Doctoral Dissertation

Temporal and Spatial Redundancy Optimization for Efficient Fault-Tolerant LSIs

Junghoon Oh

Supervisor: Professor Mineo Kaneko

School of Information Science
Japan Advanced Institute of Science and Technology

March, 2018

Abstract

Due to the downsizing of VLSIs, several reliability issues have become more explicit. Among the issues, soft-error induced degradation is one of the dominant contributors to faults on modern VLSIs. Soft-error is a transient fault which is triggered by cosmic ray induced neutron and alpha rays from radioactive contaminants in IC package materials. Soft-error lasts only short time but it can affect several spatial points simultaneously.

Approaches to dealing with soft-errors are roughly divided into the following three groups: i) approaches on the device level such as selecting of IC packing materials and improving of well structures; ii) approaches on the circuit level such as a flip-flop with additional circuits for error detection, error recovery and error avoidance; iii) approaches on the system level which includes multiple module redundancy such as concurrent error detection and triple modular redundancy. Most semiconductor designs rely on computer aided design systems and implementation of reliability on semiconductor devices on higher design level is important at the viewpoint of optimization. Nevertheless, because there is no dominant approach with a single level for soft-error tolerance and such a single level scheme imposes high overheads in terms of power, performance and chip area, a higher level approach should be assisted by approaches on other levels. Thus, it is assumed that several approaches are implemented across distinct levels in this research although the author focuses on the system level approach via high-level synthesis. As a result of high-level synthesis, fault-tolerance is implemented to datapath circuits in register-transfer level (RTL).

This dissertation proposes a method to synthesize application specific soft-error tolerant datapaths via high-level synthesis. To guarantee reliability and high real-time property, the proposed method is based on the triple redundancy of computation algorithms, which are to be realized as datapath circuits. The method reduces time overhead originated in redundancy, and at the same time, it realizes datapaths that keep multiple component soft-error tolerance. In order to mitigate time overhead, error detection parts with comparison and error correction parts with retry share resources speculatively (speculative resource sharing). Operations in the retry parts are not executed as long as no error detected. During this period, resources bound to those operations in the retry parts are in an idle state. If it is assumed that the probability of the recurrence of soft-errors in a short period is low enough, operations which are executed as retry and other operations which are executed simultaneously can share resources.

To use hardware resources more efficiently, a tripled data flow graph of a computation algorithm is partitioned into several connected subgraphs. However, the more comparison-operations are selected, the more subgraphs are partitioned. As a result, latency of the datapath improves because fine-grained subgraphs are relatively easier to fulfill the speculative resource sharing condition than coarse-grained ones. On the other hand, latency may become larger as increasing the number of comparison-operations. Thus, deciding insertions of comparison-operations is an important factor in design optimization.

In order to reduce an excessively applied fault-tolerance and mitigate time overhead for the executions of retry parts which are a disadvantage of comparison-retry (C-R)

schemes, the author introduces spatial/temporal adjacency constraint between datapath components considering a concept of localities of soft-errors. If a single soft-error has a spatial and temporal boundary, and its influence is limited against multiple component errors, several components can be executed at the same time. Majority-voting (M-V) schemes have a disadvantage that third copies should be always executed, while third copies in C-R schemes need not be executed as long as no error has occurred. Moreover, M-V mechanisms require more hardware resources although datapaths to which those schemes are implemented can achieve small latency. Because of introducing adjacency constraint, an M-V mechanism for error masking and correction instead of a C-R mechanism, and the three copies in every subgraphs can be executed concurrently. On the other hand, an advantage of C-R based error correction mechanisms is that the mechanism can reduce power consumption contributed by idling of retry parts, in case no error is detected. Nevertheless, the executions of retry parts cause time overhead, which is a drawback of the C-R mechanisms. In order to merge the advantages of both C-R schemes and M-V schemes, the author proposes a combination of the two error correction schemes to take advantage of the strengths of each scheme. In addition, a heuristic algorithm to find a latency-optimized integration of the two schemes is suggested.

It is found that the proposed method can reduce latency in several different applications without deterioration of reliability and chip area compared with a conventional C-R scheme. In addition, the experimental results show that the proposed method is more effective when a computation algorithm possesses higher parallelism and a small number of resources is available.

Key words: fault-tolerance, soft-error, multiple component error, high-level synthesis, datapath synthesis, datapath optimization, triple algorithm redundancy, concurrent error detection, comparison-retry scheme, majority-voting scheme, speculative resource sharing, adjacency constraint, mixed error correction scheme, check variable selection, integer linear programming.

Acknowledgments

First of all, I would like to express my sincere gratitude to my advisor Professor Mineo Kaneko of Japan Advanced Institute of Science and Technology for his constant encouragement and kind guidance during this work. Looking back on my school days in JAIST, everyday I have gradually grown as a Ph.D. candidate with his advice, suggestions and comments. Although it is a coincidence that I decided to enroll at JAIST and belong to Kaneko laboratory, I would not be able to acquire my degree without his support. To be honest, I had several difficult times during my Ph.D. degree, however, he believed in me and waited for me so that I have successfully finished this work.

I also wishes to thank my vice advisor Associate Professor Kiyofumi Tanaka for giving advice in different points of view in different situations such as Hokuriku hardware joint seminars, book reading seminars. Because of his advice, I have always realized my own inadequacy.

I greatly appreciate Professor Yasushi Inoguchi for his advice which make my work has been improved as a doctoral dissertation.

I would also like to thank Assistant Renyuan Zhang for his comments when he used to belong to JAIST.

I gratefully acknowledge giving cheerful comments and sharing experience provided by Assistant Professor Sungmoon Jeong as one who got a Ph.D. degree earlier than me.

I am also grateful to my minor research advisor Professor Baris Taskin at Drexel University in Philadelphia for giving me a wonderful chance in the U.S. and stimulus not only to my minor research but also to my life.

I am thankful to all members in Kaneko laboratory and Tanaka laboratory.

I would also like to thank all my friends in JAIST and all over the world who have cheered me up and helped me to keep going.

Finally, I would like to thank my family members for giving me unlimited love and waiting for me for long time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Needs for Dependable Systems

The use of computer systems including electronic devices has been increased and a lot of portions of our daily lives have relied on them [3][4]. In these days, plenty of electronic devices perform life-critical tasks in different fields such as aviation control systems, vehicle control systems, missile guidance and control systems, industrial plants, and medical appliances [3][4]. In these systems, failure of the systems may bring loss of human life. Faults on financial fields such as banking and stock markets can cause huge financial loss or loss of opportunity, as well [3]. Manufacturing industries also require dependability because failure in computer systems for large-scale production causes loss of goods and profits [5]. As a recent tragic accident of a self-driving car under an artificial intelligence control reminded us about the importance of reliability in life-critical systems, it is clear that highly reliable systems are needed in such critical applications and the needs for those systems will continue to increase.

1.2 Fault Tolerance

1.2.1 Fault, Error and Failure

A fault is a physical defect, imperfection, or flaw that occurs in a hardware or software [1][6]. An error is a deviation from a desired or intended state, which occurs as a result of a fault. Errors are usually associated with incorrect values in the system state [6][7]. A failure is a nonperformance of some action which is due or expected [1][6]. We can say that a system has a failure if it does not act according to the system specification. Although faults are causes of errors and errors are causes of failures, neither every fault causes an error nor every error causes a failure [1][6]. For instance, neutron particles from outer space at terrestrial level can hit memories in computer systems. However, not every neutron particle makes the value of a memory cell incorrect. Although a cell is upset by a fault, the value can be corrected by a mechanism such as error correction code (ECC).

1.2.2 Fault-Aware Design Techniques

There are three primary techniques to maintain normal performance of a system: fault avoidance, fault masking and fault tolerance [1].

Fault avoidance is used to prevent faults from the beginning such as design reviews, component screening, testing and other quality control methods [1]. Although an appropriate design review can eliminate many faults caused by specification mistakes, and a test can detect and remove many faults before the system is placed into operation, those techniques never remove all the faults in the system. Thus, other techniques should be implemented to increase the reliability of the system [3][1].

Fault tolerance is the ability that a system continues operating properly when faults occurred [1]. In other words, a system to which fault tolerance is implemented normally operates without failures although faults occurred. Fault tolerance can be realized by several techniques; fault masking, fault detection, fault location, fault containment and fault recovery [1][6]. i) Fault masking is any process that prevents faults in a system from introducing errors into the informational structure of that system [1]. A well known example of fault masking is majority voting. ii) Fault detection is the process of recognizing that a fault has occurred within a system [1][6]. Fault detection is often required before any recovery procedure is implemented [1]. Comparison is a technique to detect faults in the systems with duplicated components and the output results of two components are compared [6]. If the results disagree, it indicates an occurrence of a fault [6]. iii) Fault location is the process of determining where a fault has occurred so that an appropriate recovery can be implemented [1]. However, when a disagreement occurs during the comparison of two components, it is impossible to determine which of the two has failed [6]. iv) Fault containment is the process of isolating a fault and preventing the effects of that fault from propagating throughout a system [1]. This is typically achieved by frequent fault detection, by multiple request/confirmation protocols and by performing consistency checks among components [6]. v) Fault recovery is the process of remaining operational or regaining operational status when a faulty component has been identified [1]. Fault recovery can be achieved by replacement of the faulty component into a redundant backup component [1][6].

Fault removal is a set of techniques for making the number of faults in the system smaller [6]. Fault removal is performed during the development phase as well as during the operation phase of a system [6]. During the development phase, fault removal involves verification, diagnosis and correction steps [6]. Fault removal during the operation phase consists of corrective and preventive maintenance [6].

Fault forecasting is a set of techniques to estimate the number of faults at present and in the future [6]. A system behavior evaluation which concerns fault occurrence or activation can perform fault forecasting. [6].

1.2.3 Fault Classification

The complexity of electronic devices has been developing with higher transistor density and new generations of semiconductor device technology have archived miniaturization of transistors [4]. Under such circumstances, computer systems have relied on the dependability of semiconductor devices on their systems.

In general, faults on a hardware are classified with respect to fault duration into permanent, transient, and intermittent faults [4][6]. A permanent fault remains active as long

as an appropriate countermeasure is not taken. Once a component in a permanent fault fails, it never works correctly again [3][4][6]. It is usually caused by physical defects in the hardware, such as shorts in a circuit, broken interconnections, or stuck cells in a memory [6]. A transient fault remains active only for a short period of time [3][6]. Due to their short duration, transient faults are often detected through the errors which result from their propagation [6]. The causes of transient faults are mostly environmental ones, such as alpha particles, atmosphere neutrons, electrostatic discharge, electrical power drops, or overheating [6]. A transient fault which becomes active periodically or repeatedly is an intermittent fault [3][6]. An intermittent fault never goes away entirely; it swings between being inactive and active [4]. When the fault is inactive, the component functions normally; when the fault is active, the component malfunctions [4]. Intermittent faults occur due to unstable or marginal hardware [8]. Those faults can be activated by environmental changes such as higher temperature or voltage, and they usually precede the occurrence of permanent faults [8].

Soft-Errors

Due to the shrinking of feature sizes of semiconductor devices and reduced noise margins, nanoscale circuits have become increasingly more susceptible to interferences [9]. According to [6][9][10][11], transient faults among the above different types of faults are the dominant contributors to faults on modern LSIs. Among several degradation caused by transient faults, soft-error induced degradation is a major one and it has become a growing concern [9][12]. Main sources of soft-error for electronic devices are cosmic radiation or cosmic rays, which are coming from space and strike the earth [13]. Radiation induced soft-errors occur within operational lifetime of a semiconductor device when a single ionizing radiation event affects a transistor as a burst of hole-electron pairs, which is large enough to upset the state of the transistor [12].

1.2.4 Fault Tolerance Classification

Redundancy is the addition of information, resources or time beyond system requirements for normal operation [1]. In that sense, redundancy is a preparation of functional capabilities that will not be needed in a fault-free environment [6][14]. The redundancy in a system can take one of several forms; hardware, information and time redundancy [3][1]. Hardware redundancy is the addition of extra hardware for supporting fault tolerance [3][1]. Information redundancy is the addition of extra information which is appended to the original information to implement fault tolerance [1]. Time redundancy uses additional time to perform tasks for fault tolerance [3][1].

Hardware Redundancy

Hardware redundancy is achieved by providing two or more physical copies of a hardware component [6]. Due to replication, employment of hardware redundancy returns many penalties: increase in weight, size, power consumption and cost to provide multiple copies, and time to design, fabricate and test [6]. There are mainly three forms of hardware redundancy: passive, active and hybrid [1][6]. Passive techniques employ fault masking to conceal faults and prevent them from resulting in errors [1]. The active approach achieves fault tolerance with fault detection and fault removal, which removes the faulty

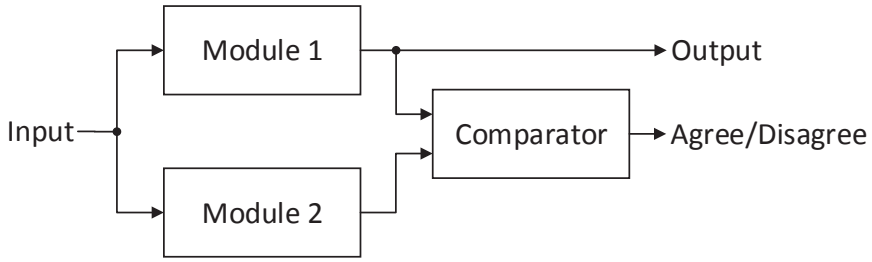

Figure 1.1: An example of duplication with comparison (DWC) [1]

hardware from the system [1]. Hybrid techniques merge the passive and active approaches to take advantages of both [1].

An M-of-N system consists of N modules and the system is required that at least M modules operate properly among N modules [4]. Thus, the system fails when fewer than M modules are functional. Triple modular redundancy (TMR) is a well-known example of this system and consists of triplicate hardware modules and voting elements (voter). In TMR, all three modules operate in parallel [3]. If at least 2 outputs from 3 modules are correct, a voter performs a majority vote of 3 outputs to mask the failure of one module, and as a result, the system performs functionally [3]. Thus, TMR scheme relies on the voter [3].

Active hardware redundancy techniques employ fault detection, fault location and fault recovery to achieve fault tolerance [1]. The basic form of active redundancy is duplication with comparison (DWC) [6]. Figure 1.1 shows the simplest DWC. In DWC, two identical modules operate in parallel and a comparison element (comparator) checks equality between the two outputs of the two modules [6]. If the results differ, an error signal is generated [6]. DWC scheme can only detect the occurrence of a fault [6]. Standby redundancy is another technique for active hardware redundancy [6][15]. If one of the n modules is active, then the remaining $n-1$ modules serve as spares [6]. A switch monitors the active module and replaces operation with a spare if a fault-detection (FD) unit detects an error [6]. The pair-and-a-spare technique combines DWC and standby redundancy [1][6]. This combined approach is similar to standby redundancy but two active modules (instead of one) work in parallel [6]. Also, similar to DWC, the results are compared to detect disagreement [6]. If the comparator generates an error signal, the switch analyzes the signal from the FD blocks and decides which module is in failure [6]. Then, the faulty module is removed from operation and switched to a spare [6].

Hybrid redundancy combines advantages of passive and active redundancy [6]. Fault masking is used to prevent the hybrid system from transiently erroneous outputs [6]. Fault detection, location and recovery are used to reconfigure a system if a fault occurs [6]. Hybrid redundancy techniques are usually implemented in safety-critical applications [6]. Although several approaches to hybrid redundancy exist, most of them are based on N-modular redundancy (NMR) with spares [1]. The basic idea of NMR with spares is to provide a basic core of N modules arranged in a voting [1]. Spares are also prepared to replace faulty modules in the NMR core [1].

Information Redundancy

Information redundancy is commonly realized by coding [4], which has traditionally been one of the most important techniques to support fault tolerance in hardware and has been used in a lot of systems [3]. The basic idea is to add extra check bits to the original data such that faults in some bits can be detected and corrected. The process of adding check bits to the original data is called encoding and the reverse process of extracting information from the encoded data is called decoding [3].

Parity codes are the oldest family of codes [6]. A parity-coded word contains d data bits and an extra check bit for the parity [4]. In case of an even parity code, the extra bit is set so that the total number of 1s in the whole $(d + 1)$-bit word is even [4]. Since the parity code is separable, parity encoding and decoding circuits can simply be implemented [4]. A parity code can only detect errors, since it is impossible to determine which bit is erroneous [6].

A unidirectional error detecting code, which is separable and has a lower overhead, is Berger code [4]. All the affected bits in unidirectional errors change in the same direction, either from 0 to 1 or from 1 to 0 but not in both directions [4]. To encode Burger code, count how many number of 1s exist in the word, express the counted number in binary representation, complement it, and append the complement number to the original data [4].

The most common extension of the parity approach is Hamming error-correcting code [1][16]. A number of memory designs employ this code for several reasons [1]: i) Hamming error correction is relatively inexpensive; Hamming codes typically need 10% to 40% of overhead due to redundancy [1]. ii) Hamming codes are efficient in terms of the time required to perform the correction process; the encoding and the decoding processes take relatively smaller time [1].

Time Redundancy

The fundamental problem of hardware and information redundancy is the penalty caused by extra hardware to implement the redundancy techniques mentioned above [1]. On the other hand, time redundancy deals with the cost of extra hardware by using additional time because the expense of time is much less important than that of hardware in a lot of situations [1].

The basic concept of time redundancy is the repetition of the same computations two or more times to detect errors [1]. If an error occurred, the computations can be performed again to check whether the error disappears or remains [1]. Such techniques are often suitable to detect transient faults, however, they are inappropriate in case of permanent faults. [1]. An assumption of time redundancy is that a fault affects two repeated computations in different ways [1]. Time redundancy can also provide error correction when the same computations are repeated three or more times, and performed a majority vote on the three [1].

1.3 Fault-Tolerant Design of Application-Specific IC

Application-specific integrated circuits (ASICs) implement customer-specified functions and there are various possibilities for the associated customization [17]. Computer aided

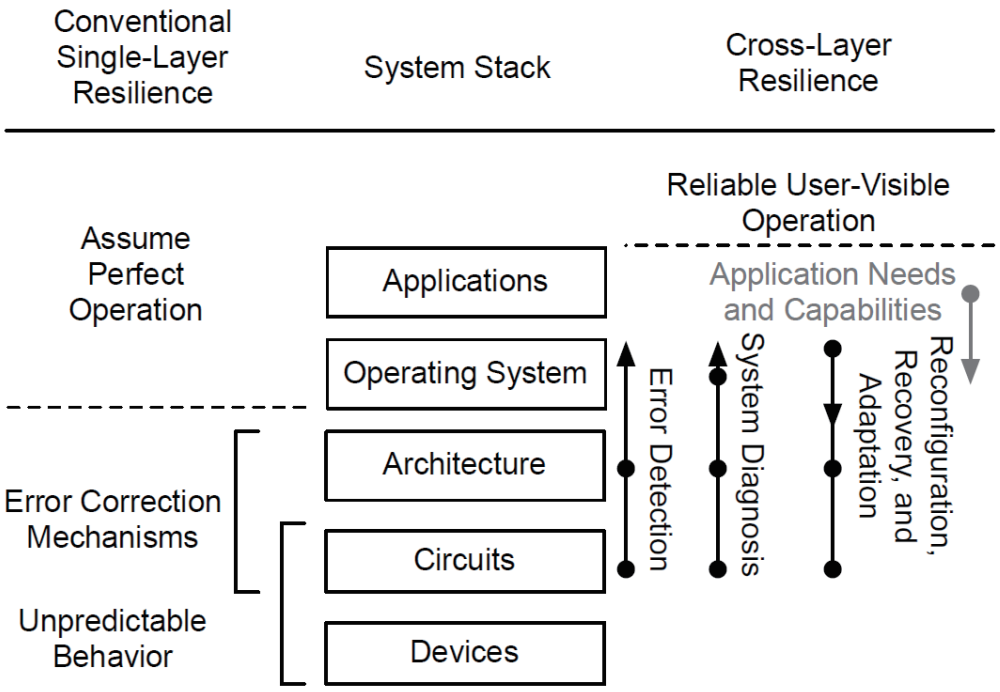

Figure 1.2: System Stack and Resilience [2]

design (CAD) tools are essential since growing design complexity and shrinking product life cycles require to develop an efficient design infrastructure which is based on a custom IC design platform [17]. However, chip design and evaluation in a low level such as the logic-gate level suffer prohibitively expensive performance penalties when applied to modern large LSI systems. In addition, the post logic-synthesis phase are too late in the design cycle to improve performance such as speed, size and reliability. Therefore, an ASIC design in more abstract and effective level (such as the register-transfer level (RTL) or even higher level) than the logic-gate level is highly desirable [18].

The current hardware fault-tolerant designs focus on mechanisms for resilience in the architecture and/or circuit levels of the system stack (Fig. 1.2) [2]. Single level approaches to fault-tolerance simplify the design of the upper levels in the system stack. However, fault-tolerant design which relies on a single level technique has its limitations and disadvantages; to guarantee high reliability, single level schemes typically replicate hardware components or computations [2]. For instance, as a circuit level approach, entire ordinary flip-flops without fault-tolerant design may be replaced with soft-error resilience flip-flops [19]–[21] (to be introduced at Chap. 2.3) to achieve fault-tolerance in the chip design with the circuit level error correction scheme. However, the above specialized flip-flops require high implementation costs in terms of power, performance and/or chip area [2][22]. In order to mitigate the above overheads induced by a single level approach, several resilience techniques can be implemented across various levels of the system stack to achieve reliability requirements [22].

High-level synthesis (HLS) is the translation process from a behavioral description to a structural description [23]. HLS is also known as behavioral-level synthesis or algorithmic-level synthesis [23][24]. A behavioral description at the algorithm level defines a precise procedure for the computational solution of a problem [24]. The constraints to be considered in high-level synthesis are chip area, performance, power consumption, reliability,

testability and cost [23]. This abstraction level synthesis allows a hardware design engineer to make decisions at an early stage of the design cycle and generates rapidly optimized hardwares in terms of the above constraints [23][25]. As HLS has matured, it has dealt with not only traditional optimization in terms of performance and power, but also fault resiliency awareness [26]. Such progression of HLS research allows us to explore the design space with several considerations at the abstraction level via HLS to obtain resiliency optimal register-transfer level (RTL) datapath circuits [26].

1.4 Research Problems and Contributions

The main purpose of this study is to synthesize single soft-error tolerant application specific datapaths via high-level synthesis. The author focuses on the following two problems: (i) tolerability against multiple component soft-errors and (ii) mitigation of hardware/time overhead due to redundancy.

1.4.1 Tolerability against Multiple Component Soft-Errors

Most studies about soft-error tolerance concern only a single fault on a single functional unit. As mentioned in Sect. 1.2.3, multiple component errors such as MCUs and MBUs are becoming one of serious problems in LSI design. To address this problem, this dissertation proposes a method to synthesize single soft-error, which affects multiple components at the same time, tolerant application-specific datapaths via high-level synthesis. The proposed method is based on triple algorithm redundancy. In the proposed design, considering the tolerability for multiple-component errors by a single soft-error, two copies are used for error detection, and the third copy is used as a retry. The detail will be stated in Chap. 3.

1.4.2 Mitigation of Hardware/Time Overhead

Due to the use of redundancy, datapaths synthesized with conventional methods suffer from hardware and time overhead. To address this problem, speculative resource sharing [27], which is an effective resource management, between two distinct copies, and adjacency constraint between components [28] are introduced. Based on the above two proposals, the author also proposes a hybrid approach of comparison-retry and majority-voting error correction schemes [29] to combine high resource efficiency and small latency which are the advantages of the two schemes. In addition, in order to achieve minimum solutions in latency, an optimized combination algorithm of the two fault-tolerant mechanisms is proposed.

1.5 Dissertation Outline

This paper is organized as follows: Chapter 2 introduces conventional studies related to this research; Chapter 3 explains the basic concepts on which the proposed method is based; Chapter 4 proposes speculative resource sharing for hardware/time overhead mitigation; Chapter 5 suggests a latency-aware check variable selection algorithm which enlarges opportunities for speculative resource sharing; Chapter 6 introduces adjacency

constraint between datapath components to mitigate time overhead further and to reduce excessively applied fault-tolerance; Chapter 7 proposes mixed error correction mechanism and its latency optimization algorithm; and Chap. 8 concludes this article.

Chapter 2

Related Works

In accordance with the scale of the region of interest (ROI), there are three general approaches to dealing with soft-errors: (i) device level, (ii) circuit level and (iii) system level.

2.1 Types of Soft-Errors

Soft-errors can sometimes describe as single event effects (SEEs) in semiconductor devices. Although SEEs caused by alpha rays and terrestrial neutrons on the ground were pointed out in the late 1970s, major researches and countermeasures have focused almost only on alpha ray-induced single event upsets (SEUs) in memory devices until the 1990s [30]. Terrestrial neutrons have been recognized as a significant source of SEUs since the early 1990s, and concerns are now growing wider and deeper with the miniaturization of the device size [30].

Table 2.1 shows different types of error modes and their possible requirements [30]. Although soft-errors rarely occur and their effects last only short periods, they can affect several spatial points simultaneously [31]–[33]. While single-bit upsets are mainly a reliability concern in memory devices and systems, multi-bit upsets (MBUs) and multi-cell upsets (MCUs) also have become a serious problem, recently [32]. According to the study by Autran et al. [34], MCUs account for 80% of the bit upsets which occurred, and the multiplicity of a MCU reaches 17 cells in 40nm SRAMs. Work by Quinn et al. [35] reports that triple modular redundancy (TMR) can be defeated by even 2-bit multi-cell upsets caused by a single soft-error. According to the study [36], a single soft-error can simultaneously affect adjacent nodes which lie on separate logic paths, and as a result, it can defeat redundant techniques which rely on majority voting.

2.2 Device Level Approaches

Purification and elimination of contamination of the IC packaging materials reduce soft-errors induced by alpha particles from the materials [10]. Adjustments to process and technology can also mitigate the occurrence of soft-errors. Changing substrate structures or doping profiles can reduce charge collection. DRAM designers have actually used multiple-well isolation [37]. Researchers have also suggested well-based mitigation technologies for CMOS logics [38] and have shown that SOI substrates reduce soft-error

Table 2.1: Error modes from terrestrial neutrons and requirements

Category	Error mode	Explanation	Possible requirement
Soft-error	SEU	single event upset	1000 FIT†/chip
			200-400 FIT/Mbit or higher
	MCU	multi cell upset	a few to 10% fo total SEUs or higher
	SET	single event transient	to be determined
	MBU	multi bit upset	0-10 FIT/chip
Pseudo-hard error	SEFI	single event functional interrupt	0-1 FIT/chip or system
	SEL	single event latchup	0-1 FIT/chip or system
Hard error	SEB	single event burnout	0-1 FIT/chip or system
	SEGR	single event gate rupture	

†1FIT = 1 error in 10^9 hours

sensitivity [10]. A use of additional capacitances increases critical charge of SRAMs and logic devices, and as a result, it can increase the device's robustness [37]. Zhang et al. [39] analyzed that an increment of cell distance and well-contact density suppresses the occurrence of SEUs as well as MCUs. However, device level approaches provide limited improvements for soft error mitigation although the expense of additional process complexity, yield loss, and substrate cost are needed for most process solutions [37].

2.2.1 Transistor Structure Aware Techniques

Baze et al. [38] developed a logic cell design technique which uses actively biased and isolated wells for SEU hardening. This technique can be implemented for clocks, asynchronous control logic and dynamic logic. Their experiment results show a 3 times improvement of transient upset immunity in an SOI process, however, it has 5 times decrease in speed, 3 times increase in area and 5 times increase in power consumption.

Black et al. [40] introduced guard contacts to mitigate the charge collection and to restore the well potential more quickly especially in PMOS devices. Mitigation of the shared charge collection in NMOS devices is accomplished through isolation of the P-wells using a triple-well solution. The filp-flop design from the physical structures with preset and clear increased 30% in maximum in size.

Nakauchi et al. [41] investigated the effect of back bias on neutron-induced multi-cell upset (MCU). They found that MCUs were strongly related to the memory cell array layout and concluded that most MCUs were induced by activation of parastic lateral npn-bipolar transistors. They also found that MCU could be drastically reduced by supplying back bias in the p-wells without any modification of error checking and correction (ECC).

Atkinson et al. [42] presented a layout technique that mitigates SETs in combinational logic. This technique works by introducing one extra drain diffusion into a cell output buffer transistor to intentionally promote charge sharing between transistors and quench the voltage pulse on the output. The mitigation using this technique will improve with technology scaling since the technique exploits charge sharing. These results will allow designers to reduce cumulative SET vulnerability of a cell library with an area penalty of $10 - 40\%$.

Zhang and Kobayashi [43] analyzed contributions of charge sharing and bipolar effects, which are two main factors when SEUs and MCUs occur in two redundant latches, by changing the position of well contacts and the well structure. Device simulation results revealed that charge sharing and bipolar effects are effectively suppressed when the well contacts are placed in the middle of the two latches.

The study [44] suggested a technique with well-silts that mitigates MBUs in multiple bit latches (MBLs). The area overhead is only 5.4% in 28nm technology as a MBL macro for processors. Neutron irradiation tests clarified that the proposed technique did not change SBU rate and consequently did not degrade processor performance.

Narasimham et al. [45] fabricated dual- and triple-well bulk CMOS SRAMs at the 28nm node and tested using alpha particles and heavy-ions over a range of supply voltages. The test results showed that triple-well designs are better suited to limiting the extent of MCU cross sections at reduced voltages, while dual-well SRAMs have better MCU cross sections and spread for nominal voltage. They claimed that commercial designs targeted for very low voltage and subthreshold voltage operation can benefit from the triple-well option because low voltage operation is important for low power applications and for improving battery life during device standby mode.

2.2.2 Layout-Based Techniques

Bit interleaving is commonly used to minimize the error rate contribution of multi-bit errors [46]. It refers to a memory layout architecture in which physically adjacent bits belong to different logic words. The result is that from an error detection and correction standpoint, two adjacent failing bits appear as two single bit errors rather than as a double bit error in the logic word. Bit interleaving rules are often defined as the minimum physical distance separating two bits belonging to the same logic word.

Baeg et al. [47] proposed a selection method of the width in an interleaving architecture in memory designs to mitigate MCU errors. They found that a use of both the interleaving technique and single-bit error correction (SEC) codes can maximize MCU mitigation.

Furuta et al. [48] measured neutron-induced MCUs on FFs in a 65nm CMOS process in order to evaluate their dependencies on the distance of FFs and well-contact density. Neutron-accelerated test results showed that the MCU-SEU ratio was up to 23.4% and was exponentially decreased by the distance between latches. Also, measurement results showed that well-contact array insertion between flip-flops can reduce MCU rates. In case of 10k-bit filp-flops, well-contact array insertion reduced the number of MCUs from 110 to 1. They can improve soft-error resilience without degradation of delay overhead by inserting well contacts between latches on radiation hardened flip-flops.

Zhang et al. [39] showed that charge sharing and bipolar effect are two main factors when MCUs occur in redundant latches. MCU is prevented when the distance between the redundant latches is increased. Total collected charge of L0 and L1 decreases by 50% by placing the well contacts adjacent to the redundant latches. MCU-SEU ratio decreases to 0.073% in this layout structure. According to the neutron experiment and device simulation results, the ratio of MCU-SEU exponentially decreased by increasing the distance of latches. Experimental results also showed that MCU rates reduced by well-contact array insertion under supply and ground rails.

2.3 Circuit Level Approaches

The most effective method of dealing with soft-errors in memory components is to use additional circuitry for error detection and correction [37]. Mainly, there are two approaches in circuit level; information redundancy based approach and specialized filp-flop based one. Circuit level approaches basically have limitations that those approaches mainly focus on sequential logic parts in a circuit.

2.3.1 Information Redundancy Based Techniques

Information redundancy based error detection consists of adding a single bit to store the parity (odd or even) of each data word, regardless of word length [37]. Before data is retrieved, the parity is computed from the stored data and compared to its parity bit [37]. Under a single fault model, the computed parity and the parity bit do not match in case of error. The parity system enables soft-error detection with a minimal cost in terms of circuit complexity and memory width (each word increases by only a single bit) [37]. There are two disadvantages of this system: i) the detected error cannot be corrected, and ii) the check won't reveal a double error because the parity will match [37].

Error detection and correction (EDAC) or error correcting codes (ECC) address the above disadvantages [37]. Error correction is achieved by adding extra bits to each data vector and encoding the data so that the information distance between any two possible data vectors is at least three [37]. The addition of more bits for parity and extra circuitry may produce larger information distance [37]. Since the information distance in those systems is 3, every single error, which is a change of ± 1 in information space, is corrected [37]. Those systems can detect double-bit errors, however, cannot correct them. Although EDAC and ECC significantly reduce failure rates, they bring a higher cost in terms of design complexity, additional memory required, and inherent latency introduced during access, parity check, and correction [37].

Naseer and Draper presented a double-error correcting (DEC) ECC implementation technique to mitigate MBU in SRAMs [49]. The experimental results showed that this DEC scheme reduces errors by 98.5% compared to only 44% reduction by conventional single-error-correcting-double-error-detecting (SEC-DED) ECC.

2.3.2 Techniques with Soft-Error Hardened Filp-Flops

Specialized flip-flops with additional circuits have been proposed to address soft-errors on circuit level. Calin et al. [50] proposed a storage element design technique called dual interlocked storage cell (DICE) which are insentive to radiation-induced SEUs. A cell for DICE uses a 4-node redundant structure which employs two conventional cross-coupled inverter latch structures that are connected to bidirectional feedback inverters. The four nodes store the data as two pairs of complementary values which are simultaneously accessed using transmission gates for write or read operation. A DICE cell consists of 10 transistors for a simple latch configuration and 12 transistors for a memory cell architecture. If two simultaneously sensitive nodes of the cell which store the same logic stage could be flipped due to the effect of a single particle impact, the immunity is lost and the cell is upset.

Naseer and Draper [51] proposed latch and filp-flop designs which are immune not

only to transient on every signal, such as clock, data, preset and clear, but also to upsets within the storage cell. The proposed design will incur an area overhead of less than 13.5% For typical ASIC macro-architectures. The speed performance of the design is directly propositional to the pulse width of the single event transients targeted. However, if a soft-error occurs at two nodes in the storage cell simultaneously, the SEU immunity of the cell will not be effective any more. Also, if the pulse width of a SET is larger than the delay in the filter, it can result in a SEU at the storage node during the setup and hold time window. Therefore, an effective estimation of the transient pulse width in a particular environment is necessary before finalizing the design decisions.

Uemura et al. [19] developed a soft-error immune latch called SEILA under dual-clock-buffers (DCB), which protects from corrupting storage data by SET on local clock, and double-height-cell (DHC), which protects from SEU on multi-nodes. According to their experimental results, SEILA can protect 99.3% SEU on multi-nodes and almost SET on local clock.

Lin and Zwolinski [20] suggested a fault tolerant flip-flop called SETTOFF in which SETs, SEUs and timing errors are detected and recovered by a time redundancy-based architecture. SETTOFF consumes 35.8% more power than a conventional flip-flop for a 10% activity rate and 48 extra transistors are added to the main flip-flop for SETTOFF.

Masuda et al. [21] have proposed a low-power redundant flip-flop named BCDMR-ACFF which operates over 1GHz clock with high reliability and low power. BCDMR-ACFF has 3.16 times area overhead but 1.16 times power overhead compared with a flip-flop based on transmission gates in case of 10% data activity.

Guo et al. [52] suggested a radiation hardened memory (RHM) cell with 12 transistors which utilizes SEU physics mechanism and layout-topology to provide fault tolerance. The RHM cell has two advantages: i) the ability to tolerate an SEU in any single sensitive node, ii) the ability to fully tolerate an MNU regardless of the stored value of memory cell, iii) and SEU tolerance capability even under PVT variation impacts, and iv) a low-power consumption. The results showed that its power consumption is only 27.7 nW, and its layout area is 4.1 μm^2, which slightly increases about 7.7% compared with DICE. However, due to the stacked structure and PMOS access transistors, the proposed cell has an overhead in access time.

2.3.3 Other Techniques

Zarandi et al. [53] proposed a new protected CLB and FPGA architecture which utilize error detection and correction codes to correct SEUs in LUTs of the FPGA. The fault detection and correction are realized by online or offline fast detection and correction cycles. Error detections and corrections of k-input LUTs are performed with a latency of 2^k clock cycles without any required reconfiguration and significant increament of area overhead. The results of power and area analysis showed that the proposed methods impose less area and power overhead compared to the traditional schemes such as duplication with comparison and TMR circuit design in FPGAs.

2.4 System Level Approaches

Traditionally, redundancy is considered as one of common system level approaches such as concurrent error detection (CED) and triple modular redundancy (TMR). As discussed

in Chap. 1.2.4, hardware redundancy generally brings several penalties [6]. To determine the best way to implement redundancy into a system, a lot of combinations should be investigated [6]. For instance, replacement of redundancy of a component to a higher level one may reduce weight. Also, reduction in target dependability may cut down cost [6].

A lot of self-checking methods for LSIs have been developed as system level approaches in the 1980's and the 1990's [54][55][56][57]. However, those works are based on primitive techniques such as simple duplication and comparison without consideration of overheads due to multiplication. Some of them assumed that faults occur under limited conditions such as unidirectional fault properties (i.e., all bit flips appear in the same direction; from 0 to 1, or from 1 to 0).

2.4.1 Algorithm-level Methodologies

Algorithm-level methodologies for soft-error tolerance using a higher level abstraction, such as behavioral synthesis, have been introduced [58][59]. Behavioral synthesis based techniques have an advantage in circuit design that can explore cost trade-offs among datapath components in earlier stages.

The study [58] proposed an algorithm-level recomputation. The proposed method is two algorithm level recomputing CED schemes using allocation and data diversity in a register transfer level (RTL) implementation. In order to synthesize the proposed transient fault-tolerant datapaths in more abstract level, they used an intermediate representation of a computation called data flow graph (DFG). RT level diversity can be achieved by changing the operation-to-operator allocation and by shifting the operands before recomputation. By enabling a fault to affect the normal result and the recomputed result in two different ways, RTL diversity yields a good CED capability with low area overhead.

Similarly, Lakshminarayana et al. [59] proposed a behavioral synthesis framework called ALPS for the construction of fault secure and area-efficient RTL circuits. They also introduced the concept of aliasing probability analysis to enhance resource sharing among operations in the duplicated control data flow graph (CDFG). The experimental results showed that ALPS can provide fault security with area overheads as low as 25.5% over a circuit synthesized without a requirement of fault security.

Concurrent Error Detection Based Schemes

Orailoglu and Karri [60] introduced a system called SYNCERE to synthesize self-recovering datapaths which detects temporary faults using redundancy based CED, while recovery from transitory faults is accomplished via checkpointing and rollback. In the proposed model, the original computations are duplicated and checkpoints are inserted to preserve th back up the system during execution. Partial results from two copies are compared at a checkpoint (duplication and comparison). Also, a single fault model at the system level is assumed. If the results of the two copies agree, the results are written into the checkpoint registers (checkpointing). On the other hand, in case of disagreement, the computation rolls back to the previous checkpoint and retries.

Antola et al. [61] proposed a datapath design of semi-concurrent self-checking devices via high-level synthesis. It is acceptable to perform checking operations not concurrently with each process iteration, but periodically since the semi-concurrent checking approach assumes low fault occurrence rates. In addition, it is assumed that at most a single fault,

such as a fault of either a function unit, a register or an interconnection, is present in the system. After the reference architecture has been identified and constructed by using any scheduling and allocation algorithm, the nominal datapath is extended to include self-checking features. The proposed approach provides that the required checking periodicity is satisfied while minimizing additional functional units by means of maximum reuse of the resources available for the nominal computation as long as error detection ability is preserved.

Based on the study [60], Wu and Karri [62] proposed a RT-level hybrid time and hardware redundancy based CED technique that uses partitioned data dependence of an input computation algorithm. A single fault induced by SEU is assumed so that a system with fault-tolerant capability is able to correct this kind of faults. Aiming to further reduce the hardware overhead associated with fault security, some of data dependencies are broken and rearranged. According to the experimental results, the proposed CED design technique can ensure the fault security of a design without involving too much overhead.

In the study by Liu and Wu [63], fault-duration and location-aware CED technique considering power efficiency and fault security is suggested. Similar to other high-level synthesis based approaches, it is assumed that the behavioral specification has been complied into a DFG, and algorithm level recomputation is employed. Their study focuses on detecting faults which occur in datapaths during runtime, such as the faults caused by SEU and SEL, and minimizing power consumption in the generated datapath for any fault scenario.

Sengupta and Bhadauria [64] suggested an exploration process of an optimal fault tolerant datapath under user specified power and delay budget via high-level synthesis. The synthesized circuits possess capabilities of masking errors through single and multi cycle transient faults. Based on algorithm-level duplication and fault tolerance, an original computation unit and its duplication are scheduled so that a fault in the original unit never propagates to its duplication unit and vise versa. Moreover, the output of the original is separately stored two times with a certain interval and compared to detect errors in the original. In order to mask a single fault in an unit, the two outputs of the original and the output of the duplication are voted. The experimental results showed that the proposed method achieved 29% reduction in hardware compared with the study [65], which is based on TMR.

Triple Modular Redundancy Based Schemes

On the other hand, triple modular redundancy (TMR) was one of the earliest methods [66] suggested to obtain a reliable system from less reliable components [67]. TMR is a technique which is firstly proposed by Von Neumann [66] and is commonly used to provide design hardening [68]. The primary shortcoming of N modular redundancy is excessive area overhead [69].

Gaitanis [67] proposed a totally self-checking triple modular redundancy (TSC-TMR) system consists of a conventional TMR system monitored by a TSC circuit with two outputs indicating information errors and internal faults. The internal fault indication is independent of the output information errors and indicates masked errors of modular units or faults in the monitoring circuit itself. The information error indication depends on the output information errors and it can be used as a stop signal preventing the propagation of

erroneous outputs. An algebraic technique performing logical operations to detect errors has been employed for the TSC multiple error checking circuit.

D'Angelo et al. [70] suggested a technique for hardware diagnosis of permanent and transient faults possibly affecting a FPGA-based TMR system implementation. More specifically, the proposed scheme allows to identify whether a detected error is due to a transient or permanent fault affecting a replicated module, or the used voter, or the proposed scheme itself. The availability of such a diagnosis method can be exploited to active a suitable recovery technique for the identified fault.

Samudrala et al. [69] proposed a design technique to harden combinational circuits mapped onto Xilinx Vertex FPGAs against SEUs. The signal probabilities of the lines can be used to detect SEU sensitive subcircuits of a given combinational circuit. The circuit can be hardened against SEUs by selectively applying TMR (STMR) to these sensitive subcircuits. However, there is an increase in the number of the voter circuits required for the STMR circuits. It is claimed that the proposed method can greatly reduce the area overhead of the hardened circuit with a small loss of SEU immunity when compared to the state-of-the-art TMR.

The study [71] proposed an approach allowing the exploration of the design space in a FPGA-based reliable system. The proposed method is based on TMR passive hardware redundancy technique, coupled with partial dynamic reconfiguration to recover from the occurrence of soft errors, affecting either the FPGA configuration memory or its temporary memory elements, used for the application computation.

Ruano et al. [72] suggested a methodology to insert selective TMR for SEU mitigation. The benefit is that this methodology results in a lower circuit complexity than the traditional TMR approach, while meeting the specified reliability level. Besides, the proposed method has been improved in performance, adding an enhancement based on an innovative topological analysis of the target circuit.

Azambuja et al. [73] proposed a method that allows the use of dynamic partial reconfiguration combined with TMR in SRAM-based FPGAs. The proposed method combines large grain TMR with special voters capable of signalizing the faulty module and checkpoint states that allow the sequential synchronization of the recovered module with Xilinx TMR (XTMR) approach. As a result, only the faulty domain is reconfigured with minimization of time and energy, which are spent in the process. In addition, the use of checkpoint states avoids system downtime, since the synchronization of the recovered module is performed while the others are kept running. According to the experimental results, the method has reduced fault recover time compared to the standard TMR, while maintaining the compatible area overhead and performance. Also, the dynamic partial reconfiguration process can be up to 45 times faster than the traditional approach.

Iwagaki et al. [65] proposed high-level synthesis for long duration transient fault tolerant datapaths based on TMR. Based on algorithm level triplication under a single fault model, the proposed algorithm performs forced-directed scheduling so that the synthesized circuits possess k_d-cycle error detection and k_c-cycle error correction capabilities. The authors claimed that the proposed approach is a reasonable alternative to a conventional TMR based method since long duration transient fault tolerance is implemented in the proposed one, and area and latency are improved.

Sengupta and Kachave proposed two methodologies that a fault tolerant HLS methodology for simultaneously providing multiple cycle and multiple unit transient fault tolerance, and a HLS methodology for low cost design solution through exploration of fault

tolerant hardware configuration and loop unrolling factor [74]. Results of the proposed approach on standard benchmarks achieved 27% design cost reduction in average and 61% power consumption reduction in average, when compared with the study [65].

CED and TMR Hybrid Schemes

The study by Krishnamohan and Mahapatra [75] presented an approach to cope with SETs in combinational and sequential logic circuits. They combined an error masking technique in non-critical paths and an error detection technique with recovery in critical paths. The proposed method also employs techniques to improve slack in circuits such as: (i) exploiting circuit delay dependence on input vectors and (ii) redistributing slack in pipeline circuits based on soft-error rate (SER) contribution of individual paths.

Stralen and Pimentel [76] proposed an approach towards early design space exploration of fault-tolerant multimedia multi-processor system-on-chip (MPSoC) called SAFE, which stands for sesame automated fault-tolerance explorer. SAFE provides a simulation-based evaluation to explore several different fault-tolerant implementations, such as time redundancy, space redundancy or hybrid of the two, in an early design stage. As a result, SAFE can produce metrics for dynamically scheduled applications for a wide range of fault-tolerant patterns.

Bolchini and Miele [77] suggested a design methodology that enhances the classical HW/SW codesign flow for hard real-time embedded systems to introduce reliability-awareness in an early design phase. In addition to the flow, the proposed method employs an extra layer for reliability-awareness having additional inputs, such as fault management requirements to be handled and fault management mechanisms to be implemented. The target hardware is a system-on-chip which consists of a given set of heterogeneous processing nodes. It is assumed that the core of each node can be either a general purpose processor (GPP) or a full-custom ASIC/FPGA module. The proposed method performs system level replication with CED for ASIC/FPGA modules and an application level task replication for GPPs. Similar to other system level approaches, this design paradigm supports a design space exploration in an early stage to identify the most promising solution in terms of overall performance.

Zhang et al. [78] presented a GUARD (guaranteed reliability in dynamically reconfigurable systems) method which allows for autonomous runtime reliability management in reconfigurable architectures. The proposed system dynamically determines which hardware between a hardened processor and a less reliable reconfigurable hardware should be used for a computation. As a result, the system guarantees a target reliability while optimizing the performance. According to their experimental results, GUARD system achieved up to 68.3% improvement in performance compared to statical fault tolerance optimization techniques.

Ito suggested a full TMR design with respect to energy minimization [79]. The study assumes that only one operational unit is affected by a single soft-error at one time and other circuit elements are fault-free. Based on algorithm level triplication, two types of TMRs are employed: the spatial TMR (S-TMR) and the temporal TMR (T-TMR). In S-TMR, three function units (FUs) are used at the same time to execute a triplicate computation, and then three majority voters are executed to mask an error. On the other hand, in T-TMR, two FUs are executed at first, then two results are compared for an equality check. After that, according to the equality, the third FU is executed.

If the comparison result confirms the equality of the two results, the third FU will not be executed and the power consumption of the third FU is saved. Furthermore, each component has two types in terms of supply voltage: high and low supply voltages. If a high supply voltage component is replaced instead of a low voltage one, the power consumption will also be reduced. In order to minimize area overhead which originates from S-TMR and maximize the energy efficiency of the circuit, an energy consumption minimization problem to select ones among TMR modes and supply voltages respectively corresponding to each operation is formulated as a mixed integer programming (MIP) model and solved by a MIP solver.

2.4.2 Non-Algorithm-Level Methodologies

There are several non-algorithm-level soft-error tolerance approaches which consider fault-tolerance architectures instead of implementation of algorithmic multiplication. Razor which is proposed by Ernst et al. [80] relies on a combination of system and circuit level techniques. In their proposal, Razor flip-flop for error detection which compares two pipeline stage values (one with a fast clock and the other with a delayed clock), and a pipeline mechanism for error recovery are implemented. In case of error, the pipeline recovery mechanism restores a correct program stage.

Das et al. [81] proposed Razor II which is an advanced version of Razor. Instead of performing both error detection and correction in Razor flip-flops, Razor II performs only detection in the flip-flops and correction through architectural replay. As a result of the modification, the processor provides both low-energy operation through dynamic supply adaption as well as soft-error tolerance [81]. The power overhead for a Razor II flip-flop as compared with a conventional flip-flop for a 10% activity factor is 28.5%.

Mitra et al. [82] introduced a dual-FPGA architecture that enables autonomous self-repair, fast fault location techniques to identify the defective portion of the system upon error detection, quick recovery from temporary failures, and reconfiguration techniques to repair the system from permanent faults. They realized the architecture within 205% area overhead and 10% degradation of maximum clock speed [82].

Avirneni and Somani proposed low overhead SEU and SET mitigation techniques called SEM and STEM using the approach of multiple clocking of data for protecting combinational logic blocks from soft errors [83]. SEM, which is based on distributed and temporal voting of three registers, unloads the soft error detection overhead from the critical path of the systems. SEM achieves an average performance improvement of 26.6% over a conventional triple modular redundancy voter-based soft error mitigation scheme, while STEM outperforms SEM by 27.4% [83]. In addition, STEM adds timing error detection capability to guarantee reliable execution in aggressively clocked designs that enhance system performance by operating beyond worst-case clock frequency while tolerating soft errors [83].

HAFTA is a fault-tolerant architecture to protect both configuration and user bits in SRAM-based reconfigurable devices against MBUs [84]. The architecture employs duplication for error detection, and checkpoint and rollback with history flip-flops for error recovery. HAFTA imposes, in average, 175% area overhead, 74% dynamic power overhead and 25% performance overhead compared with unprotected designs [84].

Zhu et al. [85] presented methods to increase the reliability of SRAM-based FPGA (SFPGA) designs under neutron induced SEUs. They introduced an analytical approach

based on probabilistic transfer matrix (PTM) to estimate the relative reliability of designs mapped into SFPGAs. In addition, the proposed method can obtain the error sensitivity rating of basic nodes in SFPGAs. The experimental results showed that partial triple modular redundancy on the sensitive sub-circuits, which are indicated by the proposed analysis method, is effective for SEU mitigation with less area overhead and less time delay than that of full TMR.

Zhang et al. [86] presented a soft-error detection scheme called AUDITOR for flip-flop based pipeline structures. The proposed scheme employs a local-audit detection which can synchronize the asynchronous error-indicating signals, thereby providing a base for accurate controls of error recovery. They also proposed the short-path compensation technique to remedy the deficiency of SET detection capability. According to the experimental results, the area overhead incurred by the proposed method is insensitive to the compensation intensity and the method can achieve perfect SEU and SET fault coverage, short detection latency at the expense of modest area overhead, while about 70%, 25%, 93% power overhead for adder, multiplier and divider pipeline, respectively.

Chapter 3

Preliminaries

This chapter introduces the fundamentals on which the proposed method is based.

3.1 Error Detection and Error Correction Scheme

This research is based on triple algorithm redundancy. Considering the tolerability against multiple component error caused by a single soft-error, and mitigating time overhead, the first copy and the second copy are used for detecting an error, and the third copy is used as a retry (comparison-retry, shortly C-R) in the proposed design. When the actual behavior of a datapath in time order is considered, a computation block can roughly be divided into three procedures. An example of these procedures using the block A in Fig. 1(b) is shown as follows:

- Proc. 1: Execute the first copy $A^{(1)}$ and the second copy $A^{(2)}$ of a computation block A. Then write the outputs of $A^{(1)}$ and $A^{(2)}$ into registers $r^{(1)}$ and $r^{(2)}$, respectively.
- Proc. 2: Execute the comparator q to compare the outputs of $A^{(1)}$ and $A^{(2)}$ and write the comparison result of q into register r_q.
- Proc. 3-1: (In case q detects an error) Execute $A^{(3)}$ and overwrite the output of $A^{(3)}$ into $r^{(1)}$.
- Proc. 3-2: (In case q detects no error) Do nothing.

After the execution of the copies of the block A, the data in $r^{(1)}$ is sent to succeeding computation blocks. Based on this C-R mechanism, since q compares two data stored in $r^{(1)}$ and $r^{(2)}$ instead of two outputs of $A^{(1)}$ and $A^{(2)}$ directly, q can detect not only erroneous outputs of $A^{(1)}$ and $A^{(2)}$ but also faults on multiplexer. Once q detects an error, $A^{(3)}$ is executed and its output is sent to $r^{(1)}$ via multiplexer which is fault-free under the single soft-error assumption. In order to realize single fault tolerant datapaths, an error occurring at one procedure must be prevented from affecting the next procedures. Chapter 4.2 will present more specific conditions for single soft-error tolerant datapaths.

3.2 Triple Algorithm Redundancy (TAR)

It is assumed that a computation algorithm to be implemented is given by a pair (G, D_P), where G is a data dependency graph and D_P is the set of primary outputs. Let O

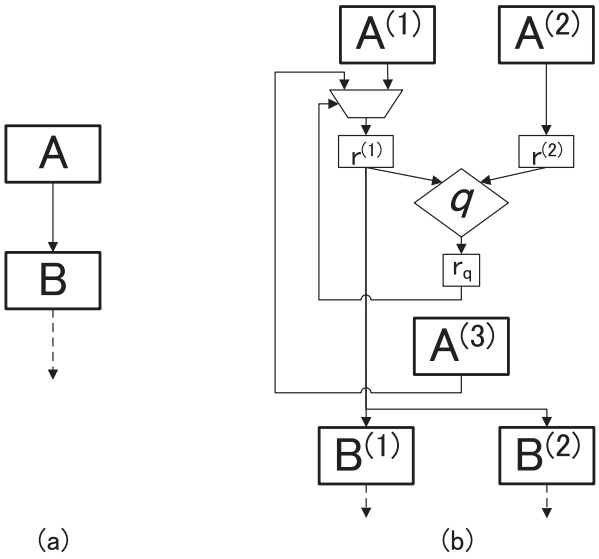

Figure 3.1: A sketch of the strategy for fault tolerance; (a) is an original computation algorithm. (b) shows a triplicate algorithm, where the first copy and the second copy are used for detecting error, and the third one is used for retry.

be the set of all operations in the original input computation algorithm, let D be the set of all variables in the original algorithm and let E be the set of all dependency between operations and variables. Then a given dependency graph G can be described as $G = (O \cup D, E)$. To synthesize a soft-error tolerant datapath, the given graph G is triplicated (three copies are denoted as $G^{(1)}$, $G^{(2)}$ and $G^{(3)}$), and then comparison-operations and dependencies related to the comparison-operations are inserted to form a resultant algorithm $(\tilde{G}, \tilde{D_P})$ which is finally mapped on hardware and time domains by high-level synthesis.

3.3 Cone-Partitioning/Comparison-Operation Insertion

It is defined that "check variables" are variables to be compared for soft-error detection, and they will be chosen selectively, not all operation results. Dependency subgraphs separated by those check variables are named "cones". Now we let D_Q ($\subseteq D$) be a set of the check variables and let Q be a set of comparison-operations corresponding to the variables in D_Q.

Definition 1 *A "cone" c_d ($\subseteq G$) denotes a subgraph which is induced by tracing back from a check variable $d \in D_Q$ to primary inputs or other check variables. Similarly, triplicate copies $c_d^{(1)}$ (called "main-cone"), $c_d^{(2)}$ (called "second-cone") and $c_d^{(3)}$ (called "retry-cone") of c_d are induced from the triplicate check variables $d^{(1)} \in G^{(1)}$, $d^{(2)} \in G^{(2)}$ and $d^{(3)} \in G^{(3)}$, respectively.*

Definition 2 *The set of main-cone $c_d^{(1)}(\subseteq G^{(1)})$, second-cone $c_d^{(2)}(\subseteq G^{(2)})$ and retry-cone $c_d^{(3)}(\subseteq G^{(3)})$ which are induced from a check variable d is called "stage".*

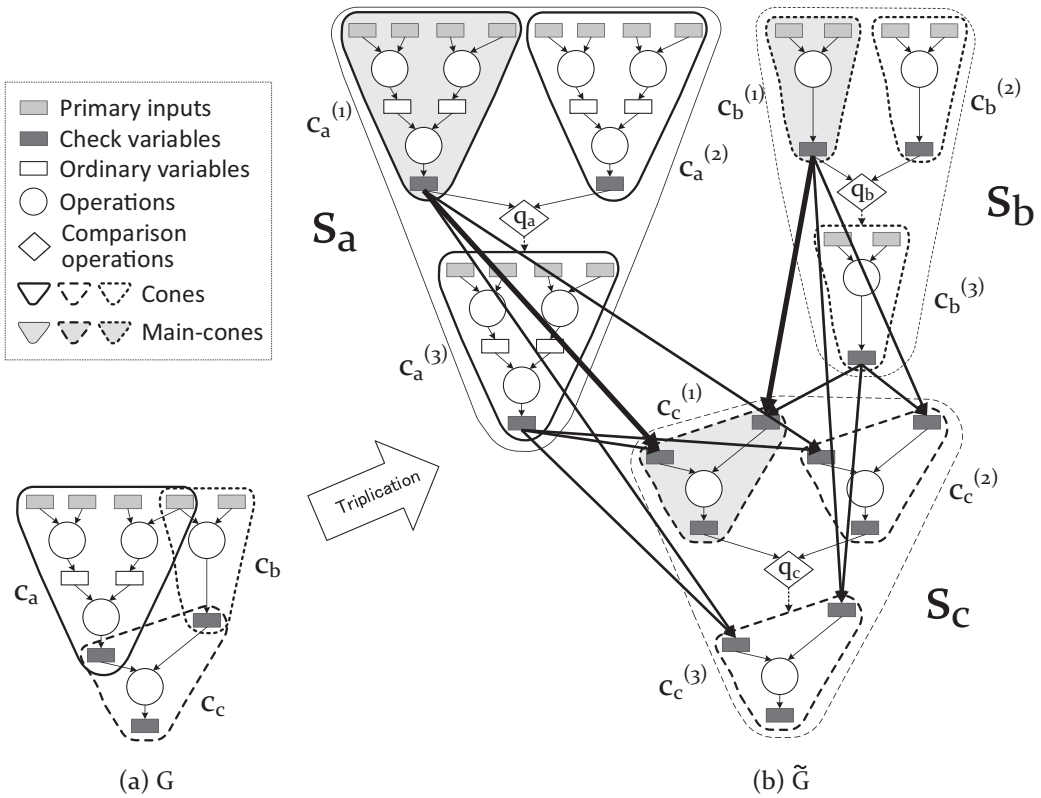

Figure 3.2: A stage s_a consists of $c_a^{(1)}$, $c_a^{(2)}$ and $c_a^{(3)}$; (a) An example of an original computation algorithm (b) An example of cone-partitioned triplicate algorithm

Definition 3 *Triplicate copies* $o^{(1)}(\in c_d^{(1)})$, $o^{(2)}(\in c_d^{(2)})$ *and* $o^{(3)}(\in c_d^{(3)})$ *of an operation* o $(\in c_d)$ *are elements of the same stage, and they are called "series operations".*

In this research, it is considered that all partitioned cones are disjoint, in other words, any pair of two cones will not share the same operation. Figure 3.2 shows an example of cone-partitioning. Selections of check variables determine the forms of the partitioned cones. The partitioned cones are triplicated so that an original cone has three same copies and those copies construct a stage. Then, comparison-operations are inserted into the corresponding stages.

The way how to select check variables for comparison-operations is an important factor for datapath optimization. Chapter 5 will present the optimization of the selection of check variables.

Chapter 4

Soft-Error Tolerant Datapath Synthesis Based on Speculative Resource Sharing

4.1 Motivation

As mentioned in Chap. 3.1 and 3.2, triple algorithm redundancy is the starting point of this research for soft-error tolerant datapath design. Concerning the tolerability against multiple component error due to a single soft-error in the proposed design, the main parts and the secondary parts are used for error detection, and the retry parts are used for error recovery.

However, a common disadvantage of redundancy is spatial or temporal overhead to achieve fault-tolerance. In order to address this problem, the author proposes an effective resource management method called speculative resource sharing. The new proposal will present at Sect. 4.3.

4.2 Conditions for Single Soft-Error Tolerant Datapaths

This section explains requested features for single soft-error tolerance and speculative resource sharing which is one of the important proposals in this study.

4.2.1 Fault/Error Model and Fault Tolerant Condition

In this study, it is assumed that a single soft-error can affect several spatial points simultaneously, hence it causes multiple component error, including errors on registers, functional units and other components at the same time. The proposed soft-error tolerant design is based on cone-level error masking which relies on error detection by comparing the results of the main-cone $c_d^{(1)}$ and the second-cone $c_d^{(2)}$ for each $d \in D_Q$, and error correction by executing the retry-cone $c_d^{(3)}$.

If the execution of a retry-cone overlaps the execution of the main-cone, the second-cone or the comparison-operation in the same stage, multiple component error due to a single soft-error may affect two cones or more (retry-cone and either main- or second-cone)

within the same stage simultaneously. For this reason, the execution order of operations in each stage should be constrained as follows:

Condition 1 *[Execution order in each stage] After a main-cone and the corresponding second-cone are executed, error detection with a comparison-operation is performed to check the results of these two cones. Subsequently, the corresponding retry-cone is executed only if the two results differ.*

In the C-R mechanism, if and only if an error is detected, the corresponding retry-cone is executed and its result is used immediately for the succeeding operations without error detection of the result. The validity of this treatment relies on the following assumption.

Assumption 1 *The probability of the recurrence of soft-errors in a short period is sufficiently low.*

On the other hand, triplicate data which are stored in three standard registers are not reliable anymore under the assumption of multiple component error caused by a single soft-error. Therefore, in order to guarantee the correctness of the inputs to a retry-cone even if its main- and second-cone are affected by a single soft-error, the author has decided to use specialized registers which inherently have a tolerance to a soft-error, such as BCDMR-ACFF [21]. More specifically, multi-bit soft-error tolerant registers are used to store input data of each cone (they are primary inputs or the outputs of main-cones and retry-cones). Also, 1-bit soft-error tolerant registers are used to store the outputs of comparison-operations. Multi-bit standard registers are used to keep other data such as interior data in cones and the outputs of second-cones.

4.3 Scheduling Algorithm under Speculative Resource Sharing (SRS)

In order to mitigate time overhead, the author proposes the concept of speculative resource sharing (SRS). In the proposed treatment, operations in a retry-cone are not executed as long as no error is detected. It means that resources bound to operations in a retry-cone are in idle state if two results of the corresponding main- and second-cone are identical. Under Assumption 1, the resources can be rebound in an idle state to other operations which have no dependency with the operations in the retry-cone. More specifically, operations in a retry-cone can share resources speculatively with operations of second-cones in different stages.

In accordance with Assumption 1, while a retry-cone is running (an error caused by a soft-error in main-cone and/or second-cone of the running retry-cone is detected), the correctness of other main-cones which are executed after the erroneous main- and second-cones of the running retry-cone is highly reliable. Based on this observation, resources can be managed more efficiently by SRS between error detection parts and error correction parts.

In Figure 4.1, if operations in $c_m^{(3)}$ and $c_n^{(2)}$ share resources speculatively ($m \neq n$ and there is no dependency between c_m and c_n), $c_n^{(1)}$ is unable to detect error when a soft-error affects some of $c_m^{(1)}$, $c_m^{(2)}$ and q_m, and as a result, $c_m^{(3)}$ is executed since the execution of $c_n^{(2)}$ is abandoned by $c_m^{(3)}$. Hence, it is needed to carefully manage the execution of $c_n^{(1)}$ so

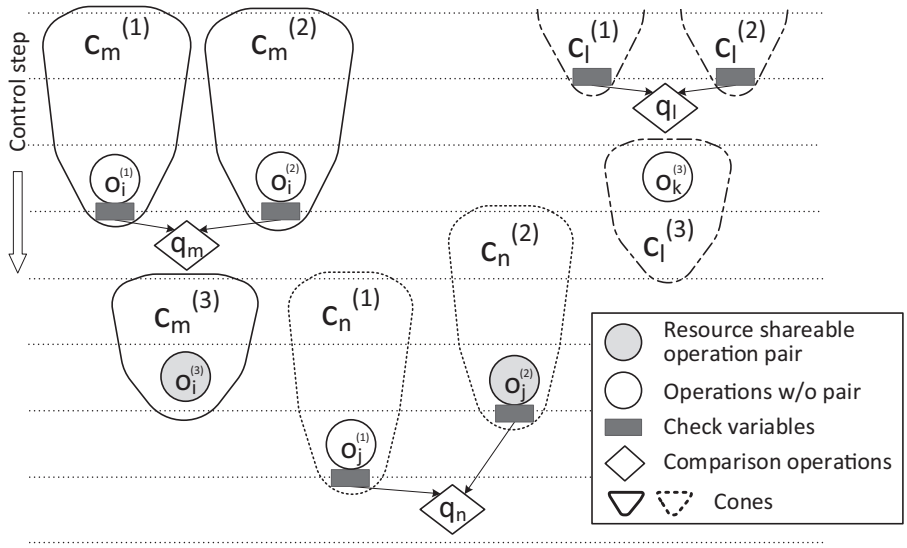

Figure 4.1: Scheduled DFG; An example of SRS between operations in second-cones and retry-cones. $o_i^{(3)}$ and $o_j^{(2)}$ can share a functional unit speculatively. However, $o_k^{(3)}$ and $o_i^{(2)}$ cannot because the execution of $c_m^{(1)}$ starts earlier than the execution of $c_l^{(2)}$ and it means that error detection and correction should be performed by each corresponding comparator and retry cone without SRS if $c_m^{(1)}$ (or $c_m^{(2)}$) and $c_l^{(1)}$ (or $c_l^{(2)}$) are affected by a single soft-error at the same time.

that the soft-error which affects $c_m^{(1)}$, $c_m^{(2)}$ and/or q_m may not affect $c_n^{(1)}$. Considering such behavior, the author introduces the second condition.

Condition 2 *[Speculative resource sharing] If and only if the execution of a main-cone $c_n^{(1)}$ is scheduled later than the execution of a comparison-operation q_m $(m \neq n)$, then SRS can be applied to a pair of operations, one in a retry-cone $c_m^{(3)}$ and the other in a second-cone $c_n^{(2)}$ which corresponds to the main-cone $c_n^{(1)}$.*

4.4 Synthesis Problem and Formulation based on ILP

In this section, the scheduling problem in a soft-error tolerant datapath synthesis considering SRS is formulated as an integer linear programming (ILP) problem.

4.4.1 Definitions of Variables and Constants

In a triplicate data flow graph $\tilde{G}$ of G, let $o_i^{(\ell)}$ be an operation in ℓth copy of G, that is, $G^{(\ell)}$. $(\ell = 1, 2, 3)$

- $x_{ij}^{(\ell)}$ is 1 if an operation $o_i^{(\ell)}$ is executed at control step j, 0 otherwise.
- z_{ij} is 1 if a comparison-operation q_i is executed at control step j, 0 otherwise.
- $u_{if}^{(\ell)}$ is 1 if an operation $o_i^{(\ell)}$ is executed on a functional unit f, 0 otherwise.
- w_{in} is 1 if a comparison-operation q_i is executed on a comparator n, 0 otherwise.

- $sp_{ii'}$ is 1 if $o_i^{(3)} (\in G^{(3)})$ and $o_{i'}^{(2)} (\in G^{(2)})$ share the same functional unit speculatively, 0 otherwise.
- ucs is the upper bound of control steps. (constant)
- K_O is the number of available functional units. (constant)
- K_Q is the number of available comparators. (constant)
- S is the upper bound of the overall schedule length.

4.4.2 ILP Formulation

It is assumed that every operation is a single-cycle operation and is bound to a single-type functional unit for transferring basic ideas with appropriate readableness. However, the formulas in this section can easily be expanded for multi-cycle operations and multi-type functional units.

General Scheduling Constraints

- Every operation $o_i^{(\ell)}$ is executed exactly once. Similarly, every comparison-operation q_i is executed exactly once.

$$\sum_{j=1}^{ucs} x_{ij}^{(\ell)} = 1\,, \quad \sum_{j=1}^{ucs} z_{ij} = 1 \tag{4.1}$$

- If $o_n^{(\ell)}$ is an immediate successor of $o_m^{(\ell)}$, $o_n^{(\ell)}$ must be scheduled after the execution of $o_m^{(\ell)}$.

$$\sum_{j=1}^{ucs} j \cdot x_{mj}^{(\ell)} < \sum_{j=1}^{ucs} j \cdot x_{nj}^{(\ell)} \tag{4.2}$$

  Dependency between operations and comparison-operations, where the immediate successors of a comparison-operation are only the operations in the corresponding retry-cone and the immediate predecessors of a comparison-operation are only operations in the corresponding main- and second-cone, is also considered.

- To minimize the overall schedule length, the author introduces a variable S which follows Eq. (4.3), and the objective of the proposed scheduling is to minimize S.

$$\sum_{j=1}^{ucs} j \cdot x_{ij}^{(\ell)} \leq S\,, \quad \sum_{j=1}^{ucs} j \cdot z_{ij} \leq S \tag{4.3}$$

Speculative Resource Sharing Constraints

In this section, constraints for SRS are shown.

- Every operation $o_i^{(\ell)}$ is bound to one functional unit. This constraint is represented by Eq. (4.4). Similarly, every comparison-operation is bound to one comparator.

$$\sum_{f=1}^{K_O} u_{if}^{(\ell)} = 1\,, \quad \sum_{n=1}^{K_Q} w_{in} = 1 \tag{4.4}$$

- Two operations $o_i^{(\ell)}$ and $o_{i'}^{(\ell')}$ which do not share a resource speculatively must not be bound to the same functional unit f at the same control step j. This constraint is given by Eq. (4.5). The same constraint between two comparison-operations is expressed by Eq. (4.6).

$$x_{ij}^{(\ell)} + u_{if}^{(\ell)} + x_{i'j}^{(\ell')} + u_{i'f}^{(\ell')} \leq 3 \tag{4.5}$$

$$z_{ij} + w_{in} + z_{i'j} + w_{i'n} \leq 3 \tag{4.6}$$

- For two operations $o_i^{(3)} (\in G^{(3)})$ and $o_{i'}^{(2)} (\in G^{(2)})$, when the main-cone of the stage to which $o_{i'}^{(2)}$ belongs is scheduled later than the execution of a comparison-operation q_m which belongs to the same stage as $o_i^{(3)}$, $o_i^{(3)}$ and $o_{i'}^{(2)}$ are a speculative resource sharable pair ($sp_{ii'}$ can take the value 1). This constraint is shown by the following:

$$\sum_{j=1}^{ucs} j \cdot z_{mj} - T \cdot (1 - sp_{ii'}) < \sum_{j=1}^{ucs} j \cdot x_{nj}^{(1)} \tag{4.7}$$

  where $o_n^{(1)}$ denotes each operation in a main-cone of the stage to which $o_{i'}^{(2)}$ belongs, and T is a sufficiently large constant larger than ucs.

- If two operations $o_i^{(3)}$ and $o_{i'}^{(2)}$ are a speculative resource sharable pair, these operations are allowed to be bound to the same functional unit f at the same control step j. This constraint is described as Eq. (4.8).

$$x_{ij}^{(3)} + u_{if}^{(3)} + x_{i'j}^{(2)} + u_{i'f}^{(2)} \leq 3 + sp_{ii'} \tag{4.8}$$

- If two operations $o_i^{(3)}$ and $o_{i'}^{(2)}$ share a resource speculatively, they cannot share a resource with another operation speculatively anymore.

$$\sum_{\text{Speculative resource sharable operations with } o_i^{(3)}} sp_{ii'} \leq 1 \tag{4.9}$$

$$\sum_{\text{Speculative resource sharable operations with } o_{i'}^{(2)}} sp_{ii'} \leq 1 \tag{4.10}$$

4.4.3 Multi-Cycle Soft-Error Tolerant Datapath

The schedule constraints and SRS conditions can be expanded so that they can manage multi-cycle soft-errors [65].

When k-cycle soft-error tolerant datapath synthesis is considered, the differences in schedule constraints are Eq. (4.2) and (4.7) in Sect. 4.4.2. Since a single soft-error can last k control steps in maximum, a comparison-operation must be executed k steps or more after the completion of its preceding operations. Similarly, immediate succeeding operations of a comparison-operation must be scheduled k steps or more after the completion of the comparison-operation. Thus it is needed to modify (4.2) as follows:

$$\sum_{j=1}^{ucs} j \cdot x_{mj}^{(\ell)} + k - 1 < \sum_{j=1}^{ucs} j \cdot z_{nj} \qquad (\ell = 1, 2) \tag{4.11}$$

$$\sum_{j=1}^{ucs} j \cdot z_{mj} + k - 1 < \sum_{j=1}^{ucs} j \cdot x_{nj}^{(3)} \tag{4.12}$$

For the same reason, (4.7) can be rewritten as the following:

$$\sum_{j=1}^{ucs} j \cdot z_{mj} + k - 1 - T \cdot (1 - sp_{ii'}) < \sum_{j=1}^{ucs} j \cdot x_{nj}^{(1)} \tag{4.13}$$

4.5 Soft-Error Tolerant Datapath Synthesis using Heuristic Scheduling Algorithm

The author proposes a heuristic scheduling and resource binding algorithm for single soft-error tolerant datapaths in this section. The proposed heuristic algorithm is based on LIST scheduling but employs a specialized priority function and a peculiar resource counting considering SRS.

In the following, let $\sigma : O^{(1)} \cup O^{(2)} \cup O^{(3)} \cup Q \to \mathbb{N}$ be an operation schedule, where $O^{(\ell)}, \ell = 1$, 2 and 3, is the set of operations in $G^{(\ell)}$.

4.5.1 Proposed Scheduling Algorithm

Algorithm 1 shows the proposed algorithm which is to perform the modified LIST scheduling under given resource constraints [27]. The following notations are employed to explain the proposed scheduling algorithm.

- $L_{cs,r}$: the list of ready operations of functional unit type r at control step cs
- $U_{cs,r}$: the list of executing operations of functional unit type r at control step cs
- $v_{r,k}$: the k-th operation of functional unit type r
- res_inuse : the number of functional units in use

Scheduling proceeds from control step 1 toward the last control step as the original LIST scheduling does. In each control step cs, at first, operations of functional unit type r which are scheduled already and are executing at control step cs, are registered to $U_{cs,r}$. Ready operations which are not yet scheduled are registered to $L_{cs,r}$. After

Algorithm 1 Modified LIST scheduling algorithm

Require: $N_{max} \leftarrow$ # of resource types
1: $cs \leftarrow 0$;
2: **while** (until all operations are scheduled) **do**
3: $cs \leftarrow cs + 1$;
4: **for** ($r \leftarrow 1$; $r \leq N_{max}$; r++) **do**
5: $res\_inuse \leftarrow 0$;
6: **for** ($k \leftarrow 1$; $k \leq$ # of operations (type r); k++) **do**
7: **if** ($v_{r,k}$ is already scheduled) **then**
8: **if** (execution of $v_{r,k}$ is not finish yet) **then**
9: Register $v_{r,k}$ in $U_{cs,r}$;
10: **end if**
11: **else if** (executions of all immediate predecessors of $v_{r,k}$ are already finished) **then**
12: Register $v_{r,k}$ in $L_{cs,r}$;
13: **end if**
14: **end for**
15: $res\_inuse \leftarrow$ **count_occupied_FU**($U_{cs,r}$);
16: **schedule_with_speculative_share**($cs, r, res\_inuse, L_{cs,r}$);
17: **end for**
18: **end while**

that, the function **count_occupied_FU** is called, which counts the number of functional units (res_inuse) presently occupied by operations in $U_{cs,r}$, and then the function **schedule_with_speculative_share** is called, which chooses operations to be scheduled to control step cs.

4.5.2 Scheduling Priority

In LIST scheduling, operations are chosen and assigned to a control step according to the priority given to each operation. Of course, the main objective of this priority is to guide the proposed scheduler toward a minimum-control-step schedule. In addition to it, increasing SRS is also aimed by this priority since SRS can increase nominally the number of operations to be executed in one control step more than the number of available functional units, which may result in a shorter schedule.

First of all, priority and latency of a stage is introduced. Priority of a stage is defined as the smallest control step of ALAP schedule [1] over all operations in the stage. Latency of a stage is defined as the difference between the smallest control step and the largest control step of the stage in ALAP schedule.

$$\text{Priority of stage } m \ = min(\bigcup_{\ell=1}^{2} \sigma_{ALAP}(c_m^{(\ell)})) \ , \tag{4.14}$$

$$\text{where } \sigma(c_m^{(\ell)}) = \{x \mid \sigma(o_i^{(\ell)}) = x, o_i^{(\ell)} \in c_m^{(\ell)}\}$$

$$\text{Latency of stage } m \ = max(\sigma_{ALAP}(c_m^{(3)})) - min(\bigcup_{\ell=1}^{2} \sigma_{ALAP}(c_m^{(\ell)})) + 1 \tag{4.15}$$

[1] This is one of time-constrained schedules that all operations are scheduled as late as possible within fixed latency.

The priority of an operation is determined by applying the following factors in this order (a tie in the first factor is broken by the second factor, a tie in the second factor is broken by the third factor, and so on).

(1) Operations which are in a stage having higher priority have higher priority. As a result, a stage which started earlier can finish earlier. Since an input computation algorithm is triplicated and comparison-operations are inserted based on cone-partitioning, if a stage which started earlier remains without being chosen and only operations in other stages are scheduled, operations in the remained stage, especially comparison-operation in the stage, can be a bottleneck on the entire schedule.

(2) Operations which are in a stage having smaller latency have higher priority. Since operations in a stage which is expected to finish earlier are easier to satisfy Condition 2, those operations are easier to find speculative resource sharing operation pairs.

(3) Operations which have smaller ALAP schedules are given higher priority.

(4) Operations which are placed in a critical path are given higher priority.

(5) Among series operations, an operation in second-cone is given higher priority than the other to find a speculative resource sharing operation pair aggressively. As a result, the possibility of SRS can be augmented.

Using these factors, priority is given to every operation.

4.5.3 Selecting Operations

Even if the priority of ready operations is fixed, selecting operations to be scheduled at the current control step is not straightforward, since we need to manage a complicated resource sharing between operations in second- and retry-cones. If speculatively sharable operations are chosen preferentially, the number of operations to be scheduled in the current control step can increase, and, as a result, latency can decrease. The proposed SRS is a special resource sharing between operations in retry-cones and those in second-cones. That is, SRS is a set of disjoint pairs of operations in retry-cones and those in second-cones, which is modeled as a matching between operations in retry-cones and those in second-cones. Hence, the author decided to manage SRS-aware resource counting by a matching on a bipartite graph. In order to construct a bipartite graph, ready operations are divided into two subsets X and Y, where X includes operations in second-cones and Y includes operations in retry-cones, and edges are allocated between operations which are speculatively resource sharable. On the other hand, operations in main-cones will not share a resource speculatively. Moreover, the case should be considered that operations in second- and retry-cones are scheduled without SRS. In order to deal with such cases using a single framework, operations in main-cones are included as vertices in set X, and two additional vertex sets X_C in Y and Y_C in X are introduced as dummy vertices to be paired with operations (vertices) scheduled without SRS. Consequently, the final bipartite graph consists of X_T which includes X and Y_C, and Y_T which includes Y and X_C.

Here let X be a set of operations in main- or secondary-cone that exist in $L_{cs,r}$. Similarly, let Y be a set of operations in retry-cone that exist in $L_{cs,r}$.

$$X = \{\, x \mid x \in L_{cs,r} \wedge x \in G^{(1)} \cup G^{(2)} \,\}$$

$$Y = \{\, y \mid y \in L_{cs,r} \wedge y \in G^{(3)} \,\}$$

In order to choose speculative resource sharing pairs aggressively, a bipartite graph $H = (X_T \cup Y_T, W)$ with X_T and Y_T will be used as its partite sets.

- One partite set X_T is defined as $X_T = X \cup Y_C$, where Y_C is a set of auxiliary vertices with $|Y_C| = |Y|$.
- The other partite set Y_T is defined as $Y_T = Y \cup X_C$, where X_C is a set of auxiliary vertices with $|X_C| = |X|$.

Vertex weight $w(z) \in \mathbb{N}$ is defined as priority of an operation z. Also, it is considered that weights of all auxiliary vertices in X_C or Y_C are 0, that is, $\forall z \in X_C \cup Y_C$, $w\,(z) = 0$. The edge set W of H is defined as follows.

$$\begin{aligned} W \;=\; & \{\, \{x, P_X(x)\} \mid x \in X \,\} \cup \{\, \{P_Y(y), y\} \mid y \in Y \,\} \\ & \cup \{\, \{x, y\} \mid x\,(\in X \cap G^{(2)}) \text{ and } y\,(\in \mathrm{Y}) \text{ can} \\ & \text{share a resource speculatively} \,\} \end{aligned}$$

where $P_X : X \to X_C$ and $P_Y : Y \to Y_C$ are arbitrary one-to-one mapping from X to X_C and from Y to Y_C, respectively. In addition, edge weight $w\,(e = \{x, y\}) \in \mathbb{Z}_+$ is defined as the sum of weights of two end vertices x and y, that is, $w\,(e = \{x, y\}) = w\,(x) + w\,(y)$.

Once H is constructed, selecting an edge $e = \{\, x, y \,\}$ from H means that two operations which are represented by the end vertices x and y of e share a resource speculatively. If an end vertex of e is an auxiliary vertex (a vertex in either X_C or Y_C), it means that the operation which is represented by the other end vertex occupies a resource without SRS. As a result, an operation selection considering SRS can be considered as a problem to find a size-constrained maximum weight matching on H.

Algorithm 2 shows a detailed description of the function **schedule_with_speculative_share** [27]. After a graph H is constructed, speculative resource sharing operation pairs are chosen by a greedy selection procedure. More specifically, matching edges are selected in descending order of the edge weights within the number of available resources. Figure 4.2 depicts an example of operation selection by Algorithm 2. In the selection procedure, the edges with weight-28, 20 and 19 are chosen in descending order of edge weights. When the edge with weight-19 is selected, the procedure at the current control step will be finished since n_r reaches the number of available resources, 3. As a result, selected operation pairs are (15, 13), (20, 0) and (10, 9). In addition, the unmatched vertex (operation) with weight-8 is carried over to the next control step.

4.6 Experimental Results

The author has implemented the proposed scheduling algorithm as a computer program. In order to evaluate the proposed heuristic algorithm, solutions obtained from the proposed heuristic algorithm are compared with solutions obtained from two different sources. One is a conventional LIST scheduling algorithm and the other is an ILP solver which solves the proposed ILP formulation. These three methods are applied to various computation algorithms, such as, 16-point fast Fourier transform (16FFT), 8-point inverse discrete cosine transform (8IDCT), 16-point FIR filter (16FIR), autoregressive filter (ARF),

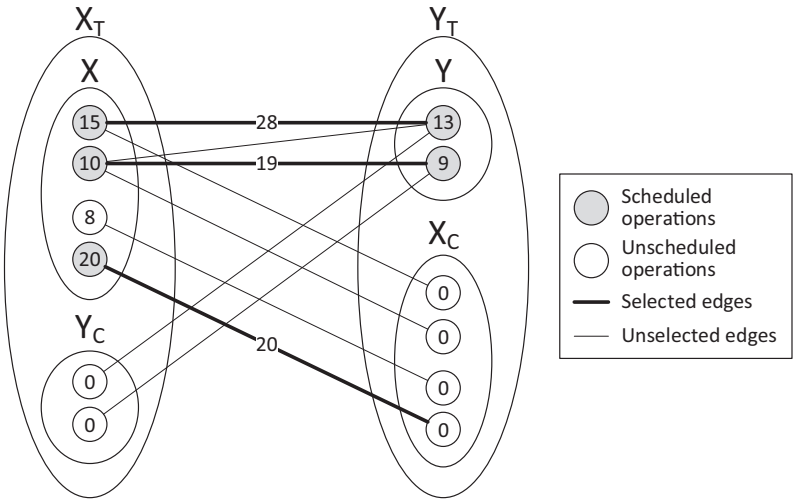

Figure 4.2: An example of operation selection produced by greedy selection procedure; Operations which have vertex weights 10 and 15 are elements of $G^{(2)}$. Other operations in X are elements of $G^{(1)}$. The number of available resources is 3.

Algorithm 2
Algorithm of function schedule_with_speculative_share()

Require: $cs \leftarrow$ Current control step
Require: $r \leftarrow$ Currently considered resource type
Require: $n_r \leftarrow$ Available # of resources with type r
Require: $L_{cs,r} \leftarrow$ List of ready operations with type r at step cs
1: Construct a bipartite graph H based on $L_{cs,r}$;
2: **for** ($i \leftarrow 0$; $i <$ Total # of W; i++) **do**
3: **if** ($n_r > 0$) **then**
4: Select an edge e from W which has the largest edge weight;
5: Schedule two end vertices of e at current control step cs;
6: Remove the edge e and its end vertices from H
7: $n_r \leftarrow n_r - 1$;
8: **else**
9: break;
10: **end if**
11: **end for**

fifth-order elliptic wave digital filter (5EWDF), and inverse discrete cosine transform with column-wise decomposition (IDCT-c). Three types of functional units, namely multipliers, ALUs and comparators, three types of registers, namely multi-bit soft-error tolerant registers, 1-bit soft-error tolerant registers and multi-bit standard registers, and multiplexers are utilized as allocated resources in datapath synthesis experiments. In addition, the maximum duration of a single soft-error is set to one control step in the experiments.

A conventional LIST scheduling algorithm (shortly, Convent.), to which SRS is not applied, and the proposed LIST scheduling algorithm with SRS (shortly, SRS) were executed on a computer which consists of a Intel Xeon E5-2670 (2.60GHz) processor and 6GB main memory. To find exact solutions from ILP formulations, the ILP solver Gurobi Optimizer 6.0.0 was performed on one node, which consists of two Intel Xeon E7-8837 (2.66GHz) processors and 128GB memory, in a massively parallel computer.

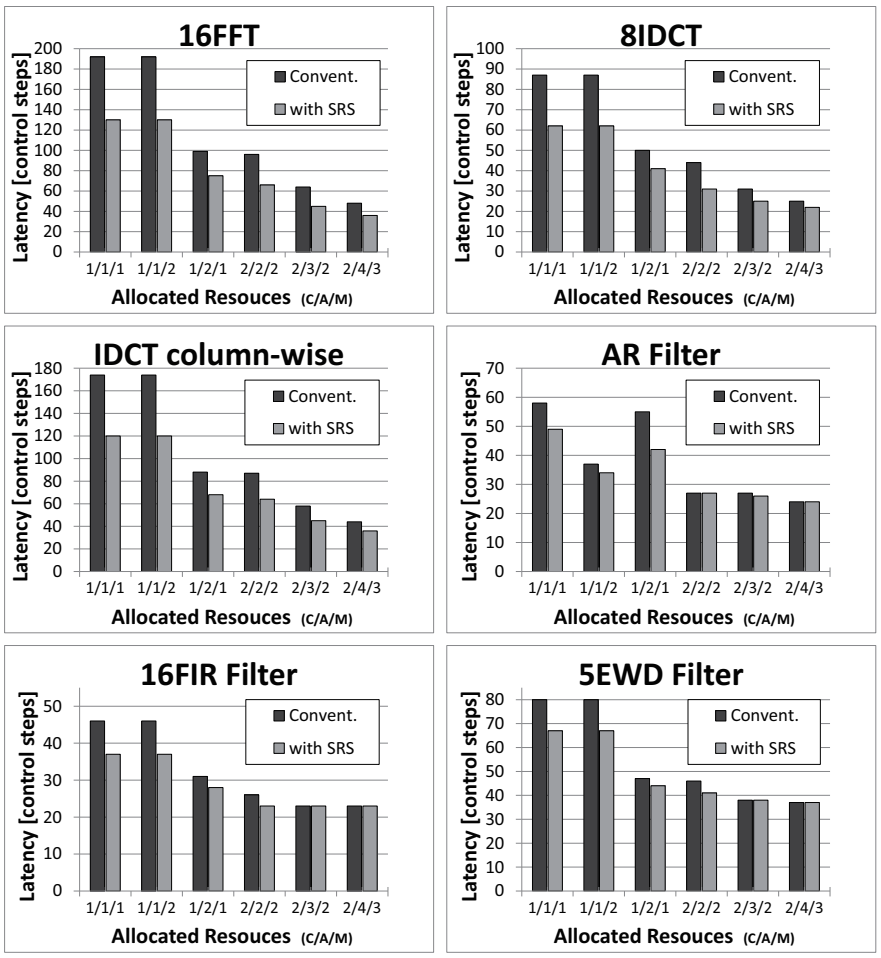

Figure 4.3: Experimental results in various computational algorithms with heuristics. Each column corresponds to a different choice of allocated resources (C: Comparator, A: ALU, M: Multiplier). Every column in each graph has two bars, black one and gray one. Black bar represents scheduling result (latency) with a conventional method and gray one represents proposed scheduling result with SRS.

Table 4.1: Latency improvement rate using heuristics

Computation algorithm	Total # of operations (triplicate algorithm)	Max. improvement rate [%] & allocated resources (C/A/M)
16FFT	328	32.3 (1/1/1)
IDCT-c	234	31.0 (1/1/1)
8IDCT	160	29.5 (2/2/2)
ARF	96	23.6 (1/2/1)
16FIRF	80	19.6 (1/1/1)
5EWDF	120	16.3 (1/1/1)

4.6.1 Performance Evaluation in Latency

Figure 4.3 and Table 4.1 show experimental results from a viewpoint of latency. Each graph in Fig.4.3 illustrates the achieved latencies using a conventional LIST scheduling algorithm and the proposed LIST scheduling algorithm under different resource alloca-

tions. In 16FFT, 8IDCT and IDCT-c, the proposed method achieved smaller latency than a conventional method for all variations of resource constraints. On the other hand, in ARF, 16FIR and 5EWDF, the proposed method obtained almost the same latency in case of larger resource constraints such as 2/3/2 and 2/4/3. Table 4.1 shows the maximum improvement rates in latency between conventional LIST scheduling results and the proposed LIST scheduling results with SRS. The second column in Table 4.1 shows the total number of operations as an indicator of the size of each computation algorithm. The third column shows the maximum latency improvement rates and allocated resources which achieve the maximum latency improvement rate. Each maximum rate is decided by choosing a best one in latency among the six rates $\big($(latency achieved with a conventional LIST scheduling $-$ latency achieved with SRS)$\div$(latency achieved with a conventional LIST scheduling)$\big)$ with the six kinds of resource allocations. It can be found from those results that, for every application algorithm, latency is improved more or less when SRS is applied. It is also found that the proposed method is more effective when a computation algorithm possesses higher parallelism (in this example, 16FFT, IDCT-c and 8IDCT) and the improvement is larger when a smaller number of resources is allocated. The reason is that, when a computation algorithm possesses higher parallelism, there are more possibilities to share resources speculatively between retry-cones and second-cones and, in consequence, many chances of SRS can conceal the extension of latency. When the number of allocated resources is small, latency becomes larger due to the resource limitation, and the latency improvement achieved by SRS is remarkable. On the other hand, when the number of allocated resources is large enough, latency is dominated mainly by a critical path length, and SRS contributes little to latency improvement.

Table 4.2 shows solutions (latencies) and their computation time obtained by a conventional method, the proposed method with SRS, and an exact method using ILP solver. In this experiment, the author limited the maximum execution time of the solver to one day. When the solver continued to work beyond one day, it was aborted and a provisional solution was adopted. When even a provisional solution was not obtained by one day run, N/A is noted instead of a solution.

The results indicate that the proposed heuristic algorithm generates slightly inferior solutions compared with exact solutions in some cases. Nevertheless, the author can claim that the proposed method with SRS using heuristics produces high quality solutions in practical computation time and the solutions obtained from heuristics are comparable in latency to the exact solutions obtained by an ILP solver.

4.6.2 Reliability against Soft-Errors

In this subsection, we will compare the reliability between datapaths with a conventional method and the proposed method. Let p be the probability that a single soft-error occurs at a control step and, in consequence, it affects several components. Also, let $P(A)$ be the probability of an event A. If we consider an event that no soft-error occurs during cs control steps, its P is described by the following equation:

$$P(\text{no soft-error in } [1,cs]) = (1-p)^{cs} \tag{4.16}$$

In general, as for an event that soft-errors occur n different control steps during latency cs, its probability is given as:

$$P(n \text{ soft-errors in } [1,cs]) = {}_{cs}C_n \cdot p^n \cdot (1-p)^{(cs-n)} \tag{4.17}$$

Table 4.2: Experimental Results in Latency

Compu. Algo.	# of Op.s	Resource Constrains (C/A/M)	Conventional method (CED+Retry)				The Proposed Method (SRS)			
			Heuristics		ILP		Heuristics		ILP	
			solution	time [s]	sol.	time [s]	sol.	time [s]	sol.	time [s]
16FFT	88	1/1/1	192	0.1	N/A	> 86400.0	130	0.1	N/A	> 86400.0
		1/1/2	192	0.1	N/A	> 86400.0	130	0.1	N/A	> 86400.0
		1/2/1	99	0.1	N/A	> 86400.0	75	0.1	N/A	> 86400.0
		2/2/2	96	0.1	96	16578.0	66	0.1	N/A	> 86400.0
		2/3/2	64	0.1	64	1459.2	45	0.1	N/A	> 86400.0
		2/4/3	48	< 0.1	48	647.8	36	0.1	N/A	> 86400.0
8IDCT	42	1/1/1	87	0.1	87	1056.3	62	0.1	63†	> 86400.0
		1/1/2	87	0.1	87	14077.8	62	0.1	62†	> 86400.0
		1/2/1	50	0.1	46†	> 86400.0	41	0.1	37†	> 86400.0
		2/2/2	44	0.1	44	130.9	31	< 0.1	N/A	> 86400.0
		2/3/2	31	0.1	30	140.8	25	0.1	N/A	> 86400.0
		2/4/3	25	0.1	23	60.5	22	< 0.1	22†	> 86400.0
IDCT-c	62	1/1/1	174	0.1	174	9668.8	120	0.1	N/A	> 86400.0
		1/1/2	174	0.1	N/A	> 86400.0	120	0.1	N/A	> 86400.0
		1/2/1	88	0.1	87	4152.6	68	0.1	N/A	> 86400.0
		2/2/2	87	0.1	87	1331.5	64	0.1	N/A	> 86400.0
		2/3/2	58	0.1	58	962.9	45	0.1	N/A	> 86400.0
		2/4/3	44	0.1	44	209.9	36	0.1	N/A	> 86400.0
ARF	28	1/1/1	58	0.1	56	4900.6	49	0.1	48†	> 86400.0
		1/1/2	37	0.1	37	30.3	34	0.1	32	148.4
		1/2/1	55	0.1	54	596.5	42	0.1	40	11051.6
		2/2/2	27	0.1	27	5.1	27	0.1	26	3784.4
		2/3/2	27	0.1	27	8.5	26	0.1	26	8319.3
		2/4/3	24	0.1	24	0.1	24	0.1	24	6205.5
16FIRF	23	1/1/1	46	0.1	46	4.9	37	< 0.1	35	1182.3
		1/1/2	46	0.1	46	5.5	37	0.1	35	373.2
		1/2/1	31	0.1	31	5.2	28	0.1	26	37.2
		2/2/2	26	0.1	25	2.0	23	< 0.1	23	54.9
		2/3/2	23	0.1	23	0.1	23	0.1	23	9.2
		2/4/3	23	0.1	23	0.1	23	0.1	23	6.7
5EWDF	34	1/1/1	80	< 0.1	80	258.5	67	0.1	64†	> 86400.0
		1/1/2	80	0.1	80	212.6	67	0.1	62†	> 86400.0
		1/2/1	47	0.1	46	11.0	44	0.1	43†	> 86400.0
		2/2/2	46	0.1	45	33.0	41	0.1	40	4778.2
		2/3/2	38	0.1	38	2.5	38	0.1	38	76.6
		2/4/3	37	0.1	37	0.4	37	0.1	37	10.1

†A provisional solution is shown. Since the solver was still operating until 24h (86400s) passed, it was aborted.
N/A: No solution is found. Since the solver was still operating without any solution until 24h passed, it was aborted.
C: Comparator/ A: ALU/ M: Multiplier

Now, a reliability $R(A)$ is defined as the error-free probability against an event A. When we consider soft-errors occur n different control steps during latency cs, there are ${}_{cs}C_n$ possible error distribution patterns. If a datapath masks errors in r_n ($\leq {}_{cs}C_n$) patterns among them, the reliability R is described as the following:

$$R(n \text{ soft-errors in } [1,cs]) = r_n \cdot p^n \cdot (1-p)^{(cs-n)} \tag{4.18}$$

Finally, overall reliability R(all possible error patterns)

$$= \sum_{n=0}^{cs} R(n \text{ soft-errors in } [1,cs])$$

$$= \sum_{n=0}^{cs} r_n \cdot p^n \cdot (1-p)^{(cs-n)} \tag{4.19}$$

where r_n is the number of masked error patterns with n soft-errors. r_n is obtained by fault injection simulation. In this simulation, faults are injected based on control step. When we consider soft-errors in n different control steps, ${}_{cs}C_n$ different patterns of fault injection tests are examined. If the datapath produces error-free primary outputs under a pattern of error, then that trial is judged as an error-masked pattern. Thus, r_n is the total number of all error-masked patterns. For example, when n is 2 and latency is 5, possible error distribution patterns are (1,2), (1,3), (1,4), (1,5), (2,3), (2,4), (2,5), (3,4), (3,5) and (4,5). If error-masked patterns are (1,4), (1,5), (2,4), (2,5) and (3,5) after fault injection tests, r_n becomes 5.

In the following numerical evaluations, the terms of larger number of soft-errors will be truncated, and the reliability considering up to 3 soft-errors will be evaluated. In addition, we consider the following:

- When a soft-error occurs at a control step, it affects all components at the control step. Therefore, this evaluation considers the worst case.
- When a standard register is in failure caused by a soft-error, the data which is stored in the register is corrupted at the same control step.
- If an ALU or multiplier is in failure due to a soft-error, incorrect output data of the ALU or multiplier is latched in a register when the control step is changed to the next step.
- If a comparator is in failure caused by a soft-error and two inputs of the comparator are identical, the comparator outputs an incorrect comparison result, that is, two inputs are different. As a result, the retry-cone is executed. On the other hand, if a comparator is in failure due to a soft-error and two inputs are different, the comparator outputs an incorrect comparison result, that is, two inputs are identical. Consequently, the retry-cone will not be executed; furthermore, the erroneous outputs of the corresponding main-cone (and second-cone) will not be corrected.

According to [34], soft error rate (SER) is 1153 FIT/Mbit on 40nm technology SRAM at ground level. In the numerical evaluations, it is assumed that a soft error rate p in every control step is 0.0001. In case the operation frequency is 1GHz, this rate can be converted to 3.6×10^{17} FIT/chip. This value seems impractically large compared with SER on the natural terrestrial environment [34]. The proposed datapath design with SRS tolerates a single soft-error, and reliability degradation due to SRS will be revealed by multiple-time soft-error. In order to highlight the reliability degradation due to SRS, the probability of multiple-time soft-error have been increased by choosing a larger probability p in the numerical evaluation.

Figure 4.4 shows the evaluation results. These results revealed that there is no huge difference except some cases in reliability between datapaths a conventional method and

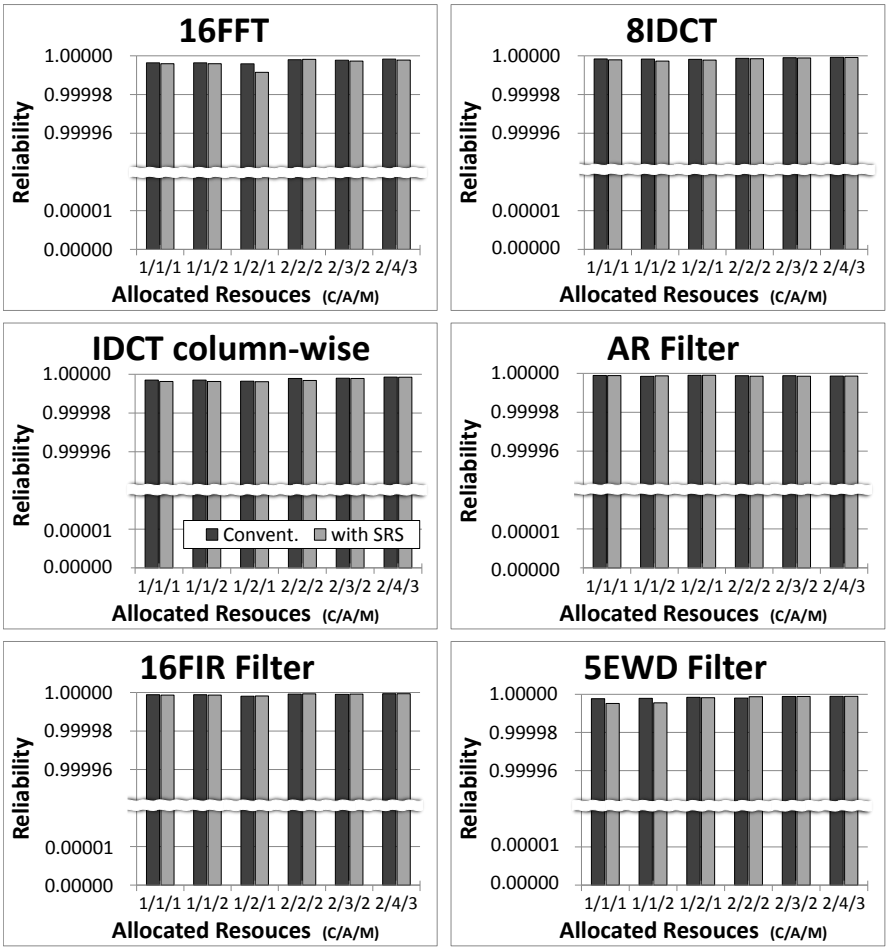

Figure 4.4: Reliability comparison for various computational algorithms. Each column corresponds to a different choice of allocated resources (C: Comparator, A: ALU, M: Multiplier). Every column in each graph has two bars, black one and gray one. Black bar represents scheduling result (latency) without a conventional method and gray one represents proposed scheduling result with SRS.

the proposed method with SRS. In case of 16FFT under resource constraints 1/2/1, the reliability of the proposed method is relatively inferior to a conventional method. We can see similar results in case of 5EWDF under 1/1/1 and 1/1/2. On the other hand, in case of 16FFT under 2/2/2 and 5EWDF under 2/2/2, the proposed method are slightly superior to a conventional method in reliability.

As mentioned earlier, these reliability evaluations consider the worst case that all operation results and all data stored in standard registers at a control step are corrupted by a single soft-error. Also, these evaluations consider more than one fault during latency *cs*. Due to applying SRS, in general, latency becomes smaller and, as a result, the probability that soft-errors occur during latency *cs* decreases. If an error occurs during the execution of a main-cone and/or a second-cone and the second error occurs during the execution of the retry-cone in the same stage, the second error will not be fixed in both conventional and proposed datapaths. However, not only the case that the second error falls in the retry-cone, but also the case that the second error falls in a main-cone (of another stage) whose second-cone shares resources speculatively with the retry-cone,

the proposed datapath can not fix the error. Thus, due to applying SRS, the probability that the entire system normally operates against more than one fault during latency *cs* decreases. As a result, the superiority in reliability between datapaths without and with SRS may change depending on the scheduling results. Despite several exceptions in the evaluation results, it can be said that the proposed method retains almost the same level of performance in soft-error tolerance compared with a conventional method based on C-R mechanism without SRS.

4.6.3 Quantitative Evaluation for Area Estimation

To evaluate hardware overhead due to triplication, a chip area is estimated by counting the number of functional units, registers and multiplexers. Left-edge algorithm is applied to register binding in this evaluation. Table 4.3 shows the number of allocated hardware resources which are needed to realize soft-error tolerant datapaths. These results clarify that there is no huge difference in estimated area between datapaths a conventional method and the proposed method with SRS when the numbers of functional units are identical between the two datapaths. However, depending on computation algorithms and available resources, the proposed method produces smaller chip areas. We can see that datapaths with a conventional method use more soft-error tolerant registers (both multi-bit and 1-bit) than datapaths designed by the proposed method. Particularly, this tendency stands out in 16FFT among other computation algorithms. The main reason of this trend can be explained as follows: If scheduling is focused on each operations without considering cone-based priorities, instead of operations in the cones which are already scheduled, new operations in other cones which are not yet scheduled may be selected. In order to store the comparison results and the outputs of main-cones and retry-cones, 1-bit soft-error tolerant registers and multi-bit soft-error tolerant registers are needed. When new operations in other cones are selected, the lifetimes of those results and output are extended and overlapped with each other. As a result, more soft-error tolerant registers are needed to store those data. In spite of minor exceptions, it can be claimed that the proposed method keeps almost the same level of performance in chip area compared with a conventional C-R method.

4.7 Summary

Concerning soft-error tolerant datapath synthesis based on triplication of an input computation algorithm, constraints for soft-error tolerance and a scheduling algorithm considering speculative resource sharing (SRS) are proposed. A datapath circuit designed by the proposed method tolerates multi-component and multi-cycle error caused by a single soft-error. From the results of soft-error tolerant datapath synthesis experiments, it is found that SRS achieves a maximum 32.3% improvement in latency, while keeping the comparable levels in reliability and in chip area compared with datapaths based on a conventional method. Specifically, it is discovered that the proposed method is more effective when an input computation algorithm possesses higher parallelism, and the number of allocated resource is relatively small. In many embedded devices, chip sizes are restricted. In such resource-limited applications, the proposed method can make latency smaller without increasing chip size. Keeping or decreasing the chip size is preferable also in yield and in static power consumption.

Table 4.3: Experimental Result in Area Estimation

Compu. Algo.	# of Op.s	Resource Constrains (C/A/M)	Conventional method (CED+Retry) # of Registers FT	FT 1bit	Standard	# of Multiplexers	The Proposed Method (SRS) # of Registers FT	FT 1bit	Standard	# of MUXs
16FFT	88	1/1/1	32	16	1	32	25	2	2	29
		1/1/2	32	16	1	32	25	2	2	29
		1/2/1	32	15	7	44	31	8	11	42
		2/2/2	32	15	1	49	26	2	1	40
		2/3/2	32	15	3	60	28	6	7	51
		2/4/3	32	14	7	65	32	5	6	57
8IDCT	42	1/1/1	16	7	1	22	15	3	5	22
		1/1/2	16	8	1	23	13	3	4	24
		1/2/1	16	7	5	27	14	4	6	25
		2/2/2	16	7	2	32	13	3	3	30
		2/3/2	16	8	2	35	15	4	5	35
		2/4/3	16	7	4	40	16	4	6	44
IDCT-c	62	1/1/1	17	8	1	22	15	3	2	22
		1/1/2	17	8	1	22	15	3	2	22
		1/2/1	19	7	8	32	17	3	10	30
		2/2/2	17	7	2	32	15	5	5	35
		2/3/2	17	6	7	40	17	3	8	38
		2/4/3	17	5	6	46	20	4	8	46
ARF	28	1/1/1	30	6	1	23	27	2	2	22
		1/1/2	27	3	4	28	27	2	3	29
		1/2/1	30	7	1	26	27	2	2	23
		2/2/2	27	2	4	26	27	2	4	26
		2/3/2	27	2	4	26	27	2	4	26
		2/4/3	28	2	6	31	28	2	6	31
16FIRF	23	1/1/1	18	4	2	18	17	2	2	18
		1/1/2	18	4	2	18	17	2	2	18
		1/2/1	17	3	11	23	17	3	11	23
		2/2/2	18	4	3	24	18	3	3	26
		2/3/2	18	3	6	30	18	3	7	30
		2/4/3	19	3	8	37	19	3	9	39
5EWDF	34	1/1/1	17	5	5	22	14	4	8	22
		1/1/2	17	5	3	26	14	4	7	26
		1/2/1	15	3	6	26	13	4	4	24
		2/2/2	14	4	5	28	13	3	4	27
		2/3/2	13	3	5	32	13	3	3	29
		2/4/3	13	3	3	32	13	4	4	32

C: Comparator/ A: ALU/ M: Multiplier

Chapter 5

Latency-Optimized Selection of Check Variables

5.1 Motivation

Speculative resource sharing, which is shown in Chap. 4 is introduced for hardware/time overhead mitigation. However, a strict constraint is imposed on SRS. To satisfy the constraint much easier and maximize the possibility of SRS, the author proposes a latency-aware cone-partitioning algorithm. The more comparison-operations are selected, the more cones, which are subgraphs of the original input data flow graph, are partitioned. As a result, latency improves because fine-grained cones are relatively easier to fulfill the constraint than coarse-grained cones. On the other hand, latency may become larger as increasing the number of comparison-operations. Thus, deciding insertions of comparison-operations is an important factor in design optimization, as mentioned in Chap. 3.3.

5.2 Optimized Check Variable Selection Algorithm under SRS

In the previous chapter, the proposed method has mainly focused on the scheduling and the binding algorithms for C-R based soft-error tolerant datapath synthesis, and check variables are treated as being given a priori. However, the possibility of SRS and the performance of a synthesized datapath depend on selecting check variables. The main objective of this thesis is to propose a method to select latency-aware sets of check variables.

5.2.1 Latency Improvement with Selecting Check Variables

From the viewpoint of the SRS condition (refer to Condition 2 in Chap. 4.3), as succeeding operations are scheduled in far later steps from the earliest preceding operations in the same cone, it becomes difficult for the successors to satisfy the SRS condition. For example, in Fig. 5.1, a later successor o_b in $c_x^{(2)}$ has less opportunity in terms of SRS than its predecessor o_a in the same cone. In order to make o_b fulfill the condition easier, o_b and its successors should be separated from their predecessors. Such separation is realized by choosing the output of o_a as a new check variable, and as a result, the cone $c_x^{(2)}$ is

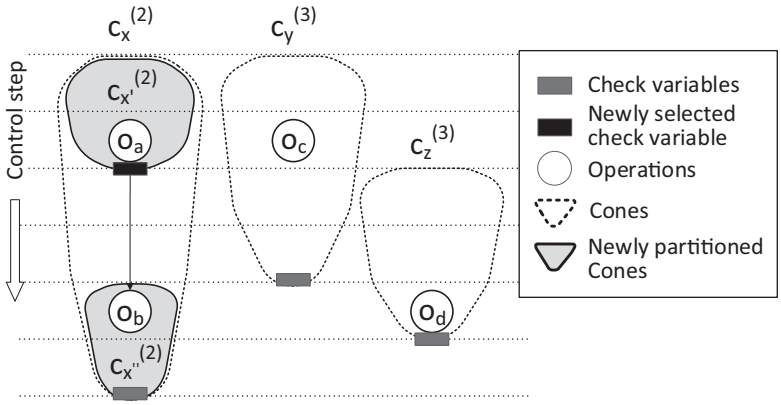

Figure 5.1: An example of cone partition in a scheduled data flow graph; $c_x^{(2)}$ can be partitioned into $c_{x'}^{(2)}$ and $c_{x''}^{(2)}$ to increase the possibility of SRS.

Algorithm 3 Latency-aware check variable selection algorithm

Require: *dfg_org* ← an original input data flow graph
1: *cones* ← **make_initial_cone_partition**(*dfg_org*);
2: *dfg_tri* ← **triplicate_data_flow_graph**(*dfg_org*, *cones*);
3: *schedule* ← **perform_operation_scheduling**(*dfg_tri*);
4: **while** (1) **do**
5: **if** (every cone consists of only a single operation) **then**
6: break;
7: **end if**
8: *op_pair* ← **find_operation_pair**(*schedule*);
9: *chk_var* ← **select_new_check_variable**(*op_pair*);
10: *cones* ← **divide_cone_into_two**(*cones*, *chk_var*);
11: *dfg_tri* ← **reconstruct_data_flow_graph**(*dfg_tri*, *cones*);
12: *schedule* ← **perform_operation_scheduling**(*dfg_tri*);
13: *results* ← **save_scheduling_result**(*schedule*, *cones*);
14: **end while**
15: **take_the_best_solution_in_latency**(*results*);

divided into two cones $c_{x'}^{(2)}$ and $c_{x''}^{(2)}$. Then, not only o_a and o_c but also o_b and o_d can share resources speculatively. Consequently, choosing a new check variable allows the chance of SRS to improve.

However, selecting new check variables may increase time overhead. In general, there is a trade-off between latency and the number of comparison-operations. Finding an operation pair and separating the two operations in the pair into two cones broaden opportunities for SRS. As a result, latency can be reduced by SRS. On the other hand, if the number of comparison-operations increases owing to choosing more new check variables, then latency may be extended due to the execution of inserted comparison-operations. Therefore, an operation pair, which consists of an operation and its immediate successor, and has a large difference in control steps between them, should be selected preferentially.

5.2.2 Check Variable Selection Algorithm

In order to overcome the strict constraint on SRS and obtain optimized solutions in latency, the author proposes Algorithm 3 [87]. The following notations are employed to describe it.

- *dfg_org*: an original input data flow graph (given)
- *dfg_tri*: a triplicate data flow graph
- *cones*: partitioned cone information
- *schedule*: an operation schedule
- *op_pair*: an operation pair for cone-partitioning
- *chk_var*: a new check variable
- *results*: saved results at the end of each iteration

Algorithm 3 shows the proposed check variable selection algorithm. First, the initial cone-partitioning is performed with an original data flow graph so that every internal operation in each cone has a single output and the resultant data flow graph is composed of maximal cones. In terms of check variables, the set of check variables inducing the initial cone-partition can be defined as the smallest set of check variables. Next, the triplication and the initial operation scheduling are performed. In the while loop, an operation and its immediate successor which belong to the same cone and have the biggest difference in control steps are found. After that, the output of the operation is selected as a new check variable. Next, the cone which includes the new variable is divided into two cones based on the variable. Then, the data flow graph is reconstructed and an operation scheduling is performed again. The operation schedule and the cone-partition are saved at the end of every iteration. This loop body is executed repeatedly until every cone consists of only a single operation. Finally, the best cone-partition in terms of latency is obtained from the saved results.

The main idea of this algorithm is iteration. During this process, cones are partitioned smaller and smaller. Since every internal operation in each cone has only one output, a new check variable divides its cone into exactly two. This iteration does not guarantee that latency always decreases monotonically. Thus, after the process, the best solution among saved ones is chosen.

5.3 Experimental Results

The proposed check variable selection algorithm have been implemented as a computer program, and applied it to G.722 ADPCM Decoder (ADPCM_D), four-by-four matrix inversion (4x4INV), four-by-four matrix multiplication (4x4MUL), 16-point fast Fourier transform (16FFT), inverse discrete cosine transform with column-wise decomposition (IDCT-c), autoregressive filter (ARF), and fifth-order elliptic wave digital filter (5EWDF), and 16-point FIR filter (16FIRF). Three types of functional units, which are multipliers, ALUs and comparators, are employed as allocated resources in datapath synthesis experiments.

To evaluate the effectiveness of the proposed algorithm in latency, three schemes have been implemented in the above applications. The three implementations are: (I) a C-R based conventional method (Conventional), (II) the C-R and SRS based preliminary scheme (SRS only) and (III) the combination of the preliminary scheme and the proposed algorithm (SRS+Auto_Cone). For (I)(Convent.) and (II)(SRS only), the smallest set of check variables are used as a given set of check variables. Note that the smallest set

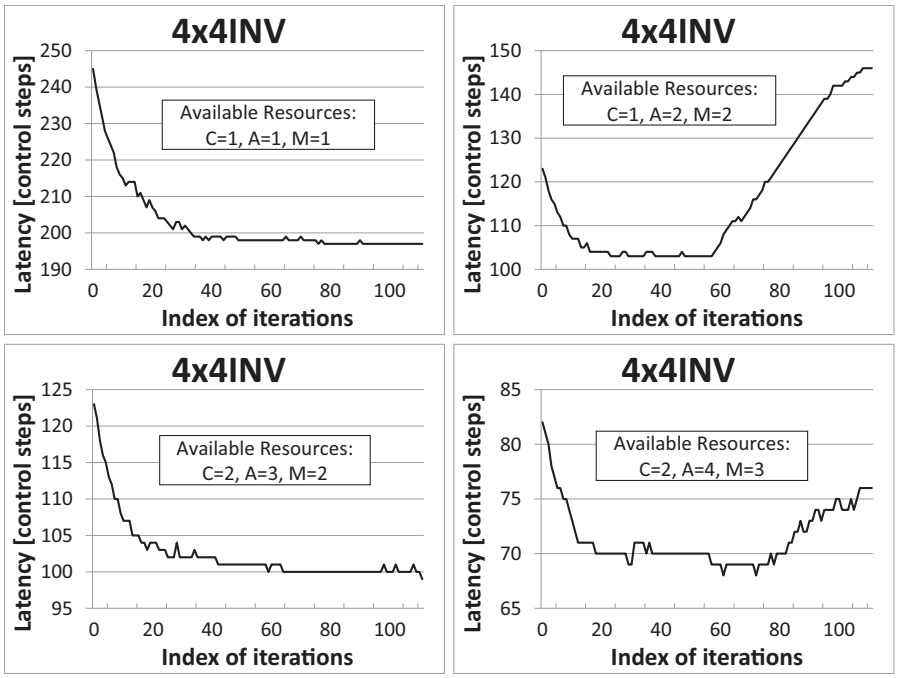

Figure 5.2: Four exploration examples to find the best latency-aware selections of check variables under different choices of allocated resources (4x4INV)

of check variables is also used as the initial choice of check variables in Algorithm 3 for (III)(SRS+Auto_Cone).

Fig. 5.2, 5.3 and 5.4 show experimental results. Fig. 5.2 and 5.3 indicate four exploration examples in case of 4x4INV and 16FFT, respectively, to find the best latency-aware set of check variables under different choices of allocated resources. It is found that the minimum latency in each case is acquired at a different index of iterations. Fig. 5.4 shows that computation algorithms which possess higher parallelism (ADPCM_D, 4x4INV and IDCT-c) achieve significant improvements in latency compared with (II)(SRS only). In addition, even though others (5EWDF, ARF and 16FIRF) have lower parallelism, they show relatively large improvements under a small number of allocated resources. 4x4MUL and 16FFT possess higher parallelism, however, they obtain limited improvements under every resource constraint comparing with (II)(SRS only). The reason is considered as follows; The partitioned cones induced by the smallest set of check variables of several benchmarks (for example, 4x4MUL and 16FFT) with (II)(SRS only) have similar topologies. Such similarity of the cones helps to share more resources speculatively and improve latency more. That is, for these applications, the smallest set of check variables may become the best selection of check variables with a higher probability, and as a result, the latency improvement is not achieved by (III)(SRS+AutoCone). On the other hand, it can be thought that, for other applications unlike 4x4MUL or 16FFT, the similarity of partitioned cone topologies will be improved by adding new check variables to the smallest set of check variables, and a certain degree of latency improvement is achieved.

The experimental results suggest that the improved solutions are found by the proposed algorithm and the best latency-aware check variable selection varies according to available resources.

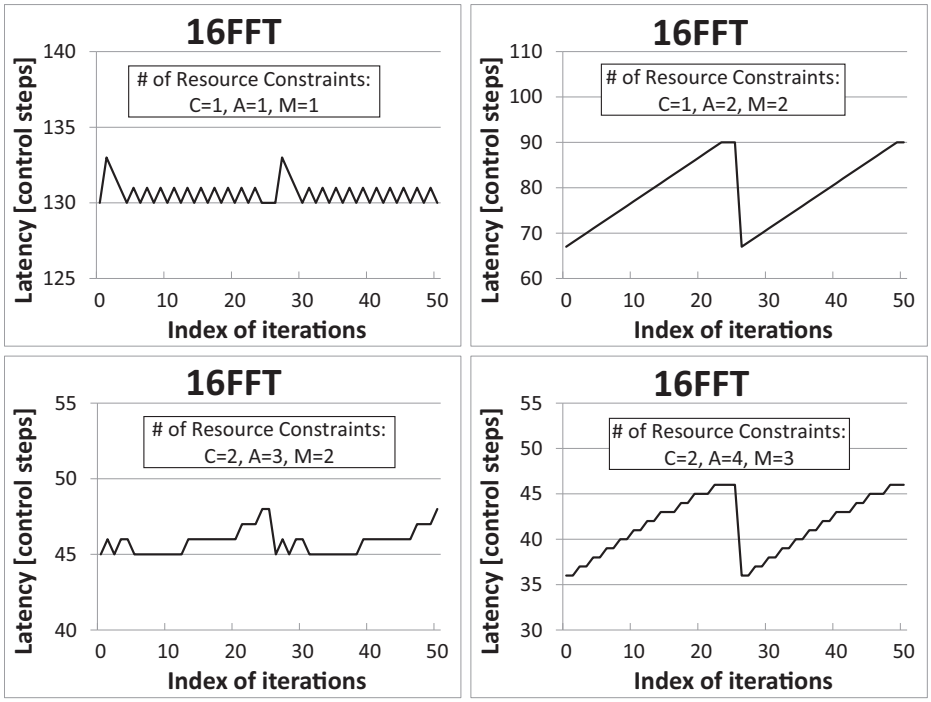

Figure 5.3: Four exploration examples to find the best latency-aware selections of check variables under different choices of allocated resources (16FFT)

5.4 Summary

Datapath circuits designed by the proposed method tolerate single soft-error-induced multiple-component errors. In the preliminary study, the author proposed an efficient resource management called speculative resource sharing (SRS). To overwhelm its strict condition, the author proposes a check variable choosing algorithm which enlarges opportunities for SRS. Experimental results revealed that the proposed method achieves improvements in latency compared to a conventional check-and-retry mechanism. The proposed method is especially effective when the input computation algorithm possesses a high degree of parallelism, and the number of allocated resources is relatively small.

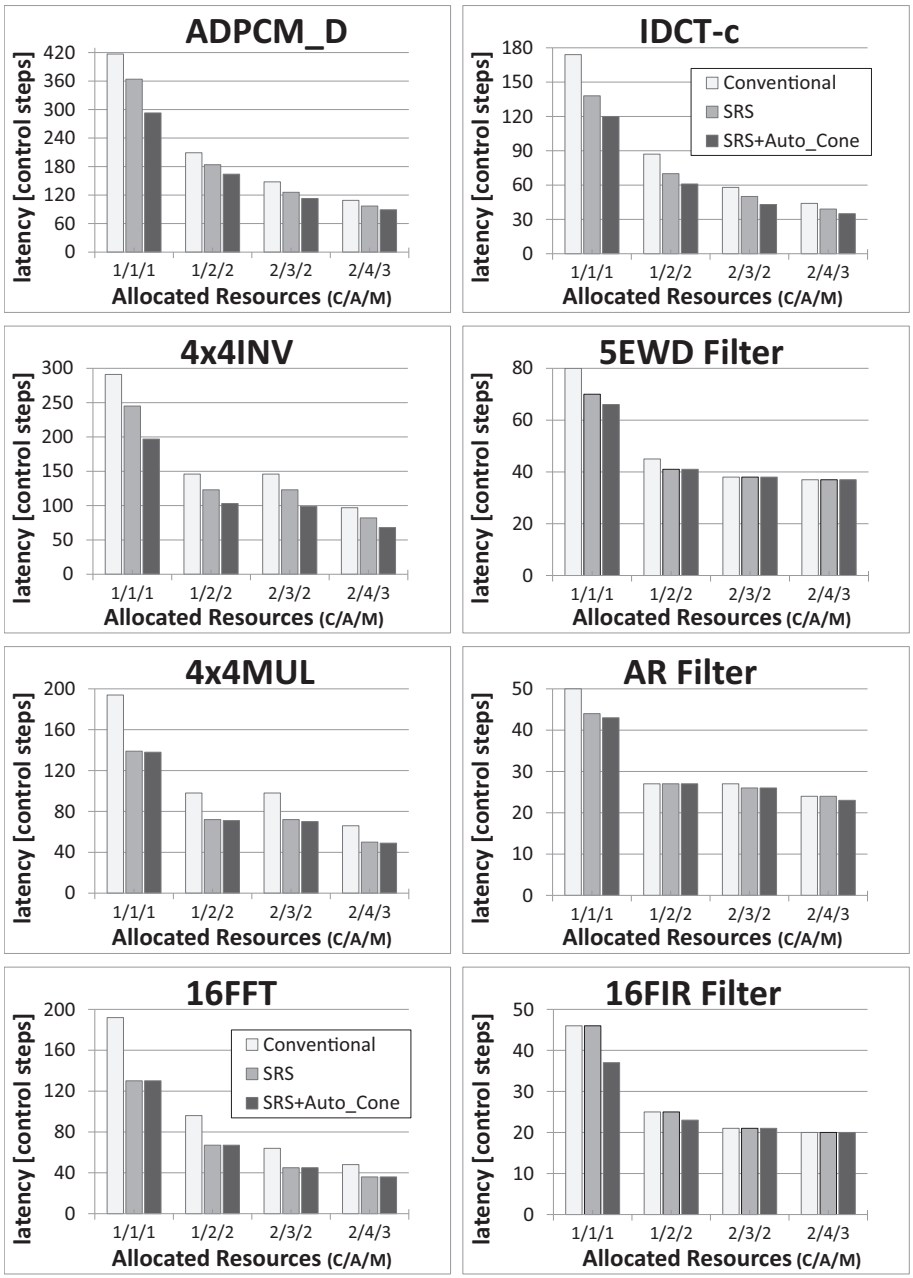

Figure 5.4: Experimental results in eight computational algorithms. Each column corresponds to a different choice of allocated resources (C: Comparator, A: ALU, M: Multiplier).

Chapter 6

Adjacency Constraint between Circuit Components

6.1 Motivation

In the study [39], Zhang et al. examined dependency of MCUs on flip-flop cell distance and well-contact density. According to their report, MCU rates are exponentially decreased by the distance between flip-flops. Also, device level approaches such as alternative placement of flip-flips and high-density well-contact arrays can reduce MCU rates. Previously, the author proposed a method to synthesize single soft-error tolerant application-specific datapaths via high-level synthesis [27]. To handle multiple component error, a C-R mechanism is used to detect and correct error due to a single soft-error. In addition, in order to mitigate time/spatial overhead, the author introduced speculative resource sharing (SRS) which is a special resource sharing between second copies and third copies, and proposed a heuristic scheduling algorithm to increase the chance of SRS. The two novel contributions of the proposed method in this chapter are (1) adjacency constraint to mitigate time overhead further and to reduce excessively applied fault-tolerance; (2) an ILP based resource binding approach to obtain optimized solution in the number of component pairs on which adjacency constraint is imposed.

6.2 Adjacency Constraint on Soft-Error Tolerant Datapaths under SRS

6.2.1 Modified Fault Model Considering Localities of Soft-Error

As mentioned in Section 4.2.1, it is assumed that a single soft-error can cause multiple component error. Thus, it means that the scope of components which are affected by one single soft-error simultaneously is an entire chip. In order to reduce such excessively applied fault-tolerance and mitigate time overhead for the executions of retry-cones which are a disadvantage of C-R scheme, the author introduces a concept of localities of soft-error. If a single soft-error has a spatial and temporal boundary, and its influence is limited against multiple component error, several components can be executed at the same time. This concept is based on the following assumptions:

Assumption 2 *[Spread of time] The effect of a single soft-error lasts no longer than k control steps.*

Assumption 3 *[Spread of space] The effect of a single soft-error does not exceed a certain spatial boundary.*

If Condition 2 can be ignored under the above two assumptions, the constraint on SRS can be relaxed and more operation pairs can share resources speculatively, as a result, latency can be improved. On the other hand, if Condition 1 can be disregarded under Assumption 2 and 3, a majority-voting mechanism for error masking and correction instead of C-R scheme can be applied, and the three copies in every stage can be executed concurrently. Then, time overhead due to retry executions can be reduced and, in consequence, latency can be smaller. Nonetheless, the comparison-retry based error correction scheme is only focused on in this chapter.

6.2.2 Temporal and Spatial Adjacency Constraint (AC)

Let *dis* be a minimum spatial distance between two components which are not affected by a single soft-error at the same time. In the resource binding phase, every operation-operation pair which should not be affected together simultaneously needs to be bound to two distinct components having a certain spatial distance more than *dis* between the components. Otherwise, the two operations should be scheduled in the scheduling phase so that executions of the two operations have a certain temporal interval which is the time duration of soft-error or more. If two operations in a pair satisfy the following Condition 3 or 4 under Assumption 2 and 3, those operations will not be concurrently erroneous although multiple component error induced by a single soft-error occurred. In a similar way, temporal/spatial adjacency constraint (AC) can also be applied to variable-variable pairs and operation-variable pairs.

Condition 3 *[Temporal adjacency constraint] After the execution of one operation (or the life time of one variable) is finished, the execution of the other operation (or the life time of the other variable) is started $k-1$ control steps later in case of k-cycle soft error.*

Condition 4 *[Spatial adjacency constraint] Two operations, two variables or one operation and one variable in a pair are bound to distinct components which keep a spatial distance dis at least.*

Condition 3 should be considered in the scheduling phase. On the other hand, Condition 4 should be considered in the resource binding phase.

6.2.3 Scheduling and Resource Binding Condition under Adjacency Constraint

The following three equations describe the requirements in scheduling and resource binding phases when AC is applied. If the two operations o_i and $o_{i'}$ which should not be erroneous simultaneously satisfy the following equation (6.1), those operations must be bound to keep Condition 3 or 4.

$$\sigma_s(o_i) \leq \sigma_e(o_{i'}) + (k-1) \ \wedge \ \sigma_s(o_{i'}) \leq \sigma_e(o_i) + (k-1) \tag{6.1}$$

If the two variables h_j and $h_{j'}$ which should not be corrupt at the same time keep the following equation (6.2), those variables must be bound to fulfill Condition 3 or 4.

$$\sigma_s(h_j) \leq \sigma_e(h_{j'}) + (k-1) \ \wedge \ \sigma_s(h_{j'}) \leq \sigma_e(h_j) + (k-1) \tag{6.2}$$

If an operator o_i and a variable h_j which should not be erroneous concurrently satisfy the following equation (6.3), o_i and h_j must be bound to fulfill Condition 3 or 4.

$$\sigma_s(o_i) \leq \sigma_e(h_j) + (k-1) \ \wedge \ \sigma_s(h_j) \leq \sigma_e(o_i) + (k-1) \tag{6.3}$$

6.3 ILP Formulation for Resource Binding

6.3.1 Definitions of Variables and Constants

- $u_{i\alpha}$ is 1 if an operation o_i is executed on a functional unit α, 0 otherwise.
- x_{jp} is 1 if a variable h_j is stored in a standard register p, 0 otherwise.
- y_{jr} is 1 if a variable h_j is stored in a multi-bit soft-error tolerant register r, 0 otherwise.
- $ac_{gg'}$ is 1 if adjacency constraint is imposed on component g and g', 0 otherwise.
- S is the total number of component pairs on which adjacency constraint is actually imposed.
- K_{ctyp} represents the available number of component type *ctyp*. Component type M, C, SR, FM and FO represent operators, comparators, standard registers, multi-bit soft-error tolerant registers and 1-bit soft-error tolerant registers, respectively.

6.3.2 ILP Formulation

It is assumed that every operation is a single-cycle operation and is bound to a single-type functional unit to present the fundamentals of proposed method concisely. However, the formulas in this section can easily be expanded for multi-cycle operations and multi-type functional units. The next sections will present ILP formulas which are related to the main proposal, however, the general conditions for high-level synthesis will be omitted through this chapter.

6.3.3 General Resource Binding Constraints

Every internal operation result in every cone or every output of every second-cone, which is represented by h_j, can be bound to a standard register. Such variables can be also bound to multi-bit soft-error registers. However, one variable should be bound to only one register.

$$\sum_{p=1}^{K_{SR}} x_{jp} + \sum_{r=1}^{K_{FM}} y_{jr} = 1 \tag{6.4}$$

6.3.4 Constraints under Speculative Resource Sharing

If o_i and $o_{i'}$ are a pair which can share a resource speculatively, the two operations should be bound to the same operator.

$$u_{i\alpha} - u_{i'\alpha} = 0 \tag{6.5}$$

6.3.5 Constraints under Adjacency Constraint

- If two operations o_i and $o_{i'}$ satisfy (6.1) and should not be erroneous at the same time, AC is imposed on between operator α and α'.

$$u_{i\alpha} + u_{i'\alpha'} - ac_{\alpha\alpha'} \leq 1 \tag{6.6}$$

  Similar ones can be formulated with variable-variable pairs under (6.2) and operation-variable pairs under (6.3).

- In order to minimize the number of pairs on which adjacency constraint is actually imposed, let us introduce a variable S which follows the following equation (6.7) and the objective of resource binding is to minimize S.

$$\sum_{\text{All component pairs under AC}} ac_{gg'} \leq S \tag{6.7}$$

6.4 Adjacency Constraint on Soft-Error Tolerant Controllers

In order to implement complete fault-tolerance on a LSI against multi-component soft-errors, it is needed to consider fault-tolerance on not only its datapath but also its controller.

6.4.1 Preliminaries

General Finite State Machine Synthesis

The classical finite state machine (FSM) synthesis consists of three main phases [88]: i) state assignment, ii) Boolean equation generation corresponding to the binary state codes, the state transit functions and the output functions, and iii) logic structure generation such as logic minimization, factorization and technology mapping.

In the synthesis of controllers under high-level specifications such as control flow graphs and state graphs, state assignment is an important phase [89]. Furthermore, the state assignment phase consists of two steps [88]. The first step is the recognition of situations in the controller specification. Each recognized situation is associated an "adjacency constraint" [1] on a group of states. The next one is the embedding in the hypercube, or binary code assignment, which is an attempt to satisfy as many AC-SAs as possible.

[1]Note that adjacency constraints in the state assignment phase are totally different from the temporal and spatial adjacency constraint on circuit components, which is the main proposal of this chapter. In order to distinguish them, adjacency constraints for state assignment is hereafter described as 'AC-SA'.

6.4.2 Conventional Methods for Fault-Tolerant Controller Design

Single Independent Decoder Architecture Design

The single independent decoder (SID) architecture is originally presented by Armstrong [90]. As the name SID suggests, the architecture assumes a single fault model. With this concept, to achieve single fault tolerance in the next-state logic and state registers, a single error correction (SEC) code with Hamming distance 3 in minimum must be used in the state assignment phase. Then, to guarantee a single fault tolerance in the next-state logic, the logic is locally optimized, and the factorization and mapping phases prevent any two next-state functions from sharing logic. After that, the fault tolerance is achieved by an independent logic block which is connected to the state register, so that an erroneous state code is rectified before the erroneous code corrupts the next-state and output functions [88].

Other Methods

The study by Lakshminarayana et al. [59] presented a behavioral synthesis method of fault secure datapaths as well as controllers under a single fault model. After datapath synthesis procedure, the following constraints are imposed on the logic synthesis phase to implement fault security [59][91]: i) all valid controller states have the same parity, and ii) the primary outputs and next state outputs of the controller are implemented by disjoint partitions of logic. The primary outputs also have a parity code which is an output of the controller. The present-state and the controller outputs are sent to a totally self-checking (TSC) parity checker. A fault in any logic partition, a single fault in any stage flip-flop or a single fault in the parity checker itself is detected by the checker.

Leveugle [92] modified the study [90] and addressed a synthesis method of single fault tolerant FSMs based on SEC codes. The proposed FSM considers single fault tolerance (SFT) in either the next state logic or the stage register. The main proposal of the method is a new state assignment algorithm for SFT-FSMs in the embedding step. They implemented the proposed method to two different architectures, which are SID [90] and distributed error correction (DEC) [93]. The experimental results showed that the proposed algorithm for DEC achieved 14.27% of improvement in area in average. Although the implementation of the proposed algorithm for SID obtained less improvement than the case of DEC, the results showed that the proposed algorithm is also effective for SID.

Iwagaki et al. [94] proposed a multi-cycle transient fault tolerant controller design having area-efficiency using SID, which is based on the works [90][92]. They showed that the proposed method with SID can successfully realize multi-cycle transient fault tolerance under a single fault model. Also, to share logic functions among as many control signals as possible, they focused on implementation of control signal generator, which is a component of the controller sending control signals to the datapath in a LSI chip, and logic optimization of the generator.

6.4.3 A Controller Design Proposal against Multi-Component Soft-Error under AC

Following to conventional methods, the author also employs SID to implement multiple component soft-error tolerance into controllers.

AC Implementation in SID

In order to tolerate soft-errors spreading to multiple components, SID-based controllers must fulfill the following conditions: i) Soft-error tolerant registers are employed as state registers. ii) AC are imposed on each logic cone in the next state logic part and each logic cone in the control signal generator. iii) Moreover, AC are imposed on among the next stage logic part, control signal generator and decoder.

6.5 Experimental Results

The author has implemented the proposed method which is mentioned in the previous sections as a computer program, and applied it to 16-point fast Fourier transform (16FFT), 8-point inverse discrete cosine transform (8IDCT), 16-point FIR filter (16FIRF), autoregressive filter (ARF), fifth-order elliptic wave digital filter (5EWDF), and inverse discrete cosine transform with column-wise decomposition (IDCT-c). Three types of functional units, which are multipliers, ALUs and comparators, and three types of registers, which are standard registers, 1-bit soft-error tolerant registers and multi-bit soft-error tolerant registers, are employed as available resources in datapath synthesis experiments. In addition, the maximum duration of a single soft-error is set to one control step through the experiments ($k = 1$). It is also assumed that cone-partitioning is given in advance. In order to find exact solutions from ILP formulations, a solver Gurobi Optimizer 6.0.0 was performed on one node, which consists of two Intel Xeon E7-8837 (2.66GHz) processors and 128GB memory, in a massively parallel computer.

6.5.1 Performance Evaluation in Latency

To evaluate the effectiveness of the proposed method in latency, the three different methods are applied to the above six applications under four distinct resource constraints. The three different methods are: (a) a conventional C-R method, (b) the preliminary method SRS based on C-R scheme and (c) the combination of SRS and the new proposal AC based on C-R scheme. Figure 6.1 shows that the mixture of SRS and AC (SRS+AC) achieves further improvements in latency. However, some benchmark applications having symmetrical topology, for instance 16FFT and 8IDCT, accomplish minor improvements since the effectiveness of resource use is already maximized by SRS. Particularly, in symmetrical data flow graphs, it is easy to match operations in retry-cones to other operations in second-cones as speculatively resource sharable pairs since the shapes of cones are the same or quite similar. In addition, when there is no difference in latency between a conventional method (a) and the preliminary method (b), AC does not show even minor improvement because it is difficult to decrease latency below critical path length although enough number of resources are available.

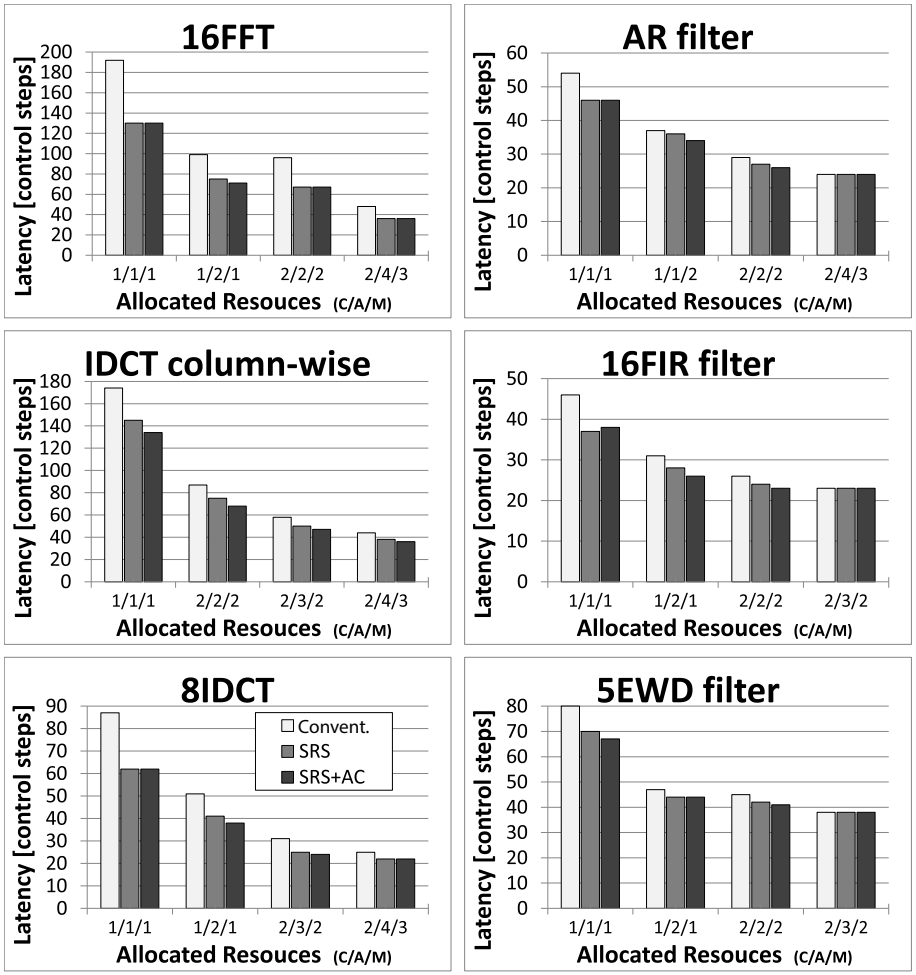

Figure 6.1: Experimental results in six computational algorithms. Each column corresponds to a different choice of allocated resources (C: Comparator, A: ALU, M: Multiplier).

6.5.2 Area Estimation

In order to examine hardware overhead due to triplication and the introduction of adjacency constraint, chip areas are estimated by simulated annealing based on the sequence-pair approach. Using the number of resources and the operation schedule, which are the inputs and outputs of scheduling phase, the formulated resource binding problems are solved by a solver. The objective of the problems is to minimize the number of component pairs on which AC is imposed. 1-bit soft-error tolerant registers are set as the smallest components which have the unit area and let the bit width of multi-bit registers be 16. According to [21], the size of a soft-error tolerant register is approximately three times bigger than a standard register. Thus, it can be estimated that the area of a 16-bit standard register is $1 \times 16 \div 3 \approx 5.3$ [a.u.]. To obtain area information of other components, [95] is referred to and all the size data are normalized (see Table 6.1). Multiplexers are not considered for the sake of simplicity. The experimental results (Table 6.2) show that the proposed method keeps almost the same level of performance in chip size compared with the preliminary method SRS while *dis* is small. However, as the number of components on which AC is imposed and *dis* are larger, then the increment of chip size is not

Table 6.1: Specifications of Function Units and Registers

Component Types		Area [a.u.]
Register	1-bit Soft-Error Tolerant	1
	16-bit Standard	5.3
	16-bit Soft-Error Tolerant	16
Operator	Comparator	2.3
	ALU	7.6
	Multiplier	42.4

concealed by placement of components any longer.

6.6 Summary

A datapath circuit designed by the proposed method tolerates multi-component and multi-cycle error caused by a single soft-error. Based on the preliminary method speculative resource sharing (SRS), this chapter introduces spatial/temporal adjacency constraint (AC) between components which prevents an excessive fault-tolerance and increases the opportunities of SRS. Furthermore, the author proposes an ILP based resource binding under AC to minimize the number of components on which AC is actually imposed after resource binding phase. From the results of datapath synthesis experiments, it is found that the combined approach of SRS and AC achieves additional improvements in latency, while keeping the comparable levels in chip areas compared with datapaths without AC. Specifically, it have been found that the proposed method is more effective when an input computation algorithm possesses relatively difficult topology to find speculatively resource sharable pairs, and the spatial adjacency constraint dis is relatively small.

Table 6.2: Experimental Results in Area Estimation

Compu. Algo-rithms	# of Resources		# of AC		Estimated Chip Area [a.u.]			
	Func. units†	Regi-sters††	Res. Binding before	Res. Binding after	SRS	SRS+AC		
						dis=8	16	32
16FFT	1/1/1	2/25/2	0	0	496.4	500.1	499.7	501.2
	1/2/1	7/31/3	95	1	654.1	642.0	640.0	660.2
	2/2/2	3/27/4	0	0	601.4	600.0	598.6	602.1
	2/4/3	6/32/4	10	1	769.4	769.3	775.7	769.1
IDCT-c	1/1/1	10/14/7	102	13	400.2	406.3	455.0	682.8
	2/2/2	9/14/7	78	9	436.0	437.6	435.5	497.3
	2/3/2	8/16/6	69	5	450.2	446.1	606.7	598.2
	2/4/3	8/16/6	46	3	511.8	510.3	516.9	537.7
8IDCT	1/1/1	4/13/3	2	2	315.4	309.9	312.1	323.1
	1/2/1	7/14/3	23	8	357.0	363.3	455.0	649.8
	2/3/2	5/14/4	12	3	414.9	412.1	416.8	417.7
	2/4/3	6/15/4	0	0	497.3	502.2	496.0	500.4
ARF	1/1/1	3/27/3	0	0	555.1	542.6	550.2	540.2
	1/1/2	10/27/2	34	14	655.9	692.5	725.8	1064.5
	2/2/2	4/27/2	4	1	601.9	602.1	603.1	640.3
	2/4/3	5/29/3	0	0	695.2	702.8	697.4	694.1
16FIRF	1/1/1	2/17/4	2	1	370.2	370.2	375.0	407.3
	1/2/1	9/17/2	41	15	427.5	434.1	491.5	712.5
	2/2/2	3/18/2	4	1	446.9	450.2	448.0	457.0
	2/3/2	7/18/3	24	9	474.8	474.1	477.3	573.2
5EWDF	1/1/1	7/15/5	18	4	370.2	378.6	418.1	483.9
	1/2/1	4/13/3	12	1	326.6	326.7	330.1	335.0
	2/2/2	5/13/3	4	1	378.6	370.5	375.0	381.1
	2/3/2	3/13/3	8	2	366.9	370.4	384.7	381.7

† comparator / ALU / multiplier

†† multi-bit soft-error tolerant / 1-bit soft-error tolerant / standard register

Chapter 7

Mixed Error Correction Scheme and Its Design Optimization

7.1 Motivation

In the previous chapters, the author introduced speculative resource sharing between two distinct copies (Chap. 4) and adjacency constraint between components to mitigate hardware/time overhead (Chap. 6). Based on the two preliminary proposals, this chapter proposes a hybrid approach of comparison-retry and majority-voting error correction schemes to combine high resource efficiency and small latency which are the advantages of the two schemes. In addition, the author proposes an optimized combination algorithm of the two fault-tolerant mechanisms to achieve minimum solutions in latency.

7.2 Combination of Two Error Correction Scheme under AC and SRS

This section proposes a combination of two distinct error correction mechanisms and a heuristic algorithm to find a latency-optimized combination of the two mechanisms.

7.2.1 Introduction of a Majority-Voting Based Error Correction Scheme and Modified Fault Model

An advantage of comparison-retry (C-R) based error correction mechanisms is that the mechanisms can reduce power consumption contributed by idling of retry parts in case no error is detected. Nevertheless, the executions of retry parts cause time overhead, which is a drawback of the mechanisms. To mitigate such time overhead, the author introduces a majority-voting (M-V) based error correction scheme. In general, the following condition should be satisfied to implement M-V mechanisms.

Condition 5 *No more than one output among the outputs of three copies in triplication is erroneous.*

If every pair of two operations in two distinct cones belonging to the same stage satisfies the adjacency constraint, that is, Condition 3 or 4, then Condition 5 is satisfied and the three cones in the stage can be executed at the same time. After that,

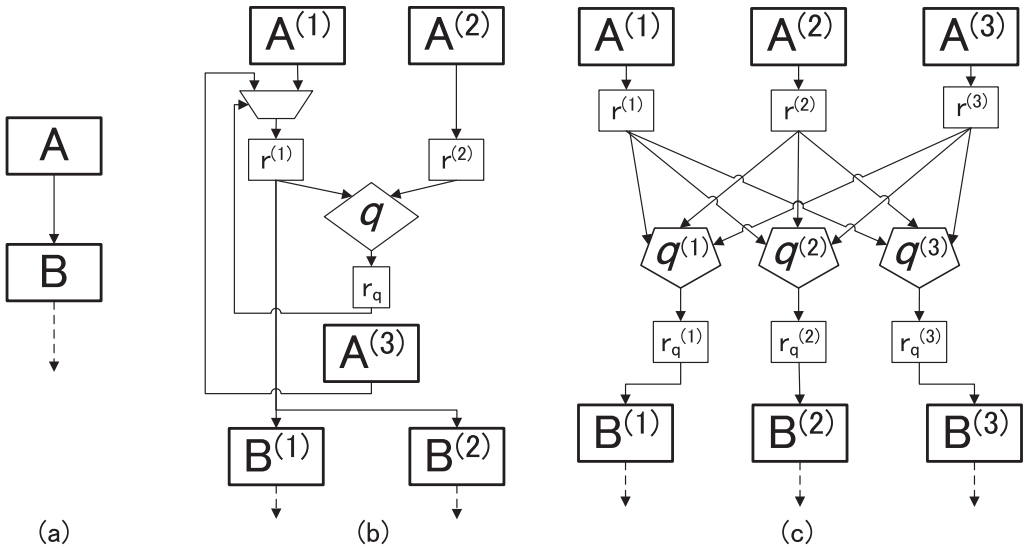

Figure 7.1: A sketch of our strategy for fault tolerance; (a) is an original computation algorithm. (b) shows a triplicate algorithm with comparison-retry mechanism, (c) depicts a triplicate algorithm with majority-voting mechanism.

three vote-operations are performed to mask at most one corrupt output data among the three outputs. Then each voting result is sent to each succeeding computation block (Fig.7.1(c)).

Nonetheless, M-V mechanisms also have a disadvantage that third copies should be always executed while third copies in C-R schemes, namely retry-cones, need not be executed as long as no error has occurred. Moreover, M-V mechanisms require more hardware resources although datapaths to which those schemes are implemented can achieve small latency. In order to merge the advantages of both C-R schemes and M-V schemes, the author decided to combine the two different types of error correction schemes in a computation algorithm. To form a modified resultant algorithm $(\tilde{G}, \tilde{D_P})$ with the combination of two schemes, first of all, the given dependence graph G is triplicated. Then operations for error correction, which are comparison- and vote-operations, and dependencies related to those operations are inserted.

7.2.2 Combination of Two Distinct Fault-Tolerant Mechanisms

Figure 7.2 illustrates an example of a hybrid of M-V and C-R mechanisms. Selecting between the C-R scheme (Fig.7.1(b)) and the M-V scheme (Fig.7.1(c)) in each stage is an important task for datapath optimization. To obtain optimized solutions in latency under resource constraints, the author proposes a heuristic search algorithm which finds an optimized combination of the two error correction schemes in latency. Algorithm 4 shows the proposed algorithm. The following notations are employed to describe the algorithm.

- dfg_org : the original input computation algorithm
- dfg_tar : a reconstructed data flow graph
- op_schdl : an operation schedule

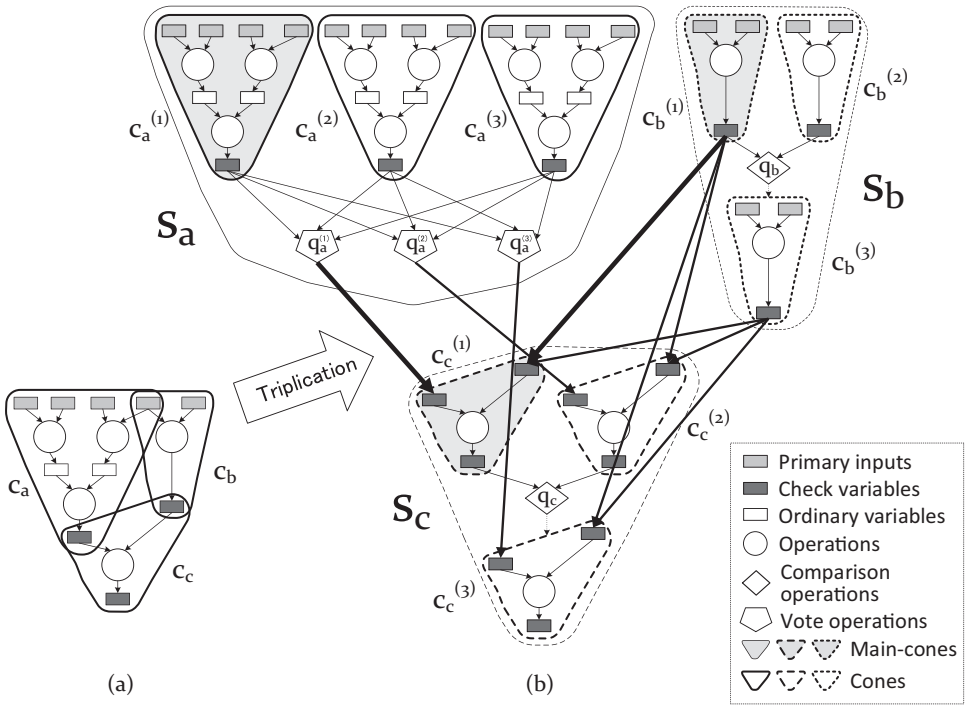

Figure 7.2: Stage s_a consists of main-cone $c_a^{(1)}$, second-cone $c_a^{(2)}$ and retry-cone $c_a^{(3)}$; (a) An example of an original computation algorithm (b) An example of cone-partitioned triplicate algorithm. The M-V mechanism is implemented in s_a and the C-R mechanism is implemented in s_b and s_c.

- *sorted_schdl* : a sorted operation schedule in ascending order of the number of scheduled operations in each control step
- *stage_num* : a selected stage to be changed into the M-V scheme

In the search algorithm (Algorithm 4), the initial scheduling is performed with the initial data flow graph so that every stage is based on the C-R scheme. In the while loop, the operation schedule is sorted to find the most inefficient use of resources. Then, in the top of the sorted schedule, one operation whose stage is based on the C-R scheme is chosen and the stage of the selected operation is converted to the M-V scheme. This reconstruction is performed in one stage at one loop. If the M-V scheme is implemented in all stages, the while loop is terminated. Finally, the best solution in latency can be obtained from the saved operation schedules.

The reason for alignment in ascending order in Step 4 is as follows: if a small number of operations are scheduled in a control step, it means that resources which do not execute the scheduled operations are in an idle state in the control step. If the error correction scheme of the stages of the operations in the control step is changed into M-V schemes, then more operations can be executed concurrently and remaining resources are occupied by those operations. As a result, latency can be improved.

Algorithm 4 The proposed algorithm to find a latency-optimized combination of two different error correction schemes

Require: $dfg\_org \leftarrow$ the original input computation algorithm

```
dfg_tar ← construct_initial_dfg_with_CV(dfg_org);
op_schdl ← perform_scheduling(dfg_tar);
while (1) do
    sorted_schdl ← sort_op_schedule_asc(op_schdl);
    stage_num ← select_one_stage_with_CV(sorted_schdl);
    if (the M-V scheme is implemented in all stages) then
        break;
    end if
    dfg_tar ← reconstruct_stage_with_MV(stage_num, dfg_tar);
    op_schdl ← perform_scheduling(dfg_tar);
    backup_if_best_op_schedule(op_schdl);
end while
```

7.2.3 Speculative Resource Sharing between Two Different Types of Error Correction Scheme under AC

When a stage m is based on the M-V mechanism and a stage n is based on the C-R mechanism, operations in the retry-cone $c_n^{(3)}$ can share resources speculatively with operations in the second-cone $c_m^{(2)}$ or in the retry-cone $c_m^{(3)}$ under the SRS condition. Similar to Section 6.2.2, if Condition 3 or 4 is satisfied instead of the SRS condition, SRS can be applied to those operation pairs.

7.3 Experimental Results

The author has implemented the proposed method as a computer program, and applied it to four-by-four matrix inversion (4x4INV), 16-point fast Fourier transform (16FFT), inverse discrete cosine transform with column-wise decomposition (IDCT-c), fifth-order elliptic wave digital filter (5EWDF), autoregressive filter (ARF) and 16-point finite impulse response filter (16FIRF). Four types of functional units, namely multipliers, ALUs, comparators and voters were utilized as allocated resources in datapath synthesis experiments. In addition, it is assumed that the cone-partitioning is given in advance and the maximum duration of a single soft-error is set to one control step in the experiments ($k = 1$).

To evaluate the effectiveness of the proposed method in latency, the four different methods were applied to the above six applications under four distinct resource constraints. The four methods is: (i) a conventional C-R method (Conventional), (ii) the C-R based preliminary method with SRS (SRS only), (iii) the C-R based preliminary method with SRS under AC (SRS+AC), (iv) a hybrid of C-R and M-V schemes with SRS under AC (SRS+AC+Hyb). To obtain operation schedules, the previously proposed scheduling algorithm which aggressively finds speculatively resource sharable pairs is employed [27].

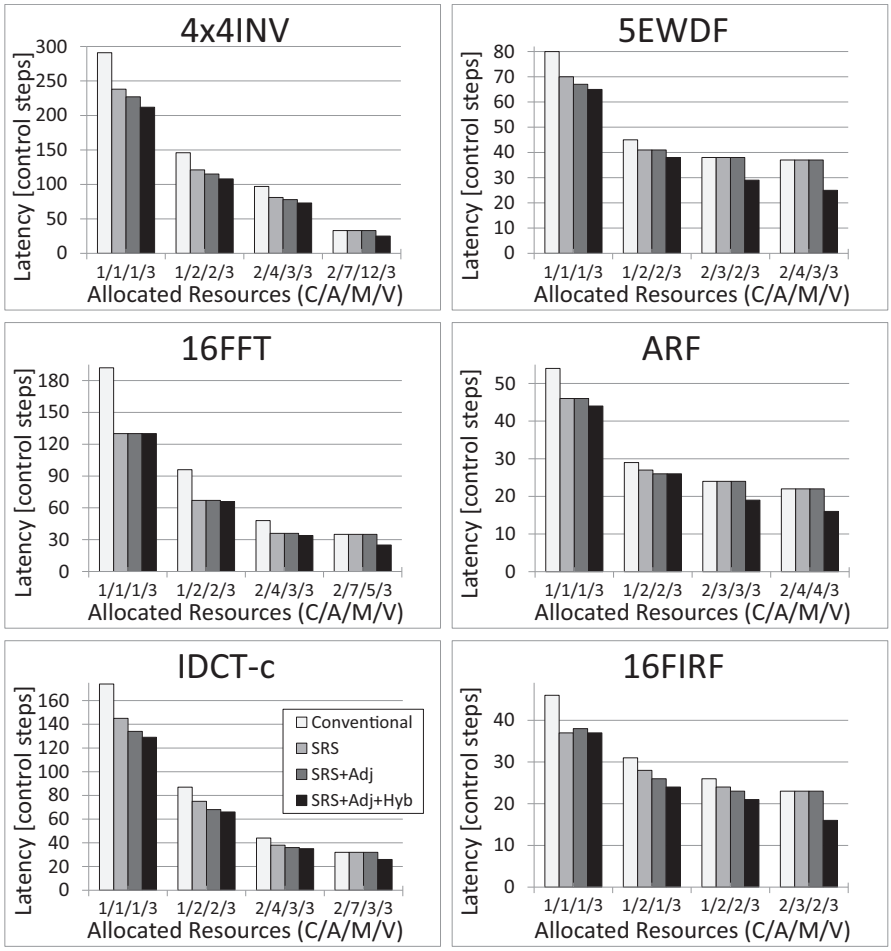

Figure 7.3: Experimental results in six computational algorithms. Each column corresponds to a different choice of allocated resources (C: Comparator, A: ALU, M: Multiplier, V: Voter).

7.3.1 Performance Evaluation in Latency

Figure 7.3 and Table 7.1 show the experimental results. In Table 7.1, the columns Mixtr. Ratio represent the mixture ratio of C-R scheme to M-V scheme in a benchmark application under a different choice of allocated resources. Also, the columns Improv. Rate indicate the latency improvement rate of the hybrid approach (SRS+AC+Hyb) compared with that of a conventional method (Conventional). The results reveal that the proposed method SRS+AC+Hyb improves latency more than the two preliminary methods, SRS and SRS+AC, in cases in which the number of allocated resources is small. On the other hand, when a relatively large number of resources is allocated, the two preliminary methods are unable to accomplish any enhancement, as can be seen in the case of 2/7/12/3 in 4x4INV and 2/4/3/3 in 5EWDF. Nevertheless, the proposed hybrid approach still achieves significant improvements in latency.

Table 7.2 and 7.3 show performance comparison in latency among conventional methods. The study [60] has two latency results in a benchmark application. The reason is that, in case of error, rollback takes time to recover from the error, and as a result, latency reaches the maximum. Otherwise, latency becomes the minimum value. However, time to execute voters and rollback mechanism in case of error is not counted in those

Table 7.1: Latency improvement rate

Application	Resource Constraint [C/A/M/V]	Mixtr. Ratio [C:V]	Improv. Rate [%]	Application	Resource Constraint [C/A/M/V]	Mixtr. Ratio [C:V]	Improv. Rate [%]
4x4INV	1/1/1/3	36:9	27.1	5EWDF	1/1/1/3	15:3	18.8
	1/2/2/3	37:8	26.0		1/2/2/3	11:7	15.6
	2/4/3/3	36:9	24.7		2/3/2/3	8:10	23.7
	2/7/12/3	36:9	24.2		2/4/3/3	5:13	32.4
16FFT	1/1/1/3	64:0	32.3	ARF	1/1/1/3	9:1	18.5
	1/2/2/3	63:1	31.3		1/2/2/3	10:0	10.3
	2/4/3/3	60:4	29.2		2/3/3/3	3:7	20.8
	2/7/5/3	36:28	28.6		2/4/4/3	1:9	27.3
IDCT-c	1/1/1/3	31:2	25.9	16FIRF	1/1/1/3	9:2	19.6
	1/2/2/3	30:3	24.1		1/2/1/3	6:5	22.6
	2/4/3/3	30:3	20.5		1/2/2/3	9:2	19.2
	2/7/3/3	16:17	18.8		2/3/2/3	8:3	30.4

Table 7.2: Achieved minimum latency comparison among conventional methods

Benchmark Applications	Study [60] CED+Rollback		Study [62] Hardware CED		Study [65] TMR		The proposed mothod SRS+AC+Hyb	
	C/A/M/V	Latency	C/A/M/V	Latency	C/A/M/V	Latency	C/A/M/V	Latency
ARF	0/4/4/8	10 (min)	No data		No data		2/4/4/3	16
	0/4/4/8	17 (max)						
16FIRF	0/4/4/4	9 (min)	NA/4/3/NA	9	No data		2/4/3/3	14
	0/4/4/4	16 (max)						
5EWDF	0/4/2/5	16 (min)	No data		3/6/3/1	10	2/7/4/3	23
	0/4/2/5	24 (max)						
8IDCT	No data		No data		3/8/5/1	13	2/7/3/3	17

NA: No specification is provided.

values. The study [62] considers only a fault-secure system. In other words, there is no mechanism to recover from errors because a fault-secure circuit provides only correct output or error signals. The results with hardware redundancy are shown in Table 7.2 and the results with time redundancy are shown in Table 7.3. The study [65] employs TMR for fault-tolerant capability. The proposed method SRS+AC+Hyb and SRS+AC are used in Table 7.2 and 7.3, respectively. In Table 7.2, the minimum latency is appeared from each method and each benchmark application, regardless of how many resources are used. On the other hand, the minimum latency under the minimum number of voters is shown from each method and each benchmark application in Table 7.3. Since the conditions and assumptions of each method in Table 7.2 and 7.3 are different from each other, simple comparison might not be practical. Nevertheless, the proposed method achieves comparable level of performance in latency compared with conventional methods.

Table 7.3: Achieved minimum latency comparison among conventional methods under the minimum number of voters

Benchmark Applications	Study [60] CED+Rollback		Study [62] Time CED		Study [65] TMR		The proposed mothod SRS+AC	
	C/A/M/V	Latency	C/A/M/V	Latency	C/A/M/V	Latency	C/A/M/V	Latency
ARF	0/4/5/2	10 (min)	No data		No data		2/4/3/0	24
	0/4/5/2	20 (max)						
16FIRF	0/4/4/1	9 (min)	NA/2/2/NA	15	No data		2/4/3/0	23
	0/4/4/1	18 (max)						
5EWDF	0/4/3/3	16 (min)	No data		3/6/3/1	10	2/4/3/0	37
	0/4/3/3	26 (max)						
8IDCT	No data		No data		3/8/5/1	13	2/4/3/0	22

NA: No specification is provided.

7.3.2 Area Estimation

Similar to Chap. 6.5.2, chip areas are estimated by simulated annealing based on the sequence-pair approach to examine hardware overhead due to triplication and the introduction of the proposed hybrid method. Using the number of resources and the operation schedule, which are the inputs and outputs of scheduling phase, the formulated resource binding problems are solved by a solver. The objective of the problems is to minimize the number of component pairs on which AC is imposed. It is assumed that all components are square in shape. Also, 1-bit soft-error tolerant registers are set as the smallest components which have the unit area and let the bit width of multi-bit registers be 16. It is assumed that the size of a soft-error tolerant register is about three times larger than a standard register [21]. Thus, it can be estimated that the area of a 16-bit standard register is $1 \times 16 \div 3 \approx 5.3$ [a.u.]. To implement multipliers and ALUs, Carry-save adder array based multiplier [96] and Han-Carlson Tree adder [96] are employed, respectively. A classical voter composed by NAND gates [97] is used for voting components (voters). Comparators are based on 4-bit magnitude comparator [98] and extended to 16-bit one. To obtain area information of a multiplier, an ALU, a voter and a comparator, NanGate 45nm cell library is used [99]. Specification of all components are normalized (see Table 7.4). For the sake of simplicity, multiplexers are excluded from the estimation.

Figure 7.4 shows the estimation results in area and it shows that the proposed hybrid method (SRS+AC+Hyb) keeps almost the same level of performance in chip size with the preliminary method (SRS only) while *dis* is small. However, similar to the results in Chap. 6.5.2, as the number of components on which AC is imposed and *dis* are larger, then the increment of chip size is not concealed by placement of components any longer. Figure 7.5 shows a relationship between chip area and latency, based on the results under $dis = 8$ in Figure 7.4. In each graph, the dotted line indicates the preliminary method (SRS only) and the solid line represents the proposed hybrid method (SRS+AC+Hyb). On the whole, each solid line is located in the left side of corresponding dotted line. It means that SRS+AC+Hyb achieves improvement in latency with combination of the two distinct fault-tolerant schemes. However, as latency becomes smaller, chip area signif-

Table 7.4: Specifications of Function Units and Registers

Component Types		Area [a.u.]
Register	1-bit Soft-Error Tolerant	1
	16-bit Standard	5.3
	16-bit Soft-Error Tolerant	16
Function Units	Comparator	3.1
	Voter	4.1
	ALU	12.4
	Multiplier	148.1

icantly becomes larger. It suggests that SRS+AC+Hyb achieves further improvement in latency to use more resources, however, it suffers from the increment of chip area as compensation.

Figure 7.6 shows placement results with 8IDCT benchmark application under 2 comparators, 7 ALUs, 3 multipliers, 3 voters. It depicts an overlap of different results based on the preliminary method (SRS only) and the proposed hybrid method (SRS+AC+Hyb) under dis =1, 2, 4, 8 and 16. Figure 7.7 shows the same results with different combination of comparison. The top left shows the case of $dis = 16$. As shown in the figure, there are huge dead space. The top right illustrates comparison between SRS only and SRS+AC+Hyb under $dis = 16$. There are small dead space in case of SRS only. The bottom shows comparison between SRS+AC+Hyb under dis =2 and 16. There are significant difference in chip area when dis is large. The bottom shows another comparison between SRS+AC+Hyb under dis =1, 2, 4 and 8. There is no huge difference among those results.

7.3.3 Reliability against Soft-Errors having spatial boundaries

In Chap. 6, the temporal effect and the spatial effect of a single soft-error are assumed as Assumption 2 and 3, respectively. To evaluate reliability of datapath circuits designed by the proposed method, it is assumed that a single soft-error affects within 1 cycle ($k = 1$, temporal spread) and a diameter dia (spatial spread).

Similar to the discussion in Sect. 4.6.2, let p_{dia} be the probability that a single soft-error occurs at a control step and, in consequence, it affects several components having a certain spatial boundary dia. Thus, Eq. (4.17) can be modified for an event that a single soft-error, having p_{dia}, occurs in a control step during latency cs, as follows:

$$P(n \text{ soft-errors with } dia \text{ in } [1,cs]) = {}_{cs}C_n \cdot p_{dia}^n \cdot (1 - p_{dia})^{(cs-n)} \tag{7.1}$$

In the same way, Eq. 4.16, 4.18 and 4.19 can be modified to replace p with p_{dia} as follows:

$$P(\text{no soft-error with } dia \text{ in } [1,cs]) = (1 - p_{dia})^{cs} \tag{7.2}$$

$$R(n \text{ soft-errors with } dia \text{ in } [1,cs]) = r_n \cdot p_{dia}^n \cdot (1 - p_{dia})^{(cs-n)} \tag{7.3}$$

R(all possible error patterns)

$$= \sum_{n=0}^{cs} R(n \text{ soft-errors with } dia \text{ in } [1,cs])$$

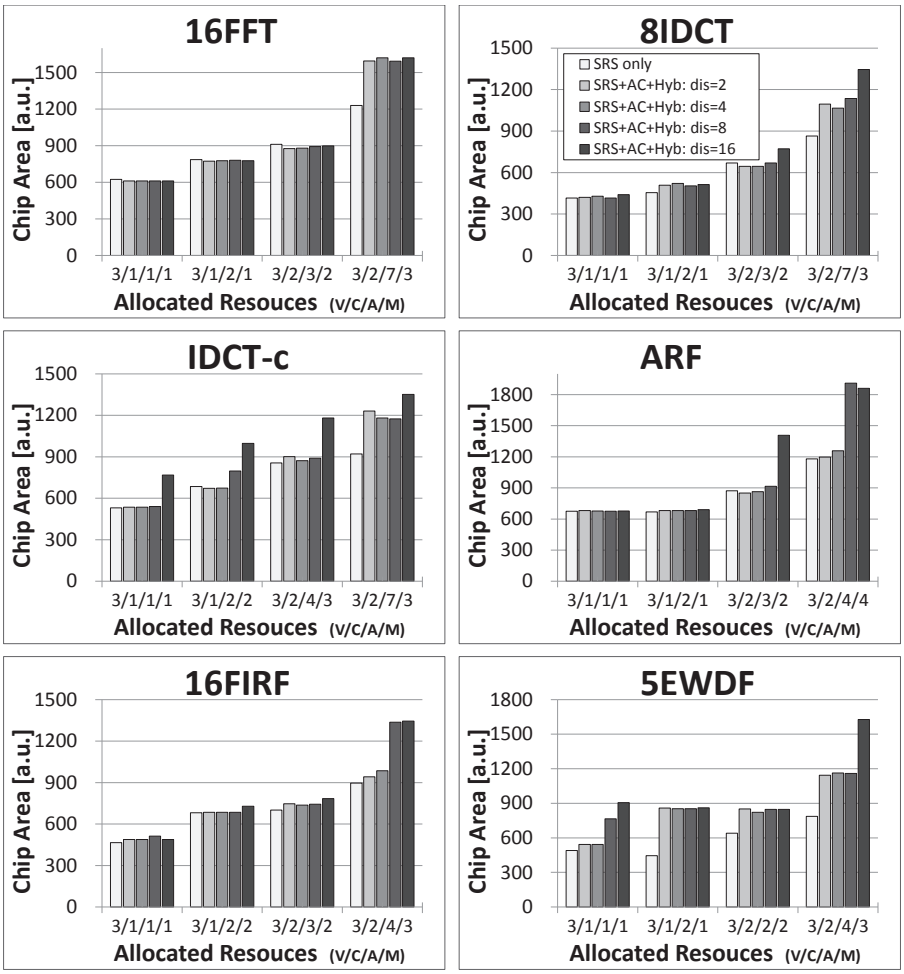

Figure 7.4: Experimental Result in Area Estimation

$$= \sum_{n=0}^{cs} r_n \cdot p_{dia}^n \cdot (1 - p_{dia})^{(cs-n)} \tag{7.4}$$

In the following numerical evaluations, it is assumed that n equals 1. It means that a single soft-error affects only one control step during the execution, in terms of temporal spread. In addition, the following is considered:

- a soft-error rate p_{dia} in every control step is assumed as 0.0001. This seems unrealistic large value, however, it is chosen to highlight the reliability degradation due to AC and the proposed hybrid method in the numerical evaluation.

- When a soft-error occurs at the coordination (x, y) at a control step, it affects all components within the boundary of the diameter dia from (x, y) at the control step. Consequently, the number of error pattern r_1 in a control step becomes the rate of the number of masked errors.

- When a standard register is in failure caused by a soft-error, the data which is stored in the register is corrupted at the same control step.

- If an ALU or multiplier is in failure due to a soft-error, incorrect output data of the ALU or multiplier is latched in a register when the control step is changed to the

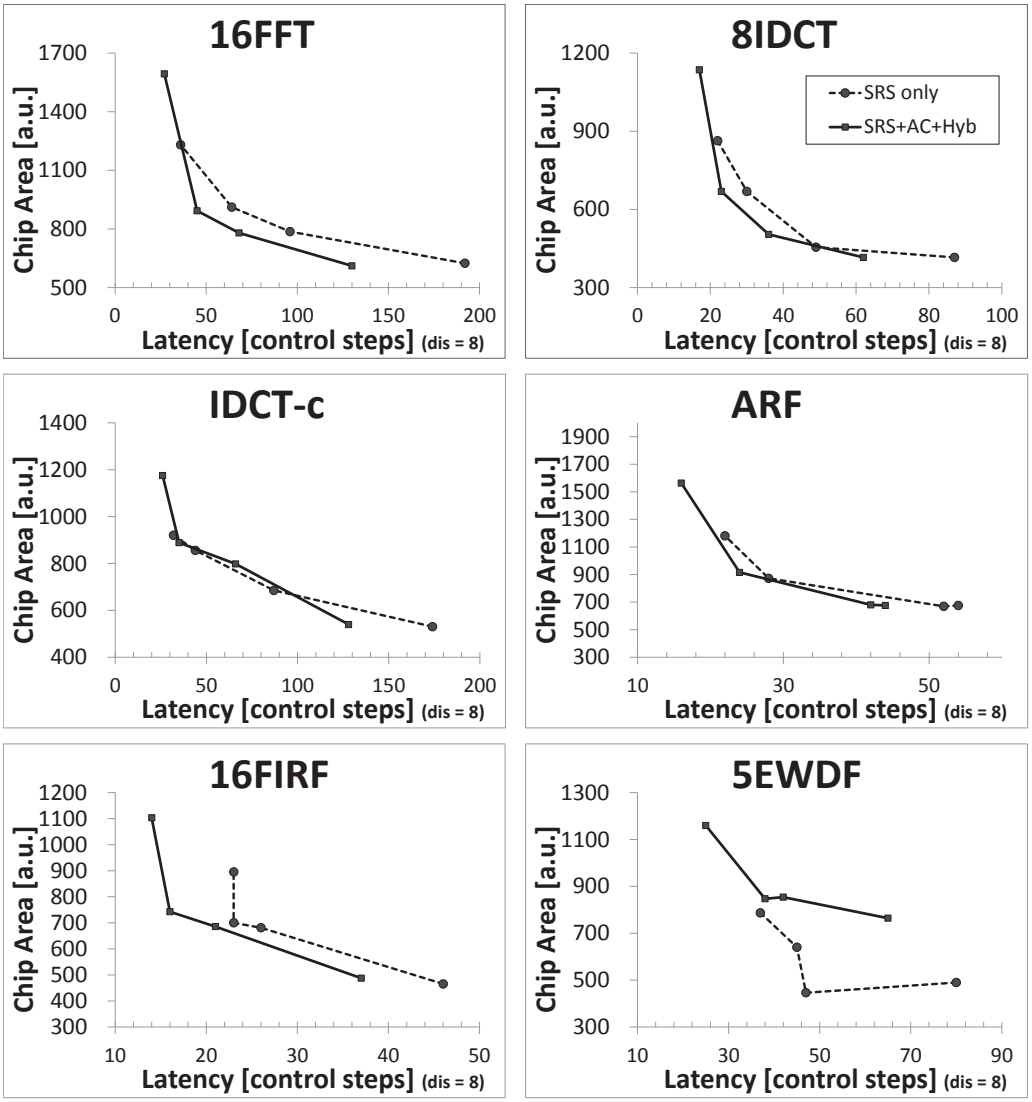

Figure 7.5: Area vs. Latency

next step.

- If a comparator is in failure caused by a soft-error and two inputs of the comparator are identical, the comparator outputs an incorrect comparison result, that is, two inputs are different. As a result, the retry-cone is executed. On the other hand, if a comparator is in failure due to a soft-error and two inputs are different, the comparator outputs an incorrect comparison result, that is, two inputs are identical. Consequently, the retry-cone will not be executed; furthermore, the erroneous outputs of the corresponding main-cone (and second-cone) will not be corrected.

- If a voter is in failure due to a soft-error, the voter outputs an incorrect result regardless of correctness of its inputs.

In the evaluation, all possible error patterns are checked whether two components on which AC are imposed are simultaneously affected by an error. There are two cases of the reliability evaluation:

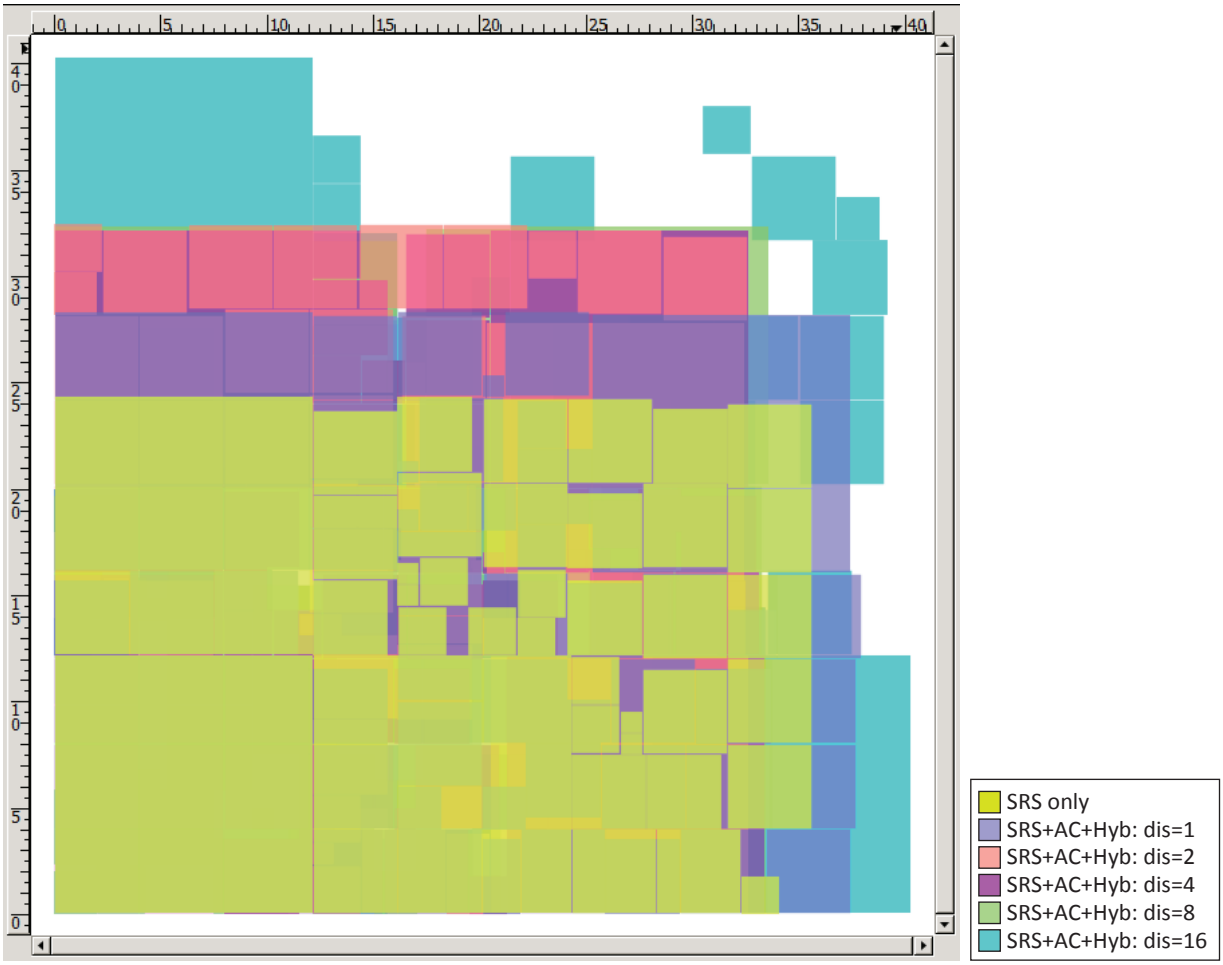

Figure 7.6: Comparison among different results based on SRS only and SRS+AC+Hyb under dis =1, 2, 4, 8 and 16

- Case 1: A single soft-error having a spatial boundary dia strikes at (x, y). The error coordination (x, y) is explored within the whole chip area in 0.5 [a.u.] intervals in the x axis and the y axis.
- Case 2: Two soft-errors having the same spatial boundary dia affect at (x_1, y_1) and (x_2, y_2) at the same time. The all combinations of (x_1, y_1) and (x_2, y_2) are investigated within the whole chip area in 0.5 [a.u.] intervals in the x axis and the y axis.

Although two components having AC are simultaneously affected by the error, the corrupt outputs from two components will not propagate at the same time if one of the two is in idle state. In that case, a corrupt output will be masked by voters or corrected by retry. If the affected component is a voter itself, the corrupt data will be corrected in the succeeding stage.

Figure 7.8 shows a placement result of AR filter benchmark under 3 voters, 4 ALUs, 4 multipliers and $dis = 4$. Except 16-bit soft-error tolerant registers, all components have their resource numbers. A, M, N and V stand for an ALU, a multiplier, a 16-bit normal register and a voter. Figure 7.9 shows a part of operation and variable schedule corresponding to Fig. 7.8, and allocated resources corresponding to operations and variables.

FTR stands for a 16-bit soft-error tolerant register. The placement in Fig. 7.8 is following spatial AC. As discussed earlier, in order to obtain a correct result in M-V scheme, at most one of the three outputs of the three cones (Main, Second and Third) are allowed to be incorrect (Condition 5). Otherwise, the erroneous outputs are never corrected by voters. Thus, every pair of two components in different cones which operate in the same control step should not be affected by a single soft-error. The two components in such pair are imposed on spatial AC. In Fig. 7.8, for example, M2, M3, A1, A4, N2, N4, N3 and N7 operate at the same control step CS2. Since N2 and N3 belong to the same cone, the pair of the two registers is not a subject of AC. Focusing on M2 at CS2, on the other hand, M2 and N2, M2 and N4, M2 and A1, M2 and N3, M2 and N7, and M2 and A4 are subjects of AC and you can confirm them in Fig. 7.8.

Figure 7.10 shows an evaluation pattern with $dia = 1$ in Case 2. In this example, the two soft-errors affect two components M2 and N4, on which AC is imposed, at the same time. If the errors occur at CS2, the erroneous outputs from the two components are not corrected by corresponding voters. Figure 7.11 illustrates an evaluation pattern with $dia = 16$ in Case 1. In this figure the boundary of a single soft-error is wider than imposed AC ($dis = 4$). If the error occurs at CS2, one single soft-error affects several components N2, A4, A3 and V1 at the same time. Since A3 and V1 are in idle state, there are no effect on those components. However, N2 and A4 belong to distinct cones, the single soft-error neutralizes voters.

Figure 7.12, 7.13, 7.14, 7.15 and 7.16 show the evaluation results. The reliability of the preliminarily proposed method (SRS only) is 1 because the two methods tolerate one single soft-error during the execution. *dis* represents spatial AC among components and *dia* indicates the spatial boundary of a single soft-error. When *dis* is larger than or equal to *dia*, the reliability of the proposed method in this chapter is 1 because AC imposed circuits guarantee to tolerate a soft-error having a spatial boundary up to *dis*. On the other hand, if *dis* is smaller than *dia*, that is, the effect of a single soft-error is wider than the assumption of the circuit, the reliability of the proposed method becomes smaller than 1 according to an increment of *dia*.

Figure 7.17, 7.18, 7.19, 7.20 and 7.21 show the evaluation results in a different condition that two soft-errors simultaneously affect at distinct spatial points at the same control step.

7.4 Summary

The proposed datapath circuit tolerates multi-component error caused by a single soft-error. In the previous chapters, the author proposed a special resource management called speculative resource sharing (SRS) and the consideration of temporal/spatial distance between components called adjacency constraint (AC). Based on these proposals, the author introduces a combination of a comparison-retry scheme and a majority-voting scheme under AC for time overhead mitigation. Furthermore, the author proposes an optimized assortment of the two distinct schemes to enhance the latency performance of datapaths. Soft-error tolerant datapath synthesis experiments reveal that the proposed method improves latency compared with a conventional comparison-retry mechanism while keeping the comparable levels in chip area and soft-error tolerance.

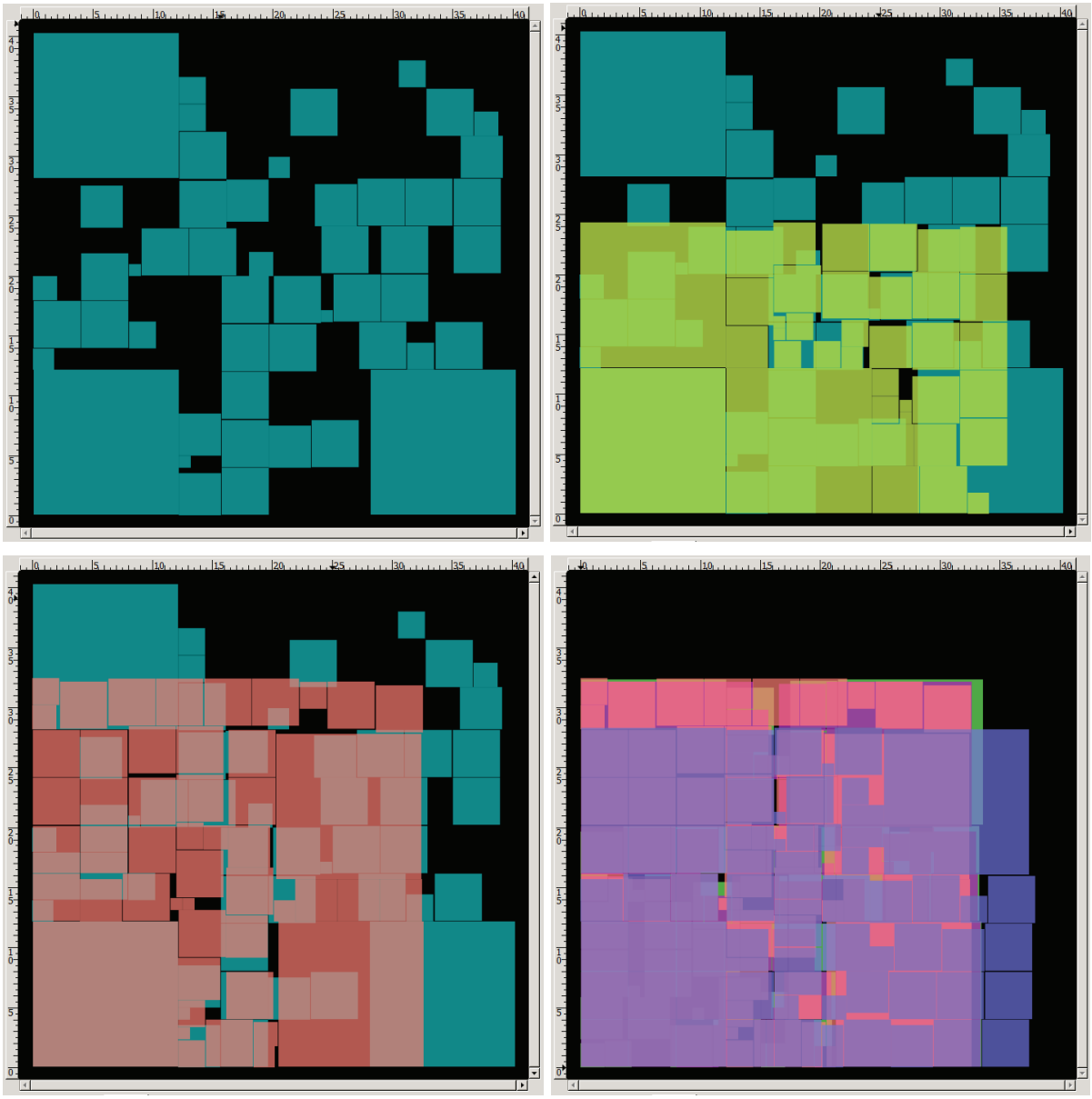

Figure 7.7: Different combination of comparison among different result based on SRS only and SRS+AC+Hyb under *dis* =1, 2, 4, 8 and 16

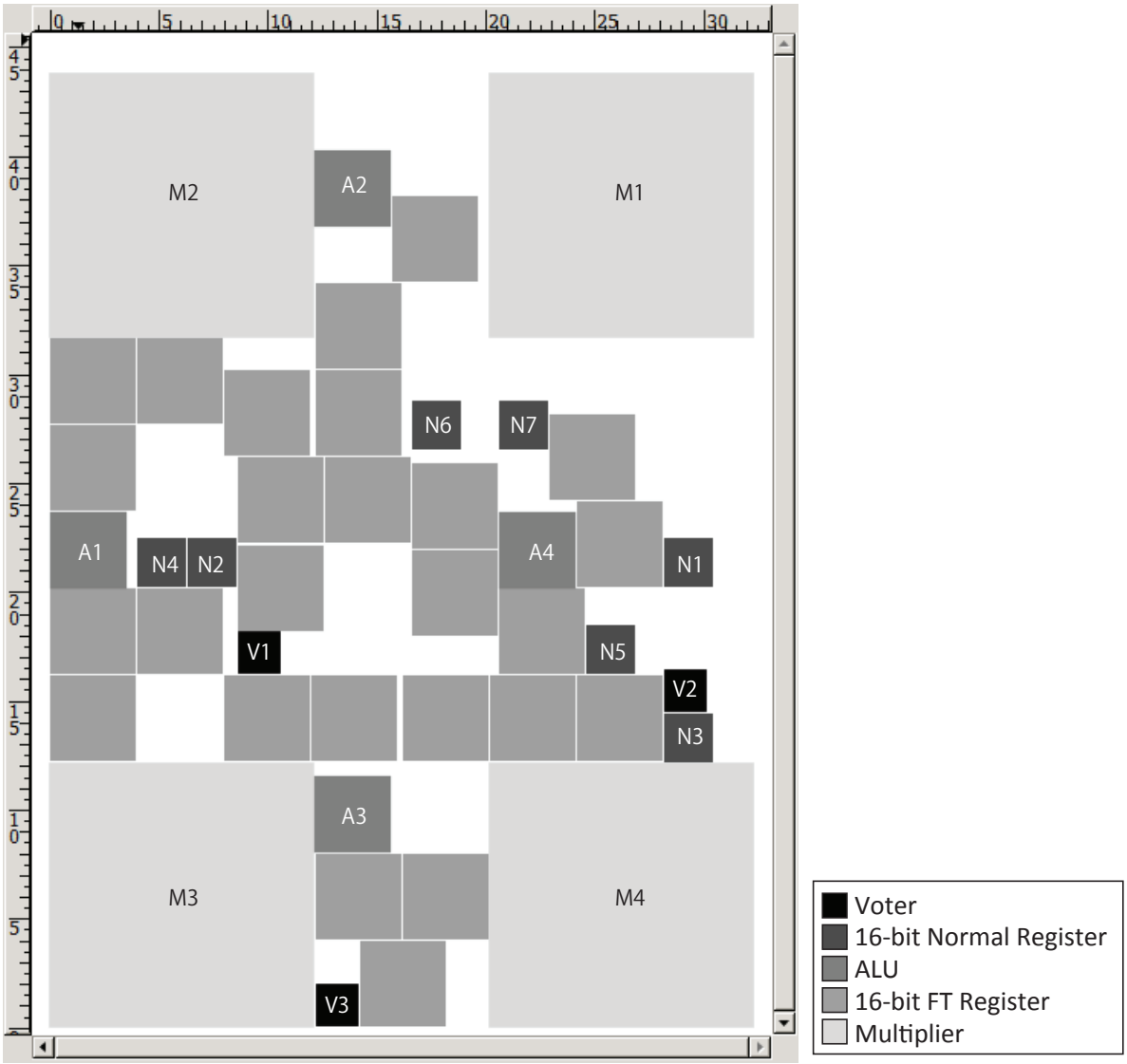

Figure 7.8: A placement result of components in case of AR filter under 3 voters, 4 ALUs, 4 multipliers and $dis = 4$

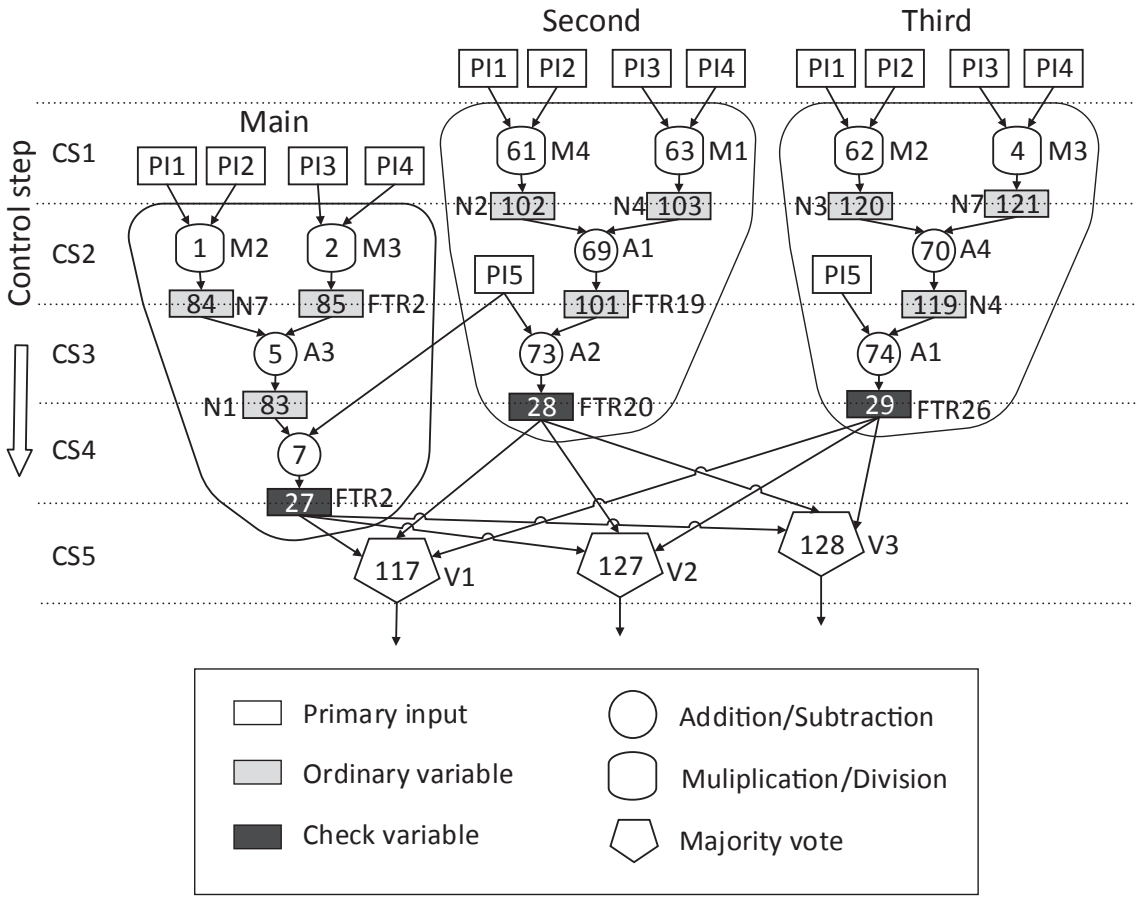

Figure 7.9: A part of operation and variable schedule in case of AR filter under 3 voters, 4 ALUs, 4 multipliers and $dis = 4$

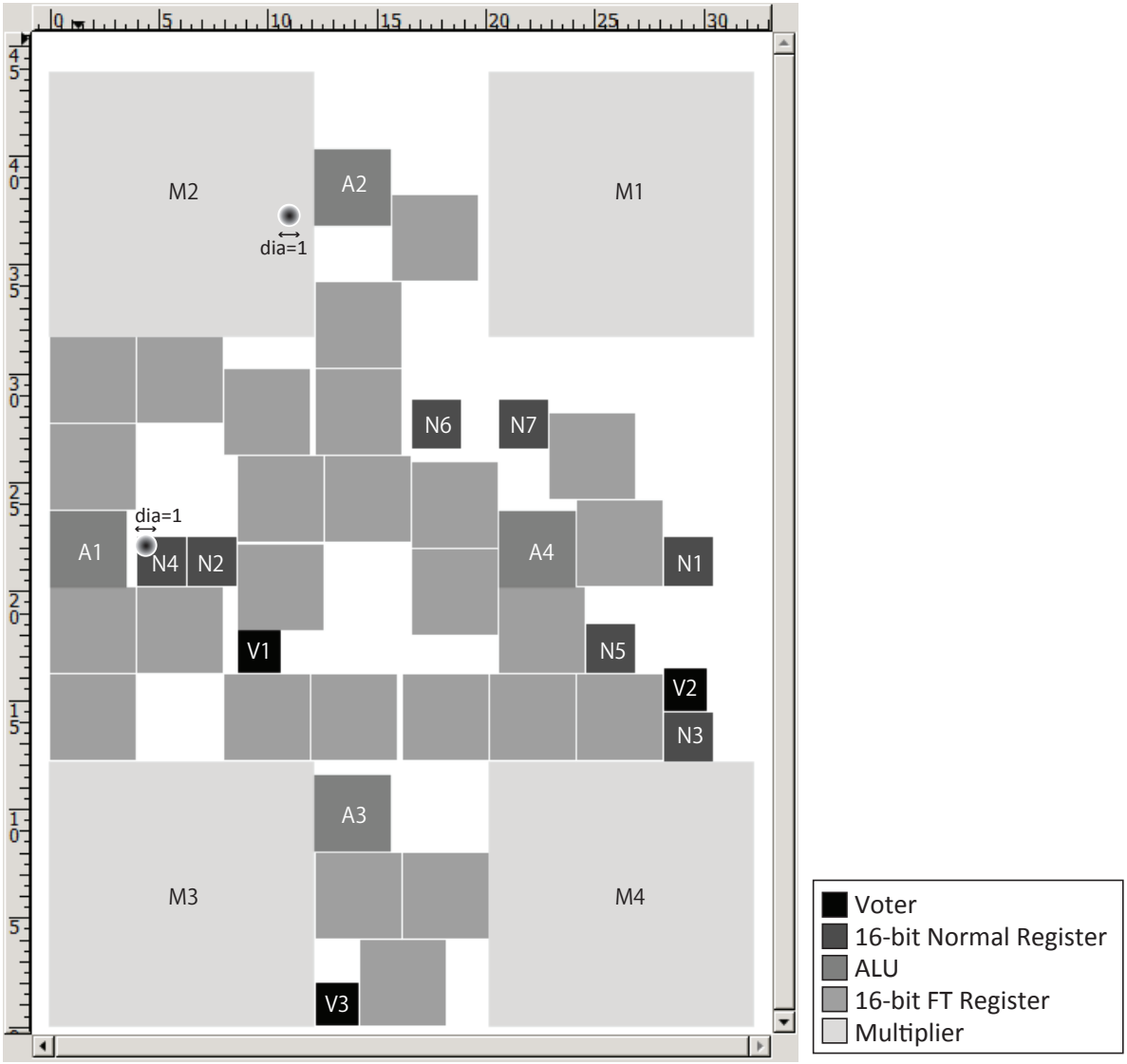

Figure 7.10: A simulation example of an effect of two soft-errors having spatial boundary $dia = 1$ (Case 2)

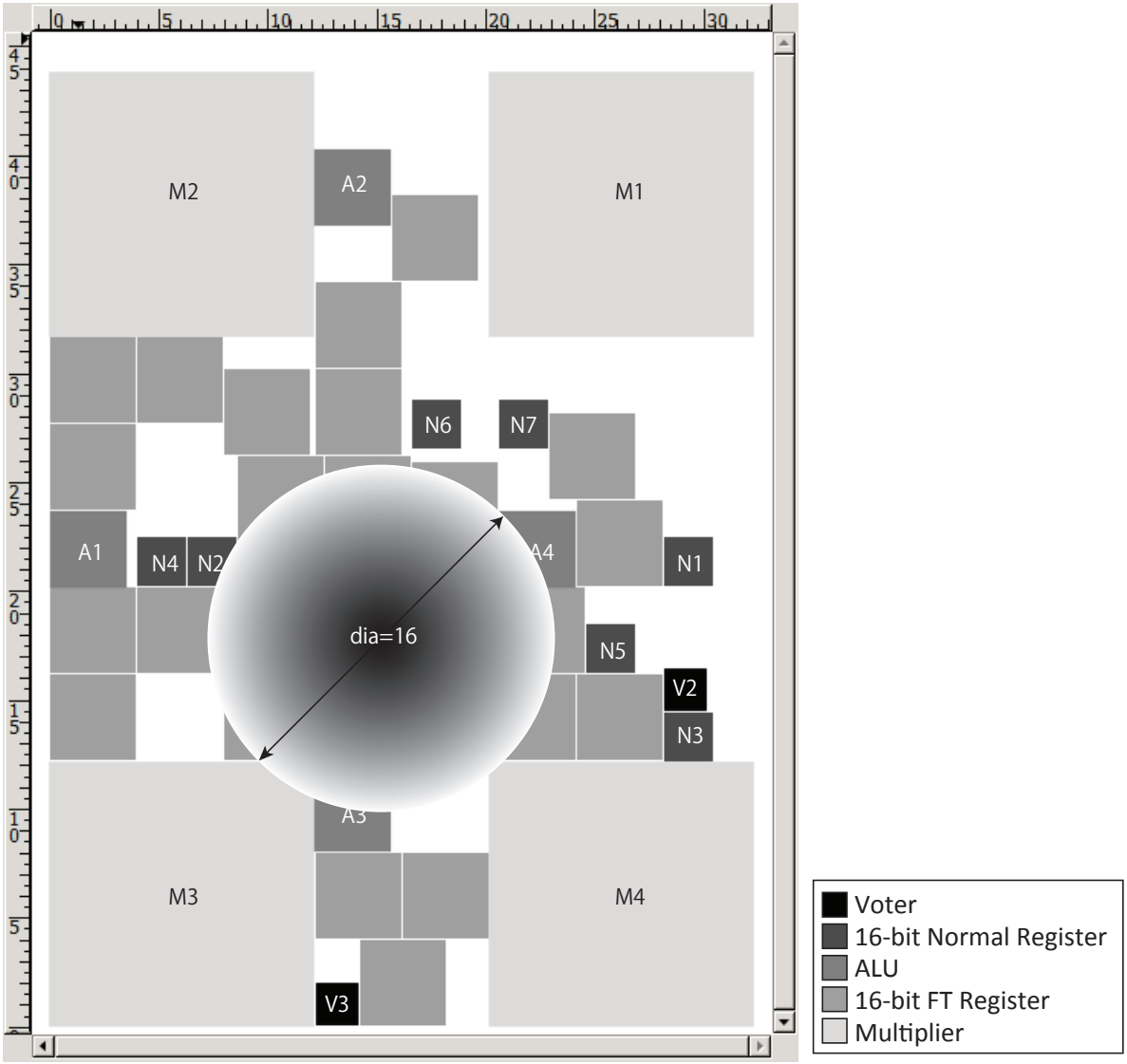

Figure 7.11: A simulation example of an effect of a single soft-error having spatial boundary $dia = 16$ (Case 1)

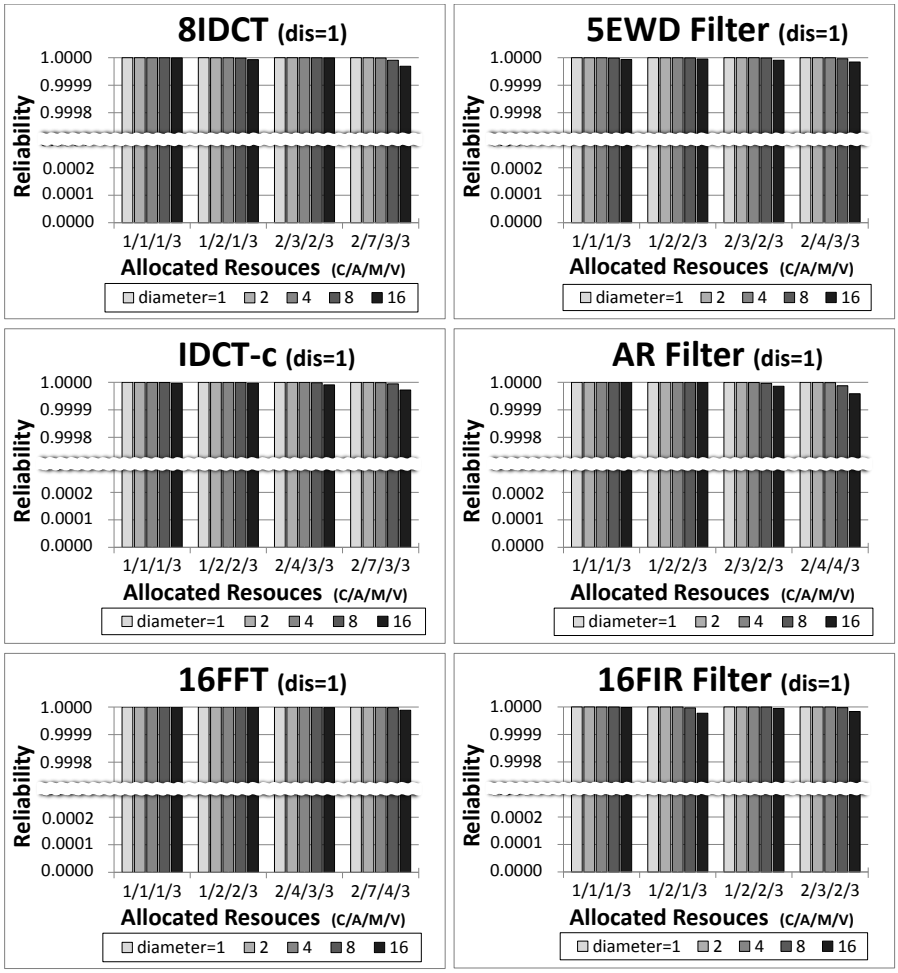

Figure 7.12: Reliability Evaluation results in six computational algorithms (Case 1, $dis = 1$): Each column corresponds to a different choice of allocated resources (C: Comparator, A: ALU, M: Multiplier, V: Voter).

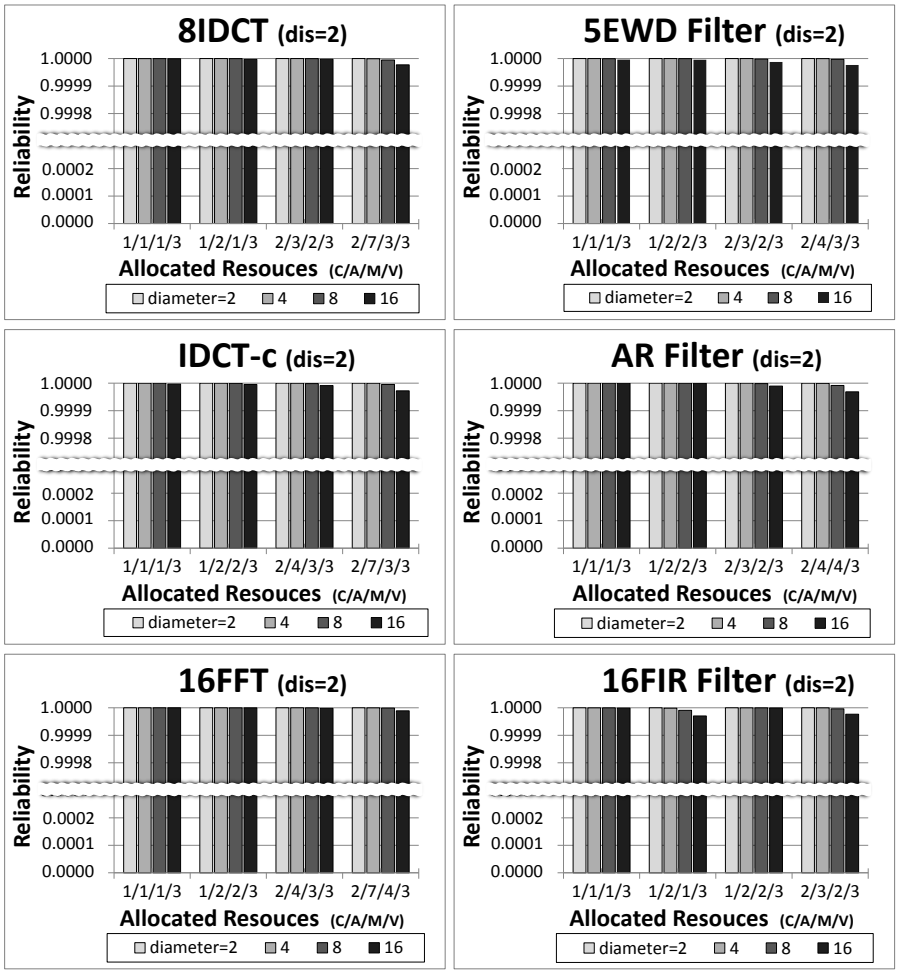

Figure 7.13: Reliability Evaluation results in six computational algorithms (Case 1, $dis = 2$): Each column corresponds to a different choice of allocated resources (C: Comparator, A: ALU, M: Multiplier, V: Voter).

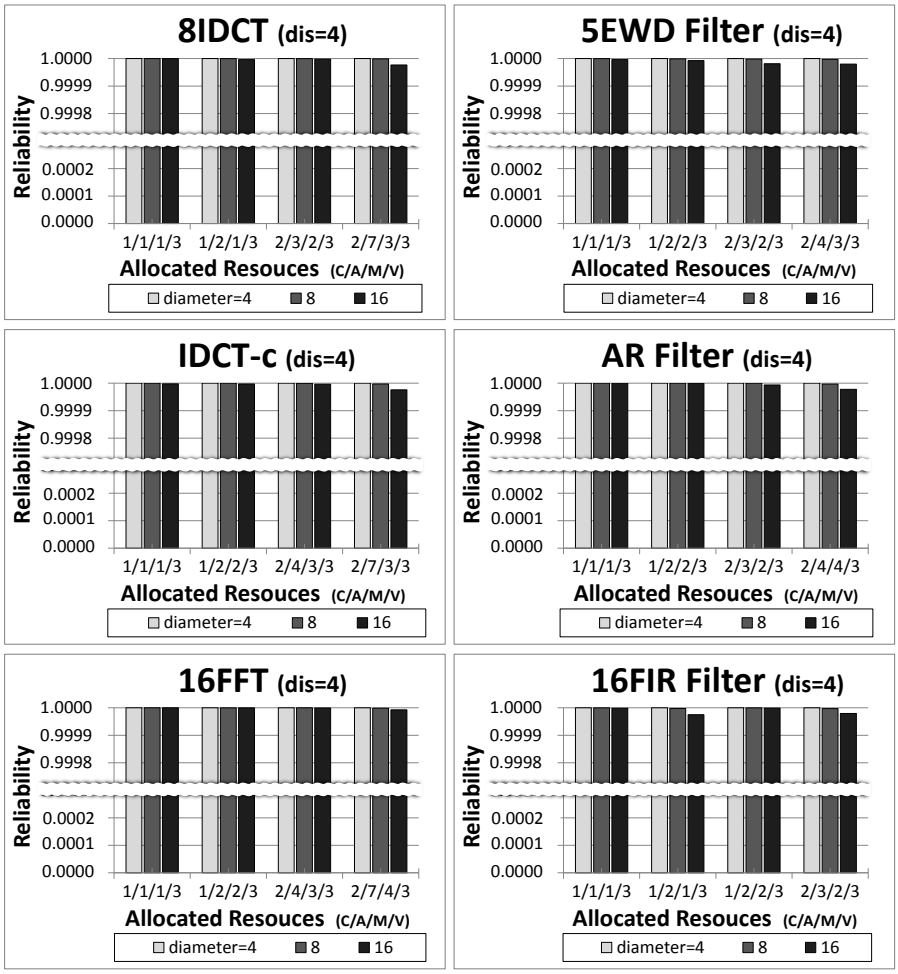

Figure 7.14: Reliability Evaluation results in six computational algorithms (Case 1, $dis = 4$): Each column corresponds to a different choice of allocated resources (C: Comparator, A: ALU, M: Multiplier, V: Voter).

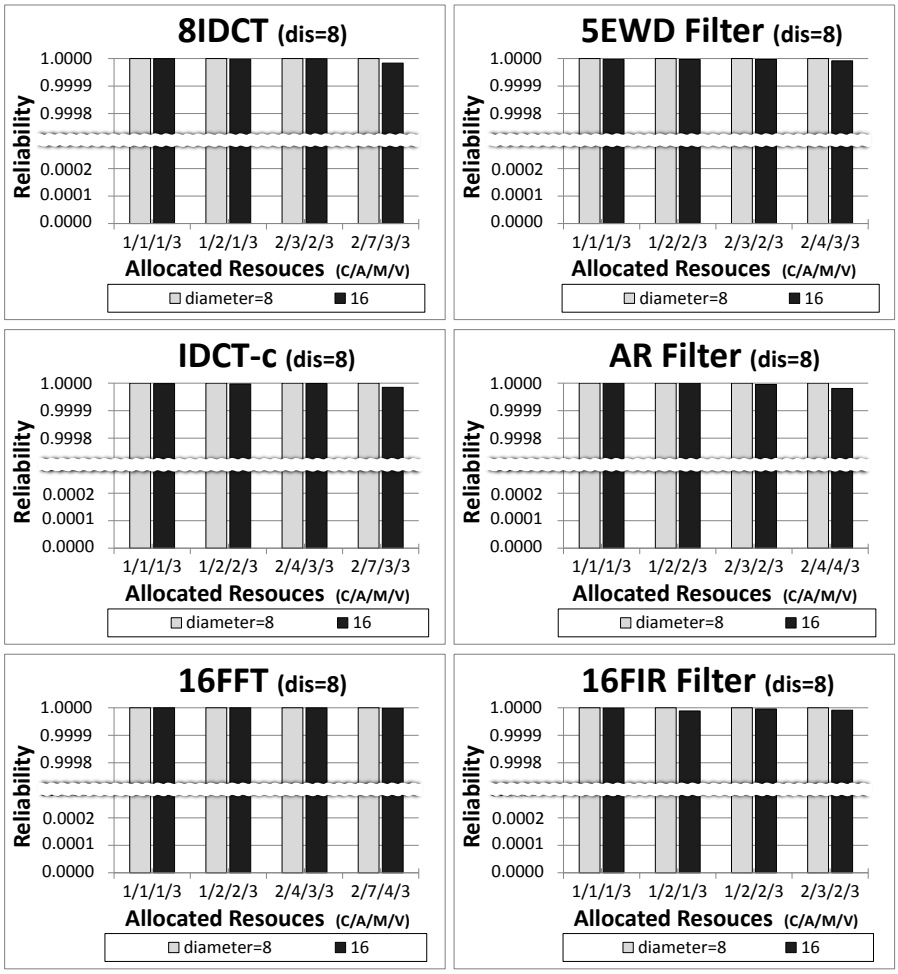

Figure 7.15: Reliability Evaluation results in six computational algorithms (Case 1, $dis = 8$): Each column corresponds to a different choice of allocated resources (C: Comparator, A: ALU, M: Multiplier, V: Voter).

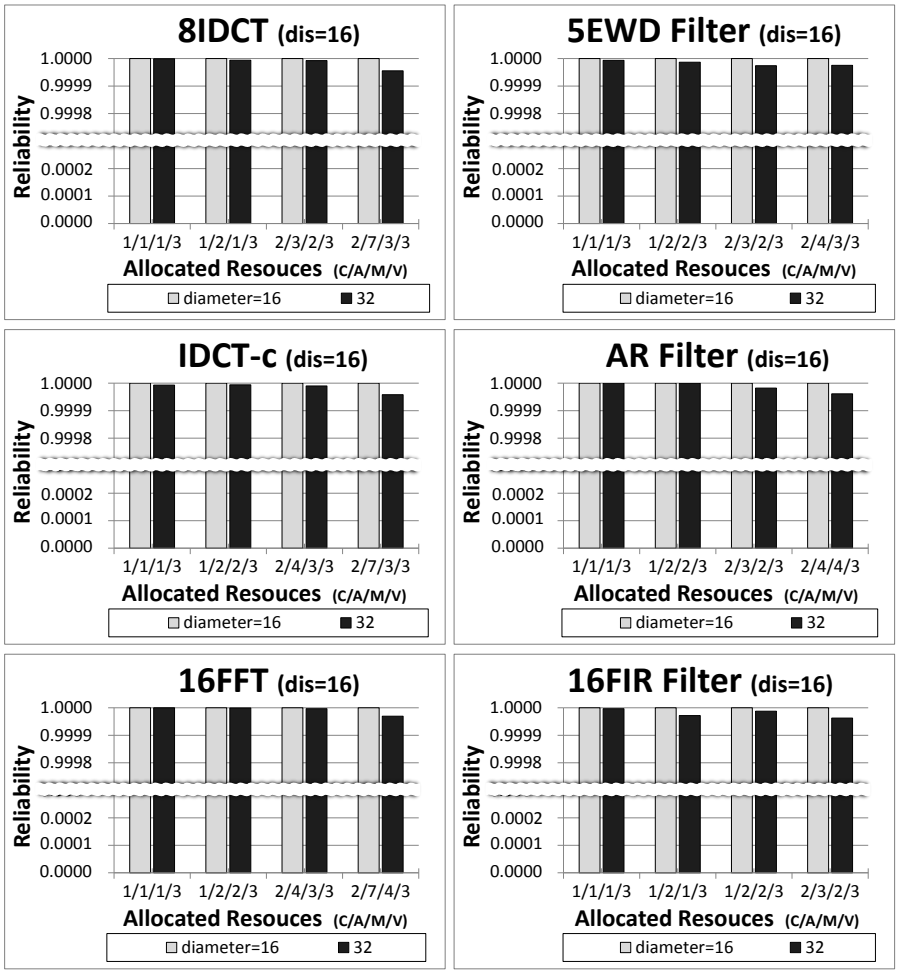

Figure 7.16: Reliability Evaluation results in six computational algorithms (Case 1, $dis = 16$): Each column corresponds to a different choice of allocated resources (C: Comparator, A: ALU, M: Multiplier, V: Voter).

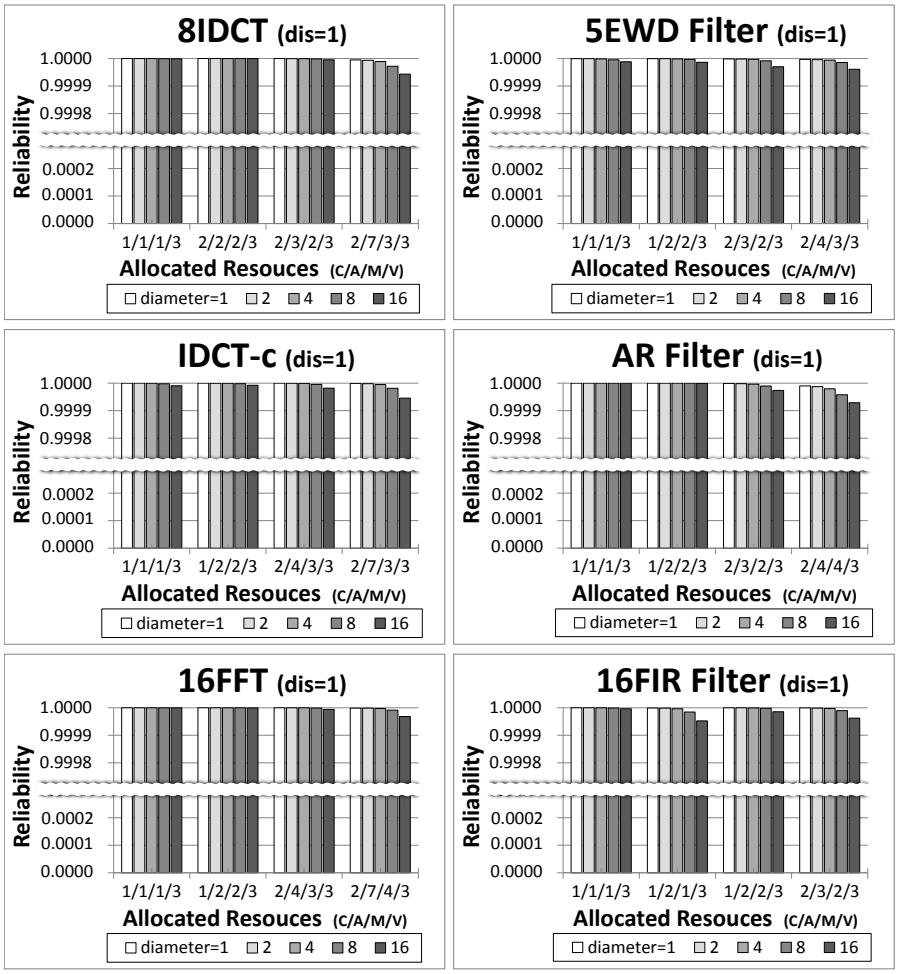

Figure 7.17: Reliability Evaluation results in six computational algorithms (Case 2, $dis = 1$): Each column corresponds to a different choice of allocated resources (C: Comparator, A: ALU, M: Multiplier, V: Voter).

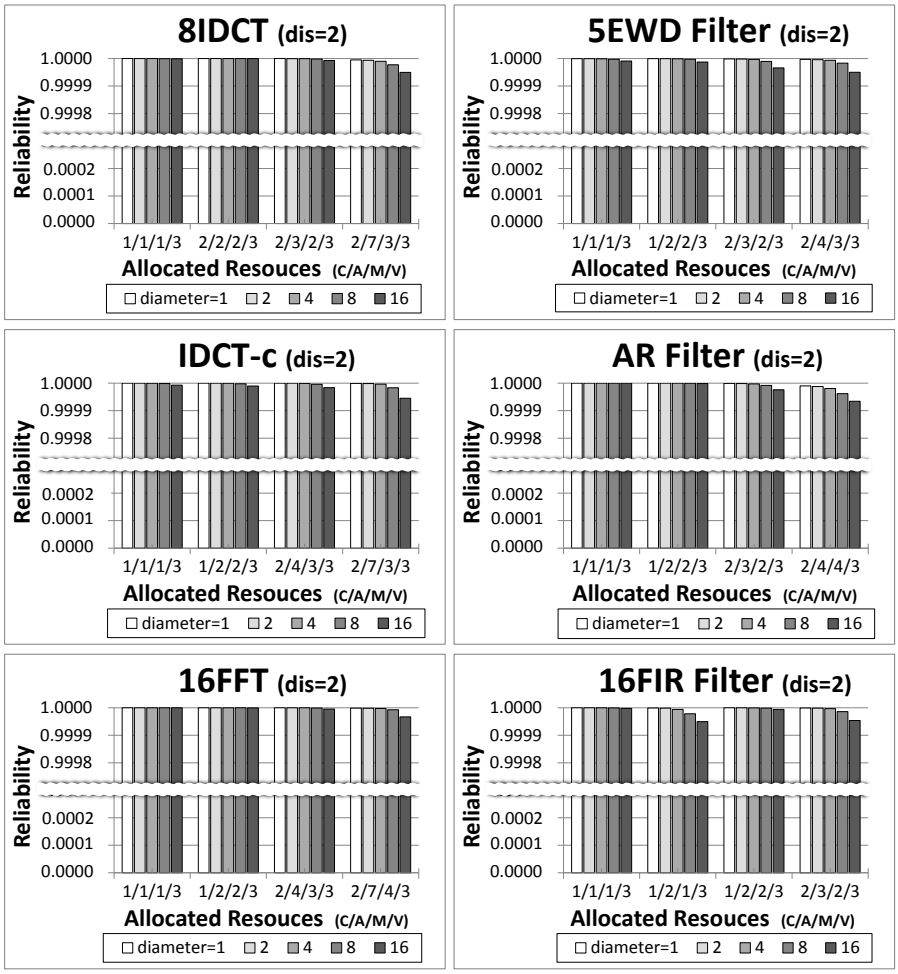

Figure 7.18: Reliability Evaluation results in six computational algorithms (Case 2, $dis = 2$): Each column corresponds to a different choice of allocated resources (C: Comparator, A: ALU, M: Multiplier, V: Voter).

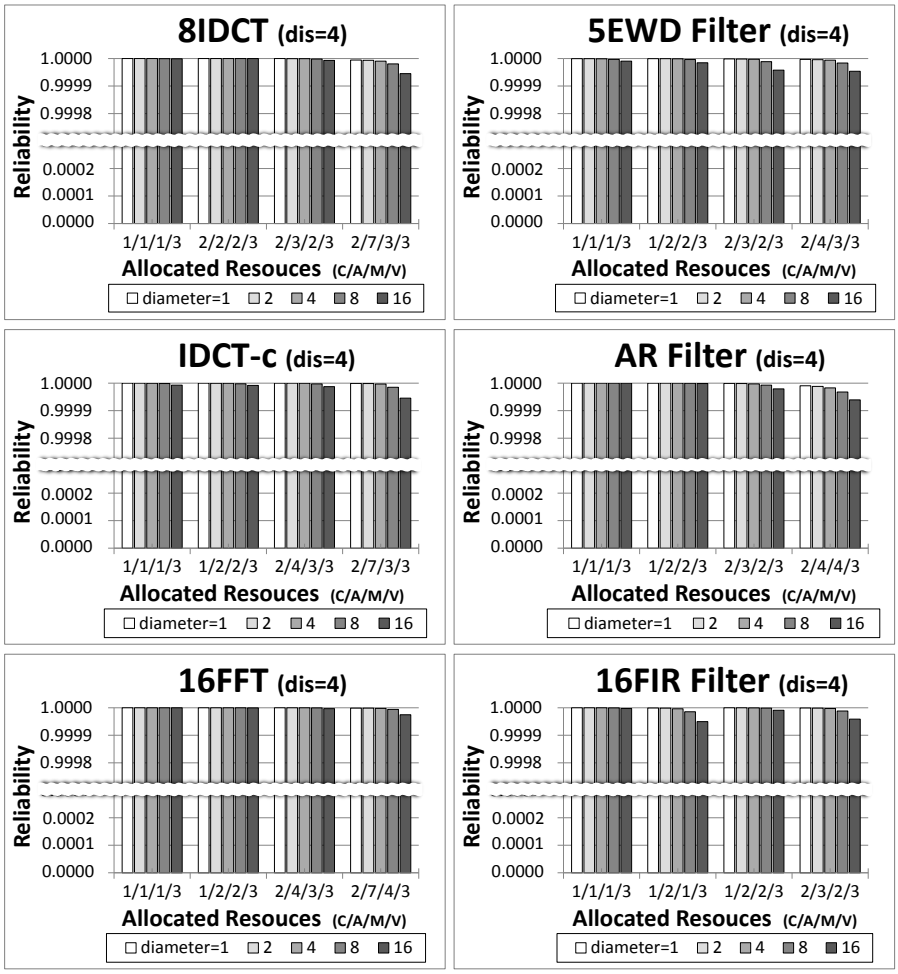

Figure 7.19: Reliability Evaluation results in six computational algorithms (Case 2, $dis = 4$): Each column corresponds to a different choice of allocated resources (C: Comparator, A: ALU, M: Multiplier, V: Voter).

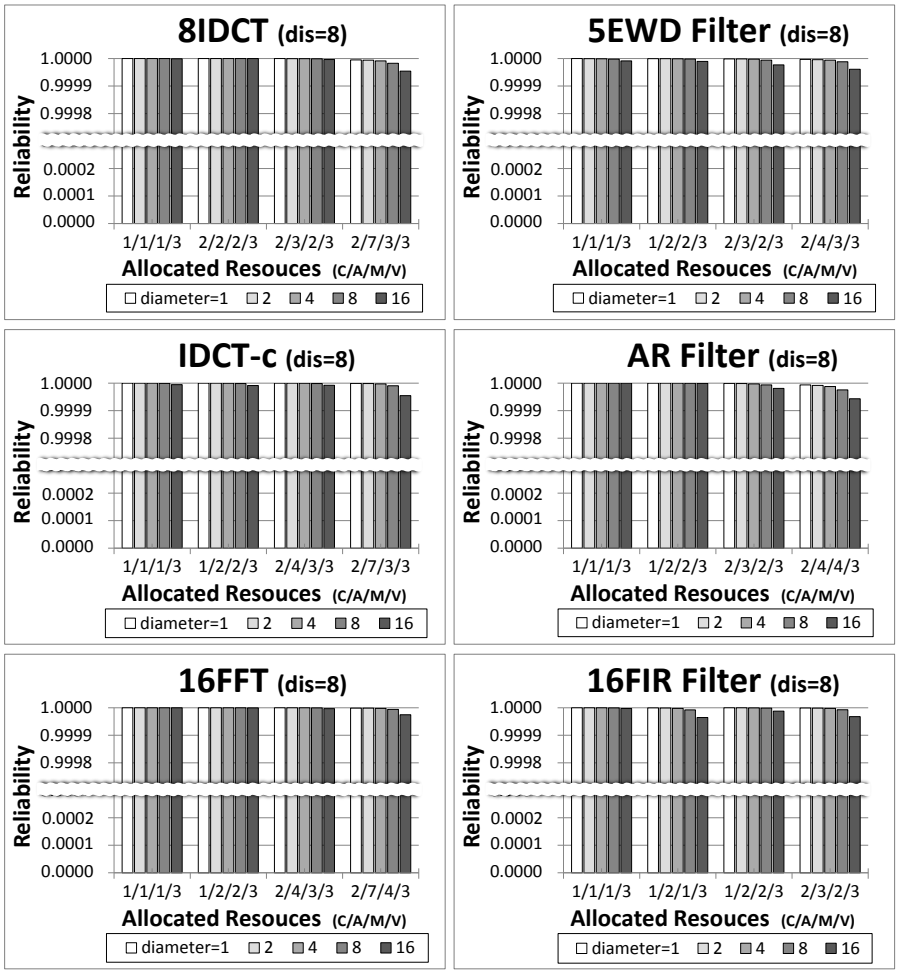

Figure 7.20: Reliability Evaluation results in six computational algorithms (Case 2, $dis = 8$): Each column corresponds to a different choice of allocated resources (C: Comparator, A: ALU, M: Multiplier, V: Voter).

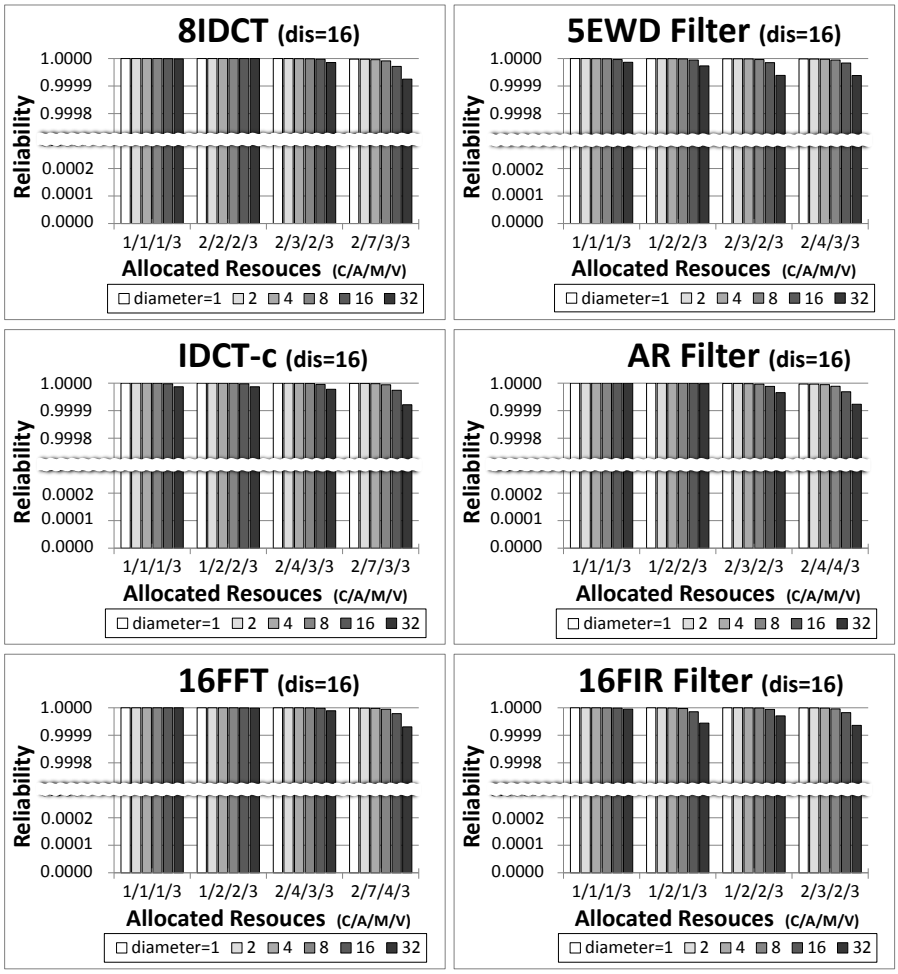

Figure 7.21: Reliability Evaluation results in six computational algorithms (Case 2, $dis = 16$): Each column corresponds to a different choice of allocated resources (C: Comparator, A: ALU, M: Multiplier, V: Voter).

Chapter 8

Conclusion

8.1 Summary of the Dissertation

Designing VLSIs from a abstraction layer such as high-level synthesis (HLS) is becoming important because of its flexibility which allows hardware engineers to make decisions at an early stage of design cycle [23]. In addition, reliable systems have been required in several different fields and those systems have relied on the dependability of semiconductor devices on their systems [4]. Under such circumstances, this dissertation deals with a soft-error tolerant application specific datapath design methodology via high-level synthesis. Datapaths designed by the proposed method tolerate multiple component errors induced by single soft-errors. Based on triple algorithm redundancy, the author suggests four novel proposals: (i) speculative resource sharing, (ii) latency-aware selection of check variables, (iii) adjacency constraint between datapath components and (iv) mixed error correction method.

In order to mitigate hardware/time overhead due to triple algorithm redundancy, speculative resource sharing (SRS) between the retry parts and the secondary parts is proposed. Because a strict constraint is imposed to implement SRS to datapath circuits, a latency-aware selection of check variables is suggested to satisfy the constraint much easier and maximize the possibility of SRS. In addition, to reduce excessively implemented fault-tolerance, the author introduces a concept of spatial and temporal localities of soft-errors, and the concept is applied as adjacency constraint (AC) between datapath components. AC allows not only the use of comparison-retry (C-R) scheme but also majority-voting (M-V) scheme. Since C-R and M-V schemes have their own advantages and drawbacks, the author decides to combine the two different types of error correction methods to merge the advantages of both schemes.

It is found that the proposed method is more effective when a computation algorithm possesses higher parallelism and a small number of resources is available. Moreover, the experimental results reveal that the proposed method improves latency, while keeping the comparable levels in chip areas and reliability compared with a conventional C-R scheme that neither SRS nor AC are implemented.

8.2 Future Work

The experimental results show improvements compared with a conventional method. Nevertheless, there are several issues which need to improve. In this dissertation, relatively small benchmark applications are used for experiments. To obtain more practical results, bigger and more complicated applications should be used for benchmark application algorithms, and power estimation should be added. Moreover, the most important thing is that a practical CAD tool should be used.

Bibliography

[1] B. Johnson, *Design and Analysis of Fault-tolerant Digital Systems*, ser. Addison-Wesley series in electrical and computer engineering. Addison-Wesley Publishing Company, 1989.

[2] N. P. Carter, H. Naeimi, and D. S. Gardner, "Design techniques for cross-layer resilience," in *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*, March 2010, pp. 1023–1028.

[3] P. Jalote, *Fault Tolerance in Distributed Systems*. PTR Prentice Hall, 1994.

[4] I. Koren and C. Krishna, *Fault-Tolerant Systems*. Elsevier Science, 2010.

[5] N. Kanekawa, E. Ibe, T. Suga, and Y. Uematsu, *Dependability in Electronic Systems: Mitigation of Hardware Failures, Soft Errors, and Electro-Magnetic Disturbances*, ser. SpringerLink : Bücher. Springer New York, 2010.

[6] E. Dubrova, *Fault-Tolerant Design*. Springer New York, 2013.

[7] N. Leveson, *SafeWare: System Safety and Computers*, ser. Computer Science and Electrical Engineering Series. Addison-Wesley, 1995.

[8] R. Ubar, J. Raik, and H. Vierhaus, *Design and Test Technology for Dependable Systems-on-chip*, ser. Premier reference source. Information Science Reference, 2010.

[9] S. Sayil, *Soft Error Mechanisms, Modeling and Mitigation*. Springer International Publishing, 2016.

[10] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 305–316, Sept 2005.

[11] M. Wilkening, V. Sridharan, S. Li, F. Previlon, S. Gurumurthi, and D. R. Kaeli, "Calculating architectural vulnerability factors for spatial multi-bit transient faults," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec 2014, pp. 293–305.

[12] J. Mathew, R. Shafik, and D. Pradhan, *Energy-Efficient Fault-Tolerant Systems*, ser. Embedded Systems. Springer New York, 2013.

[13] J. Autran and D. Munteanu, *Soft Errors: From Particles to Circuits*, ser. Devices, Circuits, and Systems. CRC Press, 2015.

[14] J. Bowen and V. Stavridou, "Safety-critical systems, formal methods and standards," *Software Engineering Journal*, vol. 8, no. 4, pp. 189–209, 1993.

[15] J. Losq, "Influence of fault-detection and switchinig mechanisms on the reliability of stand-by systems," Stanford, CA, USA, Tech. Rep., 1975.

[16] R. W. Hamming, "Error detecting and error correcting codes," *Bell Labs Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[17] H. Veendrick, *Nanometer CMOS ICs: From basics to ASICs*. Springer Tokyo, 2008.

[18] P. A. Thaker, V. D. Agrawal, and M. E. Zaghloul, "A test evaluation technique for vlsi circuits using register-transfer level fault modeling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 8, pp. 1104–1113, Aug 2003.

[19] T. Uemura, Y. Tosaka, H. Matsuyama, K. Shono, C. Uchibori, K. Takahisa, M. Fukuda, and K. Hatanaka, "Seila: Soft error immune latch for mitigating multi-node-seu and local-clock-set," in *Reliability Physics Symposium (IRPS), 2010 IEEE International*, May 2010, pp. 218–223.

[20] Y. Lin and M. Zwolinski, "Settoff: A fault tolerant flip-flop for building cost-efficient reliable systems," in *On-Line Testing Symposium (IOLTS), 2012 IEEE 18th International*, June 2012, pp. 7–12.

[21] M. Masuda, K. Kubota, R. Yamamoto, J. Furuta, K. Kobayashi, and H. Onodera, "A 65 nm low-power adaptive-coupling redundant flip-flop," *Nuclear Science, IEEE Transactions on*, vol. 60, no. 4, pp. 2750–2755, Aug 2013.

[22] S. Mitra, K. Brelsford, and P. N. Sanda, "Cross-layer resilience challenges: Metrics and optimization," in *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*, March 2010, pp. 1029–1034.

[23] S. Mohanty, N. Ranganathan, E. Kougianos, and P. Patra, *Low-Power High-Level Synthesis for Nanoscale CMOS Circuits*. Springer International Publishing, 2008.

[24] P. Michel, U. Lauther, and P. Duzy, *The Synthesis Approach to Digital System Design*, ser. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, 1992.

[25] P. Coussy, D. D. Gajski, M. Meredith, and A. Takach, "An introduction to high-level synthesis," *IEEE Design Test of Computers*, vol. 26, no. 4, pp. 8–17, July 2009.

[26] A. Sengupta and D. Kachave, "Low cost fault tolerance against kc-cycle and km-unit transient for loop based control data flow graphs during physically aware high level synthesis," *Microelectronics Reliability*, vol. 74, pp. 88 – 99, 2017.

[27] J. Oh and M. Kaneko, "Area-efficient soft-error tolerant datapath synthesis based on speculative resource sharing," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E99–A, no. 7, pp. 1311–1322, 2016.

[28] ——, "Soft-error tolerant datapath synthesis considering adjacency constraint between components," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2016, pp. 595–600.

[29] ——, "Mixed error correction scheme and its design optimization for soft-error tolerant datapaths," in *2016 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, Oct 2016, pp. 362–365.

[30] T. Nakamura, *Terrestrial Neutron-induced Soft Errors in Advanced Memory Devices*. World Scientific, 2008.

[31] T. Herault and Y. Robert, *Fault-Tolerance Techniques for High-Performance Computing*, ser. Computer Communications and Networks. Springer International Publishing, 2015.

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

[33] H. Fuketa, R. Harada, M. Hashimoto, and T. Onoye, "Measurement and analysis of alpha-particle-induced soft errors and multiple-cell upsets in 10t subthreshold sram," *Device and Materials Reliability, IEEE Transactions on*, vol. 14, no. 1, pp. 463–470, March 2014.

[34] J. Autran, S. Serre, D. Munteanu, S. Martinie, S. Semikh, S. Sauze, S. Uznanski, G. Gasiot, and P. Roche, "Real-time soft-error testing of 40nm srams," in *Reliability Physics Symposium (IRPS), 2012 IEEE International*, April 2012, pp. 3C.5.1–3C.5.9.

[35] H. Quinn, K. Morgan, P. Graham, J. Krone, M. Caffrey, and K. Lundgreen, "Domain crossing errors: Limitations on single device triple-modular redundancy circuits in xilinx fpgas," *Nuclear Science, IEEE Transactions on*, vol. 54, no. 6, pp. 2037–2043, Dec 2007.

[36] H. M. Quinn, D. A. Black, W. H. Robinson, and S. P. Buchner, "Fault simulation and emulation tools to augment radiation-hardness assurance testing," *IEEE Transactions on Nuclear Science*, vol. 60, no. 3, pp. 2119–2142, June 2013.

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

[38] M. P. Baze, S. P. Buchner, and D. McMorrow, "A digital cmos design technique for seu hardening," *IEEE Transactions on Nuclear Science*, vol. 47, no. 6, pp. 2603–2608, Dec 2000.

[39] K. Zhang, J. Furuta, K. Kobayashi, and H. Onodera, "Dependence of cell distance and well-contact density on mcu rates by device simulations and neutron experiments in a 65-nm bulk process," *IEEE Transactions on Nuclear Science*, vol. 61, no. 4, pp. 1583–1589, Aug 2014.

[40] J. D. Black, A. L. Sternberg, M. L. Alles, A. F. Witulski, B. L. Bhuva, L. W. Massengill, J. M. Benedetto, M. P. Baze, J. L. Wert, and M. G. Hubert, "Hbd layout isolation techniques for multiple node charge collection mitigation," *IEEE Transactions on Nuclear Science*, vol. 52, no. 6, pp. 2536–2541, Dec 2005.

[41] T. Nakauchi, N. Mikami, A. Oyama, H. Kobayashi, H. Usui, and J. Kase, "A novel technique for mitigating neutron-induced multi-cell upset by means of back bias," in *2008 IEEE International Reliability Physics Symposium*, April 2008, pp. 187–191.

[42] N. M. Atkinson, A. F. Witulski, W. T. Holman, J. R. Ahlbin, B. L. Bhuva, and L. W. Massengill, "Layout technique for single-event transient mitigation via pulse quenching," *IEEE Transactions on Nuclear Science*, vol. 58, no. 3, pp. 885–890, June 2011.

[43] K. Zhang and K. Kobayashi, "Contributions of charge sharing and bipolar effects to cause or suppress mcus on redundant latches," in *2013 IEEE International Reliability Physics Symposium (IRPS)*, April 2013, pp. SE.5.1–SE.5.4.

[44] T. Uemura, T. Kato, H. Matsuyama, and M. Hashimoto, "Mitigating multi-bit-upset with well-slits in 28nm multi-bit-latch," *IEEE Transactions on Nuclear Science*, vol. 60, no. 6, pp. 4362–4367, Dec 2013.

[45] B. Narasimham, J. K. Wang, N. Vedula, S. Gupta, B. Bartz, C. Monzel, I. Chatterjee, B. L. Bhuva, R. D. Schrimpf, and R. A. Reed, "Influence of supply voltage on the multi-cell upset soft error sensitivity of dual- and triple-well 28 nm cmos srams," in *2015 IEEE International Reliability Physics Symposium*, April 2015, pp. 2C.4.1–2C.4.5.

[46] J. Maiz, S. Hareland, K. Zhang, and P. Armstrong, "Characterization of multi-bit soft error events in advanced srams," in *IEEE International Electron Devices Meeting 2003*, Dec 2003, pp. 21.4.1–21.4.4.

[47] S. Baeg, S. Wen, and R. Wong, "Sram interleaving distance selection with a soft error failure model," *IEEE Transactions on Nuclear Science*, vol. 56, no. 4, pp. 2111–2118, Aug 2009.

[48] J. Furuta, K. Kobayashi, and H. Onodera, "Impact of cell distance and well-contact density on neutron-induced multiple cell upsets," in *2013 IEEE International Reliability Physics Symposium (IRPS)*, April 2013, pp. 6C.3.1–6C.3.4.

[49] R. Naseer and J. Draper, "Parallel double error correcting code design to mitigate multi-bit upsets in srams," in *ESSCIRC 2008 - 34th European Solid-State Circuits Conference*, Sept 2008, pp. 222–225.

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

[51] R. Naseer and J. Draper, "The df-dice storage element for immunity to soft errors," in *48th Midwest Symposium on Circuits and Systems, 2005.*, Aug 2005, pp. 303–306 Vol. 1.

[52] J. Guo, L. Xiao, T. Wang, S. Liu, X. Wang, and Z. Mao, "Soft error hardened memory design for nanoscale complementary metal oxide semiconductor technology," *IEEE Transactions on Reliability*, vol. 64, no. 2, pp. 596–602, June 2015.

[53] H. R. Zarandi, S. G. Miremadi, C. Argyrides, and D. K. Pradhan, "Fast seu detection and correction in lut configuration bits of sram-based fpgas," in *2007 IEEE International Parallel and Distributed Processing Symposium*, March 2007, pp. 1–6.

[54] Y. Crouzet and C. Landrault, "Design of self-checking mos-lsi circuits: Application to a four-bit microprocessor," *IEEE Transactions on Computers*, vol. C-29, no. 6, pp. 532–537, June 1980.

[55] R. M. Sedmak and H. L. Liebergot, "Fault tolerance of a general purpose computer implemented by very large scale integration," *IEEE Transactions on Computers*, vol. C-29, no. 6, pp. 492–500, June 1980.

[56] M. M. Yen, W. K. Fuchs, and J. A. Abraham, "Designing for concurrent error detection in vlsi: application to a microprogram control unit," *IEEE Journal of Solid-State Circuits*, vol. 22, no. 4, pp. 595–605, Aug 1987.

[57] N. K. Jha and S. J. Wang, "Design and synthesis of self-checking vlsi circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 6, pp. 878–887, Jun 1993.

[58] K. Wu and R. Karri, "Algorithm level re-computing: A register transfer level concurrent error detection technique," in *Proceedings of the 2001 IEEE/ACM International Conference on Computer-aided Design*, ser. ICCAD '01. Piscataway, NJ, USA: IEEE Press, 2001, pp. 537–543.

[59] G. Lakshminarayana, A. Raghunathan, and N. K. Jha, "Behavioral synthesis of fault secure controller/datapaths based on aliasing probability analysis," *IEEE Transactions on Computers*, vol. 49, no. 9, pp. 865–885, Sep 2000.

[60] A. Orailoglu and R. Karri, "Automatic synthesis of self-recovering vlsi systems," *Computers, IEEE Transactions on*, vol. 45, no. 2, pp. 131–142, Feb 1996.

[61] A. Antola, F. Ferrandi, V. Piuri, and M. Sami, "Semiconcurrent error detection in data paths," *IEEE Transactions on Computers*, vol. 50, no. 5, pp. 449–465, May 2001.

[62] K. Wu and R. Karri, "Fault secure datapath synthesis using hybrid time and hardware redundancy," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 23, no. 10, pp. 1476–1485, Oct 2004.

[63] Y. Liu and K. Wu, "Fault-duration and-location aware ced technique with runtime adaptability," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 3, pp. 507–515, March 2014.

[64] A. Sengupta and S. Bhadauria, "Error masking of transient faults: Exploration of a fault tolerant datapath based on user specified power and delay budget," in *2014 International Conference on Information Technology*, Dec 2014, pp. 345–350.

[65] T. Iwagaki, T. Nakaso, R. Ohkubo, H. Ichihara, and T. Inoue, "Scheduling algorithm in datapath synthesis for long duration transient fault tolerance," in *2014 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, Oct 2014, pp. 128–133.

[66] J. Von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," *Automata studies*, vol. 34, pp. 43–98, 1956.

[67] N. Gaitanis, "The design of totally self-checking tmr fault-tolerant systems," *IEEE Transactions on Computers*, vol. 37, no. 11, pp. 1450–1454, Nov 1988.

[68] N. Rollins, "Evaluating tmr techniques in the presence of single event upsets," *Proc. Military and Aerospace Programmable Logic Devices, Sept. 2003*, 2003.

[69] P. K. Samudrala, J. Ramos, and S. Katkoori, "Selective triple modular redundancy (stmr) based single-event upset (seu) tolerant synthesis for fpgas," *IEEE Transactions on Nuclear Science*, vol. 51, no. 5, pp. 2957–2969, Oct 2004.

[70] S. D'Angelo, C. Metra, and G. Sechi, "Transient and permanent fault diagnosis for fpga-based tmr systems," in *Defect and Fault Tolerance in VLSI Systems, 1999. DFT '99. International Symposium on*, Nov 1999, pp. 330–338.

[71] C. Bolchini, A. Miele, and M. D. Santambrogio, "Tmr and partial dynamic reconfiguration to mitigate seu faults in fpgas," in *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007)*, Sept 2007, pp. 87–95.

[72] 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*, vol. 56, no. 4, pp. 2091–2102, Aug 2009.

[73] J. R. Azambuja, F. Sousa, L. Rosa, and F. L. Kastensmidt, "Evaluating large grain tmr and selective partial reconfiguration for soft error mitigation in sram-based fpgas," in *2009 15th IEEE International On-Line Testing Symposium*, June 2009, pp. 101–106.

[74] A. Sengupta and D. Kachave, "Low cost fault tolerance against kc-cycle and km-unit transient for loop based control data flow graphs during physically aware high level synthesis," *Microelectronics Reliability*, vol. 74, no. Supplement C, pp. 88 – 99, 2017.

[75] S. Krishnamohan and N. R. Mahapatra, "Combining error masking and error detection plus recovery to combat soft errors in static cmos circuits," in *2005 International Conference on Dependable Systems and Networks (DSN'05)*, June 2005, pp. 40–49.

[76] P. v. Stralen and A. Pimentel, "A safe approach towards early design space exploration of fault-tolerant multimedia mpsocs," in *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES+ISSS '12. New York, NY, USA: ACM, 2012, pp. 393–402.

[77] C. Bolchini and A. Miele, "Reliability-driven system-level synthesis for mixed-critical embedded systems," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2489–2502, Dec 2013.

[78] H. Zhang, M. A. Kochte, M. E. Imhof, L. Bauer, H. J. Wunderlich, and J. Henkel, "Guard: Guaranteed reliability in dynamically reconfigurable systems," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2014, pp. 1–6.

[79] K. Ito, "Energy minimization of full tmr design with optimized selection of temporal/spatial tmr mode and supply voltage," *IEICE Trans. Fundamentals*, vol. 97, no. 12, pp. 2530–2539, 2014.

[80] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: a low-power pipeline based on circuit-level timing speculation," in *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, Dec 2003, pp. 7–18.

[81] S. Das, C. Tokunaga, S. Pant, W. H. Ma, S. Kalaiselvan, K. Lai, D. M. Bull, and D. T. Blaauw, "Razorii: In situ error detection and correction for pvt and ser tolerance," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 1, pp. 32–48, Jan 2009.

[82] S. Mitra, W. J. Huang, N. R. Saxena, S. Y. Yu, and E. J. McCluskey, "Reconfigurable architecture for autonomous self-repair," *IEEE Design Test of Computers*, vol. 21, no. 3, pp. 228–240, May 2004.

[83] N. D. P. Avirneni and A. Somani, "Low overhead soft error mitigation techniques for high-performance and aggressive designs," *IEEE Transactions on Computers*, vol. 61, no. 4, pp. 488–501, April 2012.

[84] Z. Ghaderi, S. G. Miremadi, H. Asadi, and M. Fazeli, "Hafta: Highly available fault-tolerant architecture to protect sram-based reconfigurable devices against multiple bit upsets," *IEEE Transactions on Device and Materials Reliability*, vol. 13, no. 1, pp. 203–212, March 2013.

[85] M. Zhu, N. Song, and X. Pan, "Mitigation and experiment on neutron induced single-event upsets in sram-based fpgas," *IEEE Transactions on Nuclear Science*, vol. 60, no. 4, pp. 3063–3073, Aug 2013.

[86] H. Zhang, Y. Li, H. Sun, and Y. Yang, "Auditor: A stage-wise soft-error detection scheme for flip-flop based pipelines," in *International Conference on Smart Computing and Communication*. Springer, 2016, pp. 439–448.

[87] J. Oh and M. Kaneko, "Latency-aware selection of check variables for soft-error tolerant datapath synthesis," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E100–A, no. 7, pp. 1506–1510, July 2017.

[88] G. Saucier and A. Mignotte, *Logic and Architecture Synthesis*, ser. IFIP Advances in Information and Communication Technology. Springer US, 1995.

[89] G. Saucier, C. Duff, and F. Poirot, "State assignment using a new embedding method based on an intersecting cube theory," in *26th ACM/IEEE Design Automation Conference*, 1989, pp. 321–326.

[90] D. B. Armstrong, "A general method of applying error correction to synchronous digital systems," *Bell System Technical Journal*, vol. 40, no. 2, pp. 577–593, 1961.

[91] S. Devadas, H. K. T. Ma, A. R. Newton, and A. Sangiovanni-Vincentelli, "A synthesis and optimization procedure for fully and easily testable sequential machines," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, no. 10, pp. 1100–1107, Oct 1989.

[92] R. Leveugle, "Optimized state assignment of single fault tolerant fsms based on sec codes," in *30th ACM/IEEE Design Automation Conference*, June 1993, pp. 14–18.

[93] J. F. Meyer, "Fault tolerant sequential machines," *IEEE Transactions on Computers*, vol. C-20, no. 10, pp. 1167–1177, Oct 1971.

[94] T. Iwagaki, Y. Ishimori, H. Ichihara, and T. Inoue, "Designing area-efficient controllers for multi-cycle transient fault tolerant systems," in *2015 20th IEEE European Test Symposium (ETS)*, May 2015, pp. 1–2.

[95] H. Akasaka, S. Abe, M. Yanagisawa, and N. Togawa, "Energy-efficient high-level synthesis for hdr architectures with clock gating based on concurrency-oriented scheduling," *IPSJ Transactions on System LSI Design Methodology*, vol. 6, pp. 101–111, 2013.

[96] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. Pearson Education, 2011.

[97] P. Balasubramanian and N. E. Mastorakis, "Power, delay and area comparisons of majority voters relevant to tmr architectures," *arXiv preprint arXiv:1603.07964*, 2016.

[98] M. Mano, *Digital Design*, 3rd ed., ser. Prentice Hall international editions. Prentice Hall, 2002.

[99] NanGate Inc., "Nangate 45nm open cell library," http://www.nangate.com/?page_id=2325, accessed: Jan. 5, 2018.

Publications

[1] J. Oh and M. Kaneko, "Soft-error tolerant datapath synthesis based on speculative resource sharing in triple algorithm redundancy," *Proceedings of the Workshop on Synthesis And System Integration of Mixed Information Technologies (SASIMI 2015)*, pp. 272–277, Mar 2015.

[2] ——, "Automated selection of check variables for area-efficient soft-error tolerant datapath synthesis," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2015, pp. 49–52.

[3] ——, "Area-efficient soft-error tolerant datapath synthesis based on speculative resource sharing," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E99–A, no. 7, pp. 1311–1322, July 2016.

[4] ——, "Soft-error tolerant datapath synthesis considering adjacency constraint between components," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2016, pp. 595–600.

[5] ——, "Mixed error correction scheme and its design optimization for soft-error tolerant datapaths," in *2016 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, Oct 2016, pp. 362–365.

[6] ——, "Latency-aware selection of check variables for soft-error tolerant datapath synthesis," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E100–A, no. 7, pp. 1506–1510, July 2017.