Low Complexity Concurrent Error Detection for Complex Multiplication

Salvatore Pontarelli, Pedro Reviriego, Chris J. Bleakley and Juan Antonio Maestro

Abstract— This paper studies the problem of designing a low complexity Concurrent Error Detection (CED) circuit for the complex multiplication function commonly used in Digital Signal Processing circuits. Five novel CED architectures are proposed and their computational complexity, area and delay evaluated in several circuit implementations. The most efficient architecture proposed reduces the number of gates required by up to 30% when compared with a conventional CED architecture based on Dual Modular Redundancy. Compared to a Residue Code CED scheme, the area of the proposed architectures is larger. However, for some of the proposed CEDs delay is significantly lower with reductions exceeding 30% in some configurations.

***Index Terms*— Complex multiplication, Concurrent Error Detection, Fault Tolerance.**

I. Introduction

In recent years, the numbers of soft errors occurring in digital Integrated Circuits (ICs) has been increasing due to reductions in feature sizes and supply voltages [1]. This has prompted renewed research interest in developing methods for detecting the occurrence of soft errors and single faults. Concurrent Error Detection (CED) seeks to identify errors in parallel with calculation of the circuit output [2]. Often CED solutions employ Dual Modular Redundancy (DMR) whereby two copies of the module to be protected are instantiated and a comparator flags an error when the outputs differ. Typically, DMR doubles the area of the module.

Digital Signal Processing (DSP) modules are common in ICs for communications and multimedia [3]. In the case of DSP, CED techniques often make use of the mathematical properties of, or the relationships between, a module's inputs and outputs to provide efficient solutions [4]. A number of publications have proposed efficient techniques for CED in arithmetic modules. In [5], [6], [7], residue codes were used to detect errors in real multiplications. The technique computes the residues, modulo *m,* of the multiplier inputs, multiplies these residues, and calculates the residue modulo *m,* of the result. The method compares this residue with the conventional multiplication output, modulo *m*. The choice of the value of *m* is critical [7], and depends on the implementation of the adders and multipliers to protect and on the required fault detection coverage. The use of higher values of *m* increases the capability of the circuit to detect errors, but also increases area and delay overhead.

Residue codes have been extended by combining the approach with time redundancy [8]. Parity prediction has also been studied as a method to protect multipliers [9]. The implementation of fault tolerant multipliers has also been studied in the context of security to prevent attacks [10].

Previous work focuses on multiplication of two real numbers. However, in DSP systems, complex numbers are commonly used. CED schemes optimized for complex multiplication would be beneficial both in terms of circuit cost (area) and circuit speed (delay). In this paper, we investigate CED for complex multiplication, an operation common to many DSP algorithms, based on arithmetic transformations.

Herein, we propose five efficient architectures for CED in complex multiplication and assess their computational complexity, circuit area and delay. To the authors' knowledge, the proposed architectures have not been described previously.

The following section provides background information on implementation of complex multiplication in ICs. Section III presents conventional and proposed schemes for CED in complex multiplication. Section IV presents an evaluation of the proposed techniques in terms of circuit area and delay for various multiplier configurations. Finally, the conclusions of the paper are summarized in section V.

II. Background

Complex multiplication of two complex numbers, $a+b \bullet j$ and $c+d \bullet j$, is defined as:

$$y = (a \bullet c - b \bullet d) + (a \bullet d + b \bullet c) \bullet j \quad (1)$$

where j represents the square root of -1. This direct implementation of the operation requires 4 real multiplications and 2 additions, for a total of 6 arithmetic operations.

In DSP systems, a more efficient, indirect, implementation is typically used [11]:

$$\begin{aligned} t_1 &= (a + b) \bullet (c + d) \\ t_2 &= a \bullet c \\ t_3 &= b \bullet d \\ y &= (t_2 - t_3) + (t_1 - t_2 - t_3) \bullet j \end{aligned} \quad (2)$$

Manuscript received March 28, 2012. This work was supported by the Spanish Ministry of Science and Education under Grant AYA2009-13300-C03. This paper is part of a collaboration in the framework of COST ICT Action 1103 "Manufacturable and Dependable Multicore Architectures at Nanoscale".

S. Pontarelli is with University of Rome "Tor Vergata", Via del Politecnico 1 - 00133 Rome, Italy (phone +39-0672597344; email: pontarelli@ing.uniroma2.it).

P. Reviriego and J.A. Maestro are with Universidad Antonio de Nebrija, C/ Pirineos, 55 E-28040 Madrid, Spain (phone: +34-914521100; fax: +34-914521110; email: {previrie, jmaestro}@nebrija.es).

C. J. Bleakley is with University College Dublin, Belfield, Dublin 4, Ireland (phone: +353-17165353; fax: +353-12695396; email: chris.bleakley@ucd.ie).

Digital Object Indentifier 10.1109/TC.2012.246

This indirect approach also known as Golub´s method only requires 3 multiplications and 5 additions, for a total of 8 arithmetic operations. This is advantageous since, in conventional ICs, multiplication is much more area expensive and slower than addition.

The implementation of complex multiplication according to equations (1) and (2) is illustrated in Figures 1 and 2. This graphical representation will be useful in the following to present the proposed Concurrent Error Detection checkers. The convention for the minus boxes is that the output is equal to the left input minus the right input. For each checker, the number of subtractions is reported together with the number of additions as they have a similar complexity.

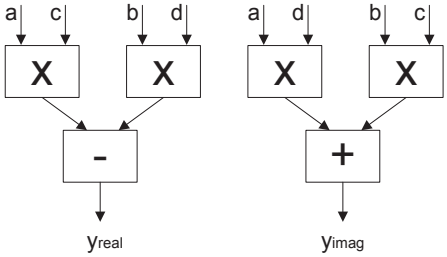

Figure 1. Direct implementation of complex multiplication.

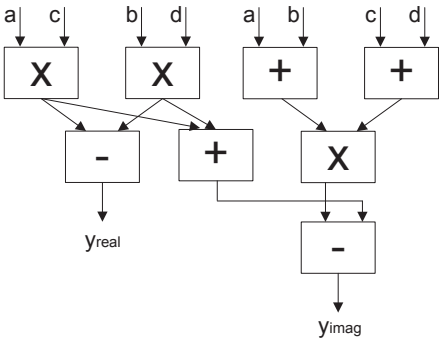

Figure 2. Indirect implementation of complex multiplication.

III. Concurrent Error Detection for Complex Multiplications

The conventional CED solution for complex multiplication is based on DMR. Using the direct implementation, this approach gives a complexity of 8 real multiplications, 4 real additions and 2 real comparisons. Using the indirect implementation, in both the original and redundant modules, leads to an overall complexity of 6 real multiplications, 10 real additions and 2 real comparisons.

Herein, we propose five novel checkers. Checker 1, based on noting that in the error-free case (see Eq. (1)):

$$y_{real} + y_{imag} = a \bullet c - b \bullet d + a \bullet d + b \bullet c \quad (3)$$

Hence, from Eq. (2), we can verify that:

$$y_{real} + y_{imag} = t_1 - 2 \bullet b \bullet d \quad (4)$$

Based on this, it would seem that y can be checked by simply adding its real and imaginary parts and comparing the result with $t_1 - 2 \bullet b \bullet d$. This would eliminate 1 multiplication in the checking module since times -2 can be implemented simply with a bit shift and bit flip. However, using this approach, an error in t_2 ($=a \bullet c$) cannot be detected since it affects y_{real} and y_{imag} in equal and opposite ways, meaning that their sum is unchanged. An additional check must be introduced to detect errors in calculation of t_2. This adds a redundant multiplication giving a complexity of 6 multiplications, 9 additions and 2 comparisons. The scheme is shown in Figure 3 where the elements of the checker are shaded.

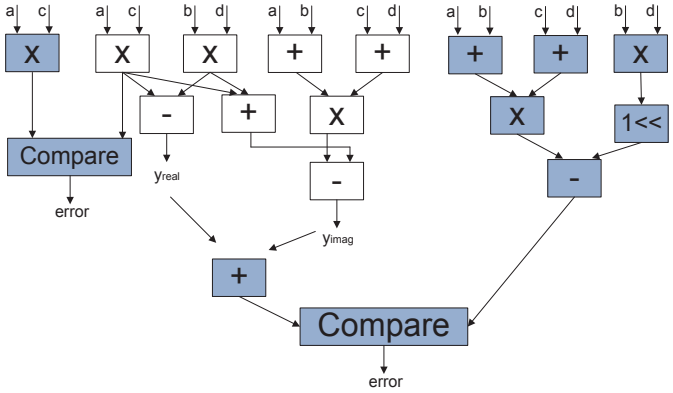

Figure 3. CED using Checker 1.

Checker 2 is based on noting that in the indirect implementation, calculation of y_{imag} shares all of the steps needed for calculation of y_{real}, except for subtraction of t_3 ($=b \bullet d$) from t_2 ($=a \bullet c$). Hence, the complex multiplication results can be checked by verifying that:

$$y_{imag} = a \bullet d + b \bullet c \quad (5)$$
$$y_{real} = t_2 - t_3$$

The scheme reduces the number of multiplications to 5 and has 7 additions and 2 comparisons. The checker is illustrated in Figure 4.

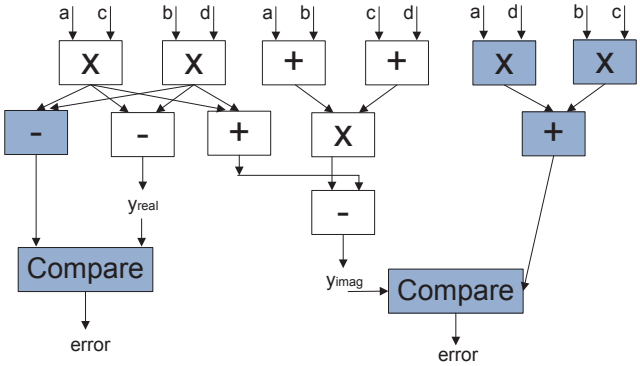

Figure 4. CED using Checker 2.

Checker 3 is based on re-arranging Eq. (4) and verifying that:

$$y_{real} + y_{imag} + 2 \bullet b \bullet d = t_1 \quad (6)$$

This checker is not efficient when used with the indirect implementation since $b \bullet d$ is not available as part of the original module and since t_2 must be checked individually (as before, errors in t_2 cancel when y_{real} is added to y_{imag}).

However, the checker can be efficiently implemented using the direct implementation for the original module. This approach has the advantage that y_{real} and y_{imag} are calculated independently and so a single error cannot affect both values. In addition, $b•d$ is already available in the original module and does not need to be calculated as part of the checker. The resulting architecture is shown in Figure 5. The complexity of this approach is 5 multiplications, 6 additions and 1 comparison.

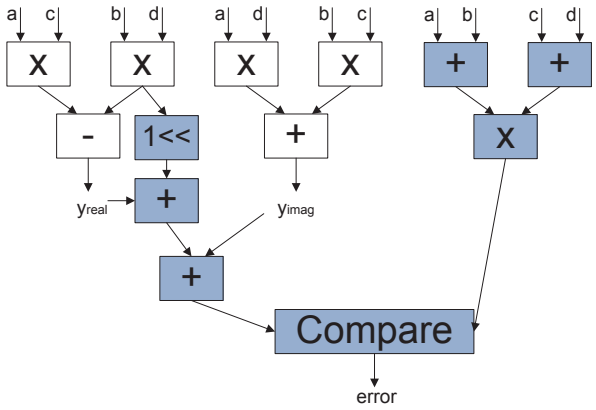

Figure 5. CED using Checker 3.

Checker 4 is based on the indirect implementation and checks the following equality:

$$y_{imag} - (a•c + b•d) = (a – b)•(d - c) \quad (7)$$

In this case the subtraction $(a•c)$-$(b•d)$ has to be duplicated, as was the case in Checker 2, as it is not covered in Eq. (7). This checker reduces the number of real multiplications required to four. The checker is illustrated in Figure 6.

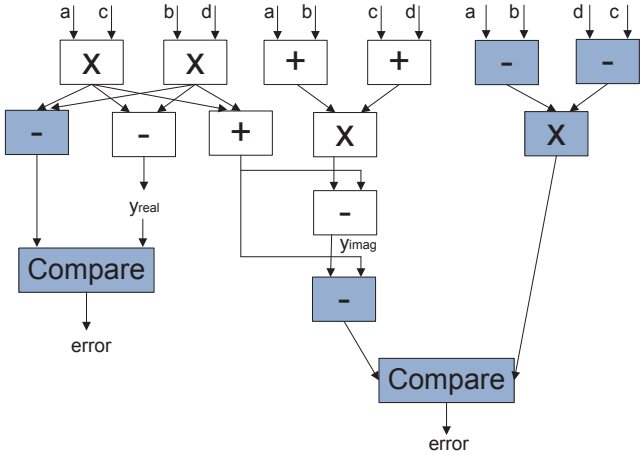

Figure 6. CED using Checker 4.

Finally, Checker 5 is based on an implementation of complex multiplication that is different from those presented in section II. The checker is shown in Figure 7. It computes the real and imaginary part as:

$$y_{real} = a•c - b•d$$
$$y_{imag} = [(a + b)•(c + d) - (a - b)•(c - d)]/2 \quad (8)$$

Then CED can be implemented by checking:

$$y_{real} + y_{imag} = (a + b)•(c + d) - 2•(b•d) \quad (9)$$

which requires only 4 multiplications. This method can be seen as an extension of the Karatsuba formula to detect errors in finite field multipliers presented in [12].

Some calculations, for example $(a + b)•(c + d)$ are used in both sides of the checker. However they have different multiplication weights. For example, the output of $(a + b)•(c + d)$ propagates to the Left Hand Side (LHS) of Eq. (9) divided by two and to the Right Hand Side (RHS) directly. Therefore, an error in $(a + b)•(c + d)$ will cause a mismatch between the LHS and RHS of Eq. (9) and an error will be detected. For example, consider the input values: a=5; b=3; c=8 and d=9. We obtain:

$$y_{real} = 5•8 - 3•9 = 13$$
$$y_{imag} = [(5+3)•(8+9)-(5-3)•(8-9)]/2=[8•17+2]/2 = 69$$
$$LHS = y_{real}+y_{imag} = 13+69 = 82$$
$$RHS = (a+b)•(c+d)-2•(b•d) = 136-2•3•9 = 136-54 = 82$$

If an error occurs in $(a+b)$ changing its value *e.g.* from *8* to ***9*** (for erroneous values we use red, bold font) the computation becomes:

$$y_{real} = 5•8-3•9 =13$$
$$y_{imag} = [(5+3)•(8+9)-(5-3)•(8-9)]/2=[\mathbf{9}•17+2]/2 = \mathbf{77}$$
$$LHS = 13+\mathbf{77} = \mathbf{90}$$
$$RHS = \mathbf{9}•17-2•3•9 = \mathbf{153}-54 = \mathbf{99}$$

Hence, the values on the LHS and RHS differ, and the error is detected.

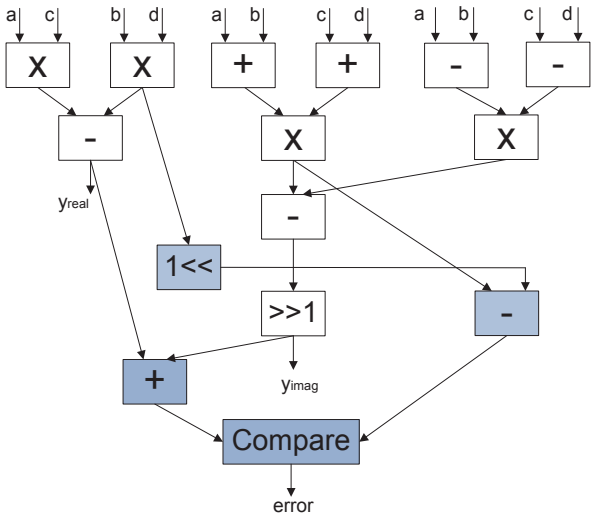

Figure 7. CED using Checker 5.

The complexity and delay of the CED schemes are summarized in Tables I and II in terms of the number of arithmetic operations. As can be seen, the Checker 5 has the lowest computational complexity. It is notable that the highest complexity multiplier structure considered leads to the simplest checker and the lowest complexity overall. For the direct implementation, the checker with the lowest complexity is Checker 3. For the indirect implementation, Checker 4 has the lowest cost, with the same number of multiplications as Checker 5. The checker based on DMR of the direct implementation is the fastest solution. All of the proposed

schemes incur in some delay overhead. In the next section circuit implementations of the checkers are presented to evaluate their circuit area and circuit delay.

TABLE I
COMPUTATIONAL COMPLEXITY OF THE MULTIPLIERS WITH CED.

Architecture	Real Multiplications	Real Additions	Comparisons
Direct DMR	8	4	2
Indirect DMR	6	10	2
Checker 1	6	9	2
Checker 2	5	7	2
Checker 3	5	6	1
Checker 4	4	9	2
Checker 5	4	8	1

TABLE II
DELAY OF THE MULTIPLIERS WITH CED.

Architecture	Real Multiplications	Real Additions	Comparisons
Direct DMR	1	1	1
Indirect DMR	1	2	1
Checker 1	1	3	1
Checker 2	1	2	1
Checker 3	1	3	1
Checker 4	1	3	1
Checker 5	1	3	1

IV. EVALUATION

All five proposed checkers were proven analytically to provide 100% accurate fault detection when implemented with full precision with respect to the input data bit-width.

To evaluate the circuit complexity and delay of checkers, they were implemented in VHDL and synthesized for the 45nm OSU FreePDK Standard Cell library [13] using Synopsys Design Compiler. The synthesis was done using independent modules for the original multiplication and the checkers so that the protection logic was not removed by Design Compiler during the optimization process.

In a first set of implementations, full precision checking was used for various input bit-widths. No truncation or rounding is done such that the outputs have approximately twice the number of bits of the inputs. In a second set of implementations, reduced precision checking is used. In all cases, two options for synthesis were considered: area optimization and speed optimization. The results for the first option provide insight into the minimum circuit area and the second provides insight into the minimum delay.

To compare with other error detection techniques, Dual Modular Redundancy (DMR) and Concurrent Error Detection using residue codes were implemented. DMR was considered for both the direct and the indirect implementations and also for reduced precision checking. For residue codes, the scheme was applied by performing the modulo operation at the inputs and outputs of the complex multiplication such that both adders and multipliers are protected. Modulo 7 was used to obtain good error coverage[1] [5]. Concurrent error detection using residue codes does not allow for truncation and therefore these schemes were not considered for reduced precision checking. The results for the unprotected implementations are also reported for completeness.

The circuit area in terms of the number of equivalent NAND2 gates for the full precision implementations is reported in Table III. Comparing the direct and indirect unprotected multipliers it can be observed that the area savings of the indirect implementation over the direct one are small. This is due to the optimizations performed by Synopsys Design Compiler, which works at lower abstraction levels, (use of carry-save representation, high radix Booth recoding, conversion from Sum of Product to Product of Sum, *etc.*) [14]. These optimizations can produce unexpected results with unconventional arithmetic such as that in Eq. (2). This has been observed previously in floating-point implementations of complex multipliers where indirect implementations had no advantage over the direct ones [15]. In our case, the indirect implementation gives better results but with only small savings over the direct implementation.

It can be observed that the DMR versions require more than two times the number of gates of the unprotected implementation as expected. Similarly to the unprotected version, there are little benefits in using the indirect DMR with respect to the direct DMR. Checker 5 has the least area with reductions of up to 30% compared with the most efficient DMR checker. The other checkers also provide significant reductions in the number of gates compared to DMR. The Residue CED scheme provides similar area to that of Checker 5 when used with the direct implementation and significantly less when used with the indirect implementation.

TABLE III
AREA OF THE MULIPLIERS WITH AND WITHOUT CED (NAND2 EQUIVALENT GATES). AREA OPTIMIZED SYNTHESIS.

Architecture	Area		
	8 bit input	12 bit input	16 bit input
Direct unprotected	1,636	3,313	5,553
Indirect unprotected	1,599	2,993	5,152
Direct DMR	3,888	6,815	11,587
Indirect DMR	3,796	6,548	10,714
Residue Direct	2,870	5,336	7,008
Residue Indirect	2,412	4,352	6,270
Checker 1	3,303	5,612	9,271
Checker 2	2,900	5,928	8,437
Checker 3	2,876	5,903	8,305
Checker 4	3,003	5,858	8,034
Checker 5	2,711	5,434	7,505

[1]Synthesis was performed by configuring Synopsys Design Compiler to implement parallel prefix structures for the adders and multipliers. For these structures, modulo 7 may not be sufficient to achieve the desired error detection coverage depending on the application. In those cases larger values of *m* would be required with the associated area overhead.

The delay results, in nanoseconds for the full precision implementations are reported in Table IV. The direct DMR implementation provides the lowest delay with an overhead of up to 8% over the unprotected version. Checker 5 has the highest delay with an overhead of up to 66% over the direct DMR implementation and up to 76% over the unprotected version. Other checkers, such as Checker 2 have only a 15% delay overhead over the direct DMR implementation and a 21% overhead over the unprotected version in the worst case. The Residue CED schemes incur delays that are significantly larger than that of Checker 2 but less than that of Checker 5.

TABLE IV
DELAY OF THE MULTIPLIERS WITH AND WITHOUT CED (NANOSECONDS). DELAY OPTIMIZED SYNTHESIS.

Architecture	Delay		
	8 bit input	12 bit input	16 bit input
Direct unprotected	1.02	1.16	1.26
Indirect unprotected	1.19	1.40	1.48
Direct DMR	1.10	1.22	1.31
Indirect DMR	1.30	1.50	1.54
Residue Direct	1.40	1.58	1.79
Residue Indirect	1.53	1.76	1.97
Checker 1	1.41	1.59	1.70
Checker 2	1.23	1.40	1.48
Checker 3	1.29	1.45	1.55
Checker 4	1.43	1.63	1.71
Checker 5	1.80	2.03	2.17

In many DSP systems, it is sufficient to detect errors that have a large magnitude. This has led to the concept of reduced precision redundancy [16]. In our case, reduced precision can be used in the elements that are needed for the CED but not for the main complex multiplication operation to reduce area. These elements are shaded in Figures 3-7. Reduced precision cannot be used with the Residue checkers. Therefore those schemes are not included in the comparison.

To evaluate the effects of reduced precision on the area and delay, implementations using 16 bits inputs for the main operation and 8 bits inputs for the checker components were evaluated. This ensures that large errors that affect the most significant bits will be detected. The magnitude of the smallest error that will be detected can be calculated by analyzing the quantization in the different checkers. Since the objective is only to study how reduced precision affects the implementation complexity and speed of the different checkers, a full analysis is not included in the paper. However, assuming that the inputs are normalized to 0.5 such that multiplications do not increase the quantization error, the largest undetected error can be bounded. As truncation is only done at the inputs of the checkers, in the worst case the truncation errors add to give a value of the number of truncated inputs times the maximum truncation error. If that value is used as threshold to detect errors then the maximum undetected error will in the worst case be twice that value (as in that case the error cannot be masked by quantization effects).

The areas of the reduced precision checker implementations are reported in Table V**.** Checker 2 provides the lowest area with a reduction of 11% compared to the smallest DMR checker. The reduced precision Checker 2 architecture provides a 17% area saving relative to the smallest full precision architecture (Checker 5). Checker 5 shows the least area saving since all multiplication operations are part of the main complex multiplication operation, as shown in Fig. 7, and must therefore be full precision.

TABLE V
AREA OF THE MULTIPLIERS WITH REDUCED PRECISION CED (NAND2 EQUIVALENT GATES). AREA OPTIMIZED SYNTHESIS.

Architecture	Area
Direct DMR	7,279
Indirect DMR	6,991
Checker 1	6,574
Checker 2	6,192
Checker 3	6,734
Checker 4	6,241
Checker 5	7,272

The delay results in nanoseconds for the reduced precision implementations are reported in Table VI. In this case, the delay for the DMR implementation increases slightly as the comparison is no longer a simple equality check. Checker 2 provides the lowest delay overhead, 13%, similar to that of a full precision implementation.

In summary, we can conclude that the Residue scheme using the indirect implementation should be used to minimize the circuit complexity (area) when delay is not an issue and full precision checking is required. Minimum delay is provided by the Direct DMR architecture. However, if near-minimum delay is acceptable, Checker 2 should be used to reduce circuit complexity. If reduced precision checking is acceptable for the application, Checker 2 provides lowest circuit area and is smaller than the smallest full precision alternative.

TABLE VI
DELAY OF THE MULTIPLIERS WITH REDUCED PRECISION CED (NANOSECONDS). DELAY OPTIMIZED SYNTHESIS.

Architecture	Delay
Direct DMR	1.34
Indirect DMR	1.51
Checker 1	1.85
Checker 2	1.52
Checker 3	1.64
Checker 4	1.84
Checker 5	1.78

V. CONCLUSIONS

This paper proposed five new Concurrent Error Detection architectures for complex multiplication. The benefits in terms of circuit complexity reductions were evaluated for various configurations showing reductions of up to 30% in the area compared to DMR. Compared to a Residue CED scheme, the

area is greater but the delay for some of the proposed architectures is significantly lower showing reductions of over 30% in some configurations. The use of reduced precision for the CED elements was also considered showing that it results in a different CED architecture providing the best results. The proposed solutions provide a wide set of efficient building blocks to implement CED in DSP systems.

REFERENCES

[1] R. Baumman "Soft errors in advanced computer systems" IEEE Design and Test of Computers, vol. 22, no. 3, pp. 258 – 266, 2005.
[2] M. Nicolaidis, "Design for soft error mitigation", IEEE Transactions Device and Materials Reliability vol. 5, no. 3, pp. 405–418, Sept. 2005.
[3] Oppenheim A. V. and Schafer R., Discrete Time Signal Processing, Prentice Hall, 1999.
[4] A. Reddy and P. Banarjee "Algorithm-based fault detection for signal processing applications", IEEE Transactions on Computers, vol. 39, no. 10, pp. 1304-1308, Oct. 1990.
[5] U. Sparmann and S.M. Reddy "On the effectiveness of residue code checking for parallel two's complement multipliers", 24th Int. Symp. On Fault-Tolerant Computing (FTCS-24), pp. 219-228, Jun. 1994.
[6] T.J. Brosnan and N.R Strader. "Modular error detection for bit-serial multiplication", IEEE Transactions on Computers, vol. 37, no. 9, pp. 1043-1052, Sept. 1988.
[7] I. Alzaher-Noufal and M. Nicolaidis "A CAD Framework for Generating Self-Checking Multipliers Based on Residue Codes" Design and Test in Europe (DATE), pp. 122-129, March 1999.
[8] R. Forsati, K. Faez, F. Moradi, A. Rahbar, "A Fault Tolerant Method for Residue Arithmetic Circuits", in proc. of IEEE International Conference on Information Management and Engineering, (ICIME'09), 2009, pp. 59-63.
[9] Z. Wang, M. Karpovsky, B. Sunar and A. Joshi, "Design of Reliable and Secure Multipliers by Multilinear Arithmetic Codes", Lecture Notes in Computer Science, 2009, vol. 5927, pp. 47-62, 2009.
[10] M. Hunger and D. Marienfeld "New self-checking booth multipliers", International Journal of Applied Mathematics and Computer Science, vol. 18, No. 3, pp. 319–328, 2008.
[11] J. W. Hartwell, "A Procedure for Implementing the Fast Fourier Transform on Small Computers", IBM Journal of Research and Development, vol. 15, pp. 355-363, 1971.
[12] S. Pontarelli, A. Salsano, "On the use of Karatsuba formula to detect errors in GF((2^n)^2) multipliers", IET Circuits, Devices & Systems, vol. 6 no. 3, pp 152-158, 2012
[13] J. E. Stine et al., "FreePDK: An open-source variation-aware design kit," in Proc. IEEE Int. Conf. Microelectronic Systems Education, (MSE'07), Jun. 2007, pp. 173–174.
[14] R. Zimmermann, "Datapath Synthesis for Standard-Cell Design", in Proc. of the 9th IEEE Symposium on Computer Arithmetic (ARITH) 2009, pp. 1207--211.
[15] E. E. Swartzlander and H. H. Saleh, "Floating-point implementation of complex multiplication", in Proc. of the IEEE Forty-Third Asilomar Conference on Signals, Systems and Computers, 2009, pp. 926-929.
[16] B. Shim and N. Shanbhag, "Energy-efficient soft error-tolerant digital signal processing," IEEE Transactions on VLSI Systems, vol. 14 no. 4, pp. 336–348, 2006.