Reliable Radix-4 Complex Division for Fault-Sensitive Applications

Mehran Mozaffari-Kermani, *Member, IEEE,* Niranjan Manoharan, and Reza Azarderakhsh, *Member, IEEE*

***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 paper, we present schemes to provide complex number division architectures based on Sweeney, Robertson, and Tocher-division with error detection mechanisms. Different error detection architectures are proposed in this paper 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 to be able to detect faults based on recomputations for different operands in addition to the unified parity (simplified detecting code) and hardware redundancy approach. 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 error detection assessments are performed for the proposed designs through fault simulations and field-programmable gate array (FPGA) implementations; the design is implemented on Xilinx Spartan-6 and Xilinx Virtex-6 FPGA families.**

***Index Terms*—Concurrent error detection (CED), field-programmable gate array (FPGA), recomputing with shifted operands (RESO), Sweeney, Robertson, and Tocher (SRT)-division.**

I. Introduction

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 multiplyadd 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 utilizes the standard formula for complex division but uses an optimized architecture to reduce the area and improve the operating frequency. Another complex divider is implemented using the coordinate rotational digital computer-like algorithms [15]. There exist other complex division techniques which are based on complex binary number system [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 algorithm to calculate the results by converting the division operation to a system of linear equations [18]. Implementing this technique is 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.

In digital systems, errors can happen through various causes including alpha particles from package decay, cosmic rays creating energetic neutrons and protons, and thermal neutrons. Counteracting natural faults has been a subject for a number of hardware architectures in different domains. In signal processing (see [19], [20]), and for cryptographic architectures, many research works have been carried out to achieve reliable and fault-immune structures (see [21]–[33]). 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 [34], [35]

Manuscript received May 15, 2014; revised August 7, 2014, December 5, 2014, and December 30, 2014; accepted January 6, 2015. Date of publication January 21, 2015; date of current version March 17, 2015. This work was supported by the Texas Instruments Faculty Award, granted to M. Mozaffari-Kermani and R. Azarderakhsh. This paper was recommended by Associate Editor H. Stratigopoulos.

M. Mozaffari-Kermani and N. Manoharan are with the Department of Electrical and Microelectronic Engineering, Rochester Institute of Technology, Rochester, NY 14623 USA (e-mail: m.mozaffari@rit.edu).

R. Azarderakhsh is with the Department of Computer Engineering, Rochester Institute of Technology, Rochester, NY 14623 USA.

Digital Object Identifier 10.1109/TCAD.2015.2394389

and recomputing with shifted operands (RESO) schemes [36].

Using redundancy techniques such as time redundancy [36] and hardware redundancy [37] is one of the most efficient methods to implement error detection in an existing design. Hardware redundancy techniques in such architectures lead to increase in hardware resources while making the design highly fault immune, 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 through encoded operands provides a more area effective way of detecting faults. The technique involves two runs; in the first run, the actual operands are used and during the second run, the operands are encoded in such a way that 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, or other reversible encoding schemes based on the application and architecture.

Various real number division techniques for hardware have been discussed in [38] and [39]. The proposed design uses a slightly modified radix-4 Sweeney, Robertson, and Tocher (SRT) division [38] to calculate the quotient and the remainder of the complex dividend and divisor. The contributions of this paper are summarized as follows.

1) We propose error detection approaches for the presented slightly modified radix-4 SRT division considering the reliability and performance metrics objectives. Unified and combined error detection approaches are used in conjunction with performance boost modifications to achieve high throughput and frequency architectures while maintaining high error coverage. Increasing the precision leads to larger area due to the additional logic used for error detection. As such, a reduced look up table (LUT) approach is employed to the quotient selection logic which favors in CED.
2) We implement the proposed fault immune architectures on Xilinx Spartan-6 and Xilinx Virtex-6 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 for constrained applications.
3) Finally, through simulations, 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 for the proposed approaches.

This paper is organized as follows. Section II discusses the preliminaries regarding radix-4 SRT division. Section III explains the presented radix-4 SRT division scheme. Section IV describes the proposed error detection techniques. Section V provides the implementation results followed by the simulation results in Section VI. Finally, the conclusion is given in Section VII.

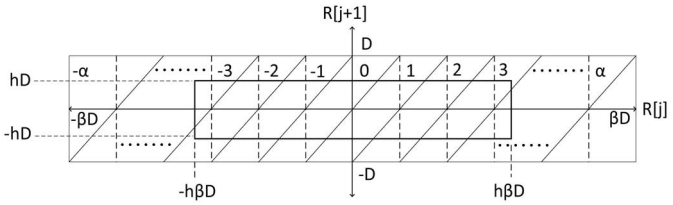

Fig. 1. New shifted remainder and old shifted remainder in radix-β (Robertson diagram).

II. Preliminaries

In what follows, we present the preliminaries for the SRT division as well as Golub's multiplication.

A. Radix-β 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-β division algorithm which yields $\log_2 \beta$ quotient bits every iteration. To achieve a 16, 32, or 64 bit precision, we require $16/\log_2 \beta$, $32/\log_2 \beta$, or $64/\log_2 \beta$ iterations, respectively. In order to achieve higher precision with less iterations, higher radix-β ($\beta > 64$) 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-β is

$$R\left[j+1\right] = \beta \times \left(R\left[j\right] - q\left[j\right] \times D\right) \tag{1}$$

where $R[j]$ is the previous partial remainder after j iterations, $R[j+1]$ is the next partial remainder at iteration $j+1$, and $q[j]$ is the quotient obtained at iteration j. At the end of iteration, the radix-β 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 using the bits in D and $R(j)$. The selected quotient bit is in the set $\{-\alpha, \ldots, -3, -2, -1, 0, 1, 2, 3, \ldots, \alpha\}$, which is sometimes referred to as quotient selection set from which the value of $q[j]$ is fetched every iteration, where α is in the range $\log_2 \beta \leq \alpha \leq (\beta - 1)$.

The Robertson diagram in Fig. 1 shows the relation between the new shifted partial remainder and the old partial remainder based on the quotient. From Fig. 1, it is seen that the remainder is bounded by $[-hd, hd)$. Generally, h is a constant which is used to restrict the quotient set $\{-\alpha, \ldots, -3, -2, -1, 0, 1, 2, 3, \ldots, \alpha\}$. In most general cases, $h = 1$ and quotient selection requires $2\alpha + 1$ comparisons and a worst case computation which requires $q[j] \times D$ for the next partial remainder ($q[j] = +\alpha/-\alpha$, for worst case). The number of comparisons and the computation can be reduced by setting the value of $h < 1$, i.e., scaling the partial remainder by β to have $[-\beta \times hd, \beta \times hd)$. Thus, any value of h which satisfies the condition $\beta \times hd - \alpha \times d \leq hd$ or $h \leq \alpha/(\beta - 1)$ can be used to reduce the quotient set, where $\log_2 \beta \leq \alpha \leq (\beta - 1)$. It is noted that even though α can be its minimum value, the precision decreases as the

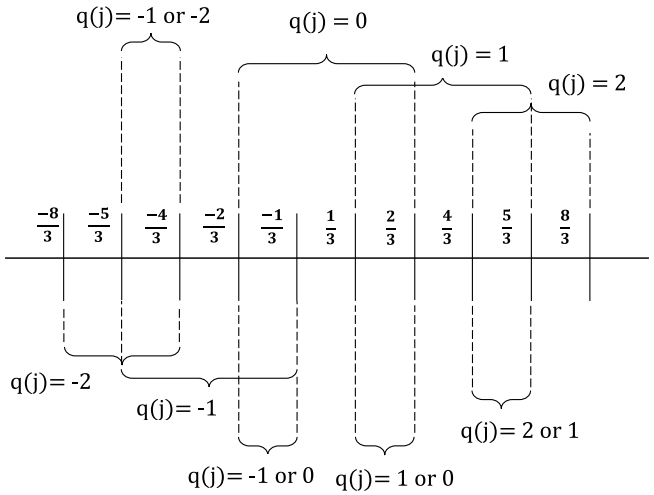

Fig. 2. Quotient selection scale.

number of comparisons decreases, requiring higher resolution (more address bits). On the contrary, this eliminates the need for the worst case $q\left[j\right] \times D$ and because $\alpha = \log_2 \beta$, the multiplication can be implemented by a shifting operation. Using this, the quotient selection digit set can be adjusted and modeled based on the specific requirements (precision versus speed).

The proposed design uses a radix-4 SRT division technique. Thus, from (1), the division scheme for radix-4 is $R\left[j+1\right] = 4 \times \left(R\left[j\right] - q\left[j\right] \times D\right)$ with two quotient sets; $\{-3, -2, -1, 0, 1, 2, 3\}$ is called the maximally redundant set and $\{-2, -1, 0, 1, 2\}$ is referred to as the minimally redundant set. The maximally redundant set requires less address bits to determine the quotient; thus, it requires a smaller LUT compared to the minimally redundant; nonetheless, it requires the computation of $3\times$ which leads to additional hardware and delay [40]. We would like to emphasize that the quotient $q\left[j\right]$ is obtained once every iteration; this is based on the quotient digit set which is determined for a particular radix based on the constant h. In turn, h can also be used to reduce the size of the quotient set ($\{-3, -2, -1, 0, 1, 2, 3\}$ or $\{-2, -1, 0, 1, 2\}$) based on the requirements (speed versus precision), this determines the precision and speed of the implemented hardware.

In the case of a maximally redundant set $\{-3, -2, -1, 0, 1, 2, 3\}$, the overlap regions between the quotient are much larger compared to the minimally redundant set $\{-2, -1, 0, 1, 2\}$. Thus, the former can operate on less precision (less address bits) compared to the latter. This leads to a smaller ROM table but it requires the computation of $3\times$ which requires a $3\times$ multiplier (additional hardware). In addition, this also leads to more delay due to carry propagation in the multipliers adder tree.

The minimally redundant quotient set is designed using $h = 2/3$. At the start of each iteration, a quotient digit is selected which will result in the next partial remainder $R\left[j+1\right]$ within the bounds. The quotient selection logic for the minimally redundant quotient set is shown in the following

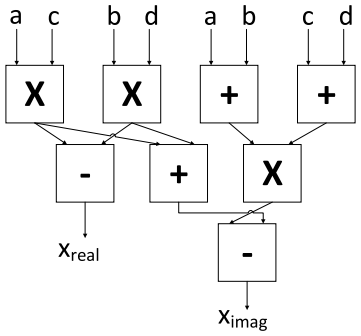

Fig. 3. Golub's multiplication.

and detailed in Fig. 2, where the quotient selection scale is illustrated based on:

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

B. Golub's Multiplication

Complex multiplications in DSP systems generally use a more efficient indirect approach [41], referred to as the Golub's method. For two complex numbers with $y = a + bj$ and $z = c + dj$, one can reach

$$\begin{aligned} x = y \times z &= (t_2 - t_3) + j \times (t_1 - t_2 - t_3) \\ x_{\text{real}} &= t_2 - t_3 \\ x_{\text{imag}} &= t_1 - t_2 - t_3 \\ t_1 = (a+b) \times (c+d),\ t_2 &= a \times c,\ t_3 = b \times d. \end{aligned} \quad (3)$$

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. 3. We note that the latter is more area efficient because multiplication requires more area compared to addition.

III. Presented Complex Division Radix-4 SRT Module

The design of the complex divider consists of several modules: 1) multiplication module; 2) normalizing module; 3) real and imaginary iteration module; 4) shared ROM for the quotient selection; and 5) an on-the-fly converter. Fig. 4 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. 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., $c^2 + d^2$, as $a + jb/c + jd = ((a + jb) \times (c + j(-d)))/((c + jd) \times (c + j(-d))) = (((a \times c) + (b \times d))/c^2 + d^2) + j(((b \times c) - (a \times d))/c^2 + d^2) = (N_{\text{real}}/c^2 + d^2) + j(N_{\text{imag}}/c^2 + d^2) = Q_{\text{real}} + jQ_{\text{imag}}$.

The normalization of numbers is performed by shifting the numbers such that $1 \leq N_{\text{real}}, N_{\text{imag}}, \left(c^2 + d^2\right) \leq 2$. The real

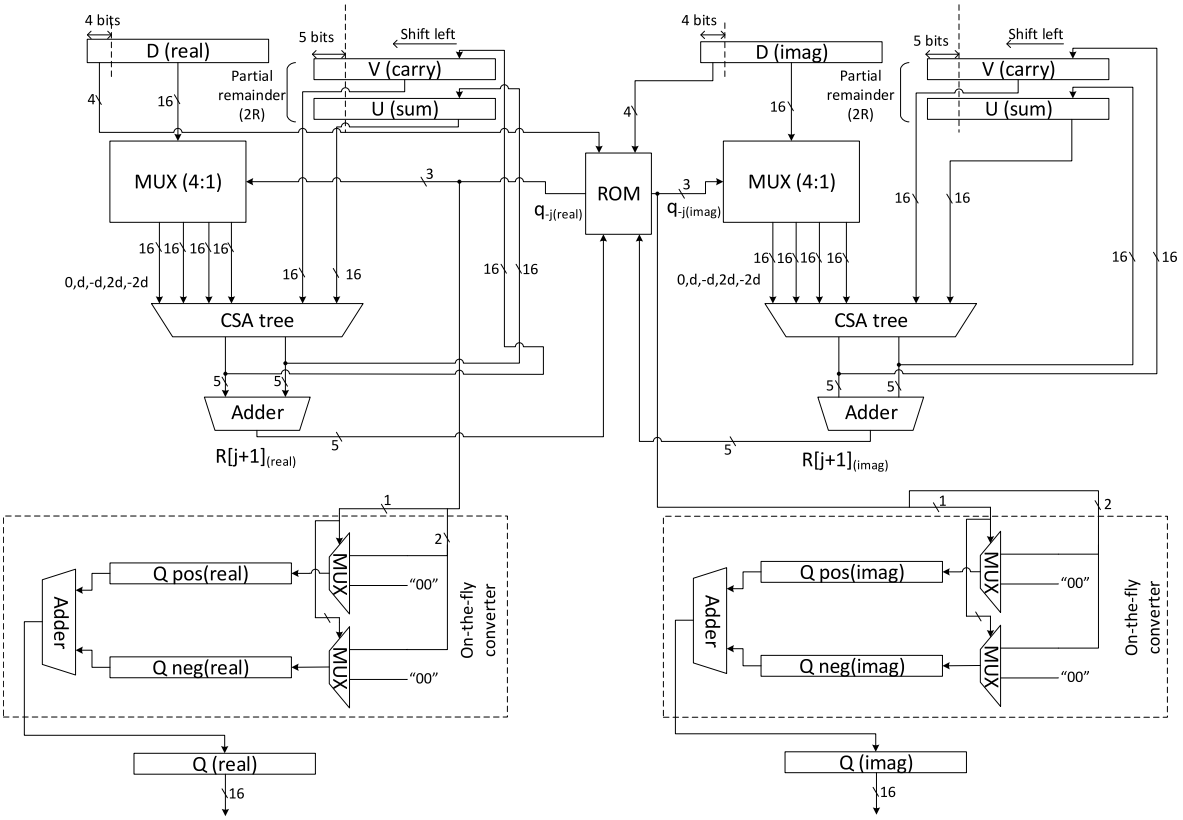

Fig. 4. Presented complex radix-4 SRT module.

and imaginary modules are essentially two radix-4 SRT blocks. As seen in Fig. 4, a single dual-port ROM is shared between the real and imaginary modules for quotient selection. The normalized divisor $c^2 + d^2$ is saved in register D and the dividend $N_{\text{real}}, N_{\text{imag}}$ is saved in U (sum), V (carry) in carry-save form (for both real and imaginary sections), the registers are shown in Fig. 4. Since the division process is a sequential one, the division is performed by repeated subtractions. The first quotient is fetched using the first few bits of D (4 bits) and U, V (5 bits). Since the partial remainder is in carry save form, an adder is used to generate the required address bits, i.e., address = U (sum) + V (carry). The quotient is saved in registers *Qpos* if the quotient is positive or in *Qneg* if it is negative. The MUX in the figure selects the product $q\left[j\right] \times D$ which can be 0, D, $-D$, $2D$, or $-2D$ based on the quotient set $\{-2, -1, 0, 1, 2\}$. The output of the MUX is then subtracted from the partial remainder and the result is stored in U (sum) and V (carry) in carry save form. 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 *Qreal* and *Qimag*, respectively. Fig. 5 shows the selection relationship between D, U, V, and $q\left[j\right]$.

Our design was focused on a balance between area, precision, and speed, and for this, the minimally redundant set was the most ideal selection. The overlap regions for the radix-4 with $\{-2, -1, 0, 1, 2\}$ as the quotient set can be seen from Fig. 5. When the selection for the partial remainder falls within these ranges, the selected quotients can be one of the two quotients within that range. In cases which use higher radices, such as radix-8, the overlap regions in the quotient set are large (span additional range of address bits). Accordingly, the quotient can be selected based on how close it is to the nonoverlap regions, this aids in precision of the final quotient; this is usually done based on the last few bits of the address and requires additional hardware. Since the overlap region is very small for our particular case, it was an acceptable trade-off between the additional hardware and precision. The shared ROM used for quotient fetch is minimized to save area; the minimized ROM structure is explained in the following.

A. ROM

The ROM look-up consists of two steps: 1) rounding and 2) quotient fetch. The partial remainder from the carry save adder (CSA) is rounded off to 5 bits, round $\left(R\left[j\right]\right)$. 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 and each entry is 3 bits wide as shown in Fig. 4.

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 2s 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

(U+V)/D	1	1.0625	1.125	1.1875	1.25	1.3125	1.375	1.4375	1.5	1.5625	1.625	1.6875	1.75	1.8125	1.875	1.9375
3.75								2	2	2	2	2	2	2	2	2
3.5							2	2	2	2	2	2	2	2	2	2
3.25					2	2	2	2	2	2	2	2	2	2	2	2
3				2	2	2	2	2	2	2	2	2	2	1,2	1,2	1,2
2.75		2	2	2	2	2	2	2	2	2	2	1,2	1,2	1,2	1,2	1,2
2.5	2	2	2	2	2	2	2	2	2	1,2	1,2	1,2	1,2	1	1	1
2.25	2	2	2	2	2	2	1,2	1,2	1,2	1,2	1,2	1	1	1	1	1
2	2	2	2	2	2	1,2	1,2	1,2	1	1	1	1	1	1	1	1
1.75	2	1,2	1,2	1,2	1,2	1	1	1	1	1	1	1	1	1	1	1
1.5	1,2	1,2	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1.25	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0,1
1	1	1	1	1	1	1	1	1	1	0,1	0,1	0,1	0,1	0,1	0,1	0,1
0.75	1	1	1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1
0.5	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0	0	0	0	0	0	0	0
0.25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-0.25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-0.5	0,-1	0,-1	0,-1	0,-1	0,-1	0,-1	0,-1	0,-1	0	0	0	0	0	0	0	0
-0.75	-1	-1	-1	0,-1	0,-1	0,-1	0,-1	0,-1	0,-1	0,-1	0,-1	0,-1	0,-1	0,-1	0,-1	0,-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0,-1	0,-1	0,-1	0,-1	0,-1	0,-1
-1.25	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0,-1
-1.5	-1,-2	-1/-2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1.75	-2	-1/-2	-1,-2	-1,-2	-1,-2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-2	-2	-2	-2	2	-2	-1,-2	-1,-2	-1,-2	-1	-1	-1	-1	-1	-1	-1	-1
-2.25	-2	-2	-2	2	-2	-2	-1,-2	-1,-2	-1,-2	-1,-2	-1,-2	-1	-1	-1	-1	-1
-2.5	-2	-2	-2	2	-2	-2	-2	-2	-2	-1,-2	-1,-2	-1,-2	-1,-2	-1	-1	-1
-2.75		-2	-2	2	-2	-2	-2	-2	-2	-2	-2	-1,-2	-1,-2	-1,-2	-1,-2	-1,-2
-3				2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-1,-2	-1,-2	-1,-2
-3.25					-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2
-3.5							-2	-2	-2	-2	-2	-2	-2	-2	-2	-2
-3.75								-2	-2	-2	-2	-2	-2	-2	-2	-2

Fig. 5. Quotient selection logic.

to half of the original size, i.e., to 256 entries and 3 bits wide for the total of 256×3.

The "quotient fetch" operation pertaining to the minimized shared ROM architecture is presented below

$$\text{round}\left(R_R\left[j\right]\right) = r_R^2 r_R^1 r_R^0 . r_R^{-1} r_R^{-2} \tag{4}$$

$$\text{round}\left(R_I\left[j\right]\right) = r_I^2 r_I^1 r_I^0 . r_I^{-1} r_I^{-2} \tag{5}$$

$$addr_R = \begin{cases} r_R^1 r_R^0 . r_R^{-1} r_R^{-2} | d^0 d^{-1} d^{-2} d^{-3} & \text{if } r_R^2 = 0 \\ \text{not}\left(r_R^1 r_R^0 . r_R^{-1} r_R^{-2}\right) + 1 | d^0 d^{-1} d^{-2} d^{-3} & \text{else} \end{cases} \tag{6}$$

$$addr_I = \begin{cases} r_I^1 r_I^0 . r_I^{-1} r_I^{-2} | d^0 d^{-1} d^{-2} d^{-3} & \text{if } r_I^2 = 0 \\ \text{not}\left(r_I^1 r_I^0 . r_I^{-1} r_I^{-2}\right) + 1 | d^0 d^{-1} d^{-2} d^{-3} & \text{else} \end{cases} \tag{7}$$

$$q_R\left(j\right) = \begin{cases} q_R^2 q_R^1 q_R^0 & \text{if} \quad r_R^2 = 0 \\ \text{not}\left(q_R^2 q_R^1 q_R^0\right) + 1 & \text{else} \end{cases} \tag{8}$$

$$q_I\left(j\right) = \begin{cases} q_I^2 q_I^1 q_I^0 & \text{if} \quad r_I^2 = 0 \\ \text{not}\left(q_I^2 q_I^1 q_I^0\right) + 1 & \text{else.} \end{cases} \tag{9}$$

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 2s complement. Similarly, in (8) and (9), the same bit of R $(U + V)$ is used to determine the output $q\left[j\right]$ or $-q\left[j\right]$.

IV. Proposed Error Detection Schemes

Two important methods for error detection are time and hardware redundancy schemes. General hardware redundancy, though simple, is 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. The total number of addresses for the ROM is 512 with each location containing 3 quotient bits. Due to the symmetrical nature of the ROM, as explained in the previous section, the total address locations can be reduced to half, i.e., 256 address locations with each location containing 3 quotient bits. Thus, the total size of the reduced ROM is 256 (address) $\times$ 3 (quotient). 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 $(256 \times 3) \times 2$. Other arithmetic blocks in the design are also duplicated and checked in real time.

Another potential method for fault diagnosis involves using multiplication to undo the division; the result is then compared with the dividend. This method serves as a motivation to our proposed approaches. This technique was implemented for three different register types 16, 32, and 64 bits and the results were drawn from it. Let us consider the first derivation on the default register configuration, i.e., quotient represented using 16 bits. Initially, it was found that the result from the product of the quotient and the divisor was approximately equal to the original result. To overcome this issue, the result of the multiplication is compared with dividend within a set threshold. In the aforementioned design, the result of the multiplication yields 24 bits (16-bit quotient and 8-bit divisor). The threshold for this case is set by discarding 16 sub LSB bits before comparison because the dividend is limited to 8 bits. Discarding 16 bits (LSB) sets the required thresholds for this operation. This restricts the detection range to be within a 8-bit number (any error in the quotient which produces a product beyond that will remain undetected). Although applicable, this approach leads to an increase in undetected errors due to the set threshold (the error coverage is roughly 40% for multiple faults). Realistically, one cannot limit the derivations just within the threshold range (and get high error coverage as a

result) and thus a relatively low error coverage is obtained. The area overhead in this case is roughly 31% based on our FPGA implementations. The major drawback though is that such method has very low and nearly unusable output precision. Using a higher input number will essentially solve this but it will definitely increase the hardware overhead. Increasing the bit widths (for example $8 \rightarrow 16$) leads to a decreased threshold range (increased detection range), but this will lead to twice the area overhead (total output precision is 32 bits compared to 16).

A variant of the scheme above is to reuse the multipliers in the original architecture to perform the required 16×8 multiplication to undo the division with three added cycles. In this case, the 16 bits are first split into the first 8 bits (MSBs) and the second 8 bits (LSBs) and stored in different registers and the LSBs are first multiplied with the 8 bits (one cycle). In the second cycle, MSBs are multiplied and finally, in the third cycle, the results are added and stored. The entire operation requires two 8-bit registers and two 24-bit registers (with proper adjustment) and the result is stored in an 8-bit register by discarding the bits. Our FPGA implementations show that the area overhead is much lower compared to the scheme above (roughly 2%); however, the throughput degradation is approximately 43% which is relatively high for high-speed applications. As mentioned above, the major drawback here is that such method has very low and nearly unusable output precision and the error coverage is low as well (less than 40%). One remedy is to use higher bit widths which leads to roughly 100% area overhead (if we use the same technique and normalize the initial operands, the operating ranges are bigger than the 8-bit case, thus we have an advantage of a higher operating range but at the expense of very high area overhead).

Another 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, 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., $((a+jb)\times 2)/((c+jd)\times 2) = Q_{\text{real}} + jQ_{\text{imag}} = (a+jb)/(c+jd)$. 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.

A. 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 [42], [43]. The error detection division architecture is divided into five modules; each individual module has its own parity or hardware redundancy schemes as described in the following. Fig. 6 shows the high level fault detection design of the unified scheme and hardware redundancy with shaded error detection modules.

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 [44]. 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. 3 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_{imag} as shown in Fig. 3. The checker verifies the following formulas which are obtained by rearranging (3):

$$x_{\text{real}} + x_{\text{imag}} = t_1 - 2 \times b \times d \tag{10}$$

$$x_{\text{real}} + x_{\text{imag}} + 2 \times b \times d = t_1. \tag{11}$$

2) Datapath Registers: The registers in the datapath are 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 raises an error flag in case of a detection (mismatch). The logic relation of a parity bit for a register of size m is shown in the following equation and Fig. 7 shows registers incorporated with parity bits and the logic relation which calculates the parity of each register. Each register is equipped with a parity bit and the parity bits are calculated when the value is clocked in the register. The error indication flag at the output indicates the faults in the register. These flags are tied to an OR gate and this allows the system to trigger an error in the case of a single register fault

$$R = R^1 R^2 R^3 \cdots R^n$$

$$P = R_1 \oplus R_2 \oplus R_3 \oplus \cdots \oplus R_n. \tag{12}$$

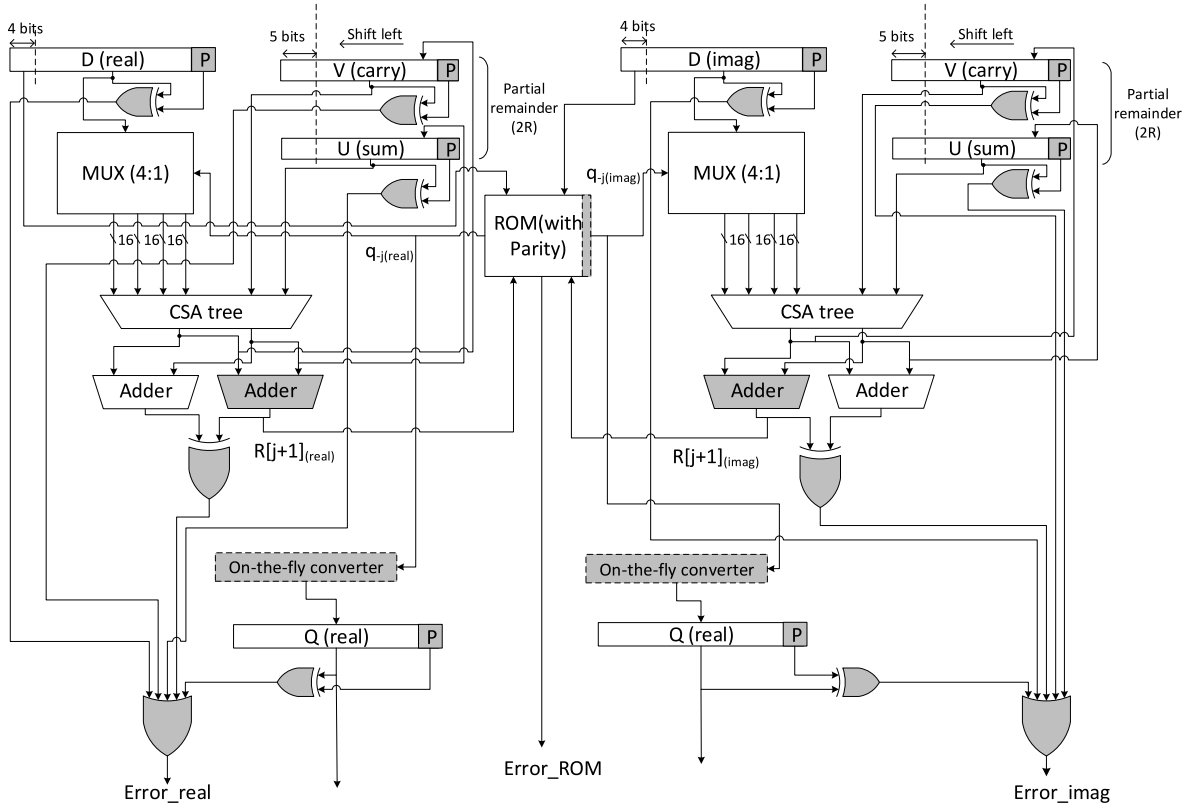

Fig. 6. Unified parity check and hardware redundancy.

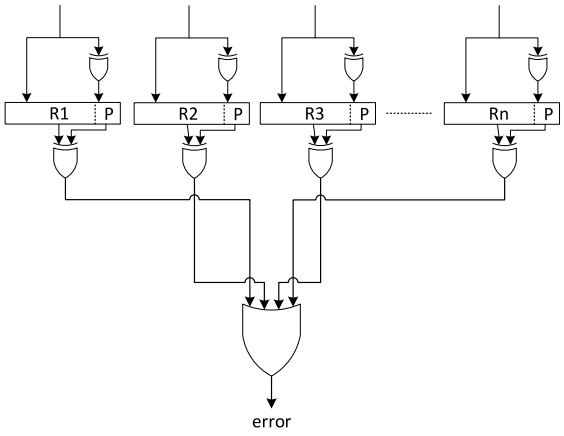

Fig. 7. Registers incorporated with parity bits.

3) ROM (Quotient Selection Logic): The memory module used in the design is critical to the operation and, thus, if it is prone to faults, it can undermine the entire objective. 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 parity prediction logic is depicted in Fig. 8.

4) CSA: The proposed design uses a CSA to compute the next partial remainder based on the previous partial remainder

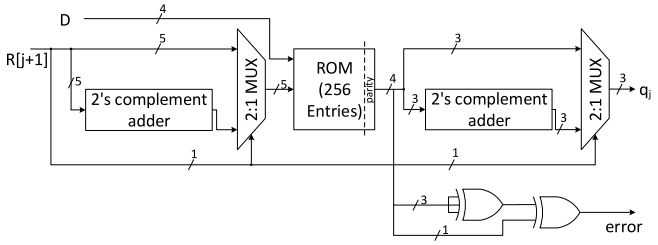

Fig. 8. Minimized ROM with the proposed parity prediction logic.

$R[j]$ and the product of the quotient and divisor, i.e., $q[j] \times D$. Since the design is aimed toward 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 [45]. The parity check equations used in the design are given below

$$\text{Sum}_e = a \oplus b \oplus cin \oplus \text{Sum} \tag{13}$$

$$C_e = (a \bullet b) \oplus (a \oplus b) \bullet (cin \oplus cout). \tag{14}$$

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.

B. Error Detection Through RESO

In this error detection model, we use RESO [36] to detect faults in the design. The RESO method, as explained before,

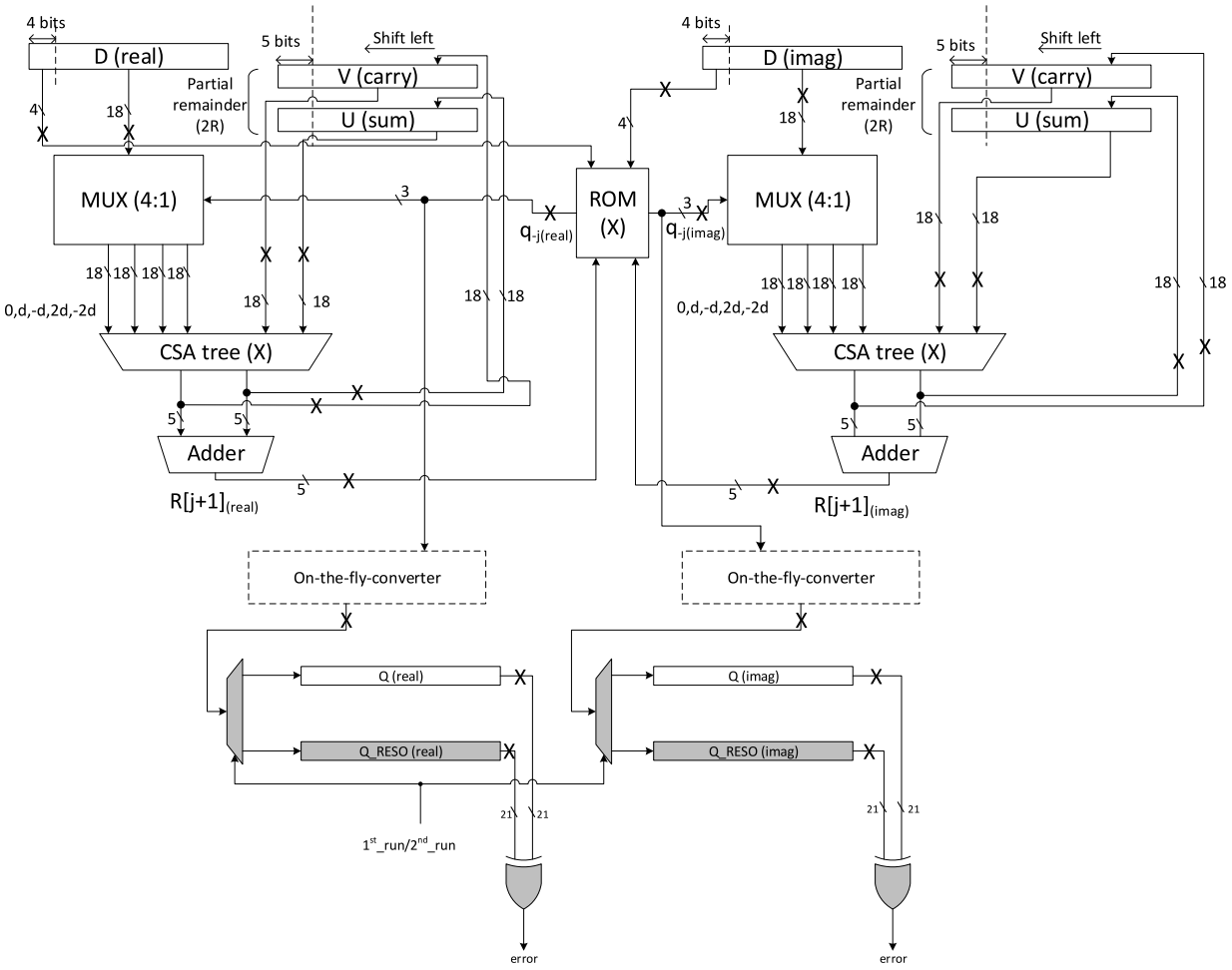

Fig. 9. Proposed divider with RESO for error detection.

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. 9 with the error detection modules shaded and the location of injections shown by crosses. The figure shows a high level circuit model and not an exact replica of the implemented FPGA design, for such modules error injections are indicated by crosses in parentheses (error injection techniques are described, in details, in Section VI). The error detection operation in RESO is illustrated using a simple register in Fig. 10. Let R be a register containing a permanent stuck-at-one fault at bit 3 (shaded bit). Assume the register is supposed to hold the value 000000111_2 (7_{10}) in the original run and the stuck-at fault changes the value in the register to 000001111_2 (15_{10}). During the second run, the value in the register is shifted one bit to the left which leads to 000011110_2 (30_{10}) instead of 000001110_2 (14_{10}). Assuming the result of the simple operation F on x' can be obtained by a decoding operation G on

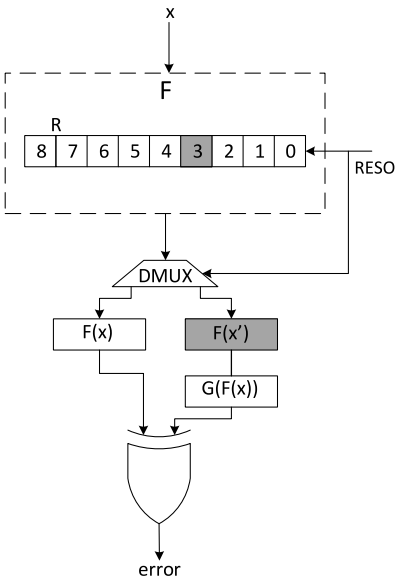

Fig. 10. RESO operation illustrative example.

$F(x')$, i.e., $G(F(x')) = F(x)$ (under the error free condition). In the case of an error bit, $F(x') \neq F(x)$ because the operation G cannot revert $F(x')$ to $F(x)$.

As shown in Fig. 9, the sizes 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.

Suppose one pipeline-register has been placed to sub-pipeline the structures. The location for placing the registers is chosen to break the timing path in to approximately equal halves. Let us denote the two halves of pipelined stages by Π_1 and Π_2. The original input is first applied to the architecture in the first cycle. In the second cycle, while the second half of the circuit (Π_2) executes this first input, the rotated variant of the first input is fed to the first half of the circuit (Π_1). This process is consecutively executed until the last rotated input is derived. We note that for detecting the errors, the outputs of the runs with the rotated-inputs are rotated back and compared against the original inputs. Therefore, any mismatch indicates an error.

To finalize this section, we present a variant of the RESO approach (a small tweak to RESO). In this technique, the bits are right shifted and the last LSB is discarded instead of left shifting and saving the MSB. This performs a divide by 2 on the divisor and dividend instead of a multiplication. Similar to the previous RESO method, no additional hardware is required to decode the second run result (due to cancelation). This enables us to incorporate RESO without any change to the existing design and including additional hardware (adding extra bit to entire design) as before. The problem with this technique is that it does not provide complete 8-bit number coverage, e.g., if only the LSB is 1 then the right shift leads to a zero (discarded last bit). Noting the small area savings for this approach, its application could be in the usage models very sensitive to area overheads of fault detection schemes.

V. 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 integrated synthesis environment version 14.5 and synthesized for Spartan-6 xc6slx16-2cgs324 and Virtex-6 xc6vlx75t-3ff484 devices [46]. Through this analysis, the performance and implementation metrics of the complex divider for both low-end and high-end FPGAs can be observed with very-high-speed-integrated-circuit hardware description language (VHDL) as the design entry.

TABLE I
FPGA Implementation Results on Spartan-6
FPGA Device xc6s1x16

Arch.	Area [Slices]	Throughput [Gbps]	Power $[mW]$
Original	228	0.559	5.43
Scheme-I	304 (33.3%)	0.413 (26.1%)	7.12 (31.1%)
Scheme-II	269 (17.9%)	0.484 (13.4%)	5.94 (9.3%)

TABLE II
FPGA Implementation Results on Virtex-6
FPGA Device xc6vlx75t

Arch.	Area [Slices]	Throughput [Gbps]	Power $[mW]$
Original	248	1.074	7.04
Scheme-I	346 (39.5%)	0.829 (22.8%)	8.90 (26.4%)
Scheme-II	283 (14.1%)	0.971 (9.5%)	7.93 (12.6%)

The complex division architecture is structured hierarchically. Specifically, it is divided into four functional modules: 1) Golub's multiplier; 2) normalizer; 3) SRT divider (real and imaginary); and 4) 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 (minimum period of 3.724 ns) for the Virtex-6 and 140.928 MHz (minimum period of 7.095 ns) 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. Based on our FPGA implementations, the delay overheads for Scheme-I (unified parity and hardware redundancy design) are 29.5% (Virtex-6) and 36.4% (Spartan-6), this is due to storing the result in carry save form rather than using traditional CPAs, 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. 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 families.

The implementation results are tabulated in Tables I and II 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 overheads are 33.3% (Spartan-6) and 39.5% (Virtex-6) which correspond to Scheme-I and 17.9% (Spartan-6) and 14.1% (Virtex-6) for Scheme-II, respectively. We note that for Scheme-II, sub-pipelining the design leads to higher frequencies at the expense of area. Since Scheme-II is aimed toward area-efficient designs compared to Scheme-I,

further sub-pipelining has not been implemented to maintain the area within acceptable limits.

We finalize this section by presenting the performance and implementation metrics of the RESO technique in which one-bit shift to the right is performed. In this scheme, the delay overhead is very close to that of the RESO method and the areas are 267 slices (down from 269 slices in the original RESO scheme) and 263 slices (down from 283 slices in the original RESO scheme), and the power consumptions are 5.82 mW (down from 5.94 mW in the original RESO scheme) and 7.80 mW (down from 7.93 mW in the original RESO scheme) for Spartan-6 and Virtex-6, respectively. We note that the small decrease of the overheads observed is at the expense of degraded fault detection rate. Based on the objectives and overhead tolerance, one can choose the scheme suitable for specific applications.

VI. Fault Injection Simulations

For the fault model in this paper, both single and multiple stuck-at-fault are considered to cover natural failures and counteract VLSI defects [47]. The applied fault model uses linear-feedback shift registers (LFSRs) to generate fault patterns. The outputs of the LFSRs are ORed or ANDed with outputs of selective locations to emulate transient faults, both stuck-at-one and stuck-at-zero faults. Due to hardware limitations in the FPGAs, the LFSRs used are clocked to the same speeds as the implemented designs. The error injection system can also be configured to inject faults once in 8, 4, and 2 clock cycles which provides different test scenarios. The different LFSRs used in the test system are as follows: 1) 16-bit LFSRs for the registers; 2) 5-bit LFSRs for the adders; and 3) 3-bit LFSRs for the ROM. The initial seeds to each of the LFSRs are different in order to produce a pseudo-random scenario. The LFSR block is made generic (this allows easy port map in VHDL). The values of the initial seeds are provided during the beginning of the simulation. Permanent faults are modeled by switching the LFSR to a constant pattern per division. The faults are injected as follows: for the first case (per eight clock cycles), one fault is injected into the 13 data path registers, the ROM, and the 5-bit adders. This provides a scenario where one fault (single or multiple) is injected to all the previously-mentioned locations, for the duration of the division process. Similarly, the second case injects two faults per division process and the third case injects four faults per division operation. For the combinational circuits, the fault model is directly ANDed or ORed with the datapath within the Golub's multiplier and the adders. The second case and the third case were primarily used for fault injection, because they provide a more uniform distribution and allow us to test for a greater faults/clock cycles. Double and multiple faults are tested separately; this is done to test the parity bits extensively. Double faults are injected using a different model; this is achieved by ORing two 16-bit registers with one bit set in both registers (not in the same bit location), different combinations are achieved by shifting the bits in the register. In addition, the fault testing for double and multiple faults are performed for two different scenarios, faults injected throughout the design and faults injected only in the parity equipped registers. The reason for this modified test scenario is to analyze the parity registers, because including the Golub's multiplier results in a very high detection rate due to its error detection scheme. 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 do not affect a particular part of the circuit but the entire circuit as a whole with a uniform distribution. Moreover, the aliasing cases are excluded from the error analysis, i.e., the cases where the injected faults 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 $1/2^{16}$ and the error stays for exactly one clock cycle. This perfectly simulates transient faults. The effect of permanent faults can be observed by switching the locations probed using an LFSR to a constant value.

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 sketch of such a proof is derived noting that the results of the formulas for the parity predictions of the blocks in the architecture will be different from the actual parities for any odd number of errors due to the characteristics of the XOR operations and these are super-sets of single errors which proves such an error coverage. The simulations for single stuck-at-fault are performed exhaustively in every bit 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 high error coverage close to 100% (there is a slight chance of having identical undetected faults). 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 1000 random inputs and 917 504 faults combining the faults injected at both registers and combinational circuits. The results of the simulations show almost full error coverage (note that this is for the case in which randomly-distributed multiple errors occur at both registers and combinational circuits).

The proposed error detection designs in Figs. 6 and 9 indicate a high level design. The error injection locations in both the designs common are indicated by a X in Fig. 9. Single stuck-at faults are exhaustively checked at all the modules and bit locations, i.e., for a 16-bit register, it is checked at 16 bits. Multiple faults are injected using LFSRs at the outputs of the Golub's multiplier, CSA (including four registers and a combinational circuit), datapath registers, and the shared ROM. In order to provide another scenario, we have disabled the fault detection in the aforementioned modules to test the coverage in the parity registers (such errors are caused by different means such as radiations). Scheme-I with only the parity registers enabled provides coverage of 100% for single faults. It has lower coverage for multiple errors, and it is known that double faults are not detected using single parities. A single parity module is capable of detecting 50% of the errors (for only the logic whose predicted parity is derived and not for the previous logic levels). At a given point in time, due to the pseudo-random nature of the LFSR (commonly-used to model natural faults) and considering stuck-at-one or zero, the output parity of a register may equate to an odd parity or an even parity with a probability of half. If we have a number of cycles, the detection probability is increased though, leading to better error coverage. We would like to emphasize that if only the detection of even number of faults is of concern, to get a very high error coverage, RESO method or its variant can be used. Scheme-II with RESO provides an error coverage of 100%. The error coverage for the right-shift variant of RESO is 100% but does not provide full coverage in the cases discussed in the previous section.

VII. Conclusion

In this paper, 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

[1] M. C. M. Teixeria, E. Assuncao, and E. R. M. D. Machado, "A method for plotting the complementary root locus using the root-locus (positive gain) rules," *IEEE Trans. Educ.*, vol. 47, no. 3, pp. 405–409, Aug. 2004.

[2] P. J. S. G. Ferreira, "Concerning the Nyquist plots of rational functions of nonzero type," *IEEE Trans. Educ.*, vol. 42, no. 3, pp. 228–229, Aug. 1999.

[3] A. C. Bartlett, "Nyquist, Bode, and Nichols plots of uncertain systems," in *Proc. Conf. Amer. Control*, San Diego, CA, USA, 1990, pp. 2033–2037.

[4] S. Mijalkovic, "Using frequency response coherent structures for model-order reduction in microwave applications," *IEEE Trans. Microw. Theory Techn.*, vol. 52, no. 9, pp. 2292–2297, Sep. 2004.

[5] (2011). *Exocortex.DSP*. [Online]. Available: http://www.exocortex.org/dsp/

[6] (2011). *CoStLy: Complex Standard Functions Library*. [Online]. Available: http://iamlasun8.mathematik.unikarlsruhe.de/ae16/CoStLy.html/

[7] R. L. Smith, "Algorithm 116: Complex division," *Commun. ACM*, vol. 5, no. 8, p. 435, 1962.

[8] G. W. Stewart, "A note on complex division," *ACM Trans. Math. Softw.*, vol. 11, no. 3, pp. 238–241, 1985.

[9] C. P. Jeannerod, N. Louvet, and J. M. Muller, "On the component-wise accuracy of complex floating-point division with an FMA," in *Proc. IEEE Symp. Comput. Arith.*, Washington, DC, USA, 2013, pp. 83–90.

[10] M. D. Ercegovac and J. M. Muller, "Complex division with prescaling of operands," in *Proc. IEEE Int. Conf. Appl.-Specific Syst. Archit. Processors*, Hague, The Netherlands, 2003, pp. 304–314.

[11] P. Dormiani, M. D. Ercegovac, and J. M. Muller, "Design and implementation of a radix-4 complex division unit with prescaling," in *Proc. IEEE Int. Conf. Appl.-Specific Syst. Archit. Processors*, Boston, MA, USA, 2009, pp. 83–90.

[12] D. Wang, M. D. Ercegovac, and N. Zheng, "A radix-8 complex divider for FPGA implementation," in *Proc. IEEE Int. Conf. Field Program. Logic Appl.*, Prague, Czech Republic, 2009, pp. 236–241.

[13] D. Wang and M. D. Ercegovac, "A radix-16 combined complex division/square root unit with operand prescaling," *IEEE Trans. Comput.*, vol. 61, no. 9, pp. 1243–1255, Sep. 2012.

[14] F. Edman and V. Oewall, "Fixed-point implementation of a robust complex valued divider architecture," in *Proc. Eur. Conf. Circuit Theory Design*, vol. 1. Cork, Ireland, 2005, pp. 143–146.

[15] J. Liu, B. Weaver, and Y. Zakharov, "FPGA implementation of multiplication-free complex division," *Electron. Lett.*, vol. 44, no. 2, pp. 95–96, Jan. 2008.

[16] T. Jamil, "An introduction to complex binary number system," in *Proc. Int. Conf. Inf. Comput.*, Phuket, Thailand, 2011, pp. 229–232.

[17] T. Jamil and S. S. Al-Abri, "Design of a divider circuit for complex binary numbers," in *Proc. World Congr. Eng. Comput. Sci.*, vol. 2. San Francisco, CA, USA, 2010, pp. 1–6.

[18] J. Liu, "DCD algorithm: Architectures, FPGA implementations and applications," Ph.D. dissertation, Dept. Electron., Univ. York, Heslington, U.K., 2008.

[19] A. Reddy and P. Banarjee, "Algorithm-based fault detection for signal processing applications," *IEEE Trans. Comput.*, vol. 39, no. 10, pp. 1304–1308, Oct. 1990.

[20] B. Shim and N. R. Shanbhag, "Energy-efficient soft error-tolerant digital signal processing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 4, pp. 336–348, Apr. 2006.

[21] C. H. Yen and B. F. Wu, "Simple error detection methods for hardware implementation of advanced encryption standard," *IEEE Trans. Comput.*, vol. 55, no. 6, pp. 720–731, Jun. 2006.

[22] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "Concurrent structure-independent fault detection schemes for the advanced encryption standard," *IEEE Trans. Comput.*, vol. 59, no. 5, pp. 608–622, May 2010.

[23] T. G. Malkin, F. X. Standaert, and M. Yung, "A comparative cost/security analysis of fault attack countermeasures," in *Proc. Int. Workshop Fault Diagn. Tolerance Cryptography*, Yokohama, Japan, 2006, pp. 159–172.

[24] M. Mozaffari-Kermani and R. Azarderakhsh, "Efficient fault diagnosis schemes for reliable lightweight cryptographic ISO/IEC standard CLEFIA benchmarked on ASIC and FPGA," *IEEE Trans. Ind. Electron.*, vol. 60, no. 12, pp. 5925–5932, Dec. 2013.

[25] M. Mozaffari-Kermani, R. Azarderakhsh, and A. Aghaie, "Reliable and error detection architectures of Pomaranch for false-alarm-sensitive cryptographic applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, to be published.

[26] G. Di Natale, M. Doulcier, M. L. Flottes, and B. Rouzeyre, "A reliable architecture for parallel implementations of the advanced encryption standard," *J. Electron. Test. Theory Appl.*, vol. 25, no. 4, pp. 269–278, Aug. 2009.
[27] M. Mozaffari-Kermani, K. Tian, R. Azarderakhsh, and S. Bayat-Sarmadi, "Fault-resilient lightweight cryptographic block ciphers for secure embedded systems," *IEEE Embedded Syst. Lett.*, vol. 6, no. 4, pp. 89–92, Dec. 2014.
[28] S. Bayat-Sarmadi, M. Mozaffari-Kermani, and A. Reyhani-Masoleh, "Efficient and concurrent reliable realization of the secure cryptographic SHA-3 algorithm," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 33, no. 7, pp. 1105–1109, Jul. 2014.
[29] M. Mozaffari-Kermani, R. Azarderakhsh, C. Lee, and S. Bayat-Sarmadi, "Reliable concurrent error detection architectures for extended Euclidean-based division over $GF(2^m)$," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 5, pp. 995–1003, May 2014.
[30] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "A lightweight high-performance fault detection scheme for the advanced encryption standard using composite fields," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 1, pp. 85–91, Jan. 2011.
[31] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "Parity-based fault detection architecture of S-box for advanced encryption standard," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Syst. (DFT)*, Arlington, VA, USA, 2006, pp. 572–580.
[32] P. Maistri and R. Leveugle, "Double-data-rate computation as a countermeasure against fault analysis," *IEEE Trans. Comput.*, vol. 57, no. 11, pp. 1528–1539, Nov. 2008.
[33] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri, "A parity code based fault detection for an implementation of the advanced encryption standard," in *Proc. Int. Symp. Defect Fault Tolerance VLSI Syst.*, Vancouver, BC, Canada, 2002, pp. 51–59.
[34] A. Hariri and A. Reyhani-Masoleh, "Concurrent error detection in Montgomery multiplication over binary extension fields," *IEEE Trans. Comput.*, vol. 60, no. 9, pp. 1341–1353, Sep. 2011.
[35] A. Reyhani-Masoleh and M. A. Hasan, "Fault detection architectures for field multiplication using polynomial bases," *IEEE Trans. Comput.*, vol. 55, no. 9, pp. 1089–1103, Sep. 2006.
[36] J. H. Patel and L. Y. Fung, "Concurrent error detection in ALUs by recomputing with shifted operands," *IEEE Trans. Comput.*, vol. C-31, no. 7, pp. 589–595, Jul. 1982.
[37] B. W. Johnson, J. H. Aylor, and H. H. Hana, "Efficient use of time and hardware redundancy for concurrent error detection in a 32-bit VLSI adder," *IEEE J. Solid State Circuits*, vol. 23, no. 1, pp. 208–215, Feb. 1988.
[38] B. Parhami, "High-radix dividers," in *Computer Arithmetic Algorithms and Hardware Designs*. New York, NY, USA: Oxford Univ. Press, 2000, pp. 228–245.
[39] M. D. Ercegovac and T. Lang, "Division by digit recurrence," in *Digital Arithmetic*, D. E. M. Penrose, Ed. San Francisco, CA, USA: Morgan Kaufmann, 2004, pp. 247–319.
[40] D. L. Harris, S. F. Oberman, and M. A. Horowitz, "SRT division architectures and implementations," in *Proc. IEEE Symp. Comput. Arith.*, Asilomar, CA, USA, 1997, pp. 18–25.
[41] J. W. Hartwell, "A procedure for implementing the fast Fourier transform on small computers," *IBM J. Res. Dev.*, vol. 15, no. 5, pp. 355–363, Sep. 1971.
[42] M. Nicolaidis, R. O. Duarte, S. Manich, J. Figueras, "Fault-secure parity prediction arithmetic operators," *IEEE Design Test*, vol. 14, no. 2, pp. 60–71, Apr. 1997.
[43] M. Nicolaidis, "Carry checking/parity prediction adders and ALUs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 1, pp. 121–128, Feb. 2013.
[44] S. Pontarelli, P. Reviriego, C. J. Bleakley, and J. A. Maestro, "Low complexity concurrent error detection for complex multiplication," *IEEE Trans. Comput.*, vol. 62, no. 9, pp. 1899–1903, Sep. 2013.
[45] S. H. Mozafari, M. Fazeli, S. Hessabi, and S. G. Miremadi, "A low cost circuit level fault detection technique to full adder design," in *Proc. IEEE Int. Conf. Electron. Circuits Syst. (ICECS)*, Beirut, Lebanon, 2011, pp. 446–450.
[46] (Jan. 26, 2015). *Xilinx*. [Online]. Available: http://www.xilinx.com
[47] L. Breveglieri, I. Koren, and P. Maistri, "An operation-centered approach to fault detection in symmetric cryptography ciphers," *IEEE Trans. Comput.*, vol. 56, no. 5, pp. 635–649, May 2007.

Mehran Mozaffari Kermani (M'11) received the B.Sc. degree in electrical and computer engineering from the University of Tehran, Tehran, Iran, in 2005, and the M.E.Sc. and Ph.D. degrees from the Department of Electrical and Computer Engineering, University of Western Ontario, London, ON, Canada, in 2007 and 2011, respectively.

He joined Advanced Micro Devices is in Markham, ON, Canada, as a Senior ASIC/Layout Designer, integrating sophisticated security/cryptographic capabilities into a single accelerated processing unit. In 2012, he joined the Department of Electrical Engineering, Princeton University, Princeton, NJ, USA, as a Natural Sciences and Engineering Research Council of Canada Post-Doctoral Research Fellow. He is currently with the Department of Electrical and Microelectronic Engineering, Rochester Institute of Technology, Rochester, NY, USA. His current research interests include emerging security/privacy measures for deeply embedded systems, cryptographic hardware systems, fault diagnosis and tolerance in cryptographic hardware, VLSI reliability, and low-power secure and efficient FPGA and application specified integrated circuit designs.

Dr. Mozaffari-Kermani was the recipient of the prestigious Natural Sciences and Engineering Research Council of Canada Post-Doctoral Research Fellowship in 2011 and the Texas Instruments Faculty Award (Douglas Harvey) in 2014. He is currently serving as a Guest Editor of the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING for the Special Issue of Emerging Security Trends for Deeply-Embedded Computing Systems 2014 and 2015 and a Technical Committee Member for a number of related conferences including International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, Fault Diagnosis and Tolerance in Cryptography Workshop, Radio Frequency Identification Security Workshop, International Workshop on Lightweight Cryptography for Security and Privacy, and International Workshop on the Arithmetic of Finite Fields.

Niranjan Manoharan received the B.Tech. degree in electronics and communication engineering from the Amrita School of Engineering, Coimbatore, India, in 2012, and the master's degree in electrical engineering at the Rochester Institute of Technology, Rochester, NY, USA, in 2014, under the supervision of Prof. M. Mozaffari-Kermani and co-supervision of Prof. R. Azarderakhsh.

He is currently with Analog Devices Inc., Norwood, MA, USA. His current research interests include computer arithmetic and digital signal processor for reconfigurable devices and application specified integrated circuits as well as fault tolerant computing.

Reza Azarderakhsh (M'12) received the B.Sc. degree in electrical and electronic engineering and the M.Sc. degree in computer engineering from the Sharif University of Technology, Tehran, Iran, in 2002 and 2005, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Western Ontario, London, ON, Canada, in 2011.

He joined the Department of Electrical and Computer Engineering, University of Western Ontario, as a Limited Duties Instructor, in 2011. He was a Natural Sciences and Engineering Research Council of Canada (NSERC) Post-Doctoral Research Fellow with the Center for Applied Cryptographic Research and the Department of Combinatorics and Optimization, University of Waterloo, Waterloo, ON, Canada. He is currently with the Department of Computer Engineering, Rochester Institute of Technology, Rochester, NY, USA. His current research interests include finite field and its application, elliptic curve cryptography, and pairing-based cryptography.

Prof. Azarderakhsh was the recipient of the prestigious NSERC Post-Doctoral Research Fellowship in 2012.