Efficient Online Self-Checking Modulo $2^n + 1$ Multiplier Design

Wonhak Hong, Rajashekhar Modugu, and Minsu Choi, *Senior Member*, *IEEE*

Abstract—Modulo $2^n + 1$ multiplier is one of the critical components in the area of data security applications such as International Data Encryption Algorithm (IDEA), digital signal processing, and fault-tolerant systems that demand high reliability and fault tolerance. Transient faults caused by electrical noise or external interference are resulting in soft errors which should be detected online. The effectiveness of the residue codes in the self-checking implementation of the modulo multipliers has been rarely explored. In this paper, an efficient hardware implementation of the self-checking modulo $2^n + 1$ multiplier is proposed based on the residue codes. Different check bases in the form $2^c - 1$ or $2^c + 1$ $(c \in N)$ are selected for various values of the input operands. In the implementation of the modulo generators and modulo multipliers, novel multiplexor-based compressors are applied for efficient modulo $2^n + 1$ multipliers with less area and lower power consumption. In the final addition stage of the modulo multipliers and modulo generators, efficient sparse-tree-based inverted end around carry adders are used. The proposed architecture is capable of online detecting errors caused by faults on a single gate at a time. The experimental results show that the proposed self-checking modulo $2^n + 1$ multipliers have less area overhead and low performance penalty.

Index Terms—Modulo $2^n + 1$ multiplier, residue arithmetic, arithmetic circuit design, compressor, online self-checking, international data encryption algorithm (IDEA).

1 Introduction

In the recent years, the number of the Internet and wireless communication nodes has grown rapidly, which involves the transmission of data over channels. The confidentiality and security requirements are becoming more and more important to protect the data transmitted and received, and consequently, secured communication of the data is given the utmost priority. Various cryptographic systems have been studied and implemented to ensure the security of these systems. International Data Encryption Algorithm (IDEA) is one of the most reliable cryptographic algorithms used for transmission of the data [10], [11]. In the hardware implementation of IDEA, the three equation major operations that decide the delay and the overall performance of IDEA cipher are modulo 2^n addition, bitwise-XOR, and modulo $2^n + 1$ multiplication. The performance of data path of the IDEA cipher significantly depends on the modulo $2^n + 1$ multiplication module. Apart from this, the modulo $2^n + 1$ module has found applications in Fermat number transform computation [12], digital signal processing [1], and fault-tolerant design of ad hoc networks [2]. Hence, the efficient and fault-secured design of the modulo $2^n + 1$ multiplier is highly desired. The protection against errors is necessary in security applications such as IDEA for reliability.

In VLSI systems, transient faults can be detected by built in online fault detection circuits and self-checking circuits have this property [24], [29], [30]. Especially, transient faults caused by internal noise or external interference are not tolerable in this high-speed computing world, these faults should be detected online. The self-checking designs detect the errors immediately as they occur and the output can be corrected by repeating the last operation. Hence, it is highly desirable to design efficient algorithms and methods that can detect the errors online, which may prevent any harm caused by the faults. Designing efficient self-checking circuits with less area overhead and performance penalty has been an important challenge in the area of fault-tolerant applications. In the recent years, various self-checking circuits [13], [14], [15], [16] using different coding schemes such as parity prediction and arithmetic codes are presented to check the functionality of the circuits. Self-checking arithmetic circuits using residue codes are reported in some of the industry applications [26]. Parity code schemes for memory systems and register files may achieve fault-secure property at low hardware cost, and sometimes, they fail to achieve the fault-secure property in arithmetic circuits [19], [21]. In the self-checking arithmetic circuits, using parity prediction scheme [23] detects errors only at the output of the circuit; however, a fault in the intermediate signal gets propagated to other output signals and remains undetected. Even though the parity codes result in less area overhead in self-checking circuits, arithmetic circuits may sometimes produce multiple output errors which are not detectable by the parity codes. Especially in case of multipliers, the circuit overhead is in the range of 40-50 percent when parity prediction codes are used [18]. Therefore, in the multipliers with large input operand width, the overall area overhead is adversely affected with parity prediction schemes.

In the literature, self-checking implementations of the arithmetic circuits such as adders and multipliers have been proposed [27], [28]. These self-checking circuits use parity prediction schemes and arithmetic code schemes in the design by trading off with the hardware overhead,

- *W. Hong is with the Department of Electrical and Electronic Engineering, Ulsan College (West Campus), 411Ho, 1-Engineering Building, Ulsan, South Korea. E-mail: whhong@mail.uc.ac.kr.*
- *R. Modugu and M. Choi are with the Department of Electrical and Computer Engineering, Missouri University of Science and Technology, Mail-Room, Rolla, MO 65409. E-mail: {rrmt4b, choim}@mst.edu.*

Manuscript received 7 June 2009; revised 15 Jan. 2010; accepted 4 Feb. 2010. Recommended for acceptance by C. Metra and R. Galivanche.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TCSI-2009-06-0261.
Digital Object Identifier no. 10.1109/TC.2010.49.

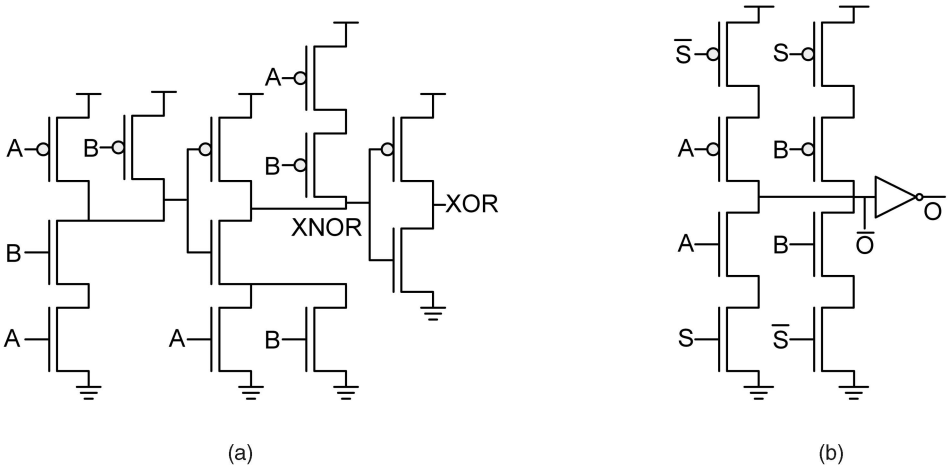


Fig. 1. CMOS implementation of 2-input. (a) XOR. (b) MUX.

performance penalty, and fault secureness. Few of the self-checking circuits are described as follows: In [22], self-checking adder circuits based on arithmetic codes were proposed [25]. These checkers using arithmetic codes suffered from hardware complexity and overhead. An alternative method to design self-checking adders [23] is by using parity prediction schemes. However, this approach detects errors only at the outputs and a fault in the carry gets propagated and remains undetected leading to unreliability. A different technique is used in [13], to design a self-checking carry-select adder. The adders are totally self-checking for both permanent and transient stuck-at-faults. A self-checking multiplier is proposed in [24], based on parity prediction scheme. The multiplier consists of AND matrix, Carry Save Adder, and a final sum-bit duplicated adder. Single stuck-at-faults in the combinational logic and all even or odd errors in one of the duplicated output registers are detected. In [16], a self-checking code-disjoint booth multiplier based on linear Carry Save Addition is designed. This implementation can detect all the single input faults, single stuck-at-faults, and all errors in the output register. In the work of [16], [35], it is described that the transient faults in the circuits create soft errors in the output latches of the combinational circuit when:

- An output is related to the faulty subcircuit with respect to the input (logical condition).
- A pulse, altered by the faults, has a significant pulse width and amplitude (electrical condition).
- A pulse, resulting from the faults, arriving at the clock transition (latching window).

Because of the numerous masking effects, these transient faults result in single bit errors. Hence, circuits which can detect single input faults, single stuck-at-faults, or multiple output faults are of usual interest. In this paper, we propose a new hardware implementation of the self-checking modulo $2^n + 1$ multiplier based on residue codes. In the proposed implementation, several techniques are used to come out with an efficient self-checking modulo $2^n + 1$ multiplier and they are listed below:

- Efficient compressors are employed in the multiplier design and modulo generators design to reduce the overhead.
- The residue code circuits with the check bases of the form $2^k - 1$ and $2^k + 1$ are efficiently designed using compressors and sparse-tree-based adders.

The resulting self-checking circuit has area overhead in the range of 20-45 percent for different values of n.

The paper is organized as follows: Section 2 introduces multiplexor-based compressors, which are used in the self-checking multiplier design [5]. Section 3 discusses the proposed implementation of the self-checking modulo $2^n + 1$ multiplier, and the efficient implementation of the modulo generators with check bases $2^k - 1$ and $2^k + 1$ is given. Experimental results showing the area overhead and performance penalty of the resulting self-checking circuits are given in Section 4. Our conclusions are drawn in Section 5.

2 Preliminaries and Reviews

2.1 MUX versus XOR

Multiplextor (MUX) is one of the logic gates used extensively in the digital design, which is very useful in efficient design of arithmetic and logic circuits. According to the CMOS implementation of MUX [6], it performs better in terms of power and delay compared to exclusive-OR (XOR). Suppose, X and Y are inputs to the XOR gate, the output is $X\overline{Y} + \overline{X}Y$. The same XOR can be implemented using MUX with inputs $X, \overline{X}$ and select bit Y. Efficient compressors have been designed using MUX and reported in [7]. In the proposed compressors [7], both output and its complement of these gates are used. This also reduces the total number of garbage outputs. Existing CMOS designs of 2:1 MUX and 2-input XOR are shown in Fig. 1 for comparison.

2.2 Description of Compressors

A $(p, 2)$ compressor has p inputs $X_1, X_2 \ldots X_{p-1}, X_p$ and two output bits (i.e., Sum bit and Carry bit) along with carry input bits and carry output bits. Its functionality can be represented by the following equation:

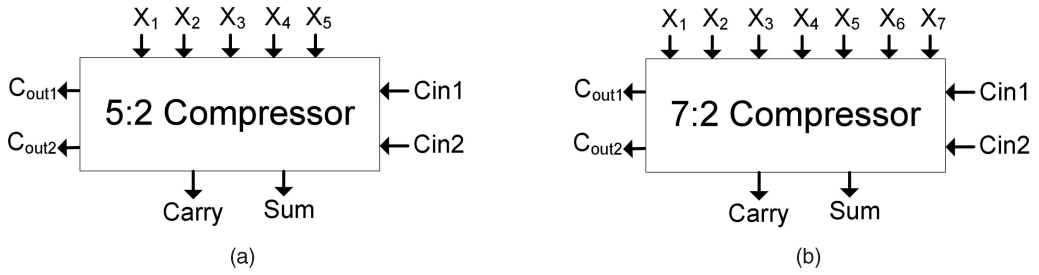


Fig. 2. Block diagrams of (a) 5:2 compressor and (b) 7:2 compressor.

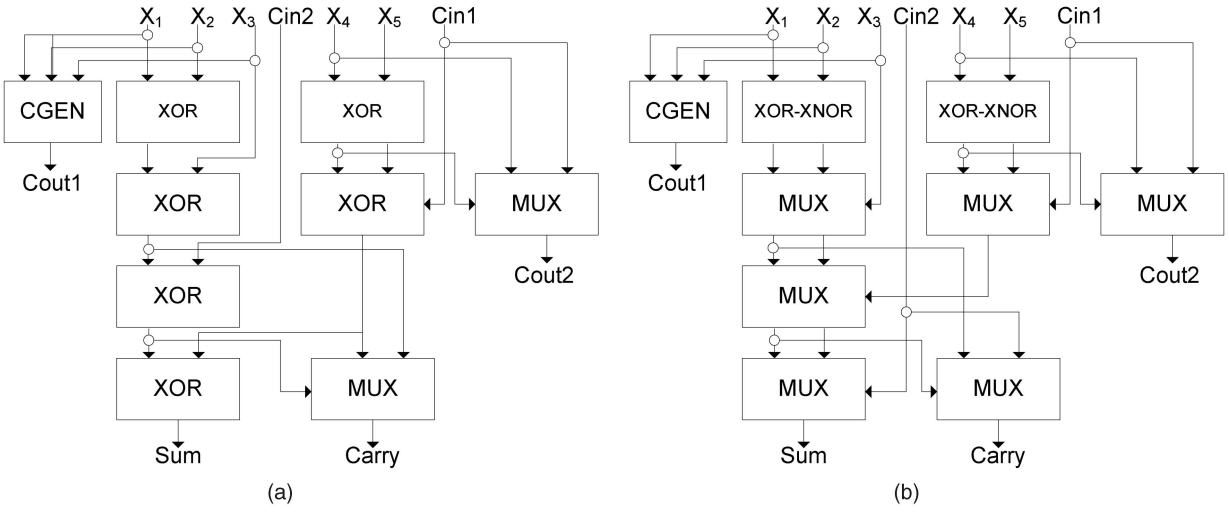


Fig. 3. Block diagrams of (a) existing XOR-based 5:2 compressor design and (b) new MUX-based 5:2 compressor design.

$$\sum_{i=1}^{p} X_i + \sum_{i=1}^{t} (C_{in})_i = Sum + 2\left(Carry + \sum_{i=1}^{p} (C_{out})_i\right). \quad (1)$$

For example, a $(5, 2)$ compressor takes five inputs and two carry inputs and generates a Sum and Carry bit along with two carryout bits. Block diagrams of 5:2 and 7:2 compressors are shown in Fig. 2. Efficient designs of the existing XOR-based 7:2, 5:2, and 4:2 compressors [8], [9] have critical path delays of $6\Delta(XOR)$, $4\Delta(XOR)$, and $3\Delta(XOR)$ (delay denoted by Δ), respectively, [8].

The newly proposed efficient compressors [7] use multiplexers in place of XOR gates, resulting in high-speed arithmetic due to reduced gate delays. Also, as shown in Fig. 1, in all the existing CMOS implementations of the XOR and MUX gates, both the output and its complement are available, but the designs of compressors available in literature do not use these outputs efficiently. In CMOS implementation of the MUX if both the select bit and its complement are generated in the previous stage, then its output can be generated with much less delay because the switching of the transistor is already completed. And also if both the select bit and its complement are generated in the previous stage, then the additional stage of the inverter can be eliminated which reduces the overall delay in the critical path. The existing XOR-based and proposed MUX-based designs of a 5:2 compressor are shown in Fig. 3, the delays of which are $\Delta(XOR) + 3\Delta(MUX)$ and $4\Delta(XOR)$. These compressors are primitive blocks of the proposed self-checking modulo multipliers. The proposed MUX-based design of the 7:2 compressor is shown in Fig. 4. CGEN block used in the noncritical path shown in Fig. 3 can be obtained from the equation $C_{outi} = (X + Y) \cdot Z + X \cdot Y$.

2.3 Sparse-Tree-Adder-Based Inverted End Around Carry Adder

In binary addition operation, the critical path is determined by the carry computation module. Among various formulations to design carry computation module, parallel prefix formulation [36] is delay effective and has regular

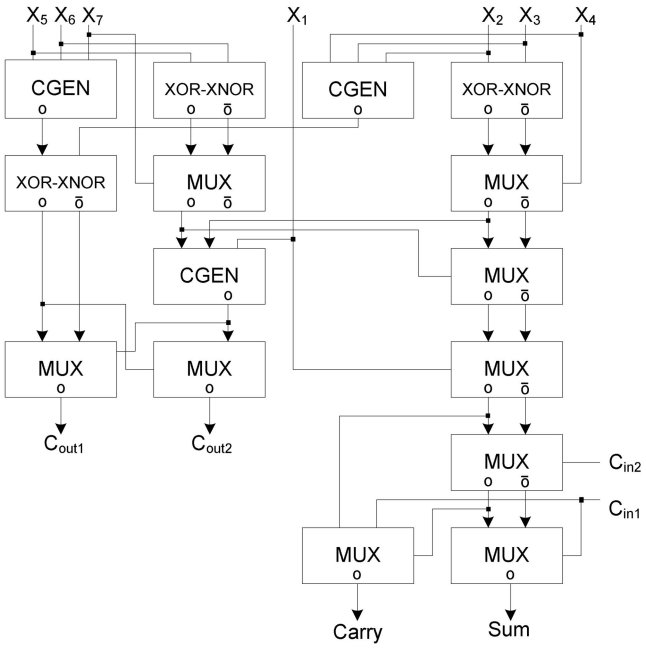


Fig. 4. Block diagram of MUX-based 7:2 compressor design.

structure suitable for efficient hardware implementation. The binary addition of two numbers using a parallel prefix network is done as follows: Let $A = a_{n-1}an-2\ldots a_1a_0$ and $B = b_{n-1}bn-2\ldots b_1b_0$ be two weighted input operands to the network. The generate bit (g_i) and propagate bit (p_i) are defined as $g_i = a_i$ AND b_i and $p_i = a_i$ OR b_i, and these generate bits can be associated using the prefix operator $\circ$ as follows: $(g_i, p_i) \circ (g_{i-1}, p_{i-1}) = (g_i + p_i \cdot g_{i-1}, p_i \cdot p_{i-1}) = (g_{i:i-1}, p_{i:i-1})$, where $+$ is the logical OR operator and $\cdot$ is the logical AND operator. The carryouts (C_i) for all the bit positions can obtained from the group generate $(G_i = C_i)$ where $(G_i, P_i) = (g_i, p_i) \circ (g_{i-1}, p_{i-1}) \circ \ldots (g_1, p_1) \circ (g_0, p_0)$.

The function of End Around Carry (EAC) adder is to feed back the carryout of the addition and add it to the least significant bit of the sum vector. Similarly, in inverted End Around Carry adders, the carryout is inverted and fed back to the least significant bit of the sum vector. The parallel prefix-network-based Inverted EAC adder [37] achieves the addition of the input operands by recirculating the generate and the propagate bits at each existing level in $log_2 n$ stages. Let $C_i^*(G_i^*)$ be the carry at bit position i in the inverted EAC, this can be related to G_i as follows:

$$(G_i^*, P_i^*) = \begin{cases} \overline{(G_{n-1}, P_{n-1})}, & for\ i = -1, \\ (G_i, P_i) \circ \overline{(G_{n-1:i+1}, P_{n-1:i+1})}, & for\ n-2 \geq i \geq 0. \end{cases} \tag{2}$$

In the above equation, $\overline{(G_i, P_i)} = (\overline{G}, P)$, where $(G_i, P_i) = (g_i, p_i) \circ (g_{i-1}, p_{i-1}) \circ \cdots \circ (g_1, p_1) \circ (g_0, p_0)$ and

$$(G_{n-1:i+1}, P_{n-1:i+1}) = (g_{n-1}, p_{n-1}) \circ (g_{n-2}, p_{n-2}) \circ \cdots \circ (g_{i+2}, p_{i+2}) \circ (g_{i+1}, p_{i+1}).$$

In some cases, it is not possible to compute (G_i^*, P_i^*) in $log_2 n$ stages, then in these cases, the equations in (2) are transformed into the equivalent ones as shown in (4) by using the following property [37]:

Suppose that $(G^x, P^x) = (g, p) \circ \overline{(G, P)}$ and $(G^y, P^y) = \overline{((\overline{p}, \overline{g}) \circ (G, P))}$:

$$\begin{aligned} G^x = g + p \cdot \overline{G} &= \overline{\overline{g + p \cdot \overline{G}}} = \overline{\overline{g} \cdot (\overline{p} + G)} \\ &= \overline{(\overline{g} \cdot \overline{p} + \overline{g}G)} = \overline{(\overline{p} + \overline{g}G)}. \end{aligned} \tag{3}$$

Therefore, $G^x = G^y$ and in (2), P^y is computed as $p \cdot P$.

To implement the parallel prefix computation efficiently, these transformations have to be applied j number of times recursively on $(G_i, P_i) \circ \overline{(G_{n-1:i+1}, P_{n-1:i+1})}$ using the following relation:

$$n - 1 - i + j = \begin{cases} n, & if\ i > \dfrac{n}{2} - 1, \\ \dfrac{n}{2}, & if\ i \leq \dfrac{n}{2} - 1. \end{cases} \tag{4}$$

The new carryouts can be computed using the following equation:

$$(G_i^*, P_i^*) = \begin{cases} \overline{(G_{n-1}, P_{n-1})}, & for\ i = -1, \\ \overline{(\overline{P_i}, \overline{G_i}) \circ (G_{n-1:i+1}, P_{n-1:i+1})}, & for\ n-2 \geq i \geq 0. \end{cases} \tag{5}$$

Hence, the transformations used above to achieve the parallel prefix computation in $log_2 n$ stages result in more number of carry merge cells and thereby adding more number of interstage wires. Parallel prefix adders suffer from excessive interstage wiring complexity and large number of cells, and these factors make parallel prefix-based adders inefficient choices for VLSI implementations. Therefore, a novel sparse-tree-based EAC and inverted EAC adders are used as the primitive blocks in this work.

In sparse-tree-based inverted EAC adders [3], [4], instead of calculating the carry term G_i^* for each and every bit position, every Kth $(K = 4, 8\ldots)$ carry is computed. The value of K is chosen based on the sparseness of the tree, generally for 16 and 32-bit adders, K is chosen as four [32], [33]. The higher value of K results in higher value of noncritical path delay compared to critical path delay of $O(log_2 n)$ which should not be the case. The proposed implementation of the sparse-tree-based Inverted End Around Carry Adder (IEAC) is explained below clearly for 16-bit operands. For a 16-bit sparse IEAC with sparseness factor (i.e., K) equal to four, the carries are computed for bit positions $-1, 3, 7$, and 11. Here, bit position -1 corresponds to the inverted carryout $\overline{((G_{15}, P_{15}))}$ of the bit position 15. The carryout equations for the 16-bit Sparse tree IEAC are as follows:

$$\begin{aligned} C_{-1}^* &= \overline{(G_{15}, P_{15})} \\ &= \overline{(g_15, p_15) \circ (g_{14}, p_{14}) \circ \cdots \circ (g_1, p_1) \circ (g_0, p_0)}, \\ C_3^* &= (G_3, P_3) \circ \overline{(G_{15:4}, P_{15:4})} = (g_3, p_3) \circ \cdots \circ (g_0, p_0) \\ &\quad \circ \overline{(g_{15}, p_{15}) \circ (g_{14}, p_{14}) \circ \cdots \circ (g_5, p_5) \circ (g_4, p_4)} \\ &= \overline{(\overline{P_3}, \overline{G_3}) \circ (G_{15:4}, P_{15:4})}, \\ C_7^* &= (G_7, P_7) \circ \overline{(G_{15:8}, P_{15:8})} = (g_7, p_7) \circ \cdots \circ (g_0, p_0) \\ &\quad \circ \overline{(g_{15}, p_{15}) \circ (g_{14}, p_{14}) \circ \cdots \circ (g_9, p_9) \circ (g_8, p_8)}, \\ C_{11}^* &= (G_{11}, P_{11}) \circ \overline{(G_{15:12}, P_{15:12})} = (g_{11}, p_{11}) \circ \cdots \circ (g_0, p_0) \\ &\quad \circ \overline{(g_{15}, p_{15}) \circ (g_{14}, p_{14}) \circ \cdots \circ (g_{13}, p_{13}) \circ (g_{12}, p_{12})}. \end{aligned}$$

Fig. 5 shows the finalized 16-bit sparse tree Inverted EAC adder. From Fig. 5, we can observe that all the carryouts are computed in $log_2 n$ stages with less number of carry merge cells and reduced interstage wiring intensity [32]. The implementation of the sparse-tree-based EAC is similar to IEAC shown in Fig. 5, except the carry is not inverted.

The Conditional Sum Generator (CSG) shown in Fig. 2C is implemented using ripple carry adder logic, and two separate rails are run to calculate the carries $C_{i+1}^*, C_{i+2}^*, C_{i+3}^*$, and C_{i+4}^* assuming the input carry C_i^* as 0 and 1. Four 2:1 multiplexers using the carry C_i^* from sparse tree network as one-in-four select line generate the final sum vector. The conditional sum generator is shown in Fig. 2C. The final sum is generated in $log_2 n$ stages in IEAC sparse tree adder with less number of cells and less interstage wiring. Hence, this approach results in low power and smaller area while providing better performance.

2.4 Modulo $2^n + 1$ Multiplier

Modulo $2^n + 1$ multiplier is extensively used in many digital signal processors and cryptographic applications. As $2^n + 1$ is an $n + 1$ bit number, the input operands can be of $n + 1$ bits.

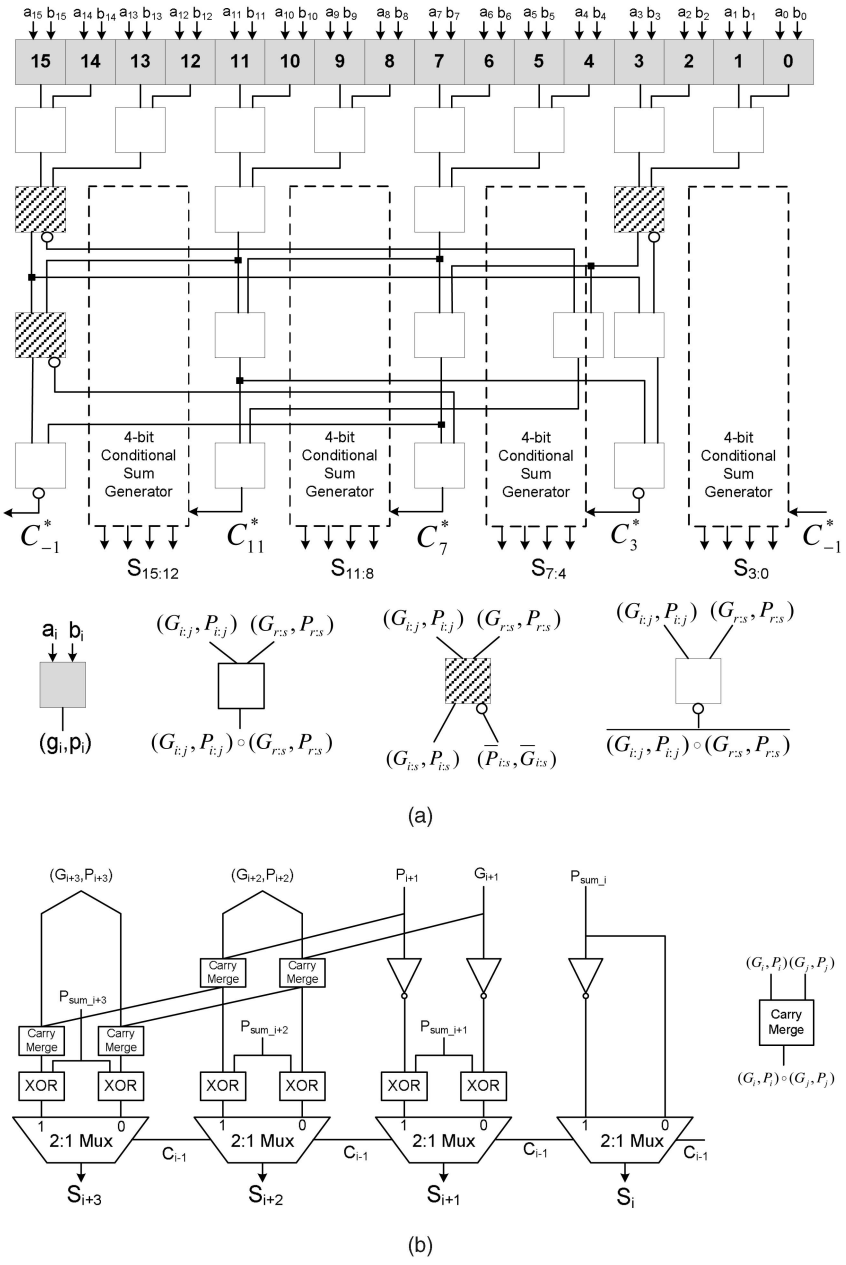


Fig. 5. (a) 16-bit Sparse-tree-based Inverted EAC adder and (b) 4-bit conditional sum generator.

A brief explanation of the algorithm and implementation of the modulo $2^n + 1$ multiplier [3], [4] is given below.

Let $|A|_B$ denote the residue of A modulo B. Let X and Y be two inputs represented as $X = x_n x_{n-1} \dots x_0$ and $Y = y_n y_{n-1} \dots y_0$, where the most significant bits x_n and y_n are ones only when the inputs are 2^n and 2^n, respectively. $|X \cdot Y|_{2^n+1}$ can be represented as follows:

$$P = |X \cdot Y|_{2^n+1} = \left| \sum_{i=0}^{n} x_i 2^i \cdot \sum_{j=0}^{n} y_j 2^j \right|_{2^n+1} = \left| \sum_{i=0}^{n} \left(\sum_{j=0}^{n} p_{i,j} 2^{i+j} \right) \right|_{2^n+1}, \quad (6)$$

where $p_{i,j} = a_i$ AND b_j.

From (6), we can observe that it results in a $2n+1 \times n+1$ partial products matrix. This matrix is modified into an $n \times n$ partial products matrix based on several assumptions [38]. The conversion of the $2n+1 \times n+1$ partial products matrix into $n \times n$ partial products matrix results in a correction factor of three. The $n \times n$ partial products matrix is reduced into one sum vector and one carry vector. A part 2 of the total correction factor three is added to the $n \times n$ partial products matrix and the other part 1 is used in the final-stage addition. In the reduction of the partial products, novel compressors are used instead of full adders in each column of the carry save adder network. These selection of the compressors is based on the input width. For a particular input width, several compressor networks are possible. The best possible compressor network consists of compressors with high order such as 7:2 and 5:2 compressors. The sum and carry vectors generated by the partial

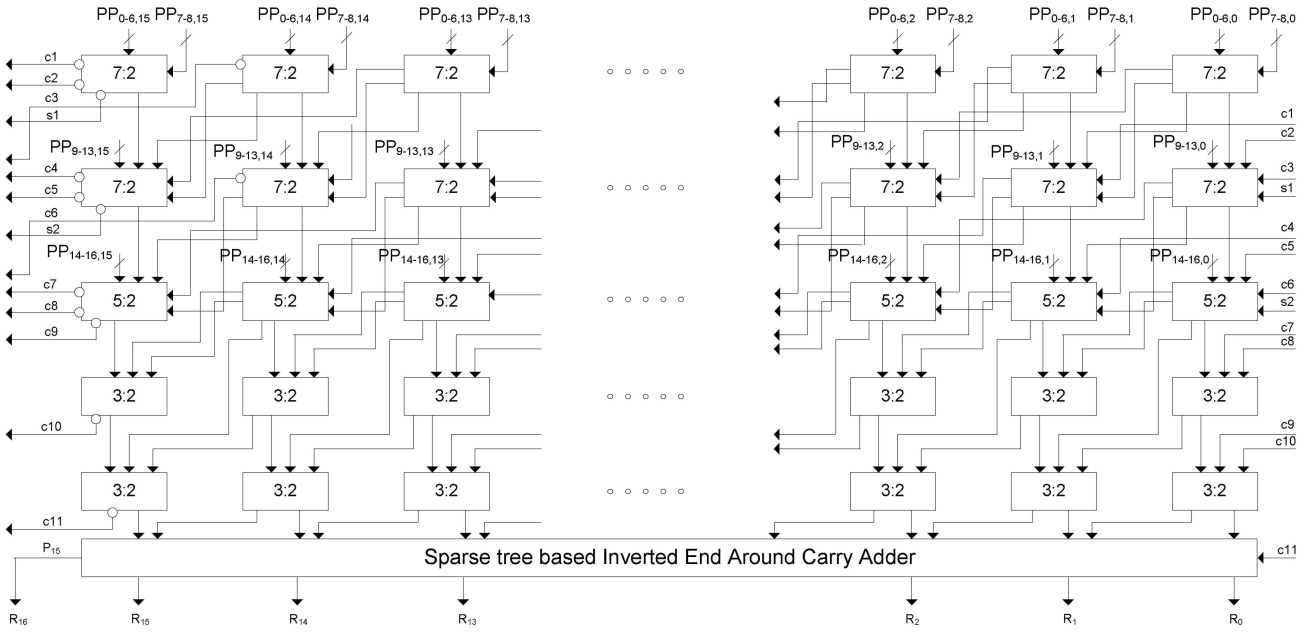


Fig. 6. Block diagram of modulo $2^{16} + 1$ multiplier suitable for hardware implementation.

products reduction module have to be added in the final-stage addition module. A part of the correction factor one left behind is used in the final-stage addition to take advantage of the following equation:

$$|Sum + Carry + 1|_{2^n+1} = |Sum + Carry + \overline{Cout}|_{2^n}. \quad (7)$$

From (7), we can observe that the inverted carryout of the addition of Sum and Carry vectors has to be fed back. Hence, adding a constant "1" to the Sum and Carry vectors results in EAC modulo 2^n addition which has regular VLSI implementation. This inverted EAC is efficiently designed using the sparse tree adder network, which has less interstage wiring and less cell density. A novel modulo $2^{16} + 1$ multiplier, which uses efficient compressors in the partial products reduction module, is shown in Fig. 6.

3 Proposed Self-checking Modulo $2^n + 1$ Multiplier Design

For any self-checking circuit to detect errors online, the circuit has to comply with a set of properties. These properties are described as follows [16]:

- Code disjointness: If each noncode input is mapped to a noncode output word, then the circuit is called code disjoint.
- Fault secure: A circuit is called fault secure if for any fault in the fault set, there is no input code word that causes the circuit to generate incorrect code word.
- Self-testing: For all faults in the fault set, there is at least one input code word that generates an output which is not a valid code word.
- Self-checking: If the circuit obeys both the self-testing and fault-secure properties, then it is called totally self-checking.

If input registers are used in the implementation, then code-disjointness ensures that the faults in the input registers are detectable. The fault-secure property requires an input fault the circuit either generates a correct output or a noncode word to detect the fault. In the self-testing circuits, each fault can be detected by at least one stimulating input vector.

In this section, the analysis of the self-checking multipliers using residue codes with various check bases is described. An analysis of the integer multipliers is also extended to the proposed implementation of the modulo $2^n + 1$ multipliers.

3.1 Self-Checking Multipliers Using Residue Codes

The self-checking two's complement multiplier is clearly studied for a fault model and appropriate check bases are designed for the same in [14]. The brief explanation of the self-checking two's complement multiplier [14] is given as follows: Let the inputs and outputs of self-checking multiplier circuit S are encoded using the error-detecting code I and O, respectively. Also, F be the fault set used for the circuit S. The fault secureness of the circuit S affected by the fault set F can be achieved by selecting the code space O such that any error in the output can be detected by a residue code check. The definition of the fault secureness with respect to a fault set F is as follows: Let $f \in F$ and S^f denote the circuit S affected by fault f. For a particular input $i \in I$, the output of $S(S^f)$ is given by $S(i)\left[S^f(i)\right]$. Then, S is fault secure for fault set $F \Longleftrightarrow$ for any input $i \in I$ and for any fault $f \in F$, $S(i) \neq S^f(i)$ implies $S^f(i) \notin O$. A block diagram of the multiplier with residue code check is shown in Fig. 7.

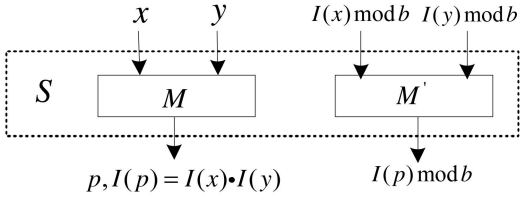


Fig. 7. Block diagram of the multiplier with residue code check.

In Fig. 7, M represents the multiplier and M' represents the modulo multiplier with residue check base $b \in N$. $I(x) \bmod b$, $I(y) \bmod b$, and $I(p) \bmod b$ represent the residues of the inputs and output with check base b. The proposed efficient implementation of the modulo generators is presented in following sections. These modulo generators use novel compressors and sparse-tree-adder-based end around carry adders. The error p^f in the multiplier output is detected if and only if:

$$\begin{aligned} &[p^f, I(p) \bmod p] \notin O, \\ \Longleftrightarrow &I(p) \bmod b \neq I(p^f) \bmod b, \\ \Longleftrightarrow &\left|I(p) - I(p^f)\right| \bmod b \neq 0. \end{aligned}$$

To simplify the analysis, let us assume a fault model for the multiplier with residue codes. For the fault set F, we assume fault caused by a primitive block in the multiplier circuit S [14]. The single fault is caused by the primitive blocks such as half adders or full adders. This fault results in change in the functional behavior of the primitive blocks thus producing an erroneous output. Let $E(M)$ denote the set of the absolute errors in the output of the multiplier circuit M caused by a single fault in the primitive blocks. Then, $E(M)$ can be represented as follows:

$$\begin{aligned} E(M) = \{&|I(p) - I(p^f)\| \\ &f \text{ fault caused by a primitive block in } M, \\ &p = M(x, y), \\ &p^f = M^f(x, y)\}/O. \end{aligned}$$

The multiplier circuit S is fault secure to the fault set $F \Longleftrightarrow$:

$$\forall e \in E(M) : e \bmod b \neq 0. \tag{8}$$

Hence, to achieve low hardware costs, the minimum check base value is selected which satisfies the above observation. The selection of the check bases in the residue-code-based circuits is an important task, since the overall checking circuit complexity and speed are mainly determined by it. In selecting the check bases, the absolute error set $E(M)$ caused by the single faults has to be characterized first. For different sets of faults in various primitive blocks, the circuit functions differently. In this paper, the fault model consists of faults in a single primitive block.

The multiplier circuit S design so far is only fault secure, i.e., only testable fault from the fault set F can be detected. In some cases, untestable faults from the fault set may affect the fault-secure property of the circuit. Hence, to achieve the fault secureness for all the faults, the multiplier circuit has to be self-testing too. A brief explanation of the self-testing property [14] is described in detail as follows:

Let S be the self-checking multiplier circuit, I be the input code space, and O be the output code space. Let the set of inputs subjected to the circuit in the fault free case be N, these inputs are given to the circuit as normal inputs $N \subset I$. Then, the circuit S is called self-testing when:

- For a fault $f \in F$, there exists an $i \in N$ such that $S^f(i) \notin O$.
- The circuit S is self-testing for fault set $F \Longleftrightarrow S$ is self-testing for every fault from the set F.

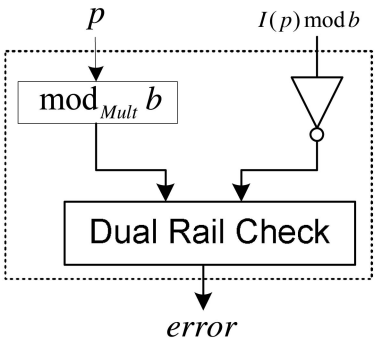


Fig. 8. Block diagram of the residue code checker.

For the multiplier circuit S with residue code check, in most applications, $N = I$. Hence, the input set for both M and M' can be subjected to arbitrary values. Thus, if M and M' are realized without internal redundancy, then the fault secureness implies that the circuit S is self-testable. A basic configuration of the multiplier with residue code checker is shown in Fig. 8. In Fig. 8, $\bmod_{Mult} b$ computes the residue of the multiplier output, i.e., $I(p)$ mod b. Dual-Rail Check compares the output from $\bmod_{Mult} b$ and output of the inverter and asserts the error signal if the two inputs are not complementary to each other. With the above residue and dual-rail checkers, the self-testing property may not be always achieved. To guarantee the self-testing property of the multiplier circuit, the structure of the checkers has to be properly designed.

The check base selection of the residue codes depends on the faults in the fault set F and the resulting absolute error in the output vector of the multiplier. Hence, before selecting the check base, the $E(M)$ set has to be fixed. As only one fault in a single primitive block such as half adder or full adder is assumed, corresponding $E(M)$ can be computed and proper check base is selected using (8). For the two's complement multiplier studied in [14], the check base b is fixed as three for most of the input cases. This result is obtained by making a particular assumption; otherwise, check base 7 is used. Suppose for an n-bit multiplier, the partial products reduction module is designed using half adders and full adders. A single fault in any one of these primitive blocks may result in an error in the sum or carry outputs. The fault causes an error set $E(M)$ which is given by

$$E(M) \subset \left\{\alpha 2^i | \alpha \in [1:3], i \in [0:2n-2]\right\} \cup \left\{5.2^{2n-3}\right\}. \tag{9}$$

The check base b achieves the fault secureness for the given multiplier if:

$$b \in N \backslash \{\gamma 2^i | \gamma \in [1:5], i \in \{0, N\}\}, \tag{10}$$

where N denotes the set of all natural numbers.

Hence, the smallest check base to satisfy (10) and achieve the fault secureness for the given multiplier is seven. Since the modulo 3 generator is widely practiced, the check base 3 is used based on the following assumption. In the partial products reduction module, half adders and full adders are assumed such that error values of the form ± 3 do not occur. Assuming this error analysis, it can be shown that under the

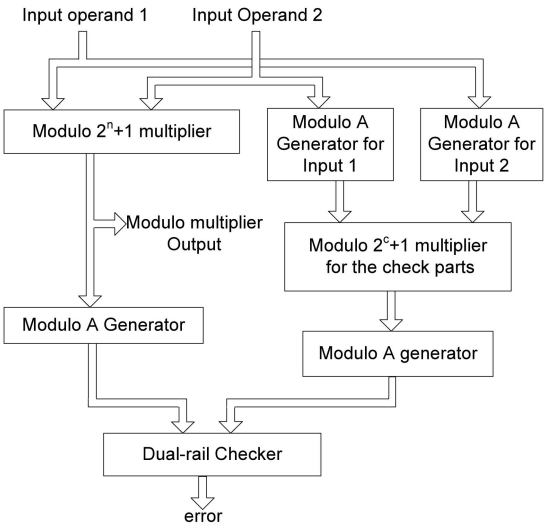


Fig. 9. Block diagram of the self-checking modulo $2^n + 1$ multiplier.

restricted fault model, the error set $E_{re}(M)$ can be represented as

$$E_{re}(M) \subset \{2^i | i \in [0 : 2n-1]\} \cup \{3.2^{2n-2}\}. \quad (11)$$

From (11), there exists only one error which cannot be detected by modulo 3 check base. This error value occurs for only one input combination which is $I(x) = I(y) = -2^{n-1}$ and a fault on the outputs of the primitive block with weight 2^{2n-2}. This case is highly improbable, and hence, is neglected.

3.2 Self-Checking Modulo $2^n + 1$ Multipliers Using Residue Codes

In this section, a fault-secure implementation of the modulo $2^n + 1$ multiplier using residue codes is given. The fault model for this self-checking multiplier includes faults affecting a single gate. And consequently, the same fault may propagate through the subsequent gates and generate errors at the multiplier outputs. Hence, to achieve fault secureness, these errors must be detected by the residue codes [18]. As shown in Fig. 3, the fault model includes any fault affecting the gates that generate the outputs sum and carryouts.

From the implementation of the modulo $2^n + 1$ multiplier [3], it consists of partial products generation module, partial products reduction module, and final-stage addition module. The partial products generation consists of basic logic gates and the partial products reduction module is implemented using compressors of different order (3:2, 4:2, and 5:2 compressors, etc.). The final-stage addition module is designed using sparse-tree-based inverted end Around carry adder. For ordinary integer multipliers, the error in the arithmetic value of the output caused by the faults is well studied in the literature [20]. In residue-code-based self-checking multipliers [14], [18], [20], [22], to detect an error, the arithmetic difference should not be divisible by the check base of the residue codes.

The block diagram of the self-checking modulo $2^n + 1$ multiplier based on residue arithmetic codes [18] is shown in Fig. 9 (i.e., A in the modulo A generator block is either $2^c - 1$ or $2^c + 1$). From the figure, we can observe that the self-checking modulo $2^n + 1$ multiplier consists of a modulo multiplier, modulo generators for the input operands followed by a modulo multiplier, and modulo generators (with check bases of the form $2^c - 1$ or $2^c + 1$) for each of the modulo multiplier outputs. In the final stage, an arithmetic code checker to check the output of the modulo multiplier is against its check part. In this case, a dual-rail checker is used.

As shown in Fig. 9, the modulo generator calculates input modulo $2^c - 1$ or $2^c + 1$. The check base selection for arithmetic circuits such as adders and multipliers is presented in the literature [14], [20]. The hardware implementation and structure of the modulo multiplier is similar to integer multipliers [3].

In the array multipliers implementation reported in [31], the partial products reduction module consists of full adders and half adders which generates Sum and Carry outputs. A fault on the set of the gates that generate these Sum and Carry signals cause an output error [18], [20]. An error on the sum signal gives an arithmetic value of $\pm 2^i$ (where i is the weight of the error signal), similarly an error on the Carry signal gives an arithmetic value of $\pm 2 \cdot 2^i$. Hence, errors produced on both of these signals give an arithmetic value of $\pm 3 \cdot 2^i$. To detect the error caused by these faults, the check base of the residue codes should be selected such that the arithmetic difference should not be divisible by the check base. The final-stage adder of the array multiplier is generally implemented using parallel prefix adders. Various algorithms are proposed to select the check bases for these fast parallel prefix adders. Unlike the ripple carry adders, the carry computation problem is logarithmic rather than linear. Hence, the error propagation is also different and causes various output errors. A brief description on the check base selection discussed in [18], [20] is presented below. The arithmetic value of the errors in the outputs is determined based on a couple of facts.

- Faults on the signals with divergent degree higher than 1 result in errors with arithmetic value $\pm 2^i$.
- Consider faults on a random signal w_i which can be propagated to the carry c_i. The error may propagate to the other carry signals c_j $(j > i)$ which structurally depend on c_i subsequently on w_i. In the actual implementation, not all the carries c_j $(j > i)$ structurally depend on w_i. Hence, two kinds of errors are possible, in the first case, all the carries c_j $(j > i)$ may depend on c_i resulting in an error with final arithmetic value $\pm 2^{i+1}$. In the second case, only a subset of the carries c_j $(j > i)$ may structurally depend on c_i producing an output error

$$\begin{aligned} \pm \big[& a_0(2^{i+1} \pm 2^{i+2} \cdots \pm 2^{i+k}) \\ & + a_1(2^{i1+1} \pm 2^{i1+2} \cdots \pm 2^{i1+k1}) \\ & + a_2(2^{i2+1} \pm 2^{i2+2} \cdots \pm 2^{i2+k2}) \cdots \\ & + a_m(2^{im+1} \pm 2^{im+2} \cdots \pm 2^{im+km}) \big], \end{aligned}$$

  where $a_0, a_1 \ldots a_m \in [0, 1]$ and $im + km < n$, and n is the width of the operands used in the adder.

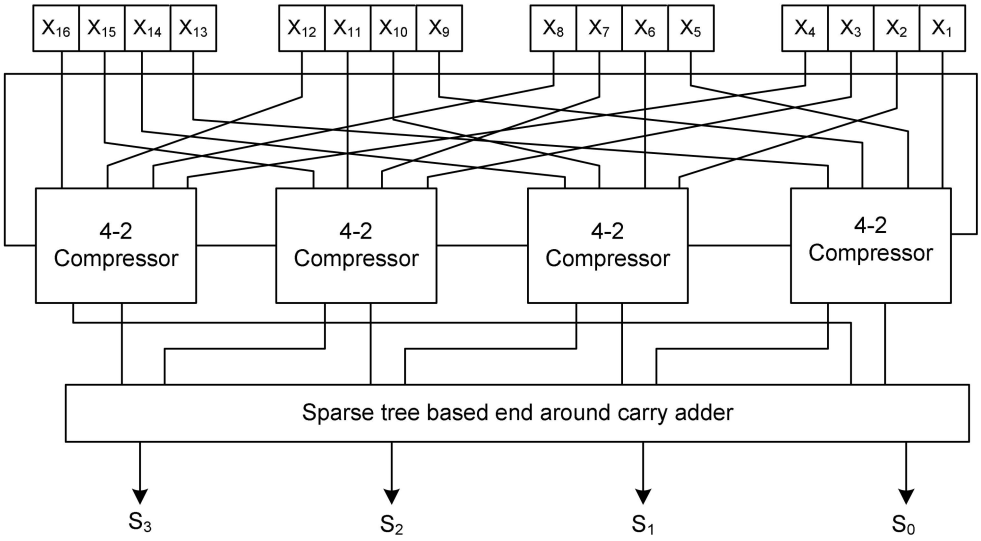


Fig. 10. Modulo generator with check base $2^4 - 1$ for input $\mathrm{width} = 16$.

To achieve the fault secureness of the self-checking modulo $2^n + 1$ multiplier, the resulting arithmetic value of the output errors caused by the faults must not be divisible by the check base. Hence, the smallest odd integer is chosen which does not divide the arithmetic value of the errors in the output. The check bases are that best suit for this operation are of the type $(2^c - 1/2^c + 1, c \in N)$ and these check bases result in efficient residue code computation [20], [14].

In the modulo $2^n + 1$ multipliers, the partial products reduction module is designed using compressors which are similar to full adders in operation. A fault in set of the gates of the compressors generate an error with arithmetic value $\pm K \cdot 2^i$ (K is a constant). The final-stage adder is designed using sparse-tree-based inverted end around carry adder. The operation of the sparse tree adder is same as parallel prefix adders, except the carries are computed at every fourth or eighth bit. Hence, the error analysis gives same output error as proven in [20]. Hence, to design efficient modulo generators, the check base of the form $(2^c - 1/2^c + 1, c \in N)$ is chosen. Efficient implementation of the modulo generators with check bases $2^4 - 1$ and $2^4 + 1$ for an input operand of width 16 are shown in Figs. 10 and 11. These modulo generators use novel sparse-tree-based end around carry adder and inverted end around carry adder, respectively. In the novel designs of the modulo generators, full adders are replaced by the efficient compressors.

Thus, the proposed self-checking modulo $2^n + 1$ multiplier based on residue codes efficiently detects all errors caused by the faults on a single gate at a time. The efficient

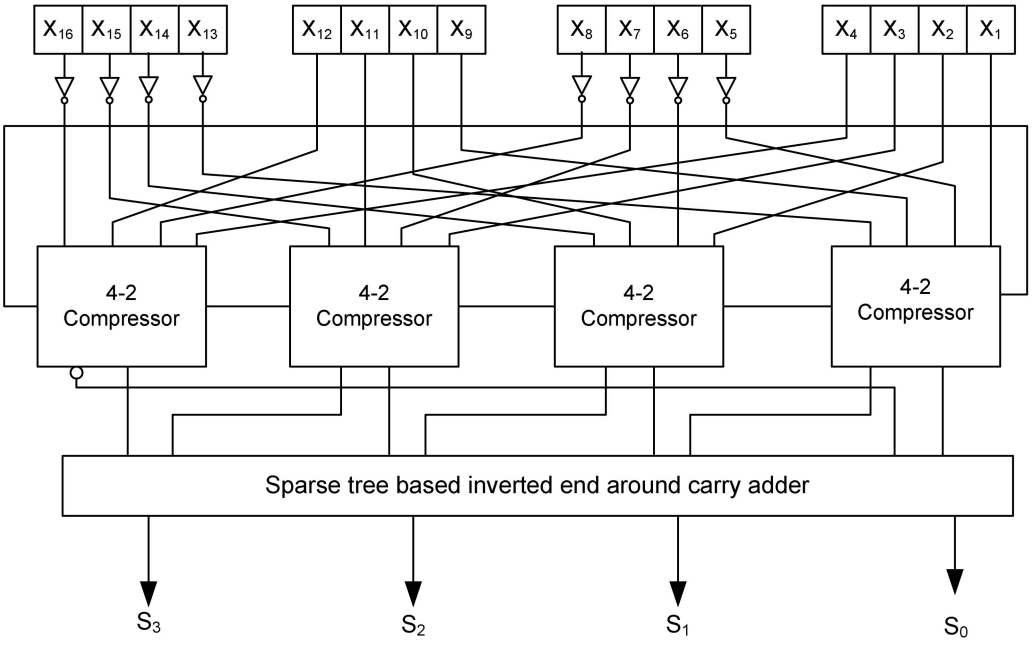


Fig. 11. Modulo generator with check base $2^4 + 1$ for input $\mathrm{width} = 16$.

TABLE 1
Area and Delay Comparison of Modulo $2^n + 1$ Multipliers with and without Self-Checking Property Using Unit-Gate Model Analysis

n	A_{MW}	A_{MWS}	% area overheard	T_{MW}	T_{MWS}	% performance penalty
8	553	4454	43	30	33	10
16	1968	2676	36	52	56	7.7
32	7825	10251	31	94	98	4.25
64	25637	30508	19	168	173	2.98

use of the compressors in the modulo generators and modulo multipliers result in good savings in terms of area overhead and delay.

4 Parametric Comparison

The efficient self-checking modulo $2^n + 1$ multiplier is obtained from the efficient use of the novel compressors in the modulo multipliers and modulo generators. In this section, the self-checking modulo $2^n + 1$ multipliers are compared against the modulo $2^n + 1$ multipliers without self-checking property. The comparisons are carried out using the unit-gate model proposed by Tyagi [34] and also experimental results are compared. The hardware overhead in the proposed implementation of the modulo $2^n + 1$ multiplier is caused by the modulo generators, dual-rail checker the modulo multiplier which is used to generate the check part for the dual-rail checker to check against the actual result of the modulo $2^n + 1$ multiplier. The performance penalty of the multiplier is caused by the dual-rail checker and the modulo generators. If the combined delay of the modulo generators and dual-rail checker are more than the delay of the modulo $2^n + 1$ multiplier, this pays penalty for the performance of the multiplier. For different values of the input operands of the modulo $2^n + 1$ multiplier, the modulo generators have different check bases to achieve the full fault secureness. The residue code check bases of the form $2^c - 1$ and $2^c + 1$ for different values of the input operands are selected and corresponding modulo generators are designed.

4.1 Unit-Gate Model Analysis

The modulo multipliers and the modulo generators contribute largely to the overall area and delay of the multiplier. In the unit-gate model presented by Tyagi [34], each 2-input monotonic gate is considered as a single-gate equivalent for both the area and delay comparisons, and the 2-input XOR gate and 2:1 MUX are considered as two-gate equivalents (area and delay). The area and delay terms of the modulo multiplier with and without self-checking property are shown below:

$$\begin{aligned} A_{MWS} &= A_{MG} + A_{Mm} + A_{DRC}, \\ T_{MWS} &= T_{MG} + T_{Mm} + T_{DRC}, \\ A_{MW} &= A_{MM}, \\ T_{MW} &= T_{MM}. \end{aligned}$$

In the above equations, A_{MWS} and T_{MWS} denote, respectively, the area and delay of the modulo multiplier with self-checking property. A_{MWS} and T_{MWS} are obtained by summing the areas and delays of the modulo generators (A_{MG}, T_{MG}), modulo multipliers (A_{Mm}, T_{Mm}), and dual-rail checker (A_{DRC}, T_{DRC}). A_{MW} and T_{MW} are the area and delay of the modulo multiplier without self-checking property, they are nothing but the area and delay of the ordinary modulo $2^n + 1$ multiplier denoted by A_{MM} and T_{MM}, respectively. The unit-gate areas and delays of these multipliers are computed and tabulated in Table 1.

4.2 Experimental Results

Even though the unit-gate model gives delay and area comparisons in terms of gate counts, the standard-cell-based implementation of the proposed compressor-based multiplier gives much more accurate delay and area estimations. The proposed self-checking modulo multipliers for various values of input length are specified in Verilog Hardware Description Language (HDL). The multiplier descriptions are mapped on a 0.18μm CMOS standard cell library using Leonardo Spectrum synthesis tool from Mentor Graphics. The design is optimized for high-speed performance. Netlists generated from synthesis tool are passed on to standard route and place tool, and the layouts are iteratively generated to get the circuits with minimum area.

In Table 2, A_1 and A_2 represent the area of the modulo $2^n + 1$ multiplier without the self-checking property and with the self-checking property, respectively. Similarly, T_1 and T_2 represent the delay of the modulo $2^n + 1$ multiplier without the self-checking property and with self-checking property, respectively.

5 Conclusions

In this paper, a new efficient self-checking modulo $2^n + 1$ multiplier based on residue codes is proposed and validated. In the proposed implementation, the self-checking modulo multiplier consists of modulo generators with check bases of the form $2^c - 1$ or $2^c + 1$ ($c \in N$), modulo multipliers, and self-checking dual-rail checkers. All the modulo components such as modulo generators and modulo multipliers are efficiently designed using novel compressors. The final-stage addition modules of the modulo multipliers and modulo generators are also efficiently designed using sparse-tree-based inverted end around carry adders. The self-checking multiplier is secured against faults affecting a single gate at a time and produce an error at the gate output, which may propagate through

TABLE 2
Experimental Results Showing the Area and Delay Comparison of Modulo $2^n + 1$ Multipliers with and without Self-Checking Property

n	A_1 (μm^2)	A_2 (μm^2)	% area overheard	T_1	T_2	% performance penalty
8	3072	4454	45	1.982	2.121	7
16	10933	14540	33	4.216	4.427	5
32	43472	56078	29	8.542	8.884	4
64	160847	194624	21	15.315	15.621	2

the subsequent gates and generate an error at the output of the modulo multiplier. These self-checking modulo multipliers are analyzed using unit-gate model and compared with the modulo multipliers without self-checking property. These models are designed for various values of the input length and simulated to generate the experimental results. The results show that the proposed self-checking multiplier results in 20 to 45 percent area overhead and two to seven percent performance penalty for $n = 64$ to eight.

ACKNOWLEDGMENTS

This work was supported by research fund of Ulsan College.

REFERENCES

[1] *Modern Applications of Residue Number System Arithmetic to Digital Signal Processing*, M.A. Soderstrand, W.K. Jenkins, G.A. Jullien, and F.J. Taylor, eds. IEEE Press, 1986.

[2] P.E. Beckmann and B.R. Musicus, "Fast Fault-Tolerant Digital Convolution Using a Polynomial Residue Number System," *IEEE Trans. Signal Processing*, vol. 41, no. 7, pp. 2300-2313, July 1993.

[3] R. Modugu, N. Park, and M. Choi, "A Fast Low-Power Modulo $2^n + 1$ Multiplier Design," *Proc. 2009 IEEE Int'l Instrumentation and Measurement Technology Conf.*, pp. 951-956, May 2009.

[4] R. Modugu and M. Choi, "Efficient High Speed and Low-Power Modulo $2^n + 1$ Multiplier," internal technical report, Missouri Univ. of Science and Technology.

[5] R. Modugu, N. Park, Y.-B. Kim, and M. Choi, "Efficient On-Line Self-Checking Modulo $2^n + 1$ Multiplier Design," *Proc. 24th IEEE Int'l Symp. Defect and Fault Tolerance in VLSI Systems*, Oct. 2009.

[6] R. Zimmermann and W. Fichtner, "Low-Power Logic Styles: CMOS versus Pass-Transistor Logic," *IEEE J. Solid-State Circuits*, vol. 32, no. 7, pp. 1079-1090, July 1997.

[7] S. Veeramachaneni, L. Avinash, R.M. Rajashekhar, and M.B. Srinivas, "Efficient Modulo $(2^k \pm 1)$ Binary to Residue Converters," *Proc. Sixth Int'l Workshop System-on-Chip for Real-Time Applications*, pp. 195-200, Dec. 2006.

[8] C-H. Chang, J. Gu, and M. Zhang, "Ultra Low-Voltage Lowpower CMOS 4-2 and 5-2 Compressors for Fast Arithmetic Circuits," *IEEE J. Circuits and Systems I*, vol. 51, no. 10, pp. 1985-1997, Oct. 2004.

[9] M. Rouholamini, O. Kavehie, A.-P. Mirbaha, S.J. Jasbi, and K. Navi, "A New Design for 7:2 Compressors," *Proc. IEEE/ACS Int'l Conf. Computer Systems and Applications 2007 (AICCSA '07)*, pp. 474-478, May 2007.

[10] A. Curiger et al., "VINCI: VLSI Implementation of the New Secret-keyBlock Cipher IDEA," *Proc. Custom Integrated Circuits Conf.*, May 1993.

[11] R. Zimmermann, A. Curiger, H. Bonnenberg, H. Kaeslin, N. Felber, and W. Fichtner, "A 177 Mb/s VLSI Implementation of the International Data Encryption Algorithm," *IEEE J. Solid-State Circuits*, vol. 29, no. 3, pp. 303-307, Mar. 1994.

[12] L.M. Leibowitz, "A Simplified Binary Arithmetic for the Fermat Number Transform," *IEEE Trans. Acoustics Speech and Signal Processing*, vol. ASSP-24, no. 5, pp. 356-359, Oct. 1976.

[13] D.P. Vasudevan and P.K. Lala, "A Technique for Modular Design of Self-Checking Carry-Select Adder," *Proc. 20th IEEE Int'l Symp. Defect and Fault Tolerance in VLSI Systems (DFT '05)*, pp. 325-333, Oct. 2005.

[14] U. Sparmann and S.M. Reddy, "On the Effectiveness of Residue Code Checking for Parallel Two's Complement Multipliers," *Proc. 24th Int'l Symp. Fault-Tolerant Computing (FTCS-24)*, pp. 219-228, June 1994.

[15] D. Marienfeld, E.S. Sogomonyan, V. Ocheretnij, and M. Gossel, "New Self-Checking Output-Duplicated Booth Multiplier with High Fault Coverage for Soft Errors," *Proc. 14th Asian Test Symp. '05*, pp. 76-81, Dec. 2005.

[16] M. Hunger and D. Marienfeld, "New Self-Checking Booth Multipliers," *Int'l J. Applied Math. and Computer Science*, vol. 18, no. 3, pp. 319-328, 2008.

[17] M. Goessel and F. Graf, *Error Detection Circuits*. McGraw-Hill, 1993.

[18] I.A. Noufal and M. Nicolaidis, "A CAD Framework for Generating Self-Checking Multipliers Based on Residue Codes," *Proc. Design, Automation and Test in Europe Conf. and Exhibition '99*, pp. 122-129, 1999.

[19] O. Garcia and T. Rao, "On the Method of Checking Logical Operations," *Proc. Second Ann. Princeton Conf. Information Science System*, pp. 89-95, 1968.

[20] U. Sparmann, "On the Check Base Selection Problem for Fast Adders," *Proc. IEEE 11th Ann. VLSI Test Symp.*, pp. 62-65, Apr. 1993.

[21] F.F. Sellers, M.Y. Hsiao, and L.W. Bearnson, *Error Detecting Logic for Digital Computers*. McGraw-Hill, 1968.

[22] W. Peterson, "On Checking an Adder," *IBM J. Research and Development*, vol. 2, pp. 166-168, Apr. 1958.

[23] E. Fujiwara and K. Haruta, "Fault-Tolerant Arithmetic Logic Unit Using Parity Based Codes," *Trans. ECE of Japan*, vol. E64, pp. 653-660, Oct. 1981.

[24] D. Marienfeld, E.S. Sogomonyan, V. Ocheretnij, and M. Gossel, "A New Self-Checking Multiplier by Use of a Code Disjoint Sum-Bit Duplicated Adder," *Proc. Ninth IEEE European Test Symp. (ETS '04)*, pp. 30-35, May 2004.

[25] W.H. Debany, A.R. Macera, D.E. Daskiewich, M.J. Gorniak, K.A. Kwiat, and H.B. Dussault, "Effective Concurrent Test for a Parallel-Input Multiplier Using Modulo 3," *Proc. IEEE VLSI Test Symp. "10th Anniversary Design, Test and Application: ASICs and Systems-on-a-Chip,"* pp. 280-285, Apr. 1992.

[26] T.J. Slegel and R.J. Veracca, "Design and Performance of the IBM Enterprise System/9000 Type 9121 Vector Facility," *IBM J. Research and Development*, vol. 35, pp. 367-381, May 1991.

[27] M. Nicolaidis, "Efficient Implementations of Self-Checking Adders and ALUs," *Proc. 23rd Int'l Symp. Fault-Tolerant Computing (FTCS-23)*, pp. 586-595, June 1993.

[28] S. Kundu and S.M. Reddy, "Embedded Totally Self-Checking Checkers: A Practical Design," *IEEE Design and Test of Computers*, vol. 7, no. 4, pp. 5-12, Aug. 1990.

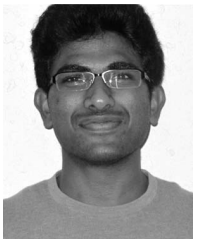
[29] B. Kiran Kumar and P.K. Lala, "On-Line Detection of Faults in Carry-Select Adders," *Proc. Int'l Test Conf. 2003 (ITC '03)*, 2003.

[30] D.P. Vasudevan, P.K. Lala, and J.P. Parkerson, "Self-Checking Carry-Select Adder Design Based on Two-Rail Encoding," *IEEE Trans. Circuits and Systems I: Regular Papers*, vol. 54, no. 12, pp. 2696-2705, Dec. 2007.

[31] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford Univ. Press, 2000.

[32] S. Mathew, M. Anders, R.K. Krishnamurthy, and S. Borkar, "A 4-GHz 130-nm Address Generation Unit with 32-Bit Sparse-Tree Adder Core," *IEEE J. Solid-State Circuits,* vol. 38, no. 5, pp. 689-695, May 2003.
[33] J. Grad and J.E. Stine, "A Multi-Mode Low-Energy Binary Adder," *Proc. 40th Asilomar Conf. Signals, Systems and Computers,* pp. 2065-2068, Oct./Nov. 2006.
[34] A. Tyagi, "A Reduced-Area Scheme for Carry-Select Adders," *IEEE Trans. Computers,* vol. 42, no. 10, pp. 1163-1170, Oct. 1993.
[35] P. Shivakumar, S.W. Keckler, M. Kistler, D. Burger, and L. Alvisi, "Modeling the Effect of Technology Trends on the Soft Error Rate of Combinatorial Logic," *Proc. Int'l Conf. Dependable Systems and Networks,* pp. 389-398, 2002.
[36] P. Kogge and H.S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Trans. Computers,* vol. 22, no. 8, pp. 786-793, Aug. 1973.
[37] H.T. Vergos, C. Efstathiou, and D. Nikolos, "Diminished-One Modulo 2n+1 Adder Design," *IEEE Trans. Computers,* vol. 51, no. 12, pp. 1389-1399, Dec. 2002.
[38] H.T. Vergos and C. Efstathiou, "Design of Efficient Modulo $2^n + 1$ Multipliers," *IET Computers and Digital Techniques,* vol. 1, no. 1, pp. 49-57, 2007.

Wonhak Hong received the BS, MS, and PhD degrees in electronics and electrical engineering from Kyungpook National University, Daegu, South Korea, in 1989, 1991, and 1996, respectively. During his graduate study, he worked on image processing technologies and developed various image and video compression applications, especially for MPEG-4. In 1995, he joined as a full-time lecturer at Ulsan College, South Korea, where he is currently a professor in the Department of Electrical and Electronic Engineering. He was also a visiting professor in the Department of Electrical and Computer Engineering at Missouri University of Science and Technology in 2009. His current research interests include computer architecture, image segmentation technologies, and novel applications of object extraction to image recognition and computer vision. He is a member of the Institute of Electronics Engineers of Korea (IEEK).



Rajashekhar Modugu received the bachelor's degree with honors from the International Institute of Information Technology—Hyderabad (IIIT-H) in 2008. He is currently working toward the master's degree at Missouri University of Science and Technology (Missouri S&T). He worked as a research assistant at Missouri S&T during 2008-2009 and as a teaching assistant at IIIT-H during 2007-2008. He has been working at Advanced Micro Devices (AMD), Inc., as a co-op engineer since January 2010. His areas of interest include computer arithmetic, VLSI design, cryptography, and fault-tolerant design. He received Pratibha Scholorship from the government of Andhra Pradesh, India, throughout his undergraduation. He has published several articles during his bachelor's and master's studies.

Minsu Choi (M'02-SM'08) received the BS, MS, and PhD degrees in computer science from Oklahoma State University in 1995, 1998, and 2002, respectively. He is currently an associate professor of electrical and computer engineering at Missouri University of Science and Technology (Missouri S&T). His research mainly focuses on Computer Architecture and VLSI, Nanoelectronics, Embedded Systems, Fault Tolerance, Testing, Quality Assurance, Reliability Modeling and Analysis, Configurable Computing, Parallel and Distributed Systems, and Dependable Instrumentation and Measurement. He has published more than 73 articles in archival journals and referred proceedings. He has won two outstanding teaching awards at Missouri S&T in 2008 and 2009. He is a senior member of the IEEE and a member of Golden Key National Honor Society and Sigma Xi.