A Cross-Layer Framework for Adaptive Processor-Based Systems Regarding Error Resilience and Power Efficiency

Von der Fakultät 1 - MINT - Mathematik, Informatik, Physik,
Elektro- und Informationstechnik
der Brandenburgischen Technischen Universität Cottbus-Senftenberg

genehmigte Dissertation

zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
(Dr.-Ing.)

vorgelegt von

M.Sc.
Mitko Veleski
geboren am 18. April 1986 in Ohrid, Mazedonien

Gutachter: Prof. Dr.-Ing. Rolf Kraemer
Gutachter: Prof. Dr.-Ing. habil. Michael Hübner
Gutachter: Prof. Dr.-Ing. Miloš Krstić

Vorsitzender: Prof. Dr.rer.nat. Peter Langendörfer

Tag der mündlichen Prüfung: 10. November 2022

DOI: https://doi.org/10.26127/BTUOpen-6145

"Imagination allows you to think of the journey worth making.
Motivation gets you started.
But, it's patience and perseverance that get you there."

— Ernie J. Zelinski

Abstract

The transistor miniaturization to nanoscale dimensions enabled building extremely powerful computing systems capable of executing complicated tasks very efficiently. Processors are at the heart of almost every computing device. The usage of nanoscopic components for processor realization opened the opportunity for placing multiple processor cores on a single chip. Increasing the processing power means improving the system performance significantly. However, implementation of such complex designs introduces some serious challenges. First, scaling the technology to near-atomic dimensions makes the circuits prone to variabilities that might cause malfunctions and eventually lead to failures. In other words, the modern computing systems are becoming less resilient. Second, putting so many transistors on such a small surface increases the system power consumption dramatically. Hence, it is of fundamental importance to come up with an adequate strategy for addressing these challenges. There are many solutions which have been successful either in improving resilience or in reducing power consumption, but a single approach that deals with both challenges appears to be missing. As a matter of fact, development of such approach is aggravated by the inverse relationship between the two metrics. However, employing techniques which would adapt the system behaviour according to the current application requirements or environmental conditions might be promising.

This dissertation proposes a cross-layer framework able to synergistically optimize resilience and power consumption of processor-based systems. It is composed of three building blocks: SWIELD multimodal flip-flop (FF), System Operation Management Unit (SOMU) and Framework Function Library (FFL). Implementation of the building blocks is performed at circuit, architecture and software layer of the system stack respectively. The SWIELD FF can be configured to operate as a regular flip-flop or as an enhanced flip-flop for protection against timing/radiation-induced faults. It is necessary to perform replacement of selected timing-critical flip-flops in a system with SWIELD FFs during design time. When the system is active, the SWIELD FFs operation mode is dynamically managed by the SOMU controller according to the current requirements. Finally, the FFL contains a set of software procedures that greatly simplify framework utilization. By relying on the framework, a system can intelligently interchange techniques such as Adaptive Voltage/Frequency Scaling, selective Triple Modular Redundancy and clock gating during operation. Additionally, a simple and convenient strategy for integration of the framework in processor-based systems is also presented. A key feature of the proposed strategy is to determine the number of SWIELD FFs to be inserted in a system. Using this strategy, the framework was successfully embedded in instances of both single- and multicore systems. Various experiments were conducted to evaluate the framework influence on the target systems with respect to resilience and power consumption. At expense of about 1% area overhead, the framework is able to preserve performance and to reduce power consumption up to 15%, depending on the number of SWIELD FFs in the system. Furthermore, it was also shown that under certain conditions, the framework can provide failure-free system operation.

Zusammenfassung

Die Miniaturisierung der Transistoren auf Strukturgrößen im Nanometerbereich ermöglichte den Bau extrem leistungsfähiger Computersysteme, die komplizierte Aufgaben sehr effizient ausführen können. Prozessoren sind das Herzstück fast aller Datenverarbeitungssysteme. Die Verwendung von nanoskopischen Komponenten für die Realisierung von Prozessoren eröffnete die Möglichkeit, mehrere Prozessorkerne auf einem einzigen Chip unterzubringen. Die Erhöhung der Verarbeitungsleistung bedeutet eine deutliche Verbesserung der Systemleistung. Die Umsetzung solch komplexer Schaltungen bringt jedoch einige ernsthafte Herausforderungen mit sich. Erstens macht die Skalierung der Technologie auf nahezu atomare Dimensionen die Schaltungen anfällig für Schwankungen, die zu Fehlfunktionen und schließlich zu Ausfällen führen können. Mit anderen Worten, die modernen Computersysteme werden weniger resilient. Zweitens erhöht sich der Stromverbrauch des Systems dramatisch, wenn so viele Transistoren auf einer so kleinen Fläche untergebracht werden. Daher ist es von grundlegender Bedeutung, eine geeignete Strategie zur Bewältigung dieser Herausforderungen zu entwickeln. Es gibt viele Lösungen, die entweder die Resilienz verbessern oder den Stromverbrauch senken, aber ein einziger Ansatz, der beide Probleme angeht, scheint zu fehlen. Die Entwicklung eines solchen Konzepts wird durch die umgekehrte Beziehung zwischen den beiden Messgrößen erschwert. Der Einsatz von Techniken, die das Systemverhalten an die aktuellen Anwendungsanforderungen oder Umgebungsbedingungen anpassen, könnte jedoch vielversprechend sein.

In dieser Dissertation wird ein schichtenübergreifendes Framework vorgeschlagen, das in der Lage ist, die Resilienz und den Stromverbrauch von prozessorbasierten Systemen synergetisch zu optimieren. Es setzt sich aus drei Bausteinen zusammen: *SWIELD Multimodal Flip-Flop (FF)*, *System Operation Management Unit (SOMU)* und *Framework Function Library (FFL)*. Die Implementierung der Bausteine erfolgt jeweils auf der Schaltkreis-, Architektur- und Softwareebene. Das SWIELD FF kann so konfiguriert werden, dass es als normales Flipflop oder als erweitertes Flipflop zum Schutz vor zeit- und strahlungsbedingten Fehlern arbeitet. Während der Entwurfszeit müssen ausgewählte zeitkritische Flipflops im System durch SWIELD FFs ersetzt werden. Wenn das System aktiv ist, wird der Betriebsmodus der SWIELD FFs vom SOMU-Controller entsprechend den aktuellen Anforderungen dynamisch verwaltet. Schließlich enthält das FFL eine Reihe von Softwareverfahren, die die Nutzung des Frameworks erheblich vereinfachen. Durch den Einsatz des Frameworks kann das System Techniken wie Adaptive Voltage/Frequency Scaling, selektive Triple Modular Redundancy und Clock Gating während des Betriebs intelligent austauschen. Außerdem wird eine einfache und komfortable Strategie zur Integration des Frameworks in prozessorbasierte Systeme vorgestellt. Eine wesentliche Eigenschaft der vorgeschlagenen Strategie besteht darin, die Anzahl der SWIELD FFs zu bestimmen, die in das System eingefügt werden sollen. Mit dieser Strategie wurde das Framework erfolgreich in Instanzen von Single- und Multicore-Systemen integriert. Es wurden verschiedene Experimente durchgeführt, um den Einfluss des Frameworks auf die Zielsysteme im Hinblick auf Resilienz und Stromverbrauch zu bewerten. Auf Kosten von ca. 1%

Flächen-Overhead ist das Framework in der Lage, die Verarbeitungsleistung zu erhalten und den Stromverbrauch, abhängig von der Anzahl der SWIELD FFs im System bis 15% zu reduzieren. Darüber hinaus wurde gezeigt, dass das Framework unter bestimmten Bedingungen einen ausfallfreien Systembetrieb gewährleisten kann.

Acknowledgements

This work would have not been possible without the help from many people and several organizations. First of all, I would like to express my most sincere gratitude to the German Academic Exchange Service (DAAD) for awarding me with a 3-year scholarship within the Graduate School Scholarship Program for doctoral students. Without such kind of support, this amazing experience would be unimaginable. Special thanks to Prof. Dr.-Ing. Heinrich Theodor Vierhaus - professor at the Brandenburg University of Technology (BTU) Cottbus-Senftenberg and former leader of the Computer Engineering research group for his help and guidance throughout the application process. I am forever grateful to Prof. Dr.-Ing. Rolf Kraemer - professor at BTU and former head of the System Design department at the Institute for High Performance Microelectronics (IHP) in Frankfurt (Oder) for accepting to be my supervisor and for his support during my doctoral studies. My warmest gratitude goes to Prof. Dr.-Ing. Miloš Krstić - professor at the University of Potsdam and head of the System Architectures department at IHP for continuously assisting me with extremely useful suggestions and remarks related to my research. Thank you both Prof. Krstić and Prof. Kraemer for pushing me bit further every time and challenging me to go beyond what I thought were my limits. The numerous discussions I had with you helped me shape and develop a scientific way of thinking. Furthermore, I would like to thank the Zoran Djindjić Internship Program for granting me scholarship back in 2012 and enabling me to spend six months in IHP where I gained a valuable experience and met my present supervisors. This internship turned out to be the springboard for my career.

I would especially like to thank my former professors from the Faculty of Electrical Engineering and Information Technologies (FEEIT) in Skopje, Prof. Dr. Aristotel Tentov for the inspiration and for instilling my interest in processor architectures as well as to Prof. Dr. Katerina Raleva for introducing me to digital circuit design. A huge thanks to Dr. Zoran Stamenković (IHP) for helping me prepare the research proposal. I owe a great debt of gratitude to Prof. Dr.-Ing. habil. Michael Hübner - professor and vice president at BTU for recognising my potential and inviting me to join his Computer Engineering research group following the end of my funding period. This work would have not been finished without his help. Of course, many thanks to my colleagues Dr.-Ing. Aleksandar Simevski, Dr.-Ing. Milan Babić, Dr. Felipe Kuentzer, Dr.-Ing. Marko Andjelković, Dr. Yuanqing Li, Dr.-Ing. Vladimir Petrović (from IHP), Dr. Marcelo Brandalero, Dr.-Ing. Marc Reichenbach and Stefan Scharoba (from BTU) for the useful remarks regarding many technical issues.

Finally, I would like to express my eternal gratitude for the endless love and the unconditional support to the members of my closest family: my wonderful wife Petra, my sister Milena, my father Aco and my late mother Atina whom I lost during my doctoral studies. Although it is devastating being unable to share this great achievement with her, I hope that she stands proud, wherever she might be now. Mom, this is to you!

In a loving memory of
D-r. Atina Veleska (1959-2020).
A respected doctor,
a caring mother and
a wonderful human being.

Contents

List of Own Publications

[VHKK20] Mitko Veleski, Michael Hübner, Milos Krstic, and Rolf Kraemer. Highly configurable framework for adaptive low power and error-resilient system-on-chip. In *2020 23rd Euromicro Conference on Digital System Design (DSD)*, pages 24–28. IEEE, 2020.

[VHKK21a] Mitko Veleski, Michael Hübner, Milos Krstic, and Rolf Kraemer. Design and implementation strategy of adaptive processor-based systems for error resilient and power-efficient operation. In *2021 24th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pages 57–62. IEEE, 2021.

[VHKK21b] Mitko Veleski, Michael Hübner, Milos Krstic, and Rolf Kraemer. Towards error resilient and power-efficient adaptive multiprocessor system using highly configurable and flexible cross-layer framework. In *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–7. IEEE, 2021.

[VKK17] Mitko Veleski, Rolf Kraemer, and Milos Krstic. An overview of cross-layer resilience design methods. In *2017 Workshop on Reliability, Security and Quality RESCUE - ETS'17 Fringe Workshop*, 2017.

[VKK18] Mitko Veleski, Rolf Kraemer, and Milos Krstic. The effects of voltage scaling on reliability and power consumption in multiprocessor systems. In *2018 Testmethoden und Zuverlässigkeit von Schaltungen und Systemen (TuZ)*, 2018.

[VKK19] Mitko Veleski, Rolf Kraemer, and Milos Krstic. SWIELD: An in situ approach for adaptive low power and error-resilient operation. In *2019 IEEE East-West Design & Test Symposium (EWDTS)*, pages 1–6. IEEE, 2019.

[VPS12] Mitko Veleski, Vladimir Petrovic, and Zoran Stamenkovic. A satellite internal communication controller: Design and implementation. In *11th WSEAS International Conference on Circuits, Systems Electronics, Control and Signal Processing*, 2012.

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

Today's most powerful computing chips accommodate several billion transistors lying on a surface with an area of a fingernail. This amazing accomplishment is a result of the continuous transistor miniaturization process referred to as *technology scaling*. Indeed, approximately every two years over the last few decades, new transistor generations have been developed. Thereby, the feature sizes of each successive generation were reduced by about 30% in comparison to the previous one [25, 26]. An illustration of the technology scaling process is shown in Figure 1.1.

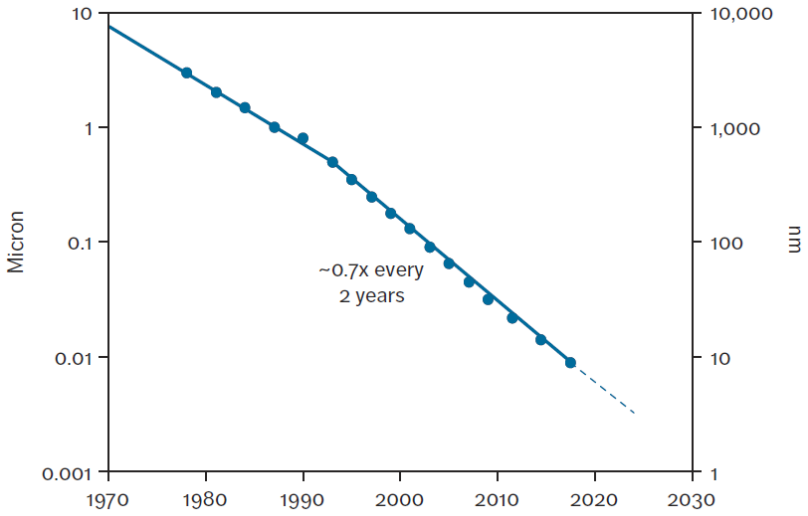


Figure 1.1: Feature size scaling trends for Intel technologies. (Source: Bohr and Young [25])

The benefits from the technology scaling are well known: every successive transistor generation was aiming to [26]:

- improve the performance by increasing operating frequencies and reducing the gate delay;

- reduce the power consumption per transistor or per gate;
- double the number of transistors that could be integrated per chip area (transistor density).

Of course, higher transistor densities allowed implementation of more functions per chip. Putting it all together, every new transistor generation enabled developing more powerful computing devices.

A great majority of the computing systems are centered around processors (single- or multicore). Table 1.1 shows how the technology scaling affected some key design parameters for processors produced by Intel. The table lists 19 different processor generations starting from the oldest.

Table 1.1: Evolution of the processors over the years. The supply voltage levels of the processors from more recent generations are not fixed, rather they can be dynamically scaled within strictly defined intervals. TDP stands for Thermal Design Power. It refers to the maximal power that the processor can consume when executing typical programs. The peak power consumption can exceed the TDP. Intel processors used only for illustration purposes. Sources: Intel and Rupp et al. [107].

Production Year	Processor Name	Feature Size	Transistor Count	Maximum Frequency	Supply Voltage (V)	TDP (W)	Technology	Area (mm^2)	Number of Cores
1971	4004	10 μm	2.3k	108 kHz	15	0.5	PMOS	12	1
1974	8080	6 μm	6k	2 MHz	5/-5/12	0.8	NMOS	20	1
1976	8085	3 μm	6.5k	3 MHz	5	0.9	NMOS	20	1
1978	8086	3 μm	29k	10 MHz	5	1.8	NMOS	33	1
1982	80286	1.5 μm	134k	12 MHz	5	3.3	NMOS	47	1
1985	i386 DX	1.5 μm	275k	33 MHz	5	2	CMOS	103	1
1989	i486 DX	1 μm	1.2M	50 MHz	5	4.5	CMOS	173	1
1993	Pentium	0.8 μm	3.1M	66 MHz	5	13	BiCMOS	294	1
1995	Pentium Pro	0.6 μm	5.5M	200 MHz	3.3	35	BiCMOS	307	1
1997	Pentium II	0.35 μm	7.5M	300 MHz	2.8	21.5	CMOS	203	1
2000	Pentium III	0.18 μm	28M	1 GHz	1.7	22	CMOS	90	1
2002	Pentium 4 Northwood	0.13 μm	55M	2.8 GHz	1.475-1.525	69.7	CMOS	145	1
2004	Pentium 4 Prescott	90 nm	125M	3.8 GHz	1.2-1.4	115	CMOS	112	1
2006	Core 2 Duo	65 nm	291M	2.66GHz	0.85-1.5	65	CMOS	143	2
2008	Core i7 940	45 nm	731M	2.9 GHz	0.8-1.375	130	HKMG CMOS	263	4
2010	Core i7 Gulftown	32 nm	1170M	3.6 GHz	0.8-1.375	130	HKMG CMOS	248	6
2012	Core i7 Ivy Bridge	22 nm	1400M	3.5 GHz	N/A	77	3D Tri-Gate FinFET	160	4
2014	Core i7 Haswell E	22 nm	2600M	3.3 GHz	N/A	140	3D Tri-Gate FinFET	355	6
2016	Core i7 Broadwell E	14 nm	3200M	3 GHz	N/A	140	FinFET	246	10

The technology scaling goals were defined by the Moore's law and the Dennard scaling rules more than four decades ago [95, 45]. In fact, the processor design has been conforming to these two statements for approximately 30 years, but they are no longer valid [61]. One can draw such conclusion simply by observing the data in Table 1.1. For example, the processor produced in 2014 contains 2600M transistors. The next generation processor, produced in 2016, according to the Moore's law, should have contained approximately 5200M transistors. [1] Instead, its transistor count is "only" 3200M. Furthermore, note that starting from 2006, the minimal supply voltage levels of the processors are no longer reduced. The operating frequencies have also stopped

[1] Given that it occupies the same area as its predecessor.

increasing and have been maintaining steady level of around 3 GHz to 3.5 GHz for more than 10 years. Therefore, the main factors that imposed serious challenges to the technology scaling process and discontinuation of Moore's law/Dennard scaling rules can be summarized as follows:

- The transistor sizes were reduced to near-atomic dimensions, close to the boundaries of quantum mechanics principles [34]. Such devices are difficult to control and highly prone to variabilities. Unstable transistors with unpredictable behaviour may easily cause malfunctions or even failures. Therefore, maintaining the device **resilience** began to draw increased attention.

- Placing such enormous number of transistors on a single chip die causes power dissipation and heating so excessive, that it became too expensive and unprofitable to employ adequate cooling technologies [60, 100]. This prevented transistors integration with the pace projected by the Moore's law.

- The reduction of the transistors supply and threshold voltages according to the Dennard scaling rules reached a point where even an inactive transistor consumed power [76]. This phenomenon is caused by the *leakage currents* whose impact on the power consumption was negligible in the earlier technologies, and therefore, disregarded by Dennard et al. However, with the further technology minimization, the share of the leakage currents in the overall power consumption increased substantially [27]. Consequently, the power density also started to grow instead of remaining constant as postulated by the Dennard scaling rules.

The listed factors had a serious impact on the processor design. In that sense, two major challenges required prompt reaction - the extremely high power dissipation and the deteriorating device resilience.

Resilience is a superset of **dependability**. Avizienis et al. define dependability of a computing system as "the ability to deliver service that can be justifiably trusted" [14]. In general, dependability should be perceived as a broad concept characterized with specific *attributes*, *threats* and *means*. The threats refer to factors that could cause dependability deterioration (*fault*, *error*, *failure*), whereas the means specify ways to maintain high level of dependability (e.g *fault tolerance*). Finally, the attributes serve for dependability assessment. Resilience is an even wider concept defined by Laprie as "the persistence of dependability when facing **changes**" [82]. Hence, a resilient system needs to preserve its dependability in a presence of possibly unexpected changes. They might appear in a form of disturbances caused either by internal or external factors. Typical representatives of changes with detrimental influence on computing systems include:

- **static** and **dynamic variations** - in the literature also known as **PVTA** (process, voltage, temperature and aging) variations [137, 69]. The ultimate effects of PVTA variations are *timing errors* [136, 118, 52, 123, 83] which could easily result in system failure if not properly handled.

- **radiation effects** or **Single Event Effects (SEEs)** - most commonly manifested as transient faults like *Single Event Upsets (SEUs)* which may lead to critical *soft errors*. Although soft errors do not physically damage the device, they could cause data corruption, system malfunction or even a failure.

Formerly, only a small segment of the processor market was concerned with dependability - mainly processors from the embedded domain for mission- or safety-critical applications. Over time, as the processor-based systems became more susceptible to variabilities, the resilience has become almost equally important for all processor-based systems regardless of their domain [91]. The concepts of dependability and resilience are thoroughly discussed in Section 2.3.

On the other hand, the problem of excessive power consumption was strongly felt in the entire processor-based systems market. This huge handicap, widely known as the *power wall* led to adoption of a new processor design paradigm. Concretely, instead of one power-inefficient processor, the designers switched to building two or more efficient processors or processor *cores* on a single chip [100, 61]. The advantages of such approach were multifold - the high operating frequencies used before switching to multiprocessors could be lowered (to prevent breaking the power wall) while the performance could still go up due to the parallelism offered as an inherent feature of using multiple processors. [2] Therefore, without impacting the performance, the power consumption problem has been overcome, but only temporarily. Before too long, many-core systems encompassing hundreds of cores on a single die emerged as a preferred platform for high performance computing applications. It is anticipated for the next-generation of many-core chips to incorporate thousands of processor cores [115]. The already gigantic number of transistors on a single chip has continued to increase, although with a lot slower pace than the Moore's projections. Regardless, the power density has reached a point where significant part of the chip components have to be turned off or operated in some low power mode during runtime to avoid overheating and potential system failure. This phenomenon known as *dark* or *dim silicon* has a strong influence on all contemporary multi-/many-core computing systems. Its emergence was a clear indication that the high power consumption is becoming a major problem again. Along with the deteriorated resilience, it is still one of the biggest challenges in the field of processor-based system design.

As suggested by the title, this dissertation proposes a solution for resilient and power-efficient processor-based systems. In the main focus are embedded safety-/mission-critical processor-based systems due to their "natural" requirements for high degree of resilience and power efficiency. However, the proposed approach can be applied to any type of processor-based system. Next section discusses the relationship between the resilience and power consumption in more details. Section 1.3 presents the current trends in processor-based system design. Finally, short overview to the thesis proposal is given in Section 1.4.

[2] Of course, adequately written software is necessary to maximize the concurrent code execution.

1.2 The Interplay Between Resilience and Power

One of the key factors that decide whether a soft or timing error will cause a failure is the presence of redundancy in the system [14, 13]. Many systems contain redundant components as an intrinsic part of their structure. However, from the aspect of dependable design, most of the times this is not sufficient. In fact, for implementation of a dependable or resilient system, one must intentionally and explicitly incorporate redundancies [12].

Redundancy can come in four distinct forms: *hardware, software, information* and *time*. Depending on the nature of faults to be mitigated, different forms of redundancy can be used. For example, against faults in hardware, usually hardware, information or time redundancy is used, while software redundancy is utilized against faults in software. Hardware redundancy is widely used in computing systems that require some form of fault tolerance. Actually, fault tolerance based on hardware redundancy, i.e. *hardware fault tolerance* is considered as the most mature area in the broad field of fault-tolerant and dependable computing [79]. Generally speaking, hardware redundancy is employed by addition of extra hardware component(s) in the system whose purpose is to detect, mask, correct and if applicable, predict faults. The obvious drawback of such approach is the inevitable overhead in terms of increased area and power consumption introduced by the additional hardware. Indeed, it has been reported by multiple authors that depending on the number of employed redundant modules, the area and power overheads can exceed 300% [103, 110].

On the other hand, utilization of techniques like voltage scaling in order to reduce the power consumption has shown to negatively affect the resilience in two ways. First, reducing the supply voltage slows down the switching speed of transistors which may result in timing errors [123, 58]. Failing to meet the timing constrains is often unacceptable and may cause serious to catastrophic consequences depending on the system purpose. Second, if voltage is scaled, transistors have lower noise margins that make the circuit more prone to electromagnetic interference and noise, while the conductivity of the metal conductors is lowered [46, 133]. Additionally, it has been also shown that the susceptibility to soft errors increases substantially with voltage reduction [33]. In fact, a previously proposed model in literature states that the *Soft Error Rate (SER)* increases exponentially with supply voltage scaling [141, 48]. The model assumes that the interarrival time of soft errors follows a Poisson distribution with rate λ_0 while the system operates at its nominal voltage level V_{max}. Furthermore, it is assumed that the supply voltage of the system can be scaled between V_{max} and V_{min}, thereby the SER corresponding to V_{min} would be $\lambda_0 10^d$. In such conditions, the SER of a system operating at supply voltage level V can be expressed as:

$$\lambda(V) = \lambda_0 10^{\frac{d(V_{max}-V)}{V_{max}-V_{min}}} \tag{1.1}$$

where where $d > 0$ is a technology-dependent constant that reflects the variations of the SER with the voltage scaling and $V_{min} < V < V_{max}$.

In essence, resilience and power consumption typically have inverse relationship and improving both metrics simultaneously is difficult. Their interplay is illustrated

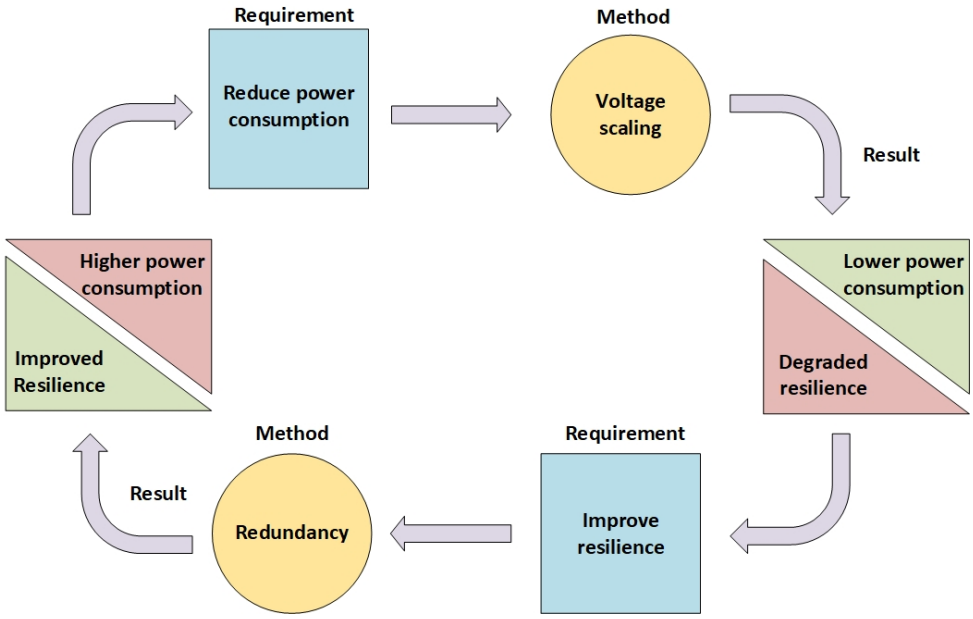


Figure 1.2: Relationship between resilience and power consumption.

in Figure 1.2. Hence, it is important to investigate methods that can optimize the trade-off between resilience and power consumption in a synergistic way.

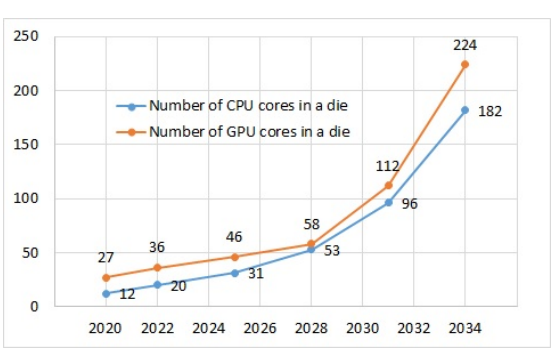
1.3 Current Trends in Processor System Design

The emergence of the dark silicon phenomenon, as discussed in Section 1.1, was a clear sign that further system performance improvement only by increasing the number of processor cores on a single chip is not possible without hitting the power wall. In order to preserve the performance enhancement trends under tight power budgets, the processor-based systems design principles had to be fundamentally revised. The fact that mechanisms such as Instruction Level Parallelism (ILP) and multiprocessing have been already fully exploited, leaving no obvious solutions in sight, forced radical changes in the way of designing processor-based systems. The new design paradigm is slowly moving away from homogeneous multiprocessor systems towards heterogeneous *Domain Specific Arcitectures (DSA)*. Despite general-purpose processors, DSAs also contain special-purpose cores able to perform only limited set of tasks, but very efficiently in terms of both performance and power. Nevertheless, the general-purpose cores are going to remain basic building blocks of every processor-based computer and will be responsible for execution of essential software such as operating systems.

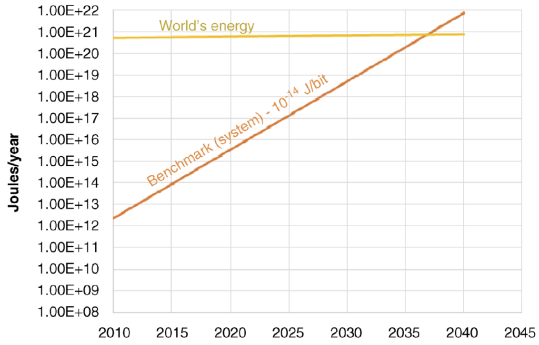
1.3.1 Power Consumption Trends

Today, in the era of artificial intelligence, big data and ubiquitous computing in general, when tremendous processing power is required, the energy consumption emerged as a first-class optimization metric for design of processor-based computing systems.

In this regard, Figure 1.3 shows some intriguing trends. According to the latest IRDS [3] report [1], the number of CPU and GPU cores in a single System-On-Chip (SoC) die is expected to increase almost exponentially in the period 2020-2034 (Figure 1.3(a)). Such vertiginous growth would inevitably lead to explosion in demands for power. Especially concerning are the projections illustrated in Figure 1.3(b) stating that the energy required for computing could possibly exceed the world's estimated energy production by 2040 [9, 104]. Therefore, employing efficient methods for power-aware computing is more than urgent and of critical priority.



(a) Number of CPU and GPU cores in a single SoC die. (Source: IRDS report 2020 edition [1])



(b) Computing energy requirements estimation. (Source: Raj et al. [104])

Figure 1.3: Projected trends of computing power and energy demands.

The chip level power consumption trends for SoCs projected by earlier ITRS [4] report [2] are shown in Figure 1.4. Two key observations can be drawn from the figure. First, a SoC produced in 2026 will consume approximately twice more power than a SoC manufactured in 2021/2022. Second, although the leakage power (in both logic and memory) accounts for a substantial fraction of the total chip power consumption, it is expected that the dynamic power consumption in logic remains the dominant component at least until 2026. Hence, it is crucial to include adequate techniques for switching power reduction in logic as an integral part of processor-based systems, as well as to investigate approaches for further improvement of those techniques.

1.3.2 Resilience Trends

The negative impact of the technology scaling on the resilience results in shortened computing system lifetime. A widely used model that illustrates the dependence of the failure rate on the system age is the so-called *bathtub curve* [79] (see Figure 2.3 on page 25). Using a bathtub-like shape as an analogy, this curve divides a computing system lifetime into three periods or phases: early life, useful life and wear out phase. During both early life and wear out phase, the failure rate is high. However, the useful

[3] International Roadmap for Devices and Systems (`https://irds.ieee.org`)
[4] International Technology Roadmap for Semiconductors

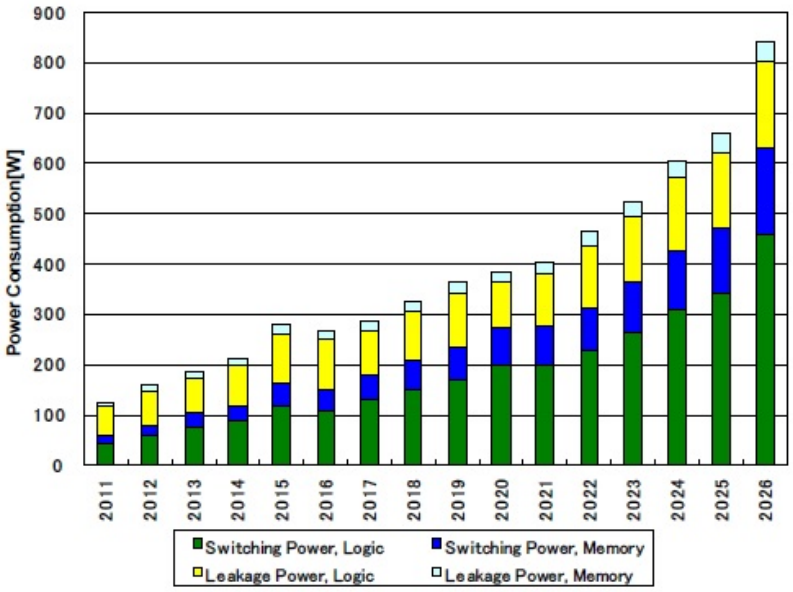
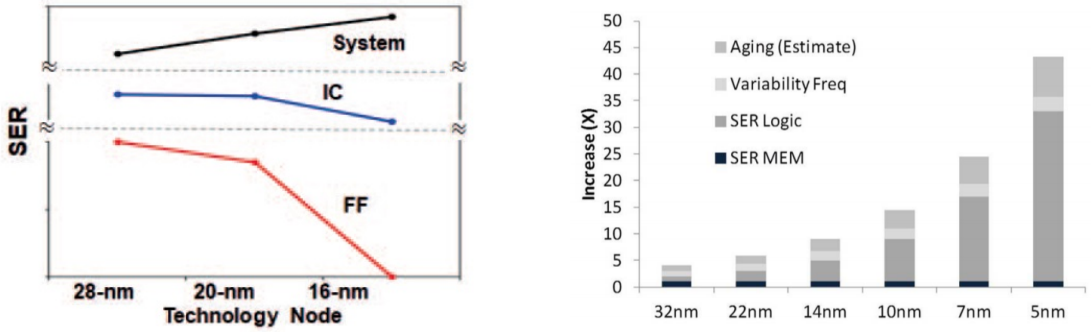


Figure 1.4: Chip-level power consumption trends for SoCs. (Source: ITRS report 2015 edition [2])

life phase is characterized by lower and fairly constant failure rate. As elaborated further in this section, the impaired resilience leads to increased failure rates during all three phases of the system lifetime. First, during production of such miniature devices, manufacturing defects are practically unavoidable. Consequently, the rate of early life failures is rising [32].

Second, reducing the transistor sizes increases the sensitivity of individual transistors to radiation-induced particles because even a lower energy particle strike can result in soft error [39]. On the other hand, smaller transistor implies smaller sensitive area. Hence, depending on the specific process, the SER at device-level either monotonically decreases or remains approximately constant with the technology scaling due to the reduced probability of a radiation event affecting an individual device. However, as technology scaling allows extremely high number of transistors to be integrated in a single chip, it has been shown that the system-level SER increases dramatically with each new transistor generation [24, 85] (Figure 1.5(a)).

Historically, Static Random Access Memories (SRAMs) accounted for the highest fraction of the overall SER in processors [77, 85]. In the most advanced technologies, however, the SER in logic largely exceeds the SER in SRAM (Figure 1.5(b)) due to the fact that SRAMs are regular structures which can be efficiently protected using well-established and relatively straightforward techniques such as error correction codes. In contrast, hardening the logic typically requires more complex approaches. Depending on the soft error type(s) to be addressed, the implementation of adequate techniques might be quite challenging and costly. SEUs are considered the dominant contributor to the SER in logic [77, 97]. Table 1.2 shows a raw SEU rate per processor manufactured in different technology. The rate is normalized relative to the 45 nm node.



(a) SER rates at flip-flop, IC and system levels. (Source: Bhuva [24])

(b) Relative increase of error rates. (Source: Daly et al. [41])

Figure 1.5: Trends in error rates as a function of a process technology.

Table 1.2: Raw SEU rate per processor in different technology. The values are normalized relative to the 45 nm node. (Source: Kim et al. [77])

Technology (nm)	Normalized SEU rate per processor
45	1X
32	1.38X
22	1.59X

As can be seen, the SEU rates per processor keep increasing with the technology scaling. Based on data from the Table 1.2 as well as from Figure 1.5, it is to expect that the SEUs and soft errors in general will remain critical resilience issue for future technologies. Once considered a threat primarily to the systems used in space applications, soft errors have lately become a great concern even for commercial electronic products at ground level [97]. If not adequately addressed, soft errors can potentially produce a higher failure rate compared to all other dependability threats combined [21]. As a result, soft error mitigation has been widely investigated topic in the last few decades.

Third, the continuous technology scaling aggravates the sensitivity of modern computing systems to dynamic variations. Along with soft errors, dynamic variations comprise a major threat to systems during their normal operation phase (useful lifetime) [32]. Last but not least, computing systems implemented in the most advanced technologies experience early aging effects [32, 128]. As stated previously, both aging effects and dynamic variations manifest themselves as timing errors. Thus, timing errors have become another great resilience concern, especially in modern technologies [52, 128].

To summarize, as a result of rapid technology scaling, the early life failure rate rises, systems are becoming more prone to errors and variations during their useful lifetime and the wear out phase starts sooner than expected. These trends lead to

significant shortening of systems' lifetime, i.e. shrinking of the bathtub curve as illustrated in Figure 1.6.

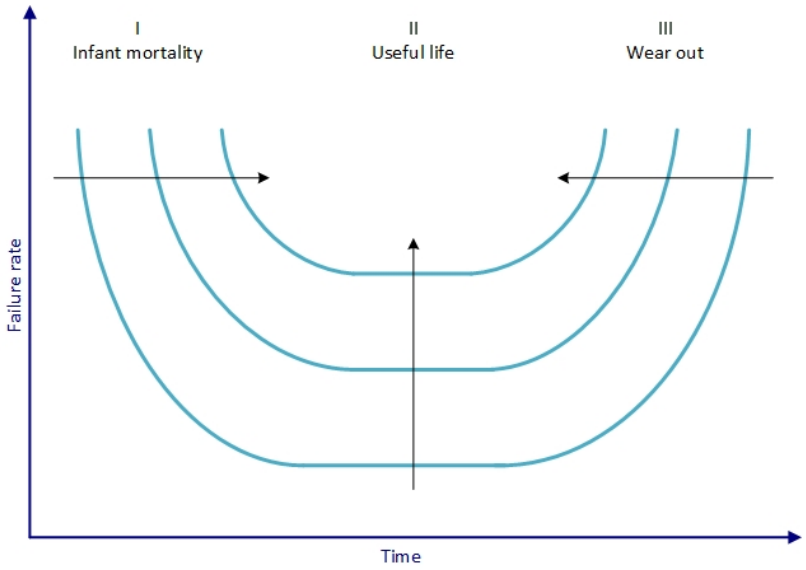


Figure 1.6: Shrinking of the bathtub curve. (Adapted from: Chandra [32])

Recently, approaches for so-called **cross-layer resilience** have become prominent. These methods are utilizing the fact that every computing system can be viewed as a stack composed of several layers. In essence, the goal is combining multiple techniques for resilience implemented at different layers across the system stack to work together. Cross-layer approaches are considered to have the potential to deliver high performance resilient systems at lower costs by distributing the responsibility for mitigating errors/addressing variations across multiple layers. It is believed that the cross-layer paradigm can contribute to building systems that will maintain the desired resilience level while utilizing the benefits of technology scaling.

1.4 Thesis Proposal

Processors' speed and efficiency have been steeply increasing for decades. However, the price of performance improvement is usually paid by degradation of other parameters. The major challenges and trends in processor-based system design dictated by the technology scaling were thoroughly elaborated in Section 1.1 and Section 1.3, respectively. This Section discusses the motivation behind the thesis and briefly introduces the proposed solution.

1.4.1 Motivation

The work presented in this dissertation is strongly motivated by the current challenges and trends in processor-based system design with regards to resilience and power consumption. A short summary of the main motivation points follows below:

- **Challenge:** the excessive power consumption has become the greatest concern not only for the modern processor-based systems, but for all the computing devices of today, regardless of their type or class.
  - **Trend:** the power demands will continue to increase tremendously in the next 10-15 years since it is expected the number of cores on a single chip to grow almost exponentially.
  - **Trend:** dynamic power in logic will remain dominant source of the chip-level power consumption.
- **Challenge:** the resilience of the contemporary processor-based systems to faults and variabilities is sharply declining.
  - **Trend:** the system-level SER increases dramatically, mostly due to soft errors in logic. The dominant contributor to the SER in logic are the SEUs.
  - **Trend:** growing dynamic variations and early aging effects lead to more frequent occurrence and higher rate of timing errors.
  - **Trend:** the cross-layer resilience methods are considered more efficient in comparison to the traditional "single-layer" approaches.
- **Challenge:** the conflicting relationship between the power consumption and the resilience hinders simultaneous optimization of the both metrics.

To address the listed challenges as well as to conform to the current trends, ideally a solution which focuses primarily on reducing both the dynamic power consumption and the SEUs in logic is required. Thereby, an adaptive adjustment of the supply voltage is preferable for optimal power consumption [30, 51, 136]. However, as a consequence of the contradicting trade off between the resilience and the power consumption, the supply voltage reduction increases the sensitivity of the system to SEUs and timing errors. Both SEUs and timing errors can be efficiently mitigated by introducing some type of redundancy [17, 97, 37, 38]. Since the timing errors are predictable [22, 136], protecting only the timing-critical flip-flops (against timing errors) in logic would possibly suffice. On the other hand, the SEUs are totally random and thus, only full-scale protection of the flip-flops in logic could "guarantee" SEU-free operation. In that scenario, using Triple Modular Redundancy (TMR) or any other form of redundancy for SEU tolerance would unavoidably lead to drastic power overhead which is exactly the opposite of the intended goal. Hence, implementation of such ideally conceptualized processor-based system - simultaneously power efficient and completely resilient is obviously not feasible.

Fortunately, most systems don't have to be completely resilient and power efficient at once and all the time. In fact, a system might often be expected to accommodate a broad spectrum of applications, sometimes with distinct demands. The applications themselves are dynamically changing their requirements with respect to the current needs as well as according to the current internal and environmental conditions. Moreover, an optimal trade-off between non-trivial and non-complementary

parameters such as resilience and power consumption might frequently be required. To provide all this, a system capable to adjust itself to variations and to operate optimally in all conditions [32] is necessary. Such a system is called **adaptive system**.

Adaptivity is an important feature of every system intended to be resilient [82]. Implementing adaptive processor-based system requires utilization of innovative design techniques at circuit, architecture and system level [32], that is, a cross-layer approach. However, besides being resilient, the system of interest needs also to be power efficient. A convenient way to build such a system is to add high degree of configurability to its components, especially to the ones responsible for providing the resilience and power management.

1.4.2 Proposed Framework and Thesis Contributions

Inspired by the facts and tendencies elaborated in the previous Subsection, this dissertation introduces a **highly configurable cross-layer framework** which enables smooth adaptation of a processor-based system to the current conditions and application requirements with respect to both resilience and power consumption without affecting the performance.

The proposed framework is composed of three building blocks:

- **SWIELD multimodal flip-flop** (in further text denoted as SWIELD FF);
- **System Operation Management Unit (SOMU)**;
- **Framework Function Library (FFL)**.

Figure 1.7 shows general implementation concept of the framework in a processor-based system. The SWIELD FF and the SOMU are implemented in hardware. Concretely, the SWIELD FF is implemented at circuit layer, whereas the SOMU belongs to the architecture layer. On the other hand, the FFL is implemented in software.

A brief description of the individual building blocks and their role in the framework is given in the following paragraphs.

The SWIELD FF can be configured to operate in three distinct modes:

1. **Regular FF** mode. A SWIELD FF can conveniently replace a standard flip-flop. This should be done during design time;
2. **In situ monitor (ISM) FF** [5] mode for implementation of voltage scaling scheme and timing errors prediction. While in this mode, the SWIELD FF acts as a delay monitor and sends feedback to the SOMU;
3. **TMR FF** mode for protection against soft errors;

Switching between the operation modes can be done dynamically, while the system is online, on demand by applications or some other system components such as sensors.

[5] In situ is a latin phrase that can be translated literally as "on site" or "in the natural/original position/place". (Source: Merriam-Webster Dictionary)

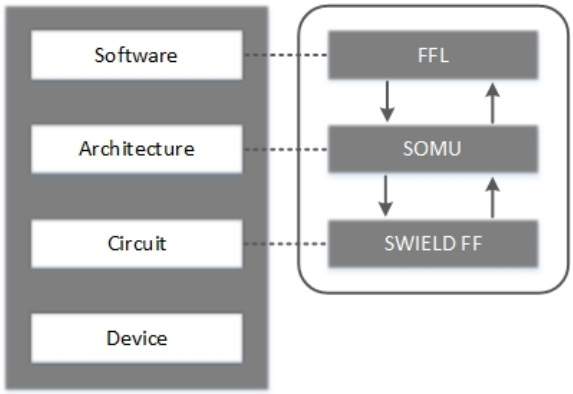


Figure 1.7: General implementation concept of the proposed framework in a processor-based system. Both the system (left) and the framework (right) are abstracted as stacks composed of several layers. Every building block of the framework is connected via dashed lines to its corresponding layer of implementation.

This concept is crucial for providing adaptivity and flexibility to the system. Of course, some extra circuitry is necessary for implementing the enhanced SWIELD FF functionalities.

As the name itself suggests, the SOMU has several important system operation management functions including:

- driving the system clock generation logic (for implementation of clock-gating and frequency scaling);
- driving the prospective voltage regulator(s) (for implementation of supply voltage scaling);
- driving the SWIELD FFs and managing their operation modes.

Most of the SOMU functions are performed on request by the FFL.

Finally, the FFL contains functions/procedures that enable simple manipulation of the framework. A programmer can configure the framework or retrieve the status of various framework-related parameters by calling an adequate FFL function without having to worry about the hardware implementation details.

A possible operation scenario of a system encompassing the proposed framework could be the following: during the system initialization phase, to ensure high performance, the default operation mode of the SWIELD FFs is set to Regular FF. When a non-critical application is executed, to save power, the user calls an FFL function that will instruct the SOMU to put the SWIELD FFs into ISM FF mode. In this way, the framework gradually reduces the supply voltage of the system to the lowest possible level that won't cause timing errors. If protection against soft errors is required, the appropriate FFL function that switches the SWIELD FFs operation mode to TMR FF can be called.

For full utilization of the benefits offered by the framework, its building blocks have to be appropriately interfaced with the processor-based system. Furthermore, as shown in Figure 1.7, the communication between the framework building blocks within the system has to be performed in a closed-loop manner. This calls for some modifications in the traditional system design flow. Another contribution of the thesis is a proposition of simple and convenient strategy for integration of the framework in processor-based systems that fits well to the standard design flow.

1.4.3 Objectives and Scope of the Thesis

The main objective of this thesis is to investigate a **synergistic approach** to building adaptable processor-based systems regarding resilience and power consumption. Having in mind their conflicting relationship, the aim is to provide an optimal trade-off between the two metrics by integration of the proposed framework in processor-based systems. Note that the focus of the thesis is on general-purpose processors.

The framework is applicable to both single- and multiprocessor systems. This is important because after hitting the power wall, most of today's computing systems (whether used in the desktop, server or embedded domain) are heavily based on multiprocessors. Multiprocessing is very powerful concept that offers flexibility as an inherent feature. On the other hand, the proposed framework encompasses techniques such adaptive voltage/frequency scaling, clock-gating and circuit-level TMR which can be interchanged in an intelligent way while the system is on-line. Putting all together, the framework should drive the multiprocessor to optimally utilize the intrinsic flexibility and the incorporated techniques depending on the current needs. In this direction, the publication [VKK18] investigates theoretical models for implementation of adaptive and flexible multiprocessor system able to provide reasonable trade-off between reliability and power consumption.

Regardless of whether the framework is integrated into single- or multiprocessor system, the goal is to optimize power consumption while preserving acceptable level of resilience by dynamically switching between the provided SWIELD FF operation modes. It should be emphasized that this thesis deals mainly with SEUs as most critical soft errors. Furthermore, it is expected the system performance to be preserved due to the framework's capability to predict and avoid timing errors.

Finally, the proposed framework integration strategy should provide implementation of adaptable processor-based system for resilient and power-efficient operation at minimal area overhead. Therefore, the logical question arises: how many and which flip-flops in the system should be replaced with SWIELD FFs during design time? The SWIELD FF is augmented with additional circuitry and, as previously pointed out, replacing every flip-flop with SWIELD FF would be counterproductive in terms of power efficiency. On the other hand, the critical flip-flops with respect to timing account for only a small portion of the overall number of the flip-flops in the system. Thus, an important question to which this dissertation is about to answer is "how much the power consumption and the resilience of a (multi)processor-based system to both timing and soft errors can be improved by utilizing the proposed framework if only timing critical flip-flops are replaced with SWIELD FFs?"

1.4.4 Thesis Structure

The remainder of this dissertation is organized as follows. Chapter 2 presents the necessary background information, crucial for understanding the thesis.

Chapter 3 gives a comprehensive overview of the related works in this area. A special attention is paid to the state-of-the-art solutions to the problems and challenges introduced in Chapter 1. Thereby, the works are classified according to relevant criteria (e.g. approaches for improving resilience, power consumption or both). The final section of the chapter elaborates the contribution of this thesis that go beyond the existing state-of-the-art approaches.

Chapter 4 introduces the proposed cross-layer framework. The structure and the features of each framework building block (SWIELD FF, SOMU, FFL) are described in details. Furthermore, the three SWIELD FF operation modes (ISM FF, TMR FF, regular) are thoroughly explained. A dedicated section discusses how to select the right SWIELD FF operation mode and the potential factors that might affect this decision.

Chapter 5 elaborates the strategy for integration of the framework in processor-based systems. Additionally, the architectures of the processor systems (both single- and multicore) used as test vehicles for concept evaluation are also presented. The chapter is wrapped up with implementation results of the framework integration in the considered processor-based systems.

Chapter 6 is devoted to evaluation of the proposed approach. The methodologies and tools used for this purpose are demonstrated at the beginning of the chapter. Next, the influence of the framework on the processor-based systems operation in terms of power consumption, resilience, area and performance is investigated through experiments. The parameters of interest are assessed in three distinct scenarios: minimal single-core system, multiprocessor for high performance as well as multiprocessor for prolonged system lifetime. Individual sections present the experimental results for each scenario. The final section in this chapter discusses the obtained results and analyses how the proposed concept compares to related works from Chapter 3.

At last, Chapter 7 summarizes the main points of this work, argues whether the challenges as well as the objectives stated in Chapter 1 are met and finally, it concludes the dissertation. A short outline of the future work that stems from the conducted research is also given.

For distinction of the own publications from regular references, different citation styles are used. Concretely, works from the literature are cited using plain style, whereas alpha style is used for citing an own publication. The complete bibliography can be found on page 119, while the own publications are listed on page xv.

Chapter 2

Background

The dissertation topic brings together several broad fields. This chapter reviews basics that are essential for comprehension of the proposed approach. Section 2.1 discusses the importance of the technology scaling process. The types and characteristics of processor-based systems are presented in Section 2.2. In Section 2.3 the concepts of dependability and resilience are reviewed. Section 2.4 elaborates on the sources of power consumption. Finally, Section 2.5 focuses on the potential origins of failures.

2.1 Importance of the Technology Scaling

Ever since the invention of the Integrated Circuit (IC) in the late 1950s, the semiconductor-based electronics has been the main technological driving force in the world. Today, cutting-edge computing devices are widely used in almost every aspect of the everyday life and the modern human society is unimaginable without them. This is a result of a tremendous and permanent cost-performance improvement of the IC technology throughout the years starting with the processor emergence in the 1970s. For more than four decades now, the processor-centric digital computing devices have been mass-produced. The IC technology advancements in combination with innovative design techniques resulted in each processor generation to be faster, more efficient and richer in functions than the preceding one. Such impressive growth, (in terms of both performance and scale of integration) has been facilitated by continuous size reduction of the most-widely used semiconductor device and basic building block of the electronic industry - the transistor abbreviated as MOSFET. The size of a MOSFET is commonly expressed through metric called *feature size*. [1]

Gordon E. Moore in 1965 famously stated that the number of transistors per chip would increase at a rate of roughly factor of two per year for at least a decade [94]. Ten years later, he revised his earlier statement and formulated what is today known as the *Moore's law*: the number of transistors per chip would double every two years [95].

Another critical observation that goes in line with the Moore's law was published by Robert Dennard *et al.* in 1974. This postulate, referred to as the *Dennard scaling*

[1] The minimum distance between the source and drain regions in a MOSFET.

rules, proposed a methodology for scaling the dimensions (channel length, width...) and characteristics (doping concentration, operating frequency, supply/threshold voltages...) of MOSFETs to build faster transistors that occupy less area and consume less power. Under such circumstances, according to the Dennard scaling rules, the *power density* of a given chip area remains constant [45]. In other words, integrating larger number of smaller, faster and more power-efficient transistors on a given chip area will not increase the overall power consumption of that chip area.

Both Moore's law and Dennard scaling rules are of fundamental importance and have been shaping the vertiginous advancement of the semiconductor industry for decades.

2.2 Processor-Based Computing Systems

Depending on the specification, intended purpose and application(s) to be executed, the processor-based computing systems are divided into three well-known distinct classes: *desktop* or *personal computers (PCs)*, *servers* and *embedded computers* [100]. Although systems from different classes, unsurprisingly, have different requirements and are designed using different hardware technologies, the power consumption problem affects every modern processor-based system, regardless to which class it belongs. Therefore, employing some kind of low power technique is nowadays a standard part of the processor-based system design flow.

On the other hand, desktop, server and embedded systems, clearly, have still different requirements regarding dependability/resilience, in spite of the fact that these two concepts are becoming more and more relevant with the technology miniaturization. For example, the processors intended to be used in the desktop domain are primarily designed to optimize the *cost-performance* ratio. Employing dependability/resilience mechanisms has obviously lower priority here and it comes down to usage of relatively simple techniques for memory protection.

In the server domain, availability is the top concern. Since the servers are expected to be reachable non-stop, their downtime is extremely costly. Depending on the application, a server downtime might cost from approximately $50,000 per hour (e.g. media companies) to $4,000,000 per hour (brokerage services) [61]. Thus, maintaining availability is a crucial task for the servers. Regarding dependability/resilience, typically techniques for memory protection and storage-/processor-level redundancies are employed.

Finally, embedded systems as the largest class of computers encompasses a broad spectrum of diverse applications starting from consumer electronics used in everyday life (mobile phones, tablets, TV sets, gaming consoles...) to *critical* systems such as ships, aeroplanes and spacecrafts. A failure of a processor-based system embedded in a plane or satellite communication controller would cause far more serious consequences (possibly disastrous) in comparison to a tablet application crash. Therefore, dependability/resilience are crucial requirements for a critical system. During the design of a critical system, special attention must be paid to selection of adequate (set of) techniques that will provide the necessary level of dependability/resilience. Of course,

different type of critical system demands different level of dependability/resilience. According to the possible consequences of a potential failure, the critical systems can be categorized as:

- **Safety-critical** - where a failure is a serious threat to human health or natural environment, i.e. may cause injuries or even loss of lives (e.g. medical, aviation, automotive systems etc.).
- **Mission-critical** - where a failure may prevent the system to successfully complete its projected objectives or goals (e.g. Unmanned Aircraft System (UAS) such as satellite or drone).
- **Business-critical** - where a failure may lead to substantial financial losses or serious reputation damage (e.g. stock trading).
- **Security-critical** - where a failure may result in an unauthorized access and theft of sensitive information (e.g. banks).

It is worth mentioning that these categories are overlapping to some extent, that is, for example, a system can be at the same time business-critical and security-critical etc [63].

Additionally, the systems operating in environments where human intervention for maintenance is not possible (such as space), are usually powered by batteries or photovoltaic cells. Hence, besides dependability/resilience, power efficiency is another key requirement here.

2.3 Dependability and Resilience Fundamentals

This section gives a brief overview of the fundamental dependability and resilience concepts. Widely cited papers by Avizienis et al. [14, 13] and Laprie [82] whose acclaimed contribution towards definition of the terms dependability and resilience respectively are used as references.

2.3.1 Dependability

In one of the earlier papers focusing on the taxonomy of dependable computing, the authors argue that the term *reliability* might be etymologically more appropriate [12]. However, the term *dependability* seemed to be better fit as it reflects "our society's ever increasing dependence upon sophisticated systems in general and especially upon computing systems". Reliability is instead used as a dependability attribute or mathematical measure of the continuous delivery of proper service.

Dependability is considered as a more general concept that encompasses:

- ***attributes*** of dependability;
- ***threats*** to dependability;

- ***means*** to attain dependability.

The complete dependability taxonomy shown in a form of a tree structure is shown in Figure 2.1.

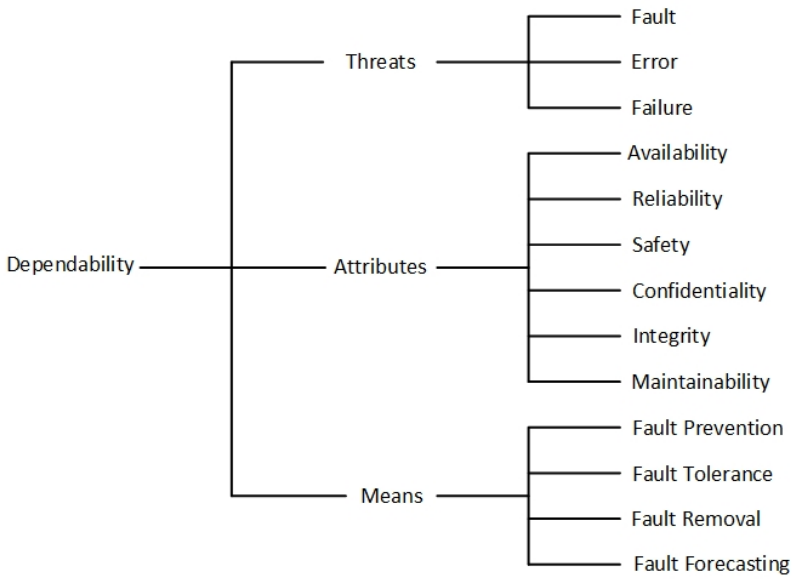


Figure 2.1: The dependability tree. (Adapted from: Avizienis et al. [14])

Dependability Attributes

A computing system delivers a **correct service** when it implements the intended function defined by a system *specification*. In this regard, there are five *primary* dependability attributes defined as:

- **availability**: readiness for correct service;
- **reliability**: continuity of correct service;
- **safety**: absence of catastrophic consequences on the users and the environment;
- **integrity**: absence of improper system alterations;
- **maintainability**: ability to undergo modifications and repairs.

In addition to the listed primary attributes, there are also *secondary* attributes which can be inferred by specializing some of the primary attributes. For example, **robustness** is an important and widely investigated secondary attribute defined as dependability with respect to external faults.

It is important to note that the threats to dependability cannot be completely avoided, that is, a system will inevitably experience a fault at some point. Therefore, it is pointless to expect from a system to be entirely reliable, safe, available etc. Instead, its dependability attribute(s) can be evaluated only by using probabilistic approaches.

Dependability Threats

When the system implements a function that differs from the specification, it delivers an incorrect service. The event that caused a transition from correct to incorrect service is called a **failure**.

An **error** is a deviation from the correct service within the system that may or may not lead to failure. In order to cause a failure, an error needs to propagate to the system output. Fortunately, not all errors cause failures. Two key factors decide whether an error will actually lead to failure:

1. The *structure* of the system, concretely, the type of *redundancy* that it contains:
    - *protective* redundancy - intentionally integrated into the system in order to prevent errors from causing failures;
    - *unintentional* redundancy - intrinsic part of the system that sometimes has the same effect as the protective redundancy.
2. The *behaviour* of the system - the part of the system that contains the error might be overwritten, or simply, never used.

The presence of an error in the system can be reported by error *message* or *signal*. In such case, the error is **detected**. A **latent** error is undetected error which exists in the system.

The hypothesized or adjudged cause of an error is called a **fault**. A fault is an abnormal condition in the system that can be caused by *internal* factors (e.g. design/production flaw, wear out...) or *external* factors (e.g radiation, malicious attack...). The faults that lead to errors are referred to as **active** faults, whereas **dormant** faults do not result in errors.

There are several ways for faults classification. One of the most commonly used is the classification according to the fault *duration* or *persistence* which divides the faults into three distinct classes: transient, intermittent and permanent. **Transient faults** have short duration and usually occur as a result of electromagnetic interference or noise. **Permanent faults** are caused by physical defects in the hardware due to design/manufacturing problems, overheating or wear out and typically remain in the system until repair is performed. Finally, **intermittent faults** reappear from time to time and are usually caused by voltage or temperature fluctuations.

Essentially, an error occurs through fault activation. [2] A propagation of an error to the system output results in failure. Finally, a failed system might cause a fault to another system which receives its service(s). Figure 2.2 illustrates the complete chain sequence of threats to dependability.

[2] A fault can be activated either by some external source or internally as a result of some computation processes/environmental conditions.

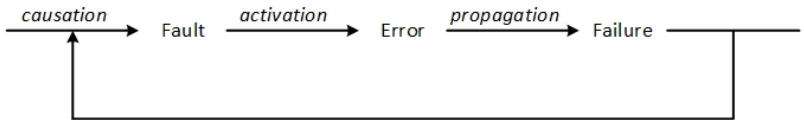


Figure 2.2: The fundamental chain of dependability threats. (Adapted from: Avizienis et al. [13])

Dependability Means

In order to attain the dependability attributes, or equivalently, to prevent the dependability threats, adequate dependability means are necessary. Dependability means is actually a set of techniques which can be categorized into four subsets:

1. **Fault prevention** - includes general engineering techniques whose goal is preventing occurrence or introduction of faults. Examples of such techniques are shielding, radiation hardening, following rigorous design rules and maintenance protocols, etc.

2. **Fault tolerance** - aims to keep delivering correct service, i.e. to avoid failures in a presence of faults. In general, it is implemented using error detection followed by *recovery*. The error detection can be accomplished in two ways: during normal system operation (*concurrent detection*) or while system operation is temporarily suspended for fault/error checking (*preemptive detection*). After an error detection message has been signalled by the system, recovery is performed. The goal is to transform the system state that contains errors and potential faults, into an error-/fault-free state. Recovery from errors is conducted by *error handling* which can be done in three ways:

   - *rollback* - restoring a previously saved error-free state in the system;
   - *rollforward* - bringing the system to a new, error-free state;
   - *compensation* - eliminating the errors from the system by relying on redundant components. If there is enough redundancy in the system, explicit error detection doesn't have to be performed since the correct state will be provided by the redundant components. This type of recovery is known as **fault masking**. A fault will be masked only if the majority of the redundant components contain the correct system state.

   Fault handling is a type of recovery that prevents reactivation of known faults. It is typically done in four steps:

   - *fault diagnosis* - identification of error causes, types and locations;
   - *fault isolation* - exclusion (logical or physical) of the faulty components from the system operation;
   - *system reconfiguration* - activation of spare components or task reassignment to failure-free components;

- *system reinitialization* - system update with the new configuration.

3. **Fault removal** - can be performed during both development and use phases of the system. While a system is being developed, the fault removal is usually conducted in three possible steps: *verification*, *diagnosis* and *correction*. First, the system needs to go through the verification process which checks whether given properties defined as *verification conditions* are met. If that's not the case, the remaining two steps have to be taken: fault diagnosis to determine why the verification failed followed by adequate corrections. Ultimately, the system needs to pass through the verification once again. This is necessary to check whether the fault removal was successful. During the system usage, fault removal is performed through *corrective* or *preventive maintenance*. The goal of the corrective maintenance is to remove faults that caused errors reported by the system, whereas the preventive maintenance aims at eliminating faults before they lead to errors during normal operation. In summary, fault removal can be seen as a means to reduce the number and severity of faults.

4. **Fault forecasting** - estimates the current number, the future rate and the possible consequences of faults. It is done by *qualitative* and *quantitative* evaluation of the system behaviour with respect to fault activation or occurrence. A well-established and widely used method for fault forecasting is **fault injection (FI)**. By employing fault injection, the behaviour of the system containing faults can be observed and adequate actions regarding fault prevention/fault tolerance/fault removal can be planned.

Common Dependability Measures

As previously pointed out, dependability attributes of a system can be evaluated only by using probabilistic approaches. Logically, quantitative dependability measures are expressed through concepts of probability theory.

Let the continuous random variable T denote the lifetime of a computing system (the time until it fails). Furthermore, let the probability density function (PDF) and the cumulative distribution function (CDF) of T be denoted by $f(t)$ and $F(t)$, respectively. Since the lifetime cannot be negative, the functions are defined for $t \geq 0$ only. The relationship between $f(t)$ and $F(t)$ is given by

$$f(t) = \frac{dF(t)}{dt} \tag{2.1}$$

As a PDF, $f(t)$ has to fulfill the condition $f(t) \geq 0$ as well as

$$\int_0^\infty f(t)dt = 1 \tag{2.2}$$

for $t \geq 0$ [16]. Next, let $F(t)$ be the probability that a system failure will occur before time t

$$F(t) = P(T \leq t) = \int_0^t f(x)dx \tag{2.3}$$

Thus, the *reliability* of a system can be expressed as the probability that the system will survive until time t, given that it was operating correctly at time 0

$$R(t) = P(T \geq t) = 1 - F(t) = 1 - \int_0^t f(x)dx = \int_t^\infty f(x)dx \tag{2.4}$$

Equation 2.4 indicates the reliability of a system starting operation at time 0 and expected to fail at future time t. However, it is of particular interest to determine the probability that an actively used system will fail during the time interval $[t, t + \Delta t]$, given that no failures occurred before time t. This measure represents the system **failure rate** (sometimes called the **hazard rate**), usually denoted by $\lambda(t)$ and can be mathematically expressed as

$$\lambda(t) = \frac{\int_t^{t+\Delta t} f(x)dx}{[(t + \Delta t) - t]R(t)} = \frac{\int_t^\infty f(x)dx - \int_{t+\Delta t}^\infty f(x)dx}{(t + \Delta t - t)R(t)} = \frac{R(t) - R(t + \Delta t)}{\Delta t R(t)} \tag{2.5}$$

Note that Equation 2.5 represents a conditional probability, the condition being the system experienced no failure prior t. That's why the denominator contains $R(t)$. If the observed time interval is infinitesimally small, the failure rate turns into the *instantaneous* failure rate. It can be calculated as the limit of $\lambda(t)$ as Δt approaches zero:

$$h(t) = \lim_{\Delta t \to 0} \frac{R(t) - R(t + \Delta t)}{\Delta t R(t)} \tag{2.6}$$

where $h(t)$ is the *hazard function*.

The failure rate $\lambda(t)$ is defined as the frequency with which a system fails, i.e. the number of failures per unit time. While any measure of time can be used, it is most commonly expressed in **Failures In Time (FIT)** which represents the number of failures in one billion (10^9) hours of system operation. Figure 2.3 shows the bathtub curve which, as said, depicts the evolution of the failure rate over a system lifetime.

The bathtub curve is a sum of three components: early life (infant mortality) failures, constant (random) failures and wear out failures. It can be divided into three characteristic phases or regions:

- Region I (*infant mortality*) - the early life of a system characterizes with high failure rate that sharply decreases. Defects related to the manufacturing process are the most common cause for the high failure rate during this phase.

- Region II (*useful life*) - following the infant mortality period, the failure rate stabilizes and remains constant during the most part of the system operational

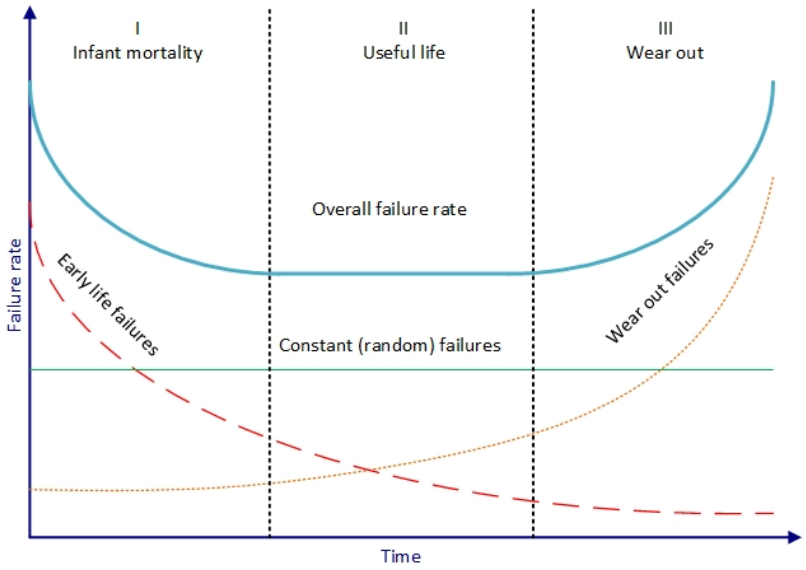


Figure 2.3: The bathtub curve.

life. The failures characteristic for this phase are random, usually caused by soft errors and dynamic variations (See Section 2.5).

- Region III (*wear out phase*) - as the system ages, the failure rate starts to increase. During this final phase of the system lifetime, components tend to fail due to exhaustion and fatigue.

Since the failure rate is constant during the useful life phase of the system ($\lambda(t) = const.$), it can be safely assumed that the random variable T (system lifetime) has an exponential distribution with parameter λ over this interval [47]. Therefore, the PDF of T would be $f(t) = \lambda e^{-\lambda t}$ and Equation 2.4 can be rewritten as

$$R(t) = \int_t^\infty f(x)dx = \int_t^\infty \lambda e^{-\lambda x}dx = e^{-\lambda t} \tag{2.7}$$

The expected value $E[T]$ is the weighted average of all possible values of T and it is given by

$$E[T] = \int_0^\infty tf(t)dt = \int_0^\infty t\lambda e^{-\lambda t}dt \tag{2.8}$$

Integrating by parts results in

$$E[T] = -[te^{-\lambda t}\Big|_0^\infty] + \int_0^\infty e^{-\lambda t}dt \tag{2.9}$$

By applying L'Hôpital's rule, the first addend from Equation 2.9 evaluates to 0 since $\lim_{t\to\infty} \frac{t}{e^{\lambda t}} = \lim_{t\to\infty} \frac{t'}{(e^{\lambda t})'} = \lim_{t\to\infty} \frac{1}{\lambda e^{\lambda t}} = 0$ and $\lim_{t\to 0} te^{-\lambda t} = 0$ whereas $e^{-\lambda t} = R(t)$ (see Equation 2.7). Therefore,

$$E[T] = \int_0^\infty R(t)dt \tag{2.10}$$

Alternatively, the second addend from Equation 2.9 can be easily transformed into $\frac{1}{\lambda}\int_0^\infty \lambda e^{-\lambda t}dt$. Recall that $\int_0^\infty \lambda e^{-\lambda t}dt$ is the integral of the PDF of T which must integrate to 1 (see Equation 2.2). Thus,

$$E[T] = \frac{1}{\lambda} \tag{2.11}$$

Equation 2.10 and Equation 2.11 represent what is known as **Mean Time To Failure (MTTF)**. It is a widely used dependability measure defined as the expected time of the occurrence of the first system failure [47]. For repairable systems, the measure **Mean Time To Repair (MTTR)** is commonly used to specify the average downtime, that is, the time period from the moment of failure to the restoration of the correct service. MTTR is usually expressed through the **repair rate** μ which represents the expected number of repairs per unit time. The relationship between MTTR and the repair rate is similar to the relationship between MTTF and the failure rate ($MTTR = 1/\mu$). MTTR reflects the *maintainability* of the system which is defined as the probability that a failed system will be repaired before time t [56].

Another frequently used dependability measure for repairable systems is the **Mean Time Between Failures (MTBF)**. It is defined as the average time of system operation between failures [64].

$$MTBF = \frac{Correct\ operation\ time}{Number\ of\ failures} \tag{2.12}$$

Assuming that a previously failed system is ideally repaired, the relationship between MTTF and MTBF can be approximated as follows:

$$MTBF = MTTF + MTTR \tag{2.13}$$

Sometimes the MTBF is misused as MTTF since MTTR is much shorter than MTTF ($MTTR << MTTF$). Indeed, the correct system operation is usually expected to last for years, whereas the repair time should not exceed few hours [64].

Reliability and maintainability are related through the concept of *availability*. Availability is defined as the probability that a system is operating correctly at any given time. It can be calculated as

$$A = \frac{MTTF}{MTTF + MTTR} = \frac{MTTF}{MTBF} \tag{2.14}$$

For non-repairable systems, the availability is equal to the reliability.

2.3.2 Resilience

With regard to computing systems, the term resilience has been commonly used for a long time as a synonym of fault tolerance. Unlike dependable computing, which is well-established and well-documented concept, resilient computing began to attract attention only quite recently. Although younger as a discipline, resilience is considered to be broader field than dependability by introducing awareness of **changes**.

A change can be classified according to its:

- *nature* into functional. environmental or technological;
- *prospect* into foreseen, foreseeable or unforeseen;
- *timing* into short term, medium term or long term.

It is very important to note that the changes can affect or modify the dependability threats either directly - by occurring in the system itself or indirectly - by occurring in its environment. Furthermore, it is possible that the changes themselves can be transformed into threats. In this regard, a change can be interpreted as a fault since it may lead to a deviation from the correct service (error). As previously stated, an error could result in a system failure. Therefore, the common thread binding the changes and the threats is the fact that both could be an initial cause for a failure. Section 2.5 discusses the potential failure origins in electronic systems.

Similar to the dependability means, Laprie in his paper [82] presents so-called **technologies for resilience**:

- *evolvability* - ability to successfully accommodate to changes;
- *assessability* - ability to retain the conception of justified confidence;
- *usability* - ability to utilize the ubiquitous character of the computing systems;
- *diversity* - ability to prevent vulnerabilities to become single points of failure.

Especially important for resilient computing systems is the capability to evolve, or to **adapt** to changes during operation time. A resilient system, which needs to be dependable in the first place, normally accommodates adequate dependability means. As descibed in Section 2.3.1, fault tolerance (and to some extent fault removal) is performed during execution time, whereas fault prevention and fault forecasting are mainly performed before the system launch. When facing a change, a resilient system needs to appropriately cope with it. Obviously, foreseen and foreseeable changes are easier to deal with than the unforeseeable ones. It is to anticipate that employed fault tolerance techniques of a dependable computing system would effectively tackle the expected faults (foreseen and foreseeable changes). However, during runtime, the system may also experience unexpected faults (unforeseeable changes) which would result in failure if not properly addressed. Therefore, by introducing an aspect of adaptivity to a dependable system, or more specifically, to its fault tolerance mechanisms, such system would have the ability to evolve, i.e. to retain its dependability

despite the changes. A convenient way to achieve this is to add a high degree of configurability to the components constituting the system, especially to the ones that implement the fault tolerance techniques.

2.4 Power Consumption Fundamentals

Historically, power consumption has been a deciding factor influencing the choice of technology for ICs production. Complementary Metal-Oxide-Semiconductor (CMOS) technology has been dominating the IC market without any serious competitors for more than three decades. However, as the MOSFET feature sizes entered the sub-micron region, power consumption has become a problem for CMOS circuits as well.

Generally speaking, the sources of power dissipation in CMOS circuits belong to one of the two major classes: **dynamic** dissipation $P_{dynamic}$ and **static** dissipation P_{static}. The total power dissipation of a circuit is equal to the sum of these two components:

$$P_{total} = P_{dynamic} + P_{static} \tag{2.15}$$

A circuit, or a circuit component consumes *instantaneous power* which can be calculated as the product of the current that flows through the component and the voltage across the component:

$$P(t) = I(t)V(t) \tag{2.16}$$

The *energy* consumed or stored by the component over time interval T can be derived by integrating the instantaneous power over T:

$$E = \int_0^T P(t)dt \tag{2.17}$$

Hence, the average power dissipated by the component over the considered interval is given by:

$$P_{avg} = \frac{E}{T} = \frac{1}{T}\int_0^T P(t)dt \tag{2.18}$$

2.4.1 Dynamic Power

When a circuit operates actively, it dissipates dynamic power. The greatest contributing factor to the dynamic power is the *switching power*, that is, the power necessary to change the logic states of a circuit component. Figure 2.4 shows a simple CMOS inverter driving a capacitive load. When the inverter input transitions from logical '1' to logical '0', the PMOS transistor turns ON and charges the output capacitance

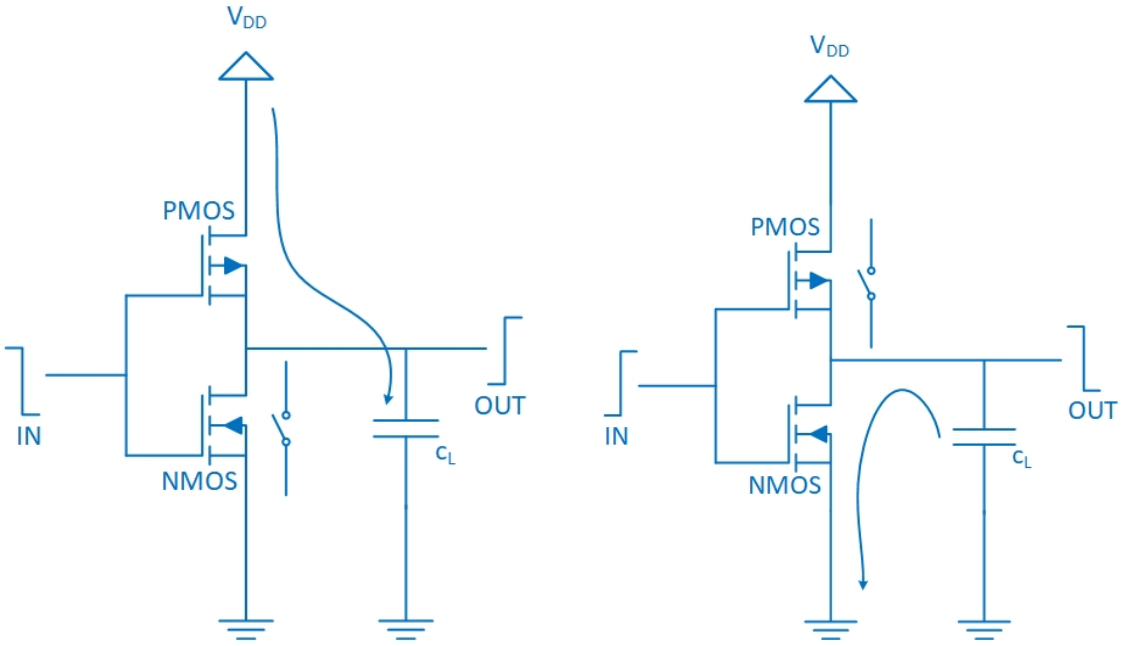
to V_{DD} (Figure 2.4(a)). The energy stored in the capacitor is

$$E_{C_L} = \int_0^{\infty} I_{V_{DD}}(t) V_{OUT} dt = \int_0^{\infty} C_L \frac{dV_{OUT}}{dt} V_{OUT} dt = C_L \int_0^{V_{DD}} V_{OUT} dV_{OUT} = \frac{1}{2} C_L V_{DD}^2 \tag{2.19}$$

whereas the energy pulled from the power supply is

$$E_{V_{DD}} = \int_0^{\infty} I_{V_{DD}}(t) V_{DD} dt = V_{DD} \int_0^{\infty} C \frac{dV_{OUT}}{dt} dt = C V_{DD} \int_0^{V_{DD}} dV_{OUT} = C V_{DD}^2 \tag{2.20}$$

which means that the output capacitance stores only half of the energy from the power supply. The other half is dissipated by the PMOS transistor in the form of heat. When the inverter input switches from logical '0' to logical '1', the PMOS transistor turns OFF while the NMOS transistor turns ON (Figure 2.4(b)). Such transition discharges the capacitor and the energy previously stored in the capacitor is dissipated by the NMOS transistor. The observed behaviour of the CMOS inverter can be generalized to any CMOS gate that drives an output capacitive load.



(a) Transition from logical '1' to logical '0'. (b) Transition from logical '0' logical '1'.

Figure 2.4: Switching power.

If a gate switches at a frequency f_{sw} during time interval T, its output capacitive load will be charged and discharged $f_{sw}T$ times. Therefore, the average switching

power consumed by the gate can be calculated according to Equation 2.18:

$$P_{sw} = \frac{1}{T} = \frac{f_{sw} T C V_{DD}^2}{T} = f_{sw} C V_{DD}^2 \tag{2.21}$$

Since the switching activities of most gates are less intensive compared to the switching activity of a clock signal, f_{sw} can be replaced with αf where α is the *activity factor* and f is the clock frequency. Thus, Equation 2.21 can be rewritten as

$$P_{sw} = \alpha f C V_{DD}^2 \tag{2.22}$$

The activity factor α actually represents the probability that a gate node changes its state between logical '0' and logical '1'. The value of α depends on the logical function implemented by the gate. For example, the activity factor of a clock is $\alpha = 1$ as it changes states every cycle. It has been empirically determined that the average activity factor of static CMOS gates is approximately 0.1 [135].

Despite the switching power, another component that contributes to the overall dynamic power is the *short-circuit* power. As illustrated in Figure 2.5, it is a power dissipated due to the the short-circuit current that flows during a brief time interval when both the PMOS and NMOS transistors are partially ON.

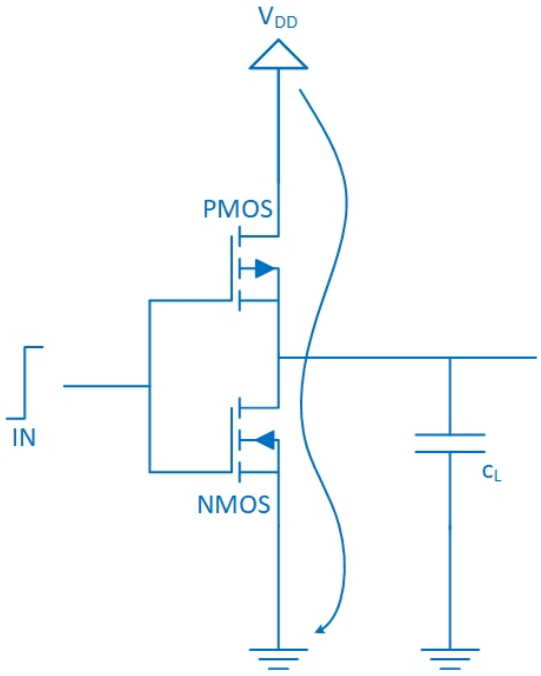


Figure 2.5: Short-circuit power.

The short-circuit power is directly proportional to the transition speed of the input signal. Therefore, unless the transition speed of the input is much slower compared to the output transition speed, the short-circuit power is only a small fraction (less than 10%) of the whole [135]. In such cases, the overall dynamic power comes down

to the switching power [53]

$$P_{dynamic} = P_{sw} = \alpha f C V_{DD}^2 \tag{2.23}$$

There are several different approaches for dynamic power reduction. Most of them consider the voltage and frequency terms of the Equation 2.23. Since the supply voltage has quadratic relation with the dynamic power, it is preferred to choose minimal value of V_{DD} in order to reduce the power consumption. For example, the system can be divided into multiple *voltage domains* where each domain will operate at its preferable supply voltage level. Furthermore, the supply voltage can be *dynamically scaled* according to the current operation mode - for example, a smart phone may operate at high supply voltage level while the user is working with a contactless payment application. On the other hand, if the smart phone is not actively used, the supply voltage can be lowered to a level that is optimized for receiving calls/messages/notifications only. In addition, if the frequency is adjusted proportionally to the supply voltage, a cubic power reduction can be achieved. Finally, since the clock characterizes with an extremely high activity factor, it would be greatly beneficial to prevent the unused components/blocks from switching in order to reduce dynamic power. This can be realized by employing an approach called *clock gating*, i.e. cutting off the clock when it is not required. Section 3.2 gives an overview of some proven techniques used for dynamic power reduction.

The low power techniques based on voltage and/or frequency scaling are highly efficient, especially regarding dynamic power savings [53]. However, lowering the supply voltage can have a negative impact on the system performance. Namely, the act of supply voltage falling below certain critical value may result in incorrect system operation due to slowing down of the transistor switching speed which leads to **timing errors** [123, 58]. A timing error occurs when a destination flip-flop fails to capture the correct data from a source flip-flop. Therefore, it is of key importance for systems that utilize such low power techniques to enable some form of timing error protection. A more detailed introduction to the timing errors is given in Section 2.5.3.

2.4.2 Static Power

In contrast to the dynamic power dissipation, static power is consumed even if there are no switching activities. It is, therefore, enough for a CMOS gate to be connected to a power supply in order to dissipate static power. Among several contributing factors to static power consumption, the most important is the *subthreshold leakage current* [15]. It is a drain-source current that "leaks" even when a transistor is supposed to be OFF. The value of the subthreshold leakage current can be approximated by the following equation [53]:

$$I_{sub} = \mu C_{ox} V_\theta^2 \frac{W}{L} e^{\frac{V_{GS}-V_{Th}}{nV_\theta}} \tag{2.24}$$

where μ is the carrier mobility, C_{ox} is the gate capacitance, V_θ is the thermal voltage kT/q (25.9 mV at room temperature), W and L are the dimensions (width and length) of the transistor, V_{GS} is the gate-source voltage, V_{Th} is the threshold voltage and the

parameter n is a function of the device fabrication process that ranges from 1.0 to 2.5.

Equation 2.24 states that the subthreshold leakage current increases exponentially with the difference between the gate-source voltage and the threshold voltage. Thus, in order to minimize the subthreshold leakage current and hence, the static power consumption, the difference $V_{GS}-V_{Th}$ should be kept as low as possible. Theoretically, this can be achieved by decreasing V_{GS} and/or by increasing V_{Th}. In practice, scaling of the supply voltage V_{DD} (and hence V_{GS}) has been performed to reduce the dynamic power consumption and to follow the Dennard scaling rules. Table 1.1 shows how the Intel processors supply voltages have been scaled across the processor generations. However, the threshold voltage V_{Th} had to be scaled at the same rate in order to maintain the performance improvement of at least 30% per technology generation [27]. In other words, increasing or keeping the threshold voltage constant while scaling the supply voltage would lead to significant performance loss. Therefore, voltage scaling achieves a trade-off between power consumption and performance: dynamic power is reduced by lowering V_{DD}, whereas V_{Th} is lowered to maintain performance which results in exponentially increased subthreshold leakage current. Unfortunately, CMOS design reached a point where supply and threshold voltages can no longer be reduced: V_{DD} must be kept above 0.5 V, otherwise the logic state consistency cannot be maintained; additionally, further V_{Th} reduction prevents complete switching-off of the transistors [60, 65].

In comparison to the dynamic power, static power was negligible in earlier technologies. However, starting from the 90 nm node, subthreshold leakage current, and hence, static power has been gradually turning into considerable source of power dissipation accounting for one-third [135] to one-half [27] of the overall chip power.

Subthreshold leakage current and static power in general are not frequency dependent (see Equation 2.24), and thus cannot be reduced by lowering the clock speed or by clock gating. While supply voltage scaling can reduce static power to some extent, power shut-off or *power gating* of the idle components/blocks can completely eliminate it.

2.5 Origins of Failures

There are numerous factors that can cause a computing system to fail. As previously stated, a failure occurs as a result of an error propagation to the system output, whereas the error itself is caused by activation of a fault. On the other hand, a change, or a *variation* in a form of disturbance within a system or in its environment may also lead to a failure. A question that naturally arises is: where do such changes and faults originate and what causes them? While the answer mostly depends on the fabrication process, it turns out that for the CMOS technology, the failure origins can be categorized into **spatial** and **temporal effects** on the one hand, and **dynamic variations** on the other hand [90, 32]. Figure 2.6 shows a categorization tree for the CMOS technology failure origins which are more comprehensively described in the following subsections.

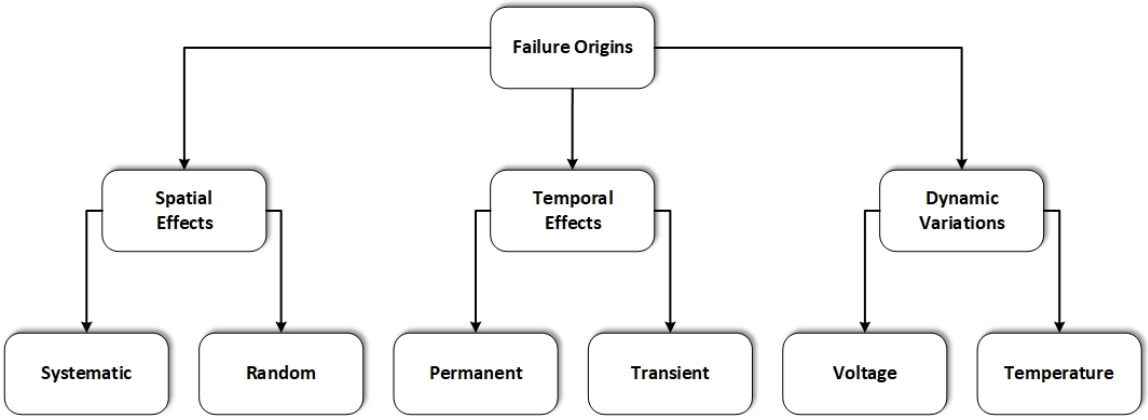


Figure 2.6: Failure origins categorization tree.

2.5.1 Spatial Effects

Also known as **process variations** or **defects**, the spatial effects are directly related to the design and manufacturing processes. They depend on factors like circuit layout, production environment and conditions, lithography variations etc. As the CMOS devices scale towards atomistic dimensions, the complexity of the fabrication process increases significantly. Consequently, the problem with the spatial effects gets worse. Typically, the spatial effects are visible immediately after the manufacturing process is finished and do not vary with time. Depending on the distance or the space between the observed defects, the spatial effects can be divided into *global* or *inter-die* and *local* or *intra-die* variations.

In general, global variations affect all transistors equally even if the distance that separates them is "long". The effect of a global variation is the same regardless of whether the transistors are located on the same or on a different die. In contrast, the effect of a local variation differs among identically designed transistors even if they are located on a same die. Most of the spatial effects occur as a result of local variations which can be further classified as *systematic* or *random*. Systematic effects are related to both the design and the manufacturing processes and may arise, for instance, due to layout characteristics or apparatus imperfections. Random effects manifest themselves as a mismatch between closely located identically designed transistors and occur as a result of an inherent randomness in the process.

Spatial effects lead to fixed changes in physical parameters of the device which, in turn, result in variations of its electrical parameters (e.g. threshold voltage or gate capacitance) [136]. As a consequence, the circuit performance is deteriorated. The spatial effects as such are out of the scope of this dissertation. However, protection against timing errors as an indirect effect of the process variations is thoroughly discussed later in the thesis.

2.5.2 Temporal Effects

In contrast to the spatial effects, the temporal effects vary over time and become visible when the circuit operates in a particular environment, under a given workload and over a certain period of time. Depending on how they impact the circuit, the temporal effects can be divided into *permanent* and *transient*.

Permanent effects inflict persistent damage to the circuit which usually leads to degraded performance, malfunction or failure. **Aging effects** are typical representatives of this category. The circuit gradually degrades as it gets older. Consequently, it usually experiences a permanent damage which in fact results in performance degradation followed by malfunction and eventually, a failure due to timing errors [118, 52]. As a result of the aggressive transistor scaling and the increasing electrical fields strength, the aging effects have become a serious problem in modern technologies. Some of the most commonly observed aging effects include: hot carrier injection (HCI), bias temperature instability (BTI), time-dependent dielectric breakdown (TDDB) and electromigration.

Unlike permanent effects, the transient effects disturb the circuit operation only temporarily. As soon as the source of the transient effect is eliminated, the circuit returns to its normal operation. There are two types of transient effects: *noise* and *electromagnetic interference (EMI)*. Noise is an undesired and random perturbation of a signal that originates from the circuit itself. In contrast, EMI is defined as the influence on a circuit that originates from external signals. The external signals are known as source signals, whereas the affected circuit is referred to as the victim circuit. The source signals can be man-made (deterministic) or natural (random). Man-made interference can be further categorized as functional [3] or accidental. For example, mobile devices or microwave ovens may produce accidental EMI, while on-chip crosstalk and simultaneous switching noise (SSN) are considered the most frequent sources of functional EMI. Finally, the most common sources of natural EMI are atmospheric noise (e.g. generated by lightnings during thunderstorms) and cosmic noise.

The *ionizing radiation* can be classified as a special type of EMI since it may originate from both natural and man-made sources. Semiconductor electronic circuits and systems are extremely vulnerable to ionizing radiation [17, 97]. During the last four decades, numerous reports on radiation-related problems of different type and severity have been published. Short retrospectives of some more significant incidents due to ionizing radiation can be found in [97, 127]. The next subsection discusses the effects of the radiation on the electronic devices in more detail.

Radiation Effects on Electronic Devices

The ionizing radiation environments contain subatomic particles or electromagnetic waves that can disrupt the electronic device operation. Indeed, phenomena such as Galactic Cosmic Rays (GCR) and Solar Particle Events (SPE) have sufficient energy to ionize the semiconductor material and to potentially cause an unwanted, non-destructive state change of the devices or even permanent circuit damage [17, 20].

[3] Occurs during normal operation of the circuits that generate the source signals.

From that aspect, the space environment is the most hostile for semiconductor technology. For example, the GCRs (coming from all directions outside the solar system) are composed of particles [4] and waves (gamma-, X-rays) that may reach extremely high energies (up to 10^{11} GeV) [17]. Furthermore, the SPEs contain protons, heavy ions and X-rays with energies of up to several GeV. Finally, the Earth's magnetic field has ability to trap charged particles in its magnetosphere [5] that possess energies between several tens to several hundreds MeV. These particles form the inner and outer Van Allen radiation belts around the Earth. In summary, the space environment is a serious threat to the electronic circuits embedded into satellites, spacecrafts and high-altitude aircrafts.

Of course, ionizing radiation is also present on the Earth itself. Alpha particles produced by radioactive contaminants in some chip packaging materials and the atmospheric neutrons are considered as major sources of terrestrial radiation. The atmospheric neutrons are created as a result of a collision between GCR particles and oxygen/nitrogen nuclei present in the Earth's atmosphere. Other sources of radiation at ground level include nuclear reactors, particle accelerators and specific industrial, research and medical applications.

In order to avoid radiation-induced faults, the electronic devices, especially those intended to operate in harsh environments, are usually designed using specific methods for **radiation hardening**. Such designed systems are referred to as **radiation-hardened** or simply **rad-hard** systems.

The electrical disturbances that disrupt the normal circuit operation as a result of a radiation event [6] are called **Single Event Effects (SEE)**. This is a general term that encompasses a whole class of destructive and non-destructive radiation-induced faults in electronic devices. SEEs can cause both transient and permanent errors. The transient errors, also known as **soft errors** are events which corrupt the data, but do not permanently damage the device. On the other hand, the **hard errors** are permanent and non-recoverable errors. Resilience to SEEs, especially to soft errors is a widely investigated topic. Since this dissertation deals with error resilient processor-based systems, a more detailed discussion with respect to soft errors is given in the next subsection.

Soft Errors

A soft error is actually an unpredictable, random error that occurs as a result of a single event effect. The error as such typically does not result in permanent damage of the device. Nevertheless, soft errors could be the reason for serious unwanted outcomes such as data corruption, malfunction or even a system crash. Both sequential and combinational logic are impacted by soft errors. Depending on the affected circuit type, different radiation hardening techniques are used for error detection, masking or correction. Recovery is possible by reset, power cycle (turning the device off and on) or by rewriting the storage element(s) if affected. The metric **Soft Error Rate**

[4] Mostly protons, but also alpha particles, heavy ions.

[5] Mainly electrons and protons.

[6] For example, strike of a highly energetic particle or electromagnetic wave.

(SER) is used for quantification of the influence of the soft errors on an electronic device. SER reflects the number of soft errors per unit time and is expressed in FITs.

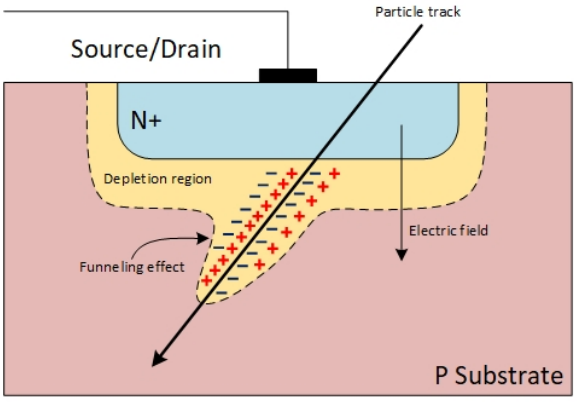


Figure 2.7: MOSFET hit by a highly energetic particle.

The impact of a highly energetic particle on a MOSFET in its source/drain region is illustrated in Figure 2.7. As a result of the ionization, electron-hole pairs are created along the particle track through the oxide and the semiconductor materials. Many of these charge carriers will recombine instantly. However, under influence of the electric field, the intensity of the drift current will increase. Hence, certain amount of charge will be collected in the depletion region which is one of the most sensitive segments of the MOSFET. Due to the impact, the depletion region is deformed into funnel-like shape. Several types of soft errors may arise if the collected charge in such sensitive circuit node exceeds a threshold level known as *critical charge* (Q_{crit}):

- **Single Event Upset (SEU)** occurs when the SEE results in an undesired state change (bit flip) of a storage element (e.g. a flip-flop or a SRAM cell). An upset of two or more storage elements is called **Multiple Cell Upset (MCU)**. If two or more bits of the same word are affected, the upset is referred to as **Multiple Bit Upset (MBU)**.
- **Single Event Transient (SET)** is a voltage or a current glitch observed at the output of a combinational logic gate. If a SET propagates and gets captured by a storage element, it turns into a SEU.
- **Single Event Functional Interrupt (SEFI)** causes termination of the normal device operation due to a bit flip in a control register or disturbance of other vital signals such as clocks or resets.
- **Single Event Latch-up (SEL)** occurs when a parasitic structure is formed in the device that creates a conducting path between the power supply source and the ground. Extremely high current will then flow between these two terminals which may permanently damage the device due to local overheating.

One of the most significant and most widely-used approaches for mitigation of soft errors is the **Triple Modular Redundancy (TMR)**. TMR is a straightforward,

efficient and very old technique first proposed by Von Neumann in the 1950s [131]. It is implemented by simple triplication and majority voting. The apparent downside of the TMR is the high overhead in terms of power, area and/or performance depending on the used redundancy type.

It is important to emphasize that not every soft error affects the system behaviour. Namely, there are several mechanisms that may impose a masking effect on an error. For example, if a SEU occurs late in the clock cycle, the corrupted value might not have enough time to reach the next stage flip-flops, especially if the path through the combinational logic is long. This is known as *temporal* masking. Furthermore, a corrupted value caused by an upset can be *logically* masked if it is, for example, OR-ed with logical '1' or AND-ed with logical '0'. Finally, in large and complex designs such as processors, *functional* masking is quite common due to the fact that not all functional units are used simultaneously. For instance, a SEU in a floating-point unit logic will not affect the result of an integer instruction etc.

2.5.3 Dynamic Variations

There are two types of dynamic variations: *voltage* and *temperature fluctuations*. The primary source of voltage fluctuations is referred to as power supply noise (PSN). PSN is defined as the fluctuation in the power supply signals, correlated to the current flow rate in a circuit under an influence of parasitic resistances and inductance of the power delivery network (PDN) [23]. In other words, the intensity of the PSN depends on the instantaneous current that flows through the PDN in the circuit. As it flows, this current passes through resistive and inductive parasitics in the circuit power grid and packaging that cause voltage drop (IR drop) and inductive noise (di/dt noise) [136]. The voltage ripple ΔV_{IRdrop} due to IR drop can be calculated by using Ohm's law:

$$\Delta V_{IRdrop} = R_{parasitic} \cdot i(t) \tag{2.25}$$

where $R_{parasitic}$ is the parasitic resistance and $i(t)$ is the instantaneous current. Furthermore, the current flow through the parasitic inductance $L_{parasitic}$ produces voltage ripple given by the following relationship:

$$\Delta V_{di/dt} = L_{parasitic} \cdot \frac{di}{dt} \tag{2.26}$$

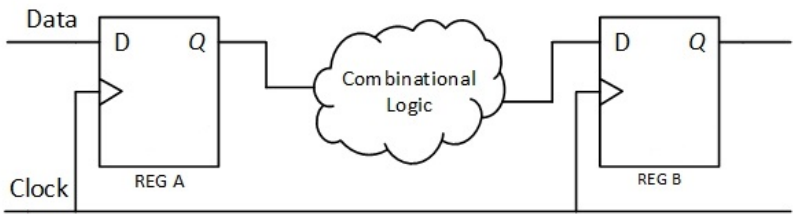
Equations 2.25 and 2.26 suggest that the voltage ripples due to IR drop and inductive noise are directly proportional to the current fluctuations which, in fact, are caused by rapid changes of the system switching activities. PSN actually occurs as a result of the combined impact of the both effects ($\Delta V_{IRdrop} + \Delta V_{di/dt}$) and has a negative influence on the functional and timing system performance. Concretely, the PSN causes voltage variations which may slow down the circuit and induce timing errors [123, 83].

The chip temperature is heavily influenced by the dissipated power. During normal operation, the chip dissipates heat that causes global, but also local temperature variations in the regions where the switching activities are more intensive. Such

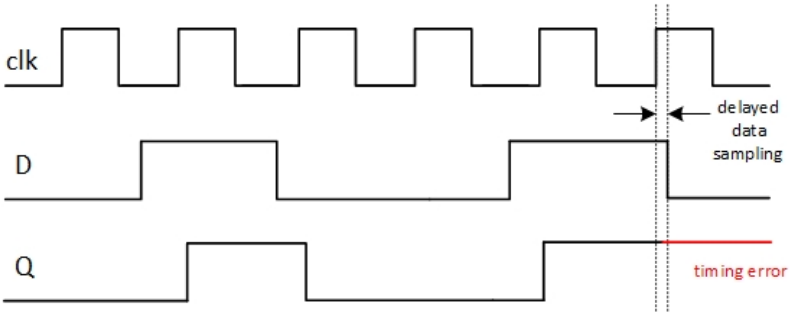
regions are called hot spots. Furthermore, any changes of the environmental temperature also result in global chip temperature fluctuations. Similarly to process and voltage variations as well as aging effects, the temperature variations may lead to timing errors due to reduced carrier mobility and increased interconnect resistance [136]. Since this thesis considers error resilience in processor-based systems, the next sub-subsection introduces the timing errors in more details.

Timing Errors

A typical computing system performs elementary operations on arbitrary data in the time period between two active clock edges. An illustration of this concept is shown in Figure 2.8(a). The process starts when the register REG A stores the incoming data on an active clock edge and immediately forwards it to the combinational logic. All operations to be performed in the combinational logic block between REG A and REG B on the data should finish before the next active clock edge. At this point, REG B stores the resultant data and forwards it to the next combinational logic block. If the clock edge fires before the resultant data stabilizes to the correct value, a timing error may occur (see Figure 2.8(b)).



(a) Data propagation between registers through combinational logic.



(b) Delayed data sampling by a register results in timing error.

Figure 2.8: Timing error due to slow data propagation.

Traditionally, a static approach known as *guard-banding* was the standardly used method for avoiding timing errors. Guard-banding consists of inserting additional safety voltage margins which guarantee correct operation under worst case conditions. As the worst case scenario occurs rather rarely, this approach led to unnecessarily high power consumption and significant energy waste. Therefore, it became preferable to optimize the power consumption adaptively, that is, by adjusting the supply voltage

according to the current operating/environmental conditions. However, as already pointed out in Subsection 2.4.1, utilization of techniques for low power operation based on supply voltage reduction may also lead to timing errors in case of voltage overscaling.

To prevent timing errors that occur as a result of voltage overscaling, dynamic approaches for protection against timing errors can be employed. Such techniques typically require support of additional hardware which clearly incurs power and area overheads. On the one hand, similarly to soft errors, timing errors can be detected or masked [37, 38]. On the other hand, unlike soft errors which are uniformly distributed in space and time, timing errors are easier to predict [22, 37].

Chapter 3

Related Work

The relevant research and industry communities have been actively working on reducing power consumption and increasing resilience for years. Indeed, the literature abounds with contributions that focus on resilience and power optimization, mainly as on two completely separate problematics. A plethora of diverse solutions at all layers of the system stack have been proposed: improved device materials, new technologies, design of hardened/fault-tolerant circuits and components, as well as development of power-aware architectures and software. However, due to the conflicting trade-off between the two metrics (explained in Section 1.2), closely-related works which consider joint power minimization and resilience improvement are quite difficult to find. A special issue on Low-Power Dependable Computing by IEEE in 2018 stresses out the criticality of developing such approaches in general [142].

This chapter reviews noteworthy publications which aim at improving power efficiency, [1] resilience [2] or both. Thereby, the focus is put on standard and adaptive techniques implemented at circuit and/or architecture level(s) in both single- and multiprocessor-based systems. Special attention is paid to solutions that use enhanced latches/flip-flops for those purposes. Such circuits are intended to be replaced with certain flip-flops/registers in the system during design time. An overview of strategies for integration and replacement of enhanced circuits in processor-based systems is also given.

The main points of the reviewed publications are briefly summarized. Additionally, the advantages and disadvantages of each approach are adequately underlined. Where applicable, the related works are compared against the solutions proposed in this dissertation. Finally, the contributions of the thesis that go beyond the state-of-the-art are recapitulated.

3.1 Improving Resilience

It was stated in Section 1.2 that increasing resilience requires intentional incorporation of redundancy in the system as well as that the hardware redundancy is widely

[1] Mainly regarding dynamic power.

[2] With respect to SEUs as well as to timing errors.

utilized for this purpose. Depending on the errors to be mitigated, different hardware redundancy schemes can be employed. For example, timing errors are mainly addressed by integration of redundant hardware that serves as a delay monitor able to perform either error **detection/correction** or error **prediction**. There are two major ways to realize delay monitors: as **replica circuits** or as **in situ monitors (ISMs)**. While the replica circuits are easier to implement, they are limited to addressing only timing errors which resulted from global variations. In contrast, the ISMs are capable to deal with both global as well as local variations. Furthermore, combining ISMs with structures able to mitigate e.g. soft errors could further increase the system resilience significantly. As a result, using ISMs is preferred over using replica circuits. Related works that rely on ISMs for resilience improvement are discussed in Subsection 3.1.1.

As previously explained, the timing errors occurrence is tightly connected to the supply voltage reduction. Therefore, quite often the techniques for low power based on voltage scaling need to be implemented along with techniques for timing error resilience. Section 3.3 elaborates on this in more detail.

The hardware redundancy for soft error mitigation is typically implemented as an *M-of-N* system, that is, as a system composed of N identical components in which at least M have to deliver correct service. All M components in the system share the same input(s). As a result of faults or variations however, the components may produce different outputs. Hence, the output of an *M-of-N* system is determined by a *majority voter*. This is a circuit which receives N inputs $x_1, x_2, ..., x_N$ and checks whether a majority M of them are equal. In such case, the value shared by the majority components is forwarded to the output. To avoid ambiguity, usually N is an odd number and $M = \lceil N/2 \rceil$ [79].

The *M-of-N* systems in the literature are also known as **N-Modular Redundant (NMR)** systems. Subsection 3.1.2 focuses on the fundamentals of NMR systems as well as on related works which utilize NMR structures as resilience mechanism.

3.1.1 In Situ Monitors (ISMs)

ISMs are intended to replace timing critical registers in the system during design time. Typically, ISMs are realized by augmenting regular registers with redundant latch/flip-flop-like structures. Functionality-wise, the ISMs are usually designed either to detect or to predict timing errors. The error detection approach usually requires less complexity, area and power. However, it introduces performance penalties as some error recovery mechanisms have to be employed to correct the detected errors. Often, such performance degradation is not acceptable. On the other hand, the error prediction approach requires more hardware per ISM, but is able to warn the system before the occurrence of a timing error and therefore, performance penalties are avoided.

A paper by Bowman et al. [28] proposed two latch-based ISM circuits for timing error detection: dynamic transition detector with a time-borrowing datapath latch (TDTB) and double-sampling static design with a time-borrowing datapath latch (DSTB). TDTB occupies smaller area and consumes less power in comparison to

DSTB, but requires higher design effort. On the other hand, DSTB is easier to implement and despite timing errors, it is also able to detect soft errors. The ISMs are evaluated on a test chip imitating a processor produced in 65 nm technology. When an error is detected, the system enters into recovery mode based on instruction replay at lower frequency. According to the authors, such designed system could save between 31% and 37% power at a cost of degraded performance.

Lin et al. introduced a self-checking ISM design for complete pipeline protection referred to as SETTOFF (Soft Error and Timing Error Tolerant Flip-Flop) [86, 87]. SETTOFF is capable of detecting timing errors, SEUs and SETs. Thereby, it is possible to correct a SEU on the fly by inverting the state of the flip-flop immediately after the error detection. For recovery from a timing error or from a SET, a pipeline replay at the architecture level is performed. The proposed approach was implemented in a 65 nm OpenRISC processor. It is reported by the authors that compared to traditional error protection techniques such as TMR, SETTOFF occupies over 30% less area and consumes 80% less power.

Soft error mitigation (SEM) as well as soft and timing error mitigation (STEM) are two ISM-based designs proposed by Avirneni et al. that rely on multiple data clocking [11, 10]. As suggested by the names of the ISMs, SEM tackles soft errors, while STEM addresses both soft and timing errors by means of error detection. SEM and STEM are both composed of three flip-flops where each flip-flop uses a different clock. The clocks operate at the same frequency but have different phases. Such approach could potentially put the flip-flops into a metastable state and thus, requires extra circuitry to deal with metastability. Additional buffers are also necessary to control the phase difference between the clock signals. Of course, insertion of additional circuitry results in larger area and power overheads. In a case of an error, SEM is able to restore correct operation without performance loss, whereas STEM requires invocation of error recovery procedure. Evaluation of the proposed ISMs is performed on a DLX processor implemented in 45 nm. No data regarding the cost of the ISMs in terms of area and power is provided by the authors.

A bit flipping ISM for timing error tolerance was presented by Valadimas et al. [130, 129]. When a timing error is detected, the ISM inverts its output value to perform error correction. It takes only one clock cycle to correct the timing error. During this period, to avoid state corruption, the system is stalled by means of clock gating. Since the approach suffers from metastability issues, incorporation of appropriate circuits to prevent the ISM from entering into metastable state is necessary. As to be expected, every insertion of extra hardware leads to increased area and power consumption. The proposed ISM was implemented into 65 nm MIPS processor for evaluation. In comparison to the non-ISM MIPS processor equivalent, the authors reported 4.5% higher power consumption and 2% area increase.

3.1.2 NMR Structures

The NMR structures perform **fault masking** rather than fault detection and correction. Such feature is of key importance for systems with tight timing constraints

where performance degradation due to error recovery is not acceptable. In general, an NMR system is able to mask $n = \lfloor N/2 \rfloor$ faults.

Assuming that the components in an NMR structure fail independently and permanently, i.e., if no component repair takes place, one can calculate the reliability of an NMR system R_{NMR} as the probability that at least M components function correctly at time t. If $R(t)$ denotes the reliability of a single component, then R_{NMR} can be calculated using the binomial distribution formula [79]:

$$R_{NMR}(t) = \sum_{i=M}^{N} \binom{N}{i} R(t)^i (1 - R(t))^{N-i} \tag{3.1}$$

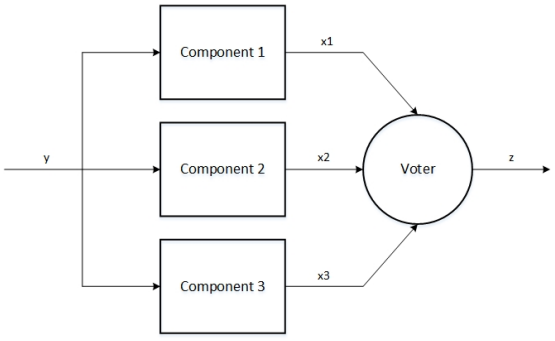
Alternatively, if n is the maximum number of faults that an NMR system is able to mask, then R_{NMR} can be calculated as:

$$R_{NMR}(t) = \sum_{j=0}^{n} \binom{N}{j} R(t)^{N-j} (1 - R(t))^j \tag{3.2}$$

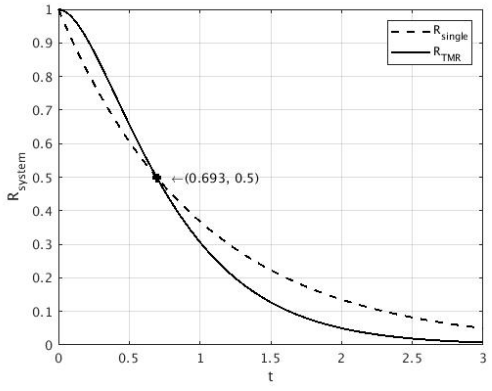
In contrast to the delay monitors which are typically implemented at circuit level, NMR structures (especially TMR), can be realized also at architecture as well as at system level.

Triple Modular Redundancy (TMR)

TMR is the most popular *M-of-N* (NMR) system which is composed of three identical components and at least one voter (Figure 3.1(a)). To put it formally, TMR is a *2-of-3* system where $N = 3$ and $M = 2$. Therefore, it requires a majority of two or three components with equal outputs that will be selected by the voter(s) as system outputs. If the TMR contains only one voter, then the voter becomes a single point of failure in the structure.



(a) Block diagram of a TMR structure.



(b) Reliability of a TMR structure compared to the reliability of a single component.

Figure 3.1: TMR structure.

Assuming an ideal voter, the reliability of a TMR system R_{TMR} can be easily calculated by replacing $N = 3$ and $M = 2$ in Equation 3.1 or $N = 3$ and $n = 1$ in Equation 3.2 respectively:

$$R_{TMR}(t) = \sum_{i=2}^{3} \binom{3}{i} R(t)^i (1-R(t))^{3-i} = \sum_{j=0}^{1} \binom{3}{j} R(t)^{3-j} (1-R(t))^j = 3R(t)^2 - 2R(t)^3 \tag{3.3}$$

For an exponentially distributed lifetime of a TMR system with constant failure rate $\lambda(t) = const.$, Equation 3.3 can be rewritten as

$$R_{TMR}(t) = 3e^{-2\lambda t} - 2e^{-3\lambda t} \tag{3.4}$$

whereas the MTTF for a TMR system $MTTF_{TMR}$ can be calculated based on Equation 2.10 as

$$MTTF_{TMR} = \int_0^\infty R_{TMR}(t) = \int_0^\infty (3e^{-2\lambda t} - 2e^{-3\lambda t})dt = \frac{5}{6\lambda} \tag{3.5}$$

Figure 3.1(b) shows a graphical comparison between the reliabilities of a single component and a TMR structure for $\lambda = 1$. The graph gives an impression that a TMR system is more reliable than a single component only up to a specific point in time - the intersection between the two curves. Additionally, Equation 3.5 suggests that even with an ideal voter, the MTTF of a TMR system is lower than the MTTF of a single component (see Equation 2.11). However, recall that Equations 3.4 and 3.5 were derived under assumption of independent and permanent faults. Of course, in reality this is not always the case. Quite frequently, due to transient faults, the components may experience temporal functionality disruption following which they continue normal operation. Hence, the reliability and the MTTF of a TMR system are significantly higher if transient faults are taken into account. Finally, while the voter in reality is never ideal and represents a single point of failure in a TMR structure, the probability of voter fault is quite low due to its low complexity and small area occupation.

TMR is the most widely used fault-tolerant technique and as previously emphasized, there are abundance of approaches in the literature which utilize some form of TMR at a certain layer of the system stack. While some works propose triplicated registers in the processor [54, 57], others introduce triplication of entire pipelines [126, 139]. Formation of core-level TMR or even NMR groups in multiprocessors systems containing more than two cores has been also a topic of investigation [119, 74].

3.1.3 Cross-Layer Approaches

Every computing system can be abstracted as a stack consisting of multiple layers. It turns out that such systems are susceptible to faults at every abstraction layer, even though it was formerly assumed that the underlying hardware of the most computing

devices [3] delivers correct service throughout the entire system lifetime [92, 91]. As a result of the aggressive technology miniaturization, this assumption does not hold anymore. An error produced at a lower layer could indeed propagate to the highest layer and cause a failure if not properly handled. A classic fault tolerant design approach would be to implement the error handling mechanisms on the same layer where the error was detected [31]. However, some studies [31, 44] argue that such approach introduces more drawbacks than advantages. Even though concentrating resilience mechanisms in the circuit and/or architecture layer simplifies the design of the higher layers, it also imposes high costs in terms of performance, area and power.

On the other hand, the systems that utilize cross-layer resilience techniques distribute the error handling responsibilities across the system stack. Such systems are considered capable to save power and chip area by handling less critical resilience threats at the higher layers. This is possible by activating/deactivating resilience mechanisms based on information provided at each layer of the system stack. A crucial prerequisite for cross-layer cooperation is considered the flow of some critical information between the layers. The right side of Figure 3.2 shows how the so-called *resilience tasks* [31] might be distributed among the layers. In the figure, the dots denote participation of the layers in a task, while the arrowed lines indicate the direction of the information flow.

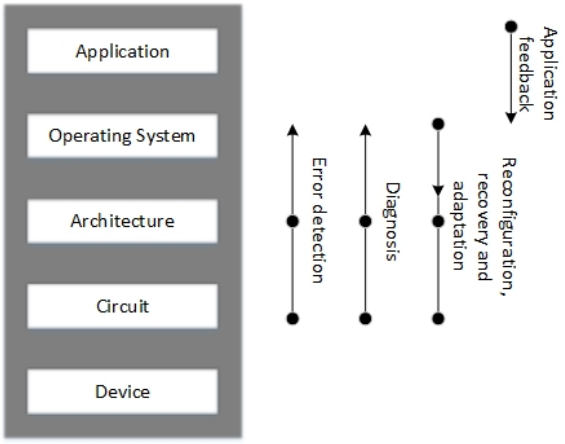


Figure 3.2: System stack and distribution of the resilience tasks across the layers. (Adapted from: Carter et al. [31])

Cross-layer data flow is illustrated by Mitra et al. [92, 91] with a simple experiment. The experiment is defined as follows: how to identify an optimal subset of a given flip-flop set in a particular design to be protected from soft errors using Built-in Soft Error Resilience (BISER)? BISER is a special flip-flop design technique which enables soft error correction with minimal area and power overheads [93]. The remaining flip-flops from the given set are to be protected using architecture level technique, concretely, single parity bit. Thereby, two sets of 50% and 90% respectively of all the flip-flops are randomly chosen to be protected against soft errors.

[3]Excluding safety- and mission-critical systems.

From the formed sets, subsets of 0%, 20%, 40%, 60%, 80% and 100% flip-flops are randomly chosen to be protected using single parity bit, whereas the remaining are protected using BISER. The reported results from the experiments show that for each case, the optimal design in terms of both area and power overheads, uses a mix of BISER and single parity bit. It is, however, assumed that the area and power costs of a BISER flip-flop are 2.3 times higher than the corresponding costs of an unprotected flip-flop. Despite focusing only on two layers, this experiment demonstrates the effectiveness of the cross-layer resilience approach. However, it addresses only one resilience mechanism - soft errors.

Cheng et al. presented CLEAR (Cross-Layer Exploration for Architecting Resilience) [35, 36] - the first cross-layer framework that spans across three or more layers. Radiation induced soft errors in processor cores are in the main focus of this work. The specific soft error types addressed are Silent Data Corruption (SDC) and Detected but Uncorrected Error (DUE). CLEAR is able to automatically explore numerous resilience techniques and to combine them across the system stack. Thereby, 798 technique combinations are reported in the paper [35]. From all combinations, the framework selects only those able to achieve required resilience level at minimal costs. CLEAR is evaluated on two different processor architectures: one simple in-order core and one complex out-of-order core. Each core executed 18 application benchmarks in their entirety.

One of the 798 combinations that were explored by CLEAR is reported as a highly promising approach. Concretely, this particular combination consists of selective circuit level hardening using LEAP-DICE [75] [4], logic layer parity checking and microarchitectural recovery. As an example, for SDC improvement of 50 times, this combination delivers 1.5 times and 1.2 times energy savings for the out-of-order and the in-order cores respectively compared to using LEAP-DICE only.

CLEAR is a very complex and extensive framework that sets new standards for development of future cross-layer resilience approaches. However, it concentrates solely on soft errors and observes their effects only on general-purpose processors.

Although showing encouraging results regarding resilience, [5] in general, the existing cross-layer approaches are often costly in terms of performance, area and power. A more detailed overview of the cross-layer resilience design methods is given in the publication [VKK17].

3.2 Improving Power Efficiency

Reducing power consumption, especially in mobile, handheld and battery-operated computing devices, has always been an important research topic. Nowadays, as a result of the challenges and trends discussed in Chapter 1, its relevance increased even further, making power optimization one of the most-widely investigated areas in modern computing. The extensive research conducted over the years led to development of numerous methods for reduction of both dynamic and static power. Some of

[4] Layout Design through Error-Aware Transistor Positioning-Dual Interlocked Storage Cell

[5] Mainly to soft errors.

the most commonly used techniques for dynamic power reduction in processor-based systems are **clock gating** and **dynamic voltage and frequency scaling (DVFS)**. The next subsections discuss these techniques in more details. While clock gating reduces exclusively dynamic power, DVFS is able to decrease also static power to some extent [99].

3.2.1 Clock Gating

Clock gating is one of the simplest and most mature low power techniques. It consists of deactivating the clock signals that synchronize registers within temporarily unused system components or blocks. Such action would actually prevent switching activities in the clock-gated registers, but also in the downstream combinational logic. Therefore, clock gating could be extremely effective in reducing dynamic power consumption. Recall that exactly the clock characterizes with the highest possible activity factor (see Section 2.4.1).

The implementation of clock gating is quite straightforward. In theory, logically ANDing the clock with an enable signal would be sufficient. In practice, however, the AND gate may produce glitches if the enable signal is not stable while the clock is active. A latch is typically used to avoid glitching (Figure 3.3).

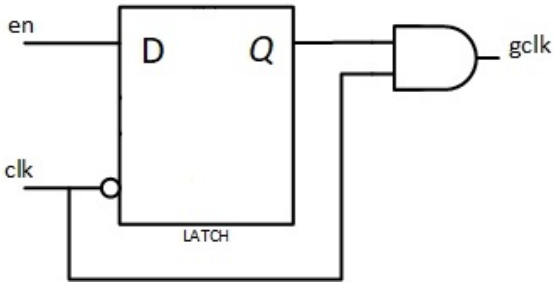


Figure 3.3: Typical clock gating implementation.

A study by Pokhrel [102] compares the power consumption of a chip with clock gating to an almost identical chip without clock gating. Thereby, power savings between 34% and 43% are reported, depending on the operation mode. Due to its implementation simplicity and high effectiveness, this technique is massively used in modern processor-based systems, regardless of their class or domain. However, utilizing clock gating makes sense only if there is a substantial quantity of idle components that can be disconnected from the clock for considerable time period. [6]

3.2.2 Dynamic Voltage and Frequency Scaling (DVFS)

DVFS is a well-established power reduction technique based on pre-characterized pairs of operating frequencies and corresponding supply voltage levels. Depending on the

[6] For example, it doesn't make sense to implement clock gating for a 1-bit register which is idle for several cycles only.

system workload or performance requirements, an adequate pair of voltage/frequency values is dynamically selected during runtime.

Processor-based systems started to utilize DVFS as a standard low power technique shortly after it was first proposed by Weiser et al [134]. As a matter of fact, most of the commercial processors manufactured in the last two decades have the ability to scale the supply voltage level within a strictly defined range. Note that, to avoid timing errors, frequency should be also adjusted accordingly together with voltage. For example, DVFS in Intel processors has been implemented through Enhanced SpeedStep technology [73]. It is a software-controlled technology that enables processors to select among multiple voltage and frequency pairs by writing into a special-purpose register [89]. As a result, supply voltage specifications for these processors are given in ranges instead of absolute numbers (see Table 1.1). Table 3.1 shows voltage/frequency pairs for the Intel Pentium M processor with Enhanced SpeedStep technology.

Table 3.1: Voltage/frequency pairs for implementation of DVFS in the Pentium M processor. (Source: Intel [89])

Voltage (V)	**Frequency (GHz)**
0.956	0.6
1.004	0.8
1.116	1.0
1.228	1.2
1.308	1.4
1.484	1.7

Utilizing DVFS is extremely practical when there are no demands for high performance computing or when a battery-operated processor-based system aims at saving energy. In such cases, the processor can select an appropriate pair of lower voltage/frequency operating points. Plenty of DVFS implementations for different processor-based systems can be found in the literature. Some of the reported power savings range from 12% in mixed criticality systems [68] to 34% in commercial processor-based systems [84].

Voltage/frequency pairs for DVFS are pre-characterized by worst-case guardbanding during design time. Therefore, DVFS is unable to adapt the system to dynamic variations in the environment nor the chip itself. This is especially relevant for contemporary processor-based systems implemented in most recent nanoscale technologies which are known to be highly susceptible to dynamic variations.

3.3 Improving Resilience and Power Efficiency

In Subsection 2.5.3 it was pointed out that dynamic variations may lead to timing errors as well as that the basic precondition for addressing timing errors is to constantly observe the system performance. For this purpose, usually hardware delay

monitors are used. Such circuits can be also practical when utilizing low power techniques based on voltage scaling, since reducing the supply voltage may as well result in timing errors.

The next subsection discusses **adaptive voltage/frequency scaling (AVFS)** - an advanced low power technique that strongly relies on hardware delay monitors. By tackling timing errors, delay monitors in AVFS-based systems have the potential to provide both power efficiency and a certain degree of resilience.

3.3.1 Adaptive Voltage/Frequency Scaling (AVFS)

Unlike DVFS, AVFS has a capability to dynamically adjust voltage/frequency levels in a system according to the current operating conditions. This is achieved by formation of a closed loop between the AVFS building blocks whose function is to observe the system performance and to act accordingly [132, 5]. Concretely, a closed AVFS loop is formed between four key components: *performance* or *delay monitor*, *AVFS controller*, *voltage regulator* and *clock divider*.

The **performance** or **delay monitor** is a specially-designed circuit with two functions. First, it constantly observes system performance in terms of timing. By monitoring circuits delay, [7] it detects/predicts timing errors. Second, it feeds the AVFS controller with information regarding the performance. Based on the feedback from the delay monitor, the **AVFS controller** decides whether it should decrease, increase or keep the same voltage/frequency levels. Finally, the **voltage regulator** and the **clock divider** driven by the AVFS controller simply set supply voltage/operating frequency respectively to the adequate values. In case of detected timing errors, the AVFS controller invokes appropriate recovery procedure.

Another notable difference between DVFS and AVFS is the fact that when using DVFS, frequency is always scaled together with voltage, which doesn't have to be the case if AVFS is used. Within such closed-loop scheme, timing errors are dynamically tackled. [8] Therefore, it is possible to simultaneously save power and preserve system performance by scaling only the voltage and keeping the frequency constant. This approach is known as **Adaptive Voltage Scaling (AVS)**. Of course, some performance penalties have to be paid for error recovery if the approach is implemented to detect timing errors. On the other hand, the prediction approach allows the system to avoid timing errors by dynamically increasing supply voltage level when necessary.

There are two main approaches for practical implementation of AVFS classified according to the performance monitor type. As previously stated, delay monitors can rely either on replica circuits or on in situ monitors.

AVFS with Replica Circuits

This approach uses replica circuits as delay monitors which operate under the same conditions as the actual circuits in order to mimic the most critical paths within the system (Figure 3.4). Adjustment of supply voltage/operating frequency is, thus,

[7] Delay may vary depending on several factors - dynamic, PVTA variations, faults etc.

[8] Either detected or predicted.

performed by measuring speed of the replica path instead of the real critical path [30, 49]. However, reproducing the exact layouts of critical paths is difficult. Therefore, simple and idealized circuits like inverter chains are used as replicas.

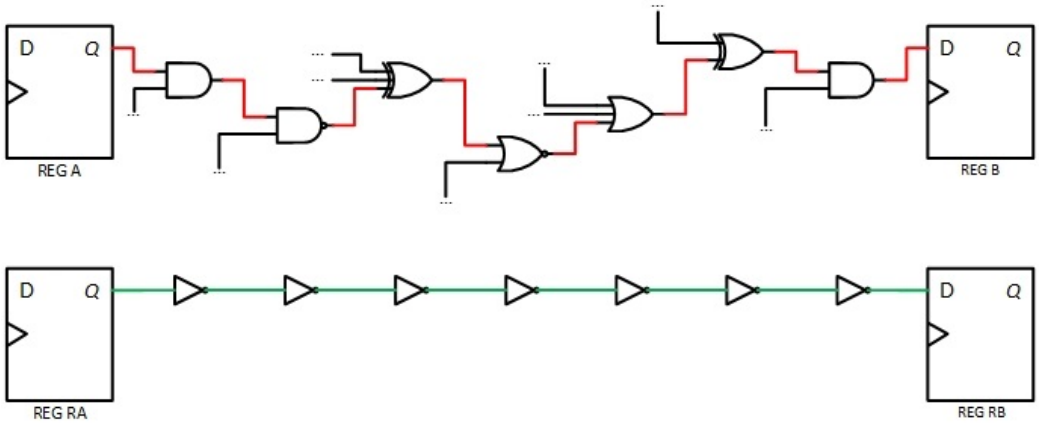


Figure 3.4: The replica path between registers RA and RB mimics the actual critical path between registers A and B.

Implementing AVFS using this approach is convenient because integrating a replica circuit does not require large design effort. Furthermore, it is suitable for detecting global variations within the core. To avoid timing errors, the replica circuit is usually designed to be slower (or longer) than the actual critical path. On the other hand, this approach is unable to deal with local variations. The impact of local variations on replica circuits differs from the impact on actual system components. Such inconsistency would eventually lead to inappropriate voltage/frequency management. Therefore, an approach that will be able to handle both global and local variations is preferred.

AVFS with In Situ Monitors

Since ISMs are located within the actual circuit, they have ability to keep track also of local variations. This is achieved either by timing error detection/correction or timing error prediction. The prediction-based approach is more practical because it allows timing error avoidance and uninterrupted system operation. Namely, after reception of a warning signal for a potential timing error, the system could take adequate action through an AVFS controller to prevent the error from occurring. [9]

One of the first and most significant works which rely on ISM-based AVS is **Razor** [51, 50]. It consists of replacing flip-flops which lie on critical paths with so-called Razor flip-flops. A Razor flip-flop is actually an ISM composed of a main (regular) flip-flop augmented with a shadow latch connected in parallel (Figure 3.5). The outputs of the regular flip-flop and the shadow latch are compared by a comparator. While the regular flip-flop samples data input signals on an active clock edge, the latch is

[9] For example, it can increase the supply voltage level.

transparent during the entire clock duty cycle. Therefore, if an input data transition arrives early enough to meet the flip-flop setup time, both the regular flip-flop and the shadow latch hold the same value which indicates correct operation. However, if an input data transition arrives late to meet the regular flip-flop setup time, the shadow latch still captures the correct data value. In this case, the compared output values differ and as a result, a timing error is signalled by the comparator. The Razor flip-flop then performs circuit-level error correction by restoring the correct data value from the shadow latch into the main flip-flop. After occurrence of a timing error, the system state is corrupted and recovery has to be invoked. Power saving in systems using Razor is achieved by adjusting the supply voltage level according to the timing error rate.

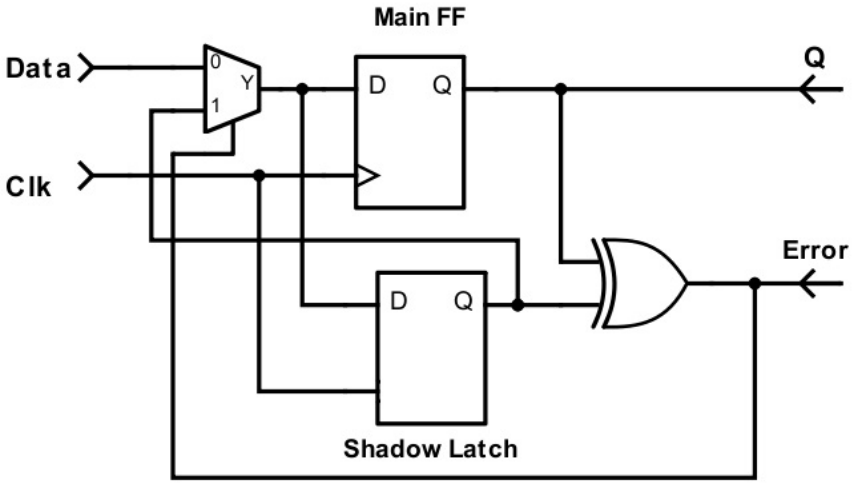


Figure 3.5: Schematic diagram of the Razor flip-flop. (Adapted from: Ersnt et al. [51])

The Razor concept made a major breakthrough towards dealing with both global and local variations as well as towards optimizing power consumption using ISM-based AVS. It was evaluated on a 64-bit Alpha processor produced in 180 nm technology. Thereby, the authors report energy savings from 12% to 38% in comparison to a system using fully-margined DVFS without Razor support. However, Razor suffers from several drawbacks: since it relies only on timing error detection, it requires recovery mechanisms which introduce performance degradation as well as power and complexity overheads. Furthermore, during longer periods without detected timing errors, the supply voltage could be overscaled which might lead to increased timing error rate later on. Additional problems related to Razor are metastability and short-path constraint. The short-path problem occurs when the Razor flip-flop cannot distinguish between a late transition from the previous clock cycle and a fast transition from the current clock cycle. In such case, the Razor flip-flop alerts a false timing error that triggers unnecessary recovery computations. This results in significantly reduced performance and increased power consumption. Additionally, a recovery

computation after a false error will result in signalization of false error again, thus causing the system to enter into an infinite loop. To handle the short-path constraint, buffers are inserted as a workaround. However, inserting buffers further increases area and power consumption which, in fact, conflicts with the primary goal of this work.

The potential of ISMs was recognized promptly and a substantial amount of research work has been conducted to improve the Razor design as well as to overcome its limitations. In this direction, Das et al. introduced RazorII [43] - a simplified ISM implementation which eliminates the metastability problem and performs error detection exclusively on circuit level. The error recovery process is left to be handled by the architectural layer using checkpointing and instruction replay. Similarly to Razor, the supply voltage level is dynamically adjusted according to the rate of detected timing errors. In addition, RazorII is capable of detecting also SEUs. Evaluation and testing of the proposed approach is performed on a 64-bit Alpha processor manufactured in 130 nm technology. The authors report energy savings of approximately 33% compared to a fully-margined non-RazorII DVFS system. Although RazorII can be considered a big step forward with respect to Razor, it still suffers from disadvantages specific to its predecessor - power and complexity overhead caused by buffers necessary to overcome the short-path constraint and performance degradation resulting from the error recovery process.

Bowman et al. [29] proposed an ISM-based approach for implementation of a processor system that is resilient to dynamic variations and hence, to timing errors. The approach relies on an ISM referred to as Error Detection Sequential design (EDS) and utilizes adaptive frequency scaling (AFS) as a low power technique. Besides reducing clock speed when necessary to avoid timing errors as well as to save power, the system is also capable to dynamically increase frequency to maximize throughput. When the EDS ISM detects a timing error, the system performs recovery process based on instruction replay at half of the operating frequency. Evaluation of the approach is performed on a LEON3 general-purpose core manufactured in 45 nm technology. The main advantage of this work in comparison to Razor is the drastic alleviation of the metastability problem. In comparison to a conventional, non-EDS system, energy reduction of 22% at equal throughput or 41% greater throughput at equal energy are reported.

Wirnshofer introduced a timing error predicting ISM for AVS implementation referred to as Pre-Error Flip-Flop (PEFF) [136]. Unlike the previously discussed error detection-oriented, Razor-like designs, the Pre-Error approach is able to detect late, but still non-erroneous data transitions (*pre-errors*). The PEFF issues a warning signal to alert the system when a pre-error is detected (Figure 3.6). Hence, additional hardware overhead and performance penalties introduced by error recovery computations are avoided. This makes the PEFF suitable for real-time applications.

A data transition is considered a pre-error if it occurs during a period called *pre-error detection window*. This is a time interval before the active clock edge. It is crucial to pick a length for the pre-error detection window as accurate and as robust as possible. The author provides a comprehensive analysis regarding the choice of a detection window length and how it affects the entire approach. In essence, longer detection windows result in higher pre-error rates and lower power savings, while

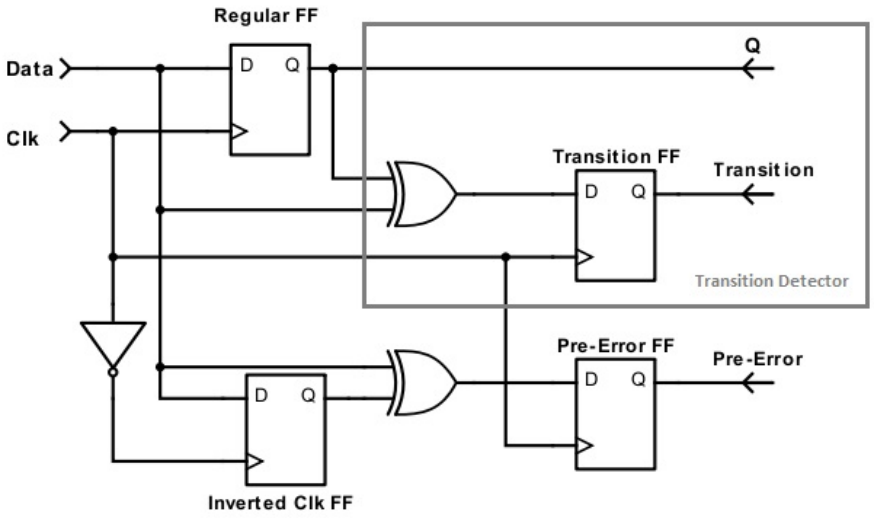


Figure 3.6: Schematic diagram of the Pre-Error flip-flop. (Adapted from: Wirnshofer [136])

shorter detection windows increase power savings, but also increase the probability of getting a timing error. [10] To define detection window length, the author exploits the clock duty cycle which means that a detection window starts with the alternative clock edge. Changing duty cycle is, however, possible only if the alternative clock edge does not trigger logic events in the system. In this direction, Shan et al. [116] proposed a timing error predicting ISM for near-threshold AVS with tunable pre-error detection window (Figure 3.7). The approach has been evaluated on a SoC chip implemented in 40 nm CMOS process.

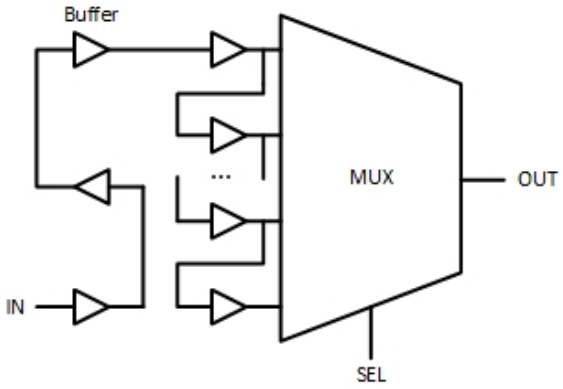


Figure 3.7: Tunable pre-error detection window circuit. (Adapted from: Shan et al. [116])

Voltage scaling in a system containing PEFF is performed based on pre-error rate. When a pre-error rate is zero or near-zero, the AVS controller can reduce the

[10] Since the number of detected pre-errors decreases.

supply voltage. On the other hand, high pre-error rate requires increasing of the voltage level. If the pre-error rate keeps some medium value, the voltage can be held constant. However, since voltage tuning is performed while the system is in operation, idle or low-activity periods might lead to excessive voltage decrease. This would most likely result in increased timing error rate later when the system activity intensifies. To prevent the voltage overscaling problem from occurring, [11] a so called transition detector is designed to observe the circuit activity. The transition detector is an integral part of the Pre-Error Flip-Flop and plays an important role in the voltage scaling process.

As shown in Figure 3.6, besides the Regular flip-flop, the proposed design contains three additional D flip-flops (Inverted Clk FF, Transition FF and Pre-Error FF). The Inverted Clk FF is a negative edge-triggered flip-flop and it is crucial for providing pre-error detection window. The Transition FF monitors the input data activity and observes whether a state change has occurred during the last clock cycle. If yes, the Transition output is set to logical '1'. Finally, the Pre-Error FF is responsible for checking if a transition took place during the pre-error detection window. In such case, the Pre-Error output is set to logical '1'. Figure 3.8 illustrates how the Transition and Pre-Error outputs are generated.

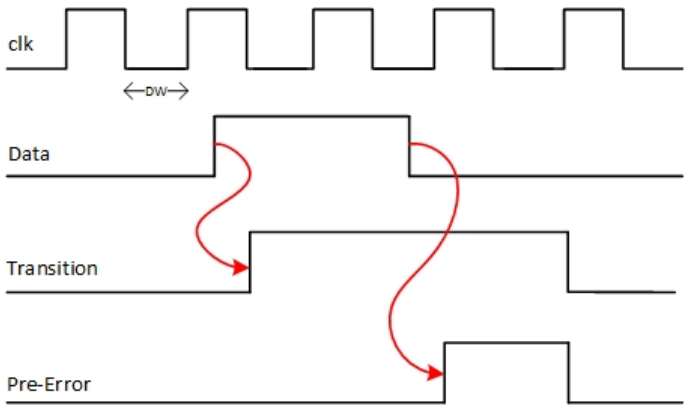


Figure 3.8: Generation of Transition and Pre-Error Outputs. DW designates the pre-error detection window.

Unfortunately, the PEFF design has not been evaluated on complex systems such as processors or SoCs. Instead, implementation and testing has been performed on pipelined multiplier and discrete cosine transform (DCT) circuits. Up to 36% power savings are reported at the expense of minimal overhead due to additional hardware necessary to build the ISM and the AVS controller. Moreover, performance is preserved as timing errors are avoided. However, the PEFF, as proposed by Wirnshofer is able to deal only with timing errors.

[11] This problem is specific to Razor-like approaches.

3.3.2 Combinations of Techniques

Very often a combination of two or more techniques is necessary in order to get resilience and power efficiency improvement. For example, systems intended to operate in hostile environments such as space are designed with error resilience in mind and fault tolerance is crucial requirement here. However, power efficiency needs to be addressed as well since such systems have to be powered by batteries or photovoltaic cells. It is interesting to note that high performance requirements of modern applications has "forced" the otherwise conservative space industry to switch to multiprocessor systems. For example, the dual-core GR712RC system has been used by the European Space Agency (ESA) for high reliability space applications [6]. Recently, the more advanced quad-core GR740 system has been introduced as an archetype of the European Next Generation Microprocessor (NGMP) [62]. Protection against radiation-induced soft errors in these designs comes down to register triplication and/or to using ECC codes on all on-chip RAM blocks which often significantly increases the area and power overheads. To reduce power consumption, clock-gating units are implemented, thus idle cores can be put to sleep mode. However, use of advanced techniques such as AVFS and ability of the design to adapt to the actual conditions in general seem to be missing.

A great deal of works are addressing resilience and power efficiency by combining DVFS and application task mapping [114, 42, 111, 18]. A comprehensive overview of such works is given by Yari-Karin et al. [140].

An approach which combines chip-level AVS and timing error-predicting flip-flops in a quadcore multiprocessor was presented by Sato et al. [112]. While notable power savings are reported by the authors, regarding error resilience, only timing errors are taken into account.

Srinivasan et al. consider dynamic core reconfiguration in Asymmetric Multicore Processors (AMPs) from perspectives of both susceptibility to soft errors and power efficiency [122]. The idea is to find the optimal configuration of a specific core which would result in best trade-off between soft error vulnerability and power consumption for the application executing on that core. Along with soft error-aware core reconfiguration, DVFS and power gating are used as low power techniques.

A power-robust quad-core multiprocessor system for space applications is introduced by Simevski et al. [121]. The system is centered around a multiprocessor framework that enables multimodal operation for dynamic adaptation to the requirements regarding resilience, lifetime and power consumption. Besides a default multiprocessor operation mode in which the cores operate independently, the framework implements two additional operation modes: *de-stress* and *fault-tolerant* mode. In de-stress mode only one core is active while the others are idle. This mode enables reduced power consumption and increased system lifetime. The authors report core-level AVS as a major power optimization technique despite the fact that the implementation presented in this work lacks a typical AVS loop. Instead, the control is transferred to the software layers. System resilience is provided by switching into fault-tolerant mode and forming core-level DMR, TMR or QMR groups. However,

error protection is missing in scenarios where the system is operating in default or in de-stress mode.

3.4 Design and Implementation Strategies

In order to produce more resilient chips, it became necessary for the conventional ASIC to undergo specific modifications. For example, the systems for space applications include standard fault-tolerant techniques known as *Radiation Hardening By Design* (RHBD) in the design flow [113]. RHBD usually consists of making registers robust to radiation by employing TMR and/or making power grids stronger [121]. This approach, however, results in dramatically increased overheads w.r.t. both area and power consumption.

On the other hand, power optimization is traditionally considered at RTL level, where most architectural decisions have already been made [12] [96]. Nevertheless, it is frequently necessary to specify power-related implementation details requiring semantics unsupported by hardware description languages (HDLs). Such power-related specifications are referred to as *power intents* - often included in power-aware design flows. Power intents are used to describe power domains, voltages, power cells [13] and power rail connections within the design [99]. On the downside, implementation of some advanced low power techniques such as AVFS is not possible using power intents only, as AVFS requires realization of closed control loop in hardware.

The previously-published works related to modified ASIC design flow regarding resilience or power efficiency move generally in two distinct directions: towards mitigation of soft errors or towards mitigation of timing errors (due to supply voltage scaling). However, the common thread of all works is the usage of particular "enhanced circuitry" that needs to be placed in the system, or more often, replaced with some of the system's flip-flop/registers in order to achieve the desired goal. In the former case, TMR flip-flops are used as soft error protection circuits, whereas in the latter case, timing errors are addressed by employing ISMs. An overview of related works for both TMR- and ISM-oriented approaches is given in Subsection 3.4.1 and 3.4.2 respectively.

3.4.1 TMR-Oriented Approaches

A method for improving design robustness to soft errors without deviating from standard ASIC design flow was proposed by Petrovic et al. [101]. The method relies on specially designed TMR flip-flops that use standard, non-hardened flip-flops from a technology library as building blocks. Evaluation is performed on a test chip containing shift registers. As radiation hardness is the main concern in this work, no data regarding power/area overhead is available.

Ruano et al. [106] introduced a methodology for automatic, selective TMR flip-flop insertion to reduce circuit complexity while meeting the specified reliability level.

[12] For example, usage of clock gating.

[13] For example switches, isolation, level shifters, etc.

By analysing circuit's topology and by using an iterative optimization algorithm, a balance is maintained between two main design constraints - the reliability level and the area cost. Unfortunately, the circuit type used for evaluation of the proposed methodology is not revealed.

A similar approach is published by Torvi et al. [125]. Namely, a framework for selective hardening to improve FIT rate is formulated as a linear optimization problem. An unspecified IP core used extensively in safety-critical automotive microcontrollers serves as a test vehicle for experiments. The authors report a soft error robustness improvement of 32% at a price of 2% area increase.

Finally, a paper by Polian et al. [103] gives a comprehensive overview of techniques for selective hardening. The obvious drawback of all these approaches is not taking the power consumption into account.

3.4.2 ISM-Oriented Approaches

ISMs can be placed at intermediate points or at end points of critical paths. To mitigate variation- or aging-induced timing errors more efficiently, ISMs are frequently placed at intermediate points. The internal nodes are shared between more critical paths, thus experience more transitions and higher activity compared to the end nodes. Therefore, delay degradation can be detected faster [108]. On the other hand, works that aim at employing some sort of adaptive regulation management in the system usually place monitors at the end of critical paths.

ISMs at Intermediate Points

Insertion of ISMs at intermediate points was first proposed by Lai et al. [81]. The goal is to reduce the number of ISMs in a design while efficient system health tracking can still be performed. Linear Programming (LP) is used to solve the ISM location selection problem. The ISM insertion is performed after the place and route phase. An evaluation using benchmarks for commercial processors showed that the total number of ISMs can be reduced by almost an order of magnitude if inserted at intermediate nodes. This comes at a price of additional 6% to 14% power overhead in comparison to the baseline designs.

Balef et al. [3, 19] propose an insertion flow for in situ monitoring of delay degradation based on dynamic monitor excitation. Thus, graph-based Static Timing Analysis (STA) is performed to determine the most critical paths. Similarly to the approach proposed by Lai et al. [81], ISMs are inserted at intermediate points along those paths. The number of ISMs is determined during synthesis, but the actual insertion is performed after place and route. The evaluation results performed on ARM Cortex M0 processor showed that the number of ISMs can be decreased elevenfold compared to end point approaches. On the downside, power and area overheads compared to the baseline designs can exceed 10% and 11% respectively.

ISMs at End Points

Of course, aging-/variation induced delay degradation monitoring can be successfully implemented also by placing ISMs at the end of critical paths. A netlist-level sensor insertion flow for in situ monitoring of timing slacks in SoCs presented by Sadi et al. [109] confirms the efficiency of such approach. However, the proposed method is not evaluated on entire system, rather individual processor units are used instead.

Obviously, just like TMR flip-flops, ISMs induce significant area and power overhead in the design unless a proper strategy for insertion is not followed. In this regard, a selective replacement method for timing error predicting flip-flops named Canary FFs is introduced by Kunitake et al. [80]. To reduce area overhead, all flip-flops at the end of critical paths that violate pre-defined timing constraints are replaced with Canary FFs following synthesis and STA. An evaluation of the method on commercial processors showed area overhead between 2% and 12% in comparison to baseline designs.

A power-aware, better-than-worst-case ISM insertion flow based on timing speculation is published by Londoño et al. [88]. ISMs are inserted at end points in a design following post-layout STA and critical path extraction. Although evaluated on a 32-bit multiplier only, power savings of 49% at a price of 5% performance loss has been achieved.

Another ISM insertion flow that aims at reducing power overhead caused by ISMs is presented by Huang et al. [67]. This approach also suggests that ISMs are to be placed at end points. Evaluation is performed on LEON3 processor and by replacing only 20% of critical flip-flops in the processor's Integer Unit (IU), power reduction of 7% to 18% is reported. The performance degradation in this case is 16%. Details regarding the ISM insertion flow are omitted.

3.5 Progress Beyond the State-of-the-Art

The thesis contributions were already briefly highlighted in Subsection 1.4.2. This section elaborates what makes the presented contributions go a step further beyond the current state-of-the-art.

As opposed to existing cross-layer approaches which focus exclusively on soft error resilience optimization, this dissertation proposes a cross-layer framework capable also of saving power on top of providing resilience to both soft and timing errors in processor-based systems. Several key features related to the structure of the framework or to its practical implementation may be pinpointed as original thesis contributions (see also the List of Own Publications on page xv):

- at the heart of the proposed framework lies the SWIELD multimodal flip-flop implemented at circuit layer of the system stack. Complete description of the SWIELD FF can be found in Section 4.1 as well as in the publication [VKK19]. Unlike solutions proposed in previously published related works, the SWIELD FF can be used as both ISM and TMR flip-flop. Structure-wise, a baseline

for the SWIELD FF design is an ISM concept similar to the Pre-Error Flip-Flop [136]. The potential of the redundant hardware within the ISM is fully utilized in order to build a configurable flip-flop with improved and enhanced functionalities. Such a design enables the cross-layer framework to synergistically combine resilience- and power-aware techniques such as AVFS, selective circuit level TMR and clock gating while the system is active. Metaphorically speaking, the SWIELD FF cuts the additional voltage safety margins like a SWord and protects against errors like a shIELD, and hence the inspiration for its name;

- a flexible framework controller enables simple and centralized management of the system operation regarding resilience and power consumption optimization. Logically, the SOMU is located at architectural layer and it serves as a bridge between SWIELD FFs and the higher layers of the system stack. Thus, all framework-related actions in hardware required by the running applications or the OS are performed fast and efficiently. Whenever necessary, the SOMU can easily switch between SWIELD FFs operation modes and if needed, take additional measures such as enabling/disabling frequency scaling or clock gating. Detailed description of the SOMU can be found in Section 4.2 and in the publications [VKK19, VHKK20].

- the library of functions and procedures FFL is the only part of the cross-layer framework implemented in software. FFL allows extremely easy manipulation of the framework functionalities. Namely, by simple call of an appropriate function or procedure, the programmer is able to set a plethora of configurable parameters, get their current status or adequately switch between SWIELD FFs operation modes. An FFL function/procedure call actually instructs the SOMU to take necessary framework-related actions in hardware. Depending on the function/procedure type, the action may be performed either at architecture or at circuit layer of the system stack. The FFL is presented in Section 4.3 and in the publication [VHKK21b].

- a simple and convenient strategy for integration of the cross-layer framework in processor-based systems is presented in Section 5.1 and in the publication [VHKK21a]. The integration strategy fits well to the standard ASIC design flow and obviously refers only to the hardware constituents of the framework (SWIELD FFs and SOMU). In contrast to the approaches discussed in Section 3.4 which are either ISM- or TMR-oriented and often evaluated only on individual hardware components rather than on entire systems, the proposed strategy can be easily applied to any complex processor-based system, both single- and multicore.

A key advantage of the proposed cross-layer framework is its complete independence on type or architecture of the "host" processor-based system. In other words, by following the integration strategy, it is possible to implement the cross-layer framework in any processor-based system that contains general-purpose core(s). Com-

mercial processors can be also utilized. The only precondition is availability of the processor's RTL code.

Finally, it is important to note that the number of cores in a multiprocessor system does not make the framework integration process more complicated. A successful integration procedure for a single core is simply mirrored on each core in a multiprocessor. Hence, the number of cores in a system is completely irrelevant.

Chapter 4

Cross-Layer Framework for Error Resilience and Power Efficiency

As already mentioned, this thesis proposes a cross-layer framework for processor-based systems. A key framework feature is the high configurability which allows dynamic adaptation of the system to current conditions and requirements regarding both resilience and power consumption. Moreover, the framework does not affect the system performance and it can be implemented at negligible area overhead of approximately 1.1% or less (see Section 5.3). The target system may be a single-core as well as a multicore processor.

The general concept for implementation of the framework was shown in Figure 1.7 on page 13. Thereby, both the framework and the target system are depicted as stacks composed of several layers. The figure only hints how the building blocks of the proposed framework should be implemented at the corresponding layers of the target system:

- the SWIELD FFs at circuit layer;
- the SOMU at architecture layer;
- the FFL at application/OS layers.

A more detailed overview of the cross-layer framework is given in Figure 4.1. The figure presents a simplified block-diagram of a layered processor-based system in which the framework has been already integrated. The rectangles surrounded by the T-like shape denote the framework's basic building blocks. It can be seen that the entities in the figure are centered around the SOMU. SOMU is the framework component which allows the processor to orchestrate the entire system operation. For example, the SOMU is responsible for clock signal distribution to the remaining system modules, thus frequency scaling and clock gating can be activated. By calling the appropriate FFL procedure, these techniques can be easily enabled/disabled. Also, dynamic supply voltage adjustment is managed by the SOMU which generates and forwards adequate voltage control words to the voltage regulator.

System adaptation for boosting resilience and power efficiency would not be possible without the multimodal SWIELD FFs. In general, the SWIELD FFs have to

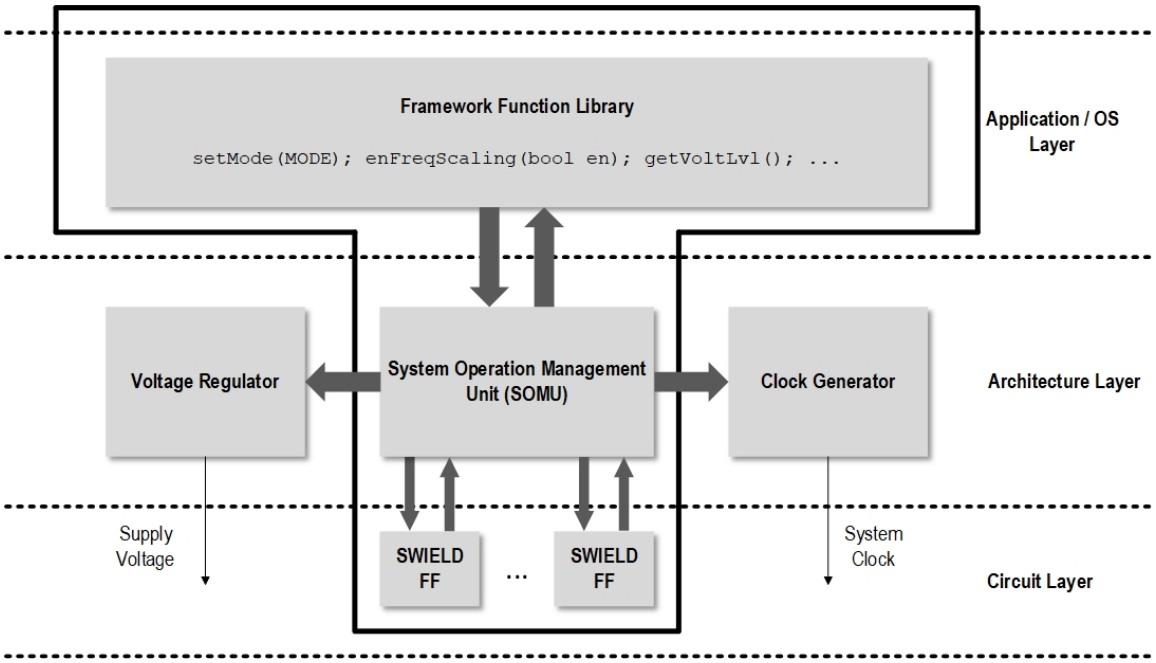


Figure 4.1: Simplified block-diagram of the proposed cross-layer framework implemented in a processor-based system.

be replaced with the timing-critical flip-flops in the system during design time. By default, a SWIELD FF is set to operate as an ISM FF which allows prediction and prevention of timing errors and hence, proper adaptive voltage/frequency scaling implementation. Thereby, the system is able to save a substantial amount of power. Changing the SWIELD FFs operation mode is a straightforward process that can be realized by a single FFL procedure call. For example, if protection against soft errors is required, the SWIELD FFs can be immediately transformed into TMR flip-flops. Alternatively, all additional SWIELD FF functionalities can be turned off which allows them to operate as conventional flip-flops.

From the previous discussion, it follows that the operation mode of the SWIELD FFs determines the operation mode of the entire system. Section 4.1 presents the SWIELD FF architecture along with a detailed description of each operation mode. Choosing the right operation mode in accordance with relevant factors is discussed in Subection 4.1.4. The remaining two framework components, the SOMU and the FFL are introduced in Section 4.2 and Section 4.3 respectively.

It is important to note that in a case of a multiprocessor system, SWIELD FFs need to replace the timing-critical flip-flops in all cores. In contrast, only one SOMU component is sufficient to manage the system operation regardless of the number of processor cores.

The entire RTL description and verification code for the hardware portion of the framework is written in VHDL.

4.1 SWIELD FF - A Multimodal Flip-Flop

One of the main contributions of this dissertation is the SWIELD configurable and multimodal flip-flop which was published in the paper [VKK19]. A schematic diagram of the SWIELD FF is presented in Figure 4.2.

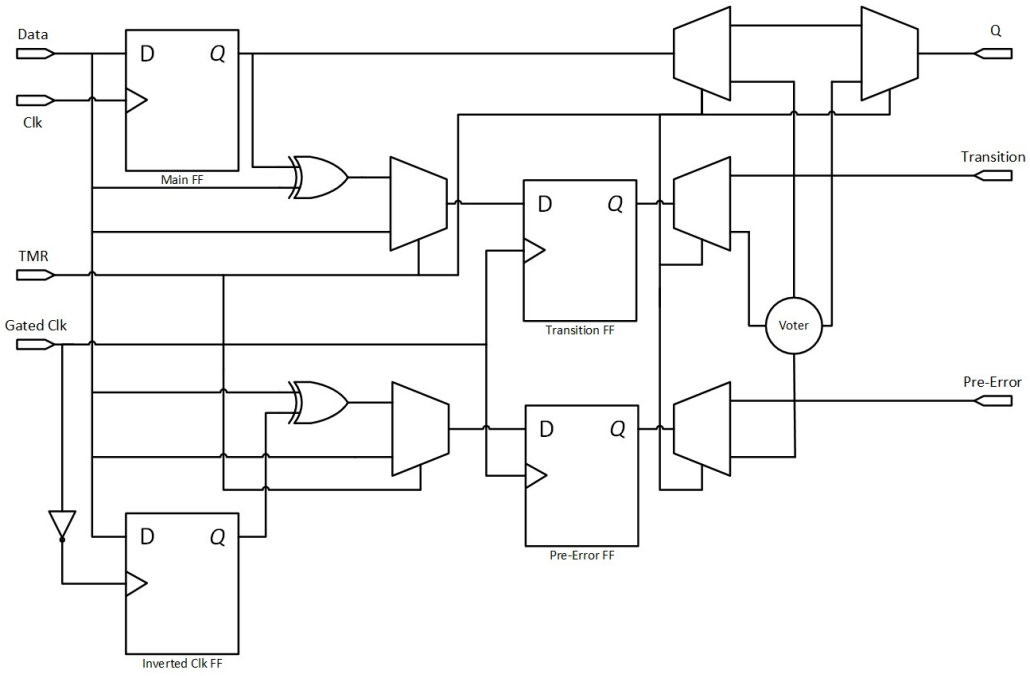


Figure 4.2: The SWIELD multimodal flip-flop.

As shown in the figure, the SWIELD FF is composed of four D flip-flops: one Main FF and three additional FFs necessary to implement timing error prediction functionality. The configurability and multimodality features of the SWIELD FF are provided by introducing two special control inputs (TMR and Gated Clk) as well as some extra combinational logic (multiplexers/demultiplexers and voter). Note that the TMR and Gated Clk inputs originate from the SOMU component which is responsible for driving these signals (See Section 4.2).

The Data input to the SWIELD FF is actually the adequate input signal to the Main FF. On the other hand, the SWIELD FF output Q could reflect either the output of the Main FF or the output of the voter depending on the current operation mode. While the Main FF is clocked by an "always-on" clock, the Inverted Clk FF, the Transition FF as well as the Pre-Error FF are synchronized by the Gated Clk. This clock is in phase with the main clock as they both originate from the same clock generator (see Figure 4.1). However, the Gated Clk goes through a clock gating logic which allows the framework to turn this clock off whenever necessary. Thus, the Gated Clk can be used to clock-gate the redundant flip-flops within the SWIELD FF, idle components or even entire inactive cores (in a case of a multiprocessor system). The roles of the remaining SWIELD FF input/output signals and subcomponents

are explained in the next subsections which are dedicated to the individual operation modes.

4.1.1 Operation as ISM FF

If the processor-based system executes a program or a task which is not critical in terms of error resilience or even in terms of performance (e.g. managing a passenger entertainment unit in a plane), the SWIELD FF can be put into ISM FF operation mode. By doing so, as suggested by the mode name itself, the SWIELD FF acts as an ISM capable to prevent timing errors. At the same time, a power saving AVFS scheme is realized, possibly without affecting the system performance. This can be achieved only if frequency scaling is turned off.

When the ISM FF operation mode is active, the Q output of the SWIELD FF reflects the output of the Main FF, whereas the TMR input is assigned logical '0' by the SOMU. Under such configuration, the schematic diagram of the SWIELD FF can be equivalently represented as shown in Figure 4.3. Here, the multiplexers/demultiplexers behave as small delay buffers which, as confirmed by the performed simulations, do not disrupt the intended functionality and the overall operation of the SWIELD FF. It should be also noted that the voter from Figure 4.2 does not play a role in this scenario.

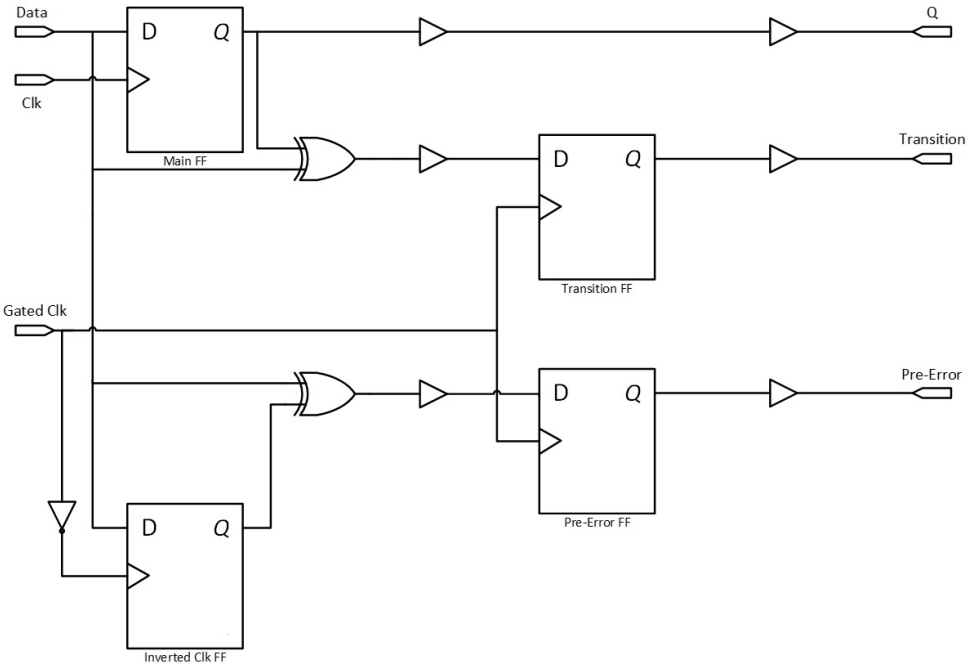


Figure 4.3: Equivalent representation of the SWIELD FF as an ISM FF.

As an ISM, the SWIELD FF monitors the corresponding critical path delay and sends feedback to the AVFS controller. Concretely, by observing its input data transitions, the SWIELD FF predicts potential timing errors and adequately warns the SOMU through the Transition and Pre-Error outputs. The generation of Transition

and Pre-Error signals is similar to the concept described in Section 3.3.1 and illustrated in Figure 3.8 on page 55. Hence, if an input data transition occurs during the pre-error detection window, a timing pre-error warning signal is issued.

Note that the Transition and Pre-Error outputs from each SWIELD FF in the system should be logically OR-ed together before connecting to the SOMU (Figure 4.4). Thereby, the overall Transition and Pre-Error rates are provided.

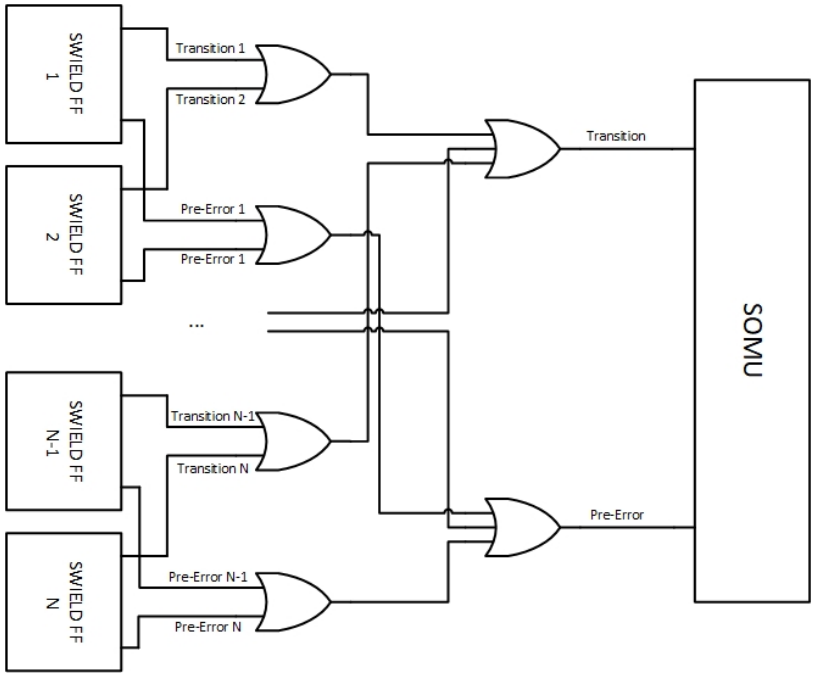


Figure 4.4: Obtaining the overall Transition and Pre-Error rates.

Based on the intensity of the pre-error warnings, the SOMU instructs the voltage regulator/frequency scaling logic to adequately adjust the supply voltage/operating frequency respectively in order to save power and prevent timing errors. During an observation interval of N_{tr} input data transitions, the number of pre-errors n_{pe} is counted and a decision is made whether the voltage/frequency should be adjusted or not (see Figure 4.5). If n_{pe} is less than some previously defined lower bound, that is, less than the minimal number of allowed pre-errors $n_{pe\downarrow}$, the voltage is reduced by a step of ΔV_{DD}. For a pre-error count above a previously defined upper threshold value, i.e., above the maximal number of allowed pre-errors $n_{pe\uparrow}$, the voltage is adequately increased by ΔV_{DD}. If the number of pre-errors n_{pe} lies within the interval $[n_{pe\downarrow}, n_{pe\uparrow}]$, the supply voltage level is not changed. The operating frequency can, but doesn't have to be scaled together with the supply voltage. When found useful, the frequency scaling can be enabled by calling the appropriate FFL procedure. Alternatively, the framework can be configured to automatically turn on frequency scaling if, for example, the supply voltage level drops below some predefined threshold value V_{TL}.

The voltage/frequency scaling and therefore, the amount of saved power depend directly on the implemented AVFS scheme, which relies on the pre-error count n_{pe}. The pre-error count can be viewed as a random variable. Namely, a SWIELD FF input

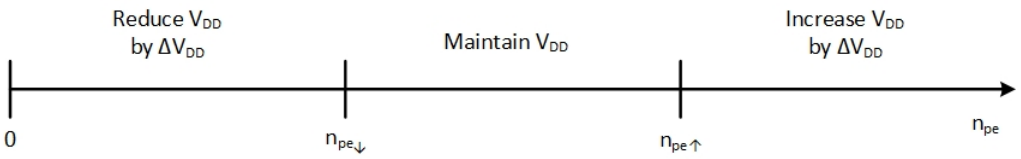


Figure 4.5: AVFS scheme for supply voltage regulation.

data may or may not cause pre-error warning. This is dependent on the propagation delay of the signal. On the other hand, the signal propagation delay depends on more factors. For example, despite on the path length to the SWIELD FF, it also depends on the current operating conditions (temperature, voltage) as well as on the executing instruction/instruction operands. Hence, an occurrence of a pre-error can be understood as a stochastic process. Consequently, the voltage/frequency are increased or decreased only with a certain probability.

Other important factors that have influence on the pre-error count are the following:

- the pre-error detection window length T_{dw};
- the number of input data transitions per observation interval N_{tr};
- the minimal number of allowed pre-errors $n_{pe\downarrow}$;
- the maximal number of allowed pre-errors $n_{pe\uparrow}$;
- the number of SWIELD FFs in the system N_{SFF}.

In order to maximize the amount of saved power, certain trade-offs between these parameters need to be investigated. For example, the number of SWIELD FFs N_{SFF} in the system is strongly correlated to the pre-error detection window length T_{dw}. In addition, there is a relationship between the number of transitions per observation interval N_{tr} and the number of minimal/maximal allowed pre-errors $n_{pe\downarrow}/n_{pe\uparrow}$ which can be expressed by using the binomial distribution formula [105, 136]. For instance, the probability that the pre-error count n_{pe} is lower than the minimal number of allowed pre-errors $n_{pe\downarrow}$ during an observation interval of N_{tr} input data transitions can be calculated as

$$P(n_{pe} < n_{pe\downarrow}) = P_{V_{DD}\downarrow} = \sum_{n_{pe}=0}^{n_{pe\downarrow}-1} \binom{N_{tr}}{n_{pe}} \cdot (P_{pe})^{n_{pe}} \cdot (1 - P_{pe})^{N_{tr}-n_{pe}} \qquad (4.1)$$

which is exactly the probability $P_{V_{DD}\downarrow}$ that the SOMU instructs the voltage regulator to reduce the supply voltage by ΔV_{DD}. Each addend in Equation 4.1 represents the probability of getting precisely n_{pe} pre-errors during the defined observation interval. Summing all addends from $n_{pe} = 0$ to $n_{pe} = n_{pe\downarrow-1}$ gives the probability $P(n_{pe} < n_{pe\downarrow})$. P_{pe} designates the probability of a pre-error occurrence when the

system is operating at supply voltage level V_{DD}. As previously mentioned, the pre-error occurrence depends on many factors and thus, determining P_{pe} is extremely challenging process that requires very long simulation time. Concretely, it would be necessary to observe the timing behaviour of the system and check whether pre-errors are produced for all possible input patterns under given pre-error detection window T_{dw} and observation interval N_{tr}. The same process has to be repeated for every applicable supply voltage level. In a case of a complex processor-based system, performing such procedure is practically infeasible due to the tremendously high number of valid inputs (all instructions combined with all possible operands).

However, by running simulations on multiple computing systems in parallel, Wirnshofer et al. [138] were able to extract the P_{pe} values for a relatively simple ISM-equipped circuit (16-bit multiplier). The authors replaced the three most timing-critical flip-flops in the design with Pre-Error Flip-Flops and simulated the delays of 100,000 random input patterns. An observation interval of $N_{tr} = 1000$ clock cycles (rather than transitions) was chosen, whereas the pre-error detection window length T_{dw} was set to be 30% of the clock period T_{clk}. Using the extracted values for P_{pe}, the authors managed to calculate the probability $P(n_{pe} < n_{pe\downarrow}) = P_{V_{DD}\downarrow}$. The experimental results confirmed several expected outcomes, for example, that the probability of reducing the supply voltage level is directly proportional to the timing slacks. However, some other interesting considerations are presented as well. Namely, it was shown that the probability $P_{V_{DD}\downarrow}$ decreases if the observation interval N_{tr} and/or the pre-error detection window length T_{dw} are increased. Moreover, since the pre-error occurrence is a stochastic process, a possibility of voltage overscaling that leads to timing errors cannot be entirely eliminated. In this direction, the authors concluded that an observation interval of approximately $N_{tr} = 1000$ clock cycles results in favourable trade-off between power saving and timing error prevention. Allegedly, larger values of N_{tr} only slightly improve the mentioned trade-off. On top of that, longer observation intervals would slow down the system reaction if internal or external operating conditions change. Furthermore, the authors claim to have determined that the optimal detection window length T_{dw} should be between 20% and 30% of the clock period T_{clk} (see Figure 4.6). When the multiplier is put into operation under such settings, power savings between 31% (corresponding to timing error rate of $1 \cdot 10^{-13}$) and 43% (corresponding to timing error rate of $1 \cdot 10^{-2}$) are reported. According to the authors, it is possible to save similar amount of power by combining different values of T_{dw}, $n_{pe\downarrow}$ and $n_{pe\uparrow}$. Finally, the published results confirm that the probability $P_{V_{DD}\downarrow}$ increases for higher values of the minimum allowed pre-errors $n_{pe\downarrow}$ which is also expected.

So far, only the case for supply voltage reduction was considered as an example. However, the same approach can be applied to determine the probabilities that the supply voltage is increased ($P_{V_{DD}\uparrow}$) or maintained ($P_{V_{DD}\leftrightarrow}$). Therefore, $P_{V_{DD}\uparrow} = P(n_{pe} > n_{pe\uparrow})$ and accordingly $P_{V_{DD}\leftrightarrow} = P(n_{pe\downarrow} \leq n_{pe} \leq n_{pe\uparrow})$ can be calculated by adequate reformulation of Equation 4.1 with respect to n_{pe} and $n_{pe\downarrow}$.

It is worth mentioning that, for every supply voltage level, exactly one of the three potential outcomes is possible, that is, the voltage level is either increased, preserved

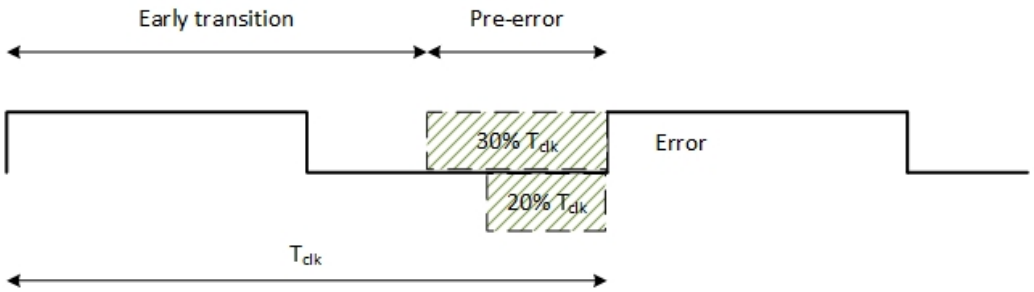


Figure 4.6: Pre-error detection window. The shaded regions designate the optimal T_{dw} lengths. (Adapted from Wirnshofer et al. [138])

or decreased. Hence, the following equation has to be satisfied:

$$P_{V_{DD}\downarrow} + P_{V_{DD}\uparrow} + P_{V_{DD}\leftrightarrow} = 1 \tag{4.2}$$

From the previous discussion, it can be concluded that the pre-error count n_{pe} is an appropriate metric for implementation of efficient AVFS scheme. The AVFS scheme may be modified by tuning parameters such as the observation interval (N_{tr}), the minimum/maximum allowed pre-errors ($n_{pe\downarrow}/n_{pe\uparrow}$) or the pre-error detection window length (T_{dw}). Thus, an optimal trade-off between power efficiency and acceptable timing error rate can be achieved.

One of the biggest advantages of the proposed framework is the fact that it allows dynamic adjustment of the AVFS-relevant parameters (pre-error observation interval and minimum/maximum allowed pre-errors) by calling appropriate FFL procedures. This provides the programmer an ability to modify the implemented AVFS scheme according to the current application requirements/operating conditions while the system is in operation.

Putting the SWIELD FF into ISM FF operation mode results in significant power savings as elaborated further in Chapter 6. Typically, AVFS is employed as a main low power technique. However, it is possible to use it in combination with other techniques such as clock- or power gating of idle components or even entire cores (in case of multiprocessor systems) to further boost the power efficiency.

4.1.2 Operation as TMR FF

Flip-flop triplication is a widely used approach for providing fault tolerance, especially to radiation-induced faults (SEEs). SEEs are considered as one of the most critical threats to the resilience of computing systems.

The fact that the SWIELD FF contains several redundant flip-flops is used to enable on-demand reconfiguration of its internal components. As a result, the SWIELD FF can be dynamically transformed into a TMR FF. This means that besides against timing errors, the proposed framework is capable to protect against SEEs too, concretely against SEUs.

In order to switch the SWIELD FF into TMR FF operation mode, the framework needs to set the value of the TMR input to logical '1'. By doing so, a TMR structure composed of the Main FF, the Transition FF and the Pre-Error FF is formed. Note that the Inverted Clk FF cannot be considered for this purpose as it is a falling edge-triggered flip-flop. Actually, the Inverted Clk FF is entirely irrelevant while this operation mode is active. The equivalent schematic representation of the SWIELD FF as a TMR FF is illustrated in Figure 4.7.

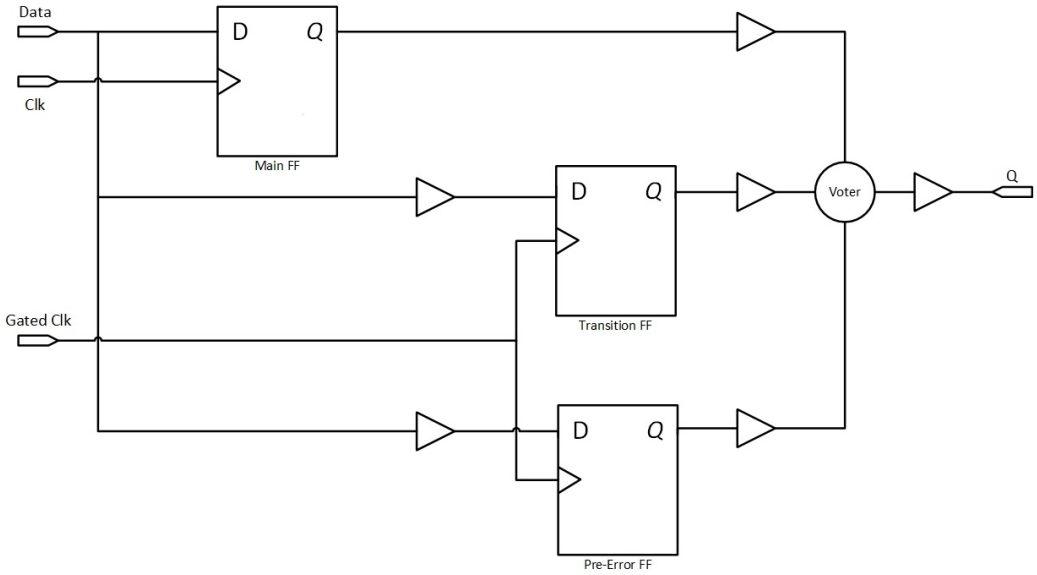


Figure 4.7: Equivalent representation of the SWIELD FF as a TMR FF.

As can be seen in the figure, the Q output of the SWIELD FF in TMR FF mode is driven by the voter. Every clock cycle, the voter propagates the majority value to the output without interrupting operation. Similarly to the case when the SWIELD FF acts as an ISM FF, the multiplexers/demultiplexers can be treated as delay elements which do not disturb the intended functionality. However, in TMR FF mode, the voters impose overhead to the SWIELD FFs in terms of increased delay on paths which are already considered timing-critical.[1] It turns out that this overhead puts an upper limit to the number of SWIELD FFs which can be placed in the system. One obvious reason is that a higher number of SWIELD FFs increases the probability of a timing error occurrence due to dynamic variations. Furthermore, the probability that two (or more) SWIELD FFs share the same critical path (either directly or via a feedback loop in the processor pipelines) increases with the number of SWIELD FFs. Prolonging critical paths in this manner may also lead to timing violations. A simple and practical algorithm for determining the optimal number of SWIELD FFs to be placed in the system [VHKK21a] is presented in Section 5.1. Additionally, the proposed algorithm is able to find an optimal T_{dw}.

While TMR FF operation mode is in force, the values of the Transition and Pre-Error outputs are irrelevant and therefore neglected by the SOMU. Thus, voltage

[1]Recall that the SWIELD FFs are replaced with timing-critical flip-flops.

scaling cannot be performed and as a result the framework has to fix the supply voltage to a constant level (not necessarily the nominal level). However, frequency scaling can be used if necessary.

Switching to TMR FF mode is useful whenever the system is (expecting to be) exposed to higher soft error rates. In fact, SER can be monitored in real time by employing specially-designed radiation particle detectors [7]. Based on the information provided by such detectors, the system can decide to put the SWIELD FFs into TMR FF mode.

Unfortunately, the SWIELD FF is not able to behave as a TMR FF and as an ISM FF at the same time. Thus, simultaneous protection against both timing and soft errors cannot be provided. However, one of these two distinct operation modes can be well used in combination with other techniques for error resilience and/or power consumption improvement. Having in mind that only a selected subset of flip-flops in the system is replaced with SWIELD FFs, it might be beneficial to combine the TMR FF operation mode with additional fault tolerance technique(s). Depending on the actual system and its purpose, techniques like ECC or full triplication of register files [6]) can be employed.

4.1.3 Operation as Regular FF

The SWIELD FF can be configured to work as any other conventional flip-flop. This is referred to as Regular FF operation mode. In order to put the SWIELD FF into Regular FF mode, the framework needs to turn off Gated Clk and to set TMR to logical '0'. Thereby, the equivalent schematic diagram of the SWIELD FF can be presented as shown in Figure 4.8.

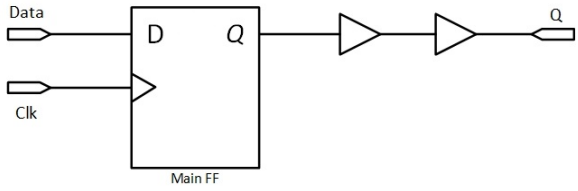


Figure 4.8: Equivalent representation of the SWIELD FF as a Regular FF.

In this operation mode, the Q output of the SWIELD FF reflects the output of the Main FF since the path through the voter is avoided. The Transition and the Pre-Error flip-flops are switched off and thus, not considered by the framework.

One might wonder what is the benefit of the SWIELD FF acting as a standard flip-flop. In fact, the Regular FF operation mode would be the most logical choice when high performance operation is required. The supply voltage might be kept at nominal level in this case. Additionally, this mode could be favourable in scenarios where the system reduces the supply voltage to some near-optimal level [2] by exploiting AVFS scheme. At that point, the framework could simply switch off the voltage scaling.

[2] An optimal supply voltage level would be a level low enough to save power and high enough not to induce faults.

This would prevent the Inverted Clk FF, the Transition FF, the Pre-Error FF as well as the additional combinational logic within the SWIELD FF from switching. A substantial amount of power could be saved in this way. However, operation at sub-nominal supply voltage level makes the system vulnerable, especially to timing errors. Potential dynamic variations would increase its vulnerability even further. Therefore, to avoid errors, it might be useful to combine the Regular FF operation mode with frequency scaling. Of course, this scenario is applicable only if the currently running application is not performance-critical.

4.1.4 Selecting the Right Operation Mode

The framework is able to dynamically switch between the SWIELD FF operation modes. This raises the question: which operation mode should be selected in a specific situation? It turns out that there are several factors influencing the mode selection. Assuming a safety- or a mission-critical embedded system, the decision-making factors can include the following:

- system type (e.g. autonomously-driving vehicle, plane, submarine, satellite...);
- current application running on the system (e.g. pedestrian detection, instrument landing system (ILS), image processing...);
- power supply type (e.g. rechargeable batteries, generators, photovoltaics...);
- current power budget and available power management techniques;
- system environment (ground, water, air, space);
- current radiation levels in the environment;
- system state (current age, time spent in active operation, correct functionality of individual components...);
- system structure (contains redundancy or not and if yes, which type...).

For example, let's consider a processor-based system intended to manage the communication between internal hardware components of a satellite [VPS12]. The primary function of such system is to connect the existing hardware to the on-board computer. When a gyroscope or a magnetorquer require an access to the computer regarding attitude control, the system must perform the processing in a precise way. Hence, the framework could activate TMR FF operation mode in order to reduce the probability of a SEU occurrence. On the other hand, when data from light sensors or cameras needs to be processed for non-critical purposes like weather forecasting, the framework could put the system into ISM FF mode. Finally, for high performance computations, the operation mode could be switched to Regular FF mode.

4.2 System Operation Management Unit (SOMU)

This section explains in details how the SOMU performs the intended functions from a technical point of view.

Figure 4.9 shows a simplified block diagram of the SOMU. The main system clock Clk enters the SOMU from where it branches into at least two clock signals for different domains. As suggested by its name, the Always-on Clk is continuously active and cannot be stopped. It is used to synchronize essential components such as processor core in a single-core system, memory/interrupt controllers, bus arbiters etc.

At least one Gated Clk signal is generated as an output intended to clock-gate the redundant flip-flops within the SWIELD FF. In case of a multiprocessor system, Gated Clk signals for clock gating idle cores and/or memories could be also generated. Note that these signal(s) originate from the CLOCK GATING LOGIC block.

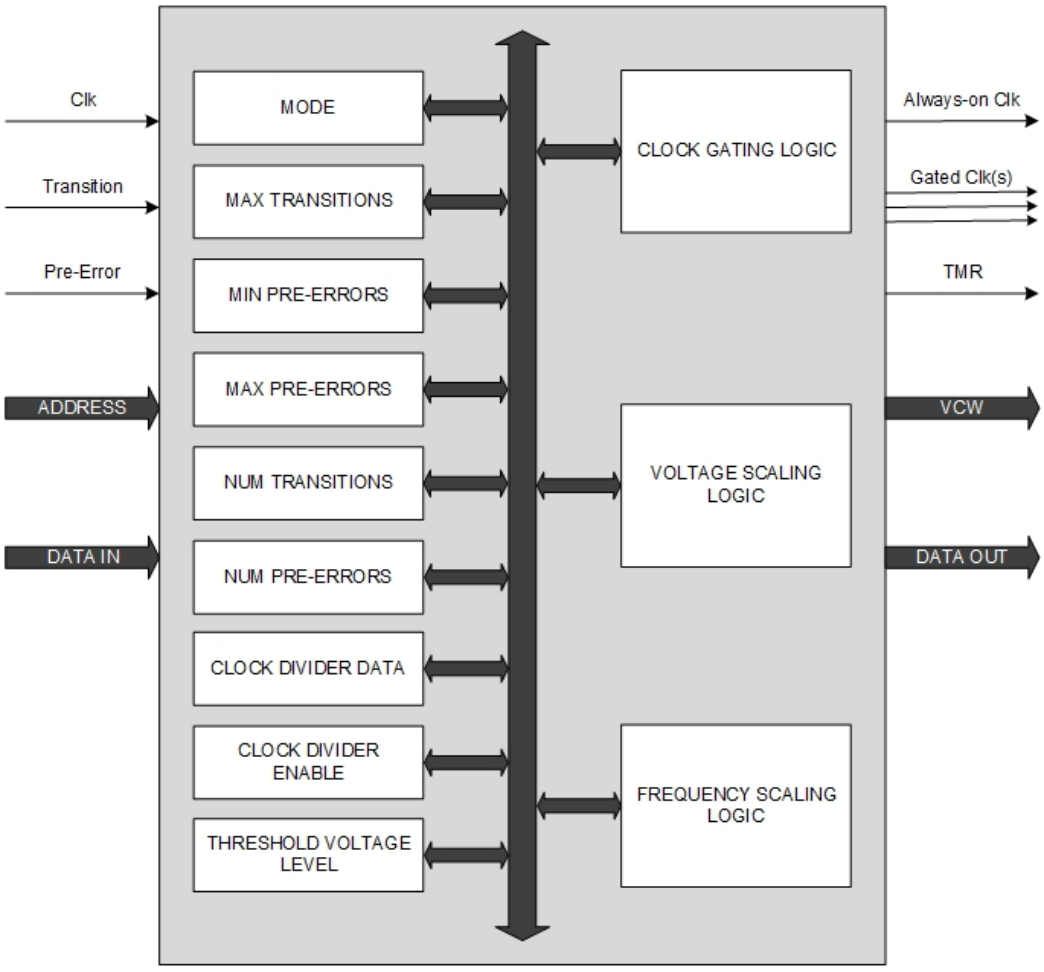


Figure 4.9: Simplified block diagram of the SOMU.

The Transition and Pre-Error inputs represent the corresponding logically OR-ed outputs from the SWIELD FFs (see Figure 4.4). The SOMU contains a set of registers which can be written or read-out via the system internal data bus. Table 4.1 shortly

describes the function of every SOMU register. The values stored in the registers together with the Transition and Pre-Error inputs provide the necessary information for driving the CLOCK GATING, VOLTAGE SCALING and FREQUENCY SCALING logic blocks.

Table 4.1: SOMU registers.

Register	Description	Type
MODE	Operation mode of the SWIELD FFs	Read/Write
MAX TRANSITIONS	Number of transitions per observation interval N_{tr}	Read/Write
MIN PRE-ERRORS	Minimal number of allowed pre-errors $n_{pe\downarrow}$	Read/Write
MAX PRE-ERRORS	Maximal number of allowed pre-errors $n_{pe\uparrow}$	Read/Write
NUM TRANSITIONS	Current number of transitions	Read only
NUM PRE-ERRORS	Current number of pre-errors	Read only
CLOCK DIVIDER ENABLE	Enable/disable frequency scaling (FS)	Read/Write
CLOCK DIVIDER DATA	Value that determines the scaled clock period T_{clk}	Read/Write
THRESHOLD VOLTAGE LEVEL	Automatically activate FS at the specified voltage level V_{TL}	Read/Write

The operation mode of the SWIELD FFs is set by writing appropriate value in the MODE register. Table 4.2 shows how the operation modes are encoded by a 2-bit binary code. Based on the current operation mode, the TMR as well as the appropriate Gated Clk outputs are adequately generated.

Table 4.2: MODE register. The difference between the two ISM FF modes is in the FS activation method (see explanation in the text below).

Binary code	SWIELD FF Operation mode
00	Regular FF
01	ISM FF (Manual FS activation)
10	TMR FF
11	ISM FF (Automatic FS activation)

By writing the MAX TRANSITIONS, MIN PRE-ERRORS and MAX PRE-ERRORS registers, the parameters required by the VOLTAGE SCALING LOGIC block are defined. The NUM TRANSITIONS and NUM PRE-ERRORS are read-only registers that contain the current cumulative numbers of transitions and pre-errors respectively. Thus, when the SWIELD FFs operate in ISM FF mode, the Transition and Pre-Error inputs are observed during an observation interval MAX TRANSITIONS = N_{tr}. Then, the VOLTAGE SCALING LOGIC compares the current number of pre-errors to the values $n_{pe\downarrow}$ and $n_{pe\uparrow}$ stored in the MIN PRE-ERRORS and MAX PRE-ERRORS registers respectively. An appropriate VCW (Voltage Control Word) output is generated depending on the comparison result. The VCW output instructs the voltage regulator(s) to adjust the supply voltage level according to the implemented AVFS scheme as described in Subsection 4.1.1.

The system is clocked by the Always-on Clk. Driver of this signal is the FREQUENCY SCALING LOGIC block. Implementation of frequency scaling is allowed by manipulating with the CLOCK DIVIDER ENABLE and CLOCK DIVIDER

DATA registers. When storing a non-zero value into the CLOCK DIVIDER ENABLE register, the frequency scaling is enabled independently of the current supply voltage level. However, this doesn't activate frequency scaling by default. In order to actually activate frequency scaling, a non-zero value needs to be written into the CLOCK DIVIDER DATA register. This value should be an exponent of 2 which determines by how much the clock period T_{clk} is prolonged. For example, writing 1 in this register would double the clock period, whereas 2 would increase it fourfold and so on. The default value of the CLOCK DIVIDER DATA register is 0, which implies system operation at maximum frequency even if frequency scaling is enabled in the CLOCK DIVIDER ENABLE register.

At last, scaling the frequency in dependence on the current supply voltage level is also possible. Namely, the user can store a voltage control word that corresponds to a pre-defined, critically low voltage level V_{TL} in the THRESHOLD VOLTAGE LEVEL register. If the SWIELD FFs are set to operate as ISM FFs with automatic FS activation (see Table 4.2), the frequency scaling will be activated as soon as the supply voltage level drops below V_{TL}.

Note that faults, especially SEUs, might affect the SOMU as well. To prevent SEU-induced errors and potential data corruption, the SOMU registers could be triplicated. Thereby, to avoid area and power overheads, the triplication may be performed only partially, by protecting exclusively the used register bits. For example, recall that the MODE register utilizes only two bits.

4.3 Framework Function Library (FFL)

This section elaborates on the software part of the cross-layer framework in more details. The FFL is actually a set of procedures whose role is to simplify the management of the framework blocks implemented in hardware. By running an appropriate FFL procedure, the programmer is able to easily configure framework-related parameters or to retrieve their current status.

Generally, the FFL procedures are realized in a similar manner to `set()` and `get()` functions. Using setters and getters is a conventional software development approach for updating and retrieving variable values. Additionally, by introducing constructs like enumeration, typical for high-level programming languages, the framework manipulation becomes even more intuitive. For example, Figure 4.10 shows how one of the essential FFL procedures, `setSWIELDMode(MODE)` is realized using pseudocode:

Hence, writing an adequate value in the MODE register sets the SWIELD FFs operation mode. The other framework registers can be written in a similar way as shown in Figure 4.10. For instance, the statement **`store`** `1000, [SOMU_MAXTR_REG]` means storing the value 1000 in the MAX TRANSITION register. Alternatively, the statement can be encapsulated within a `set()` procedure and the execution would be simplified to calling e.g. `setMaxTrans(1000);`.

Quite often it is practical to check the current status of certain parameters in the framework. Figure 4.11 shows the `getNumPE()` procedure which retrieves the current cumulative number of pre-errors during the present observation interval.

```
type MODE_TYPE = (REGULAR_FF, ISM_FF-MAN_FS, TMR_FF, ISM_FF-AUTO_FS);

void setSWIELDMode(MODE_TYPE MODE) {
  case MODE of
    REGULAR_FF: store REGULAR_FF, [SOMU_MODE_REG]; //write in MODE reg.
    ISM_FF-MAN_FS: store ISM_FF-MAN_FS, [SOMU_MODE_REG];
    TMR_FF: store TMR_FF, [SOMU_MODE_REG];
    ISM_FF-AUTO_FS: store ISM_FF-AUTO_FS, [SOMU_MODE_REG];
    others: store ISM_FF-AUTO_FS, [SOMU_MODE_REG];
}
```

Figure 4.10: Procedure that sets the SWIELD FF operation mode.

```
void getNumPE() {
  load NR_PE, [SOMU_NUM_PRE-ERRORS_REG]; //read the register contents
  print NR_PE;
}
```

Figure 4.11: Retrieving the current number of pre-errors. The `load` statement copies the contents from the NUM PRE-ERRORS register into the variable NR_PE.

In order to switch the SWIELD FFs operation mode to REGULAR FF or to TMR FF, it is sufficient to call the corresponding procedure with the adequate argument, i.e. `setSWIELDMode(REGULAR_FF)` or `setSWIELDMode(TMR_FF)`. On the other hand, switching to ISM FF operation mode might require calling more than one FFL procedures. For example, if the user needs to modify the AVFS-related parameters (N_{tr}, $n_{pe\downarrow}$ and $n_{pe\uparrow}$), in total four procedures need to be invoked. Figure 4.12 shows a code sequence that puts the SWIELD FFs into ISM FF operation mode and sets values for the AVFS parameters.

```
setSWIELDMode(ISM_FF-MAN_FS);        //switch to ISM FF operation mode
setMaxTrans(1000);           //store 1000 in MAX TRANSITIONS register
setMinPreErrors(1);              //store 1 in MIN PRE-ERRORS register
setMaxPreErrors(20);            //store 20 in MAX PRE-ERRORS register
```

Figure 4.12: Switching to ISM FF mode and setting AVFS parameters.

Furthermore, depending on whether frequency scaling is required and how it is to be activated (manually or automatically), there are two possible scenarios to realize the AVFS scheme. The corresponding code sequences are shown in Figure 4.13 and Figure 4.14 respectively.

The FFL is extremely practical because it hides all the tiny details of the actual framework operation from the programmer. Such approach enables the higher levels of the system stack (OS or application) to only specify the name of the procedure and the adequate argument (if required) instead of memory addresses or bit positions

```
setSWIELDMode(ISM_FF-MAN_FS);
setFSEnable(TRUE);                          //enable frequency scaling
setClkDivider(1);                           //double the clock period
```

Figure 4.13: Manual frequency scaling activation. The clock period will be doubled following the execution of the shown code sequence.

```
setSWIELDMode(ISM_FF-AUTO_FS);
setFSEnable(TRUE);
setClkDivider(1);
setThresholdVL(0x"FFFC0"); //set threshold voltage level codeword
```

Figure 4.14: Automatic frequency scaling activation. The clock period will be automatically doubled when the supply voltage level falls below, e.g 1.08 V encoded as the codeword 0x"FFFC0".

within a register. It is, therefore enough for the user to know the procedure signatures within the FFL in order to manipulate with the framework.

Finally, the FFL, like any other software might be subject to cyberattacks with malicious intent. Although software protection is out of the scope of this dissertation, the literature might offer appropriate mechanism against FFL corruption.

Chapter 5

Implementation and Integration of the Framework

Chapter 4 provided a comprehensive presentation of the cross-layer framework by describing its building blocks in details. This chapter elaborates thoroughly on integration of the proposed framework in actual processor-based systems.

First, a strategy for integration of the cross-layer framework in complex systems is introduced in Section 5.1. Then, the architectures of both single- and multicore processor systems encompassing the proposed framework are demonstrated in Section 5.2. Finally, Section 5.3 presents the implementation results.

5.1 Framework Integration Strategy

It was previously emphasized that realization of AVFS scheme requires interconnecting the hardware-implemented framework components in a closed-loop fashion. This fact calls for certain modifications in the traditional system design flow. In the further text, a strategy for integrating the hardware portion of the framework into complex processor-based systems is described in details.

As depicted in Figure 5.1, the proposed strategy consists of adding two extra steps to the conventional design flow:

1. Interfacing the SOMU to the processor-based system. It is performed on RT level in a similar manner to connecting any other system component;

2. Insertion of the SWIELD FFs in the processor-based system. It is achieved by replacing SWIELD FFs with timing-critical flop-flops. This step requires a modification of the synthesized gate-level netlist.

The SOMU is designed to be flexible, portable and scalable in order to minimize the effort for interfacing to the target processor-based system. As a result, the interfacing process comes down to:

- instantiation of the SOMU entity in the top-level RTL module of the processor-based system design;

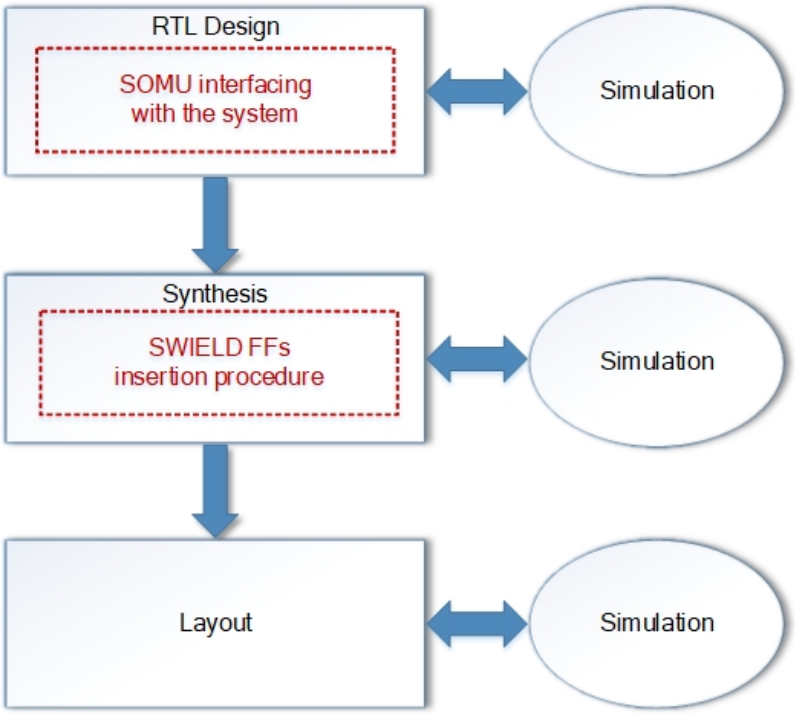


Figure 5.1: Framework integration flowchart. The blocks enclosed by solid lines (in blue) denote the classical design flow phases, while the internal blocks enclosed by dashed lines (in red) designate the additional steps in the proposed strategy. The location of the internal blocks indicates the phases during which the additional steps need to be taken.

- proper connection of its input/output signals to the rest of the components that need to be in direct communication with the SOMU.

Recall that the SOMU contains internal registers which should be accessible to the processor(s) through the system data bus. Therefore, adequate address decoding needs to be performed. Eventually, the SOMU in a processor-based system acts as a peripheral unit with memory-mapped registers.

On the other hand, the procedure for inserting SWIELD FFs in the system is a bit more complicated and demanding. Additionally, a careful analysis is required before making the decision regarding the number and the location of the SWIELD FFs to be inserted. Figure 5.2 illustrates all steps that need to be taken during the insertion procedure. Note that some of these steps have to be repeated.

After the SOMU is successfully interfaced to the system, a synthesis is performed. In this phase, the gate-level netlist and the STA reports are obtained. The generated reports help to identify the timing-critical flip-flops, that is, the flip-flops which are candidates for replacement with SWIELD FFs. To determine whether a given flip-flop will be classified as critical, the pre-error detection window (T_{dw}) is used as a metric. Algorithm 1 shows how T_{dw} is determined. Ultimately, all flip-flops with slack shorter than the pre-error detection window are considered critical.

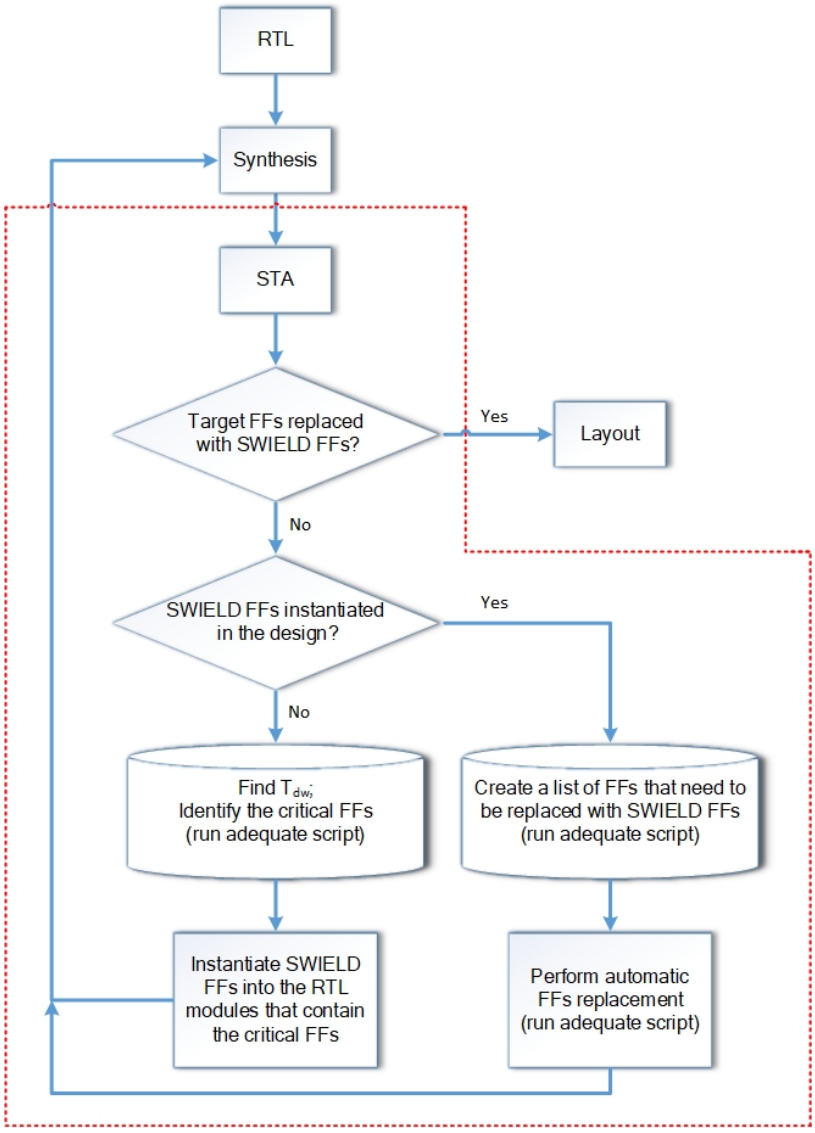


Figure 5.2: A flowchart providing detailed display of the SWIELD FFs insertion procedure as part of the system design flow. The blocks surrounded by the red dashed line denote the necessary individual steps of the procedure.

The initial value of the pre-error detection window is set to half clock cycle - line 1 in Algorithm 1. However, assigning such high value to T_{dw} has several non-negligible drawbacks. Namely, setting too long pre-error detection window implies more critical flip-flops to be replaced with SWIELD FFs. This results in higher area overheads, often too high to be considered. Furthermore, the SWIELD FF voters, as emphasized in Subsection 4.1.2, contribute to prolonging the critical paths while TMR FF operation mode is active. Hence, too many SWIELD FFs acting as TMR flip-flops may cause system crash due to timing errors. Finally, too long pre-error detection window results in more timing pre-error warnings. Increased number

Algorithm 1: Find T_{dw} and identify the critical FFs.

```
1  Set the default value of T_dw to T_dw = T_clk/2;
2  while T_dw > 0 do
3      Identify the critical flip-flops and their number;
4      Perform replacement;
5      Check timing;
6      Run simulation;
7      if mode(SWIELD_FF)==TMR_FF ∧ no timing violations then
8          break;
9      else
10         Gradually reduce T_dw (E.g. for 5% or 10%);
11 Report T_dw;
12 Report critical flip-flops;
```

of timing pre-error warnings leads to reduced power savings provided by the AVFS scheme. Therefore, the value for the pre-error detection window is gradually decreased until a correct system operation under SWIELD FFs in TMR FF mode is assured (line 10). As soon as a value for T_{dw} that meets the condition in line 7 is found, a list and the number of critical flip-flops that satisfy the defined criteria can be obtained (lines 11 and 12). However, this is not necessarily the final set of critical flip-flops to be replaced. For example, if so many SWIELD FFs in the system introduce higher area overhead than some predefined constraint, the number of critical flip-flops (and SWIELD FFs) can be further reduced by decreasing T_{dw} until an acceptable value is reached.

Identification of critical flip-flops in the system (line 3) and their replacement with SWIELD FFs (line 4) is automated by using specially developed scripts. A script that parses STA reports according to the determined value of T_{dw} is used to find the number and the identifiers of the critical flip-flops in the system. The list of the retrieved critical flip-flops together with the synthesized gate-level netlist are inputs to the script for automatic flip-flop replacement. This script locates the specified critical flip-flops in the system and adequately rewires SWIELD FFs to their inputs and outputs. The script output is a modified gate-level netlist that contains SWIELD FFs instead of the identified critical flip-flops which can, but doesn't have to be located at the end of the critical paths. Figure 5.3 illustrates the process of automatic critical flip-flop identification and replacement.

Finally, the newly-generated netlist is used for synthesis of the modified system and hereby a timing check is performed. If the timing constraints are met, a post-synthesis gate-level timing simulation is run to confirm the correct system operation. Afterwards, the system design flow can proceed towards the next stage. In case of timing violations, one can always reduce the pre-error detection window, the clock frequency or both. However, note that frequency reduction will result in performance degradation.

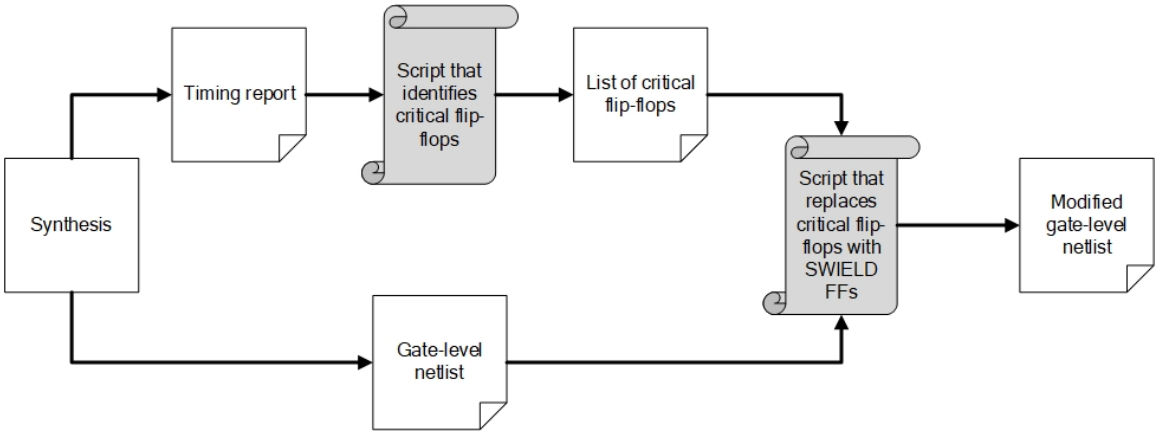


Figure 5.3: Automatic critical flip-flop identification and replacement.

5.2 Integration in Processor-Based Systems

The cross-layer framework introduced in Chapter 4 is designed to be integrable in any typical general-purpose processor. Of particular interest is integration in complex processor-based systems that include at least one general-purpose processing core. By leveraging the framework, the systems become easily adaptable, which is a crucial feature for providing power efficiency and error resilience.

5.2.1 Architecture of the Processor-Based Systems

In all thesis-related implementations, the systems are based on the LEON2 processor core [55]. However, it is important to note that any other general-purpose processor can be used as well. The LEON line of processor cores has been used by ESA in space missions for more than 20 years [6]. LEON2 is an open-source VHDL model of a 32-bit processor that conforms to the SPARC V8 instruction set architecture (ISA) [72]. The model is highly configurable and specifically designed for embedded applications. It features a single instruction issuing, classic RISC pipeline with five stages. By default, LEON2 includes hardware multiplier and divider, separate instruction and data caches as well as interfaces to a floating-point unit (FPU) and mathematical co-processors. Communication with the external world is performed via an Advanced Microcontroller Bus Architecture (AMBA) bus [8]. Throughout this work it is assumed that the cores and the peripheral components are already verified and functionally correct.

As a proof of concept, the strategy presented in Section 5.1 was followed to embed the framework in both single- and multicore LEON2-based systems. Implementation-specific details of the realized systems are given in the further text.

Single-core Processor-Based System

The single-core implementation represents a minimal system that serves as a proof-of-concept test vehicle. Figure 5.4 depicts a block diagram of the system.

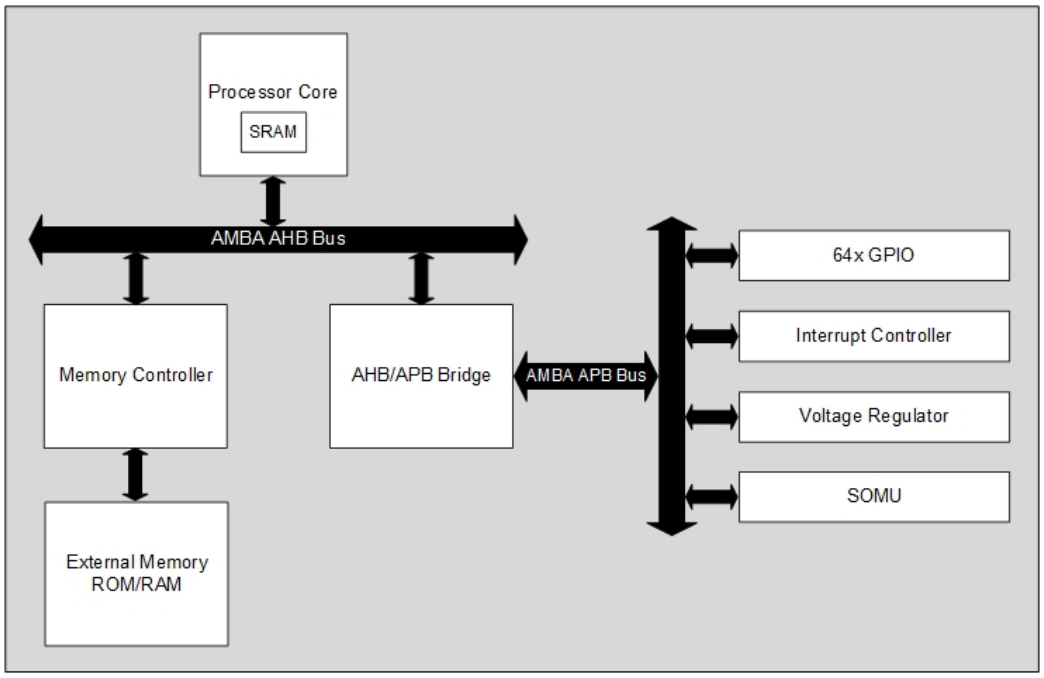


Figure 5.4: Single-core LEON2 processor-based system.

A memory controller provides access to an off-chip memory. Every word loaded from or stored to the memory is protected against SEUs by additional Single Error Correction - Double Error Detection (SEC-DED) Hsiao code [66].

Using caches may lead to reduction of the program execution time determinism [121] which is crucial for safety-/mission-critical systems. Therefore, the cache memories were replaced with local scratchpad SRAM memory. If, for example, the program is stored in a scratchpad SRAM, the memory requests are not forwarded to the off-chip memory. This leads to a significant reduction of the main memory requests. As a result, the system performance can be drastically improved since instructions are provided by the local SRAM. Note that the scratchpad SRAM can be clock-gated. In such case, the memory request is forwarded by the processor to the off-chip memory at the same address. Thus, the programmer is able to reconfigure the system and switch off the local SRAM to save power. In that case, however, performance penalties might have to be paid.

The system contains several standard on-chip peripheral components such as interrupt controller and programmable GPIO ports. Furthermore, to enable interfacing of the SOMU with the system, its RTL code is adapted according to the AMBA specification. Hereby, appropriate address decoding is performed and thus the internal SOMU registers are easily accessible by the processor core.

Finally, the power supply of the system is provided by a voltage regulator [1] able to dynamically adjust the supply voltage. Incorporation of a voltage regulator in the system is of key importance for implementation of AVFS scheme. However, design and implementation of the voltage regulator is out of the scope of this work.

[1]The voltage regulator in this thesis is presented as a behaviour model only.

Multicore Processor-Based System

Following a successful integration in a minimal, single-core system, it is important to investigate how the proposed framework scales with more powerful and more complex multiprocessor-based systems. For this purpose, a homogeneous multiprocessor LEON2-based system with four identical cores has been implemented (Figure 5.5).

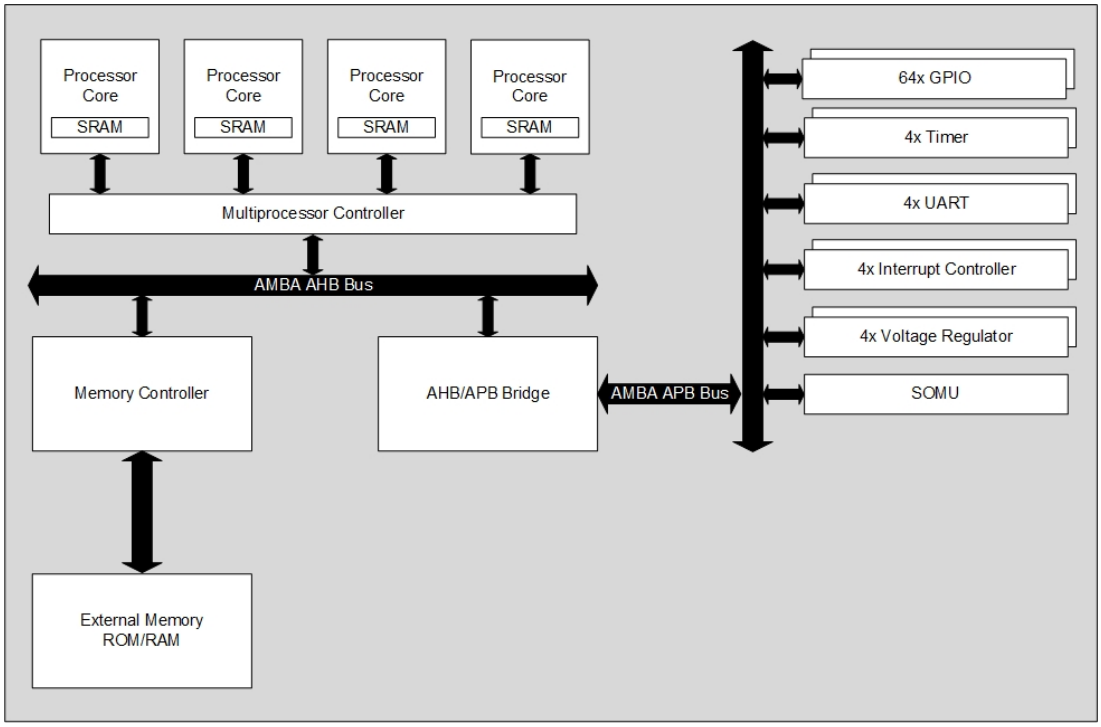


Figure 5.5: Multicore LEON2 processor-based system.

The multiprocessor may be viewed as a superset of the single-core system since it is composed of the same components in addition to some extra hardware which is not part of the minimal implementation. Concretely, the multiprocessor system contains:

- 4x LEON2 cores;
- 4x interrupt controllers;
- 4x timers;
- 4x UARTs;
- 4x voltage regulators;
- 1x multiprocessor controller.

The four processor cores are granted fair access to the memory or peripheral components through bus arbiter. Here, the off-chip memory is distributed among the cores. In a case of multiprocessor-based safety-/mission critical systems, it makes

even more sense to replace the caches with internal scratchpad SRAM because of two reasons. First, the complexity is significantly smaller since implementation of cache coherence mechanisms is not necessary. Second, when the local SRAM is accessed, the memory request is not forwarded to the off-chip memory, which reduces the bus contention. Therefore, each core is equipped with internal scratchpad SRAM.

In order to leverage the intrinsic flexibility offered by the multicore system and to enhance the adaptability provided by the framework, a multiprocessor controller similar to the one proposed by Simevski et al. [117, 121] is incorporated. Basically, such controller is able to put the multiprocessor in one of the following sub-modes: [2]

1. De-stress sub-mode (DSSM);
2. Fault-tolerant sub-mode (FTSM);
3. High performance sub-mode (HPSM).

During DSSM only one core is active at a time while the remaining are clock-gated. The goal is to interchange an active core according to some pattern (e.g. round robin), so the difference between the core ages would be as small as possible. In other words, the DSSM is supposed to reduce aging effects and to extend system lifetime. Keeping some of the cores idle also increases power efficiency.

FTSM is realized by formation of core-level NMR group where $N \in \{2, 3, 4\}$ is the number of cores. The NMR group is tightly synchronized to run the same task. That means, the N cores execute the same instructions simultaneously in redundant fashion to enhance the fault-tolerance. Each clock cycle, the outputs from the N cores are compared by a voter able to initiate appropriate action (interrupt or reset) if a mismatch is detected.

The HPSM implies regular multiprocessor operation where the cores operate independently of each other. There are no fault-tolerant or power-/aging-aware techniques employed while the HPSM is active.

Note that in a case of multiprocessor system, the AVFS scheme is implemented on a core level, i.e., each core is powered by its own voltage regulator. As a consequence, slight modifications of the SOMU structure and FFL procedures are required to accommodate the framework. Concretely, the SOMU registers must reflect the number of cores in the system, thus each core would be able to read and write its own AVFS-related parameters. For a multiprocessor with N cores, every SOMU register needs to have N instances. The only exception is the MODE register because the SWIELD FF operation mode is global and defined on system level. Similarly, the related FFL procedures have to explicitly state which core should take the corresponding action. For example, the `getNumPE(2)` will return the number of pre-errors registered in the core number two, whereas `setFSEnable(3, TRUE)` will enable the frequency scaling for the core number three.

Finally, it should be mentioned that if the framework is to be used in FTSM, a special attention must be paid when activating AVFS. Core-level voltage/frequency

[2] The term "sub-mode" is chosen to avoid mistaking for the framework operation modes.

scaling might affect the processing speed of each core. As voting needs to performed on every clock cycle, it is crucial that the cores remain in constant synchronization.

5.3 Implementation Results

This section presents the results acquired after synthesizing the framework components implemented in hardware, (the SWIELD FF and the SOMU) as well as the processor-based systems described previously. Comprehensive evaluation of the framework regarding power consumption and error resilience with respect to different system operation scenarios is given in Chapter 6.

Synthesizing of the designs is done by translation of the RTL code into structural, gate-level description based on the IHP 130 nm technology library. The translation is performed using the tool Design Compiler by Synopsys. Due to the specific methodology for design and implementation of the framework in processor-based systems, a non-standard, bottom-up approach was used to perform the synthesis. Concretely, the processor cores and the internal scratchpad memories were synthesized first and afterwards consecutively interfaced at higher hierarchical levels until the top level was reached. Special TCL scripts were created for this purpose. The clock period for all considered designs is set to $T_{clk} = 21$ ns.

All framework-containing systems are synthesized under the same, worst-case conditions which imply traditional guard-banding and slow process corner. In such scenario, the temperature is set to $T = 125$ °C and the supply voltage level to $V_{DD} = 1.08$ V. The presented results show relevant data regarding the structure of the synthesized designs, the occupied area and the estimated power consumption. However, it is crucial to emphasize that, in this case, the synthesis tool estimates the power consumption statically, i.e. without considering the switching activities of the logic gates.

Table 5.1: Synthesis reports for the SWIELD FF and comparison of the results against the Pre-Error Flip-Flop [136] and a classic TMR flip-flop.

	SWIELD FF	**Pre-Error FF** [136]	**TMR FF**
Number of Combinational Cells	16	11	11
Number of Sequential Cells	4	4	3
Number of Nets	26	35	29
Combinational Area (μm^2)	149.31	85.05	69.93
Non-combinational Area (μm^2)	120.96	120.96	90.72
Overall Design Area (μm^2)	**284.71**	**214.11**	**167.88**
Estimated Power Consumption (μW)	**3.25**	**2.83**	**1.98**

Table 5.1 shows the data obtained from the synthesis reports for the SWIELD FF. In addition, the corresponding synthesis results of the Pre-Error Flip-Flop [136] and a conventional TMR flip-flop are included for the purpose of comparison. The decision to compare exactly against these architectures was made based on their similarities to the SWIELD FF in terms of both structure and features. As expected, due to its

higher complexity, the SWIELD FF occupies more area and is estimated to consume more power compared to the both Pre-Error Flip-Flop and TMR flip-flop. Concretely, one SWIELD FF is approximately 25% larger in area than a Pre-Error Flip-Flop and around 41% larger than a TMR flip-flop. Additionally, the power consumed by a SWIELD FF is approximated to be almost 13% and 39% higher than the power consumption of a Pre-Error Flip-Flop and a TMR flip-flop respectively. However, as it will be shown in the next chapter, this does not mean that a processor-based system relying on Pre-Error Flip-Flops or on TMR flip-flops is more power-efficient than an identical system based on SWIELD FFs. On the contrary, the ability provided by the framework to dynamically switch between the SWIELD FF operation modes allows a system not only to work in power-aware manner, but also to achieve a trade-off between power consumption and error resilience (see Section 6.2).

Table 5.2: Synthesis report data for the SOMU. Implementations suitable for both single- and multicore processor-based systems are considered.

	SOMU (single-core)	**SOMU (quad-core)**
Number of Cells	1966	13217
Number of Nets	2154	15431
Combinational Area (mm^2)	0.014	0.087
Non-combinational Area (mm^2)	0.008	0.040
Overall Area (mm^2)	**0.023**	**0.136**
Estimated Power Consumption (mW)	**0.137**	**0.729**

The synthesis results for the SOMU are given in Table 5.2. Thereby, two different instances are considered: one suitable for a single-core system and the other for a quad-core multiprocessor implementation. Although the latter is almost six times larger in area than the former, it accounts only for 0.9% of the entire quad-core system area (Table 5.4). In the case of the singe-core implementation, it turns out that the SOMU occupies only 0.5% of the system area (Table 5.3).

Table 5.3: Synthesis report data for the single-core processor-based system. The column $N_{SFF} = 0$ designates a baseline design, that is, a system without framework.

Number of SWIELD FFs (N_{SFF})	**0**	**55**	**115**	**135**	**150**
Number of Cells	55744	56840	57453	57931	58385
Number of Nets	59547	59874	61736	62374	62948
Overall Area (mm^2)	**4.508**	**4.515**	**4.524**	**4.529**	**4.533**
Estimated Power Consumption (mW)	**4.228**	**4.618**	**4.709**	**4.738**	**4.761**

The framework with varying number of SWIELD FFs was embedded into several instances of single- and quad-core processor-based systems. Tables 5.3 and 5.4 show how the area and the estimated power consumption of the different instances are affected by the framework components depending on the SWIELD FF count. For all

Table 5.4: Synthesis report data for the multicore processor-based system. The column $N_{SFF} = 0$ designates a baseline design, that is, a system without framework.

N_{SFF} **per core**	**0**	**55**	**115**	**135**	**150**
Number of Cells	240355	237369	244874	246372	248535
Number of Nets	250824	251809	261234	263372	266015
Overall Area (mm^2)	**11.938**	**11.941**	**12.017**	**12.037**	**12.057**
Estimated Power Consumption (mW)	**14.226**	**15.018**	**15.379**	**15.487**	**15.586**

instances comprising the framework, it was determined that the critical flip-flops to be replaced reside in the Integer Units (IUs) [3] of the processor cores.

The critical flip-flop replacement was performed according to the procedure described in Section 5.1. Under the specified synthesis conditions, the `while` loop in Algorithm 1 could only be exited after reducing T_{dw} to 40% of the default T_{dw} value. This corresponds to 150 critical flip-flops or around 13% of the flip-flops in the IU. To observe potential trends, the T_{dw} was further reduced to 30%, 20% and 10% of the default T_{dw}. The processor system instances implemented to reflect these values of T_{dw} were determined to contain 135, 115 and 55 critical IU flip-flops respectively. Expressed in percentages, 12%, 10% and 5% of the flip-flops in each core IU were replaced with SWIELD FFs.

Instances of processor-based systems without the cross-layer framework, otherwise identical were used as **baseline designs**. The baseline designs are actually reference points for quantification of the framework influence on processor-based systems. In total, two baseline designs were implemented: one single-core and one quad-core system.

Furthermore, the presented integration strategy was utilized for insertion of Pre-Error and TMR flip-flops into both single-/quad-core system instances. The goal was to implement similar systems that rely on different enhanced flip-flop architectures. By doing so, it is possible to expand the comparison beyond the baseline designs, primarily regarding the power consumption. Comparing error resilience to such systems does not make much sense because the Pre-Error Flip-Flops prevent exclusively timing errors, whereas TMR flip-flops are expected to yield similar results as the SWIELD FFs. Note that for each system instance with framework relying on SWIELD FFs, there are two corresponding instances with equal number of Pre-Error/TMR flip-flops respectively.

By observing the data from Tables 5.3 and 5.4, it can be noticed that the overall area of processor-based systems containing the framework increases with the number of inserted SWIELD FFs. However, for both the single- and multicore implementations, the framework-imposed area overhead is significantly low. In the case of single-core systems, even when $N_{SFF} = 150$, the framework accounts only for 0.5% area increase compared to the baseline design. Insertion of a framework with 150

[3] The IU actually contains the processor pipeline, the register file as well as the hardware multiplier and divider.

SWIELD FFs per core in a quad-core system results in 1.1% larger area than the corresponding baseline design.

Regarding power consumption, the synthesis tool estimates up to 12.6% and 9.6% increases compared to the baseline designs for the single-core and quad-core systems respectively. These estimations refer to frameworks with $N_{SFF} = 150$. However, the synthesis tool does not provide realistic data with respect to the power consumption of systems comprising the framework. As it will be explained in Chapter 6, the framework utilization does not lead to increased power consumption.

Chapter 6

Evaluation

In order to determine whether the projected dissertation goals are accomplished, the proposed approach must be properly evaluated. To recap, the main hypothesis of this work states that the cross-layer framework is able to synergistically improve both power consumption and error resilience in processor-based systems by providing adaptability. Hence, it is necessary to observe system operation while the framework is being actively used and to appropriately assess the parameters of interest. At the same time, the influence of the framework on other important factors such as performance and area needs to be taken into account too. Evaluating approaches that rely on complex designs like processors is quite challenging and may take a lot of time. Nevertheless, conducting experiments and investigating their outcome could be helpful in drawing conclusions necessary to prove or disregard the main hypothesis.

This chapter thoroughly elaborates on the experimental setup and the conducted experiments. Thereby, the obtained results are used for quantitative evaluation of the proposed concept in distinct scenarios. Section 6.1 gives a general overview of the used evaluation methodologies and tools. The results are given in Section 6.2. Individual subsections present the findings for different scenarios. Both single- and quad-core LEON2-based systems are utilized as test vehicles. Section 6.3 wraps up this chapter with a results-related discussion.

6.1 Methodologies and Tools

Following a successful framework integration in processor-based systems, the next step is to investigate how the systems' operation is affected, especially regarding the parameters important for this work. Numerous experiments were performed for that purpose. In general, the conducted experiments can be classified in two groups:

- experiments for power consumption estimation;
- experiments for error resilience evaluation.

All experiments were performed as post-synthesis gate-level timing simulations on single- and quad-core systems. Wherever applicable, the same set of experiments were conducted also on baseline/Pre-Error-/TMR-based designs for comparison purposes.

Prior to performing the experiments, it is of utmost significance to ensure that the processor-based systems are functioning correctly. LEON2 is supplied as a completely verified, ready-to-use model. Nevertheless, it is still necessary to confirm the system correctness following the insertion of the framework. Therefore, an exhaustive testbench is prepared and run. On top of the default test programs included in the LEON2 IP core package, the testbench is augmented with the FFL procedures and some extra test functions. The additional test functions are simple but practical as they generate various instruction types. This is extremely important because voltage reduction through AVFS depends directly on the input patterns to the ISMs [136], i.e., the SWIELD FFs. A short description of the additional test functions is given below:

- `*int mulTest(*int, *int)` - returns the element-wise product of two integer arrays. The main purpose of this function is to test the hardware multiplier by generating multiply instructions.
- `*int divTest(*int, *int)` - function intended to test the hardware divider by generating mainly division instructions. It performs an element-by-element division of two integer arrays.
- `*int addArrays(*int, *int)` - computes the element-wise algebraic sum of two integer arrays by generating mostly addition/subtraction instructions.
- `*int qSort(*int)` - recursive function that returns a sorted array of integers according to the QuickSort algorithm. It generates logical and control transfer instructions among others.

Figure 6.1 shows a wrapper function that serves as a system test program used for realization of the planned experiments. It takes an input parameter that reflects the SWIELD FF operation mode to be experimented on. When the wrapper function is called, the actual operation mode is automatically passed to the FFL procedure `setSWIELDMode(mode)` as an argument. Hereby, the SWIELD FFs in the system are effectively put to the adequate operation mode.

```
void sysTestProgram(MODE_TYPE mode) {
  int N = 5000;                              //number of array elements
  int a[N], b[N], c[N], d[N], z[N];           //declare integer arrays
  setSWIELDMode(mode);                //set the adequate SWIELD_FF mode
  c = mulTest(a,b);                      //test the hardware multiplier
  d = divTest(a,b);                        //test the hardware divider
  z = addArrays(c,d);                    //compute the resultant array
  qSort(z);                                 //sort the resultant array
}
```

Figure 6.1: A wrapper function serving as a simple system test program. The execution time can be controlled by changing the `N` parameter.

The wrapper function loads two integer arrays from memory and calls the additional test functions which perform the previously described operations on the arrays. To ensure diversity of input patterns for the SWIELD FFs, the arrays are filled by alternating positive, negative, small and large numbers as inputs. The resultant array is stored back to memory. Note that the program execution time is defined by the number of elements in the arrays.

6.1.1 Power Consumption Estimation

If the wrapper system test function is called with argument `mode=ISM_FF-MAN_FS` or `mode=ISM_FF-AUTO_FS`, the framework will adjust the system supply voltage according to an AVFS scheme. The voltage scaling algorithm has been already described in Subsection 4.1.1 and the actual AVFS parameters are stored in the corresponding SOMU registers. For quantification of the power saved by the AVFS scheme, the same set of experiments was conducted also on baseline/Pre-Error-/TMR-based systems. Then, a comparative analysis of the results was performed.

A post-synthesis simulation relying on conventionally-characterized standard cell library is unable to precisely calculate power consumption if the supply voltage is adaptively scaled. Therefore, to come up with reasonable power estimations, several assumptions based on real-world data are made:

- the operation begins at nominal supply voltage level $V_{DD} = V_{max} = 1.2$ V for all considered systems;
- a baseline/TMR-based system maintains the nominal supply voltage level throughout the entire operation;
- the supply voltage of a system relying either on the cross-layer framework or on Pre-Error flip-flops is gradually reduced by a voltage regulator from nominal 1.2 V at the operation start (t_{start}), to the lowest possible value V_{opt} that will not cause timing errors;
- the scaled supply voltage settles at level $V_{opt} = 1.08$ V at time t_{settle} and maintains the same value until the execution of the test program is finished (t_{end});
- a voltage regulator is able to scale the supply voltage from the nominal value of 1.2 V down to a minimal value $V_{min} = 0.8$ V by 20 discrete steps of 20 mV each;
- the voltage regulator has settling time of approximately 500 ns, i.e. it takes around 500 ns for the voltage regulator to shift from one voltage level to the next [120, 136].

Given that a processor-based system operates under the assumed conditions, its power consumption can be calculated as:

$$P_{sys} = \sum_{i=1}^{n} P_i \Big|_{t_{start}}^{t_{end}} = \sum_{i=1}^{n} (P_i \Big|_{t_{start}}^{t_{settle}} + P_i \Big|_{t_{settle}+\Delta t}^{t_{end}}) \tag{6.1}$$

where P_i denotes an active processor core in an n-core system, $i = n = 1$ for a single-core system while $i \in \{1, 2, 3, 4\}$ and $n = 4$ in case of a quad-core system. For simplicity, the dissipation of the voltage regulator and the additional AVFS-related components is not considered. Furthermore, it is assumed that the power consumed by the remaining, non-core system components is included in P_i.

Equation 6.1 can be applied to both baseline/Pre-Error-/TMR-based systems and systems with integrated framework. Note that for a system relying on the framework or on Pre-Error flip-flops, the first addend in the equation will always have lesser value than the corresponding baseline/TMR-based system. This is because the supply voltage in the former case scales from V_{DD} to V_{opt} during the interval $[t_{start}, t_{settle}]$, whereas in the latter case it stays fixed at V_{DD}. To determine the value of the second addend in Equation 6.1, a dynamic time-based calculation is performed. This method takes into account the switching activities of all gates in the design during simulation time. It was practically implemented using the commercial tool PrimePower from Synopsys [124]. The tool requires switching activity file (usually .vcd or .saif) as an input to perform the calculation. For the purpose of this work, such files were generated within the environment of the simulator Xcelium from Cadence [71].

In order to mimic behaviour of a voltage regulator as well as supply voltage reduction, the synthesis of baseline/TMR-based systems was performed under different conditions compared to systems relying on the framework or on Pre-Error flip-flops. While the latter were synthesized under worst case conditions (see Section 5.3), the former were synthesized under typical case conditions which implies supply voltage level of $V_{DD} = 1.2$ V and temperature of $T = 25$ °C.

Essentially, the power estimation tool starts calculating at simulation time $t_{settle+\Delta t}$ and stops at t_{end}. This is applied to all considered systems, despite the fact that baseline/TMR-based designs do not perform AVFS. To ensure results consistency, the time $t_{settle+\Delta t}$ valid for the corresponding AVFS-capable systems [1] is mapped as a starting point for power consumption calculation also for baseline/TMR-based systems. Finally, the obtained results for every simulation scenario are compared against each other.

Table 6.1: Default values of the AVFS scheme parameters.

Parameter/SOMU register	**Default Value**
MODE	01 (ISM FF - Manual FS activation)
MAX TRANSITIONS	1000
MIN PRE-ERRORS	1
MAX PRE-ERRORS	20
CLOCK DIVIDER ENABLE	0 (FS disabled)

Table 6.1 shows the default values of the framework parameters related to the implemented AVFS scheme. It is possible to modify these values at any time while the system is active by calling the appropriate FFL procedures. However, an AVFS-

[1] Systems with equivalent number of Pre-Error/SWIELD FFs.

based operation is always initiated with the default parameter values because they are written into the corresponding SOMU registers immediately following system reset.

6.1.2 Error Resilience Evaluation

As pointed out in Subsection 2.3.1, fault injection is a widely-used method for fault forecasting. It is accomplished by deliberate insertion of faults in a system with fault protection mechanisms. Thereby, the system behaviour in a presence of faults is monitored. Quantification of the system error resilience can be performed by observing how many faults resulted in errors. Finally, the effectiveness of the used fault-tolerant mechanisms is evaluated based on the FI results.

FI can be performed in several ways and at different system levels. Generally, the FI techniques may be divided into the following categories [143, 78]:

- hardware-based fault injection - realized at physical level, that is, faults are injected into the actual hardware of the target system. Some of the methods utilized for hardware FI include: heavy-ion radiation, electromagnetic interference, laser fault injection, introduction of power supply disturbances, modification of the circuit pin values etc. Although hardware FI provides the most realistic scenario for exposing the system to faults, the practical realization of this technique is very expensive and it has potential to damage the system under test.

- simulation-based fault injection - unlike the hardware-based FI, the simulation-based FI uses modelled systems and faults. System modelling can be done at different abstraction levels (behavioural, RTL, gate), whereas fault injection and monitoring of the consequent system behaviour are performed using dedicated software tools. Simulation-based FI is cheaper and does not pose a risk of damaging the target system. It also allows evaluation of the error resilience early during the system design flow. However, using models reduces the accuracy of the entire process and may drastically increase the simulation time depending on the model abstraction level.

- emulation-based fault injection - relies on Field Programmable Gate Arrays (FPGAs) for system emulation and speeding-up of the fault simulation. It has been introduced as an alternative approach that reduces the execution time in comparison to the simulation-based FI.

In this work, the Xcelium Fault Simulator (XFS) from Cadence is used for error resilience evaluation. XFS is a commercial tool that is part of an end-to-end flow for system design and verification [70]. By using XFS it is possible to perform simulation-based fault injection which consists of several phases:

- fault instrumentation - selecting the potential fault nodes in the system and specifying the fault models to be injected. Fault node can be a signal, a gate or an entire module. Supported fault models are: Stuck-at-0, Stuck-at-1, SEU

and SET. One or more fault models may be specified for each fault node by writing appropriate commands in a *fault specification file*.

- good simulation - generating reference values for the fault simulation. During this phase, so-called *strobe points* are defined. The strobe points allow monitoring of specific signals in the system whose values are recorded and used for comparison in the next phase. There are two types of strobe points:
  - functional - usually the system primary outputs should be specified as functional strobe points.
  - checker - certain internal system nodes that are of particular interest for observation during the FI campaign may be selected as checkers.
- fault simulation - injecting of faults is performed. The FI campaign may be:
  - random - XFS relies on the contents in the fault specification file. The fault nodes, fault types and the injection times are randomly selected.
  - targeted - dedicated command is used to specify the fault nodes, fault types and injection times. The contents of the fault specification file are ignored.

  The strobe point values in this phase are compared to the ones recorded during the good simulation.
- fault reporting - generation of post-simulation reports which help determine the status of the injected faults. An injected fault can be either:
  - detected - different values have been recorded on the strobe points during good and fault simulations;
  - undetected - equal values have been recorded on the strobe points during good and fault simulations.

  Depending on the status recorded on the functional and checker strobe points, a fault can be further classified as:
  - dangerous detected - fault is detected on both functional and checker strobe points;
  - dangerous undetected - faults are detected only on functional strobe points;
  - unobserved detected - faults are detected only on checker strobe points;
  - unobserved undetected - faults are undetected on both functional and checker strobe points.

  Note that if a fault is detected at a functional strobe point, it means that an error propagated to the system outputs and potentially caused a failure. On the other hand, if a checker detects a fault, the system should be able to take action and prevent failure from occurring. Based on this information, the status of an injected fault in a system can be defined as follows:

– **detected and tolerated by a SWIELD FF** - if XFS classified it as unobserved detected;
– **logically masked** - if XFS classified it as unobserved undetected;
– **propagated** - if XFS classified it either as dangerous detected or as dangerous undetected.

The XFS fault injection flow is illustrated in Figure 6.2. As can be seen, to run FI campaigns, it is necessary to feed the XFS environment with specific data that describes the system structure as well as its functional and timing behaviour. A gate-level netlist provides information regarding the structure, whereas the functional behaviour is defined in a testbench. The timing behaviour can be derived from a post synthesis-generated file [2] that contains the delays of all gates and interconnects in the considered system.

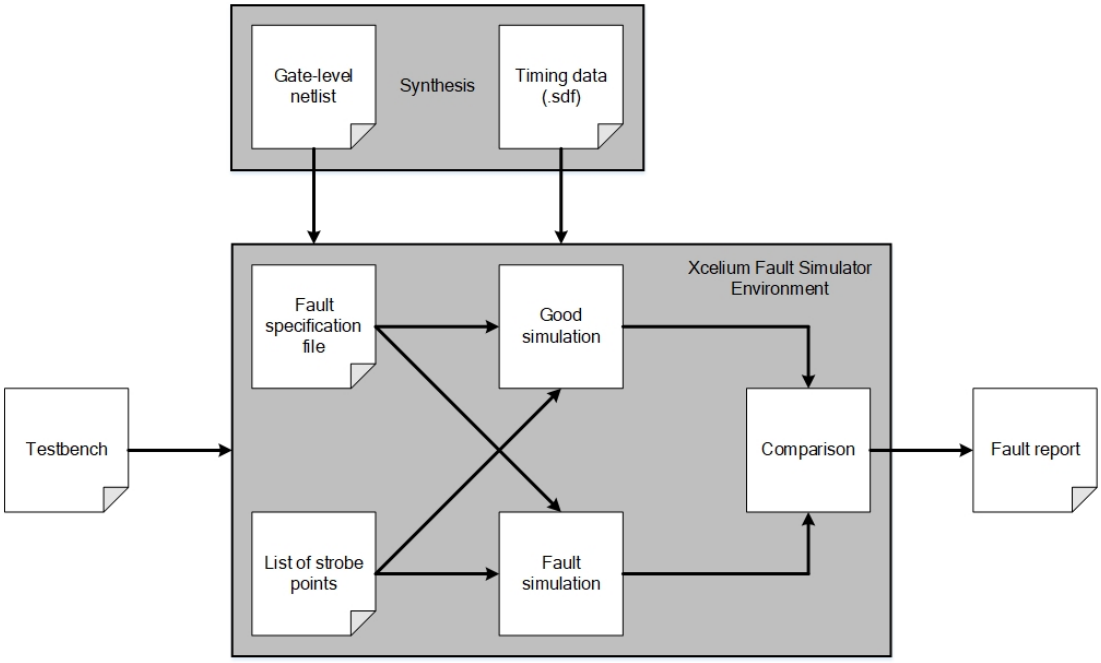


Figure 6.2: Fault injection flow of the Xcelium Fault Simulator.

When the wrapper system test function from Figure 6.1 is called with argument `mode=TMR`, the framework enables protection against SEUs. To evaluate system error resilience provided by the SWIELD FFs in TMR operation mode, numerous FI campaigns are performed. The base FI scenario is defined as follows:

- all injected faults are of type SEU;
- all faults are injected into the IU(s); [3]
- all faults are injected at random times;

[2]Typically, the Standard Delay Format (SDF) is used for this purpose.
[3]Recall that all SWIELD FFs are located in the IU(s).

- all primary outputs of the system are selected as functional strobe points;
- all SWIELD FFs are selected as checker strobe points.

The number of injected faults is correlated to the system test execution time using a metric called *mean fault injection rate* ($\overline{FI_r}$). This metric is introduced in order to ensure that the faults are injected in reasonable quantities. For each FI campaign, the mean fault injection rate can be calculated according to the following relationship:

$$\overline{FI_r} \approx \frac{1\ fault \cdot N_{clk\_cycles/sim\_length}}{4000\ clk\_cycles} \tag{6.2}$$

XFS does not take supply voltage into account and hence, lacks a built-in functionality for correlation of FI rates to supply voltage. To overcome this limitation, having in mind the model discussed in Section 1.2 which states that SER increases exponentially with supply voltage reduction, voltage-dependent FI rates can be calculated according to Equation 1.1. Let the mean fault injection rate correspond to the nominal supply voltage level $V_{DD} = V_{max}$. By replacing λ_0 with $\overline{FI_r}$ and setting $d = 1$ for simplicity, the equation can be rewritten as:

$$FI_r(V) = \overline{FI_r} 10^{\frac{V_{max} - V}{V_{max} - V_{min}}} \tag{6.3}$$

Assuming that the supply voltage level of the considered system was previously reduced by an AVFS scheme from $V_{DD} = V_{max}$ to V_{opt} (see Subsection 6.1.1), the corresponding voltage-dependent fault injection rate can be expressed as:

$$FI_r(V_{opt}) = \overline{FI_r} 10^{\frac{V_{max} - V_{opt}}{V_{max} - V_{min}}} \tag{6.4}$$

If the actual values for V_{max}, V_{opt} and V_{min} from Subsection 6.1.1 are replaced in Equation 6.4, then it simplifies to:

$$FI_r(V_{opt}) = \overline{FI_r} 10^{0.3} \approx 2 \cdot \overline{FI_r} \tag{6.5}$$

which means that when the system operates at supply voltage level V_{opt}, it should expect approximately twice more faults compared to the case when it operates at nominal level $V_{max} = V_{DD}$. For observation of possible trends, additional FI rates can be obtained by halving/quartering the mean FI rate and doubling the voltage-dependent FI rate.

As it will be shown in next sections, the actual system execution times and therefore, the FI rates differ from scenario to scenario.

6.2 Results

This section presents the experimental results and the corresponding analyses regarding error resilience and power consumption of the processor-based systems in which the proposed framework was integrated. The results are organized according to the

number of processor cores in each considered system and the corresponding operation scenario.

The first scenario deals with a minimal, single-core system (see Section 5.2.1). Such a design can plausibly demonstrate fundamental processor system operation and thus, it was considered suitable for providing preliminary evaluation of the framework. On the other hand, both the second and the third scenario involve much more complex, quad-core system. As pointed out, despite standard, high performance-oriented operation, the multiprocessor allows advanced functionalities such as switching to aging-aware mode. Therfore, Scenario II investigates how the framework influences the operation of a quad-core system in HPSM, whereas Scenario III deals with a multiprocessor switched to operate in DSSM.

The complete results regarding Scenario I have been published in the papers [VHKK20, VHKK21a], whereas the paper [VHKK21b] presents the findings regarding Scenarios II and III.

6.2.1 Scenario I: Single-Core Processor System

The single-core systems have not been completely abandoned. In fact, they are still being used today, mainly in the domain of low-power microcontroller design. For example, it has been shown that parallel architectures are not suitable for energy-constrained systems like nano-size drones due to their specific requirements in terms of energy, peak-power and predictability [40, 98]. Therefore, it is to expect that the single-core processors will continue to be relevant at least for the next decade [4].

Power Consumption

Table 6.2 summarizes the power consumption results for all single-core processor system instances. The results are organized according to the **number of enhanced flip-flops** (N_{EFF}) in each instance. Of course, a baseline design does not contain any enhanced flip-flops. On the other hand, the number of enhanced flip-flops in Pre-Error/TMR-based instances is determined by the number of SWIELD FFs (N_{SFF}) in each considered system that incorporates the cross-layer framework.

Table 6.2: Power consumption results (in mW) for single-core systems.

		Instance ($N_{EFF} = N_{SFF}$)				
		0	55	115	135	150
Type of system	Baseline	6.06	N/A	N/A	N/A	N/A
	Pre-Error FF-based	N/A	5.08	5.19	5.24	5.3
	TMR FF-based	N/A	5.99	6.07	6.1	6.12
	SWIELD FF-based	N/A	5.13	5.2	5.26	5.29

It is easily noticeable that the systems able to perform AVFS are more power-efficient than the baseline design. Concretely, the cross-layer framework-based instances can save between 15% ($N_{SFF} = 55$) and 13% ($N_{SFF} = 150$) power, whereas

the instances relying on Pre-Error flip-flops consume from 16% ($N_{EFF} = 55$) to 13% ($N_{EFF} = 150$) less power in comparison to the baseline design. Hence, if the only goal is to reduce the power consumption as much as possible, the obvious solution would be to implement a system based on Pre-Error Flip-Flops replacing approximately 5% of the timing-critical flip-flops. However, lower N_{EFF} means shorter T_{dw} which, in fact, increases the risk of timing errors (see Subsection 3.3.1). On the other hand, if error resilience is required in addition to power efficiency, a system with integrated cross-layer framework would be a better choice since the Pre-Error Flip-Flops protect exclusively against timing errors. An interesting feature to observe in Table 6.2 is that the power-efficiency difference between the framework-based systems and the instances relying on Pre-Error flip-flops decreases as the N_{EFF} increases. Note that when $N_{EFF} = N_{SFF} = 150$, the framework-based system relying on SWIELD FFs saves more power than the corresponding instance relying on Pre-Error flip-flops.

Figure 6.3 illustrates how a framework integration affects the area and the power consumption of a single-core system depending on N_{SFF}. The graphs are based on data from Table 5.3 regarding area and Table 6.2 regarding power consumption.

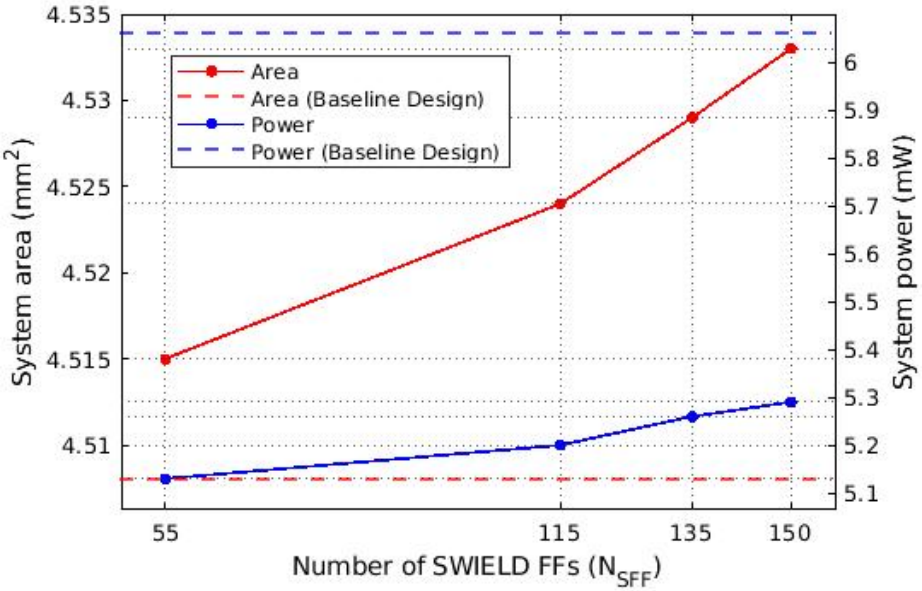


Figure 6.3: Area and power consumption as functions of the number of inserted SWIELD FFs in a single-core system. The dashed lines indicate area and power consumption of the baseline design.

As expected, the results in Table 6.2 confirm that the TMR-based instances are the most power-hungry of all considered systems. Just like the baseline design, the instances relying on TMR flip-flops are incapable of voltage scaling. The fact that such systems maintain supply voltage at nominal level ($V_{DD} = V_{max}$) during the entire operation, makes them unsuitable for low power computing.

Finally, it should be emphasized that all processor system instances relying on enhanced flip-flops have gone through synthesis and thus, through an optimization

process twice. This is in accordance with the presented framework integration strategy, or more precisely, with the SWIELD FF insertion procedure (see Section 5.1 on page 81). In contrast, the baseline design is implemented according to conventional design flow and therefore, it is synthesized only once. As a result, the power calculation tool estimated that the TMR-based instance from Table 6.2 with $N_{EFF} = 55$ consumes less power than the baseline design. In practice, unfortunately, every system that relies on TMR will be consuming more power than its corresponding baseline design no matter how many times it goes through an optimization process. The Pre-Error/SWIELD FF-based systems however, save power due to their capability to perform AVFS. As explained previously, the amount of saved power by AVFS depends on many factors.

Error Resilience

It was stated in Subsection 6.1.2 that the mean fault injection rate $\overline{FI_r}$ for error resilience evaluation experiments is correlated to the test program execution (simulation) time of each scenario. In the case of single-core processor system, the simulation is about 371,028 ns long, which corresponds to approximately $N = 17,670$ clock cycles. By replacing the value for N in Equation 6.2, the mean fault injection rate for this scenario can be fixed to $\overline{FI_r} = 5\ faults/execution\_time$.

Table 6.3 shows the fault injection results for a single-core system that contains the proposed cross-layer framework. As already pointed out, the faults which are randomly injected directly into one of the constituent flip-flops within the SWIELD FF are considered *detected* because the SWIELD TMR structure outvotes such faults. On the other hand, all faults observed at the primary outputs of the system are classified as *propagated* faults.

Table 6.3: Fault injection results for a single-core system. The status of the injected faults is denoted as: D - detected and tolerated by SWIELD FF in TMR mode; M - logically masked; P - propagated.

	Fault injection rate														
	$\overline{FI_r}/4 \approx 1$			$\overline{FI_r}/2 \approx 3$			$\overline{FI_r} = 5$			$2 \cdot \overline{FI_r} = 10$			$4 \cdot \overline{FI_r} = 20$		
	Status														
N_{SFF}	D	M	P	D	M	P	D	M	P	D	M	P	D	M	P
55	1	0	0	0	3	0	1	3	1	3	6	1	5	14	1
115	1	0	0	1	1	1	2	3	0	6	3	1	8	11	1
135	1	0	0	1	1	1	2	3	0	2	7	1	7	11	2
150	1	0	0	1	2	0	2	3	0	5	5	0	11	7	2

One tendency is rather obvious in Table 6.3: N_{SFF} is directly proportional to the probability that a SWIELD FF will be target of a fault. In other words, higher number of SWIELD FFs in the system increases the chances that a fault will be detected and outvoted. That being said, it is of key importance to note that the detected faults are in fact potentially propagated and failure causing faults for a

baseline design. This is a clear indication that the framework significantly increases the system error resilience.

Another interesting observation that arises from the results in Table 6.3 is the following: during some fault injection campaigns, the system experienced no propagated faults to the primary outputs, that is, the test program execution finished failure free. The shaded cells in the table denote those FI campaigns. For example, such outcomes are observed during the campaigns with FI rate $\overline{FI_r}/4$ in all cases of N_{SFF}. It means that the proposed framework might be sufficient for system protection against SEUs in environments where lower SERs are expected. Additionally, it is crucial to emphasize that the test program finished without failures also following more intensive FI campaigns, mainly when $N_{SFF} = 150$. Especially significant are the cases of fault injection rates $\overline{FI_r}$ and $2 \cdot \overline{FI_r}$. Recall that the former corresponds to a system operating at supply voltage $V_{DD} = V_{max}$, whereas the latter designates a FI rate that the system can expect when its voltage level is reduced to V_{opt}. Therefore, the framework is capable to keep a system error resilient while it operates at lower supply voltage level for power efficiency.

6.2.2 Scenario II: Multicore System for High Performance

The fundamental idea behind the concept of multiprocessing was to increase the computation speed. To avoid overheating and potential system breakdown, operating frequencies and supply voltages were reduced, whereas performance was boosted by leveraging core-level parallelism. Nevertheless, keeping all available cores active for high-speed processing is expensive in terms of power. In addition, more active cores imply larger area susceptible to faults.

To investigate how the proposed framework influences the power consumption and the error resilience of a high performance multiprocessor, the same experiments were performed on a quad-core system operating in HPSM. This subsection presents the obtained results.

Power Consumption

The experimental results regarding power consumption of the implemented quad-core processor system instances operating in HPSM are shown in Table 6.4. Similar trends to those elaborated in Scenario I can also be observed here. Namely, the instances which leverage AVFS are able to save notable amount of power with respect to the baseline design: the systems that perform supply voltage scaling using Pre-Error Flip-Flops are between 15% ($N_{EFF} = 150$) and 17% ($N_{EFF} = 55$) more power-efficient while the instances relying on the proposed framework and SWIELD FFs can reduce the power consumption from 12% ($N_{SFF} = 150$) to 15% ($N_{SFF} = 55$). Once again, a system employing TMR flip-flops, even only in selected locations, cannot contribute to power saving, despite the simulation results shown in Table 6.4 indicating between 0.5% ($N_{EFF} = 150$) and 2% ($N_{EFF} = 55$) power reduction. As emphasized in the previous subsection, such outcome is due to the multiple optimization cycles applied to the TMR-based instances.

Table 6.4: Power consumption results (in mW) for a quad-core multiprocessor system operating in HPSM.

		Instance ($N_{EFF} = N_{SFF}$)				
		0	55	115	135	150
Type of system	Baseline	21.3	N/A	N/A	N/A	N/A
	Pre-Error FF-based	N/A	17.6	18.0	18.1	18.2
	TMR FF-based	N/A	20.8	21.1	21.2	21.3
	SWIELD FF-based	N/A	18.1	18.4	18.7	18.8

The influence of the cross-layer framework on a quad-core multiprocessor for high performance operation regarding area and power consumption is pictorially shown in Figure 6.4. The graphs are constructed using data from Tables 5.4 and Table 6.4 respectively. Note that the x-axis reflects the number of inserted SWIELD FFs in each processor core. By comparing the graph shapes in Figures 6.3 and 6.4, one can easily conclude that there is an obvious consistency between the results from Scenarios I and II, especially regarding power consumption.

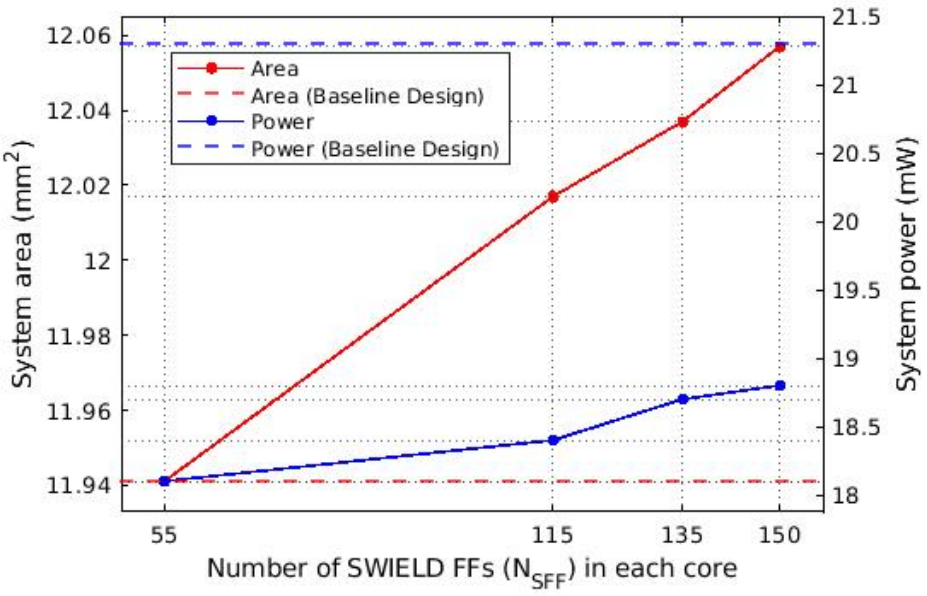


Figure 6.4: Area and power consumption as functions of the number of inserted SWIELD FFs in each core of a quad-core system for high performance operation. The dashed lines indicate area and power consumption of the baseline design.

Error Resilience

When a quad-core multiprocessor system is actively using all of its cores for high performance operation, the test program runs for 817,026 ns or equivalently, around

$N = 38,906$ clock cycles. Therefore, according to Equation 6.2, the mean fault injection rate for this scenario would correspond to $\overline{FI_r} = 10\ faults/execution\_time$.

Table 6.5 presents the results from the fault injection campaigns performed on a quad-core multiprocessor operating in HPSM. As can be seen, the percentage of detected faults is rather high. Actually, it reaches 50% for the campaigns with FI rates $\overline{FI_r}$, $2 \cdot \overline{FI_r}$ and $4 \cdot \overline{FI_r}$ in the case when the number of SWIELD FFs per core is $N_{SFF} = 150$. Moreover, for the same N_{SFF} value, the test program finishes successfully without propagated faults following most of the FI campaigns. Identically to Table 6.3, this is true for the campaigns with FI rates $\overline{FI_r}$ and $2 \cdot \overline{FI_r}$ (see the shaded cells in Table 6.5). Hence, the capability of the cross-layer framework to provide error resilient and power-efficient processor system operation is once again reaffirmed in this Scenario.

Table 6.5: Fault injection results for a quad-core multiprocessor system operating in HPSM. The status of the injected faults is denoted as: D - detected and tolerated by SWIELD FF in TMR mode; M - logically masked; P - propagated.

	Fault injection rate														
	$\overline{FI_r}/4 \approx 3$			$\overline{FI_r}/2 = 5$			$\overline{FI_r} = 10$			$2 \cdot \overline{FI_r} = 20$			$4 \cdot \overline{FI_r} = 40$		
N_{SFF}	Status														
	D	M	P	D	M	P	D	M	P	D	M	P	D	M	P
55	0	3	0	1	4	0	2	8	0	2	15	2	7	27	4
115	1	1	1	2	3	0	5	4	1	7	12	1	15	21	3
135	1	1	1	2	3	0	5	1	1	7	10	3	16	21	3
150	1	2	0	2	3	0	5	5	0	10	10	0	20	18	1

6.2.3 Scenario III: Multicore System for Prolonged Lifetime

If the current application does not require high performance processing, the system may rely only on one active core, while the rest could be deactivated. Generally, it might be necessary to use more than one, i.e., minimum required active cores to successfully run the application. Of course, the exact number of active cores should be determined according to the application requirements. In this scenario, only one active core at a time is considered.

Core deactivation can be realized by cutting off its clock or power supply. Thus, the workload is handled by the currently active core for a defined time period. Afterwards, a previously idle core is activated and the workload is transferred to it. Finally, the most recently active core is deactivated. The core activation/deactivation scheme can be systematically repeated according to some algorithm such as round robin. Offloading processing tasks from cores relaxes their activities and hence, slows down their aging. As a result, the useful system lifetime is considerably increased [117].

The third and final evaluation scenario explores the effects of the proposed framework on a quad-core system that is set to operate in DSSM. By activating only one

core at a time, DSSM automatically reduces power consumption. Nevertheless, it may be useful to investigate how the framework can broaden the system applicability by introducing error resilience and power-aware processing. Such features might be of special interest for mission-critical and long life systems like satellites or space crafts.

Power Consumption

Summary of the power consumption evaluation results for quad-core system instances that operate in DSSM is given in Table 6.6. It is interesting to note that here, in contrast to Scenarios I and II, the instances that rely on the cross-layer framework performing AVFS, do not contribute to significant power reduction. For example, the instance with $N_{SFF} = 55$ consumes only 6.5% less power than the baseline design, whereas the one containing $N_{SFF} = 150$ requires 0.5% more power. Even the instances that rely on Pre-Error Flip-Flops for AVFS only slightly reduce the power consumption compared to the baseline design: 7% in the case when $N_{EFF} = 150$ and 9% for the instance that contains $N_{EFF} = 55$. The systems based on TMR flip-flops in this scenario perform worse regarding power compared to the baseline design. This is true almost for all instances using TMR flip-flops, although the difference is not so large due to the multiple optimization cycles.

Table 6.6: Power consumption results (in mW) for a quad-core multiprocessor system operating in DSSM.

		Instance ($N_{EFF} = N_{SFF}$)				
		0	55	115	135	150
Type of system	Baseline	8.67	N/A	N/A	N/A	N/A
	Pre-Error FF-based	N/A	7.89	7.99	8.03	8.06
	TMR FF-based	N/A	8.6	8.68	8.71	8.73
	SWIELD FF-based	N/A	8.11	8.47	8.61	8.71

Figure 6.5 depicts the framework effects on the power consumption of a quad-core system put in DSSM for prolonged lifetime. Note that the area graph is identical to Scenario II simply because the occupied area does not depend on the multiprocessor operation mode/sub-mode. On the other hand, the graph reflecting the power consumptions of the implemented instances demonstrates quite different trends in comparison to the first two scenarios. Such outcome suggests that the framework has limited potential to enhance the power efficiency of a multiprocessor in DSSM. Keeping only one active core already reduces the power consumption of a multicore system significantly. Instead, providing error resilience can be the main focus when utilizing the framework.

Error Resilience

In this scenario, the test program is executed in 1,337,280 ns or in $N = 63,680$ clock cycles. Based on Equation 6.2, the corresponding mean fault injection rate can be set to $\overline{FI_r} = 15$.

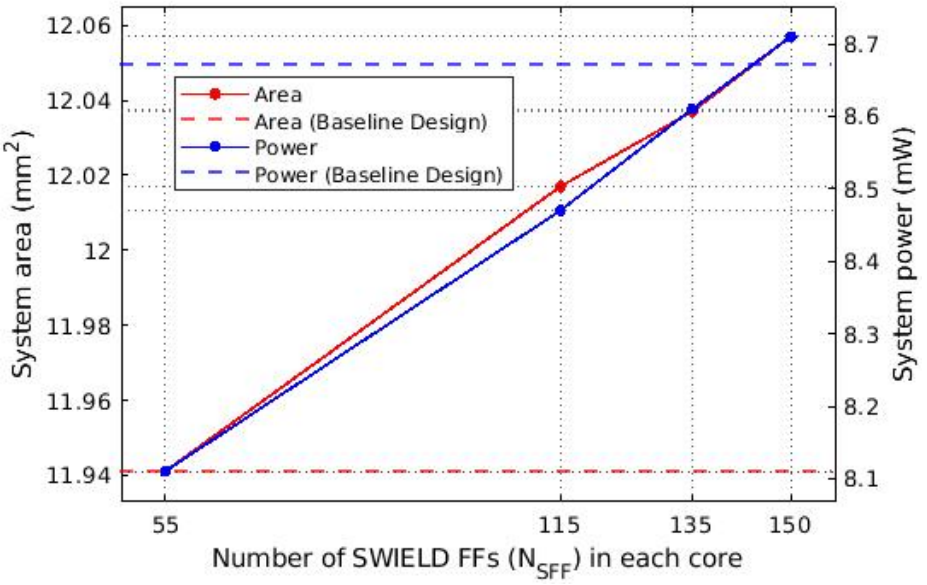


Figure 6.5: Area and power consumption as functions of the number of inserted SWIELD FFs in each core of a quad-core system for prolonged lifetime. The dashed lines indicate area and power consumption of the baseline design.

The fault injection results are shown in Table 6.7. When a system is put in DSSM, only one core operates at a time, whereas faults are scattered throughout the entire multiprocessor. Therefore, a large fraction of the injected faults ends up in an inactive area, which is exactly why the percentages of both detected and propagated faults for this case is lower. Nonetheless, the detected faults percentage may reach almost 30% when $N_{SFF} = 150$.

Table 6.7: Fault injection results for a quad-core multiprocessor system operating in DSSM. The status of the injected faults is denoted as: D - detected and tolerated by SWIELD FF in TMR mode; M - logically masked; P - propagated.

	Fault injection rate														
	$\overline{FI_r}/4 \approx 4$			$\overline{FI_r}/2 \approx 8$			$\overline{FI_r} = 15$			$2 \cdot \overline{FI_r} = 30$			$4 \cdot \overline{FI_r} = 60$		
	Status														
N_{SFF}	D	M	P	D	M	P	D	M	P	D	M	P	D	M	P
55	0	0	0	0	2	1	0	2	1	0	7	1	1	11	4
115	1	1	0	0	1	0	3	2	0	5	3	0	7	3	2
135	1	1	0	0	2	0	2	3	0	6	6	0	7	10	0
150	1	1	0	2	2	0	3	3	0	9	6	0	11	9	1

Obviously, the framework managed to protect the systems against faults during the most of the FI campaigns. However, as the FI rate increases or equivalently, as

the supply voltage would presumably decrease beyond V_{opt}, propagated faults start to appear. To avoid failures, during higher FI rates, additional online error protection mechanisms are required. For example, recall that the multiprocessor can be dynamically put into FTSM. It is of particular interest to determine under what circumstances such step(s) should be taken. Hence, the relationship between a FI rate and the time of the first propagated fault from the corresponding FI rate is investigated. For illustration, let's take as an example the case from Table 6.5 when $N_{SFF} = 150$ and the FI rate is equal to $FI_r = 4 \cdot \overline{FI_r} = 40$. Furthermore, let the occurrence time of the first (and in this case, the only) propagated fault to be recorded during the 28,044 clock cycle. Then, a simple graph which delimits execution times and FI rates without propagated faults can be constructed by drawing a straight line in a log-log plot between two points $M1(t, 1)$ and $M2(1, FI_r(t))$. Here, t denotes the time stamp of the first propagated fault and FI_r is the current FI rate. The graph is presented in Figure 6.6.

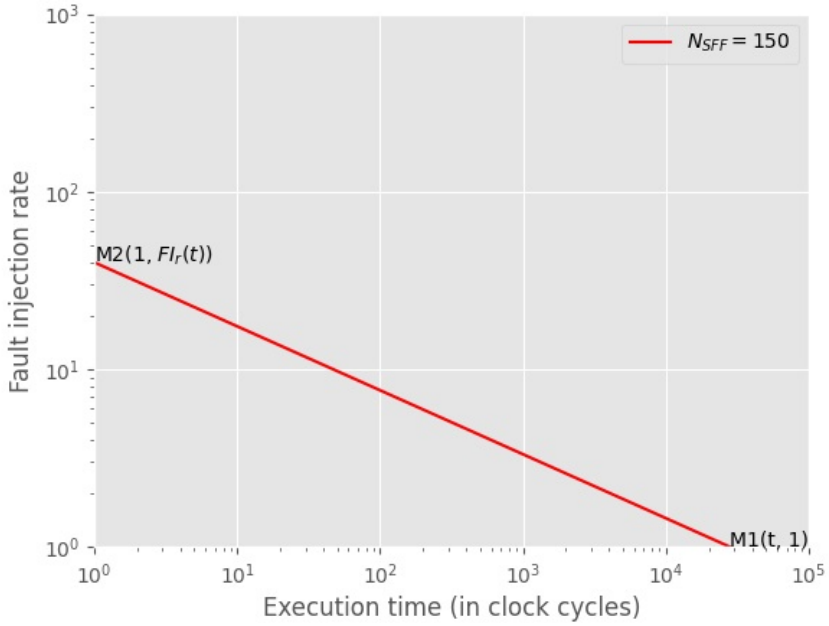


Figure 6.6: Relationship between fault injection rate and a failure-free program execution. The red line delimits whether activation of additional error protection mechanisms is necessary. Logarithmic scale is used on both axes.

An equation of a straight line in the form $logy = mlogx + logb$ between the points $M1$ and $M2$ can be inferred by applying log to the both sides of the template $y = bx^m$ where $logb$ is the intercept of the line with the $logy$ axis and m is the slope of the line. In the concrete case, the two points are defined as follows: $M1(28044, 1)$ and $M2(1, 40)$. After calculating m and $logb$, the equation of the line between $M1$ and $M2$ is:

$$logy = -0.36logx + 1.60 \tag{6.6}$$

If a point lies on the area above this line, the system would require activation of extra online error protection mechanisms in order to finish program execution without failures. For example, a multiprocessor could be put into FTSM, whereas a single-core system might consider using AVFS more conservatively or disabling it completely to reduce the risks of soft errors. A point $M_p(x_p, y_p)$ is above the line $Ax + By + C = 0$ if $y_p - y > 0$, that is, if $y_p > y$. Concretely, a point M_p on the graph from Figure 6.6 is above the line from Equation 6.6 if

$$logy_p > 1.60 - 0.36logx_p \tag{6.7}$$

Therefore, if the program execution time and the system operation environment [4] are known, one can estimate whether the SEU protection provided by the SWIELD FFs, i.e. the cross-layer framework suffices and if not, when additional online error protection mechanisms are required. As already stated in Subsection 4.1.2, SER may be monitored in real time using special particle detectors, thus the system can manage its error protection mechanisms according to the SER intensity.

6.3 Discussion

The experimental results unequivocally showed the advantages of the proposed approach. It was demonstrated in all three scenarios that the cross-layer framework makes the considered processor-based systems more power-efficient and more error-resilient by adaptation to the current application needs and environmental conditions. Indeed, when the SWIELD FFs are put into ISM FF mode, the framework is able to reduce power consumption up to 15%, depending on the N_{SFF} in the system. The amount of saved power is inversely proportional to the number of inserted SWIELD FFs.

By switching the SWIELD FFs to operate in TMR FF mode, the framework significantly increases the system resilience to SEUs. Namely, from each of the many FI campaigns, up to 55% injected faults were tolerated by the SWIELD FFs. Thereby, note that the FI campaigns performed as part of the evaluation experiments are far more intensive than the most severe radiation events. According to Hansen et al. [59], during one of the most extreme SPEs, up to 5 upsets per day have been recorded. [5] For comparison, the highest FI rates in this work can be approximated to $\approx 45,000$ faults per second. In contrast to the experiments for power consumption estimation, it is clear that higher N_{SFF} in the system provides better error resilience. Nevertheless, when at most 13% of the critical flip-flops in the core IU were replaced with SWIELD FFs, the majority FI campaigns finished without failures, including those intended to mimic conditions with lower supply voltages and higher fault rates. However, there is

[4] The expected SER heavily depends on the system operation environment.

[5] The authors presented SEU rate measurements that relied on 4Kx32-bit CMOS SRAMs as particle detectors.

no guarantee that a fault will not propagate to cause a failure, especially if the system is operating under conditions of increased radiation. To avoid failures, a relationship between the execution time and the fault rate is established, thus proper planning for dynamic activation of additional error protection mechanisms can be done in a timely manner.

Table 6.8: Comparison of this work against related works from Chapter 3. The cells marked with * indicate that the location of the enhanced flip-flop depends on the solution of the replacement algorithm.

Reference	Type of Error Mitigation	Timing Error Mitigation Method	Location of TMR flip-flop/ISM
[101]	Soft errors	N.A.	Everywhere
[106]	Soft errors	N.A.	*
[125]	Soft errors	N.A.	*
[81]	Timing errors	Prediction	Intermediate points
[3, 19]	Timing errors	Prediction	Intermediate points
[80]	Timing errors	Prediction	End points
[88]	Timing errors	Prediction	End points
[67]	Timing errors	Detection	End points
[130, 129]	Timing Errors	Detection	End points
[51, 50]	Timing Errors	Detection	DE and EX pipeline stages
[43]	SEUs and Timing errors	Detection	FE, DE, EX and MEM pipeline stages
[136]	Timing errors	Prediction	End points
This work	**SEUs and Timing errors**	**Prediction**	*****

Table 6.9: Comparison of this work against related works from Chapter 3. The minus sign preceding some of the values in the cells from the column Power Overhead, actually indicates power saving. The cells marked with ** indicate that no explicit area overhead is specified, however, some area reduction compared to other approaches is reported.

Reference	Area Overhead	Power Overhead	Performance Degradation	Test Vehicle
[101]	N.A.	N.A.	N.A.	Shift registers
[106]	**	N.A.	N.A.	N.A
[125]	2%	N.A	N.A.	Unspecified IP Core
[81]	N.A.	6% - 14%	N.A.	Commercial processors
[3, 19]	11%	10%	N.A.	ARM Cortex M0
[80]	2% to 12%	N.A.	N.A.	Toshiba MeP & Renesas M32R
[88]	**	-49%	5%	32-bit multiplier
[67]	N.A.	-7% to -18%	16%	LEON3 IU
[130, 129]	2%	4.5%	One clock cycle	MIPS
[51, 50]	N.A.	-12% to -38%	Less than 3%	Alpha
[43]	N.A.	-33%	N.A.	Alpha
[136]	5%	-30%	No	DCT
This work	**0.5% to 1.1%**	**-6.5% to -15%**	**No**	**LEON2-based systems**

At last, let's analyse where does this work stand with respect to several comparable publications reviewed in Chapter 3. A comparison criteria that relies on standard metrics was established (performance, power and area) as well as on certain features which are considered important from aspect of this dissertation, e.g., the error types targeted for mitigation, test vehicle and so on.

The comparison data gathered from related works is summarized in Tables 6.8 and 6.9. It is easily noticeable that the work conducted in this dissertation outperforms

the related works in almost every aspect: at cost of negligible area overhead, the proposed approach enables implementation of adaptable processor-based systems able to significantly reduce power consumption and improve SEU resilience while preserving the performance. The solutions presented in the related works achieve favourable results in some segments of the comparison criteria, but none managed to implement an entire system capable to synergistically tackle all of the considered critical metrics.

Chapter 7

Conclusion

Processors play central role in virtually every computing device. As a consequence of the aggressive technology scaling, the contemporary computing systems are facing two major challenges: excessive power consumption and increased susceptibility to faults. These two metrics are not complementary to each other and addressing both at the same time is difficult. Hence, adaptivity is becoming increasingly important feature for the modern computing systems. This dissertation presents a cross-layer framework for reducing power consumption and improving resilience of processor-based systems in a synergistic way. Once embedded in the system, the framework can dynamically switch between low power and error resilient operation modes according to the current needs.

The proposed framework is constituted of three building blocks deployed at circuit, architecture and software layer of the system stack respectively. At the very basis of the framework lies the SWIELD multimodal flip-flop. Depending on the current application requirements and environmental conditions, the SWIELD FF can be configured to operate either as an in situ monitor or as a TMR flip-flop. Of course, operation as a conventional flip-flop is also possible. When a system runs a non-critical task in terms of performance or resilience, the SWIELD FF may be switched to in situ monitor mode. This allows close observation of the system performance and prevention of timing errors. Simultaneously, an adaptive voltage/frequency scaling scheme is realized for power saving, potentially without performance degradation. If protection against radiation-induced faults is required, the SWIELD FF can be put into TMR FF mode. Finally, when high performance is the sole requirement, the system can use the SWIELD FF as a regular flip-flop. Management of the SWIELD FF operation modes is done by the SOMU - the framework component implemented at architecture level. In fact, the SOMU is responsible for orchestrating the entire system operation: it distributes the clock signal to the remaining system components and initiates clock gating as well as voltage/frequency scaling when necessary. At last, the FFL is the only framework component implemented in software. It can be understood as a procedure library that allows easy and intuitive manipulation of the features offered by the framework.

A strategy for integration of the framework in processor-based systems is another contribution of this work. Insertion of SWIELD FFs in a target system is a

crucial phase of the integration process. Namely, the SWIELD FFs are intended to replace certain timing-critical flip-flops in the system. The proposed strategy is able to conveniently determine how many and which critical flip-flops are candidates for replacement. Eventually, the SWIELD FF insertion is performed automatically. By following this strategy, the cross-layer framework was successfully integrated in several instances of both single-core and multicore processor systems.

The approach proposed in this dissertation covers several broad topics and consequently, it demands large effort in terms of investigation and experimenting for proper evaluation. Therefore, three distinct scenarios involving either single- or multicore processor system instances for specific application were created. The considered instances differ in the number of inserted SWIELD FFs. Numerous simulation-based experiments were conducted in order to credibly assess the influence of the framework on the systems' operation with respect to the parameters of interest. The experimental results presented in Section 6.2 showed similar trends in all three scenarios. Depending on the number of SWIELD FFs, the framework has potential to reduce systems power consumption up to 15% in comparison to the baseline designs. Furthermore, while occurrence of timing errors is prevented, a significant improvement of the system resilience with respect to SEUs has also been observed. Concretely, replacing only 13% of the timing-critical flip-flops in the processor cores with SWIELD FFs can typically provide failure-free operation. This is true for most of the performed FI campaigns, including some of those that mimic harsher conditions. However, as failure-free execution cannot be guaranteed, a correlation between the execution time and the fault rate in a form of mathematical model was inferred based on the FI results. Using this model, one can appropriately plan the system mission and prepare for dynamic activation of additional resilience mechanisms, given that the protection provided by the framework is not enough. Finally, it is of key importance to emphasize that the framework can achieve such results without performance impact at negligible area overhead.

From the previous discussion, it can be concluded that the dissertation objectives postulated in Subsection 1.4.3 have been met. To sum up, by utilizing the proposed cross-layer framework, a balanced level between resilience and power efficiency in processor-based systems is indeed achievable. The final section of this thesis discusses the potential approach improvements that can be performed in future.

7.1 Future Work

Along with the features of the cross-layer framework that were already investigated and thoroughly presented in the previous chapters, there are several opportunities stemming from the conducted research which could be explored as part of a follow-up work. The first logical step would be physical design and production of a chip that would implement processor-based system supported by the proposed framework. This would open possibilities to asses the framework contribution towards system power efficiency by chip measurements. In addition, performing irradiation tests on the chip could help evaluate the system resilience to SEUs.

Although the SWIELD FF, as such, already offers several distinct functionalities, further improvement of its architecture and features is still possible. For example, by appropriate incorporation of transient filters, the SWIELD FF could make the system resilient also to SETs. Moreover, if suitable mechanism for identifying permanent faults is to be included in its structure, the SWIELD FF could become the first flip-flop to provide practically total resilience. Of course, inclusion of extra hardware would make the SWIELD FF even more complex and power-hungry. Therefore, another aspect to consider is the possibility to reorganize or redesign the SWIELD FF constituent components [1] in order to reduce its complexity. An expertise in transistor-level design might be required for this purpose. Finally, it should be noted that both SOMU and FFL need to be adequately extended to reflect any new functionalities introduced to the SWIELD FF.

[1] For example, instead of the traditional AND-OR voter, simpler architectures like NAND-only could be used.

List of Abbreviations

AMBA	Advanced Microcontroller Bus Architecture, page 81
AMP	Asymmetric Multicore Processor, page 56
ASIC	Application-Specific Integrated Circuit, page 57
AVFS	Adaptive Voltage/Frequency Scaling, page 50
AVS	Adaptive Voltage Scaling, page 50
BISER	Built-in Soft Error Resilience, page 46
BTI	Bias Temperature Instability, page 34
BTU	Brandenburgische Technische Universität, page ix
CDF	Cumulative Distribution Function, page 23
CLEAR	Cross-Layer Exploration for Architecting Resilience, page 47
CMOS	Complementary Metal Oxide Semiconductor, page 28
CPU	Central Processing Unit, page 7
DAAD	Deutscher Academischer Austauschdienst, page ix
DCT	Discrete Cosine Transform, page 55
DE	Instruction Decode Pipeline Stage, page 107
DSA	Domain Specific Architectures, page 6
DSSM	De-Stress Multiprocessor Sub-Mode, page 84
DSTB	Dual Sampling with Time Borrowing, page 43
DUE	Detected but Uncorrected Error, page 47
DVFS	Dynamic Voltage and Frequency Scaling, page 48
ECC	Error Correction Code, page 56

EMI Electromagnetic Interference, page 34

ESA European Space Agency, page 56

EX Instruction Execute Pipeline Stage, page 107

FE Instruction Fetch Pipeline Stage, page 107

FFL Framework Function Library, page 12

FI Fault Injection, page 23

FIT Failures In Time, page 24

FPGA Field Programmable Gate Array, page 93

FPU Floating-Point Unit, page 81

FS Frequency Scaling, page 73

FTSM Fault-Tolerant Multiprocessor Sub-Mode, page 84

GCR Galactic Cosmic Rays, page 34

GPIO General-Purpose Input/Output, page 82

GPU Graphics Processing Unit, page 7

HCI Hot Carrier Injection, page 34

HDL Hardware Description Language, page 57

HKMG High-K Metal Gate, page 2

HPSM High Performance Multiprocessor Sub-Mode, page 84

IC Integrated Circuit, page 17

IEEE Institute of Electrical and Electronics Engineers, page 41

IHP Institut für Helbleiterphysik, page 85

ILP Instruction Level Parallelism, page 6

ILS Instrument Landing System, page 71

IP Intellectual Property, page 90

IRDS International Roadmap for Devices and Systems, page 7

ISA Instruction Set Architecture, page 81

ISM In Situ Monitor, page 42

ITRS	International Technology Roadmap for Semiconductors, page 7
IU	Integer Unit, page 87
LEAP-DICE	Layout Design through Error-Aware Transistor Positioning-Dual Interlocked Storage Cell, page 47
LP	Linear Programming, page 58
MBU	Multiple Bit Upset, page 36
MCU	Multiple Cell Upset, page 36
MEM	Memory Access Pipeline Stage, page 107
MOSFET	Metal Oxide Semiconductor Field Effect Transistor, page 17
MTBF	Mean Time Between Failures, page 26
MTTF	Mean Time To Failure, page 26
MTTR	Mean Time To Repair, page 26
NGMP	Next Generation Microprocessor, page 56
NMR	N-Modular Redundant, page 43
PDF	Probability Density Function, page 23
PDN	Power Delivery Network, page 37
PEFF	Pre-Error Flip-Flop, page 53
PSN	Power Supply Noise, page 37
PVTA	Process Voltage Temperature Aging, page 3
RHBD	Radiation Hardening By Design, page 57
RISC	Reduced Instruction Set Computer, page 81
RTL	Register Transfer Level, page 57
SAIF	Switching Activity Interchange Format, page 92
SDC	Silent Data Corruption, page 47
SDF	Standard Delay Format, page 95
SEC-DED	Single Error Correction - Double Error Detection, page 82
SEE	Single Event Effects, page 35

SEFI	Single Event Functional Interrupt, page 36
SEL	Single Event Latch-up, page 36
SEM	Soft Error Mitigation, page 43
SER	Soft Error Rate, page 5
SET	Single Event Transient, page 36
SETTOFF	Soft Error and Timing Error Tolerant Flip-Flop, page 43
SEU	Single Event Upset, page 36
SoC	System-On-Chip, page 7
SOMU	System Operation Management Unit, page 12
SPARC	Scalable Processor Architecture, page 81
SPE	Solar Particle Events, page 34
SRAM	Static Random Access Memory, page 8
STA	Static Timing Analysis, page 58
STEM	Soft and Timing Error Mitigation, page 43
TCL	Tool Command Language, page 85
TDP	Thermal Design Power, page 2
TDTB	Transition Detector with Time Borrowing, page 42
TMR	Triple Modular Redundancy, page 36
UART	Universal Asynchronous Receiver/Transmitter, page 83
UAS	Unmanned Aircraft System, page 19
VCD	Value Change Dump, page 92
VHDL	Very High-Speed Integrated Circuit Hardware Description Language, page 62
XFS	Xcelium Fault Simulator, page 93

Bibliography

[1] International Roadmap for Devices and Systems (IRDS) 2020 Edition - IEEE IRDS.

[2] International Technology Roadmap for Semiconductors (ITRS) 2015 Edition - IEEE ITRS.

[3] Hadi Ahmadi Balef, Hamed Fatemi, Kees Goossens, and José Pineda de Gyvez. Effective in-situ chip health monitoring with selective monitor insertion along timing paths. In *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, pages 213–218, 2018.

[4] Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, and Robert I. Davis. An empirical survey-based study into industry practice in real-time systems. In *2020 IEEE Real-Time Systems Symposium (RTSS)*, pages 3–11. IEEE, 2020.

[5] Bharadwaj Amrutur, Nandish Mehta, Satyam Dwivedi, and Ajit Gupte. Adaptative techniques to reduce power in digital circuits. *Journal of Low Power Electronics and Applications*, 1(2):261–276, 2011.

[6] Jan Andersson, Magnus Hjorth, Fredrik Johansson, and Sandi Habinc. LEON processor devices for space missions: First 20 years of LEON in space. In *2017 6th International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, pages 136–141. IEEE, 2017.

[7] Marko Andjelkovic, Junchao Chen, Aleksandar Simevski, Zoran Stamenkovic, Milos Krstic, and Rolf Kraemer. A review of particle detectors for space-borne self-adaptive fault-tolerant systems. In *2020 IEEE East-West Design & Test Symposium (EWDTS)*, pages 1–8. IEEE, 2020.

[8] ARM. AMBA specification rev. 2.0. *http://www.arm.com*, 1999.

[9] Semiconductor Industry Association. Rebooting the IT revolution. In *Rebooting the IT Revolution Workshop, Washignton*, 2015.

[10] Naga Durga Prasad Avirneni and Arun Somani. Low overhead soft error mitigation techniques for high-performance and aggressive designs. *IEEE Transactions on Computers*, 61(4):488–501, 2011.

[11] Naga Durga Prasad Avirneni, Viswanathan Subramanian, and Arun K. Somani. Low overhead soft error mitigation techniques for high-performance and aggressive systems. In *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*, pages 185–194. IEEE, 2009.

[12] Algirdas Avizienis and J-C. Laprie. Dependable computing: From concepts to design diversity. *Proceedings of the IEEE*, 74(5):629–638, 1986.

[13] Algirdas Avizienis, J-C. Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, 1(1):11–33, 2004.

[14] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, et al. *Fundamental concepts of dependability*. University of Newcastle upon Tyne, Computing Science, 2001.

[15] John E. Ayers. *Digital integrated circuits: analysis and design*. CRC Press, 2018.

[16] Bilal M. Ayyub and Richard H. McCuen. *Probability, statistics, and reliability for engineers and scientists*. CRC press, 2016.

[17] Marta Bagatin and Simone Gerardin. *Ionizing radiation effects in electronics: from memories to imagers*. CRC press, 2019.

[18] Farshad Baharvand and S. Ghassem Miremadi. ARMOR: Adaptive reliability management by on-the-fly redundancy in multicore embedded processors. In *2015 IEEE 21st Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 215–224. IEEE, 2015.

[19] Hadi Ahmadi Balef, Kees Goossens, and José Pineda de Gyvez. Chip health tracking using dynamic in-situ delay monitoring. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 304–307. IEEE, 2019.

[20] Janet L. Barth, C.S. Dyer, and E.G. Stassinopoulos. Space, atmospheric, and terrestrial radiation environments. *IEEE Transactions on nuclear science*, 50(3):466–482, 2003.

[21] Robert C. Baumann. Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Transactions on Device and materials reliability*, 5(3):305–316, 2005.

[22] Mikel Anton Bezdek. Utilizing timing error detection and recovery to dynamically improve superscalar processor performance. 2006.

[23] Pritam Bhattacharjee, Prerna Rana, and Alak Majumder. Understanding of on-chip power supply noise: Suppression methodologies and challenges. In *Recent Trends in Communication Networks*. IntechOpen, 2019.

[24] B. Bhuva. Soft error trends in advanced silicon technology nodes. In *2018 IEEE International Electron Devices Meeting (IEDM)*, pages 34–4. IEEE, 2018.

[25] Mark T. Bohr and Ian A. Young. CMOS scaling trends and beyond. *IEEE Micro*, 37(6):20–29, 2017.

[26] Shekhar Borkar. Design challenges of technology scaling. *IEEE micro*, 19(4):23–29, 1999.

[27] Shekhar Borkar. Low power design challenges for the decade (invited talk). In *Proceedings of the 2001 Asia and South Pacific Design Automation Conference*, pages 293–296, 2001.

[28] Keith A. Bowman, James W. Tschanz, Nam Sung Kim, Janice C. Lee, Chris B. Wilkerson, Shih-Lien L. Lu, Tanay Karnik, and Vivek K. De. Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance. *IEEE Journal of Solid-State Circuits*, 44(1):49–63, 2008.

[29] Keith A. Bowman, James W. Tschanz, Shih-Lien L. Lu, Paolo A. Aseron, Muhammad M. Khellah, Arijit Raychowdhury, Bibiche M. Geuskens, Carlos Tokunaga, Chris B. Wilkerson, Tanay Karnik, et al. A 45 nm resilient microprocessor core for dynamic variation tolerance. *IEEE Journal of Solid-State Circuits*, 46(1):194–208, 2010.

[30] Thomas D. Burd, Trevor A. Pering, Anthony J. Stratakos, and Robert W. Brodersen. A dynamic voltage scaled microprocessor system. *IEEE Journal of solid-state circuits*, 35(11):1571–1580, 2000.

[31] Nicholas P. Carter, Helia Naeimi, and Donald S. Gardner. Design techniques for cross-layer resilience. In *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pages 1023–1028. IEEE, 2010.

[32] V. Chandra. Dependable design in nanoscale CMOS technologies: challenges and solutions. In *Workshop on Dependable and Secure Nanocomputing*, 2009.

[33] Vikas Chandra and Robert Aitken. Impact of technology and voltage scaling on the soft error susceptibility in nanoscale CMOS. In *2008 IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems*, pages 114–122. IEEE, 2008.

[34] Tze-Chiang Chen. Overcoming research challenges for CMOS scaling: Industry directions. In *2006 8th International Conference on Solid-State and Integrated Circuit Technology Proceedings*, pages 4–7. IEEE, 2006.

[35] Eric Cheng, Shahrzad Mirkhani, Lukasz G. Szafaryn, Chen-Yong Cher, Hyungmin Cho, Kevin Skadron, Mircea R. Stan, Klas Lilja, Jacob A. Abraham, Pradip Bose, et al. CLEAR: Cross-layer exploration for architecting resilience: Combining hardware and software techniques to tolerate soft errors in processor cores.

In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2016.

[36] Eric Cheng, Shahrzad Mirkhani, Lukasz G. Szafaryn, Chen-Yong Cher, Hyungmin Cho, Kevin Skadron, Mircea R. Stan, Klas Lilja, Jacob A. Abraham, Pradip Bose, et al. Tolerating soft errors in processor cores using clear (cross-layer exploration for architecting resilience). *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(9):1839–1852, 2017.

[37] Mihir Choudhury, Vikas Chandra, Kartik Mohanram, and Robert Aitken. TIMBER: Time borrowing and error relaying for online timing error resilience. In *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pages 1554–1559. IEEE, 2010.

[38] Mihir R. Choudhury and Kartik Mohanram. Masking timing errors on speedpaths in logic circuits. In *2009 Design, Automation & Test in Europe Conference & Exhibition*, pages 87–92. IEEE, 2009.

[39] Cristian Constantinescu. Impact of deep submicron technology on dependability of VLSI circuits. In *Proceedings International Conference on Dependable Systems and Networks*, pages 205–209. IEEE, 2002.

[40] Francesco Conti, Daniele Palossi, Andrea Marongiu, Davide Rossi, and Luca Benini. Enabling the heterogeneous accelerator model on ultra-low power microcontroller platforms. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1201–1206. IEEE, 2016.

[41] John Daly, Bill Harrod, Thuc Hoang, Lucy Nowell, Bob Adolf, Shekhar Borkar, Nathan DeBardeleben, Mootaz Elnozahy, Mike Heroux, David Rogers, et al. Inter-agency workshop on hpc resilience at extreme scale. *National Security Agency Advanced Computing Systems*, 2012.

[42] Anup Das, Akash Kumar, Bharadwaj Veeravalli, Cristiana Bolchini, and Antonio Miele. Combined DVFS and mapping exploration for lifetime and soft-error susceptibility improvement in MPSoCs. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2014.

[43] Shidhartha Das, Carlos Tokunaga, Sanjay Pant, Wei-Hsiang Ma, Sudherssen Kalaiselvan, Kevin Lai, David M. Bull, and David T. Blaauw. RazorII: In situ error detection and correction for PVT and SER tolerance. *IEEE Journal of Solid-State Circuits*, 44(1):32–48, 2008.

[44] André DeHon, Heather M. Quinn, and Nicholas P. Carter. Vision for cross-layer optimization to address the dual challenges of energy and reliability. In *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pages 1017–1022. IEEE, 2010.

[45] Robert H. Dennard, Fritz H. Gaensslen, Hwa-Nien Yu, V. Leo Rideout, Ernest Bassous, and Andre R. LeBlanc. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974.

[46] Yuvraj Singh Dhillon, Abdulkadir Utku Diril, and Abhijit Chatterjee. Soft-error tolerance analysis and optimization of nanometer circuits. In *Design, Automation, and Test in Europe*, pages 389–400. Springer, 2008.

[47] Elena Dubrova. *Fault-tolerant design*. Springer, 2013.

[48] Alireza Ejlali, Marcus T. Schmitz, Bashir M. Al-Hashimi, Seyed Ghassem Miremadi, and Paul Rosinger. Energy efficient SEU tolerance in DVS-enabled real-time systems through information redundancy. In *Proceedings of the 2005 international symposium on Low power electronics and design*, pages 281–286, 2005.

[49] Mohamed Elgebaly and Manoj Sachdev. Efficient adaptive voltage scaling system through on-chip critical path emulation. In *Proceedings of the 2004 international symposium on Low power electronics and design*, pages 375–380, 2004.

[50] Dan Ernst, Shidhartha Das, Seokwoo Lee, David Blaauw, Todd Austin, Trevor Mudge, Nam Sung Kim, and Krisztián Flautner. Razor: circuit-level correction of timing errors for low-power operation. *IEEE Micro*, 24(6):10–20, 2004.

[51] Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev Rao, Toan Pham, Conrad Ziesler, David Blaauw, Todd Austin, Krisztian Flautner, et al. Razor: A low-power pipeline based on circuit-level timing speculation. In *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, pages 7–18. IEEE, 2003.

[52] Andreas Floros, Yiorgos Tsiatouhas, and Xrysovalantis Kavousianos. Timing error detection and correction by time dilation. In *IFIP/IEEE International Conference on Very Large Scale Integration-System on a Chip*, pages 271–285. Springer, 2008.

[53] David Flynn, Rob Aitken, Alan Gibbons, and Kaijian Shi. *Low power methodology manual: for System-On-Chip design*. Springer Science & Business Media, 2007.

[54] Jiri Gaisler. A portable and fault-tolerant microprocessor based on the SPARC v8 architecture. In *Proceedings International Conference on Dependable Systems and Networks*, pages 409–415. IEEE, 2002.

[55] GaislerResearch. LEON2 processor user's manual. *Version 1.0.30, XST Edition, July*, 2005.

[56] Jean-Claude Geffroy and Gilles Motet. *Design of dependable computing systems*. Springer Science & Business Media, 2013.

[57] Massoud M. Ghahroodi, Emre Ozer, and David Bull. SEU and SET-tolerant ARM Cortex-R4 CPU for space and avionics applications. In *Proc. of the Workshop on Manufacturable and Dependable Multi-core Architectures at Nanoscale*, 2013.

[58] Meeta S. Gupta, Jarod L. Oatley, Russ Joseph, Gu-Yeon Wei, and David M. Brooks. Understanding voltage variations in chip multiprocessors using a distributed power-delivery network. In *2007 Design, Automation & Test in Europe Conference & Exhibition*, pages 1–6. IEEE, 2007.

[59] D.L. Hansen, K. Jobe, J. Whittington, M. Shoga, and D.A. Sunderland. Correlation of prediction to on-orbit SEU performance for a commercial 0.25-um CMOS SRAM. *IEEE Transactions on Nuclear Science*, 54(6):2525–2533, 2007.

[60] Nor Zaidi Haron and Said Hamdioui. Why is CMOS scaling coming to an end? In *2008 3rd International Design and Test Workshop*, pages 98–103. IEEE, 2008.

[61] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Elsevier, 2011.

[62] Magnus Hijorth, Martin Aberg, Nils-Johan Wessman, Jan Andersson, Remy Chevallier, Russel Forsyth, Rolad Weigand, and Luca Fossati. GR740: Radhard quad-core LEON4FT system-on-chip. *DASIA 2015-DAta Systems in Aerospace*, 732:7, 2015.

[63] Mike Hinchey and Lorcan Coyle. Evolving critical systems. In *ECBS*, page 4. Citeseer, 2010.

[64] Eugene R. Hnatek. *Practical reliability of electronic equipment and products*, volume 116. CRC press, 2002.

[65] Mark Horowitz, Elad Alon, Dinesh Patil, Samuel Naffziger, Rajesh Kumar, and Kerry Bernstein. Scaling, power, and the future of CMOS. In *IEEE InternationalElectron Devices Meeting, 2005. IEDM Technical Digest.*, pages 7–pp. IEEE, 2005.

[66] Mu-Yue Hsiao. A class of optimal minimum odd-weight-column SEC-DED codes. *IBM Journal of Research and Development*, 14(4):395–401, 1970.

[67] Che-Min Huang, Tsung-Te Liu, and Tzi-Dar Chiueh. An energy-efficient resilient flip-flop circuit with built-in timing-error detection and correction. In *VLSI Design, Automation and Test (VLSI-DAT)*, pages 1–4. IEEE, 2015.

[68] Pengcheng Huang, Pratyush Kumar, Georgia Giannopoulou, and Lothar Thiele. Energy efficient DVFS scheduling for mixed-criticality systems. In *2014 International Conference on Embedded Software (EMSOFT)*, pages 1–10. IEEE, 2014.

[69] Vincent Huard, F. Cacho, F. Giner, M. Saliva, A. Benhassain, Dinesh Patel, N. Torres, S. Naudet, Abhishek Jain, and C. Parthasarathy. Adaptive wearout management with in-situ aging monitors. In *2014 IEEE International Reliability Physics Symposium*, pages 6B–4. IEEE, 2014.

[70] Cadence Design Systems. Inc. Xcelium fault simulator user guide. *San Jose, CA*, 2019.

[71] Cadence Design Systems. Inc. Xcelium XRUN user guide. *San Jose, CA*, 2019.

[72] SPARC International Inc. and David L. Weaver. *The SPARC architecture manual*. Prentice-Hall Englewood Cliffs, 1994.

[73] Intel. Speedstep® technology for the Intel® pentium® M processor. *White Paper, Intel*, 2004.

[74] Xabier Iturbe, Balaji Venu, Emre Ozer, and Shidhartha Das. A triple core lock-step (TCLS) ARM® Cortex®-R5 processor for safety-critical and ultra-reliable applications. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*, pages 246–249. IEEE, 2016.

[75] Lee Hsiao-Heng Kelin, Lilja Klas, Bounasser Mounaim, Relangi Prasanthi, Ivan R. Linscott, Umran S. Inan, and Mitra Subhasish. LEAP: Layout design through error-aware transistor positioning for soft-error resilient sequential cell design. In *2010 IEEE International Reliability Physics Symposium*, pages 203–212. IEEE, 2010.

[76] Nam Sung Kim, Todd Austin, David Baauw, Trevor Mudge, Krisztián Flautner, Jie S Hu, Mary Jane Irwin, Mahmut Kandemir, and Vijaykrishnan Narayanan. Leakage current: Moore's law meets static power. *computer*, 36(12):68–75, 2003.

[77] Taeyoung Kim, Zeyu Sun, Hai-Bao Chen, Hai Wang, and Sheldon X.-D. Tan. Energy and lifetime optimizations for dark silicon manycore microprocessor considering both hard and soft errors. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(9):2561–2574, 2017.

[78] Maha Kooli and Giorgio Di Natale. A survey on simulation-based fault injection tools for complex systems. In *2014 9th IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 1–6. IEEE, 2014.

[79] Israel Koren and C. Mani Krishna. *Fault-tolerant systems*. Morgan Kaufmann, 2020.

[80] Yuji Kunitake, Toshinori Sato, Hiroto Yasuura, and Takanori Hayashida. A selective replacement method for timing-error-predicting flip-flops. *Journal of Circuits, Systems, and Computers*, 21(06):1240013, 2012.

[81] Liangzhen Lai, Vikas Chandra, Robert C. Aitken, and Puneet Gupta. Slackprobe: A flexible and efficient in situ timing slack monitoring methodology. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(8):1168–1179, 2014.

[82] Jean-Claude Laprie. From dependability to resilience. In *38th IEEE/IFIP Int. Conf. On dependable systems and networks*, pages G8–G9. Citeseer, 2008.

[83] Patrik Larsson. Power supply noise in future IC's: a crystal ball reading. In *Proceedings of the IEEE 1999 Custom Integrated Circuits Conference (Cat. No. 99CH36327)*, pages 467–474. IEEE, 1999.

[84] Etienne Le Sueur and Gernot Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 international conference on Power aware computing and systems*, pages 1–8, 2010.

[85] Tuo Li, Jude Angelo Ambrose, Roshan Ragel, and Sri Parameswaran. Processor design for soft errors: Challenges and state of the art. *ACM Computing Surveys (CSUR)*, 49(3):1–44, 2016.

[86] Yang Lin and Mark Zwolinski. SETTOFF: A fault tolerant flip-flop for building cost-efficient reliable systems. In *2012 IEEE 18th International On-Line Testing Symposium (IOLTS)*, pages 7–12. IEEE, 2012.

[87] Yang Lin, Mark Zwolinski, and Basel Halak. A low-cost, radiation-hardened method for pipeline protection in microprocessors. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(5):1688–1701, 2015.

[88] Sebastian Moreno Londoño and José Pineda de Gyvez. A better-than-worst-case circuit design methodology using timing-error speculation and frequency adaptation. In *2012 IEEE International SOC Conference*, pages 15–20. IEEE, 2012.

[89] Intel Pentium M. Processor with 2-MB L2 cache and 533-MHz front side bus, 2005.

[90] Elie Maricau and Georges Gielen. *Analog IC reliability in nanometer CMOS*. Springer Science & Business Media, 2013.

[91] Subhasish Mitra, Kevin Brelsford, Young Moon Kim, Hsiao-Heng Kelin Lee, and Yanjing Li. Robust system design to overcome CMOS reliability challenges. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 1(1):30–41, 2011.

[92] Subhasish Mitra, Kevin Brelsford, and Pia N. Sanda. Cross-layer resilience challenges: Metrics and optimization. In *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pages 1029–1034. IEEE, 2010.

[93] Subhasish Mitra, Ming Zhang, Norbert Seifert, TM Mak, and Kee Sup Kim. Soft error resilient system design through error correction. In *VLSI-SoC: Research Trends in VLSI and Systems on Chip*, pages 143–156. Springer, 2008.

[94] Gordon E. Moore et al. Cramming more components onto integrated circuits, 1965.

[95] Gordon E. Moore et al. Progress in digital integrated electronics. In *Electron devices meeting*, volume 21, pages 11–13. Maryland, USA, 1975.

[96] Fahad Bin Muslim, Affaq Qamar, and Luciano Lavagno. Low power methodology for an ASIC design flow based on high-level synthesis. In *2015 23rd International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 11–15. IEEE, 2015.

[97] Michael Nicolaidis. *Soft errors in modern electronic systems*, volume 41. Springer Science & Business Media, 2010.

[98] Daniele Palossi. *On the Autonomous Navigation of Nano-UAVs*. PhD thesis, ETH Zurich, 2019.

[99] Goran Panić. *A methodology for designing low power sensor node hardware systems*. PhD thesis, BTU Cottbus-Senftenberg, 2015.

[100] David A. Patterson and John L. Hennessy. *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*. Elsevier Science & Technology Books, 2017.

[101] Vladimir Petrovic and Milos Krstic. Design flow for radhard TMR flip-flops. In *2015 IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits & Systems*, pages 203–208. IEEE, 2015.

[102] Khem C. Pokhrel. Physical and silicon measures of low power clock gating success: An apple to apple case study. *Synopsys Users Group (SNUG)*, 2007.

[103] Ilia Polian and John P. Hayes. Selective hardening: Toward cost-effective error tolerance. *IEEE Design & Test of Computers*, 28(3):54–63, 2010.

[104] Baldev Raj, Marcel Van de Voorde, and Yashwant Mahajan. *Nanotechnology for Energy Sustainability, 3 Volume Set*. John Wiley & Sons, 2017.

[105] Sheldon M. Ross. *Introduction to probability and statistics for engineers and scientists*. Academic Press, 2020.

[106] O. Ruano, J.A. Maestro, and P. Reviriego. A methodology for automatic insertion of selective TMR in digital circuits affected by SEUs. *IEEE Transactions on Nuclear Science*, 56(4):2091–2102, 2009.

[107] Karl Rupp, M. Horovitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten. Years of microprocessor trend data. *by karlrupp. net.[Online*, 42.

[108] Somayeh Sadeghi-Kohan, Arash Vafaei, and Zainalabedin Navabi. Near-optimal node selection procedure for aging monitor placement. In *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, pages 6–11. IEEE, 2018.

[109] Mehdi Sadi, L. Winemberg, and M. Tehranipoor. A robust digital sensor IP and sensor insertion flow for in-situ path timing slack monitoring in SoCs. In *2015 IEEE 33rd VLSI Test Symposium (VTS)*, pages 1–6. IEEE, 2015.

[110] Siva Satyendra Sahoo, Bharadwaj Veeravalli, and Akash Kumar. Cross-layer fault-tolerant design of real-time systems. In *2016 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 63–68. IEEE, 2016.

[111] Mohammad Salehi, Alireza Ejlali, and Muhammad Shafique. Run-time adaptive power-aware reliability management for manycores. *IEEE Design & Test*, 35(5):36–44, 2017.

[112] Toshonori Sato, Takahito Yoshiki, and Takanori Hayashida. Multicore power management utilizing error-predicting flip-flop. In *2011 International Conference on Complex, Intelligent, and Software Intensive Systems*, pages 606–611. IEEE, 2011.

[113] Oliver Schrape, Anselm Breitenreiter, Steffen Zeidler, and Milos Krstic. Aspects on timing modeling of radiation-hardness by design standard cell-based TMR flip-flops. In *2019 22nd Euromicro Conference on Digital System Design (DSD)*, pages 639–642. IEEE, 2019.

[114] Rishad A. Shafik, Bashir M. Al-Hashimi, and Krishnendu Chakrabarty. Soft error-aware design optimization of low power and time-constrained embedded systems. In *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pages 1462–1467. IEEE, 2010.

[115] Muhammad Shafique and Siddharth Garg. Computing in the dark silicon era: Current trends and research challenges. *IEEE Design & Test*, 34(2):8–23, 2016.

[116] Weiwei Shan, Longxing Shi, and Jun Yang. In-situ timing monitor-based adaptive voltage scaling system for wide-voltage-range applications. *IEEE Access*, 5:15831–15838, 2017.

[117] Aleksandar Simevski. *Architectural framework for dynamically adaptable multi-processors regarding aging, fault tolerance, performance and power consumption.* PhD thesis, BTU Cottbus-Senftenberg, 2014.

[118] Aleksandar Simevski, Rolf Kraemer, and Milos Krstic. Low-complexity integrated circuit aging monitor. In *14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pages 121–125. IEEE, 2011.

[119] Aleksandar Simevski, Rolf Kraemer, and Milos Krstic. Investigating core-level n-modular redundancy in multiprocessors. In *2014 IEEE 8th International Symposium on Embedded Multicore/Manycore SoCs*, pages 175–180. IEEE, 2014.

[120] Aleksandar Simevski, Oliver Schrape, and Carlos Benito. Comparative analyses of low-power IC design techniques based on chip measurements. In *2018 16th Biennial Baltic Electronics Conference (BEC)*, pages 1–6. IEEE, 2018.

[121] Aleksandar Simevski, Oliver Schrape, Carlos Benito, Milos Krstic, and Marko Andjelkovic. PISA: Power-robust multiprocessor design for space applications. In *2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–6. IEEE, 2020.

[122] Sudarshan Srinivasan, Israel Koren, and Sandip Kundu. Online mechanism for reliability and power-efficiency management of a dynamically reconfigurable core. In *2015 33rd IEEE International Conference on Computer Design (ICCD)*, pages 327–334. IEEE, 2015.

[123] Ashish Srivastava, Dennis Sylvester, and David Blaauw. *Statistical analysis and optimization for VLSI: timing and power.* Springer Science & Business Media, 2006.

[124] Synopsys. Primepower reference manual. *Mountain View, CA*, 2020.

[125] Pavan Vithal Torvi, V.R. Devanathan, and V. Kamakoti. Framework for selective flip-flop replacement for soft error mitigation. In *2015 28th International Conference on VLSI Design*, pages 381–386. IEEE, 2015.

[126] Emmanuel Touloupis, James A. Flint, Vassilios A. Chouliaras, and David D. Ward. Study of the effects of SEU-induced faults on a pipeline protected microprocessor. *IEEE Transactions on Computers*, 56(12):1585–1596, 2007.

[127] Gaurang R. Upasani. Soft error mitigation techniques for future chip multiprocessors. 2016.

[128] Stefanos Valadimas and Angela Arapoyanni. Timing error tolerance in pipeline based core designs. In *Proceedings of the 18th Panhellenic Conference on Informatics*, pages 1–6, 2014.

[129] Stefanos Valadimas, Angela Arapoyanni, and Yiorgos Tsiatouhas. Timing error mitigation in microprocessor cores. In *2016 IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 772–775. IEEE, 2016.

[130] Stefanos Valadimas, Yiorgos Tsiatouhas, and Angela Arapoyanni. Timing error tolerance in small core designs for SoC applications. *IEEE Transactions on Computers*, 65(2):654–663, 2015.

[131] J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components, 1956.

[132] Alice Wang and Samuel Naffziger. *Adaptive techniques for dynamic processor optimization: theory and practice*. Springer Science & Business Media, 2008.

[133] Hongxia Wang, Samuel V. Rodriguez, Cagdas Dirik, and Bruce Jacob. Electromagnetic interference and digital circuits: An initial study of clock networks. *Electromagnetics*, 26(1):73–86, 2006.

[134] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced CPU energy. In *Mobile Computing*, pages 449–471. Springer, 1994.

[135] Neil H.E. Weste and David Harris. *CMOS VLSI design: a circuits and systems perspective*. Pearson Education India, 2015.

[136] Martin Wirnshofer. *Variation-aware adaptive voltage scaling for digital CMOS circuits.* Springer, 2013.

[137] Martin Wirnshofer, Nasim Pour Aryan, Leonhard Heiss, Doris Schmitt-Landsiedel, and Georg Georgakos. On-line supply voltage scaling based on in situ delay monitoring to adapt for PVTA variations. *Journal of Circuits, Systems and Computers*, 21(08):1240027, 2012.

[138] Martin Wirnshofer, Leonhard Heiß, Georg Georgakos, and Doris Schmitt-Landsiedel. A variation-aware adaptive voltage scaling technique based on in-situ delay monitoring. In *14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pages 261–266. IEEE, 2011.

[139] Jun Yao, Shogo Okada, Masaki Masuda, Kazutoshi Kobayashi, and Yasuhiko Nakashima. DARA: A low-cost reliable architecture based on unhardened devices and its case study of radiation stress test. *IEEE Transactions on Nuclear Science*, 59(6):2852–2858, 2012.

[140] Sina Yari-Karin, Ali Sahraee, Javad Saber-Latibari, Mohsen Ansari, Nezam Rohbani, and Alireza Ejlali. A comparative study of joint power and reliability management techniques in multicore embedded systems. In *2020 CSI/CPSSI International Symposium on Real-Time and Embedded Systems and Technologies (RTEST)*, pages 1–8. IEEE, 2020.

[141] Dakai Zhu, Rami Melhem, and Daniel Mossé. The effects of energy management on reliability in real-time embedded systems. In *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.*, pages 35–40. IEEE, 2004.

[142] Dakai Zhu, Muhammad Shafique, Man Lin, and Sudeep Pasricha. Guest editorial: Special issue on low-power dependable computing. *IEEE Transactions on Sustainable Computing*, 3(3):137–138, 2018.

[143] Haissam Ziade, Rafic A. Ayoubi, Raoul Velazco, et al. A survey on fault injection techniques. *Int. Arab J. Inf. Technol.*, 1(2):171–186, 2004.