February 2017

PhD Thesis

Low-Cost Self-Repairing Binary Signed-Digit Adder with Multiple Fault Detection and Localization

Graduate School of Chosun University

Department of Computer Engineering

Hossein Moradian Sardroudi

오류의 국지화
저비용 다중오류 자가회복
부호 이진수 가산기 구조

2017 년 2 월 24 일

조선대학교 대학원

컴 퓨 터 학 과

호세인 모라디언살드루디

Low-Cost Self-Repairing Binary Signed-Digit Adder with Multiple Fault Detection and Localization

Advisor: Prof Jeong-A Lee

This Thesis is submitted to the Graduate School of Chosun University in partial fulfillment of the requirements for the award of a PhD degree

2016 년 10 월

Graduate School of Chosun University

Department of Computer Engineering

Hossein Moradian Sardroudi

Hossein Moradian Sardroudi
의 박사학위논문을 인준함

위원장 조선대학교 교수 조범준 (인)

위 원 조선대학교 교수 문인규 (인)

위 원 조선대학교 교수 이지은 (인)

위 원 전남대학교 교수 최덕재 (인)

위 원 조선대학교 교수 이정아 (인)

2016 년 12 월

조선대학교 대학원

Dedication

I dedicate this thesis to my beloved parents, thank you for always being there for me, supporting me and encouraging me to be the best that I can be.

Acknowledgements

This research would not have been possible without the support of many people. I am heartily thankful to my supervisor, Professor Jeong-A Lee, who was abundantly helpful and offered invaluable assistance, support and guidance. Without her continued support and interest, this thesis would not have been the same as presented here. My sincere appreciation also extends to all my labmates and friends who have provided assistance at various occasions. Their views and tips were useful indeed. Also I wish to express my love and gratitude to my beloved parents and families for their supports and endless love, through the duration of my studies. Lastly, I offer my regards and blessings to all of those who supported me in any respect during the completion of the research.

Table of Contents

List of Tables

List of Figures

Abbreviations

LDT: long duration transient

MCT: multi-cycle transient

CPU: central processing unit

GPU: graphics processing unit

DSP: digital signal processing

FFT: fast fourier transform

FIR: finite impulse response

SET: single event transient

SEU: single event upset

SER: soft error rate

SDC: silent data corruption

DUE: detected unrecoverable error

OS: operating system

EEC: error correction codes

LET: linear energy transfer

TSC: totally self-checking

DMR: dual modular redundancy

BCP: Berger check prediction

SFS: strongly fault secure

FA: full-adder

ED: error detection

EC: error correction

DWC: Duplication-With-Comparison

TMR: Triple Modular Redundancy

QMR: Quadruple Modular Redundancy

SCA: Self-Checking Adder

CLA: Carry-Lookahead Adders

RCA: ripple-carry adder

CSA: carry-select adder

CSK: carry-skip adder

RCLA: ripple-carry/carry-lookahead adder (hybrid adder)

SDA: signed-digit adder

BSDN: binary signed-digit number

LSI: left-shifted inputs

RSI: right-shifted inputs

RESO: recomputation with shifted operands

TLSI: triple left shifted inputs

TRSI: triple right shifted inputs

QSD: quaternary signed-digit

Abstract

Low-Cost Self-Repairing Binary Signed-Digit Adder with Multiple Fault Detection and Localization

By: Hossein Moradian Sardroudi

Advisor: Professor Jeong-A Lee, Ph. D

Department of Computer Engineering

Graduate School of Chosun University

The advent of advanced microelectronic technologies and scale downing into nanometer dimensions has made current digital systems more susceptible to faults and increases the demand for reliable and high-performance computing. Current solutions have so far used the parity prediction scheme to increase reliability and detect fault in adder modules, but they add perceptible area overhead to the circuit. In this research, we present two new efficient methods for fault detection and localization, in addition to the full error-correction, targeting stuck-at and multi-cycle transient (MCT) faults in radix-2 signed-digit adders through a combination of time and hardware redundancy. Signed-digit adder, which eliminates carry propagation chain, can execute addition operation independent of the length of operands, in constant time. The confined carry propagation implies remarkable advantage in terms of error detection, localization, and correction. We developed a new low-cost technique, for fault-localization and error-correction, which utilizes the self-dual concept in binary signed-digit adders. In addition, we use the self-checking full adder that can identify a fault based on internal functionality to detect any fault in the adder modules. The detection of a fault is followed by input inversion, recomputation, and appropriate output inversion to correct the error and localize the fault. The error-correction method employs fault masking by utilizing the self-dual concept, which is based on the fact that in the existence of a fault, the designed technique results in a fault-free complement of the expected output when fed by the complement of its input operands. Additionally, the existence of any fault in the input lines of the adder modules can be identified by low-cost

parity checking error-detection approach, and a faulty module can be localized by comparing the faulty output from the first computation with the fault-free output from the recomputation. Higher reliability with less computational time and reduced hardware-area overhead are the distinct features of our new designs. In these methods, all of the single stuck-at and MCT faults can be localized and corrected, whereas previous approaches were unable to localize and correct with 100% reliability even with longer time durations and greater hardware cost. Based on the experimental results, the area occupied by our designs is up to 50% less, compared with the area used by previous designs. In addition to the area reduction, our design approaches result in a higher reliability with less power consumption and low time delay compared with a previous related works.

[KEYWORDS]: Reliability, Fault tolerance, Fault detection, Error correction, Fault localization, Self-Repairing, Signed-digit adder

초록

오류의 국지화 기반 저비용 다중오류 자가회복 부호 이진수 가산기 구조

호세인 모러디안 살드루디

지도교수 : 이정아

컴퓨터공학과

조선대학교 대학원

반도체 소자의 집적화 기술의 발전으로 나노미터 수준으로 축소되면서, 디지털 시스템의 결함은 더 취약해졌고, 고성능 컴퓨팅에 대한 신뢰성 요구는 더욱 더 증가하게 된다. 가산기 회로에서 신뢰성을 높이기 위한 기존의 방법에서는 패리티 예측 기법을 사용하여 오류를 탐지하지만, 이와 관련된 상당한 하드웨어 오버 헤드를 회로에 추가한다. 본 논문에서는 가산기의 오류 탐지와 오류 회복을 보다 효율적으로 구현하기 위하여, 오류의 국지화에 기반한 저비용 오류 자가회복 가산기 구조를 제시한다. 고착오류(Stuck-at Faults)와 Multi-Cycle Transient(MCT) 오류를 대상으로 부호 이진수 가산기에서 오류 검출 및 오류 위치 파악을 통한 두 가지의 효율적인 오류 자가회복 방법을 제시한다.

부호 이진수 가산기는 캐리 전달 체인이 제거되어 기존의 가산기와 달리, 덧셈을 완료하는 시간은 피연산자의 길이에 관계 없이 일정하다. 부호 이진수 가산기의 제한된 캐리 전파는 오류 검출, 오류의 지역화 및 오류회복 측면에서 주목할만한 이점을 제시한다. 본 논문에서는 결함 위치 및 오류 회복을 위해 부호 이진수 가산기에서 Self-Dual 개념을 사용하는 새로운 저비용 기술을 개발했다. 또한 전가산기(Full Adder)회로의 로직 특징을 기반으로 한 오류 자가탐지 전가산기를 이용하여 가산기 모듈의 작동오류를 탐지한다. 부호 이진수 가산기의 세부 모듈인 전가산기 회로의 작동오류가 탐지되면 입력 반전, 재연산 및 적절한 출력 반전에 의해 해당 오류를 정정하고 지역화한다. 오류 정정 방법은 Self-Dual 개념을 활용한 오류 마스킹을 사용한다. 즉, 오류가 탐지되었을 경우, 부호 이진수 가산기에서 피연산자의 입력을 보수화하면, 출력은 오류가 없을 것이며, 이 출력의 보수값이 오류가 회복된 출력이 되는 Self-Dual 성질을 이용한다. 또한, 전가산기 모듈의 입력 라인에서의 임의의 결함은 저비용 패리티 검사 오류 검출 방법에 의해 식별 될 수 있고, 오류가

있는 전가산기모듈은 첫 번째 계산의 오류 출력을 재연산 한 fault-free 출력과 비교하여 모두 지역화 할 수 있다. 부호 이진수 가산기의 입력라인 결함탐지와 전가산기 작동오류 자가탐지를 활용하여, 계산 시간 단축은 물론이고, 높은 신뢰성을 하드웨어 영역 오버 헤드 감소로 달성할 수 있었다. 기존의 방법에서 더 긴 시간과 하드웨어 비용으로도 지역화 및 오류 정정을 100% 달성할 수 없었던 반면, 고착오류 (Stuck-at Faults)와 Multi-Cycle Transient(MCT) 오류 모두 지역화 및 정정될 수 있었다. 실험 결과에 따르면, 본 논문에서 제시한 오류의 국지화 기반 다중오류 자가회복 부호 이진수 가산기 구조는 기존에 제시된 자가회복 이진수 가산기 구조보다 회로 오버헤드가 줄어서, 그 회로 크기가 최대 50% 축소되었으며, 높은 신뢰성과 저전력, 그리고 낮은 지연시간을 제공한다.

[KEYWORDS]: 신뢰성, 결함극복, 오류 검출, 오류 수정, 오류의 국지화, 오류 자가 회복 부호 이진수 가산기

1 Introduction

Experimental researches have shown that the increasing complexity of digital circuits, reduced clock cycles, and reduction in hardware (i.e. transistor) size, along with the presence of radiation and other environmental conditions, has made current hardware systems more prone to faults [1].

Moreover the incidence of multiple faults caused by a single particle hit increase because of reduction in transistor size and decreasing the distance between neighbor devices. Increasing the clock frequency of circuits to achieve high speed systems, leads to transient faults remaining longer than one cycle. These facts disqualify the use of several existing soft error moderation methods based on temporal redundancy, and necessitate the development of new fault tolerant methods to handle this different scenario. Consequently a solution that eases building trustworthy schemes by supervision errors caused by unreliable modules and mechanisms is needed [1] [2] [3] [4] [5]. In addition, a fault-tolerant characteristic is not only for mission-critical and safety systems but also for the common computing systems.

As the central modules of the processing elements, arithmetic operators are also vulnerable to diverse environmental effects. In nearly all digital components, adders are the vital components existing in the arithmetic operators. Therefore, designing a reliable adder that can detect faults and correct the errors is an important challenge and a prominent goal.

Novel low cost fault tolerance methods are proposed in this research. New techniques working at diverse concept levels, as the chosen substitute to deal with stuck-at faults and those faults produced by multi-cycle transient (MCT) that will upset CMOS devices to be manufactured in upcoming technologies.

A. Motivations

In recent the development of semiconductor technology has carried growing concerns about the reliability of systems to be deliberate using those devices, while continually providing innovative devices with unmatched speed, size, and power consumption characteristics. According to Moore's law while CMOS technology keeps developing, thus approaching the physical boundaries carry out by spending of only a few atoms to organize the device's channel [6] [7], the development of another technologies, able to take digital systems away from those bounds, come to be a huge challenge to be faced by researchers. But then again even the most hopeful alternative technologies developed so far bring alongside the similar adverse characteristic: devices manufactured using them are more susceptible to manufacturing defects and transient faults than nanoscale CMOS, making the reliability aim even more hard to be reached.

In new technologies the decreasing reliability of CMOS devices is a result of several diverse problems arising from the physical features of those devices:

• Functional temperature restrictions and the lower power consumption levied by embedded and portable systems necessities lead to the usage of lower operational voltages, which consecutively infer smaller critical charges, making the devices more vulnerable to transient pulses, meanwhile those devices can be upset by even particles with quite slight energy [8]. Accordingly, the incidence of single event upsets (SEUs) and single event transients (SETs) has been increasing in recent years, and became a concern not only for avionics applications or systems targeting space, but also for those intended for critical operations meant to be used at sea level [9]. Consistent with the international technology roadmap for semiconductors 2008 update, under 65nm, single-event upsets (soft-errors) influence field-level product reliability, not only for embedded memories, but for latches and logic also [10]. Furthermore, while several error detection and correction methods have been suggested and are

in present use to shield devices against those effects, in this manner stabilizing the soft error rate (SER) through technology nodes, the protection of combinational logic alongside transient faults is still an open issue.

• Small size devices allow the building of circuits with higher densities, where the space among neighbor devices is reduced among consecutive technology nodes. Such little spaces allow that two or more silicon devices to be hit by a single particle at the same time, by this means producing multiple concurrent faults, a risk that was not reflected until recently, and consequently is not eased by currently existing fault tolerance techniques [11] [12]. Consecutively, this multiple concurrent faults situation can lead to catastrophic concerns when well established and proven techniques in use under the single fault model are used with upcoming technologies.

• Transient faults appear only for a short time and then disappear. Therefore, detecting transient faults is more important than correcting them because after the fault disappears, the module functions correctly. Designing circuits with shorter cycle times became possible with the new fast devices, however unfortunately, at the same bound of the cycle times the duration of transient pulses does not scale [13] [14], leading to a condition where transient pulses might come to be longer than the cycle time of the circuits. By increasing the operating frequencies in recent nanometer-technology-based circuits, transient fault phenomena that previously occurred for durations as short as and considerably below one clock cycle can be now deliberated as long-duration transient (LDT) or multi-cycle transient (MCT) faults [15]. In fact, the effects of MCT faults clearly have a considerably higher chance of not being covered, therefore they also stand a more chance of making system failures [16]. Existing soft error mitigation methods either are not able to deal with this new situation due the high performance overheads that they would impose to get by such long duration transients, or do impose very high power consumption and area overheads, which will need the growth of new low cost system level modification methods [17].

• In addition all those adverse effects over reliable system operation caused by CMOS technology development, the manufacturing of digital systems is correspondingly affected by growing defect rates as a result of process deviations, higher complexity for manufacturing test as a result of augmented components density in the circuits, and further related problems. Stuck-at fault is one of the faults that can be appeared due to manufacturing problems. To cope with this new scenario new method needs to be introduced. In case of permanent stuck-at faults, circuit manufacturers apply different error detection patterns to detect faulty modules after fabrication, but it is not guaranteed that all permanent faults can be detected in the testing phase. In addition, stuck-at faults can appear in circuits afterward because of different reasons such as aging, accidental overvoltage, temperature, or other environmental conditions. Therefore detecting and correcting stuck-at faults during the time that the module is online, without interrupting the system, is important.

On the other hand complicacy in a circuitry and delay in propagation are some of the main problems that every digital circuitry has to deal with. In binary system of numbers, speed of computation relies on propagation & formulation of carry particularly when quantity of bits gets raised. Two basic approaches for reducing the delays caused by signal propagation in arithmetic operations are speeding up or eliminating the carry and borrow chains. The speeding up approach is exemplified by carry-skip and carry-lookahead designs which reduce the ripple-carry delay of O(n) to a delay of O($\sqrt{n}$) and O(log n), respectively, for n-digit operands. The second approach (eliminating the carry and borrow chains) is exemplified by signed-digit number representation methods which limit the propagation of carries and borrows at the expense of some overhead in storage and data-path width and in processing time for the initial conversion and the final reconversion. The signed-digit number system has advantages of performing fast computations with the help of characteristics of redundancy that perform carry- and borrow-free operations of addition, subtraction, multiplication and division.

The application for fast arithmetic operations is wide, including central processing unit

(CPU), graphics processing unit (GPU), cryptography, digital signal processing (DSP), fast fourier transform (FFT), finite impulse response (FIR) Filter, and applications that spend a large proportion of the time for arithmetic calculation.

B. Objectives of Research

The objectives of this research are to present new efficient procedures that will achieve single and multi-fault detection and localization, in addition to full error correction, in the binary signed-digit number (BSDN) adders in which the faulty modules can be identified with lesser area, time, and power overhead.

The objectives can be listed as:

- To study existing fault tolerance methods
- To propose new method for fault detection
- To suggest a technique for error localization
- To present a new process that will achieve full error correction
- To analyse and evaluate the proposed methods by comparing with previous methods

C. Research Scope

- Number system:
  - Radix-2 signed digit
- Fault Type:
  - A single/multi stuck-at and MCT faults
- Design Tools:
  - Xilinx Design Tools (ISE Design Suit 14.7, Vivado 2015.2)

- Implementation Language:
  - Verilog HDL
- Simulation:
  - ModelSim Student Edition 10.3c (Mentor Graphics)
  - MATLAB Simulink R2015a
- Synthesis:
  - Vivado 2015.2
  - Synopsys Design Compiler

D. Organization of the Research

The organization of this thesis is as follows. Chapter 2 gives an overview of fault types, self-checking circuits, signed-digit number system and self-checking signed-digit adders. Specific techniques for each proposed method are provided in Chapter 3. Chapter 4 discusses the performance of proposed methods and presents the area overhead, maximum timing delay and power consumption of techniques beside with comparison with previously related works. Chapter 5 provides concluding remarks and future research opportunities.

2 Related Work

In this chapter, we present the main technical terms used in the research, and discuss the related works regarding stuck-at and multi cycle transient faults. In addition signed-digit number system and previous works in self-checking signed-digit adder systems are presented in this section.

A. Failure

Failure is defined as a system malfunction that causes the system to not meet its correctness, performance, or other guarantees. A failure arises when a system differs from its normal behavior. A failure is, however, simply a special case of an error showing up at a boundary where it becomes visible to the user. This could be a silent data corruption (SDC) event, such as a change in the bank account, which the user sees. This could also be a detected unrecoverable error (DUE) caught by the system but not corrected and may lead to temporary unavailability of the system itself. For instance, if the word that was written in system memory, differs from the word is read from the same address in the memory it is failure, or, an ATM machine could be unavailable temporarily due to a system reboot caused by a radiation-induced bit flip in the hardware. Alternatively, a disk could be considered to have failed if its performance degrades by 1000x, even if it continues to return correct data.

B. Fault

User-visible *errors*, such as soft errors, are a manifestation of underlying *faults* in a computer system. Faults in hardware structures or software modules could arise from defects, imperfections, or interactions with the external environment. Examples of faults include

manufacturing defects in a silicon chip, software bugs, or bit flips caused by cosmic ray strikes.

Typically, faults are classified into three broad categories—permanent, intermittent, and transient. The names of the faults reflect their nature. Permanent faults remain for indefinite periods till corrective action is taken. Oxide wearout, which can lead to a transistor malfunction in a silicon chip, is an example of a permanent fault. Intermittent faults appear, disappear, and then reappear and are often early indicators of impending permanent faults. Partial oxide wearout may cause intermittent faults initially. Finally, transient faults are those that appear and disappear. Bit flips or gate malfunction from an alpha particle or a neutron strike is an example of a transient fault:

- Permanent faults (stuck at 1 or stuck at 0)
- Transient faults (noise on the power supply)
- Intermittent fault (a transistor that is going to break down ; one time it is working correctly and the next time not)

Faults in a computer system can occur directly in a user application, thereby eventually giving rise to a user-visible error. Alternatively, it can appear in any abstraction layer underneath the user application. In a computer system, the abstraction layers can be classified into six broad categories (Figure 2.1)—user application, OS, firmware, architecture, circuits, and process technology. Software bugs are faults arising in applications, OSs, or firmware. Design faults can arise in architecture or circuits. Defects, imperfections, or bit flips from particle strikes are examples of faults in the process technology or the underlying silicon chip.

A fault in a particular layer may not show up as a user-visible error. This is because of two reasons. First, a fault may be masked in an intermediate layer. A defective transistor—perhaps arising from oxide wearout—may affect performance but may not affect correct operation of an architecture. This could happen, for example, if the transistor is part of a branch predictor. Modern architectures typically use a branch predictor to accelerate performance but have the

ability to recover from a branch misprediction. Second, any of the layers may be partially or fully designed to tolerate faults. For example, special circuits—radiation-hardened cells—can detect and recover from faults in transistors. Similarly, each abstraction layer, shown in Figure 2.1, can be designed to tolerate faults arising in lower layers. If a fault is tolerated at a particular layer, then the fault is avoided at the layer above it. The next section discusses how faults are related to errors.

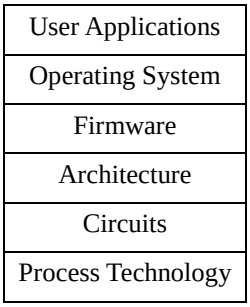

Figure 2.1 Abstraction layers in a computer system.

C. Error

Errors are manifestation of *faults*. Faults are necessary to cause an error, but not all faults show up as errors. Figure 2.2 shows that a fault within a particular scope may not show up as an error outside the scope if the fault is either masked or tolerated. The notion of an error (and units to characterize or measure it) is fundamentally tied to the notion of a *scope*. When a fault is detected in a specific scope, it becomes an error in that scope. Similarly, when an error is corrected in a given a scope, its effect usually does not propagate outside the scope. This thesis tries to use the terms *fault detection* and *error correction* as consistently as possible. Since an error can propagate and be detected again in a different scope, it is also acceptable to use the term *error detection* (as opposed to fault detection).

Three examples are considered here. The first one is a fault in a branch predictor. No fault in a branch predictor will cause a user-visible error. Hence, there is no scope outside which a

branch predictor fault would show up as an error. In contrast, a fault in a cache cell can potentially lead to a user-visible error. If the cache cell is protected with error correction codes (ECC), then a fault is an error within the scope of the ECC logic. Outside the scope of this logic where our typical observation point would be, the fault gets tolerated and never causes an error. Consider a third scenario in which three complete microprocessors vote on the correct output. If the output of one of the processors is incorrect, then the voting logic assumes that the other two are correct, thereby correcting any internal fault. In this case, the scope is the entire microprocessor. A fault within the microprocessor will never show up outside the voting logic.

Like faults, errors can be classified as permanent, intermittent, or transient. As the names indicate, a permanent fault causes a permanent or hard error, an intermittent fault causes an intermittent error, and a transient fault causes a transient or soft error. Hard errors can cause both infant mortality and lifetime reliability problems and are typically characterized by the classic bathtub curve, shown in Figure 2.3. Initially, the error rate is typically high because of either bugs in the system or latent hardware defects. Beyond the infant mortality phase, a system typically works properly until the end of its useful lifetime is reached. Then, the wearout accelerates causing significantly higher error rates. The silicon industry typically uses a technique called *burn-in* to move the starting use point of a chip to the beginning of the useful lifetime period shown in Figure 2.3. Burn-in removes any chips that fail initially, thereby leaving parts that can last through the useful lifetime period. Further, the silicon industry designs technology parameters, such as oxide thickness, to guarantee that most chips last a minimal lifetime period.

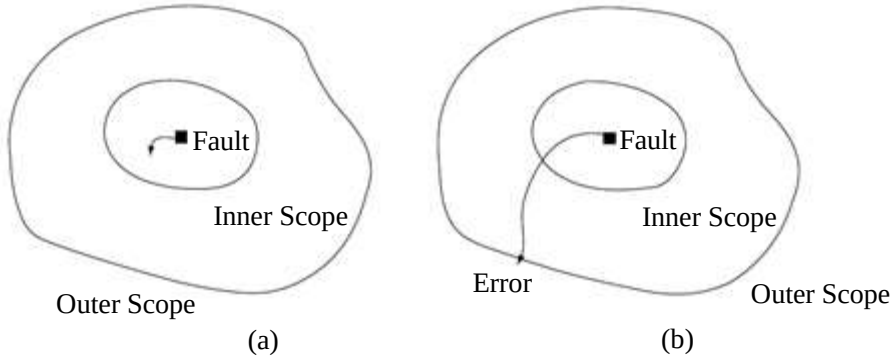

Figure 2.2 (a) Fault within the inner scope masked and not visible outside the inner scope. (b) Fault propagated outside the outer scope and visible as an error [1].

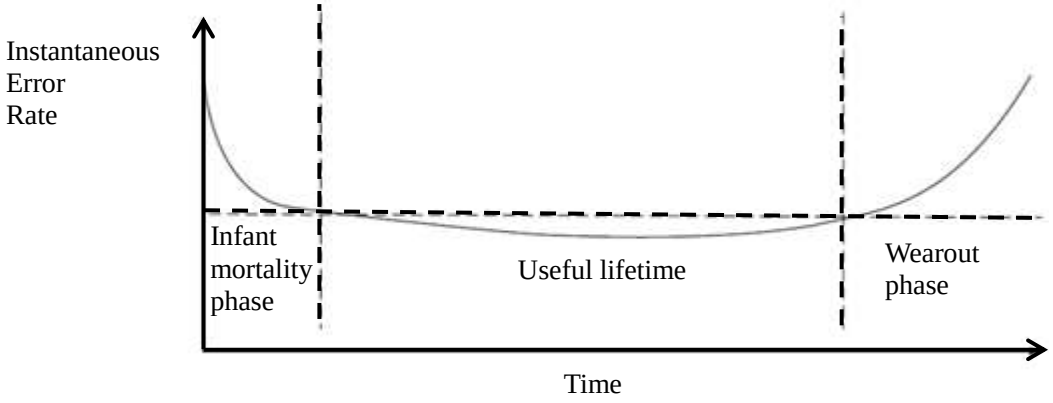

Figure 2.3 Bathtub curve presenting the connection between failure rate, infant mortality, useful lifetime, and wearout phase [1].

D. Stuck-at Fault

Stuck-at fault is a functional fault on a Boolean (logic) function implementation and individual signals and pins are assumed to be *stuck* at logical '1' and '0'. A logic stuck-at 1 means when the line is applied logic 0, it produces a logical error. A logical error means 0 becomes 1 or vice versa. The number of stuck-at faults can be single or multiple, depending on the number of lines or nodes affected. Stuck-at 1 does not mean line is shorted to VDD, and

stuck-at 0 does not mean line is grounded. It is an abstract fault model.

E. Multi Cycle Transient Fault

In addition to higher densities, the accessibility of faster circuits is a new feature of upcoming technologies that carries strong concerns to the error tolerance topic, since it has been expected that, for those technologies, even particles with diffident linear energy transfer (LET) values will creates transients lasting longer than the expected cycle time of circuits [13], [14]. In [18] for first time the undesirable impact of the effects of such multi cycle transients (MCTs) or long duration transients (LDTs) on the overhead imposed by presently used of time redundancy based error modification methods has been presented.

The first important phase in this study was the analysis of the effects of what has been entitled multi cycle transients (MCT) or long duration transient (LDT) on soft errors mitigation methods. This prediction was embedded in published works as to the effects of radiation on semiconductor devices in different technologies [13], [14], and in [18] has been challenged with the expected cycle times for inverters chains with different sizes, attained through simulation. When opposing the development of the width of radiation induced transient pulses through technologies with that of the cycle times of circuits, one could finds that, scaling trend for the width of the transient pulses is not clear while the cycle times decrease in a fairly linear form. Moreover, it has been proved that the extent of transient pulses will exceed the expected cycle time of circuits for technologies beyond the 130 nm node [18]. Table 2.1 shows this fact with data for a 10-inverter chain. The transient width figures in Table 2.1 has been extracted from [13] and [14], while those for propagation delays have been estimated using parameters from the Predictive Technology Model web site through simulation, [19].

Table 2.1 Transient Widths vs. Propagation Delay across Technologies (ps)

Technology (nm)	180	130	100	90	70	32
Transient width for LET = 10 MeV-cm^2/mg	140	210	168	n.a.	170	n.a.
Transient width for LET = 20 MeV-cm^2/mg	277	369	300	n.a.	240	n.a.
10-inverter chain propagation delay	508	158	n.a.	120	n.a.	80

n.a. = not available

In this new consequence the study of the behavior of temporal redundancy based techniques has exposed that they cannot deal with MCTs/LDTs, caused by the intolerable performance overhead that they would impose. In the other hand, area redundancy based techniques, which could cope with MCTs/LDTs, impose space and power overheads that are not matched to the requirements of a number of applications areas. From this analysis, in this research, the necessity to work at advanced abstraction levels to face this different scenario has been defined, and the search for low cost methods to detect and correct errors produced by MCTs/LDTs at system, circuit and algorithm levels has taken place.

F. Standard Adder Types

Ripple-carry adder (RCA) is simplest and the most well-known adder. The RCA design is regular, easy to implement and with the lowest area and power costs among all existing adders. Its primary limitation is the long critical path, which grows linearly (O(n)) with the word width n. Faster adders with shorter delays but increased area and power consumption also exist [20]. The well-known types are:

- carry-lookahead adder (CLA); O($log(n)$);
- carry-select adder (CSA); O($\sqrt{n}$);
- carry-skip adder (CSK); O($\sqrt{n}$); and
- ripple-carry/carry-lookahead adder (hybrid adder) (RCLA); O($log(n)$).

- signed-digit adder (SDA); O(1)

For implementing a scalable-adder design, not only the delay is important, but also area and power consumption of the adder are important too. The adder should also lend itself to *functionality segmentation*, that is, the capability to divide the adder up in parts, without give in the benefits and functionality of the adder in question; it should, therefore, have a regular structure. Clearly, in terms of area and power, the RCA is always preferred when it is deemed fast enough for its intended purpose. It also has the most regular structure of all known adder types. The CSK, as well as the CLA, and particularly the RCLA hybrid, need some more analysis, mostly because of their high-speed potential and their regular structures. CSA has no regular structure which makes scalable implementation very difficult. Previous studies such as Rabaey and [21] show that CSA overheads are significantly greater than those of the CLA however the delay increases. In conventional RCA, carry propagation significantly reduces the operation performance, especially when the size of the operands increases. Using a redundant number system eliminates or reduces the carry propagation in the addition operation [22]. The advantage of using a redundant system is to obtain an addition or subtraction operation with a complexity of O(1), which is independent of the digit length of the operand, and results in a fast adder circuit design [23].

G. Self-Checking Circuits

Self-checking usually emphasizes the detection of faults and overlooks the overhead associated with fault recovery. Concepts of self-checking and fault tolerance system can be stated as:

- A system is *fault secure* if it indicates a fault as soon as it occurs or if it remains unaffected by a fault
- A system is *self-testing* if in response to every generated fault it produces a

non-coded output

- A system will be *totally self-checking (TSC)* if it is both *fault secure* and *self-testing*

Self-checking systems require redundancy, which can be broadly classified as:

- Time-based redundancy
  - Same hardware will perform a single operation in different intervals of time
- Hardware-based redundancy
  - Using more than one hardware to produce either the same, inverted or coded output. In order to indicate the fault, the comparison between the outputs of the redundant and the actual hardware will be used.

1. Fault Detection

Errors can interrupt program execution, corrupted data, or halt a running program. Error detection approaches try to guarantee system reliability by detecting errors and turning out correct results or data. Different methods within each technique have their own advantages and disadvantages. Arithmetic codes, redundancy codes, Berger codes, and parity codes are sorts of error detection approaches. Computer architect's goals are reducing area, increasing speed, and reducing power consumption of an error detection technique [24].

Area/Time Redundancy

Redundancy infers several computations of the identical inputs for an assumed circuit. A comparison phase of the results will recognize the presence of an error if a fault happens in any of the computations. Redundancy may be attained temporally or spatially. Temporal or time redundancy is as a result of repeating computations using the same hardware, while spatial or

hardware redundancy duplicates hardware for concurrent computations.

Hardware or spatial redundancy is the most common method of redundancy [25]. Due to simultaneous error detection, increased timing for hardware redundancy is not a concern. Simultaneous error detection is the method of identifying and reporting errors while, simultaneously, acting out normal processes of the system [25]. Dual modular redundancy (DMR) is the modest hardware redundancy system. DMR duplicates the circuit and compares the outputs of the two circuits. A fault propagating through one of the circuits will flag an error when the two circuit outputs are compared [26]. DMR makes available 100% error detection, however it needs 100% overhead plus the comparator circuit.

Time or temporal redundancy is an error recognition structure that cuts extra hardware at the cost of spending more time [27]. Depending on the application of the processor, time redundancy may be more affordable than extra hardware. The simple perception of temporal redundancy is detecting the error by repetition of computations. The principal methods that use time redundancy are alternating logic, recalculation using rotated operands (RERO), and recalculation using shifted operands (RESO) [28] [29] [30].

Berger Codes

Berger codes offer error detection for logic and arithmetic operations. The only well-known method for self-checking systems other than hardware duplication and two-rail encoded systems is the strongly fault secure (SFS) Berger check prediction (BCP). A SFS BCP is further effective than a two-rail encoded scheme. For all unidirectional errors Berger codes are usable. In Unidirectional errors, both 1 to 0 and 0 to 1 errors can arise but they do not take place concurrently in a single data word [31] [32]. The binary demonstration of the number of 0's in data bits as the check code is used for the encoding system [33].

Parity Prediction

Parity prediction circuits only provide detection for arithmetic operations. The word “prediction” implies parity is “predicted”, however, parity “prediction” is not a abstract method, but it calculates the parity of the operands and outcome for comparison [1]. Amongst all acknowledged self-checking adder systems, parity prediction adders need the lowest hardware overhead [34].

Even and odd parity are two types of parity prediction. This study uses even parity to produce parity bits. If there is an odd number of 1’s in the result or operand, the parity bit is set to 1 when using even parity.

In the parity prediction circuit, the parity generator uses a series of logic XOR gates to make a parity bit. For the logic and parity circuit, the logic unit is a replication of only the logic unit in the circuit. Figure 2.4 expresses the gate level for producing an even parity bit for an operand *X*.

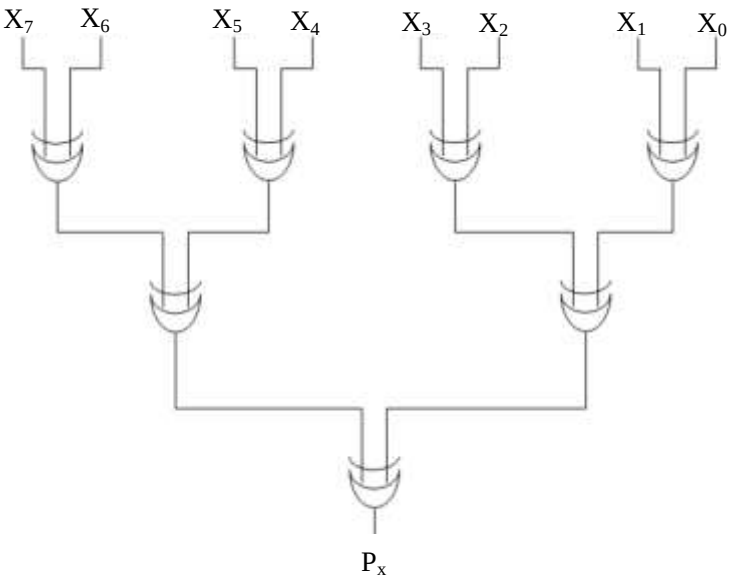

Figure 2.4 Chain of Logic XOR gates to generating even parity.

There exist several parity-prediction designs which are fault-secure for single errors. For a full-adder (FA) design any parity-prediction scheme contains the following parts:

- *PA* and *PB*: parity generators for the operands (XOR-trees)
- PC: parity generator for the internal carries
- PS: parity generator for the parity of the sum
- predictor/comparator: requires *PA*, *PB*, and *PC* to predict (compute) *PS* and compare it with the real *PS*, and signal an error on mismatch

Carry errors always manifest as multiple errors of even number. Therefore, the parity-prediction scheme would normally not be able to detect these types. [35] proposed a parity-prediction scheme with duplicated carry circuits which is fault-secure and is also covering all carry errors. When the carry circuit of every full-adder is duplicated, it is possible to compare the carry outputs of the full adder and the duplicated-carry circuit. Only one carry circuit is assumed to be erroneous at a time, since the aim is fault secureness for single errors. When the compare signals are fed to the checker, we can detect single carry errors since the maximum number of detected carry-circuit errors is one, and, thus, odd. [35] called this scheme "Duplicate carry with parity check". In fact, the scheme can be simplified by avoiding the actual comparison among the normal carries and duplicated carries. Based on the duplicated carries it is sufficient to generate parity *PC* instead of the normal ones in order to achieve the fault-secure property. In case of a carry error, the parity of the duplicated carry always contains one error less than *PC*. Furthermore, the number of errors in the sum *S* is always equal to the number of errors in *PC*. This makes the error detectable. This scheme is called "Duplicate carry with parity check II" [36].

2. Fault Localization

Fault localization, a central aspect of fault administration is a process of inferring the exact basis of a failure from a set of observed failure signs. Since faults are unavoidable in digital systems, for the accessibility, reliability, and robustness of a system, their rapid detection and isolation is critical. In large and complex systems, automating fault diagnosis is

critical. It has been a attention of study movement since the beginning of modern digital systems, which produced numerous fault localization techniques. However, the requirements carried out on fault localization methods have changed as digital systems offering new capabilities and becoming more complex.

3. Fault-Tolerant Adders

Fault tolerance means that apart from detecting (certain) errors, the circuit is also capable of correcting them. According to the literature, error detection (ED)/error correction (EC) methods to achieve fault tolerance (at the architectural level) can be facilitated through:

(i) hardware replication
(ii) time redundancy
(iii) ED/EC codes
(iv) or various combinations of the above techniques

The most widely used ED technique is hardware replication. It has been utilized for many years due to its high fault coverage. One can duplicate a circuit and feed both the circuit outputs to a comparator; in case of comparison mismatch, an error is detected. Such systems with resource replication are also called duplex or Duplication-With-Comparison (DWC) [37] [20] [35].

An example of a fault-tolerant adder using hardware replication is the TMR-adder (Triple Modular Redundancy). Here, three adder units are present, working concurrently and serving their results to a majority voter. This voter looks for a match between any two of the three inputs and outputs that result. Another fault-tolerant adder based on hardware replication is the Quadruple Modular Redundancy (QMR) adder [38]. Two adders and a comparator make a Self-Checking Adder (SCA). Since, it is not possible to attain information about which of the two adders in a SCA is faulty in case of error, the QMR has two SCAs, which the outputs of the two SCA adders are connected to a multiplexer. In total, the QMR scheme contains four adders.

The fault coverage of these fault-tolerant adders is very high. However, the power and area costs of the TMR and QMR are often excessive because of the replicated adder structure and extra checker circuitry needed.

Another type of fault-tolerant adder employing hardware redundancy is described in [39] and uses alternative computations. In every bit slice of the adder, the sum and carry are computed in two different ways and compared with each other. In case of an error, one of the redundant components is brought in to continue the calculations, saving 11.1% in transistor count compared to the QMR adder. [40] presented an asynchronous adder which achieves fault tolerance through dynamic self-reconfiguration. The adder is built based on a fault tolerant array. In case of a fault, reconfiguration logic will try different configurations until a workable instance is found. Depending on the degree of fault coverage (single faults to quadruple faults), the area overhead of these self-healing adders varies between 102% and 326% and is claimed to be lower than traditional redundancy methods (TMR and QMR). Note that this technique can be also employed in sequential adders at the cost of extra ED logic.

[41] employ the properties of radix-2, signed-digit representation to build a fault-tolerant adder. Repair is based on graceful degradation: either by recomputing the result with shifted operands and utilizing the intersection of the achieved results to recover the correct output or based on a reduced-dynamic-range approach, where the result is obtained faster but with fewer output digits. A low-cost, fault-tolerant technique for carry-lookahead adders was proposed by [42]. Correction of all single-bit and multiple-bit transient faults is claimed. The power and area costs of this method are significantly lower than those of the TMR adder or the duplicated adder with parity checking. The introduced additional delay is small, in particular compared to parity-checked adders. This technique is, however, only applicable for carry-lookahead Adders (CLAs). Valinataj represented a novel self-checking carry-lookahead adder with multiple error detection/correction capability [43]. The proposed self-checking architecture utilizes a modified parity prediction scheme combined with a partial triple modular redundancy

distributed in all of the bit slices. This scheme fits carry-lookahead adder and its arithmetic logic unit (ALU) counterpart in which the carry logic occupies a large part of the circuit. Figure 2.5 illustrated a bit slice of the proposed carry-lookahead adder.

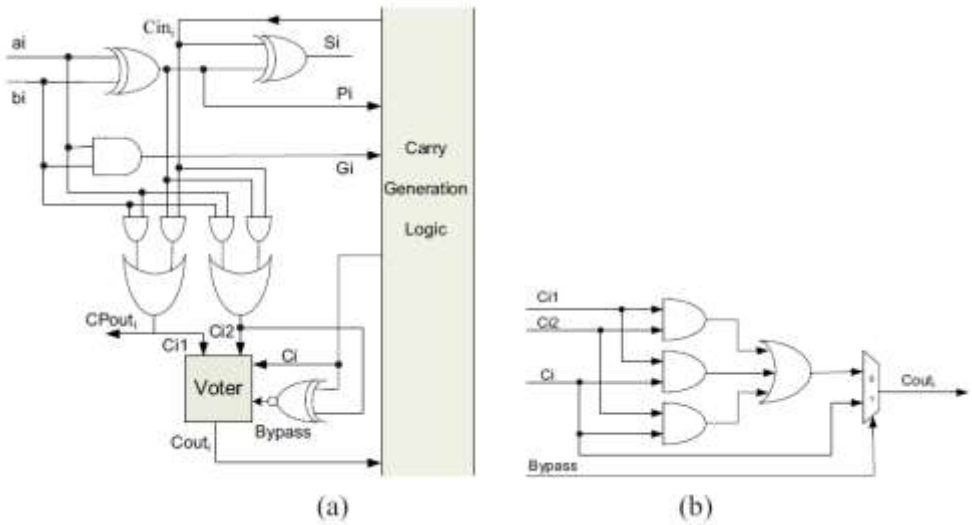

Figure 2.5 (a) A bit slice of the proposed carry lookahead adder. (b) The structure of the voter in (a).

Another fault-tolerance adder is presented in [44]. This scheme presents the design philosophy of a fault tolerant conditional sum adder that uses hot-standby technique, which is an online swapping process of faulty components of a circuit by fault-free spares without interrupting the normal operation of the system.

In [45] authors introduced new method for self-repairing ordinary adders. In an ordinary FA, self-checking is accomplished using the observed relevance among inputs, sum, and carry out. As mentioned in [45] and listed in Table 2.2 when all three inputs, namely, addend (A), augend (B), and input carry (C_{in}), are equal, the sum (Sum) and output carry (C_{out}) bits will be equal. When one of the three inputs is different, the sum and carry output bits will be complemented. These relationships can be used to design a self-checking adder with the cost of an equivalence tester (E_{qt}), as shown in Figure 2.6.

Table 2.2 Relationships between the inputs and outputs in an FA

A	B	C_{in}	Sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

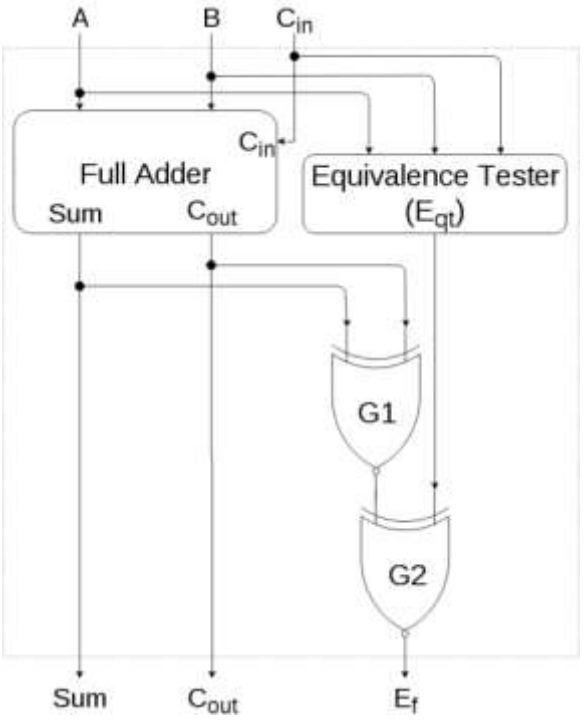

Figure 2.6 Self-checking FA [45]

The equivalence tester (E_{qt}) checks the equivalence of the inputs. G1 tests the equivalence of the outputs (Sum and C_{out}), and G2 (E_f) tests the equivalence of E_{qt} and G1.

$$Equivalence\ tester\ (E_{qt}) = \overline{(\bar{A}.\bar{B}.\overline{C_{in}}) + (A.B.C_{in})}$$

$$Error\ (E_f) = Sum\ \Theta\ Cout\ \Theta\ E_{qt} = G1\ \Theta\ E_{qt}$$

In an error-free calculation, when E_{qt} is logic 0 (all inputs are equal), the output of G1 and G2 must be logic 1 and 0, respectively; if E_{qt} is logic 1, then both G1 and G2 gates must be equal to logic 0. In any other case, a fault will be highlighted [45]. To prevent the non-detection of faults, Sum and C_{out} of the FA will not be sharing any logic; thus, any fault occurring in the internal logic associated with an individual component will only make that single component faulty and, therefore, easily detectable by comparison. Figure 2.7 shows the design of one FA without logic sharing.

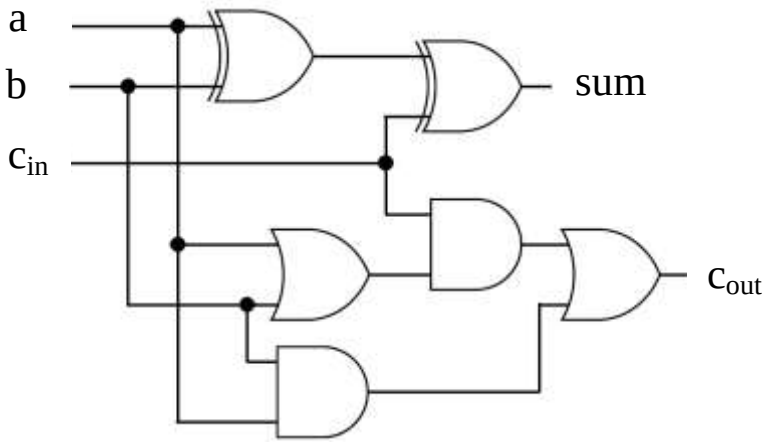

Figure 2.7 FA without logic sharing [17]

H. Signed-Digit Number systems

This part presents the elementary concepts of signed-digit arithmetic. Signed-digit systems were considered with the purpose of obtaining *totally parallel addition* [46], where carry propagation is eliminated. Carry propagation is reduced by making each digit of the subsequent sum a function of only two input numbers. This is became possible by the redundancy of the number demonstration meanwhile a appropriate intermediate demonstration of the operand digit summation, x_i+y_i, is designated so that the ending addition outcome can be produced with

half-adders that do not need nor produce carry signals. The totally parallel addition process can also be used to do subtraction operations. The subsequent parts present the principal features of signed-digit schemes and the basic principles of the resultant addition algorithm.

The algebraic value of a signed-digit number is given by

$$Z = \sum_{i=-n}^{m} z_i r^{-i}$$

where r is a positive integer called the *radix*.

In a redundant representation each number can assume more than r values, with radix r, while digits can assume exactly r values in conventional number representations. The condition of a unique representation for the algebraic value $Z = 0$ should satisfy the values of the radix and the number digits, z_i. It is formerly easy to verify that if, and only if, all digits of signed-digit representation have the value $z_i = 0$, the algebraic value Z is zero. It is also obvious that the sign of the most significant non-zero digit is the sign of the algebraic value Z. In the same way, if the sign of every non-zero z_i digit of Z changed, the result is the additive inverse of Z, the signed-digit representation of $-Z$.

Figure 2.8 shows the fully parallel addition method in the signed-digit arithmetic scheme. if two conditions are satisfied, the addition of two digits x^* and y^* is fully parallel.

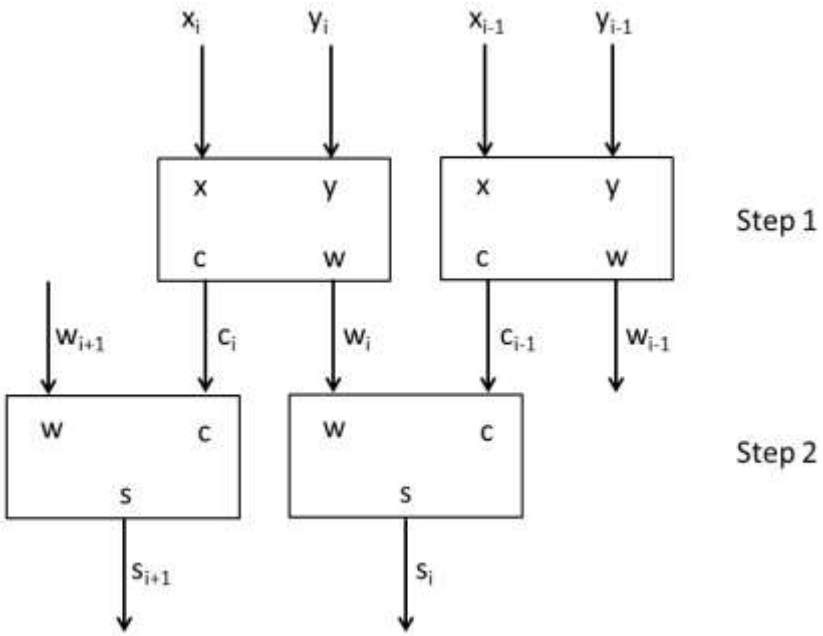

Figure 2.8 Fully parallel addition method in signed-digit number representation [46]

First, the sum (s_i) is function of the operand digits (x_i and y_i) and the carry digit (c_{i-1}) from the neighboring digit position. Second, the carry digit to the next location (c_i) is only function of the operand digits (x_i and y_i). Fully concurrent subtraction (x_i - y_i) is take in as the fully concurrent addition of x_i and the additive reverse of y_i, namely, $x_i - y_i = x_i + (-y_i)$.

Fully concurrent addition of two digits is completed in two phases, as represented in Figure 2.8. In the first phase, an intermediate carry output (c_i) and an interim sum output (w_i) are produced such that

$$x_i + y_i = rc_i + w_i \qquad (1)$$

In the next phase, the final sum digit s_i is attained as

$$s_i = w_i + c_{i-1} \qquad (2)$$

For each of the variables involved in the two-phase addition procedure the allowed and required digit values can be resultant from the definition of fully concurrent addition and from the addition procedure defined by (1) and (2). Follows are the primary results of such

derivation (see [46] for a comprehensive analysis):

1. The smallest adequate set of values for the carry digit is $c_i = \{-1, 0, 1\}$

2. For the magnitude of the intermediate sum the upper bound is $|w_i| <= r - 2$

3. For the radix value the lower bound is $r > 2$

4. For an odd radix ($r0 >= 3$) the minimum set of values for operand digits (x_i and y_i) comprises of the sequence of $r_0 + 2$ integers

$$\{-\frac{1}{2}(r_0+1), \dots, -1, 0, 1, \dots, \frac{1}{2}(r_0+1)\}$$

5. For an even radix ($r_e >= 4$) the required minimum set of values for operand digits (x_i and y_i) consists of the sequence of $r_e + 3$ integers

$$\{-(\frac{1}{2}r_e+1), \dots, -1, 0, 1, \dots, \frac{1}{2}r_e+1\}$$

Minimum sets are the only acceptable for radix-3 and radix-4 schemes. For $r > 4$, still, there is more than one valid set of digit values. The order of integers

{-a, -(a-1), …, -1, 0, 1, …, a-1, a}

satisfies the requirements for signed-digit number demonstrations, where

$$\frac{1}{2}(r_0+1) \le a \le r_0 - 1 \quad or \quad \frac{1}{2}r_e + 1 \le a \le r_e - 1$$

r_0 is an odd integer ($r_0 >= 3$), and r_e is an even integer ($r_e >= 4$). All signed-digit number representations can be designated in terms of the acceptable radix values and the acceptable z_i digit values. When $a = ½ (r_0 + 1)$ *or* $a = ½ r_e + 1$ the redundancy of a signed-digit system is *minimal*, and when $a = r_0 - 1$ or $a = r_e - 1$ the redundancy is *maximal*.

1. Signed-Digit Adders

The totally parallel addition algorithm that described in (1) and (2) are used to add two signed-digit numbers. The instructions for attaining w_i, c_i, and s_i can be determined given the set of permitted values of w_i, w_{min} and w_{max}, as follows. From algorithm (1),

$w_i = (x_i + y_i) - rc_i$,

where

$$c_i = \begin{cases} 0 & if\ w_{min} \le x_i + y_i \le w_{max} \\ 1 & if\ x_i + y_i > w_{max} \\ -1 & if\ x_i + y_i < w_{min} \end{cases}$$

and

$s_i = w_i + c_{i-1}$

For clarifying the method here we explain the addition of two signed-digit radix-10 numbers:

The allowed values for the digits are

$$w_i = \bar{5}, \bar{4}, \bar{3}, \bar{2}, \bar{1}, 0, 1, 2, 3, 4, 5$$

$$c_i = \bar{1}, 0, 1$$

$$s_i, x_i, y_i = \bar{6}, \bar{5}, \bar{4}, \bar{3}, \bar{2}, \bar{1}, 0, 1, 2, 3, 4, 5, 6$$

Digits with negative values are identified with a bar above the integer. The addition operands are $x = 4.\bar{2}\bar{5}14\bar{3}$ (value X = 3.75137), and $y = \bar{2}.\bar{3}\overline{3021}$ (value Y = 2.32979). The addition method is as follows:

augend x:	4	.	$\bar{2}$	$\bar{5}$	1	4	$\bar{3}$
addend y:	$\bar{2}$	.	$\bar{3}$	$\bar{3}$	0	2	1
Step (1):	0+2		$0+\bar{5}$	$\overline{10}+2$	0+1	$10+\bar{4}$	$0+\bar{2}$
Step (2):	0		$\bar{1}$	0	1	0	
Sum s:	2	.	$\bar{6}$	2	2	$\bar{4}$	$\bar{2}$

The resulting sum is$s = 2.\bar{6}22\bar{4}\bar{2}$which as an algebraic value S = 1.42158 = ٣،75137 - 2.32979

2. Binary Signed-Digit Number (BSDN) Addition

In binary signed-digit number system, each digit is encoded by two or more bits and

computation is achieved by means of conventional binary logic families, for example, dynamic or static CMOS.

As mentioned before the key motivation for using signed-digit number representation in an arithmetic operation is to eliminate the carry propagation chain [46]. The advantage of carry elimination is that it results in designing a high-speed adder circuit. Therefore, the delay time for adding two signed-digit numbers is independent of the word length and constant, which is the key to fast addition. This section briefly explains the theory of BSDN addition.

Any number *a* in BSDN can be represented as [41]

$$a = \sum_{i=0}^{n-1} x_i 2^i$$

where $x_i \in \{-1,0,1\}$ and *n* is the number of ternary digits. M.D. Ercegovac and T. Lang in [47] explained two methods for BSDN addition, double-recoding method and using information from a previous digit position method. Given two operands, *a* and *b*, operation of addition can be performed by one of these methods.

Double recoding method

In this method, the addition can be considered as transformed from {-2, -1, 0, 1, 2} domain to {-1, 0, 1} domain. For this recoding, first, we recode from the digit set {-2, …, 2} to an intermediate digit set (for instance, {-2, …, 1}).

$$a_i + b_i = 2h_i + s_i \in \{-2,-1,0,1,2\}$$

$$h_i \in \{0,1\}$$

$$s_i \in \{-2,-1,0,1\}$$

$$q_i = h_{i-1} + s_i \in \{-2,-1,0,1\}$$

Next, we transform the intermediate digit set into the BSDN digit set {-1, 0, 1}.

$$q_i = 2c_i + w_i \in \{-2,-1,0,1\}$$

$$c_i \in \{-1,0\}$$

$$w_i \in \{0,1\}$$

$$z_i = c_{i-1} + w_i \in \{-1,0,1\}$$

Utilizing information from the preceding digit position

In this method, first, we calculate intermediate carry (c_i) and partial sum (w_i) as $a_i + b_i = 2c_i + w_i$, as listed in Table 2.3. The result is $z_i = c_{i-1} + w_i$.

Table 2.3 Calculating partial sum and intermediate carry [48]

a_i, b_i	Previous digit position (i-1)	c_i	w_i
0, 0	-	0	0
-1, -1	-	-1	0
1, 1	-	1	0
-1, 1	-	0	0
-1, 0	(ai-1 + bi-1) < 0	-1	1
	Otherwise	0	-1
1, 0	(ai-1 + bi-1) > 0	1	-1
	Otherwise	0	1

A binary signed digit, for instance, can be encoded by two or more binary digits. In two bits representation, two bits can denote each binary signed-digit using several encoding activities. Parhami [49] proposed two encoding activities. One is "*sign"* and "*value," (s,v),* encoding. The other is "*negative"* and "*positive," (n,p),* encoding. In the *(s,v)* encoding, -1, 0, and 1 are represented by (1, 1), (0, 0), and (0,1), respectively. In the *(n, p)* encoding, -1, 0, and 1 are represented by (1, 0), (0, 0), and (0, 1), respectively. In the present research, to encode the BSDN, *(n,p)* representation that considers digit 0 representation by either (0, 0) or (1, 1) is used.

3. Self-Checking Signed-Digit Adders

In [48], a technique to devise a self-checking binary signed-digit adder concerning stuck-

at class faults using a parity checker was suggested. To evaluate the correctness of the binary signed-digit adder output, parity properties can be used. As mentioned earlier, based on the BSDN encoding representation method three different values (-1, 0, 1) can be held by digit x_i and two bits are needed for the binary representation. $P(x_i)$ is defined as the parity of x_i, which is *XOR* of the two bits that form digit x_i, and $P(X)$ as the parity of the signed-digit number X, which is *XOR* of the parity $P(x_i)$ of all digits x_i. We use this definition to define $P(A)$ and $P(B)$ as parities of the addend and augend, $P(W)$ and $P(C)$ as parities of the partial sum and intermediate carry, and $P(Z)$ as the parity of the addition outcome. According to Table 2.3, the following properties are demonstrated [48]:

Property 1.

$$P(Z) = P(C) \oplus P(W) \text{ [48]}$$

By calculating the parity of c_{i-1} and w_i, for any possible combination of c_{i-1} and w_i, we obtain $P(z_i) = P(c_{i-1}) \oplus P(w_i)$. Therefore, the following statements hold according to the associative property of the *XOR* operation:

$$P(Z) = \bigoplus_{i=0}^{n-1} P(z_i) = \bigoplus_{i=0}^{n-1} \left(P(w_i) \oplus \mathrm{P}(c_{i-1})\right)$$

$$= \bigoplus_{i=0}^{n-1} P(w_i) \oplus \bigoplus_{i=0}^{n-1} P(c_{i-1}) = P(W) \oplus \mathrm{P(C)}.$$

Property 2.

$$P(W) = P(A) \oplus P(B) \text{ [16]}$$

By evaluating the parity of a_i and b_i, we obtain $P(w_i) = P(a_i) \oplus P(b_i)$ for any possible combination of a_i and b_i. Again, according to the associative property of the *XOR* operation, we obtain

$$P(W) = \bigoplus_{i=0}^{n-1} P(w_i) = \bigoplus_{i=0}^{n-1} \left(P(a_i) \oplus \mathrm{P}(b_i)\right)$$

$$= \bigoplus_{i=0}^{n-1} P(a_i) \oplus \bigoplus_{i=0}^{n-1} P(b_i) = P(A) \oplus \mathrm{P}(B).$$

Figure 2.9 shows the architecture of this design composed of the following components:

- Binary signed-digit adder block: accepts two signed-digit numbers *a* and *b* as addend and augend, respectively, and calculates *z* as the result of the addition
- Parity Prediction module: produces the value of *P(C)*
- Error Indicator 1 output: checks Property 2 and in case of any mismatch issues an error signal
- Error Indicator 2 output: checks Property 1 and in case of mismatch issues an error signal

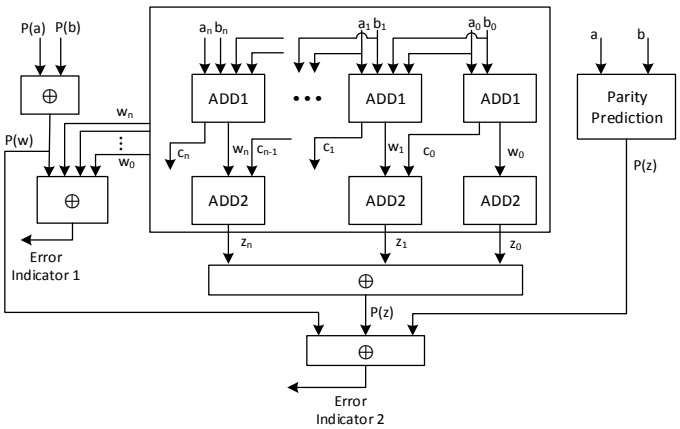

Figure 2.9 BSDN Self-checking adder [48]

In [41] authors extended the method and added error correction and localization capability using recomputation with left- and right- shifted operands. This algorithm is shown in Figure 2.10.

In this method, the stuck-at faults are classified into three types to properly localize any fault.

- Fault Type 1: Error in one of ADD1 outputs (ci, wi) or in ADD2 output (si), ending up with undesirable transforming of up to one bit

- Fault Type 2: Fault in the input's least significant bit (LSB) causes an alteration up to two bits
- Fault Type 3: Fault in the input's most significant bit (MSB) causes a variation up to three bits

Referring to above fault-type definitions, in the context of the algorithm shown in Figure 2.10, the fault-localization technique seems to be incorrect. A Type 3 (Type 2) fault causes a modification of up to three (two) digits, which means that a Type 3 (Type 2) fault can alter one or two or three (one or two) bits. Thus “Inequalities=1” (“Inequalities=2”) should signify a fault Type 1 or 2 or 3 (Type 2 or 3). We show a corrected algorithm in Figure 2.11.

To explain the error correction and fault localization shown in Figure 2.11 we completed the algorithm and added the error correction part. This completed algorithm is illustrated in Figure 2.12. In this algorithm Z and Z_F are defined as the correct and faulty outputs (first computation), respectively, and Z_{LSI} and Z_{RSI} as the outputs acquired using the left-shifted inputs (LSI) (second computation) and the right-shifted inputs (RSI) (third computation), respectively.

Let us consider two signed-digit numbers $A=\{a(0), a(1), \cdots, a(n)\}$ and $B=\{b(0), b(1), \cdots, b(n)\}$. Because by the left-shifted operation the most significant digits [$a(n)$ and $b(n)$] are lost, the equality relationship $Z(i) = Z_{LSI}(i+1)$ is invalid for $Z(n)$ and $Z(n+1)$, whereas it is valid only for $0 \leq i \leq n-1$. Similarly, least significant digits [$a(0)$ and $b(0)$] in the right-shifted operation, are lost, and their carries can also be lost. Thus, the $Z(i) = Z_{RSI}(i-1)$ equality only for $3 \leq i \leq n+1$ is valid. Figure 2.13 shows an n-digit addition process using the original, LSI, and RSI.

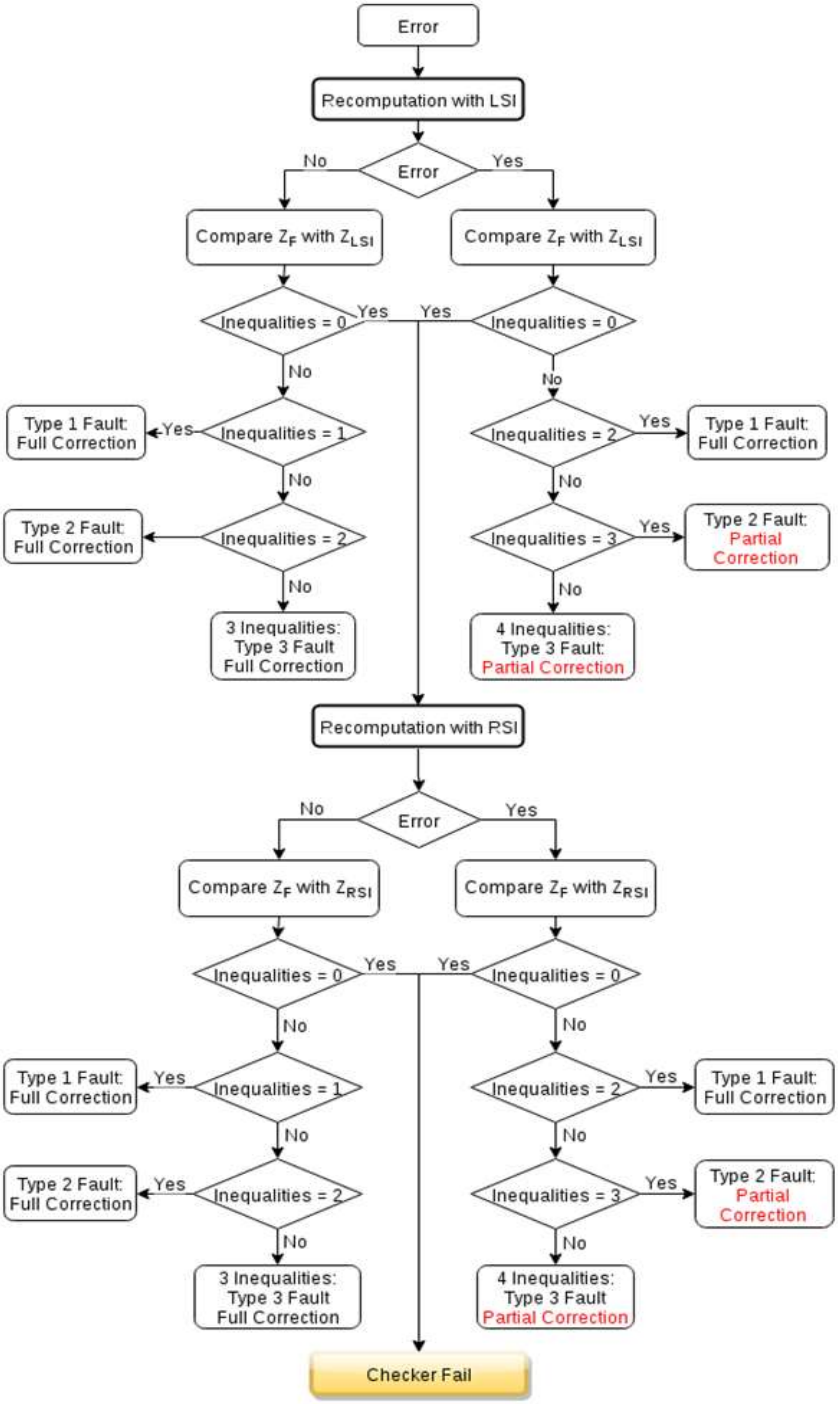

Figure 2.10 Fault detection, correction, and localization algorithm [41]

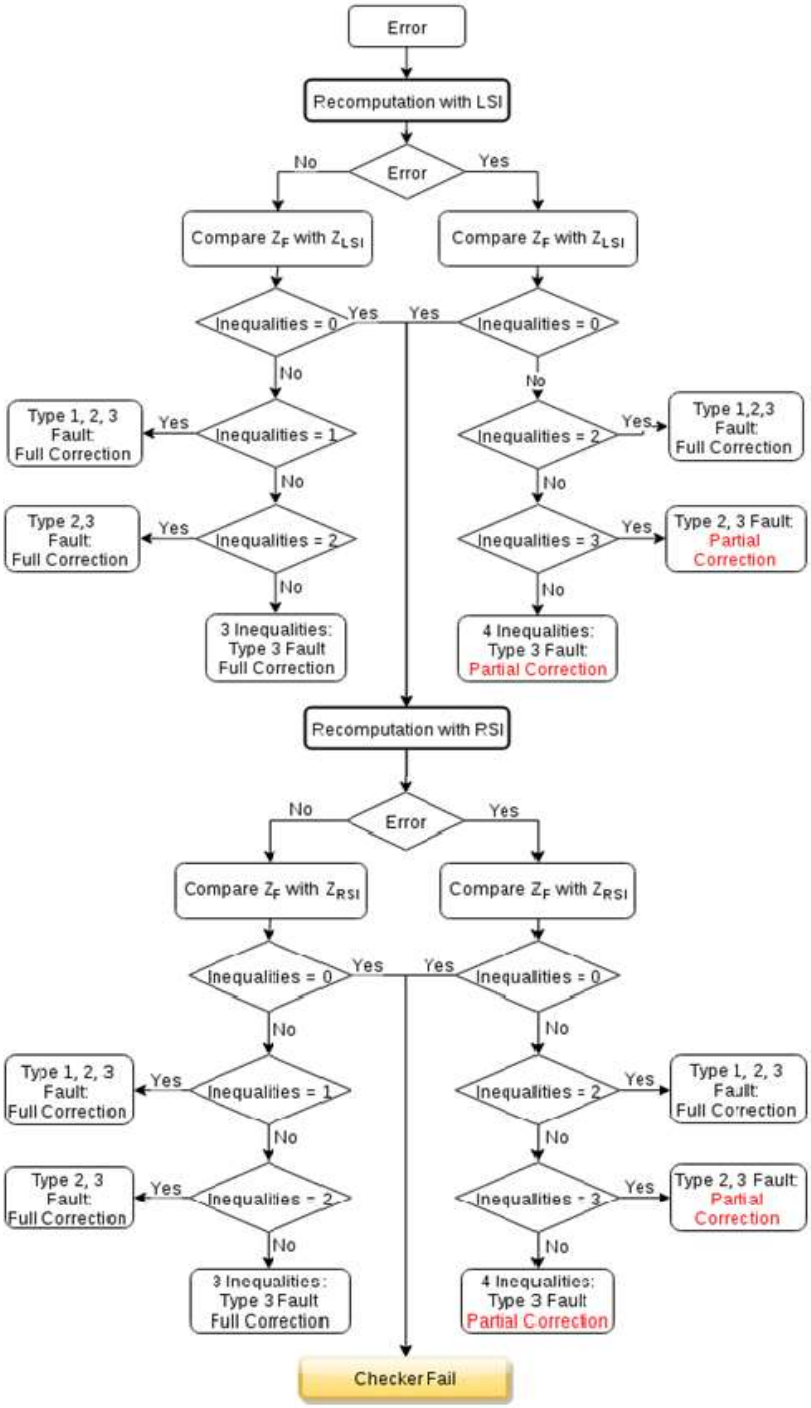

Figure 2.11 Corrected fault detection, correction, and localization algorithm proposed in [41]

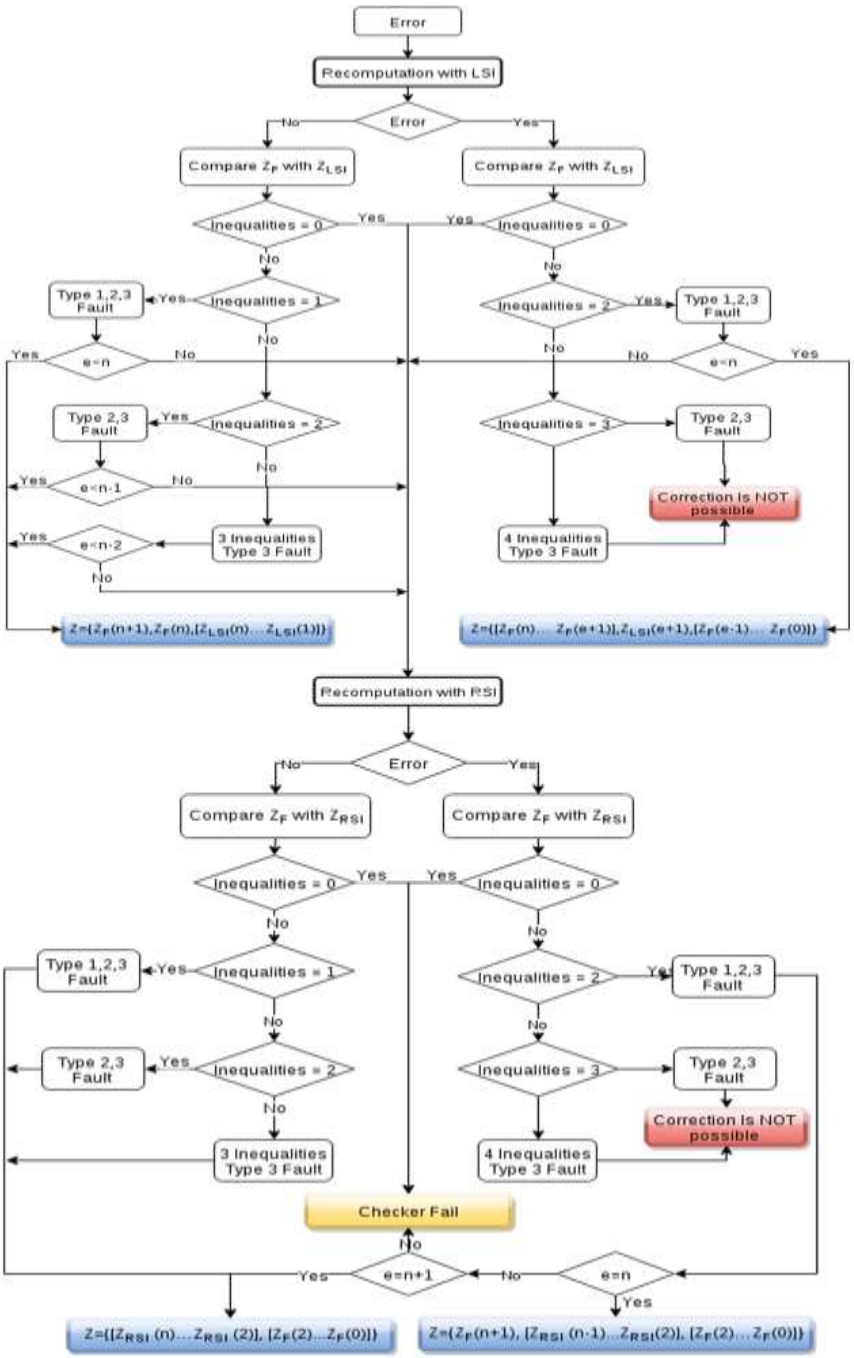

Figure 2.12 Completed and corrected fault detection, correction, and localization algorithm proposed in [41]

Position	*(n+1)*	*(n)*	*(n-1)*	...	*(2)*	*(1)*	*(0)*
A:		a_n	a_{n-1}	...	a_2	a_1	a_0
B:		b_n	b_{n-1}	...	b_2	b_1	b_0
W:		w_n	w_{n-1}	...	w_2	w_1	w_0
C:	c_n	c_{n-1}	c_{n-2}	...	c_1	c_0	-
Z:	z_{n+1}	z_n	z_{n-1}	...	z_2	z_1	z_0

a. Original Inputs

A:		a_{n-1}	a_{n-2}	...	a_1	a_0	-
B:		b_{n-1}	b_{n-2}	...	b_1	b_0	-
W:		w_{n-1}	w_{n-2}	...	w_1	w_0	-
C:	c_{n-1}	c_{n-2}	c_{n-3}	...	c_0	-	-
Z:	z_n	z_{n-1}	z_{n-2}	...	z_1	z_0	-
	$Z_{LSI}(n+1)$	$Z_{LSI}(n)$	$Z_{LSI}(n-1)$	...	$Z_{LSI}(2)$	$Z_{LSI}(1)$	$Z_{LSI}(0)$

b. LSI

A:		-	a_n	...	a_3	a_2	a_1
B:		-	b_n	...	b_3	b_2	b_1
W:		-	w_n	...	w_3	w_2	w_1
C:	-	c_n	c_{n-1}	...	c_2	c_1	-
Z:	-	z_{n+1}	z_n	...	z_3	z_2	z_1
	$Z_{RSI}(n+1)$	$Z_{RSI}(n)$	$Z_{RSI}(n-1)$	...	$Z_{RSI}(2)$	$Z_{RSI}(1)$	$Z_{RSI}(0)$

c. RSI

Figure 2.13 n-digit BSD addition

Once a parity error is detected and an error signal is issued either by Error Indicator 1 or Error Indicator 2, the error-correction and fault-localization algorithm is started. Whenever a fault is discovered, the addition is executed again using the LSI. The two results from the first (original inputs) and second (LSI) computation are compared, and the number of unequal digits and position of inequality (e) is calculated (if more than one unequal digit exists, the higher position is considered as the position of inequality, e). Depending on the position and number of unequal digits, fault localization and error correction can be done in this step, or the operation should be performed again using the RSI.

After computing with LSI, if no difference is found between Z_F and Z_{LSI}, the operation is performed using RSI. However, different cases can be considered if an inequality exists between Z_F and Z_{LSI},:

Case A: The parity is correct (no error signal is activated).

- If only one unequal digit exists between Z_F and Z_{LSI} and e<n (error is not located in position (n) or (n+1)) or if two inequalities and e<n-1 or three inequalities and e<n−2 exist, then the following relationship generates a correct output; otherwise, the RSI recomputation is required:

$$Z = \{ Z_F(n+1), Z_F(n), [Z_{LSI}(n) \ \cdots \ Z_{LSI}(1)]\}$$

Case B: The parity is wrong (at least one of the error indicators signals an error))

- If two inequalities exist and e<n, then the following relationship computes the correct output; otherwise, correction is not possible:

$$Z = \{[Z_F(n) \ \cdots \ Z_F(e+1)], Z_{LSI}(e+1), [Z_F(e-1) \ \cdots \ Z_F(0)]\}.$$

If RSI recomputation is needed, then the addition proceeds with the RSI. The two results (Z_F and Z_{RSI}) are compared, and the number of unequal digits and the position of inequality (e) are again calculated. Again, if no difference is detected between Z_F and Z_{RSI}, then the checker is expected to be faulty. However, two cases can be considered if an inequality exists between Z_F and Z_{RSI} again.

Case A: The parity is correct (no error signal is activated).

- If error signal is not activated, it means that the calculated result (Z_{RSI}) is correct, and we only need to replace the first three digits from Z_F. Therefore, the following relationship gives the correct output:

$$Z = \{[Z_{RSI}(n) \ \cdots \ Z_{RSI}(2)], [Z_F(2) \ \cdots \ Z_F(0)]\}.$$

Case B: The parity is wrong (error signal is activated)

- If two inequalities exist and e=n, then the correct result can be calculated by

$$Z = \{Z_F(n+1), [Z_{RSI}(n-1) \ \cdots \ Z_{RSI}(2)], [Z_F(2) \ \cdots \ Z_F(0)]\}.$$

- If two inequalities exist and e=n+1, then the following relationship generates the correct result:

$$Z = \{[Z_{RSI}(n) \ \cdots \ Z_{RSI}(2)], [Z_F(2) \ \cdots \ Z_F(0)]\}.$$

- If more than two inequalities exist, then correction is not possible.

Figure 2.12 shows the complete fault detection, localization, and correction algorithm in detail.

Later, Alavi and Faez [50] used the same self-checking signed-digit adder in [48], changed the correction and localization algorithm in [41], and improved the correction capability to obtain full correction using the "Recomputation with Triple-Shifted Operands" method. In this method to find the fault position, the RESO (Recomputation with Shifted Operands) technique suggested in [51], [52] is used. This method uses the same fault type classifications as [41].

In the following explanations Z is defined as the correct output and Z_F as the faulty output (the output when an error indicator signal is high) and define Z_{LS} and Z_{RS} as the correct outputs achieved by using the Left and Right Shifted Inputs (LSI, RSL) respectively. Also we mark Z_{FLS} and Z_{FRS} as the outputs obtained with the shifted operands when an error indicator sign is active. Finally Z_{TLS} and Z_{TRS} are the outputs obtained with the Triple Left and Right Shifted Inputs (TLSI, TRSI). It must be considered that the shifted operands can activate the error signal again or not depending on the fault type and its influence manner. The digits composing the operands are referred to with the corresponding lower case letter and the index i shows the position of the digit.

Because the most significant digits (a(n-1), b(n-1)) are lost with the left shift operation, then the equality relation z(i) = z_{LS} (i +1) is valid only for $0 \leq i \leq n\text{-}2$ and the equality is not valid for z(n-1), z(n). On the other hand, as the least significant digits (a(0), b(0)) are lost with the Right shift operation and their Carries can also be lost, therefore the equality relation z(i) = z_{RS} (i −1) will be valid only for $3 \leq i \leq n$ and the equality is not valid for z(0), z(1), z(2) . Once a parity error is detected, the procedure (algorithm) is started as follows:

A. First Phase

Whenever a fault is detected by the parity checker, the operation is performed once more with the LSI and two different cases can be considered:

1) The parity is correct, i.e., the output is Z_{LS} (CASE A)

2) The parity is wrong, i.e., the output is Z_{FLS} (CASE B)

CASE A: If neither of the error signals are activated, then the recomputed value Z_{LS} is correct. By comparing between $z_F(i)$ and $z_{LS}(i+1)$ for $0 \leq i \leq n\text{-}2$, the faulty digit(s) can be localized and corrected. In particular, beginning from the least significant digit, when the first inequality between $z_F(i)$ and $z_{LS}(i+1)$ is encountered, the position (i+1) is the location of the faulty digit. Besides, the number of inequalities depends on the fault type, in other word, the fault of type 1, 2, 3 produces 1, 2, 3 inequality(s) respectively. Therefore the following relation is hold:

$$\begin{cases} z_{LS}(j+1) = z_F(j), & 0 \leq j \leq n-2 \text{ for the correct digits} \\ z_{LS}(i+1) \neq z_F(i), & 0 \leq i \leq n-2 \text{ for the faulty digits} \end{cases}$$

After comparing, there will be one of the below states:

a) If the number of inequalities is zero: The fault is probably in the position of digit n-1 or n. So, the operation is performed again with the RSI (in the second phase).

b) If the position i of faulty digit is in $0 \leq i \leq$ n-4: Using the bellow relation the value of Z_F is fully corrected and the algorithm ends.

$$z(i) = \begin{cases} z_{LS}(i+1) & for\ 0 \leq i \leq n-2 \\ z_F(i) & for\ i = n-1, n \end{cases}$$

c) If the position i of faulty digit is in n-3 $\leq i \leq$ n-2: Using the relation $z(i) = z_{LS}(i+1)$ for $0 \leq i \leq$ n-2 , the value of Z_F is partially corrected, but to determine the values of z(n-1), z(n), the operation must be recomputed with the RSI (in the second phase).

CASE B: If the parity checker detects a fault after calculating of Z_{LS}, then the procedure will be continued as follows: By comparing $z_F(i)$ and $z_{FLS}(i+1)$ for $0 \leq i \leq$ n-2, the

position(s) of faulty digit(s) can be localized as CASE A. The inequalities can be generated by both an error in the computation of original result and an error in the recomputed result. Therefore depending on the occurred fault type, at most up to 4 inequalities are detected. Thus the bellow relation is truthful:

$$\begin{cases} z_{FLS}(j+1) = z_F(j), & 0 \le j \le n-2 \; for \; the \; correct \; digits \\ z_{FLS}(i+1) \neq z_F(i), & 0 \le i \le n-2 \; for \; the \; faulty \; digits \end{cases}$$

After comparing, there will be one of the following states:

a) If the number of inequalities is zero: The fault is probably in the position of digit n-1 or n. So, the operation is performed again with the RSI (in the second phase).

b) If the position i of faulty digit is in 4 $\le i \le$ n-1: Using the relation z(i) = z_F(i) for 0 $\le$ i $\le$ 3 , the value of Z_F is partially corrected, but to recover the rest of the digits the operation must be performed again with the RSI (in the second phase).

c) If the position i of faulty digit is in 1 $\le i \le$ 3: Then,

1. If the number of inequalities is one and $z_{FLS}(0) \neq 0$: i.e., the fault is occurred in digit 0, then using the below relation the value of Z_F is fully corrected and the algorithm ends.

$$z(i) = \begin{cases} z_{FLS}(i+1) & for \; 0 \le i \le n-2 \\ z_F(i) & for \; i = n-1, n \end{cases}$$

2. If the number of inequalities is equal to two and $z_{FLS}(0) = 0$: Assume that the digit f be faulty, so by using the below relation the value of Z_F is fully corrected and the algorithm ends.

$$z(i) = \begin{cases} z_{FLS}(i+1) & if \; i = f \\ z_F(i) & if \; i \neq f \end{cases}$$

3. Otherwise: The correction will be done partially and to achieve full correction the second phase of the algorithm must be completed with the TLSI.

B. Second Phase

Depending on the state has been occurred in the first phase, the second phase is continued with either RSI or TLSI. First assume that the second phase is completed with the TLSI, so the output value will be Z_{TLS}. In this case, the digits of Z_F and Z_{FLS} are compared again and as a

result one of the following states occurs:

a) If $z_{FLS}(0) = 0$ (suppose that the digit f be faulty):

$$z(i) = \begin{cases} z_{TLS}(i+3) & if\ i = f, f+1 \\ z_{FLS}(i+1) & if\ i = f+2 \\ z_F(i) & else \end{cases}$$

b) if $z_{FLS}(0) \neq 0$, i.e., the digit 0 is faulty:

$$z(i) = \begin{cases} z_{TLS}(i+3) & if\ i = 0,1 \\ z_{FLS}(i+1) & if\ i = 2 \\ z_F(i) & else \end{cases}$$

Now suppose that the second phase is continued with the RSI. As the first phase, here two different cases can also be considered:

1) The parity is correct, i.e., the output is Z_{RS} (CASE A').

2) The parity is wrong, i.e., the output is Z_{FRS} (CASE B').

CASE A': If the parity checker does not show any contradiction, thus the recomputed value Z_{RS} is correct. By comparing $z_F(i)$ and $z_{RS}(i-1)$ for $3 \leq i \leq n$, the faulty digits can be localized and corrected. Beginning from the digit 3, at the first inequality appeared between $z_F(i)$ and $z_{RS}(i-1)$, the position i is the location of the faulty digit. Therefore, the following relation is hold:

$$\begin{cases} z_{RS}(j-1) = z_F(j), & 3 \leq j \leq n\ for\ the\ correct\ digits \\ z_{RS}(i-1) \neq z_F(i), & 3 \leq i \leq n\ for\ the\ faulty\ digits \end{cases}$$

At the end of comparing, there will be one of the below states:

a) If the number of inequalities is zero: It is concluded that the adder is correct, but the parity checker is out of order. Thus the outputs are correct however the self-checking ability is lost (END of Alg.).

b) If the number of inequalities is non-zero: Using the bellow relation the value of Z_F is fully corrected and the algorithm ends.

$$z(i) = \begin{cases} z_{RS}(i-1) & for\ 3 \leq i \leq n \\ z_F(i) & for\ 0 \leq i \leq 2 \end{cases}$$

CASE B': If the parity checker detects a fault after calculating of Z_{RS}, then the procedure will be continued as follows: The faulty digits produce some inequalities between z and z_{RS}. By comparing $z_F(i)$ and $z_{FRS}(i-1)$ the position of faulty digit can be localized as CASE A'. After comparing, there will be one of the following states:

a) If the number of inequalities is zero: It is concluded that the adder correct, but the parity checker is out of order. Thus the outputs are correct however the self-checking ability is lost (END of Alg.).

b) If there are two inequalities and the position f of faulty digit is 4 or 5: Using the following relation, the value of Z_F is corrected completely and the algorithm is finished.

$$z(i) = \begin{cases} z_{FRS}(i-1) & if\ i = f \\ z_F(i) & else \end{cases}$$

c) Otherwise: The correction will be done partially and to achieve full correction the third phase of the algorithm must be completed with the TRSI.

C. Third Phase

In this phase the operation is carried out again with the TRSI (Triple Right Shifted Inputs). Therefore, the output is Z_{TRS} . The digit of Z_F and Z_{FRS} are compared once more and by using the bellow relation the digits of Z_F is corrected completely. At the end, the algorithm is completed.

$$z(i) = \begin{cases} z_{FRS}(i-1) & if\ i = f \\ z_{TRS}(i-3) & if\ i = f+1, f+2 \\ z_F(i) & else \end{cases}$$

Here this point is essential to mention that, not the algorithm will always be continued to the third phase, but it depends on the occurred fault type. In other word, in the best situation, the algorithm terminates in the first phase and the value of output is fully corrected while in the worst case, it's needed the algorithm to be continued to the third phase.

Although this method can have full correction but it requires up to four times recomputation using shifted inputs to be able to localize a fault and correct the error, and it

incurs extra time and area overhead and makes the original represented algorithm more complex.

3 Proposed Designs

Our proposed techniques employ the signed-digit number encoding and self-checking adder implementation as [41]. To localize the fault and correct errors, we use altering logic [30], which is the so-called I/O inversion [53] method, instead of the recomputation using the shifted-operand method [28] based on the *self-dual* concept. A logic function is self-dual if and only if $f(x) = \sim f(\sim x)$, where f is a logic function, x is a vector of the logic variables, and $\sim x$ is the 1's complement of x.

The correction method is based on the fact that if one stuck-at-0 (1) or MCT fault occurs in one line of a circuit, the faulty line is permanently set to 0 (1) and flips from 1 (0) to 0 (1). However, when the line value is 0 (1), the fault will be masked. We used this property and masked the fault by recomputation using the complemented inputs when any of the error indicators signals a fault. Figure 3.1 shows this property.

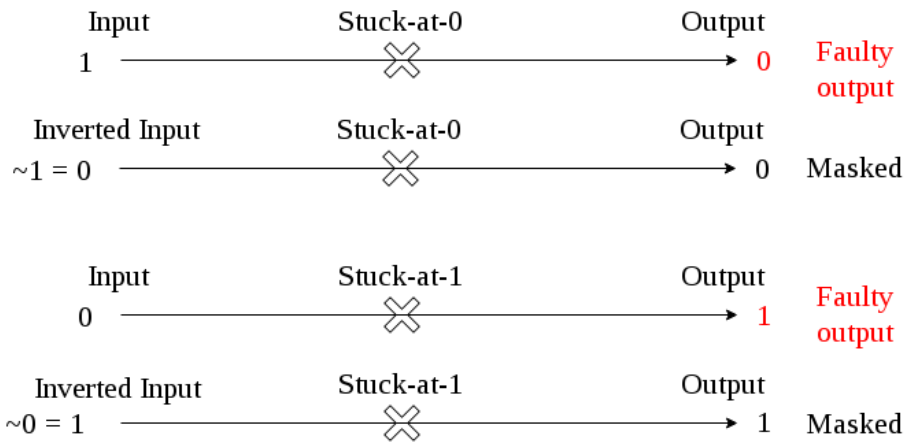

Figure 3.1 Masking of a fault using complemented input

In a symmetric signed-digit number, we can compute the 1's complement by changing the sign of each digit from positive to negative, and vice versa. It is worth mentioning that the digit "0" must be left unchanged.

$$a = 2\bar{1}0\bar{3}7 \quad (18977)_{10}$$

$$\bar{a} = \bar{2}103\bar{7} (-18977)_{10}$$

Consequently, the inversion of each number in BSDN can be calculated by only changing 1 to -1 and -1 to 1. For example

$$a = 10\bar{1}10\bar{1}\bar{1}0 \qquad (106)_{10}$$

$$\bar{a} = \bar{1}01\bar{1}0110 \quad (-106)_{10}$$

Using the BSDN encoding method representation in Chapter II, inversion (calculation of 1's complement) of BSDN is straightforward; it is computed in a manner similar to the conventional binary number 1's complement. The BSDN 1's complement calculation is listed in Table 3.1. In addition, we assume that 0 is represented by either 00 or 11.

Table 3.1 BSDN Complement

a	$\bar{a}$
01 (+1)	10 (-1)
10 (-1)	01 (+1)
00 (0)	11 (0)
11 (0)	00 (0)

Therefore, to compute the BSDN 1's complement, all bits should be inverted as shown below:

$$a = 1\ 0\ \bar{1}\ 1\ 0\ \bar{1}\bar{1}\ 0 \rightarrow \quad 01 \quad 00 \quad 10 \quad 01 \quad 00 \quad 10 \quad 10 \quad 00$$

$$\bar{a} = \bar{1}\ 0\ 1\ \bar{1}\ 0\ 1\ 1\ 0 \rightarrow \quad 10 \quad 11 \quad 01 \quad 10 \quad 11 \quad 01 \quad 01 \quad 11$$

Suppose *a* and *b* are the addend and augend in a BSDN system, respectively, and *s* is the result ($s = a + b$). Then, following the calculation rule listed in Table 2.3, c_i and w_i can be

computed in such a manner that $a_i + b_i = 2c_{i+1} + w_i$, and the sum value s_i is obtained using $s_i = c_i + w_i$. According to the property of self-duality, if $\bar{s} = \bar{a} + \bar{b}$ only, then this addition technique is self-dual. In Table 2.3, the first four rows denote that inversion of the addend (a) and augend (b) will indicate inversion of partial sum (w) and intermediate carry (c), as listed in Table 3.2.

Table 3.2 Intermediate Carry and Partial Sum, First Four Rows

ai, bi	ci	wi	$\bar{a_i}, \bar{b_i}$	$\bar{c_i}$	$\bar{w_i}$
0, 0	0	0	0, 0	0	0
-1, -1	-1	0	1, 1	1	0
1, 1	1	0	-1, -1	-1	0
-1, 1	0	0	1, -1	0	0

For the last two rows in Table 2.3, if adder inputs *(a, b)* are inverted, the outputs *(c, w)* are subsequently inverted, as listed in Table 3.3. Consequently, the calculation function is self-dual for the partial sum (w) and intermediate carry (c).

Table 3.3 Intermediate Carry and Partial Sum, Last Two Rows

a_i, b_i	Previous digit position (i-1) a_{i-1}, b_{i-1}		c_i	w_i	$\bar{a_i}, \bar{b_i}$	Previous digit position (i-1) $\bar{a_{i-1}}, \bar{b_{i-1}}$		$\bar{c_i}$	$\bar{w_i}$
-1, 0	$(a_{i-1} + b_{i-1}) < 0$	0, -1	-1	1	1, 0	0, 1	$(\bar{a_{i-1}} + \bar{b_{i-1}}) > 0$	1	-1
		-1, -1				1, 1			
	Otherwise	0, 0	0	-1		0, 0	Otherwise	0	1
		0, 1				0, -1			
		1, 1				-1, -1			
		1, -1				-1, 1			
1, 0	$(a_{i-1} + b_{i-1}) > 0$	0, 1	1	-1	-1, 0	0 ,-1	$(\bar{a_{i-1}} + \bar{b_{i-1}}) < 0$	-1	1
		1, 1				-1, -1			
	Otherwise	0, 0	0	1		0, 0	Otherwise	0	-1
		0, -1				0, 1			
		-1, -1				1, 1			
		1, -1				-1, 1			

Lastly, Table 3.4 demonstrates that $s_i = c_i + w_i$, which is considered a carry-free addition

is also self-dual.

Table 3.4 Carry Free Addition

ci, wi	si	$\overline{c_i}, \overline{w_i}$	$\overline{s_i}$
0, 0	0	0, 0	0
0, 1	1	0, -1	-1
0, -1	-1	0, 1	1
1, -1	0	-1, 1	0

As a result, all output (the partial sum, final sum, and intermediate carry) values will be inverted if all input (addend and augend) values are inverted. Therefore, the proposed BSDN addition function is self-dual ($\bar{s} = \bar{a} + \bar{b}$)

1. Self-checking binary signed-digit adder using parity prediction (SBSA-PaP)

The correcting and locating of a stuck-at or MCT faults by the proposed BSDN adder can be done using the self-dual property. To do this, we first, using normal inputs calculate the addition operation. If any error is detected by the error indicators, 1's complement of the inputs is used to recompute the operation. In standard functioning, in either ADD1/ADD2 inputs or outputs in Figure 2.9, occurrence of any stuck-at or MCT faults will root to the bit flipping i.e. 0 to 1 or 1 to 0. Earlier it was established that, with inverted inputs, recomputation of an operation will facade any plausible error, and the calculated sum will be correct because all 0's become 1's and all 1's become 0's (10 to 01, 01 to 10, 11 to 00, and00 to 11).Therefore, a right value will be shown by the bit where the subsequent stuck-at or MCT faults aroused. Following recalculation, if an error is signaled again, it is established that the checker fail or more than one stuck-at or MCT faults has occurred. However, if there is no trigger of the error signal, then the recomputed outcome is the 1's complement of the correct end result, and applying any further error-correction procedure is not necessary. The localization is almost identical to the

method in [41]. Fault is localized based on fault types (Type 1, Type 2, and Type 3) as described in Chapter II.H.3. Figure 3.2 illustrates the proposed method.

According to the algorithm shown in Figure 3.2, if any of the error indicators, signal an error, the addition operation will continue with recomputation using inverse inputs. After recomputation, the outputs of error indicators are checked. Here, two cases can be considered:

Case A (Any of the error indicators signal an error again): The outputs of the first and second computations are compared. If all corresponding bits are inverses of each other (Equalities = 0), then we can conclude that the addition output is correct and the signaled errors are because of checker failure. Otherwise, in case of finding any corresponding equal bits between the results that are obtained from first and second computations, we conclude that there are more than one stuck-at or MCT fault in the adder circuit.

Case B (After recomputation, none of the error indicators signal error): Again, the outputs of the first and second computations are compared. If all corresponding bits are inverses of each other (Equalities = 0), then we can conclude that the addition output is correct and the signaled error (in first computation) was because of a transient fault. However, if there are equal corresponding bits (Equalities > 0), then based on the number of equal bits, we can localize the faulty module as illustrated in the algorithm.

To explain the fault localization and error correction algorithm in detail, we define Z as the correct output and Z_F as the faulty output, respectively, in the first computation and Z_{INV} and $Z_{F\text{-}INV}$ as the correct and faulty outputs, respectively, obtained using the inverse inputs in the recomputation.

Suppose $Z_F = z_F(n)\ldots z_F(i)\ldots z_F(0)$ is the faulty output calculated in the first step and $Z_{INV} = z_{INV}(n)\ldots z_{INV}(i)\ldots z_{INV}(0)$ is the result of the calculation with the inverted inputs in the second step. For all error-free bits, $z_F(i)$ is the inverse of $z_{INV}(i)$, but for the erroneous bits, $z_F(i)$ is equal to $z_{INV}(i)$. Therefore, by counting the number of equal bits between Z_F and Z_{INV}, we can locate the source of the error.

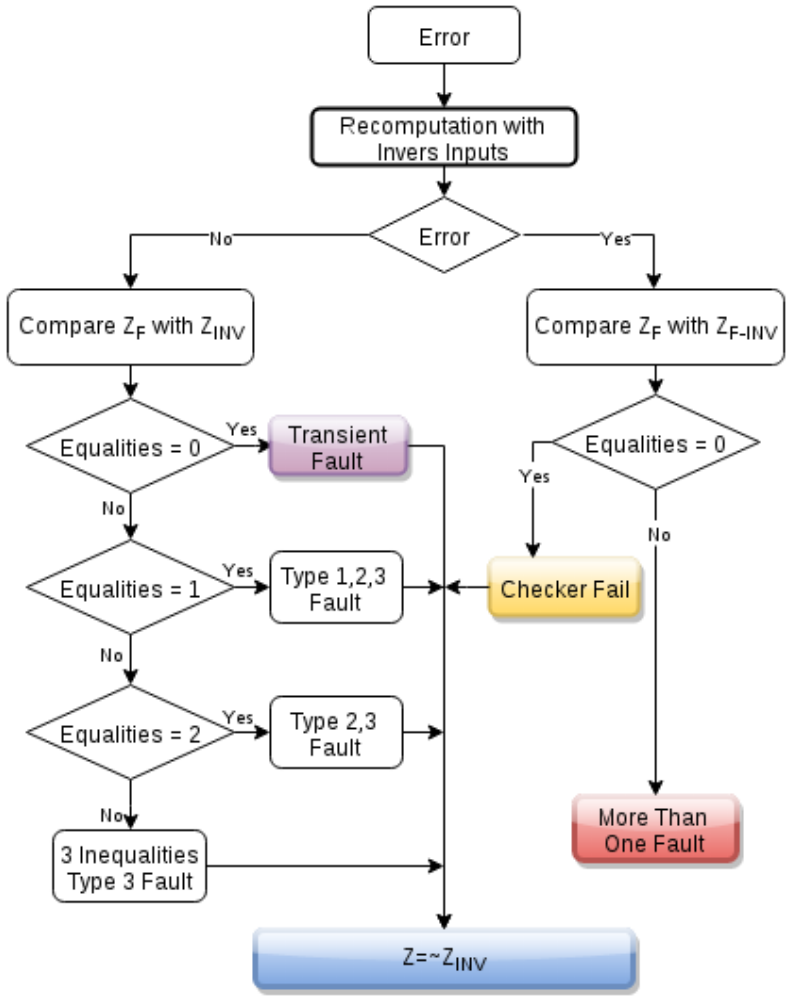

Figure 3.2 Fault detection, correction, and localization algorithm for SBSA-PaP implementation

The explained method uses the same self-checking signed-digit adder as [41]. In order to come up with efficient design we aim to use different self-checking binary signed-digit adder beside with self-dual property.

2. Self-checking binary signed-digit adder

Different designs have been presented for radix-2 signed-digit adder implementation [54] [55] [56] [57] [58] [47]. In the present study, we use the presented method in [47]. This

implementation uses the double-recoding method. It comprises two levels of FAs, as shown in Figure 3.3.

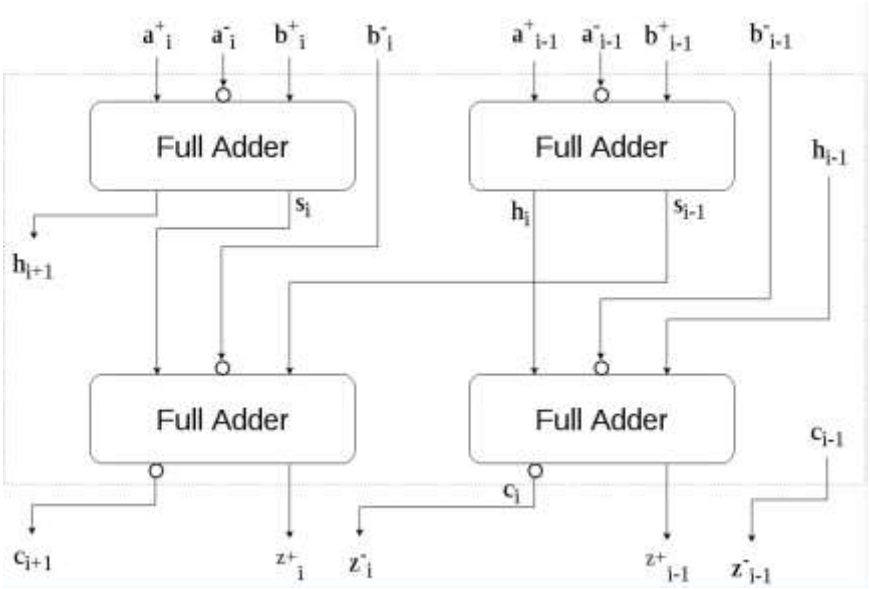

Figure 3.3 Binary signed-digit adder [47]

According to the design in Figure 3.3 we can implement the binary signed-digit adder using a conventional FA. To implement new self-checking radix-2 signed-digit adder, we use the self-checking FA in [45]. We replace the ordinary FAs in Figure 3.3 with the self-checking FA as illustrated in Figure 2.6. Figure 3.4 shows the design of a self-checking adder for BSDN. In this design, a fault in any adder activates *E1* or *E2* error signals, and the faulty module can be localized.

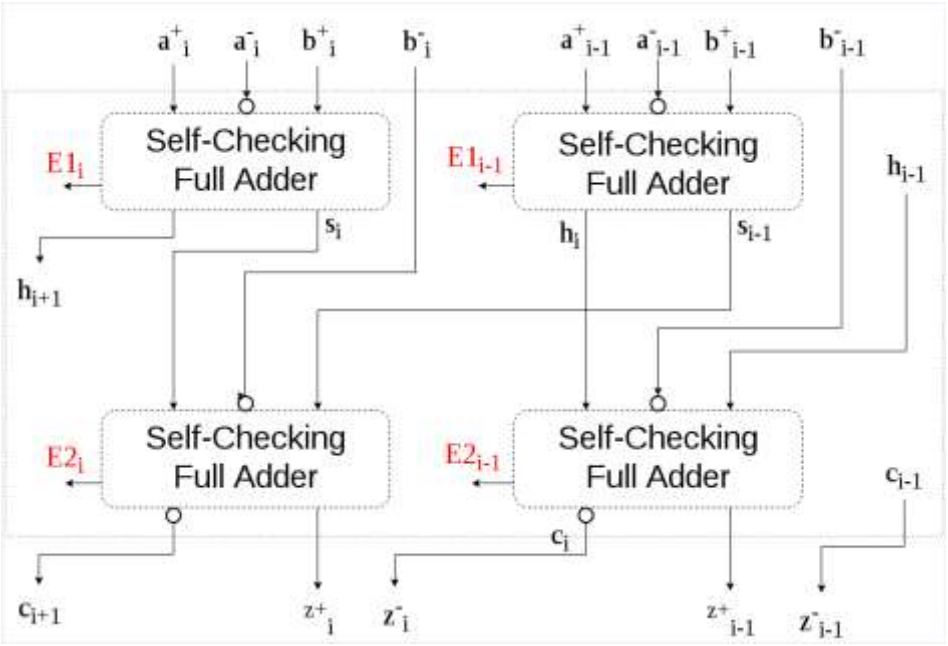

Figure 3.4 Binary signed-digit adder using a self-checking FA

To protect the input bits, we use a parity checking technique. Here, we have two assumptions. First, we only have access to the pre-calculated parity of the addend and augend, and we do not have access to each input digit. Second, we do not have pre-calculated parity for the input digits, but we have individual access to all digits of the addend and augend.

3. Self-checking binary signed-digit adder using pre calculated input parities (SBSA-PCP)

In any FA, the parity of the input bits is equal to the parity of the sum. Therefore, to reduce the hardware size, we can use the sum output bit instead of calculating the parity of the three input bits.

$P(A, B, C_{in}) = P(Sum)$

In addition, according to Table 3.5, the parity of one ordinary number is equal to the parity

of the corresponding signed-digit number. Hence, we can utilize the following relationships, where *A* and *B* are two *n*-bit numbers (addend and augend). $a_0, a_1, \ldots, a_{n-1}$ and $b_0, b_1, \ldots, b_{n-1}$ are individual binary bits representing *A* and *B*, respectively, and *n* is the total number of bits representing each of these variables. a_i^+, a_i^- and b_i^+, b_i^- are signed-digit representations of a_i and b_i bits, respectively:

Table 3.5 Parity of an ordinary number and the corresponding signed-digit representation based on the explained encoding method

a	Signed digit representation (a^+ a^-)	P(a)	P(a^+ a^-)
1	01	1	1
-1	10	1	1
0	00 / 11	0	0

$A = \{ a_0, a_1, \ldots, a_{n-1}\}$

$P(A) = P(a_0, a_1, \ldots, a_{n-1})$

$P(a_i) = P(a_i^+, a_i^-)$

$P(A) = P(a_0^+, a_0^-, a_1^+, a_1^-, \ldots, a_{n-1}^+, a_{n-1}^-)$ (1)

$P(B) = P(b_0^+, b_0^-, b_1^+, b_1^-, \ldots, b_{n-1}^+, b_{n-1}^-)$ (2)

$P(A, B) = P(a_0^+, a_0^-, b_0^+, b_0^-, a_1^+, a_1^-, b_1^+, b_1^-, \ldots, a_{n-1}^+, a_{n-1}^-, b_{n-1}^+, b_{n-1}^-)$. (3)

As mentioned earlier, in one FA, instead of calculating the parity of three inputs, we can use the sum bit. Therefore, as shown in Figure 3.4, we can use s_i as the parity of the adder inputs, but we must note that one of the adder inputs is complemented (a_i^-). Thus, we can obtain the equality

$$P(a_i^+, \overline{a_i^-}, b_i^+) = P(s_i). \quad (4)$$

We add $\overline{b_i^-}$ to both sides of the equality.

$$P(a_i^+, \overline{a_i^-}, b_i^+) + P(\overline{b_i^-}) = P(s_i) + P(\overline{b_i^-})$$

$$P(a_i^+, \overline{a_i^-}, b_i^+, \overline{b_i^-}) = P(s_i, \overline{b_i^-}) \quad (5)$$

In the parity, we have

$$P(\bar{x}) = \sim P(x).$$

Therefore:

$$P\left(a_i^+, \overline{a_i^-}, b_i^+, \overline{b_i^-}\right) = \sim P\left(a_i^+, a_i^-, b_i^+, \overline{b_i^-}\right) = \sim[\sim P(a_i^+, a_i^-, b_i^+, b_i^-)] =$$

$$P(a_i^+, a_i^-, b_i^+, b_i^-) \qquad (6)$$

Referring to Eq. (5) and (6):

$$P(a_i^+, a_i^-, b_i^+, b_i^-) = P(s_i, \overline{b_i^-})$$

$$\Rightarrow P(a_i, b_i) = P(s_i, \overline{b_i^-})$$

$$\Rightarrow P(a_0, a_1, \dots, a_{n-1}, b_0, b_1, \dots, b_{n-1}) = P(s_0, s_1, \dots, s_{n-1}, \overline{b_0^-}, \overline{b_1^-}, \dots, \overline{b_{n-1}^-})$$

$$\Rightarrow P(A, B) = P(s_0, s_1, \dots, s_{n-1}, \overline{b_0^-}, \overline{b_1^-}, \dots, \overline{b_{n-1}^-}) \qquad (7)$$

Referring to Eq. (7), by obtaining the parity of addend (*A*) and augend (*B*), any odd number of faults in the input lines can be detected by calculating the parity of the intermediate sum (*s*) and the augend LSB (b^-). Figure 3.5 shows this design.

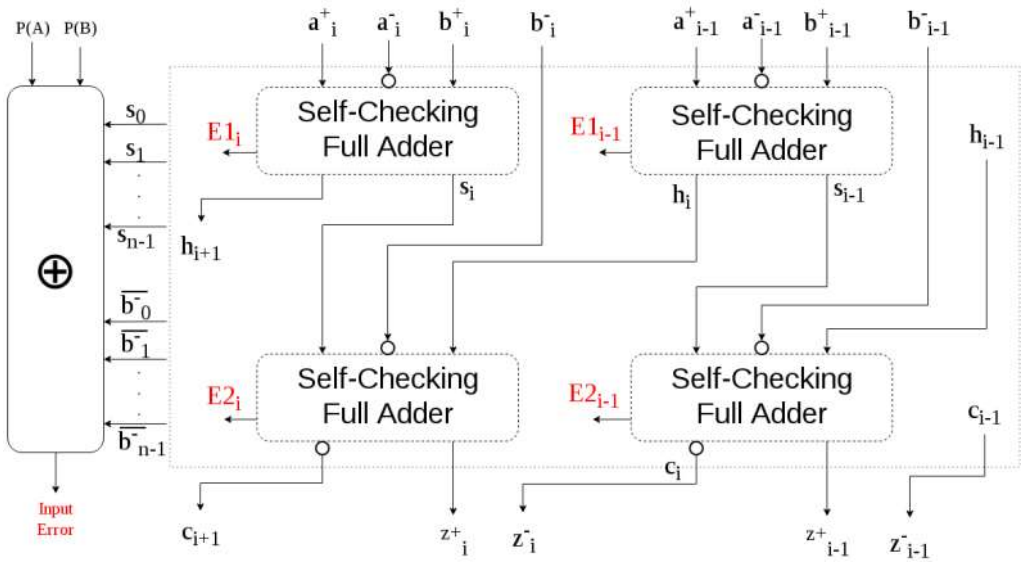

Figure 3.5 SBSA-PCP

4. Self-checking binary signed-digit adder using input bit parities (SBSA-IBP)

For the second assumption, if we can access all digits in the addend and augend, we can calculate the parity of each signed-digit number, e.g., $P(a_i^+ a_i^-)$ and compare it with the parity of the corresponding ordinary bit, e.g., $P(a_i)$. Therefore, any fault in the inputs can be detected and localized. Figure 3.6 shows this design.

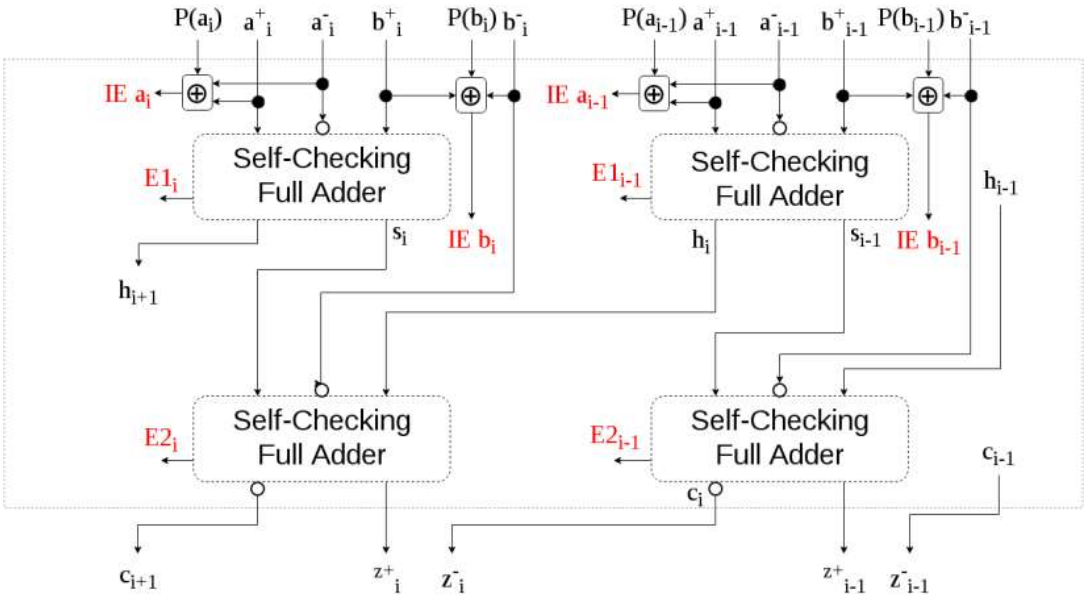

Figure 3.6 SBSA-IBP

5. Error-correction and fault-localization method in SBSA-PCP and SBSA-IBP

To correct errors, here also we use the recalculation based on the self-dual concept [53], [59], [30].

As reported in [53] and by referring to the truth table of FAs listed in Table 3.6, both the sum and output-carry functions are self-dual, which means that inverting all FA inputs inverts

the value of both sum and output-carry. As explained earlier, the BSDN adder design in this study consists only of FAs. Therefore, the described BSDN adder is also self-dual, which means that if we complement the BSDN adder inputs, all intermediate values and outputs will also be complemented. Therefore, if a stuck-at or MCT fault occurs in the input lines, FA modules, or output lines and causes flipping of the bit value from *1* to *0* or *0* to *1*, it will be masked when we apply the complemented value. Therefore, the recalculated result is the inverted correct result. By inverting this result, we can obtain the correct final output.

Table 3.6 Truth table for FA with original and complemented inputs

A	B	C_{in}	Sum	C_{out}	$\overline{A}$	$\overline{B}$	$\overline{C_{in}}$	$\overline{Sum}$	$\overline{C_{out}}$
0	0	0	0	0	1	1	1	1	1
0	0	1	1	0	1	1	0	0	1
0	1	0	1	0	1	0	1	0	1
0	1	1	0	1	1	0	0	1	0
1	0	0	1	0	0	1	1	0	1
1	0	1	0	1	0	1	0	1	0
1	1	0	0	1	0	0	1	1	0
1	1	1	1	1	0	0	0	0	0

To localize the fault in our first design (SBSA-PCP), if the fault is located in one of the adder modules, we can localize it by referring to the adder module number that issued the error signal because each adder has its own error signal. However, if the fault is located in one of the input lines, we can locate the faulty input module by comparing the faulty result obtained in the first calculation and correct the inverted result obtained in the recomputation. Figure 3.7 shows the algorithm for the error correction and fault localization according to this design. In this algorithm, we define Z as the correct output and Z_F as the faulty output in the first computation and Z_{INV} and $Z_{F\text{-}INV}$ as the correct and faulty outputs, respectively, obtained using the inverted inputs in the recomputation.

By using normal operands, we first perform the addition operation. After the calculation, if any of the error indicators detects and signals error, the operation will be followed by recomputation using the inverse operands. During normal operation, if any stuck-at or MCT fault occurs either in the self-checking FA input or output lines, that fault will cause the flipping bit from *1* to *0* or *0* to *1*. We have earlier confirmed that if the operation is recalculated using the 1's complemented operands, any possible fault will be covered, and the calculated sum will be correct because all 0's become 1's and all 1's become 0's (*01* to *10*, *10* to *01*, *00* to *11*, and *11* to *00*). Therefore, the bit will show the correct value where the corresponding stuck-at or MCT fault occurred. If any of the error indicators again issues an error after the recomputation, by comparing the two obtained results from the first and second calculations, if the results complement each other, we can conclude that a checker failure occurred; otherwise, more than one stuck-at or MCT fault that at least one of them was masked in first calculation. However, if no error signal is activated after the recomputation, by referring to the first error signal source, if it is input error, we compare the two calculated results from the first and second computations and determine the faulty module number. Otherwise, if the error is issued by the adder module, we report the adder module number as a faulty module. The recomputed result is the 1's complement of the correct result, and applying any further error-correction procedure is not necessary.

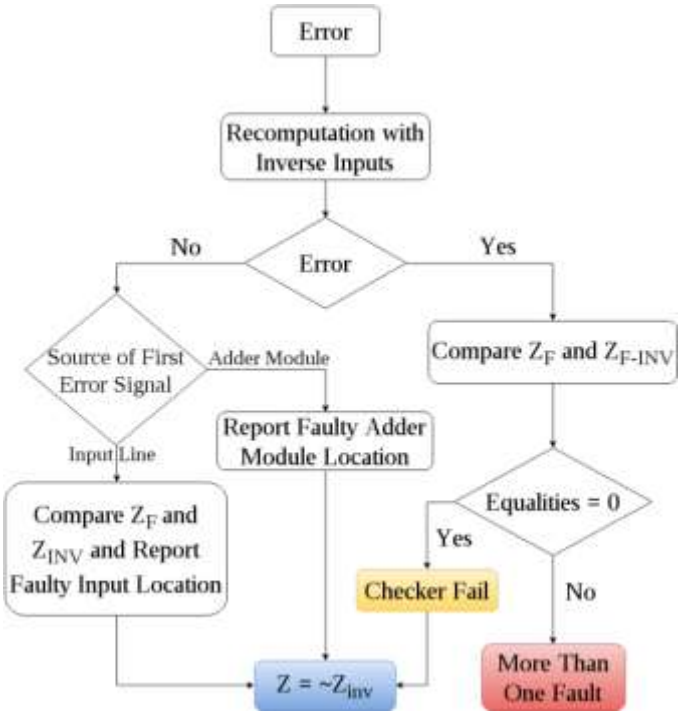

Figure 3.7 Algorithm of the fault correction and localization for the SBSA-PCP implementation

In one FA, any change in one of the input lines changes either the sum or both the sum and carry outputs. Therefore any fault in the a_i^+, a_i^-, or b_i^+ lines makes either the s_i or s_i and h_{i+1} lines faulty. Consequently, any fault in the b_i^-, s_i, or h_i lines makes either the z_i^+ or z_i^+ and z_{i+1}^- output(s) faulty. Thus, any fault in one of the a_i^+, a_i^-, b_i^+, and b_i^- inputs can make a minimum of one (z_i^+) and a maximum of four (z_i^+, z_{i+1}^-, z_{i+1}^+, and z_{i+2}^-) faulty digits. We can conclude that any fault in the *i*th input digit position makes the z_i output faulty. Thus, by comparing both faulty and fault-free results, the lowest equal bit indicates the faulty input digit position.

We suppose that $Z_F = z_F(n)\ldots z_F(i)\ldots z_F(0)$ is the faulty output calculated in the first step and $Z_{INV} = z_{INV}(n)\ldots z_{INV}(i)\ldots z_{INV}(0)$ is the result of the calculation with the inverted inputs in the second step. For all error-free bits, $z_F(i)$ is the inverse of $z_{INV}(i)$, but for the erroneous bits, $z_F(i)$ is equal to $z_{INV}(i)$. Therefore, comparing Z_F and Z_{INV} can locate the error source.

For example, we assume that the result in the first computation is Z_F =*11100100* and an

input fault signal is activated. Therefore, we recalculate the addition using the inverted inputs, and the result is $Z_{INV} = 00010111$. The comparison of these two results in Figure 3.8 shows that Z_1^- and Z_1^+ are equal. Therefore, we can conclude that the fault is located in module number *1*.

	Z_3^+	Z_3^-	Z_2^+	Z_2^-	Z_1^+	Z_1^-	Z_0^+	Z_0^-
Z_F	1	1	1	0	0	1	0	0
Z_{INV}	0	0	0	1	0	1	1	1
	√	√	√	√	X	X	√	√

Figure 3.8 Example of the comparison results from first and second addition operations

The fault localization in our second design (SBSA-IBP) does not require any comparison process because all input lines have a dedicated fault signal. In this implementation, when any fault is activated, similar to the previous implementation, recomputation will done using the inverse inputs. If the fault signal is again activated by comparing the first and second obtained results, we can determine that the error signal is activated due to a checker failure or existence of more than one fault that at least one of them was masked in first calculation. However, if none of the error signals is activated after the second calculation, the faulty module number that activated the error signal in the first computation is reported as the faulty module. Figure 3.9 shows the fault-localization and error-correction algorithm for the SBSA-IBP implementation.

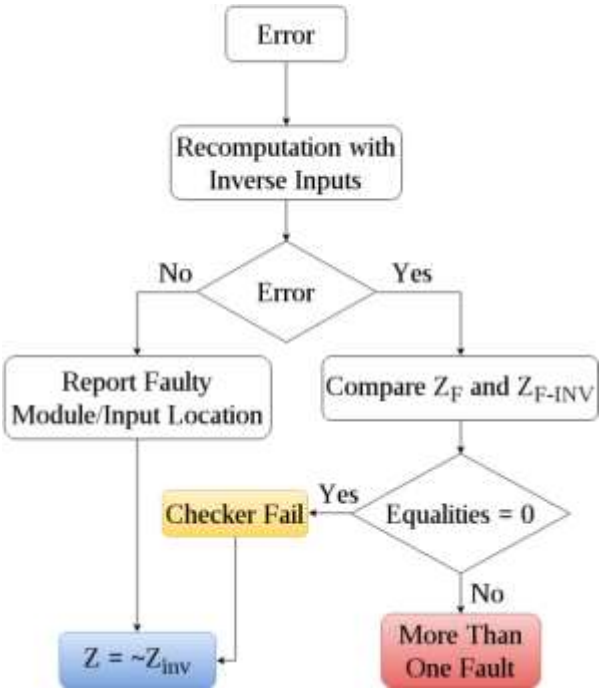

Figure 3.9 Algorithm of the fault correction and localization for SBSA-IBP implementation

4 Performance Evaluation and Comparison

In this chapter, a description of the results obtained from the implementations of the proposed methods and other related methods that could provide us with useful information regarding the efficiency of the proposed methods is presented. To validate the efficiency of the proposed method, we compare our proposed methods with the algorithm in [41]. We discuss the capability of error correction along with the time, area, and power overhead.

1. Error detection

The presented method in [41] and SBSA-PaP method can detect single/odd stuck-at faults both in the input lines or adder modules, and because of the use of a parity checker to detect a fault, these two methods are not able to detect an even number of faults. Because our SBSA-PCP method uses the parity checker to check the existence of faults in the input lines, it can detect a single/odd stuck-at or MCT fault in the input lines. In addition, for the adder modules, it can detect all faults in the separate modules. The SBSA-IBP can detect any number of faults either in the input lines or the adder modules. Table 4.1 lists the error-detection capability of the discussed methods.

Table 4.1 Capability of fault detection

	Fault in input lines		Fault in adder modules	
	Single	**Multiple**	**Single**	**Multiple**
Cardarili's method [41]	Yes	No	Yes	Only odd number of faults
SBSA-PaP	Yes	No	Yes	Only odd number of faults
SBSA-PCP	Yes	No	Yes	Yes
SBSA-IBP	Yes	Yes	Yes	Yes

2. Error correction and fault localization

First we describe the capability of error correction in Cardarilli's method [41]. In an *n*-digit BSDN addition, 5*n*+1 digits are involved [*n*-digit addend (*a*), *n*-digit augend (*b*), *n*-digit partial sum (*w*), *n*-digit intermediate carry (*c*), and *n+1* digit sum (*z*)]. Each digit in BSDN uses two bits for representation; therefore, 10*n*+2 possible locations are available to accommodate a fault.

The proposed process in [41] specifies the correction and localization of all Type 1 faults. Hence, a fault that occurs in *c*, *w*, or *z* is correctable and fully localized. Therefore, any fault in 6*n*+2 bits (*w: 2n*, *c: 2n*, *z: 2n+2*) can be fully corrected and localized. Types2 and 3 faults are correctable only if the recomputation results in fault masking, and none of the error indicator activates an error signal; else, only a partial correction is possible. Because in one bit the fault masking chance is 50%, thorough correction and localization of only half of the Types 2 and 3 faults is possible.

Comprehensively, the probability of full and partial error corrections in the method in [41] can be calculated as

Probability of Type 1 error (error in *w*, *c*, and *z*):

$$\frac{6n+2}{10n+2}$$

Probability of Type 2 error (error in the LSB bit of *a* and *b*):

$$\frac{2n}{10n+2}$$

Probability of Type3 error (error in the MSB bit of *a* and *b*):

$$\frac{2n}{10n+2}$$

Probability of error correction for Type 1 error: 100%

Probability of error correction for Type 2 error: 50%

Probability of error correction for Type 3 error: 50%

Probability of full error correction:

$$100\%\left(\frac{6n+2}{10n+2}\right)+50\%\left(\frac{2n}{10n+2}\right)+50\%\left(\frac{2n}{10n+2}\right)$$

Full error correction: $\frac{8n+2}{10n+2}\cong\frac{4}{5}$

Partial error correction: $\frac{2n}{10n+2}\cong\frac{1}{5}$

A rough approximation shows that the partial and full error correction percentage does not dependent on digit length *n*.

In our proposed procedures, because the value of all 10*n*+2 bits flips in the second recomputation, any stuck-at or MCT fault is masked; therefore, a fault in *a*, *b*, *c*, *w*, or *z* is fully corrected and localized, and the correction algorithm can be applied without referring to the type of the fault (Type 1, 2, or 3 fault). Table 4.2 lists the error-correction probability of the explained methods.

Table 4.2 Probability of error correction

Method	Percentage	
	Input lines	Adder modules
Cardarili's method [41]	50%	100%
SBSA-PaP, SBSA-PCP, SBSA-IBP	100%	100%

With regard to fault localization, presented method in [41] and SBSA-PaP method localize

the fault based on the fault type. A fault in the MSB of input line can cause one to three inequalities, a fault in the LSB can cause up to two inequalities, and a fault in the adder modules can create one inequality. Therefore, when the number of inequality is one, we cannot locate the fault because it can be caused either by the input line (MSB or LSB) or one of the adder modules. In SBSA-PCP, because the error indicators for the adder modules and input lines are separated, the source of the fault can be specified as either due to the adder modules or input line. Because each adder has its own error indicator, any fault in any adder can be localized. To localize the fault in the input lines, comparing the calculated results from the first and second computations can specify the faulty line number. For the SBSA-IBP implementation, all faults can be localized because all adder modules and input lines have separate error indicators.

3. Time, Area, and Power Overhead

To achieve time, area, and power comparison, Verilog HDL is used to code up the method of [41] and our proposed methods. The functional simulation is performed by Modelsim SE while Synopsys Design Compiler is used for synthesis. In addition, for accurate comparison, we designed all methods for 8, 16, 32, 64 and 128 digits.

Time overhead

To analyze the time overhead, first, the delay time in the binary signed-digit adder and self-checking binary signed-digit adder was calculated. Because the signed-digit adder is a parallel adder, the delay time for all lengths of digits is equal. Because SBSA-PCP and SBSA-IBP methods use different signed-digit adder implementation compared with the methods in [41] and SBSA-PaP method, they have different delay times. For both our implementations, the delay time in the binary signed-digit adder is 0.66 ns, and those in the self-checking binary

signed-digit adder are 0.78 ns and 0.85 ns for the SBSA-PCP and SBSA-IBP implementations, respectively. In an error-free calculation, the Cardarili's method [41], and our SBSA-PaP, SBSA-PCP and SBSA-IBP methods take almost the same time, which are approximately 0.71, 0.71, 0.78, and 0.85 ns, respectively, for all digit lengths. Table 4.3 lists the time delays in the absence of a fault.

Table 4.3 Time delay for fault-free addition (ns)

Method	Time				
	8 Digit	16 Digit	32 Digit	64 Digit	128 Digit
Cardarili's method [41]	0.71	0.71	0.71	0.71	0.71
SBSA-PaP	0.71	0.71	0.71	0.71	0.71
SBSA-PCP	0.78	0.78	0.78	0.78	0.78
SBSA-IBP	0.85	0.85	0.85	0.85	0.85

For any fault, all methods need to perform the recomputation step(s) to localize and correct the faulty digit. Table 4.4 lists the estimated time for the localization and correction algorithm in all methods. As mentioned earlier, in our methods, correction can be straightforwardly done by inverting the result obtained in the second computation; however, the Cardarili's method [41] needs a different calculation to obtain a correct result. Therefore, the time delays in our proposed methods are quite less compared with that in the Cardarili's method [41].

Table 4.4 Time delay for the fault-localization and error-correction algorithm only (ns)

Method	Time				
	8 Digit	16 Digit	32 Digit	64 Digit	128 Digit
Cardarili's method [41]	2.52	3.42	5.17	9.78	18.49
SBSA-PaP	1.7	1.9	2.05	2.14	2.34
SBSA-PCP	0.89	1.02	1.16	1.34	1.49
SBSA-IBP	0.35	0.43	0.51	0.60	0.78

For any fault, the Cardarili's method [41] needs to perform a minimum of two addition processes (computation using normal inputs and LSI) with one localization and correction process and a maximum of three addition processes (computation using normal inputs, LSI, and RSI) with two localization and correction processes. In our proposed methods, for any fault, only two addition processes along with one localization and correction process are needed. According to Table 4.3 and Table 4.4, we can estimate the delay times in the mentioned methods for faulty addition, as listed in Table 4.5.

Table 4.5 Approximate time delay for faulty addition (ns)

Method	Time				
	8 Digit	16 Digit	32 Digit	64 Digit	128 Digit
Cardarili's method [41] LSI	3.94	4.84	6.59	11.2	19.91
Cardarili's method [41] RSI	7.17	8.97	12.74	21.69	39.11
SBSA-PaP	3.12	3.32	3.47	3.56	3.76
SBSA-PCP	2.45	2.58	2.72	2.9	3.05
SBSA-IBP	1.91	1.99	2.07	2.16	2.34

By referring to Table 4.4 and Table 4.5, we can prove the superiority of our proposed methods compared with that of the Cardarili's method [41]. The tables also indicate that in our methods, the delay time slightly depends on the input digit length compared with the

Cardarili's method [41]. Figure 4.1 shows that the delay time increases by increasing the number of input digits.

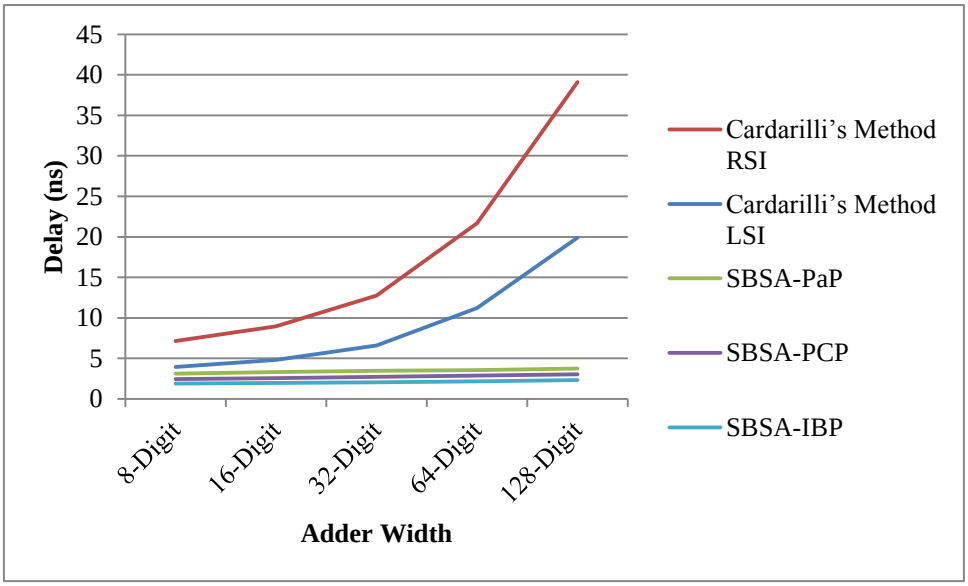

Figure 4.1 Approximate delay time in the presence of a fault.

Area overhead

Table 4.6 lists the occupied area in the unprotected conventional binary signed-digit adder and self-checking binary signed-digit adder under two different implementations. The minimum area overhead for the self-checking BSDN adder is approximately 67% in the implementation using information from the previous digit position and 76% using the double-recoding method of the corresponding conventional BSDN adder. As listed in Table 4.7, adding the fault-localization and error-correction algorithm increases the occupied area in all methods. However, in the SBSA-PCP method, the increase in the area is not as large as that of other described technique. Further, it significantly requires lesser area than the other approaches. SBSA-IBP method needs a slightly larger area to detect and localize the fault, but still it needs a lesser area than SBSA-PaP and method in [41].

Table 4.6 Occupied area in the conventional and self-checking BSDA (μm2)

Implementation Method		Area				
		8 Digit	16 Digit	32 Digit	64 Digit	128 Digit
Using information from the previous digit position	Unprotected BSDA	520	1030	2050	4091	8176
	Self-checking BSDA (Figure 2.9)	870	1721	3472	6870	13737
Using double-recoding method	Unprotected BSDA (Figure 3.3)	460	899	1778	3535	7053
	SBSA-PP (Fig. 8)	815	1579	3198	6351	12666
	SBSA-IP (Fig. 9)	1251	2321	4697	9321	18910

Table 4.7 Occupied whole area (μm^2)

Method	Area				
	8 Digit	16 Digit	32 Digit	64 Digit	128 Digit
Cardarili's method [41]	2476	3873	7420	14109	29033
SBSA-PaP	1511	2884	5497	11456	23181
SBSA-PCP	1162	2159	4358	8651	17560
SBSA-IBP	1251	2321	4697	9321	18910

Table 4.8 shows the percentage area overhead in different adder design based on original adder. According to the table Cardarili's method [41] overhead is more than 300 percent compared with unprotected signed-digit adder. Our designed adder, SBSA-PCP, has less area overhead among all other designs. In order to compare out designs with other fault-tolerance designs we show the area overhead in two other adders. Table 4.8 shows that the fault-tolerance

carry-lookahead adder presented in [43] has average 130 percent overhead compared with original unprotected carry-lookahead adder, while our SBSA-PCP design had average less than 120 percent area overhead. Protected carry select adder presented in [44] shows less area overhead compared with other designs but this is calculated without considering occupied areas by LUTs and Registers. In addition, the probability of error detection/correction in carry select adder in [44] is less than other designs.

Table 4.8 Time complexity and percentage of area overhead in different adders based on original unprotected adder

Method	Area overhead (compared with unprotected adder)					Time Complexity
	8 Digit	16 Digit	32 Digit	64 Digit	128 Digit	
Cardarili's method [41]	376.1%	276.0%	261.9%	244.9%	255.1%	O(1)
SBSA-PaP	190.6%	180.0%	168.1%	180.0%	183.5%	O(1)
SBSA-PCP	123.5%	109.6%	112.6%	111.5%	114.8%	O(1)
SBSA-IBP	140.6%	125.3%	129.1%	127.8%	131.3%	O(1)
Carry-lookahead adder [43]	137.5%	136.9%	136.6%	136.5%	N/A	O(log(n))
Carry select adder [44] without LUTs/Reg	84.9%	75.15%	68.19%	65.45%	N/A	O($\sqrt{n}$)

Figure 4.2 shows the increase in the area with the increase in the number of input digits in the four methods.

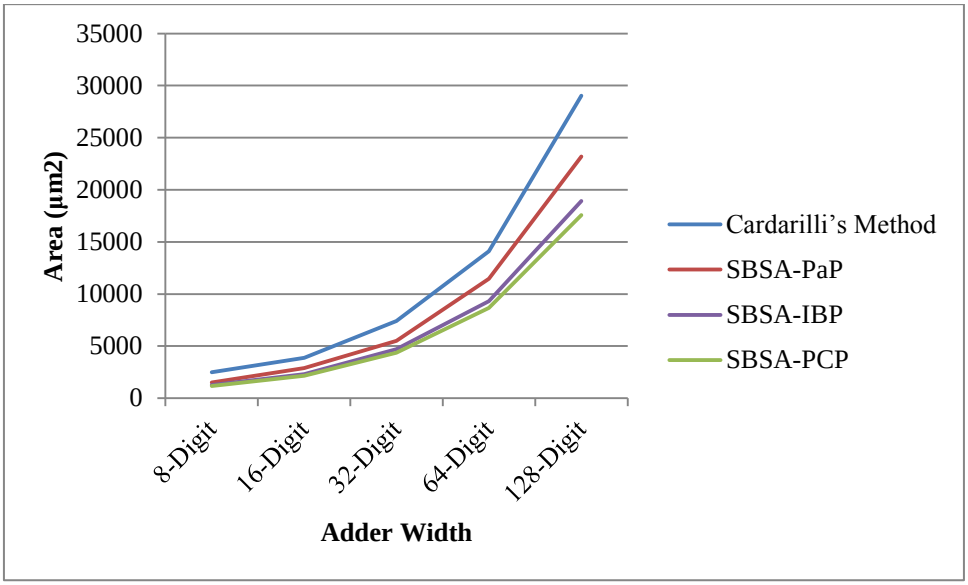

Figure 4.2 Increase in the area by increasing the number of input digits (µm2).

Power overhead

Finally, the power consumption comparison results of the method in [41] with that of our proposed methods are listed in Table 4.9, which shows that the power consumption of our proposed methods is significantly lesser than the method in [41].

Table 4.9 Power consumption (µW)

Method	Power				
	8 Digit	16 Digit	32 Digit	64 Digit	128 Digit
Cardarili's method [41]	171	288	614	1200	2480
SBSA-PaP	146	264	592	1190	2470
SBSA-PCP	92	174	367	758	1660
SBSA-IBP	105	137	240	573	1541

Figure 4.3 shows the results obtained from the five compared methods. The methods in [41] and SBSA-PaP method almost overlapped and incur more power consumption than others.

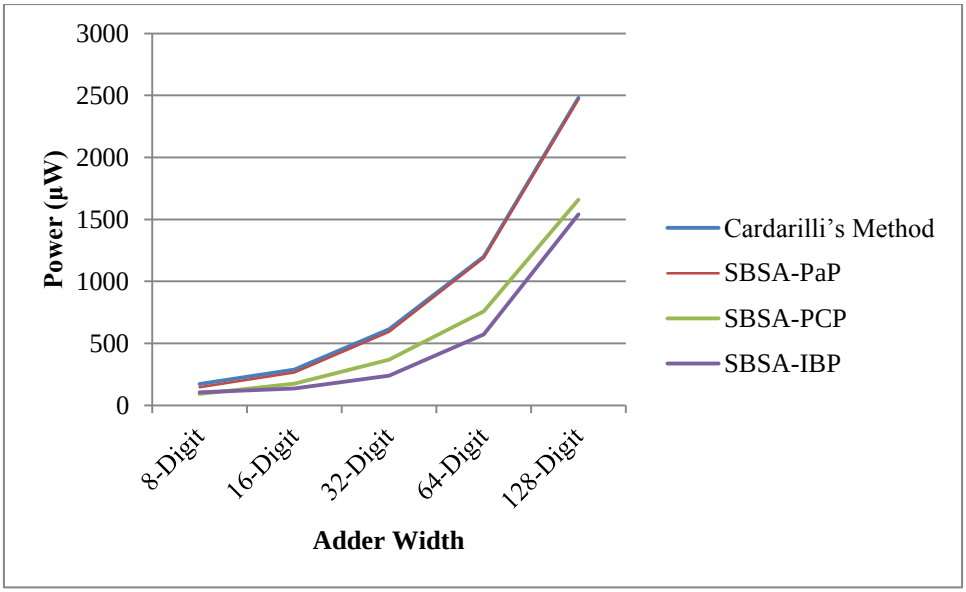

Figure 4.3 Increased in power consumption with the increase in the number of input digits (µW).

The results prove the superiority of the proposed methods over the previously suggested methods. Our methods lead in all aspects such as lower application time, area overhead, and power consumption, along with high error-correction and fault-localization capability with more fault coverage.

5 Conclusion

In this chapter, the conclusion on the overall study will be discussed. At first findings of the thesis are briefly reviewed, and then the contributions of the methods are discussed. And finally a future work is suggested to extend the proposed methods in a way to act more efficient.

This research has presented new fault-localization, detection, and error-correction methods for a self-checking BSDN adder scheme. Through the use of inverted operands, the new methods with low-complexity, state that these capabilities are achievable. Fault-localization and error-correction schemes rely on exploitation of the self-dual function model. Furthermore, with respect to full error-correction capabilities using recomputation with complemented operands, comparison with the help of obtained results led to the localization of faulty bit(s). Accomplishment of fault localization and error correction using the proposed approaches are performed in one phase only.

The research indicated that for error correction using self-dual property beside with parity prediction structure is more efficient than error correction method using recomputation with shifted operands. In addition a novel, low-complexity approach shows that error detection and localization capabilities are more efficient and achievable using the FA property, which takes advantage of the observed relationship between the inputs and outputs in an FA, beside with the self-dual function concept to set up fault-localization and error-correction procedures.

Future work will aim towards the attainment of fault-localization and error correction properties at low cost in different radix signed-digit number adders by employing the proposed techniques. For instance the future work will include the use of these methods in radix-4 quaternary signed-digit (QSD) number adders to obtain fault-tolerance property at low cost. In Radix-4 the operations on greater numbers like 64, 128 –bit, that is carry free, can be implemented with a persistent delay and low complicacy. QSD system uses four states instead

of two states in binary form. In QSD, every digit is presented by a number in between -3 to 3.

QSD acts as a support for the new state-of-the-arts areas like quantum computing, DNA computing, optical computing, and quantum dot cellular automata ant etc.

6 References

[1] Mukherjee, Shubu. "Architecture design for soft errors." Morgan Kaufmann, 2011.

[2] Mehdizadeh, Nima, Mohammad Shokrolah-Shirazi, and Seyed Ghassem Miremadi. "Analyzing fault effects in the 32-bit OpenRISC 1200 microprocessor." IEEE Third International Conference on Availability, Reliability and Security (ARES), 2008.

[3] Meixner, Albert, Michael E. Bauer, and Daniel Sorin. "Argus: Low-cost, comprehensive error detection in simple cores." 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2007.

[4] Pellegrini, Andrea, et al. "CrashTest'ing SWAT: accurate, gate-level evaluation of symptom-based resiliency solutions." Proceedings of the Conference on Design, Automation and Test in Europe Consortium (EDA), 2012.

[5] Hong, Seokin, and Soontae Kim. "Lizard: Energy-efficient hard fault detection, diagnosis and isolation in the ALU." IEEE International Conference on Computer Design (ICCD), 2010.

[6] Kim, Nam Sung, et al. "Leakage current: Moore's law meets static power." Computer 36.12, pp. 68-75, 2003.

[7] Thompson, Scott E., et al. "In search of ‘forever,’ continued transistor scaling one new material at a time." IEEE Transactions on Semiconductor Manufacturing 18.1, pp. 26-36, 2005

[8] Velazco, Raoul, Pascal Fouillat, and Ricardo Reis, eds. "Radiation effects on embedded systems". Springer Science & Business Media, 2007.

[9] Heijmen, Tino. "Radiation-induced soft errors in digital circuits--A literature survey.", 2002.

[10] Lisboa, Carlos Arthur Lang, and Luigi Carro. "XOR-based low cost checkers for combinational logic." IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems, 2008.

[11] Rossi, Daniele, et al. "Multiple transient faults in logic: An issue for next generation ICs?." 20th IEEE International Symposium on Defect and Fault

Tolerance in VLSI Systems (DFT'), 2005.

[12] Rusu, Claudia, et al. "Multiple event transient induced by nuclear reactions in CMOS logic cells." 13th IEEE International On-Line Testing Symposium (IOLTS), 2007.

[13] Dodd, Paul E., et al. "Production and propagation of single-event transients in high-speed digital logic ICs." IEEE Transactions on Nuclear Science 51.6, pp. 3278-3284, 2004.

[14] Ferlet-Cavrois, V., et al. "Statistical analysis of the charge collected in SOI and bulk devices under heavy lon and proton irradiation—Implications for digital SETs." IEEE Transactions on Nuclear Science 53.6, pp. 3242-3252, 2006.

[15] Huang, Ryan H-M., and Charles H-P. Wen. "Advanced soft-error-rate (SER) estimation with striking-time and multi-cycle effects." 51st ACM/EDAC/IEEE Design Automation Conference (DAC), 2014.

[16] Bastos, Rodrigo Possamai, et al. "A New Recovery Scheme Against Short-to-Long Duration Transient Faults in Combinational Logic." Journal of Electronic Testing 29.3, pp. 331-340, 2013.

[17] Lisboa, C. A., Marcelo Ienczczak Erigson, and Luigi Carro. "System level approaches for mitigation of long duration transient faults in future technologies." 12th IEEE European Test Symposium (ETS), 2007.

[18] Lisboa, C., M. Erigson, and L. Carro. "A low cost checker for matrix multiplicatio." IEEE Latin-American Test Workshop, LATW. Vol. 8, 2007.

[19] Arizona State University, "Predictive technology model web site," [Online]. Available: http://www.eas.asu.edu/~ptm.

[20] Behrooz, Parhami. "Computer arithmetic: Algorithms and hardware designs." Oxford University Press 19, 2000.

[21] Rabaey, J. M., and M. Pedram. "Low Power Design Methodologies Kluwer Academic Publishers." Norwell, MA, 1996.

[22] González, Alejandro F., and Pinaki Mazumder. "Redundant arithmetic, algorithms and implementations." Integration, the VLSI journal 30.1, pp. 13-53, 2000.

[23] Schneider, Klaus, and Adrian Willenbücher. "A New Algorithm for Carry-Free Addition of Binary Signed-Digit Numbers." 22nd IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2014.

[24] Bickham, Ryan. "An Analysis of Error Detection Techniques for Arithmetic Logic Units". Dissertation, Vanderbilt University, 2010.

[25] Johnson, Barry W., James H. Aylor, and Haytham H. Hana. "Efficient use of time and hardware redundancy for concurrent error detection in a 32-bit VLSI adder." IEEE Journal of Solid-State Circuits 23.1, pp. 208-215, 1988.

[26] Golander, Amit, Shlomo Weiss, and Ronny Ronen. "Synchronizing redundant cores in a dynamic DMR multicore architecture." IEEE Transactions on Circuits and Systems II: Express Briefs 56.6, pp. 474-478, 2009.

[27] Nicolaidis, Michael. "Time redundancy based soft-error tolerance to rescue nanometer technologies." 17th IEEE Proceedings VLSI Test Symposium, 1999.

[28] Patel, Janak H., and Leona Y. Fung. "Concurrent error detection in ALU's by recomputing with shifted operands." IEEE Transactions on Computers 100.7, pp. 589-595, 1982.

[29] Reynolds, Dennis A., and Gernot Metze. "Fault detection capabilities of alternating logic." IEEE Transactions on Computers 100.12, pp. 1093-1098, 1978.

[30] Ngai, Tat, Chen He, and EE Jr Swartzlander. "Enhanced concurrent error correcting arithmetic unit design using alternating logic." IEEE Proceedings International Symposium on Defect and Fault Tolerance in VLSI Systems, 2001.

[31] Nicolaidis M., Manich S. "Fault-secure parity prediction arithmetic operators." IEEE Design and Test of Computers, 1997.

[32] Saitoh, Yuichi, and Hideki Imai. "Multiple undirectional byte error-correcting codes." IEEE Transactions on Information Theory 37.3, pp. 903-908, 1991.

[33] Koob, Gary M., and Clifford G. Lau, eds. "Foundations of Dependable Computing: System Implementation". Springer Science & Business Media, Vol. 285, 1994.

[34] Nicolaidis, Michael. "Efficient implementations of self-checking adders and ALUs." The Twenty-Third International Symposium on Fault-Tolerant Computing,

(FTCS-23), Digest of Papers, 1993.

[35] Sellers, Frederick F., Muyue Xiao, and Leroy W. Bearnson. "Error detecting logic for digital computers." McGraw-Hill, 1968.

[36] Riemens, Danny P., et al. "Towards scalable arithmetic units with graceful degradation." ACM Transactions on Embedded Computing Systems (TECS) 13.4, 2014.

[37] Mitra, Subhasish, and Edward J. McCluskey. "Which concurrent error detection scheme to choose?." IEEE Proceedings International Test Conference, 2000.

[38] Townsend, Whitney J., Jacob A. Abraham, and E. E. Swartzlander. "Quadruple time redundancy adders [error correcting adder]." 18th IEEE Proceedings International Symposium on Defect and Fault Tolerance in VLSI Systems, 2003.

[39] Khedhiri, Chiraz, et al. "A fault tolerant adder based on alternative computation." International Journal of Design, Analysis and Tools for Integrated Circuits and Systems Vol. 3, No. 1, pp. 14-18, 2012.

[40] Peng, Song, and Rajit Manohar. "Fault tolerant asynchronous adder through dynamic self-reconfiguration." IEEE International Conference on Computer Design, 2005.

[41] Cardarilli, Gian-Carlo, et al. "Fault localization, error correction, and graceful degradation in radix 2 signed digit-based adders." IEEE Transactions on Computers 55.5, pp. 534-540, 2006.

[42] Namazi, Alireza, et al. "A low-cost fault-tolerant technique for Carry Look-Ahead adder." 15th IEEE International On-Line Testing Symposium, 2009.

[43] Valinataj, Mojtaba. "A novel self-checking carry lookahead adder with multiple error detection/correction." Microprocessors and Microsystems 38.8, pp. 1072-1081, 2014.

[44] Mukherjee, Atin, and Anindya Sundar Dhar. "Real-time fault-tolerance with hot-standby topology for conditional sum adder." Microelectronics Reliability 55.3, pp. 704-712, 2015.

[45] Akbar, Muhammad Ali, and Jeong-A. Lee. "Self-repairing adder using fault

localization." Microelectronics Reliability 54.6, pp. 1443-1451, 2014.

[46] Avizienis, Algirdas. "Signed-digit numbe representations for fast parallel arithmetic." IRE Transactions on Electronic Computers 3, pp. 389-400, 1961.

[47] Ercegovac, Miloš D., and Tomas Lang. "Digital arithmetic." Elsevier, 2004.

[48] Cardarilli, Gian-Carlo, et al. "Error detection in signed digit arithmetic circuit with parity checker [adder example]." 18th IEEE Proceedings International Symposium on Defect and Fault Tolerance in VLSI Systems, 2003.

[49] Parhami, Behrooz. "Carry-free addition of recoded binary signed-digit numbers." IEEE Transactions on Computers 37.11, pp. 1470-1476, 1988.

[50] Alavi, Seyyed Roohollah, and Karim Faez. "Fault Localization and Full Error Correction in Radix2 Signed Digit-Based Adders." IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, 2007.

[51] Patel, Janak H., and Leona Y. Fung. "Concurrent error detection in ALU's by recomputing with shifted operands." IEEE Transactions on Computers 100.7, pp. 589-595, 1982.

[52] John Barry, W. "Design and Analysis of Fault-Tolerant Digital Systems.", 1989.

[53] Oikonomakos, Petros, and Paul Fox. "Error correction in arithmetic operations by I/O inversion." 12th IEEE International On-Line Testing Symposium (IOLTS'06), 2006.

[54] Takagi, Naofumi, Hiroto Yasuura, and Shuzo Yajima. "High-speed VLSI multiplication algorithm with a redundant binary addition tree." IEEE Transactions on Computers 100.9, pp. 789-796, 1985.

[55] Kuninobu, Shigeo, et al. "Design of high speed MOS multiplier and divider using redundant binary representation." 8th IEEE Symposium on Computer Arithmetic (ARITH), 1987.

[56] Takagi, Naofumi. "Studies on hardware algorithms for arithmetic operations with a redundant binary representation." Kyoto University, 1988.

[57] Kuninobu, Shigeo, Tamotsu Nishiyama, and Takashi Taniguchi. "High speed MOS multiplier and divider using redundant binary representation and their

implementation in a microprocessor." IEICE Transactions on Electronics 76.3, pp. 436-445, 1993.

[58] Makino, Hiroshi, et al. "An 8.8-ns 54× 54-bit multiplier with high speed redundant binary architecture." IEEE Journal of Solid-State Circuits 31.6, pp. 773-783, 1996.

[59] Moradian, Hossein, and Jeong-A. Lee. "Low-Cost Fault Localization and Error Correction for a Signed Digit Adder Design Utilizing the Self-Dual Concept." IEEE Euromicro Conference on Digital System Design (DSD), 2015.