International Journal of Current Research and Modern Education (IJCRME)
ISSN (Online): 2455 – 5428 & Impact Factor: 3.165
Special Issue, NCFTCCPS - 2016

IMPLEMENTATION OF COMPLEX DIVIDER WITH ERROR COMPUTATION

S. Preetha* & S. Jeya Anusha**
Department of Electronics and Communication Engineering, TJS Engineering College, Chennai, Tamilnadu

Abstract:

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. Proposed schemes, extensive error detection assessments are performed for the proposed designs through fault Complex division is commonly used in various applications in signal processing and control theory including astronomy and nonlinear RF measurements. 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 simulations and field programmable gate array (FPGA) implementations; the design is implemented on Xilinx Spartan-6 and Xilinx Virtex-6 FPGA families.

Key Words: FPGA, Error Detector & Complex Division

1. 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, Nyquist plot, and Nichols plot. Microwave systems also use complex division arithmetic to find the frequency response and transfer functions. Because of its complexity, the operation has been mainly implemented in software. This has been further improved by using different algorithms to prevent overflows and provide precise results. Other optimizations 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, this approach is based on operand pre scaling and digit recurrence. Such an algorithm has later been implemented on FPGAs with different radices. Furthermore, a radix-16 combined complex divider/square root module has also been presented in based on the same algorithm. Moreover, a complex divider has been presented in 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. There exist other complex division techniques which are based on complex binary number system (CBNS).

A. 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.

B. 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 test function very high error coverage.

2. 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 adds 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 2k-bits. Whereas, the quotient of a 2k-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 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 $r0$ is set to $r0 = z$, where z is the dividend. In *ith* sequence, the quotient digit qi is selec ted based on the partial remainder $ri$and divisor d. This is followed by the subtraction of $2 \times qi \times d$ (shifted version) from the partial remainder ri. 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 1

$$ri = 2 \times ri - qi \times (2i \times d) \; with \; r0 = z \; and \; ri = 2 \times r \qquad (1)$$

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.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 no redundant digit set 0, 1, 2, ... , $a - 1$, 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-β Dividers or High Radix Dividers: This architecture is the most practical to implement compared to the previous ones. Based on the radix-β, 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-β dividers.

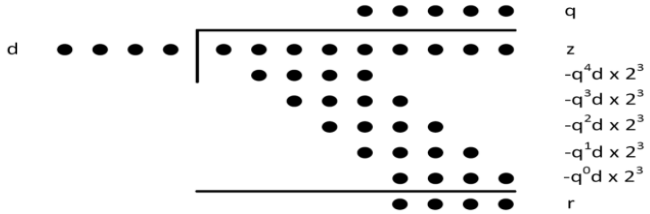

Figure 1: Dot representation of basic hardware division.

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 $log2(\beta)$ quotient bits every iteration. To achieve a 16, 32, or 64 bit precision, we require $16/log2(\beta)$, $32/log2(\beta)$, or $64/log2(\beta)$ iterations, respectively. In order to achieve higher precision with less iteration, 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:

$$R[j + 1] = \beta \times (R[j] - q\,[j] \times D) \qquad (2)$$

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-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 (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 α is in the range $(\beta-1)\ 2 \le \alpha \le (\beta - 1)$. The Robertson diagram in Fig. 4.2 shows the relation between the new shifted partial Remainder and the old partial remainder based on the quotient.

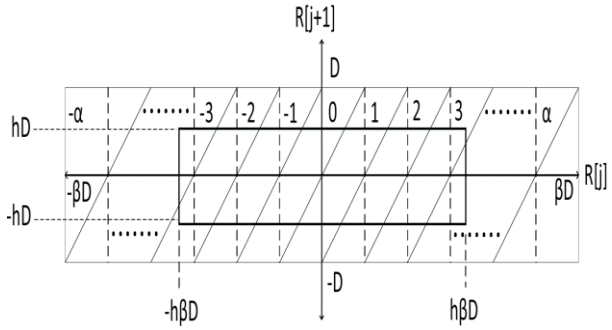

Figure 2: New shifted remainder and old shifted remainder in radix-β (Robertson diagram).

From Fig. 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 $h \leq \alpha (\beta-1)$. 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 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 is the quotient selection regions. The quotient qi is selected based on the position of the uncertainty rectangle, in this case both $qi = \beta$ or $\beta + 1$ will yield the correct result.

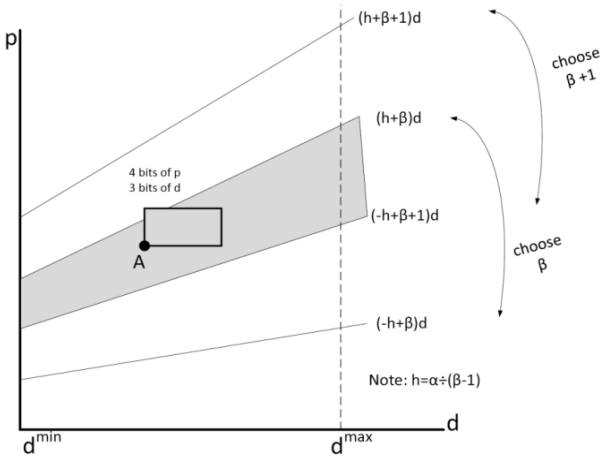

Figure3: p-d plot quotient selection for radix-β division.

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× which leads to additional hardware and delay. The quotient selection logic for the minimally redundant quotient set is shown in the following and detailed in Fig .4. Where the quotient selection scae is illustrated based on:

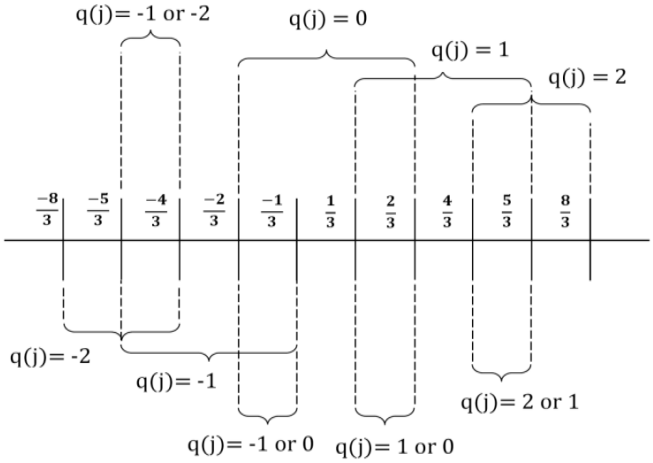

Figure 4: The quotient selection scale

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

B. Golub's Multiplication:

Complex multiplications in DSP systems generally use a more efficient indirect approach, referred to as the Golub's method. For two complex numbers with *a* and *c* as the real parts and *b* and *d* as imaginary parts, one can reach

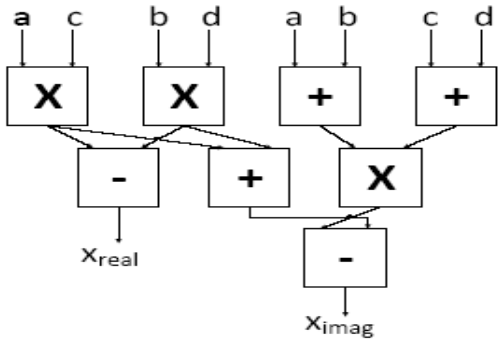

Figure 5: Golub's multiplication

$t1 = (a + b) \times (c + d)$, $t2 = a \times c$, $t3 = b \times d$

$x = (t2 - t3) + j \times (t1 - t2 - t3)$ (4)

$xreal = t2 - t3$

$ximag = t1 - t2 - t3$

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

C. 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 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., $c^2 + d^2$:

$$\frac{a+jb}{c+jd}=\frac{(a+jb)\times(c-jd)}{(c+jd)\times(c-jd)}=\frac{x}{c^2+d^2}+j\frac{y}{c^2+d^2}. \quad (5)$$

The normalize normalizes the numbers such that $1 \le x, y, (c2 + d2) \le 2$. The real and imaginary iteration modules are essentially two radix-4 SRT blocks. As seen in Fig. 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. 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.

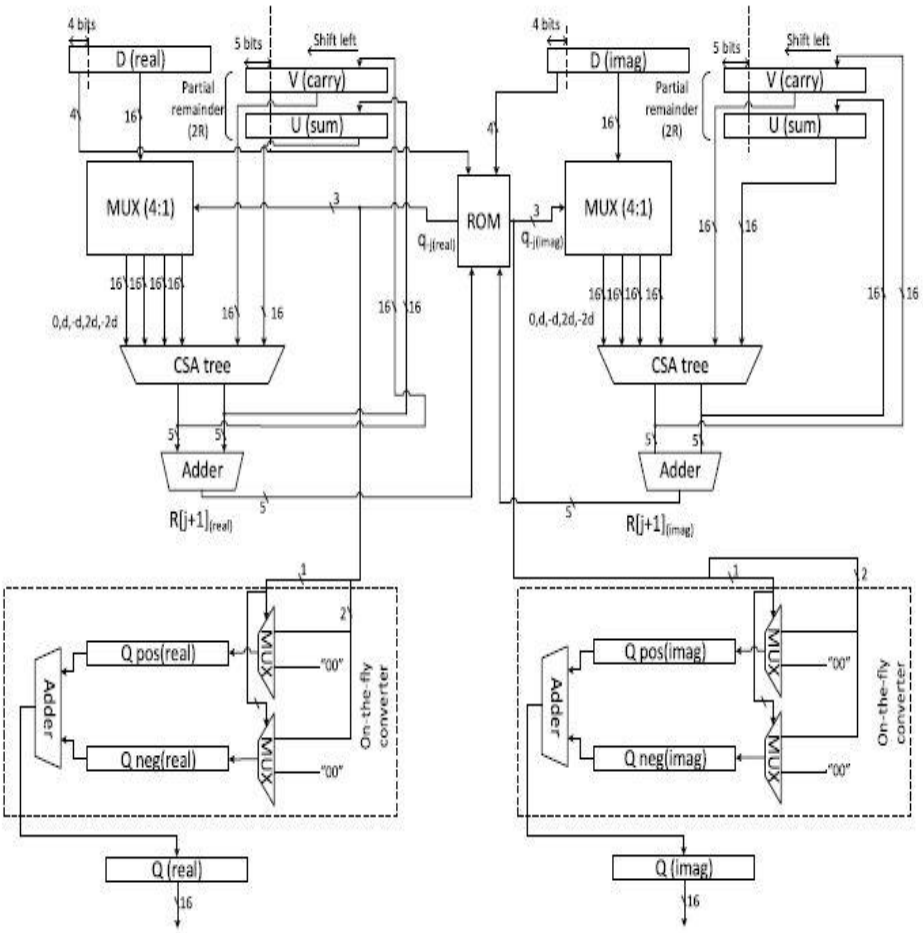

Figure 6: Presented complex radix-4 SRT module.

(initial 5 bits of *U* and *V* and 4 bits of *D*), i.e., 0, *D*, −*D*, 2*D*, and−2*D*. 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 *Qreal* and *Qimag*, respectively. The shared ROM used for quotient fetch is minimized to save area;the minimized ROM structure is explained in the followings.

D. 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 29 = 512 entries as shown in Fig.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:

$$\text{round}\left(R_R[j]\right) = r_R^2 r_R^1 r_R^0 . r_R^{-1} r_R^{-2} \quad (6)$$

$$\text{round}\left(R_I[j]\right) = r_I^2 r_I^1 r_I^0 . r_I^{-1} r_I^{-2} \quad (7)$$

$$addr_R = \begin{cases} r_R^1 r_R^0 . r_R^{-1} r_R^{-2} | d^0 d^{-1} d^{-2} d^{-3} & \text{if } r_R^2 = 0 \\ \text{not}\left(r_R^1 r_R^0 . r_R^{-1} r_R^{-2}\right) + 1 | d^0 d^{-1} d^{-2} d^{-3} & \text{else.} \end{cases} \quad (8)$$

$$addr_I = \begin{cases} r_I^1 r_I^0 . r_I^{-1} r_I^{-2} | d^0 d^{-1} d^{-2} d^{-3} & \text{if } r_I^2 = 0 \\ \text{not}\left(r_I^1 r_I^0 . r_I^{-1} r_I^{-2}\right) + 1 | d^0 d^{-1} d^{-2} d^{-3} & \text{else.} \end{cases} \quad (9)$$

$$q_R(j) = \begin{cases} q_R^2 q_R^1 q_R^0 & \text{if } r_R^2 = 0 \\ \text{not}\left(q_R^2 q_R^1 q_R^0\right) + 1 & \text{else} \end{cases} \quad (10)$$

$$q_I(j) = \begin{cases} q_I^2 q_I^1 q_I^0 & \text{if } r_I^2 = 0 \\ \text{not}\left(q_I^2 q_I^1 q_I^0\right) + 1 & \text{else.} \end{cases} \quad (11)$$

	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

Figure 7: Quotient selection logic.

As shown in (8) and (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 2's complement. Similarly, in (10) and (11), 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.

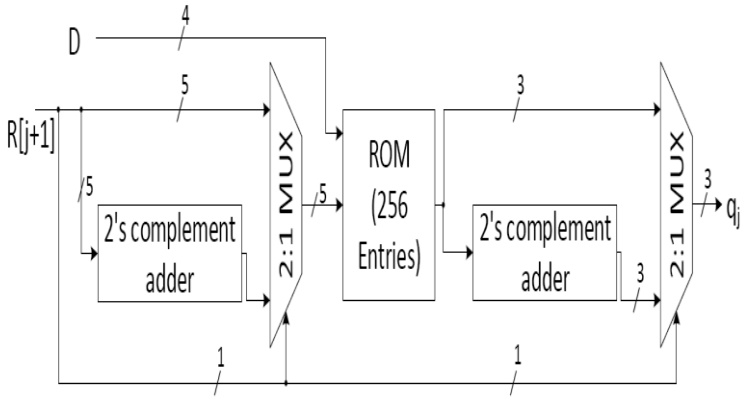

Figure 8: Minimized ROM

3. 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 × 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.

$$(a+jb)\times 2(c+jd)\times 2=X+Y=(a+jb)(c+jd) \quad (12)$$

This requires no additional decoding hardware at the end of the operation to obtain the original result. In order to incorporate all possible $2n$ 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. 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. 10, shows the main CED blocks in the data path of the proposed error detection scheme. The following subsections explain the individual fault technique applied to the module.

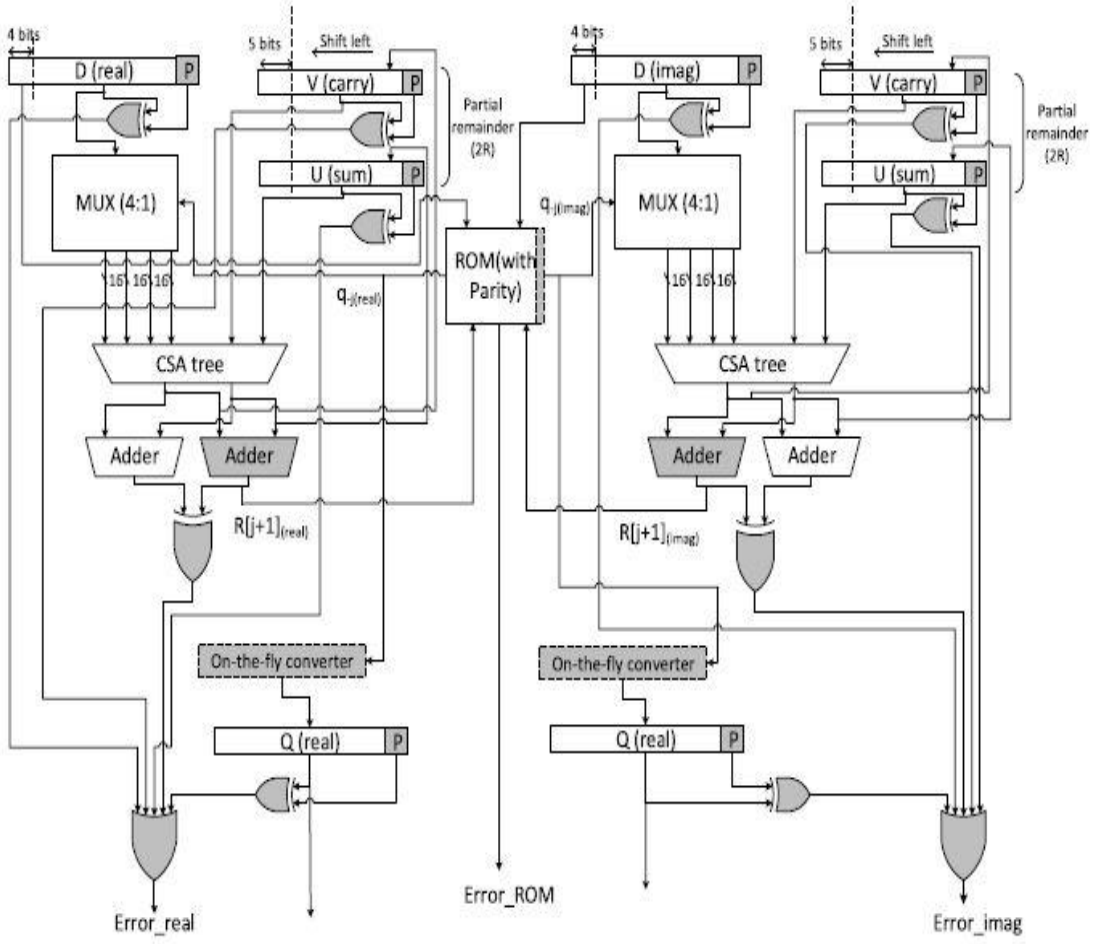

Figure 9: Radix-4 complex SRT module with unified parity check

B. 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. 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 form $a \times c + b \times d$ to *ximag* as shown in Fig. 5. The checker verifies (13), (14) which are obtained by rearranging (3). The structure of the concurrent error detection multiplier is shown in Fig. 10.

$$xreal + ximag = t1 - 2 \times b \times d, \qquad (13)$$

$$xreal + ximag + 2 \times b \times d = t1. \qquad (14)$$

C. Datapath Register:

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. 11, shows a generalized representation of the theory explained above. The bit P is the individual parity bit of registers R1, R2,. . . . . ., Rn. Some cases may also contain multiple parity bits in the same register, to detect errors.

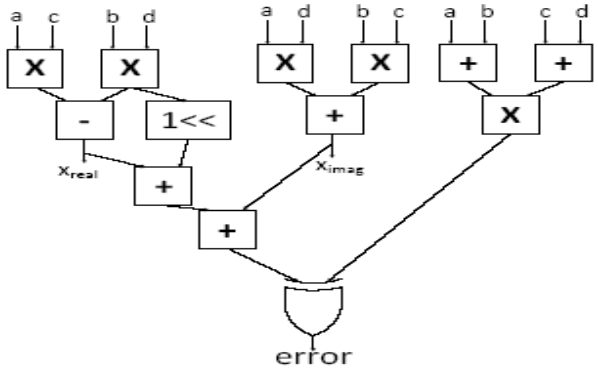

Figure 10: CED Multiplier

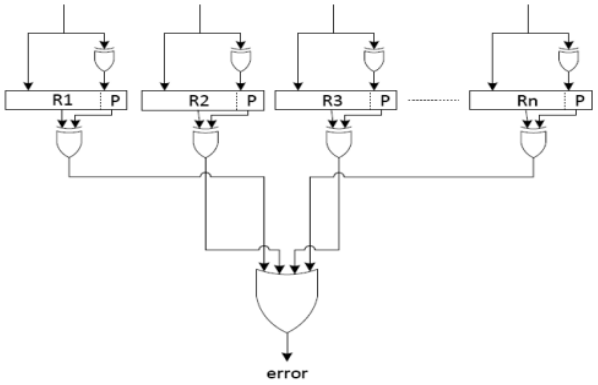

Figure 11: Parity check registers

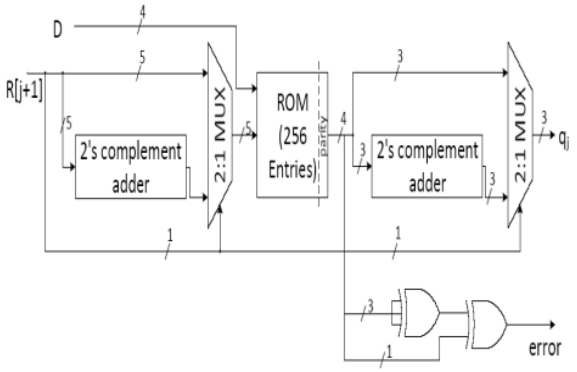

Figure 12: Minimized ROM with the proposed parity prediction logic

D. ROM (Quotient Selections 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. 12

4. 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 $F(x')$.

A. 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. 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 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 1 216 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 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 alarms 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, $1 - pn$ is the

probability of error detection for *n* parity modules. The hardware redundant modules consist of a total of 6 × 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 $1 - 0.5208 = 99.999\%$,

5. Result and Discussion:

The two proposed architecture is studied and analyzed. The two architectures are coded using VHDL hardware language. After written code, the code is simulated using Modelsim EDA tool. The simulation result of both architectures is shown below. Using Xilinx EDA tool the above code is synthesized. The synthesized result is also shown

6. FPGA Implementations and Benchmarks:

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.

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.

8. 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. C. Bartlett, "Nyquist, bode, and nichols plots of uncertain systems," in Proc. Conf. American Control, May. 1990, pp. 2033–2037.
4. S. Mijalkovic, "Using frequency response coherent structures for model-order reduction in microwave applications," IEEE Trans. Microwave Theory and Techniques, vol. 52, no. 9, pp. 2292–2297, Sep. 2004.
5. R. L. Smith, "Algorithm 116: Complex division," Communications of the ACM, vol. 5, no. 8, p. 435, 1962.
6. G. W. Stewart, "A note on complex division," ACM Trans. Mathematical Software, vol. 11, no. 3, pp. 238–241, 1985.

7. 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. Computer Arithmetic, 2013, pp. 83–90.
8. M. D. Ercegovac and J. M. Muller, "Complex division with prescaling of operands," in Proc. IEEE Int. Conf. Application-Specific Systems, Architectures, and Processors, 2003, pp. 304–314.
9. 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. Applicationspecific Systems, Architectures and Processors, 2009, pp. 83–90.
10. Wang, M. D. Ercegovac, and N. Zheng, "A radix-8 complex divider for FPGA implementation," in Proc. IEEE Int. Conf. Field Programmable Logic and Applications, 2009, pp. 236–241.
11. 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, 2012.
12. Edman and V. Oewall, "Fixed-point implementation of a robust complex valued divider architecture," in Proc. European Conf. Circuit Theory and Design, Aug. 2005.
13. J. Liu, B. Weaver, and Y. Zakharov, "FPGA implementation of multiplication-free complex division," Electronic Letters, vol. 44, no. 2, pp. 95–96, Jan. 2008.
14. T. Jamil, "An introduction to complex binary number system," in Proc. Int. Conf. Information and Computing, 2011, pp. 229–232.
15. T. Jamil and S. S. Al-Abri, "Design of a divider circuit for complex binary numbers," in Proc. World Congress on Engineering and Computer Science, vol. 2, 2010.
16. J. Liu, "DCD algorithm: Architectures, FPGA implementations and applications," in Diss. University of York, 2008.
17. S. Srinivasan, "FLAW: FPGA lifetime awareness," in Design Automation Conference, 2006, pp. 630–635.
18. C. Guérin, V. Huard, and A. Bravaix, "The energy-driven hot-carrier degradation modes of nMOSFETs," IEEE Trans. Device and Materials Reliability, vol. 7, no. 2, pp. 225–235, Jun. 2007.
19. D. K. Schroder and J. A. Babcock, "Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing," Journal of Applied Physics, vol. 94, no. 1, pp. 1–18, Jul. 2003.
20. P.J. Clarke, A. K. Ray, and C. A. Hogarth, "Electromigration-a tutorial introduction," Int. Journal of Electronics, vol. 69, no. 3, pp. 333–338, Feb. 1990.
21. D. Esseni, J. D. Bude, and L. Selmi, "On interface and oxide degradation in VLSI MOSFETs-part I: Deuterium effect in che stress regime," IEEE Trans. Electron Devices, vol. 49, no. 2, pp. 247–253, Feb. 2002.
22. D. Esseni, J. D. Bude, and L. Selmi, "On interface and oxide degradation in VLSI MOSFETs-part II: Fowler-nordheim stress