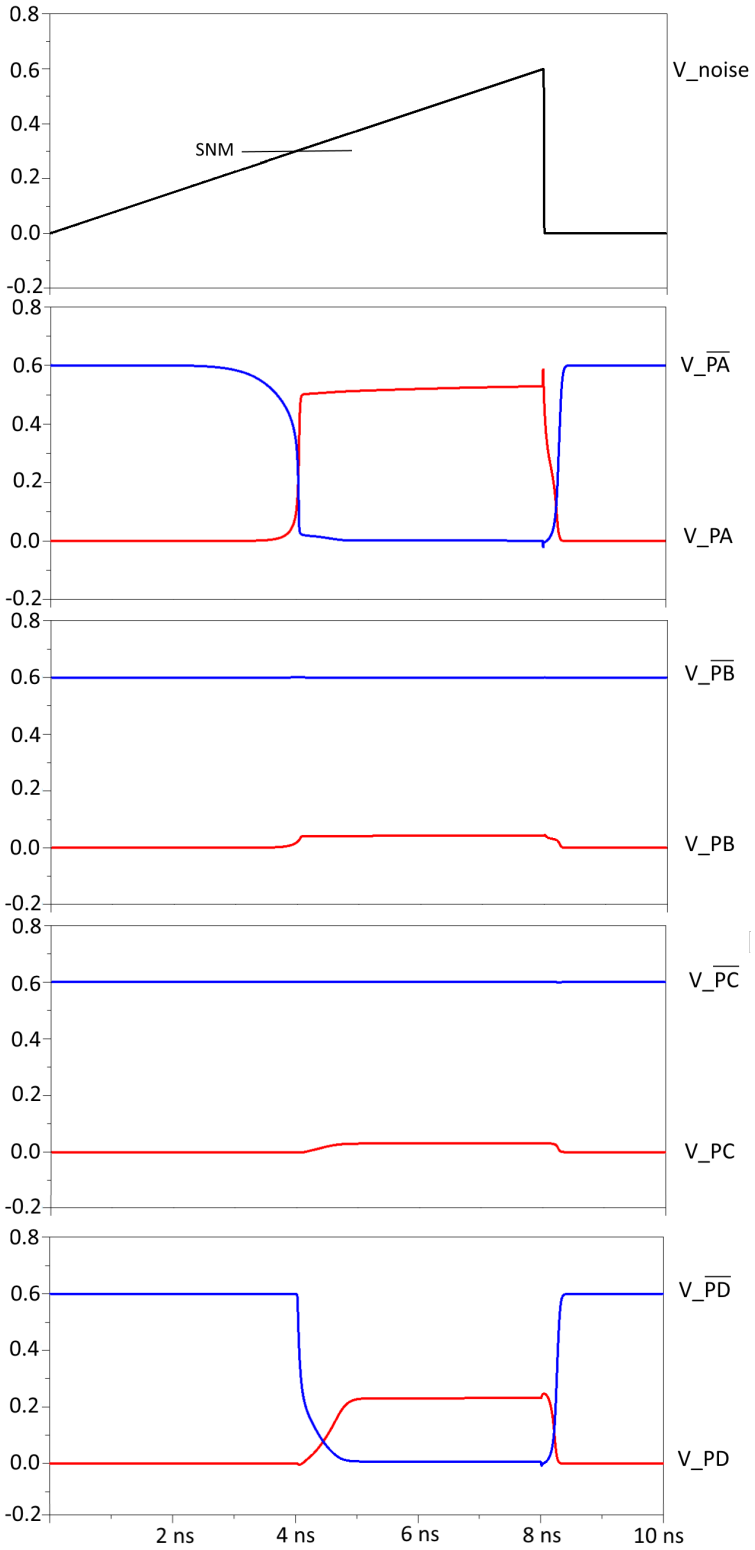

Figure 4.17: Self error correction in JSRAM with NMOS1 joiner.

here it is possible to corrupt the cell. Nevertheless, such possibility is very low, due to significant increase in required Q_{crit}. Since the injected current should flow into two cells via the joiner transistor and affect both cells. For example, if particle hits $\overline{PA}$, it affects $\overline{PD}$ as well; or if particle hits PA, it also affects PB.

Simulations are performed for JSRAM, three latches in Figure 4.10 and, their hardened versions while they are in hold state. In our simulations, we experimented with rectangular pulse widths ranging from 0.1 picoseconds to 1 nanoseconds. Pulse amplitude, on the other hand, is increased up to 10000mA. In the case of JSRAM, simulation results show that nodes keeping '1' (here $\overline{PA}$, $\overline{PB}$, $\overline{PC}$, and $\overline{PD}$) are fully immune against any pulse width and amplitude. However, nodes keeping '0' (here PA, PB, PC, and PD) are vulnerable. Nevertheless, for both JSRAM and JLatch, the required pulse amplitude (or equivalently required Q_{crit}, as it is the area under the pulse shape) is about three orders of magnitude (1000X) larger than what is needed to flip a typical 6T SRAM cell or a typical latch (see Figures 4.19 and 4.20). Considering exponential relation between critical charge and SER as shown in Equation 4.1 [98], we assume such high energy particles are very rare. Therefore we conclude that, for realistic scenarios, JSRAM and JLatch (and then JFF) are fully immune against any single particle strike. As depicted, 7T/14T cell [84], a relevant work discussed in section of related works, is not much better than 6T SRAM cell. The Q_{crit} charge is almost doubled and SER is half [70], when compared to typical 6T cell which does not have a considerable improvement compared to JSRAM (see Figure 4.19).

$$SER = K \times \phi \times A \times \exp\left(-\frac{Q_{crit}}{Q_s}\right) \tag{4.1}$$

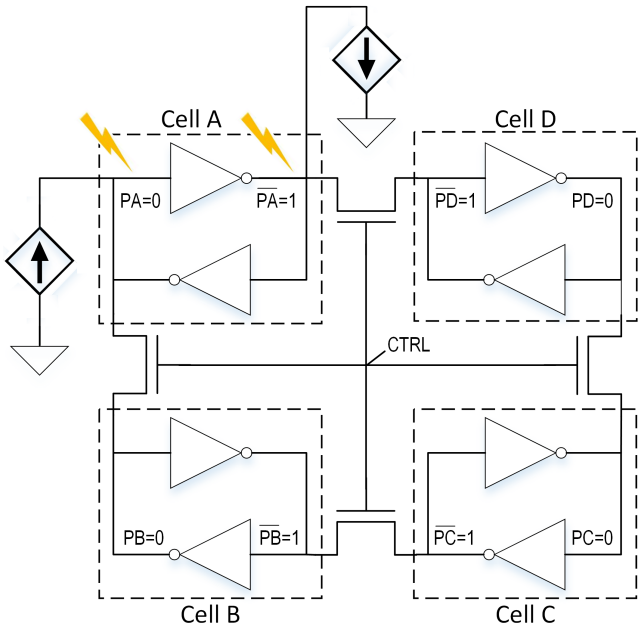

Figure 4.18: Particle strike is modeled by current injection into circuit nodes.

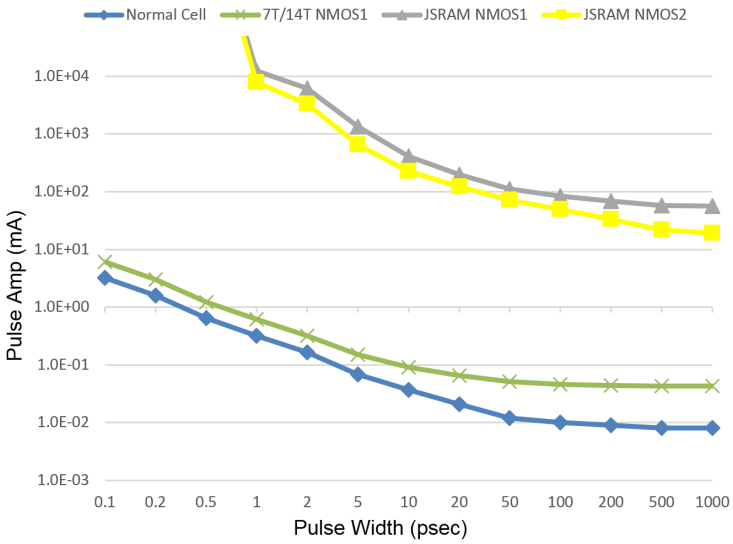

Figure 4.19: JSRAM: maximum tolerable amplitude of injected current pulse, for every pulse width on x-axis.

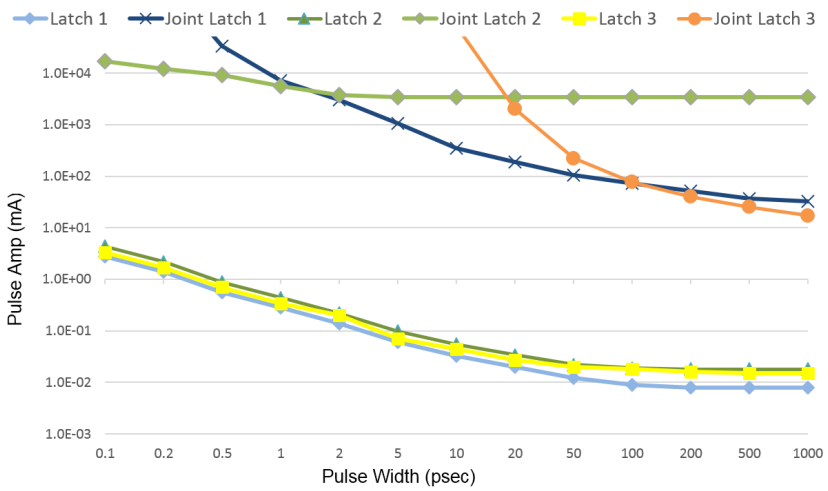

Figure 4.20: JLatch: maximum tolerable amplitude of injected current pulse, for every pulse width on x-axis.

4.8.5 Comparison between JFF and TMR-FF

In the literature, there are some well-known hardening solutions like DICE [76], as discussed earlier. But due to large and fixed overheads, they are only applicable to special and none-commercial products. However, our solution is reconfigurable and applicable to commercial-grade FPGAs. Hence, they are not directly comparable. We compare JLatch/JFF with TMR-DFF which is a well-known technique used in user HDL codes.

In TMR, unless the majority voter is also replicated (which imposes large overheads), reliability is still a concern due to the voter failure which directly affects the output. However, in JFF, no explicit single point of failure hardware voter exists, thereby eliminating such concerns. Moreover, the delay of voter is not added to critical path like conventional majority voters. In Figure 4.21, the delay of normal DFF, TMR DFF, and JDFF (as was shown in Figure 4.12) with implementation of latch 3 (as was shown in Figure 4.10) are compared for V_{dd} ranging from 0.5V to 0.9V. Delay is measured as the time between clock edge and the time in which output signal reaches $V_{dd}/2$ while a minimum size inverter is at output as load. Delay is selected from the maximum of the two values, low-to-high and high-to-low. As the simulation results show (see Figure 4.21),

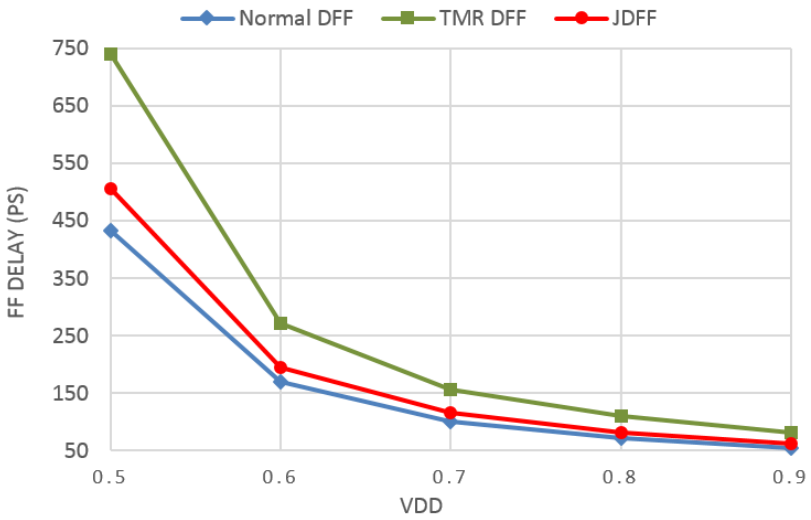

Figure 4.21: Comparison between delay of a normal DFF, a triplicated DFF with a majority voter at output (TMR DFF), and a Joint-DFF (JDFF).

the voting delay is significantly reduced form 50%-80% in the case of TMR DFF, to 10%-20% in the case of JDFF. This is mainly due to the fact that voter is not in series at output as it is the case in usual voting architectures.

4.9 Summary

In this work, we propose JSRAM, a novel and simple circuit technique for achieving a flexible trade-off between capacity and robustness inside SRAM based memories. SRAM has various applications such as CPU's cache, CPU's register file, FPGA BRAMs, and FPGA Look-Up Tables (LUTs). With special circuit design, we achieve promising improvements in Static Noise Margin (SNM), especially in near threshold V_{dd} and particularly for critical Read-SNM. This assists in achieving lower operating voltages. Significant improvement in critical charge (Q_{crit}) is also achieved to practically bring immunity against soft errors. Moreover, JSRAM provides a novel self-correcting technique to immediately correct any occurrence of single faults within the cell. Area overhead for having flexibility of JSRAM structure is four transistors per four SRAM cells or equivalently one

transistor per each SRAM cell. Similarly, based on the same idea, we propose Joint latch (JLatch) and Joint Flip-Flop (JFF), for achieving a flexible trade-off between available sequential logic resource size (number of user latches and FFs) and reliability, in FPGAs.

Chapter 5

Conclusion

In this thesis, we aimed to introduce a set of solutions at different design levels, all based on customized hardware design, to improve the safety, reliability, or performance of computer systems. Specifically, they are applicable to processors and FPGAs, as two fundamental processing resources in the age of AI. All of the proposed ideas, are based on resource replication, including redundant data, redundant processors, or multiple accelerators. More specifically, our solutions target 1) safety and reliability at the system-level using redundant processors, 2) performance at the architecture-level using multiple accelerators, and 3) reliability at the circuit-level through the use of redundant transistors. We summarize the contributions of this thesis below.

In Chapter 2, the significance and contribution of some prevalent parameters in the design of safety-critical computers are discussed. While previous studies considered different metrics separately, they did not take these metrics into account collectively. For example, some of the safety design parameters like failure diagnostic coverage (C) or common cause failure (CCF) ratio have been studied separately. Most often, it is not very clear that which part of the system is the Achilles heel and how design can be improved to reach standard safety levels. Motivated by such design ambiguities, we aimed to study the effect of various design parameters on safety, in some prevalent safety configurations, namely, 1oo2

(1-out-of-2, i.e. one fault-free processor out of two), and 2oo3 (2-out-of-3, i.e. two fault-free processors out of three). By employing Markov modeling, we analyzed the sensitivity of safety to important parameters including the failure rate of processor, failure diagnostic coverage, CCF ratio, test, and repair rates. This study aimed to provide a deeper understanding of the influence of variation in design parameters over safety. Consequently, to meet the appropriate safety integrity level, instead of improving some parts of a system blindly, it will be possible to make an informed decision on more relevant parameters. Most importantly, it was shown that the significant presence of Common Cause Failures (CCFs) is an important factor for safety analyses. It was observed that the parameters which have a considerable effect on CCF rate are more appropriate candidates for safety level enhancement, while some others barely can have a noticeable impact on the result.

In Chapter 3, the design and implementation of an HLS-based, FPGA-accelerated, high-throughput/work-efficient, synthesizable template-based graph processing framework has been presented. At the age of AI, hardware support for graph processing, as one of the most fundamental communication modeling methods, can be great assistance to industry and science. The template framework is simplified for easy mapping to FPGA, even for software programmers. The framework is particularly implemented on Intel state-of-the-art Xeon+FPGA platform, but it can be extended to other CPU-FPGA platforms. While a fixed and clock-wise precisely designed deep-pipeline architecture, written in SystemC, is responsible for processing graph vertices, the user implements the intended iterative graph algorithm by specializing only a single module in C/C++. This way, efficiency, and high-performance can be achieved with better programmability and productivity. In high-performance mode, the high-throughput pipeline achieves maximum edge throughput with the minimum number of accelerators. In addition, the work-efficient mode significantly reduces total run-time with a novel active-list design. With similar programming efforts, it is shown that the proposed template outperforms a high-throughput OpenCL baseline by up to 50% in terms of edge throughput. Furthermore, the novel work-efficient design significantly improves execution time and power consumption by up to 100X.

In Chapter 4, Joint SRAM/latch/Flip-Flop (JSRAM/JLatch/JFF) cell, a novel circuit-level hardening solution is introduced. Specifically, we implement a configurable static memory cell that exploits a trade-off between reliability and resource size. JSRAM technique is applicable to any SRAM structure like cache memory, register file, FPGA block RAM, or FPGA look-up table (LUT). In fault-prone conditions, the structure can be configured in such a way that four cells are combined together at the circuit level to form one large and robust memory bit. The proposed solution mainly focuses on transient faults. JSRAM enhances read stability by increasing critical Read Static Noise Margin (Read-SNM) which leads to a reduction in Bit Error Rate (BER), particularly at lower V_{dd} levels. Hold stability and critical charge are increased similarly to mitigate against soft errors, caused by radiation or noise in harsh environmental conditions. The joint structure is highly resistant to Multi Bit Upsets (MBUs) by selecting far enough cells, while it is fully immune against single faults, thanks to a novel self-correcting technique. The solution provides reconfigurability of reliability with negligible performance and area overhead with only one extra transistor per cell for joiner switches which stays idle during the normal full-memory usage mode. Unlike prevalent hardware redundancy techniques, like TMR, there is no explicit majority voter at the output, in this technique. Consequently, voter failure and latency are of less concern.

Bibliography

[1] P. Hokstad and K. Corneliussen, "Loss of safety assessment and the IEC 61508 standard," *Reliability Engineering and System Safety*, vol. 83, no. 1, pp. 111–120, 2004.

[2] "IEC 61508, functional safety of electrical/electronic/programmable electronic safety-related systems," standard, International Electrotechnical Commission (IEC), 2010.

[3] "ISA-TR84.00.02, Safety Instrumented Functions (SIF) – Safety Integrity Level (SIL) Evaluation Technique," Standard, Instrumentation Society of America (ISA), 2002.

[4] G. Chen, D. Sylvester, D. Blaauw, and T. Mudge, "Yield-driven near-threshold SRAM design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 11, pp. 1590–1598, 2009.

[5] Y. Park, M. Shankar, B.-H. Park, and J. Ghosh, "Graph databases for large-scale healthcare systems: A framework for efficient data management and data services," in *Proceedings of the IEEE 30th International Conference on Data Engineering Workshops (ICDEW)*, pp. 12–19, 2014.

[6] T. Aittokallio and B. Schwikowski, "Graph-based methods for analysing networks in cell biology," *Briefings in Bioinformatics*, vol. 7, no. 3, pp. 243–255, 2006.

[7] Twitter Corporation, "Twitter Q1 2019 earnings report." `https://investor.twitterinc.com/financial-information/quarterly-results`, 2019. [Online].

[8] H. R. Ghasemi, S. C. Draper, and N. S. Kim, "Low-voltage on-chip cache architecture using heterogeneous cell sizes for high-performance processors," in *Proceedings of the IEEE 17th International Symposium on High Performance Computer Architecture*, pp. 38–49, 2011.

[9] G. Memik, M. T. Kandemir, and O. Ozturk, "Increasing register file immunity to transient errors," in *Proceedings of the Design, Automation and Test in Europe (DATE)*, pp. 586–591, 2005.

[10] H. Ahangari, Y. I. Özkök, A. Yıldırım, F. Say, F. Atik, and O. Ozturk, "Analysis of design parameters in SIL-4 safety-critical computer," in *Proceedings of the annual Reliability and Maintainability Symposium (RAMS)*, 2017.

[11] H. Ahangari, F. Atik, Y. I. Özkök, A. Yıldırım, S. O. Ata, and O. Ozturk, "Analysis of design parameters in safety-critical computers," *IEEE Transactions on Emerging Topics in Computing*, vol. 8, no. 3, pp. 712–723, 2020.

[12] J. H. Wensley, L. Lamport, J. Goldberg, M. W. Green, K. N. Levitt, P. M. Melliar-Smith, R. E. Shostak, and C. B. Weinstock, "SIFT: Design and analysis of a fault-tolerant computer for aircraft control," *Proceedings of the IEEE*, vol. 66, no. 10, pp. 1240–1255, 1978.

[13] H. Kim, H. Lee, and K. Lee, "The design and analysis of AVTMR (all voting triple modular redundancy) and dual–duplex system," *Reliability Engineering and System Safety*, vol. 88, no. 3, pp. 291–300, 2005.

[14] M. Idirin, X. Aizpurua, A. Villaro, J. Legarda, and J. Melendez, "Implementation details and safety analysis of a microcontroller-based SIL-4 software voter," *IEEE Transactions on Industrial Electronics*, vol. 58, no. 3, pp. 822–829, 2011.

[15] X. Chen, G. Zhou, Y. Yang, and H. Huang, "A newly developed safety-critical computer system for China metro," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 2, pp. 709–719, 2013.

[16] W. Mechri, C. Simon, and K. BenOthman, "Switching markov chains for a holistic modeling of SIS unavailability," *Reliability Engineering and System Safety*, vol. 133, pp. 212–222, 2015.

[17] H. Jin, M. A. Lundteigen, and M. Rausand, "New PFH-formulas for k-out-of-n: F-systems," *Reliability Engineering and System Safety*, vol. 111, pp. 112–118, 2013.

[18] M. Chebila and F. Innal, "Generalized analytical expressions for safety instrumented systems' performance measures: PFDavg and PFH," *Journal of Loss Prevention in the Process Industries*, vol. 34, pp. 167–176, 2015.

[19] F. Innal, Y. Dutuit, and M. Chebila, "Monte carlo analysis and fuzzy sets for uncertainty propagation in SIS performance assessment," *World Academy of Science, Engineering and Technology, International Journal of Mathematical, Computational, Physical, Electrical and Computer Engineering*, vol. 7, pp. 1567–1575, 2013.

[20] J. Borcsok, S. Schaefer, and E. Ugljesa, "Estimation and evaluation of common cause failures," in *Proceedings of the second International Conference on Systems (ICONS)*, pp. 41–41, IEEE, 2007.

[21] A. Torres-Echeverria, S. Martorell, and H. Thompson, "Design optimization of a safety-instrumented system based on RAMS+ C addressing IEC 61508 requirements and diverse redundancy," *Reliability Engineering and System Safety*, vol. 94, no. 2, pp. 162–179, 2009.

[22] P. Hokstad, "Probability of Failure on Demand (PFD)-the formulas of IEC 61508 with focus on the 1oo2d voting," in *Proceedings of the European Safety and Reliability Conference (ESREL)*, 2005.

[23] J. Ilavsky, K. Rastocny, and J. Zdansky, "Common-cause failures as major issue in safety of control systems," *Advances in Electrical and Electronic Engineering*, vol. 11, no. 2, pp. 86–93, 2013.

[24] "IEC 61165, Application of Markov Techniques," Standard, International Electrotechnical Commission (IEC), 2006.

[25] D. J. Smith, *Reliability, Maintainability and Risk: Practical Methods for Engineers*. Butterworth-Heinemann, 2017.

[26] K. Rástočny and J. Ilavský, "Quantification of the safety level of a safety-critical control system," in *Proceedings of the International Conference on Applied Electronics*, 2010.

[27] H. Ahangari, M. Ozdal, and O. Ozturk, "HLS-based high-throughput and work-efficient synthesizable graph processing template framework," *IEEE Transactions on Parallel and Distributed Systems (submitted)*, 2020.

[28] M. Burtscher, R. Nasre, and K. Pingali, "A quantitative study of irregular programs on GPUs," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, pp. 141–151, 2012.

[29] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Ong Gee Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra, and G. Boudoukh, "Can FPGAs beat GPUs in accelerating next-generation deep neural networks?," in *Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 5–14, 2017.

[30] J. Cong, Z. Fang, M. Lo, H. Wang, J. Xu, and S. Zhang, "Understanding performance differences of FPGAs and GPUs," in *Proceedings of the IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 93–96, 2018.

[31] S. Yesil, M. M. Ozdal, T. Kim, A. Ayupov, S. Burns, and O. Ozturk, "Hardware accelerator design for data centers," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 770–775, 2015.

[32] Amazon Corporation, "Enable faster FPGA accelerator development and deployment in the cloud." `https://aws.amazon.com/ec2/instance-types/f1`, 2019. [Online].

[33] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, *et al.*, "A cloud-scale

acceleration architecture," in *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016.

[34] E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, *et al.*, "Serving DNNs in real time at datacenter scale with project brainwave," *IEEE Micro*, vol. 38, no. 2, pp. 8–20, 2018.

[35] Intel Corporation, "Rapid design methods for developing hardware accelerators." `https://github.com/intel/rapid-design-methods-for-developing-hardware-accelerators`, 2019. [Online].

[36] S. Grossman, H. Litz, and C. Kozyrakis, "Making pull-based graph processing performant," *ACM SIGPLAN Notices*, vol. 53, no. 1, pp. 246–260, 2018.

[37] M. M. Ozdal, S. Yesil, T. Kim, A. Ayupov, S. Burns, and O. Ozturk, "Architectural requirements for energy efficient execution of graph analytics applications," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 676–681, 2015.

[38] A. Mukkara, N. Beckmann, M. Abeydeera, X. Ma, and D. Sanchez, "Exploiting locality in graph analytics through hardware-accelerated traversal scheduling," in *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018.

[39] J. Zhang and J. Li, "Degree-aware hybrid graph traversal on FPGA-HMC platform," in *Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 229–238, 2018.

[40] A. Basak, S. Li, X. Hu, S. M. Oh, X. Xie, L. Zhao, X. Jiang, and Y. Xie, "Analysis and optimization of the memory hierarchy for graph processing workloads," in *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 373–386, 2019.

[41] M. Yan, X. Hu, S. Li, A. Basak, H. Li, X. Ma, I. Akgun, Y. Feng, P. Gu, L. Deng, *et al.*, "Alleviating irregularity in graph analytics acceleration: A hardware/software co-design approach," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 615–628, 2019.

[42] T. J. Ham, L. Wu, N. Sundaram, N. Satish, and M. Martonosi, "Graphicionado: A high-performance and energy-efficient accelerator for graph analytics," in *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016.

[43] N. Challapalle, S. Rampalli, L. Song, N. Chandramoorthy, K. Swaminathan, J. Sampson, Y. Chen, and V. Narayanan, "GaaS-X: Graph analytics accelerator supporting sparse data representation using crossbar architectures," *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 433–445, 2020.

[44] M. M. Ozdal, S. Yesil, T. Kim, A. Ayupov, J. Greth, S. Burns, and O. Ozturk, "Energy efficient architecture for graph analytics accelerators," in *Proceedings of the ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 166–177, 2016.

[45] A. Ayupov, S. Yesil, M. M. Ozdal, T. Kim, S. Burns, and O. Ozturk, "A template-based design methodology for graph-parallel hardware accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 2, pp. 420–430, 2018.

[46] G. Dai, T. Huang, Y. Chi, N. Xu, Y. Wang, and H. Yang, "Foregraph: Exploring large-scale graph processing on multi-FPGA architecture," in *Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 217–226, 2017.

[47] S. Zhou, C. Chelmis, and V. K. Prasanna, "High-throughput and energy-efficient graph processing on FPGA," in *Proceedings of the IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 103–110, 2016.

[48] S. Zhou, R. Kannan, V. K. Prasanna, G. Seetharaman, and Q. Wu, "Hitgraph: High-throughput graph processing framework on fpga," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 10, pp. 2249–2264, 2019.

[49] P. Yao, L. Zheng, X. Liao, H. Jin, and B. He, "An efficient graph accelerator with parallel data conflict management," in *Proceedings of 27th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2018.

[50] Q. Wang, L. Zheng, J. Zhao, X. Liao, H. Jin, and J. Xue, "A conflict-free scheduler for high-performance graph processing on multi-pipeline FPGAs," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 17, no. 2, 2020.

[51] H. R. Zohouri, N. Maruyama, A. Smith, M. Matsuda, and S. Matsuoka, "Evaluating and optimizing OpenCL kernels for high performance computing with FPGAs," in *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 409–420, 2016.

[52] N. Siret, M. Wipliez, J. F. Nezan, and F. Palumbo, "Generation of efficient high-level hardware code from dataflow programs," in *Proceedings of the Design, Automation and Test in Europe (DATE)*, 2012.

[53] J. d. F. Licht, S. Meierhans, and T. Hoefler, "Transformations of high-level synthesis codes for high-performance computing," *Computing Research Repository (CoRR)*, vol. abs/1805.08288, 2018.

[54] Intel Corporation, "Intel FPGA SDK for OpenCL pro edition: Programming guide." `https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/opencl-sdk/aocl_programming_guide.pdf`, 2020. [Online].

[55] Intel Corporation, "Intel FPGA SDK for OpenCL pro edition: Best practices guide." `https://www.intel.com/content/`

dam/www/programmable/us/en/pdfs/literature/hb/opencl-sdk/aocl-best-practices-guide.pdf, 2020. [Online].

[56] J. H. Reif, "Depth-first search is inherently sequential," *Information Processing Letters*, vol. 20, no. 5, pp. 229–234, 1985.

[57] G. E. Blelloch, J. T. Fineman, and J. Shun, "Greedy sequential maximal independent set and matching are parallel on average," in *Proceedings of 24th annual ACM Symposium on Parallelism in Algorithms and Architectures*, pp. 308–317, 2012.

[58] Texas A&M University, "The suitesparse matrix collection." https://sparse.tamu.edu, 2020. [Online].

[59] H. Ahangari, G. Yalcin, O. Ozturk, O. Unsal, and A. Cristal, "JSRAM: A circuit-level technique for trading-off robustness and capacity in cache memories," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2015.

[60] H. Ahangari, I. Alouani, O. Ozturk, and S. Niar, "Reconfigurable hardened latch and Flip-Flop for FPGAs," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2017.

[61] H. Ahangari, I. Alouani, O. Ozturk, S. Niar, and A. Rivenq, "Register file reliability enhancement through adjacent narrow-width exploitation," in *Proceedings of the International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*, 2016.

[62] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007.

[63] M. Wirthlin, "High-reliability FPGA-based systems: space, high-energy physics, and beyond," *Proceedings of the IEEE*, vol. 103, no. 3, pp. 379–389, 2015.

[64] R. C. Pang, S. P. Young, and T. J. Bauer, "Block RAM with configurable data width and parity for use in a field programmable gate array," Feb. 12 2002. US Patent 6,346,825.

[65] XILINX Corporation, "Xilinx 7 series fpgas, memory resources, user guide." `http://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf`, 2016. [Online].

[66] R. D. Wittig, S. Mohan, and R. A. Carberry, "FPGA configurable logic block with multi-purpose logic/memory circuit," Nov. 21 2000. US Patent 6,150,838.

[67] E. Seevinck, F. J. List, and J. Lohstroh, "Static-noise margin analysis of MOS SRAM cells," *IEEE Journal of Solid-state Circuits*, vol. 22, no. 5, pp. 748–754, 1987.

[68] B. H. Calhoun and A. P. Chandrakasan, "Static noise margin variation for sub-threshold SRAM in 65-nm CMOS," *IEEE Journal of Solid-state Circuits*, vol. 41, no. 7, pp. 1673–1679, 2006.

[69] A. Pavlov and M. Sachdev, *CMOS SRAM Circuit Design and Parametric Test in Nano-scaled Technologies: Process-aware SRAM Design and Test*. Springer, 2008.

[70] S. Yoshimoto, T. Amashita, S. Okumura, K. Yamaguchi, M. Yoshimoto, and H. Kawaguchi, "Bit error and soft error hardenable 7T/14T SRAM with 150-nm FD-SOI process," in *Proceedings of the IEEE International Reliability Physics Symposium (IRPS)*, 2011.

[71] V. Ferlet-Cavrois, L. W. Massengill, and P. Gouker, "Single event transients in digital CMOS—a review," *IEEE Transactions on Nuclear Science*, vol. 60, no. 3, pp. 1767–1790, 2013.

[72] E. Ibe, H. Taniguchi, Y. Yahagi, K.-i. Shimbo, and T. Toba, "Impact of scaling on neutron-induced soft error in SRAMs from a 250-nm to a 22-nm design rule," *IEEE Transactions on Electron Devices*, vol. 57, no. 7, pp. 1527–1538, 2010.

[73] R. Baumann, "Soft errors in advanced computer systems," *IEEE Design & Test of Computers*, vol. 22, no. 3, pp. 258–266, 2005.

[74] H. Nan and K. Choi, "High performance, low cost, and robust soft error tolerant latch designs for nanoscale CMOS technology," *IEEE Transactions on Circuits and Systems*, vol. 59, no. 7, pp. 1445–1457, 2012.

[75] L. Chang, D. M. Fried, J. Hergenrother, J. W. Sleight, R. H. Dennard, R. K. Montoye, L. Sekaric, S. J. McNab, A. W. Topol, C. D. Adams, *et al.*, "Stable SRAM cell design for the 32-nm node and beyond," in *Digest of Technical Papers, Symposium on VLSI Technology*, 2005.

[76] T. Calin, M. Nicolaidis, and R. Velazco, "Upset hardened memory design for submicron CMOS technology," *IEEE Transactions on Nuclear Science*, vol. 43, no. 6, pp. 2874–2878, 1996.

[77] P. Dodd, M. Shaneyfelt, J. Schwank, and J. Felix, "Current and future challenges in radiation effects on CMOS electronics," *IEEE Transactions on Nuclear Science*, vol. 57, no. 4, pp. 1747–1763, 2010.

[78] A. Keys, J. H. Adams, J. D. Cressler, M. C. Patrick, M. A. Johnson, and R. C. Darty, "Radiation hardened electronics for space environments (RHESE) project overview," in *International Planetary Probes Workshop*, 2008.

[79] L. H.-H. Kelin, L. Klas, B. Mounaim, R. Prasanthi, I. R. Linscott, U. S. Inan, and M. Subhasish, "LEAP: Layout design through error-aware transistor positioning for soft-error resilient sequential cell design," in *Proceedings of the IEEE International Reliability Physics Symposium (IRPS)*, 2010.

[80] S. M. Jahinuzzaman, D. J. Rennie, and M. Sachdev, "A soft error tolerant 10T SRAM bit-cell with differential read capability," *IEEE Transactions on Nuclear Science*, vol. 56, no. 6, pp. 3768–3773, 2009.

[81] D. Krueger, E. Francom, and J. Langsdorf, "Circuit design for voltage scaling and ser immunity on a quad-core itanium® processor," in *Proceedings of the IEEE International Solid-State Circuits Conference-Digest of Technical Papers*, 2008.

[82] T. Loveless, S. Jagannathan, T. Reece, J. Chetia, B. Bhuva, M. McCurdy, L. Massengill, S.-J. Wen, R. Wong, and D. Rennie, "Neutron-and proton-induced single event upsets for D-and DICE-flip/flop designs at a 40 nm technology node," *IEEE Transactions on Nuclear Science*, vol. 58, no. 3, pp. 1008–1014, 2011.

[83] B. Maric, J. Abella, and M. Valero, "APPLE: Adaptive performance-predictable low-energy caches for reliable hybrid voltage operation," in *50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013.

[84] H. Fujiwara, S. Okumura, Y. Iguchi, H. Noguchi, Y. Morita, H. Kawaguchi, and M. Yosimoto, "Quality of a bit (QoB): A new concept in dependable SRAM," in *Proceedings of the 9th International Symposium on Quality Electronic Design (ISQED)*, pp. 98–102, 2008.

[85] S. Paul, F. Cai, X. Zhang, and S. Bhunia, "Reliability-driven ECC allocation for multiple bit error resilience in processor cache," *IEEE Transactions on Computers*, vol. 60, no. 1, pp. 20–34, 2011.

[86] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. Hoe, "Multi-bit error tolerant caches using two-dimensional error coding," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2007.

[87] G. Yalcin, A. Seyedi, O. S. Unsal, and A. Cristal, "Flexicache: Highly reliable and low power cache under supply voltage scaling," in *Proceedings of the Latin American High Performance Computing Conference*, 2014.

[88] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu, "Trading off cache capacity for reliability to enable low voltage operation," *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3, 2008.

[89] C. Carmichael, M. Caffrey, and A. Salazar, "Correcting single-event upsets through Virtex partial configuration," *XILINX Application Notes XAPP216 (v1.0)*, 2000.

[90] J. McCollum, R. Lambertson, J. Ranweera, J. Moriarta, J.-J. Wang, F. Hawley, and A. Kundu, "Reliability of antifuse-based field programmable gate arrays for military and aerospace applications," in *Proceedings of the MAPLD International Conference*, ACTEL Corporation, 2001.

[91] XILINX Corporation, "Radiation-hardened, space-grade Virtex-5QV family overview," *DS192 (v1.3)*, 2012.

[92] XILINX Corporation, "XILINX TMRTOOL." `www.XILINX.com/ise/optional_prod/tmrtool.htm`, 2016. [Online].

[93] J. Hu, S. Wang, and S. G. Ziavras, "In-register duplication: Exploiting narrow-width value for improving register file reliability," in *International Conference on Dependable Systems and Networks (DSN'06)*, pp. 281–290, 2006.

[94] J. Hu, S. Wang, and S. G. Ziavras, "On the exploitation of narrow-width values for improving register file reliability," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 7, pp. 953–963, 2009.

[95] M. Kandala, W. Zhang, and L. T. Yang, "An area-efficient approach to improving register file reliability against transient errors," in *Proceedings of the 21st IEEE International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*, vol. 1, pp. 798–803, 2007.

[96] P. Reviriego, J. A. Maestro, S. Baeg, S. Wen, and R. Wong, "Protection of memories suffering MCUs through the selection of the optimal interleaving distance," *IEEE Transactions on Nuclear Science*, vol. 57, no. 4, pp. 2124–2128, 2010.

[97] H. Fujiwara, S. Okumura, Y. Iguchi, H. Noguchi, H. Kawaguchi, and M. Yoshimoto, "A dependable SRAM with 7T/14T memory cells," *IEICE Transactions on Electronics*, vol. 92, no. 4, pp. 423–432, 2009.

[98] P. Hazucha and C. Svensson, "Impact of CMOS technology scaling on the atmospheric neutron soft error rate," *IEEE Transactions on Nuclear science*, vol. 47, no. 6, pp. 2586–2594, 2000.

[99] *XCell Journal, XILINX*, vol. 86, 2014 Q1.

[100] W. Zhao and Y. Cao, "Predictive technology model (ptm) website." `http://ptm.asu.edu`, 2012. [Online].

[101] C. Shin, *Advanced MOSFET designs and implications for SRAM scaling*. PhD thesis, UC Berkeley, 2011.

[102] P. Jain and V. Zhu, "Judicious choice of waveform parameters and accurate estimation of critical charge for logic SER," in *Proceedings of the Workshop of Dependable Secure Nanocomputing*, 2007.