Reliable and Fault-Resilient Schemes for Efficient Radix-4 Complex Division

Niranjan Manoharan

Follow this and additional works at: https://scholarworks.rit.edu/theses

Recommended Citation

This Master’s Project is brought to you for free and open access by RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
RELIABLE AND FAULT-RESILIENT SCHEMES FOR EFFICIENT RADIX-4 COMPLEX DIVISION

by

Niranjan Manoharan

A Graduate Paper Submitted

in

Partial Fulfillment

of the

Requirements for the Degree of

MASTER OF SCIENCE

in

Electrical Engineering

Approved by:

Prof. (Dr. Mehran Mozaffari-Kermani)

Prof. (Dr. Sohail A. Dianat)

DEPARTMENT OF ELECTRICAL AND MICROELECTRONIC ENGINEERING

COLLEGE OF ENGINEERING

ROCHESTER INSTITUTE OF TECHNOLOGY

ROCHESTER, NEW YORK

MAY, 2014
Acknowledgements

I am grateful to my advisor, Dr. Mehran Mozaffari-Kermani, whose support and guidance helped me to understand the concepts clearly and complete my Masters. Special thanks also go to other professors, who helped me in my field of study, Dr. Dorin Patru, Dr. Marcin Lukowiak and Dr. Reza Azarderakhsh, for their constructive comments and participation.

This graduate paper is dedicated to my parents.
Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text.

Niranjan Manoharan

2014
Abstract

Complex division is commonly used in various applications in signal processing and control theory including astronomy and nonlinear RF measurements. Nevertheless, unless reliability and assurance are embedded into the architectures of such structures, the sub-optimal (and thus erroneous) results could undermine the objectives of such applications. As such, in this thesis, we present schemes to provide complex number division architectures based on (Sweeney, Robertson, and Tocher) SRT-division with fault diagnosis mechanisms. Different fault resilient architectures are proposed in this thesis which can be tailored based on the eventual objectives of the designs in terms of area and time requirements, among which we pinpoint carefully the schemes based on recomputing with shifted operands (RESO) to be able to detect both natural and malicious faults and with proper modification achieve high throughputs. The design also implements a minimized look up table approach which favors in error detection based designs and provides high fault coverage with relatively-low overhead. Additionally, to benchmark the effectiveness of the proposed schemes, extensive fault diagnosis assessments are performed for the proposed designs through fault simulations and FPGA implementations; the design is implemented on Xilinx Spartan-VI and Xilinx Virtex-VI FPGA families.
Contents

List of Figures iv

List of Tables v

1 Introduction 1
  1.1 Complex Division . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
  1.2 Fault Detection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
    1.2.1 Faults and Degradation . . . . . . . . . . . . . . . . . . . . . . . . 2
    1.2.2 Fault Detection Techniques . . . . . . . . . . . . . . . . . . . . . . 4
      1.2.2.1 Concurrent Error Detection . . . . . . . . . . . . . . . . 4
      1.2.2.2 Off-Line Fault Detection . . . . . . . . . . . . . . . . . 5
      1.2.2.3 Roving Fault Detection . . . . . . . . . . . . . . . . . 6
  1.3 Reconfigurable Hardware . . . . . . . . . . . . . . . . . . . . . . . . . . 7
  1.4 Objectives . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
  1.5 Thesis Outline . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

2 Preliminaries 11
  2.1 Hardware Based Dividers . . . . . . . . . . . . . . . . . . . . . . . . . . 11
  2.2 Radix-β SRT Division . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
  2.3 Golub’s Multiplication . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

3 Presented Complex Division Radix-4 SRT Module 18
### Contents

3.1 ROM ............................................................................................................. 19

4 Proposed Error Detection Schemes ............................................................ 22

4.1 Unified Parity Check and Hardware Redundancy .................................... 23

4.1.1 Golub's Multiplier .................................................................................. 25

4.1.2 Datapath Registers ................................................................................ 25

4.1.3 ROM (Quotient Selection Logic) ........................................................... 27

4.1.4 Carry Save Adder ................................................................................... 27

4.2 Error Detection through RESO ................................................................. 28

5 Error Simulations ......................................................................................... 31

6 FPGA Implementations and Benchmark .................................................... 33

7 Conclusion .................................................................................................... 36

References ....................................................................................................... 37
## List of Figures

2.1 Dot representation of basic hardware division. ........................................ 13
2.2 New shifted remainder and old shifted remainder in radix-$\beta$ (Robertson
diagram). ................................................................. 14
2.3 p-d plot quotient selection for radix-$\beta$ division. ................................. 15
2.4 The quotient selection scale. ....................................................... 16
2.5 Golub’s multiplication. .............................................................. 17

3.1 Presented complex radix-4 SRT module. ........................................... 19
3.2 Quotient selection logic. .............................................................. 21
3.3 Minimized ROM. ................................................................. 21

4.1 Radix-4 complex SRT module with unified parity check and hardware re-
dundancy. ................................................................. 24
4.2 CED Multiplier. ............................................................... 26
4.3 Parity check registers. .............................................................. 26
4.4 Minimized ROM with the proposed parity prediction logic. ................. 27
4.5 Full added with parity check. .................................................... 28
4.6 Proposed divider with RESO for fault diagnosis. ............................. 30
List of Tables

6.1 FPGA Implementation results of the original complex division module and the proposed error detection designs on Spartan-6 FPGA device xc6s1x16. 34

6.2 FPGA Implementation results of the original complex division module and the proposed error detection designs on Virtex-6 FPGA device xc6vlx75t. 34
Chapter 1

Introduction

1.1 Complex Division

Complex division is a critical mathematical operation with applications in various fields such as signal processing, control theory, microwave systems, quantum mechanics, and the like. Control theory uses complex division arithmetic to find the root locus [1], Nyquist plot [2], and Nichols plot [3]. Microwave systems also use complex division arithmetic to find the frequency response [4] and transfer functions. Because of its complexity, the operation has been mainly implemented in software [5], [6]. This has been further improved by using different algorithms [7], [8] to prevent overflows and provide precise results. Other optimizations [9] have been proposed to make use of the fused multiply-add (FMA) instructions available on different processors to improve the component-wise accuracy.

A technique for high radix complex division has been proposed in [10], this approach is based on operand prescaling and digit recurrence. Such an algorithm has later been implemented on FPGAs with different radices [11], [12]. Furthermore, a radix-16 combined complex divider/square root module has also been presented in [13] based on the same algorithm. Moreover, a complex divider has been presented in [14] which uses the standard formula for complex division but uses an optimized architecture to reduce area and improve the operating frequency. Another complex divider is implemented using the coordinate rotational digital computer (CORDIC)-like algorithms [15]. There exist
other complex division techniques which are based on complex binary number system (CBNS) \[16\], where complex numbers are represented in binary. Additionally, a division algorithm based on \[16\] has been implemented in \[17\]. This design has drawback in terms of accuracy as it is based on a new system and extensive research is still needed to reach more accurate results. There has also been a complex division scheme which uses dichotomous coordinate descent (DCD) algorithm to calculate the results by converting the division operation to a system of linear equations \[18\]. Implementing this technique is very complicated and area inefficient. It is noted that none of the aforementioned systems provides reliability and hardware assurance for the underlying architectures. Indeed, in the presence of defects in very-large-scale integration (VLSI) architectures of such important computer arithmetic calculations, erroneous outputs resulting from sub-optimal reliability assurance could undermine the respective eventual objectives.

### 1.2 Fault Detection

Fault is a problem that results in a complete failure of a piece of equipment, or even involves specific hardware. A problem in digital system can be defined as a bit inversion in digital hardware, i.e., 0 to 1 or 1 to 0. As technology becomes scaled, manufacturing large defect-free integrated circuits becomes difficult. There is also the issue of device degradation over large periods of time. ASICs and FPGAs are primarily affected by these degradation issues which make them less reliable over time. Future FPGAs, beyond the 45nm technology will have low reliability such that fault tolerance or other recovery methods will be unavoidable in large FPGAs. This section provides some insight on some common faults and degradation. FPGAs are highly reconfigurable; this provides interesting opportunities for fault detection and tolerance.

#### 1.2.1 Faults and Degradation

Digital circuits incur degradation in many ways \[19\], some of the leading ways are as follows:
• The *hot-carrier* effect leads to a buildup of trapped charges in the gate-channel interface region [20]. This leads to degradation in electron mobility and increased threshold voltage in CMOS transistors. This in turn leads to reduced switching speeds and hence leads to increased delays. This phenomenon is also caused as a result of negative-bias temperature instability, which exhibits a similar behavior [21].

• *Electromigration* is a phenomenon in which metal ions migrate, which leads to a lack of holes and voids in interconnect. Eventually these can cause faults by creation of shorted circuits or open circuits [22].

• *Time-Dependent Dielectric Breakdown* causes an increase in leakage current by affecting the transistor gates, eventually this leads to a short circuit. The reason for this is charge trap creation within the gate dielectrics which diminishes the potential barrier if forms [23, 24]

In addition to degradation, there are a couple of other faults that can affect FPGAs. These are highly relevant to this thesis because the proposed techniques uses these fault models.

• Manufacturing defects can affect circuit nodes which cause a stuck-at-0 or 1 or they may switch slowly and cause a timing issue. Interconnect networks can also be affected, leading to short or open circuits and open or closed transistors [25].

• The most common type of fault in FPGAs comprises of Single Event Upsets (SEUs) and Single Event Transient (SETs) caused by certain types of radiation. These types of faults primarily affect circuits in aviation, space applications and nuclear research where devices are subjected to higher levels of radiation. Another most common and frequently occurring type of fault is in SRAM cells, where particular cells maybe flipped. This error remains in the memory until a refresh occurs; this process is known as scrubbing. Scrubbing may not be relevant during permanent faults but is highly useful during transient faults.
1.2 Fault Detection

1.2.2 Fault Detection Techniques

Fault detection primarily has two purposes; alerting the supervising process that action needs to be taken for the system to remain operational and secondly, the defective components are identified so that it can be repaired. Usually, these two stages are covered simultaneously or it can have more than one stage comprising of different strategies. Fault detection strategies can be categorized into three types:

- **Redundant/concurrent error detection**: This technique uses additional circuitry to detect a potential fault/error. The most frequently used techniques are parity detection and hardware redundancy.

- **Off-line test methods**: This methodology uses external circuitry to detect faults in an FPGA/ASIC when it is not in operation. Some examples of off-line test circuits are Built-In-Self-Test (BIST) and Automated-Test-Pattern-Generator (ATPG).

- **Roving test methods**: These techniques take a complicated approach but are useful in pinpointing a faulty location in a FPGA circuit. Roving performs a scan of the entire FPGA structure and checks for defects by replacing them with a test function.

1.2.2.1 Concurrent Error Detection

Error detection of this kind is mainly used to detect errors due to SEUs and SETs. These detection methods as mentioned before involve adding logic to the original module to implement the detection mechanism. In the event of an error, there is a disagreement between the included logic and the original module, over which a particular calculation is processed and this is indicated by some sort of error detection mechanism, usually a comparator. One of the simplest and most used methods is modular redundancy. A functional block is replicated, usually two or three times. The outputs of these replicated modules are compared; any differences in the results trigger the error mechanism. Concurrent Error Detection (CED) allows a more area efficient design compared to modular redundancy. The data lines and registers are widened to store error coding algorithms.
1.2 Fault Detection

such as parity. Error validation algorithms are included at the outputs of these lines or
registers to detect faults (parity check registers).

The major drawback of modular redundancy is the additional overhead required for
its implementation, which can be three times in case of TMR [26] or more in NMR. In
addition to this, it provides very limited resolution for the identification of the faulty
component. The fault detection can only be limited to a particular block which imple-
ments modular redundancy. This can be overcome by breaking down the circuit and
adding additional error detection logic, with the expense of area. In CED, the efficiency
of error coverage comes at the expense of additional area. These methods do not provide
coverage of multiple SEUs or SETs.

Error coverage through redundancy does not have to be restricted to circuit area. It
is possible to detect errors with the trade-off with latency/throughput. [27], [28] proposes
a method where operations are carried out twice. During the second run the operations
are encoded in a different way so that they yield a different output. The output of both
runs are passed through a suitable decoder and then compared at the output. In certain
cases, data-checking (parity) and redundant systems are incorporated into an FPGA
configuration. [29] implements a FPGA system with built in redundancy.

1.2.2.2 Off-Line Fault Detection

One of the most widely used techniques for fault detection is off-line fault detection.
This is usually used to identify manufacturing defects in FPGAs. Any circuit which does
this without the help of an external circuit, i.e., built into the FPGA but not a part of
the original design. These techniques work by loading one or more test patterns into
the FPGA. The entire system consists of a test pattern generator, an output response
analyzer and between them the device under test. To include full coverage the system
will have to test not only the logic and interconnect, but also the configuration network.
Many recent consumer grade FPGAs have this built into the development boards. This
eliminates the need of a large number of different test configurations. Compared to ASICs,
FPGAs have the advantage of a fixed reconfigurable structure, which removes the need of
dedicated test structures to be built into the circuit. As FPGAs are reconfigurable, test patterns should be optimized so that they can be used for various designs. The major advantage of BIST is that it does not interfere with normal operation of the FPGA. It also covers complicated systems such as PLLs and clock networks. The major drawback of BIST is that it can detect faults only when the circuit is not operational, i.e., only when a dedicated test mode is run, it can detect faults. This is usually done at system startup or when an error event triggers the BIST check.

Recent trends use BIST to test individual LUT [30–32] properties like timing and stuck-at-faults. [33] uses a BIST system to test interconnects which reduces time through a large degree. In [25] and [34] a hierarchical approach is used to locate stuck-at faults, short circuits and open circuits with high accuracy.

1.2.2.3 Roving Fault Detection

This technique exploits run-time configuration to adapt BIST techniques on-line, with minimum area overhead. In roving detection, the FPGA is split equally into a certain number of regions. One of the regions is subjected to BIST testing while the rest carry out the desired operation. Over time the region under test is switched with the operational region, in this order the entire FPGA array is scanned for faults. This method is very popular in partial reconfiguration. Roving techniques have lower area compared to modular techniques; the overhead comprises of just one self-test region and a controller to manage the reconfigurable (swapping) process and better than off-line methods because the circuit can be operational. The speed, while better than off-line testing methods, it is not comparable to redundancy techniques. The detection speed of a roving technique depends on the speed of the roving cycle and the operation time. The best roving tests are reported to have latency of less than one second [35].

In roving tests, performance is impacted in two ways. Firstly, the connections between adjacent functional areas are stretched as the test region is moved though the FPGA, resulting in longer signal delays which in turn affect the system clock speed, in the range of 2.5% to 15% [36]. Secondly, in current FPGAs the functional blocks are halted as
they are swapped. It is reported that a $250 \, \mu s$ pause is required between swaps. Most of the initial testing in the field of roving test has been carried out by Emmert, Stroud and Abramovici [35, 36]. The system called Roving STARs uses two test sections one for rows and other for columns. Another variation of roving test was proposed in [37] which uses buses rather than segmented interconnects. Though, this system has no impact on system clock, it limits the application base.

1.3 Reconfigurable Hardware

Hardware modules containing a large number of switching-circuit components are called reconfigurable hardware modules. The modules in general are called field programmable gate arrays (FPGAs), general FPGAs contain an array of simple logic cells, some storage, and interconnection path in each cell that can be configured to perform specific functions by loading a predetermined bit pattern. The predetermined bit patterns are loaded into a layer called a configurable logic block (CLB) which rests below the logic layer. This layer is responsible for the connection pattern between different components in the FPGA. Once the bit pattern is loaded into the CLB, the actual customization is very fast compared to the standard process of producing custom ICs. This allows the user to use the same hardware for different implementations. Even though, the designs presented in this material are suitable for ASIC, FPGAs have gained popularity in recent years, hence this is an attractive option to implement the proposed designs.

1.4 Objectives

The numbers of soft errors occurring in digital circuits is rising due to a number of reliability challenges including reductions in feature sizes and supply voltages. Counteracting natural faults has been a subject for a number of hardware architectures in different domains. For cryptographic architectures, many research works have been carried out to achieve reliable and fault-immune structures, see, for instance, [44–49]. Moreover, concerning the finite field arithmetic architectures, various concurrent error detection (CED) multipliers for polynomial basis and normal basis of $GF(2^m)$ have been proposed using parity codes and recomputing with shifted operands (RESO) schemes [27].

Hardware redundancy techniques in such architectures leads to increase in hardware resources while making the design highly fault resilient, without considerable loss in the total time. Indeed, such a reliability assurance method for error detection is commonly used in systems where efficiency is preferred over area. On the other hand, time redundancy provides a more area effective way of detecting faults. The technique involves two runs; in the first run, the actual operands are used, during the second run the operands are encoded in such a way the original result can be obtained after the operation is complete. The encoding is usually done by shifting through RESO, rotating through recomputing with rotated operands (RERO), or other reversible encoding scheme based on the application and architecture.

Various real number division techniques for hardware have been discussed in [50], [51]. The proposed design uses a slightly modified radix-4 (Sweeney, Robertson, and Tocher) SRT division [50] to calculate the quotient and the remainder of the complex dividend and divisor. The contributions of this are summarized as follows:

- We propose fault diagnosis approaches for the presented slightly modified radix-4 SRT division considering the reliability, resiliency, and performance metrics objectives. Unified and combined fault resilience approaches are used in conjunction with performance boost modifications to achieve high throughput and frequency architectures while maintaining high error coverage. Although, such designs provide error detection capability, it is limited to 16-bit precision result. Increasing the
precision leads to larger area due to the additional logic used for error detection. As such, a reduced lookup table approach is employed to the quotient selection logic which favors in concurrent error detection. To the author’s knowledge, the aforementioned designs are not presented before.

• Through simulations for various fault models, we benchmark the error detection capabilities of the proposed schemes. The results of these simulations show acceptable error detection capabilities which ensures reliability and hardware assurance of the proposed approaches.

• Finally, we implement the proposed fault immune architectures on Xilinx Spartan-VI and Xilinx Virtex-VI FPGA families. Our results show that the proposed efficient error detection architectures can be feasibly utilized for reliable architectures of the presented complex division structures making them suitable for the required performance, reliability, and implementation metrics to achieve for constrained applications.
1.5 Thesis Outline

The structure of the thesis is as follows:

- **CHAPTER 2**: This chapter explains briefly different hardware based dividers in general. This is followed by a detailed description of Radix-β (High radix dividers) dividers. In addition, a basic description of Golub’s multiplier is also described.

- **CHAPTER 3**: This chapter describes the presented design used for complex division and the optimized ROM (quotient selection table) used in the design.

- **CHAPTER 4**: In this chapter a brief description of the utilized concurrent error detection (CED) schemes is given. The hardware designs of the two proposed fault detection schemes are described. In addition, the fault detection techniques used in different modules are also presented.

- **CHAPTER 5**: Describes the fault model used for testing the proposed designs. The chapter also discusses the error coverage results of the two architectures.

- **CHAPTER 6**: The FPGA benchmarks of two different FPGAs (Spartan-6 and Virtex-6) for the proposed architectures are given in this section.

- **CHAPTER 7**: Possible future work and applications are described in this chapter.
Chapter 2

Preliminaries

2.1 Hardware Based Dividers

Division process in hardware based systems has a similar nature to multiplication. Multiplication in general hardware systems is performed using shift and add operations, i.e., repeated or sequential additions. Likewise, division also involves repeated or sequential subtractions. Unlike multiplication operations, division techniques have the added complexity of estimating a quotient digit based on the value of the partial remainder. Another property of hardware division different from multiplication is that; in multiplication the product of two k-bit numbers can always be represented by $2^k$-bits. Whereas, the quotient of a $2^k$-bit number divided by a $k$-bit number is not always $k$-bits, it can have a size more than $k$-bits, one example of this is the division of floating point or fractional numbers. Thus, an overflow check is required before the commencement of a division operation. In addition, the division process has to be carried out from top to bottom, i.e., divisor must be subtracted from the dividend. Unlike multiplication, where the partial products can be produced from top to bottom or bottom to top. The general division is always iterative since it requires a quotient fetch phase; the division process for hardware is as follows, initially the partial remainder $r^0$ is set to $r^0 = z$, where $z$ is the dividend. In $i^{th}$ sequence, the quotient digit $q^i$ is selected based on the partial remainder $r^i$ and divisor $d$. This is followed by the subtraction of $2 \times q^i \times d$ (shifted version) from the partial
remainder $r^i$. Each successive number to be subtracted from the partial remainder has to be shifted by one bit to the left. An alternative approach is to shift the partial remainder so that it aligns with the next term to be subtracted. A better representation of the operation is given in (2.1)

$$r^i = 2 \times r^i - q^i \times (2^i \times d) \quad \text{with} \quad r^0 = z \quad \text{and} \quad r^i = 2 \times r \quad (2.1)$$

On microprocessor based systems, where there is no specific hardware for division, one can use shift and add instructions to perform division. It is to be noted that the number of sequential subtractions increase relative to the bit length of the division operands, i.e., larger the size of the dividend and divisor, greater the number of subtractions. The Fig. 2.1, below shows a clear representation of the aforementioned division technique. Most advanced arithmetic circuits use dedicated division circuits. This provides faster calculation times with low power consumption. Some examples of the most commonly used division techniques are:

- **Restoring hardware dividers**: The quotient fetch operation is performed on a non-redundant digit set $0, 1, 2, \ldots, a - 1$, where $a$ is in the power of 2. The subtraction is continued until the partial remainder reaches a value less than the divisor or until the partial remainder is negative. In the end, the addition of the divisor to the partial remainder produces the correct remainder. This step is called the restoring step, hence the name.

- **Non-restoring hardware dividers**: This divider architecture follows the same process as above but eliminates the final negative partial remainder by a slight modification. This makes it faster compared to the latter.

- **Radix-$\beta$ dividers or high radix dividers**: This architecture is the most practical to implement compared to the previous ones. Based on the radix-$\beta$, the number of quotient digits obtained each iteration can be increased hence reducing the total number of iterations.

The next subsection explains in detail the operation of radix-$\beta$ dividers.
2.2 Radix-\(\beta\) SRT Division

The most critical part of the complex divider module is the SRT divider which is applied to both the real and imaginary parts in parallel. The presented design uses a radix-\(\beta\) division algorithm which yields \(\log_2(\beta)\) quotient bits every iteration. To achieve a 16, 32, or 64 bit precision, we require \(\frac{16}{\log_2(\beta)}\), \(\frac{32}{\log_2(\beta)}\), or \(\frac{64}{\log_2(\beta)}\) iterations, respectively. In order to achieve higher precision with less iterations, higher radix-\(\beta\) (\(\beta > 64\)) SRT dividers can be used at the expense of more area on respective platforms.

The general iterative formula used in the SRT division for radix-\(\beta\) is:

\[
R[j + 1] = \beta \times (R[j] - q[j] \times D), \tag{2.2}
\]

where \(R[j]\) is the previous partial remainder after \(j\) iterations and \(R[j + 1]\) is the next partial remainder at iteration \(j\). At the end of iteration, the radix-\(\beta\) quotient is calculated based on the first few bits of the divisor \((D)\) and the partial remainder \((R[j])\).

The next partial remainder \((R[j + 1])\) is then calculated based on \(R[j]\) and the product of \(q[j]\) and the divisor \(D\). The quotient selection is done by referring to an LUT (look-up table) using the bits in \(D\) and \(R(j)\). The selected quotient bit is in the set \(\{-\alpha, ..., -3, -2, -1, 0, 1, 2, 3, ..., \alpha\}\), where \(\alpha\) is in the range \(\frac{(\beta-1)}{2} \leq \alpha \leq (\beta - 1)\).

The Robertson diagram in Fig. 2.2 shows the relation between the new shifted partial
2.2 Radix-β SRT Division

Figure 2.2: New shifted remainder and old shifted remainder in radix-β (Robertson diagram).

remainder and the old partial remainder based on the quotient. From Fig. 2.2, it is seen that the remainder is bounded by \([-hd, hd]\), where \(h\) is a constant which determines the quotient set \(\{-\alpha, ..., -3, -2, -1, 0, 1, 2, 3, ..., \alpha\}\) and \(h < 1\). Then, the new shifted partial remainder \(\beta \times R[j + 1]\) will be in the range \([-\beta \times hd, \beta \times hd]\). Applying the worst case values to the original range, we have \(\beta \times hd - \alpha \times d \leq hd\) or \(h \leq \frac{\alpha}{(\beta - 1)}\). Using this, the quotient selection digit set can be adjusted based on the specific requirements.

We can also use a p-d plot (shifted partial remainder vs divisor) as a graphical aid to better understand the quotient digit selection process. The p-d plot can also be used to derive the required precision, i.e., the number of bits (partial remainder and divisor) required for the quotient selection. For example in Fig. 2.3, assume point A is the intersection of 4 bits of p and 3 bits of d, the rectangle around A represents the truncation range from point A. The region between the lines are the quotient selection regions. The quotient \(q^i\) is selected based on the position of the uncertainty rectangle, in this case both \(q^i = \beta\) or \(\beta + 1\) will yield the correct result.

The proposed design uses a radix-4 SRT division technique. Thus, from (2.2), the division scheme for radix-4 is \(R[j + 1] = 4 \times (R[j] - q[j] \times D)\) with two quotient sets \(\{-3, -2, -1, 0, 1, 2, 3\}\) called the maximally redundant set and \(\{-2, -1, 0, 1, 2\}\) the minimally redundant set. The maximally redundant set is faster and smaller compared to the minimally redundant; nonetheless, it requires the computation of \(3 \times \) which leads to additional hardware and delay [52].

The quotient selection logic for the minimally redundant quotient set is shown in the
Figure 2.3: p-d plot quotient selection for radix-β division.
2.3 Golub’s Multiplication

Complex multiplications in DSP systems generally use a more efficient indirect approach \[53\], referred to as the Golub’s method. For two complex numbers with \( a \) and \( c \) as the

\[
q(j + 1) = \begin{cases} 
-2 & -\frac{8}{3} \times D < R[j + 1] < -\frac{4}{3} \times D \\
-1 & -\frac{5}{3} \times D < R[j + 1] < -\frac{1}{3} \times D \\
0 & -\frac{2}{3} \times D < R[j + 1] < \frac{2}{3} \times D \\
1 & \frac{1}{3} \times D < R[j + 1] < \frac{5}{3} \times D \\
2 & \frac{4}{3} \times D < R[j + 1] < \frac{8}{3} \times D
\end{cases} \tag{2.3}
\]

Figure 2.4: The quotient selection scale.

following and detailed in Fig. 2.4, where the quotient selection scale is illustrated based on:
real parts and $b$ and $d$ as imaginary parts, one can reach:

$$t_1 = (a + b) \times (c + d), \quad t_2 = a \times c, \quad t_3 = b \times d$$

$$x = (t_2 - t_3) + j \times (t_1 - t_2 - t_3)$$

$$x_{\text{real}} = t_2 - t_3$$

$$x_{\text{imag}} = t_1 - t_2 - t_3$$

(2.4)

Figure 2.5: Golub’s multiplication.

The direct implementation of complex multiplication requires four real multiplications and two additions. The indirect implementation, on the other hand, requires three real multiplications and five additions, as shown in Fig. 2.5. We note that the latter is more area efficient because multiplication requires more area compared to addition.
Chapter 3

Presented Complex Division Radix-4 SRT Module

The design of the complex divider consists of several modules: multiplication module, normalizing module, real and imaginary iteration module, shared ROM for the quotient selection, and an on-the-fly converter. Fig. 3.1 shows the high level block diagram of the complete design. In this figure, the multiplication in the numerator is performed using Golub’s multiplication method. This leads to fewer resources with lower hardware complexity compared to the traditional multiplication approach. The multiplication module multiplies the complex conjugate of the denominator to the numerator and denominator to rationalize the denominator to a real number, i.e., \( c^2 + d^2 \):

\[
\frac{a + jb}{c + jd} = \frac{(a + jb) \times (c - jd)}{(c + jd) \times (c - jd)} = \frac{x}{c^2 + d^2} + j\frac{y}{c^2 + d^2}. \tag{3.1}
\]

The normalizer normalizes the numbers such that \( 1 \leq x, y, (c^2 + d^2) \leq 2 \). The real and imaginary iteration modules are essentially two radix-4 SRT blocks. As seen in Fig. 3.1, a single dual-port ROM is shared between the real and imaginary modules for quotient selection. The normalized divisor is saved in register \( D \) and the dividend is saved in \( U, V \) in carry-save form (for both real and imaginary sections), as shown in Fig. 3.1. The dividend is then subjected to repeated subtractions and table look-up to obtain the required number of quotient bits, in this case 16 bits. The MUX in this figure selects
the divisor $D$ based on the value of the quotient fetched (initial 5 bits of $U$ and $V$ and 4 bits of $D$), i.e., $0, D, -D, 2D,$ and $-2D$. The carry-save adder (CSA) calculates the new partial remainder based on the MUX output and the present values and saves it in $U$ and $V$. This process is repeated until the desired number of quotient bits is reached for both real and imaginary parts. The on-the-fly converter is essentially an adder which subtracts the positive quotients from the negative quotients and saves them in $Q_{\text{real}}$ and $Q_{\text{imag}}$, respectively. The shared ROM used for quotient fetch is minimized to save area; the minimized ROM structure is explained in the followings.

### 3.1 ROM

The ROM look-up consists of two steps; rounding and quotient fetch. The partial remainder from the CSA is rounded off to 5 bits, round $(R[j])$. A combination of the partial remainder (5 bits) and the divisor (4 bits) is used as the address to the ROM for the quotient fetch. The ideal ROM table contains a total of $2^9 = 512$ entries as shown in Fig. 3.2.

The ROM is symmetrical between the positive and negative quotients. When the round $(R[j])$ is positive, the positive quotient is fetched and vice versa. The shared ROM
used in the design is condensed to just the positive section of the table and a combinational circuit which generates the 2’s complement of \( q[j] \) if \( R[j] \) is negative. This can easily be determined by verifying the MSB of \( R[j] \) which reduces the ROM size to half of the original size, i.e., to 256 entries.

The “quotient fetch” operation pertaining to the minimized shared ROM architecture is presented below:

\[
\text{round}(R_R[j]) = r_R^2 r_R^1 r_R^0 r_R^{-1} r_R^{-2} \quad (3.2)
\]

\[
\text{round}(R_I[j]) = r_I^2 r_I^1 r_I^0 r_I^{-1} r_I^{-2} \quad (3.3)
\]

\[
addr_R = \begin{cases} 
  r_R^2 r_R^1 r_R^0 r_R^{-1} r_R^{-2} | d^0 d^{-1} d^{-2} d^{-3} & \text{if } r_R^2 = 0 \\
  \text{not} \left(r_R^2 r_R^1 r_R^0 r_R^{-2}\right) + 1 | d^0 d^{-1} d^{-2} d^{-3} & \text{else}
\end{cases} 
\quad (3.4)
\]

\[
addr_I = \begin{cases} 
  r_I^2 r_I^1 r_I^0 r_I^{-1} r_I^{-2} | d^0 d^{-1} d^{-2} d^{-3} & \text{if } r_I^2 = 0 \\
  \text{not} \left(r_I^2 r_I^1 r_I^0 r_I^{-2}\right) + 1 | d^0 d^{-1} d^{-2} d^{-3} & \text{else}
\end{cases} 
\quad (3.5)
\]

\[
q_R(j) = \begin{cases} 
  q_R^2 q_R^1 q_R^0 & \text{if } r_R^2 = 0 \\
  \text{not} \left(q_R^2 q_R^1 q_R^0\right) + 1 & \text{else}
\end{cases} 
\quad (3.6)
\]

\[
q_I(j) = \begin{cases} 
  q_I^2 q_I^1 q_I^0 & \text{if } r_I^2 = 0 \\
  \text{not} \left(q_I^2 q_I^1 q_I^0\right) + 1 & \text{else}
\end{cases} 
\quad (3.7)
\]
As shown in (3.4) and (3.5), the first 4 bits of $D$ and $R (U + V)$ are concatenated to form the ROM addresses. The first bit of $R (U + V)$ is used as a select line for the multiplexers to select the two inputs, i.e., original or 2’s complement. Similarly, in (3.6) and (3.7), the same bit of $R (U + V)$ is used to determine the output $q[j]$ or $-q[j]$. The circuit operation of the proposed ROM is denoted in Fig. 3.3.

Figure 3.3: Minimized ROM.
Chapter 4

Proposed Error Detection Schemes

The most prominent method for error detection is information redundancy (usually divided into time and hardware redundancy). General hardware redundancy, though simple, is very efficient in detecting faults. We do not present a modular hardware redundant scheme separately, for the sake of brevity, but its implementation has been carried out to record results. Specifically, since the scheme is general hardware duplication, all the registers in the data path are duplicated and the content is compared with its duplicate in real time. The main drawback of this scheme is that it results in a 100% increase in area. Register duplication is not just limited to the registers in the data path but also in the ROM leading to an increased ROM size of $512 \times 3$. Other arithmetic blocks in the design are also duplicated and checked in real time. A more practical approach and low-complexity technique is using error detecting codes, such as parity, for error detection. The logic relation for parity is simple to implement and is more area efficient compared to duplication. The challenging part is incorporating the parity checker in the proposed design. Error checks need not be performed on all the sections of the design. It is sufficient to check the parts which are capable of propagating errors. This reduces the area by eliminating unnecessary logic in the design.

The second scheme incorporates time redundancy using RESO. Using this technique, an internment error, i.e., transient faults occurring during one of the runs can be detected; moreover, it can detect permanent faults. This provides RESO with an added advantage
compared to traditional time redundancy schemes. As mentioned before RESO involves two runs, one uses the actual operands and the other uses shifted operands. Since this is a division operation there is an added advantage in using RESO for error detection, the shifting performed in RESO is a mere multiplication by 2 in the numerator and the denominator which leads to the same final result, i.e., \( \frac{(a+jb)\times 2}{(c+jd)\times 2} = X + jY = \frac{(a+jb)}{(c+jd)} \).

This requires no additional decoding hardware at the end of the operation to obtain the original result. In order to incorporate all possible \( 2^n \) combinations of an \( n \)-bit register in RESO, an \( n + 1 \) bit register design is required.

In what follows, we present two error detection schemes, where details for the error detection technique and implementation used in the design are presented.

### 4.1 Unified Parity Check and Hardware Redundancy

To implement an effective error detection approach, the propagation of faults throughout the circuit needs to be assessed. In the proposed complex divider architecture, the occurrence of single or multiple faults may lead to random error propagation through the circuit. Considering the operation’s iterative nature, faults may also propagate to circuit locations which lay before the affected region. This leads to the inclusion of parity registers throughout the circuit which are prone to propagating faults. The error detection structure is developed by comparing the actual parity and the predicted parity. The error detection division architecture is divided into 5 modules; each individual module has its own parity or hardware redundancy schemes as described in the following. Fig. 4.1, shows the main CED blocks in the datapath of the proposed error detection scheme. The following subsections explain the individual fault technique applied to the modules.
Figure 4.1: Radix-4 complex SRT module with unified parity check and hardware redundancy.
4.1 Unified Parity Check and Hardware Redundancy

4.1.1 Golub’s Multiplier

As mentioned before, Golub’s technique of complex multiplication is the most efficient way to multiply two complex numbers. For error detection in this particular module, we use the checker 3 method mentioned in [54]. This was found to be the most efficient in terms of area compared to other error detection techniques for adders and multipliers. Such an implementation slightly alters the design of the multiplication module presented in Fig. 2.5 in order to incorporate the error detection scheme. This separates the calculation of real and imaginary parts and, in turn, prevents the error propagation from \(a \times c + b \times d\) to \(x_{\text{imag}}\) as shown in Fig. 2.5. The checker verifies (4.1),(4.2) which are obtained by rearranging (2.3). The structure of the concurrent error detection multiplier is shown in Fig. 4.2.

\[
x_{\text{real}} + x_{\text{imag}} = t_1 - 2 \times b \times d, \quad (4.1)
\]

\[
x_{\text{real}} + x_{\text{imag}} + 2 \times b \times d = t_1. \quad (4.2)
\]

4.1.2 Datapath Registers

The registers in the datapath are very vital to the operation and are also capable of randomizing the error propagation in the design. Each register is incorporated with a parity bit and the contents are checked in real time to detect faults. The collective output comparisons of the actual and predicted parity are connected to an OR gate which rises an error flag in case of a detection. Fig. 4.3, shows a generalized representation of the theory explained above. The bit P is the individual parity bit of registers R1, R2, \ldots, Rn. Some cases may also contain multiple parity bits in the same register, to detect errors [55, 56].
Figure 4.2: CED Multiplier.

Figure 4.3: Parity check registers.
4.1.3 ROM (Quotient Selection Logic)

The memory module used in the design is very critical to the operation and, thus, if it is prone to faults, it can undermine the entire objectives. The width of the ROM is extended to one extra bit to incorporate parity. The arrangement is capable of detecting all single stuck-at faults. During a quotient fetch operation, the quotient corresponding to the address bits is checked for correctness at the output of the ROM using the respective parity bits. In the case of a bad memory location, the circuit raises an error flag. The ROM with the proposed parity prediction logic is depicted in Fig. 4.4.

4.1.4 Carry Save Adder

The proposed design uses a carry-save adder to compute the next partial remainder based on the previous partial remainder $R[j]$ and the product of the quotient and divisor, i.e., $q[j] \times D$. Since the design is aimed towards error detection, CSAs provide faster computation of the results as the carry is not propagated but saved; this reduces the delay of the adder. In addition to this, the CSAs provide easy incorporation of error detection. The error detection scheme used in the CSA incorporates parity checks for individual full adders [57]. The parity check equations (4.3),(4.4) used in the design is given below:

$$\text{Sum}_e = a \oplus b \oplus \text{cin} \oplus \text{Sum},$$  \hspace{1cm} (4.3)
4.2 Error Detection through RESO

In this error detection model, we use RESO \[27\] to detect faults in the design. The RESO method, as explained before, uses the same hardware without any modification. This makes it efficient in applications where low-area is a requirement. Let us assume \( F \) is the function to be performed on a particular operand \( x \). Then, \( F(x) \) is the result of the functional module. The initial result \( F(x) \) is stored in a register. The operation is repeated with a shifted version of the same operand \( x \), in this case \( x' \). The operation is repeated with \( x' \) to obtain \( F(x') \), in such a way the original result \( F(x) \) can be restored with a simple operation on \( F(x') \). The division architecture for the implemented RESO

Figure 4.5: Full added with parity check.

\[
C_e = (a \cdot b) \oplus (a \oplus b) \cdot (cin \oplus cout).
\] (4.4)

The parity check equations are used to check the correctness of both “sum” and “carry”. In the case where one of them is incorrect, the module raises an error flag. The structure for a single fault detection full adder is shown in Fig. 4.5. In case of a \( n \)-bit full adder this can be arrayed, like in a traditional \( n \)-bit adder and the error signals can be connected to an OR gate.

4.2 Error Detection through RESO

In this error detection model, we use RESO \[27\] to detect faults in the design. The RESO method, as explained before, uses the same hardware without any modification. This makes it efficient in applications where low-area is a requirement. Let us assume \( F \) is the function to be performed on a particular operand \( x \). Then, \( F(x) \) is the result of the functional module. The initial result \( F(x) \) is stored in a register. The operation is repeated with a shifted version of the same operand \( x \), in this case \( x' \). The operation is repeated with \( x' \) to obtain \( F(x') \), in such a way the original result \( F(x) \) can be restored with a simple operation on \( F(x') \). The division architecture for the implemented RESO
structure is shown in Fig. 4.6.

As shown in Fig. 4.6, the size of the registers are increased to an extra bit to incorporate shifting of all $2^8$ possible combinations. The applied RESO method in this case is limited to one left shift, i.e., multiplication by 2. Multiple left shifts are possible, but at the expense of increased register size which leads to increased CSA, ROM, and address registers. In accordance with the principle explained above, the initial complex dividend and divisor are passed through the divider. The quotient is stored in a register. In the second run, the initial operands are shifted and passed through the divider. This new result is compared with the previous quotient and in case of a mismatch, an error flag is triggered. Time-redundancy techniques tend to increase the number of cycles for a particular operation. Hence, we have sub-pipelined the design in order to alleviate the throughput. The addition of the pipeline registers does indeed increase the area slightly, but it is negligible compared to hardware redundant or parity-based schemes.
Figure 4.6: Proposed divider with RESO for fault diagnosis.
Chapter 5

Error Simulations

For the fault model in this thesis, both single and multiple stuck-at-fault are considered to cover natural failures and counteract VLSI defects [58]. The fault model is constructed using linear-feedback shift registers (LFSRs) through maximum tap length polynomials, which are ORed or ANDed with the actual output to simulate both stuck-at-one and stuck-at-zero faults for transient and permanent faults [58]. The faults are injected at different locations in the circuit and checked for the error indication flags. This provides a more real-world scenario, because naturally-occurring faults does not affect a particular part of the circuit but the entire circuit as a whole with a uniform distribution. Moreover, the false-positive cases are excluded from the error analysis, i.e., the cases where the injected faults also produce the same output as the original. The same fault model is used for both the presented error detection modules (parity and RESO). The model simulates both single and multiple faults in the circuit by flipping the bits from 0 to 1 and vice versa. Since we use a 16-bit LFSR, the probability of the bits flipping is $\frac{1}{2^{16}}$ and the error stays for exactly one clock cycle. This perfectly simulates both transient and permanent faults.

The first proposed scheme uses a combination of signatures and hardware redundancy and it can be analytically proven that the detection rate for single-bit stuck-at-fault for parity prediction blocks is 100%. The simulations for single stuck-at-fault are performed exhaustively in every byte and every operation to confirm the theoretical results. We
mainly concentrate on the multiple stuck-at-fault scenarios, by performing extensive analysis through simulations. For hardware redundant blocks, it is highly unlikely for two transient faults or permanent faults of the same nature to occur simultaneously. Hence, the detection rate for these modules will achieve a 100% error coverage. As for parity prediction blocks used in the design, the faults may or may not be detected, based on the parity they generate. To counter this, different parts of the circuit are equipped with parity blocks so that at least one of them alarm the errors. The RESO module uses two runs to detect faults, both permanent and transient ones. Since the operands are shifted on the second run, a transient fault leads to different results, because both the shifted and original operands are supposed to generate the same result. A similar condition occurs in the case of a permanent fault. Consider a stuck-at-one fault at the LSB of a register in the design. During the shifted second run, the fault at the LSB, leads to different results due to the iterative nature of the design.

The simulations have been performed by applying 1,000 random inputs and 9,17,504 multiple faults. The results of the simulations show more than 99.999% of the errors are detected for both the hardware and time redundant modules. A theoretical analysis has also been performed to confirm the simulation results. Let \( p \) be the probability of a parity module detecting a fault. Then, \( 1 - p^n \) is the probability of error detection for \( n \) parity modules. The hardware redundant modules consist of a total of \( 6 \times 2 + 1 \) parity modules (real and imaginary iterative modules). Each iterative module performs 8 iterations. Therefore, we have \( p = 0.5 \) and \( n= 13 \times 8 \times 2 \), the error detection probability is calculated as \( 1 - 0.5^{208} = 99.999\% \), which matches with the simulation results.
Chapter 6

FPGA Implementations and Benchmark

In this section, through FPGA implementations on two diverse families, we present the overhead evaluation results, i.e., area, delay, and throughput. The benchmarking has been carried out for the original and the error detection structures of the discussed complex division SRT module. The FPGA implementations are done using ISE version 14.5 and synthesized for Spartan-6 xc6slx16-2cgs324 and Virtex-6 xc6vlx75t-3ff484 devices [77]. Through this analysis, the performance and implementation metrics of the complex divider for both low-end and high-end FPGAs can be observed with VHDL as the design entry.

The complex division architecture is structured hierarchically. Specifically, it is divided into 4 functional modules: Golub’s multiplier, normalizer, SRT divider (real and imaginary), and ROM. Each of these parts is implemented individually and port-mapped at the top level. The error detection designs for individual modules are tested to verify functionality. The original design (no error detection) runs at a maximum frequency of 268.557 MHz for the Virtex-6 and 139.758 MHz for the Spartan-6, after performing synthesis and place-and-route. Each iteration takes two clock cycles; the operation requires a total of eight iterations. The critical path delay of 5.23 ns for the Spartan-6 and 3.76 ns for the Virtex-6 is considered from the input of the Golub’s multiplier through the ROM
Table 6.1: FPGA Implementation results of the original complex division module and the proposed error detection designs on Spartan-6 FPGA device xc6s1x16.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Area [Slices]</th>
<th>Throughput [Gbps]</th>
<th>Power [mW]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>228</td>
<td>0.559</td>
<td>5.43</td>
</tr>
<tr>
<td>Scheme-I</td>
<td>304 (33.33%)</td>
<td>0.510 (8.76%)</td>
<td>5.79 (6.62%)</td>
</tr>
<tr>
<td>Scheme-II</td>
<td>269 (17.98%)</td>
<td>0.367 (34.34%)</td>
<td>6.60 (21.54%)</td>
</tr>
</tbody>
</table>

Table 6.2: FPGA Implementation results of the original complex division module and the proposed error detection designs on Virtex-6 FPGA device xc6vlx75t.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Area [Slices]</th>
<th>Throughput [Gbps]</th>
<th>Power [mW]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>248</td>
<td>1.074</td>
<td>9.86</td>
</tr>
<tr>
<td>Scheme-I</td>
<td>346 (39.52%)</td>
<td>1.064 (0.93%)</td>
<td>10.93 (10.85%)</td>
</tr>
<tr>
<td>Scheme-II</td>
<td>283 (14.11%)</td>
<td>0.677 (36.64%)</td>
<td>12.36 (25.35%)</td>
</tr>
</tbody>
</table>

till the output of the quotient register. The total delay of Scheme-I (unified parity and hardware redundancy design) is 5.237 ns (Spartan-6) and 1.600 ns (Virtex-6). The total delay increase from the original design to Scheme-I is 36.452% (Spartan-6) and 29.56% (Virtex-6), this is due to storing the result in carry save form rather than using traditional CPA’s, which may lead to higher delays. Scheme-II employs the RESO architecture discussed in previous sections and it leads to additional processing time since it requires twice the original number of iterations, eight for the original operands and eight for the shifted operands. As mentioned before, the design in Scheme-II has been sub-pipelined to relatively improve the throughput. A delay increase of 16.326% (Spartan-6) and 10.625% (Virtex-6) from the original design is observed. Though, this leads to an increase in area, it is low compared to Scheme-I. The synthesis and place and route are performed using the same settings, i.e., speed optimization, minimum area, and no DSP blocks, for both FPGA platforms.

The implementation results are tabulated in Tables 6.1 and 6.2 and the area (slices), throughput, and power consumption (at 140 MHz) for both designs are presented. The overheads are also denoted in the parentheses in both tables. The maximum area over-
heads are 33.33% (Spartan-6) and 39.516% (Virtex-6) which correspond to Scheme-I and 17.98% (Spartan-6) and 14.11% (Virtex-6) for Scheme-II, respectively. The maximum power overheads are 6.62% (Spartan-6) and 10.85% (Virtex-6) which correspond to Scheme-I and 21.54% (Spartan-6) and 25.35% (Virtex-6) for Scheme-II, respectively. As for Scheme-II, sub-pipelining the design leads to higher frequencies at the expense of area. Since Scheme-II is aimed towards area-efficient designs compared to Scheme-I, further sub-pipelining has not been implemented to maintain the area within acceptable limits. Based on the simulation results, both schemes provide the high error coverage of 99.99%. One can refer to [59]-[76] for similar sub-block works on fault detection in cryptography.
Chapter 7

Conclusion

We have presented two complex divider architectures capable of error detection. The division scheme uses the SRT division algorithm to obtain the quotient and remainder. We also propose a new ROM look-up technique which reduces the number of ROM entries 50%, and, thus, significantly reduces the area and power consumption. Scheme-I utilizes signatures and hardware redundant blocks to incorporate error detection. Traditional error detection architectures are limited to the use of either parity (less area but not complete error coverage) or hardware redundancy (maximum error coverage but impractical hardware complexity). Scheme-I uses a combination of both the former and latter to achieve maximum error coverage using relatively less hardware complexity. Scheme-II achieves its error detection by utilizing redundancy in time. Due to the nature of the operation, no extra logic is required to compare the results, apart from the pipeline registers. Benchmark circuits were evaluated with both single and multiple stuck-at-fault. The error coverage simulations show results of more than 99.999% coverage for the proposed designs. The proposed designs are extendable through increasing the efficiency of the original design with fault detection capability by moving to higher radices such as radix-8 or radix-16. Moreover, they provide different options to embed error detection in many complex time-efficient and area-efficient arithmetic applications.
References


