Area and Power Efficient Self-Checking Modulo 2^n+1 Multiplier

B.Mounika
M.Tech,Student,
DECS, Dept. of ECE
Vardhaman College of Engineering,
Hyderabad, INDIA

Sangeeta Singh
Associate Professor,
Dept. of ECE
Vardhaman College of Engineering,
Hyderabad, INDIA

ABSTRACT

Modulo 2^n+1 multiplier is the key block in the circuit implementation of cryptographic algorithm such as IDEA and also widely used in the area of data security applications such as residue arithmetic, digital signal processing, and data encryption that demands low-power, area and high-speed operation. In this paper, a new circuit implementation of an area and power efficient self-checking modulo 2^n+1 multiplier based on residue codes are proposed. Modulo 2^n+1 multiplier has the three major stages: partial product generation stage, partial product reduction stage, and the final adder stage. The last two stages determine the speed and power of the entire circuit. An efficient self-checking modulo 2^n + 1 multiplier based on residue codes are proposed to detect errors online at each single gate during the data transmission and produce an error at the gate output, which may propagate through the subsequent gates and generate an error at the output of the modulo multiplier. The proposed self-checking modulo multipliers for various values of input are specified in Verilog Hardware Description Language (HDL), simulated by using XILINX ISE and synthesized using cadence RTL encounter tool.

General Terms

IDEA algorithm , Online self-checking, Security.

Keywords

Compressors, Modulo 2^n+1 Multiplier, Self Checking multiplier, Sparse Tree Based Inverted End around Carry.

1. INTRODUCTION

Now-a-days, the information safety is given utmost priority. The security requirements and confidentiality of data transmission through channels, is becoming more and more important due to rapid increase in popularity of internet and wireless communication nodes, which makes cryptography play a vital role in this information age. So various cryptographic algorithms have been studied and carried out to ensure security for data transmission. In cryptography, a symmetric key algorithm such as International Data Encryption Algorithm (IDEA) is one of the most reliable cryptographic algorithms used for secure data transmission. Modulo 2^n +1 multiplier is one of key block in the circuit implementation of IDEA. Hence, an area and power efficient self checking modulo 2^n +1multiplier is proposed to protect data against errors in security applications.

The transient and permanent faults occurred during the transmission of data can be detected online by using self-checking circuits and systems. So self-checking circuits are designed using various coding schemes such as arithmetic codes to check the functionality of the circuits. In recent years, Residue number system (RNS) is widely used in modulo arithmetic applications, so in many of the industrial applications self-checking arithmetic circuits are designed using residue codes to detect faults online as soon as they occur during data transmission.RNS has become so popular because computations are performed so efficiently that is the, computations of large integers are performed parallel by using set of small integers so that overall performance of the system increases.

In past decades, different architectures of modulo 2^n +1 multiplier are proposed. According to Cruiger's[3], three different multiplication architectures are proposed: The first architecture is designed by using a (n+1) × (n+1) bits multiplier followed by modulo adders to correct errors caused by carry. The second architecture is realized by using modulo 2^n +1 adder, which consists of a carry-save adder and a final carry-select addition unit to reduce design complexity [2]. The third architecture, is realized by modifying the second architecture by reducing the circuit area significantly and by introducing a bit-pair recoding scheme in the carry-save adder block [3] operating speed is increased. The last two architectures are suitable for full-custom design [2], because they increase design challenges such as layout and fabrication complexity. Later, Zimmermann [4], implemented a new high speed, low power modulo 2^n +1 multiplier which has a three major parts: partial products generation stage, partial product reduction stage, and the final addition stage.

In this paper, area and power efficient self checking modulo 2^n +1 multiplier based on residue codes are implemented. Section 2 introduces 3 stages of modulo 2^n +1 multiplier and different type of compressors used in modulo 2^n +1 multiplier [1]. Section 3 discusses the implementation of the self-checking modulo 2^n +1 multiplier, modulo generators and dual rail checker. Experimental results showing the simulation results of self-checking modulo 2^n+1 multiplier circuits are given in Section 4.

2. MODULO 2^n+1 MULTIPLIER

Modulo 2^n +1 multiplier is widely used in many data security applications such as digital signal processors and cryptographic applications .Modulo 2^n +1 multiplier today has

characteristics of high speed, low power and small area which is suitable for VLSI implementation. Modulo 2^n+1 multiplier consists of three major parts: partial products generation block, partial product reduction block, and the final addition stage. The last two stages determine the speed and power of the whole circuit. The enhancements can be done in the partial product reduction stage and the final adder stage to achieve higher speed and lower power as these two stages are the critical path of the multiplier.

2.1 Algorithm for Partial Product Generation Stage

Consider A and B are two inputs represented as $A = a_n a_{n-1} a_{n-2} \ldots a_1 a_0$ and $B = b_n b_{n-1} b_{n-2} \ldots b_1 b_0$, then A·B modulo 2^n+1 is represented as follow [1]:

$$|A \times B|_{2^n+1} = \left| \sum_{i=0}^{n} a_i 2^i \times \sum_{i=0}^{n} b_j 2^j \right|_{2^n+1}$$

$$= \left| \sum_{i=0}^{n} \left(\sum_{j=0}^{n} p_{i,j} 2^{i+j} \right) \right|_{2^n+1} \quad \ldots\ldots (1)$$

Where $p_{i,j} = a_i b_j$

Let us consider two 8 bit inputs, A=119=001110111, B=87=001010111. The initial output of the partial product generation stage is as shown in Figure1

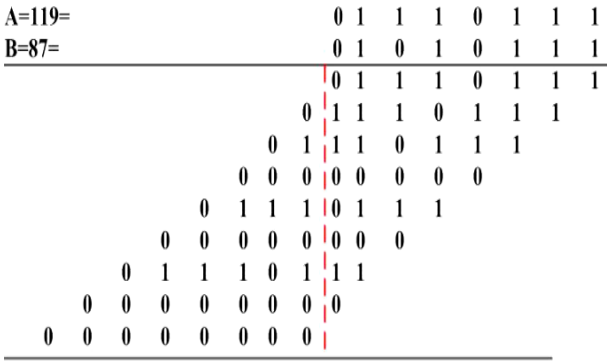


Figure 1: The initial output of the partial product generation stage

The left half (left of the dashed line) of the initial partial products shown in Figure1 need to be inverted and repositioned as shown in Figure2 to obtain the final n×n partial product matrix.

0	1	1	1	0	1	1	1
1	1	1	0	1	1	1	1
1	1	0	1	1	1	1	0
0	0	0	0	0	1	1	1
0	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1
1	1	1	0	0	0	1	0
0	1	1	1	1	1	1	1

Figure 2: The n×n partial product matrix

Due to the reposition operation, the final n × n partial product matrix results in a correction factor of:

$$COR_1 = 2^n[(2^1-1)+(2^2-1)+\ldots.+(2^{n-1}-1)]$$

$$= 2^n(2^n - n - 1) \quad \ldots\ldots. (2)$$

2.2 Algorithm for Partial Product Reduction Stage

In this stage, the final n×n partial product matrix is reduced into a sum vector and a carry vector using compressors. The carryout bit of each level of the compressor has to be fed back as the carry-in bit of the next subsequent level, this leads to a correction factor of:

$$COR_2 = |2^n(n-1)|_{2^n+1} \quad \ldots\ldots. (3)$$

For a n-bit modulo 2^n+1 multiplier, the final correction factor can be calculated as:

$$COR = COR_1 + COR_2 = |2^n(2^n-2)|_{2^n+1} = 3 \quad \ldots\ldots (4)$$

In partial product reduction stage, a correction factor of '2' is added and remaining correction factor of '1' is added to the final adder stage due to the inverted carry feedback.

0	1	1	1	0	1	1	1
1	1	1	0	1	1	1	1
1	1	0	1	1	1	1	0
0	0	0	0	0	1	1	1
0	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1
1	1	1	0	0	0	1	0
0	1	1	1	1	1	1	1
0	0	0	0	0	0	1	0

Figure 3: The final partial product matrix with the correction factor

In this partial product reduction stage compressor compresses the partial product matrix shown in Figure 3 to a final sum vector and a carry vector, as shown in Figure 4.

Figure 4: Initial output of partial product reduction stage

The nth bit of carry vector is inverted and repositioned as shown in Figure 5.

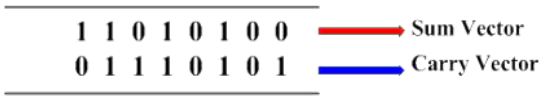


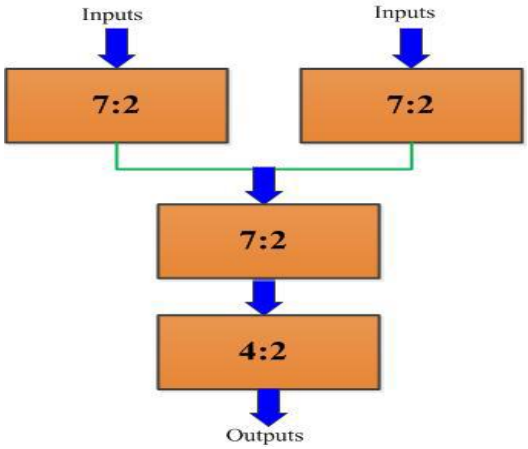
Figure 5: The output of the partial product reduction stage after repositioning

The final sum vector and carry vector after repositioning are modulo 2^n+1 added in final addition stage.

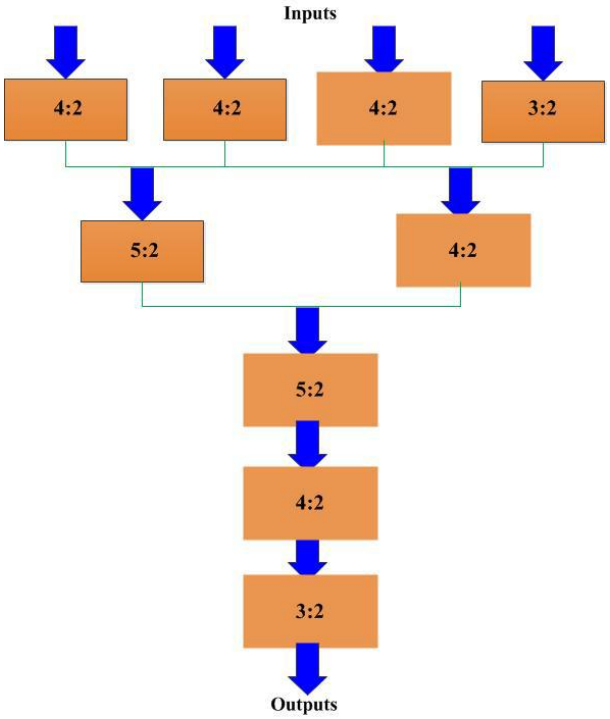
2.3 Description of Compressors

The partial product reduction stage determines the speed and power of the entire modulo 2^n+1 multiplier [1]. So, a group of low-power high-speed compressors must be designed in this stage. Traditional compressors are designed with full adders, but these designs occupy too much chip area and consume more power .To overcome this, now compressors are designed with MUX and XOR-XNOR sub circuits. The new MUX based compressor architecture meets the requirement of low power and high speed since it use less number of transistors compared to the full adder based compressors.

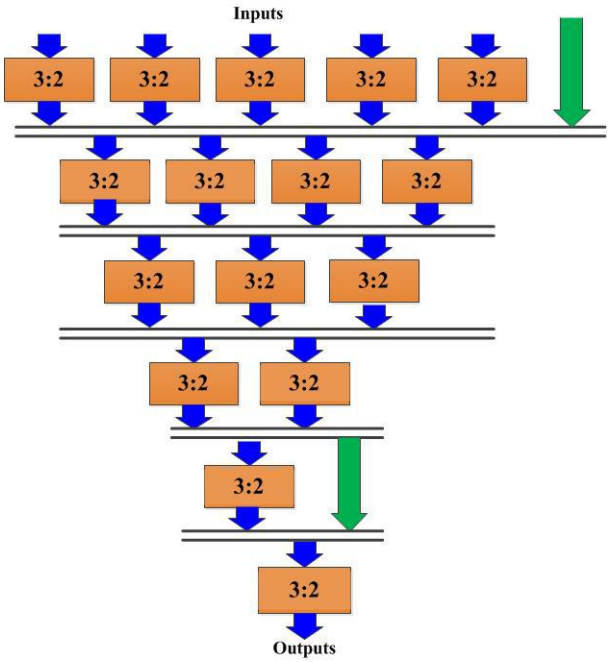
For a 16-bit compressor, different architectures can be designed, more possible architectures are discussed, as shown in Fig. 6(a), 6(b), 6(c) respectively.



6(a)



6(b)



6(c)

Figure 6: Possible compressor architectures for a 16-bit modulo 2^n+1 multiplier

Among these architectures, 6(c) is the best choices, because of it's less delay, consumes less power, and occupies small silicon area compared to remaining architectures. So this architecture 6(c) is widely used in low-power high-speed applications.

2.4 Algorithm for the Final Addition Stage

The final adder stage is a sparse tree based inverted End-Around-Carry (EAC) adder which is revised from conventional Kogge-Stone adder. Three major operations performed in final addition stage are:

a) Computation of carry-generate bits G_i ,the carry-propagate bits P_i ,

b) Carry computations,

c) 4-bit conditional sum generator.

The carry-generate bits G_i, the carry-propagate bits P_i are computed from the final sum vector and carry vector of partial product reduction stage, for every $i, 0 \le i \le n-1$ according to:

$$G_i = A_i B_i \qquad P_i = A_i + B_i \qquad \text{..... (5)}$$

Carry computation is performed using parallel prefix operator, which associates pair of carry generate and carry propagate signals and was defined in [7] as:

$$(G,P)\,o(G',P') = (G + P.G', P.P') \qquad \text{..... (6)}$$

$$(G_{k:j}, P_{k:j}) = (G_k, P_k) \circ (G_{k-1}, P_{k-1}) \circ .... \circ (G_j, P_j) \qquad \text{..... (7)}$$

Here ($G_{k:j,}$ $P_{k:j)}$ denotes generate and propagate term with k>j. Since for every carry, $C_i=G_{i:0}$, several algorithms have been introduced for computing all the carries using only 'o' operators. Among all algorithms, sparse tree based inverted end around adder is preferred because the area of the design and the delay of the final adder stage is significantly reduced, while the wire interconnection problem is also solved. The design of sparse tree adder relies on the use of a sparse parallel-prefix carry computation unit and conditional sum generator blocks. The sparsity of Inverted End Around adder refers to the number of carry-out bits generated by the adder. For example, sparsity of 4 means generating carry-out for every fourth bit. Increase in sparsity increases the speed of the sparse-tree adder.

Let us consider a 16-bit sparse tree inverted EAC adder, and a sparsity of 4, the carry out equations are computed as follow [2]:

$$C^*_{-1} = \overline{(G_{15}, P_{15})} \qquad \text{..... (8)}$$

$$C^*_3 = (G_3, P_3) \circ \overline{(G_{15:4}, P_{15:4})} \qquad \text{..... (9)}$$

$$C^*_7 = (G_7, P_7) \circ \overline{(G_{15:8}, P_{15:8})} \qquad \text{..... (10)}$$

$$C^*_{11} = (G_{11}, P_{11}) \circ \overline{(G_{15:13}, P_{15:12})} \qquad \text{..... (11)}$$

The 4-bit conditional sum generator computes two sets of sum bits corresponding with the possible values of the incoming carry. When the carry is computed, the correct sum is selected without any delay overhead.

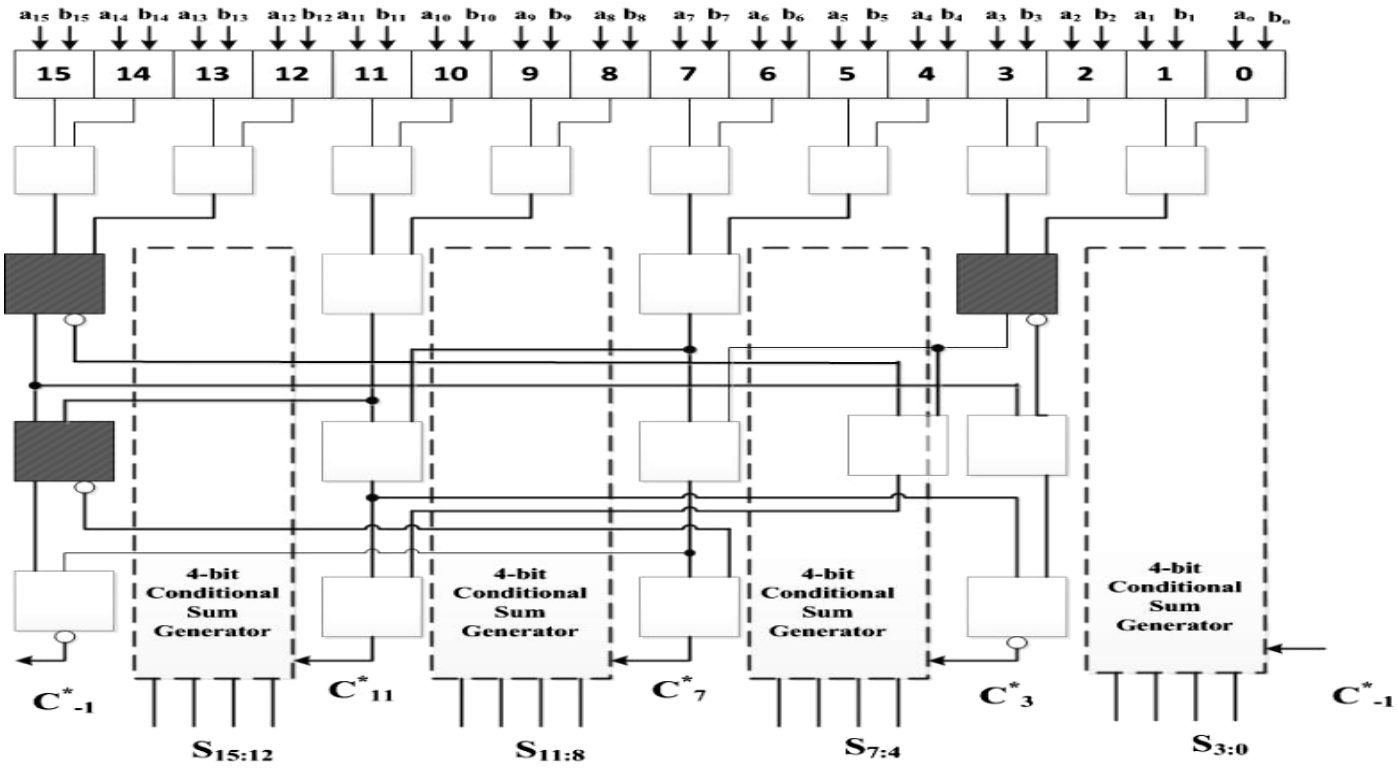


Figure 7: Sparse tree inverted EAC adder with sparsity of 4

3. SELF-CHECKING MODULO 2^n+1 MULTIPLIER DESIGN

Error detection and correction in binary multiplier makes use of residue codes required for computing the residue modulo A of a binary number. So, a self-checking circuit which extracts the residue of a binary input number of arbitrary widths, with respect to any odd modulus A is designed. A self-checking circuit should satisfy 3 properties to detect errors online. The properties are:

- **Fault secure**: For any fault in the fault set, if the circuit do not generate an incorrect code word for any input code, then the circuit is called fault secure circuit.
- **Self-testing**: For all faults in the fault set, at least one input code word generates an output which is not a valid code word.
- **Self-checking**: If the circuit satisfies self testing and fault-secure properties, then it is called self-checking circuit.

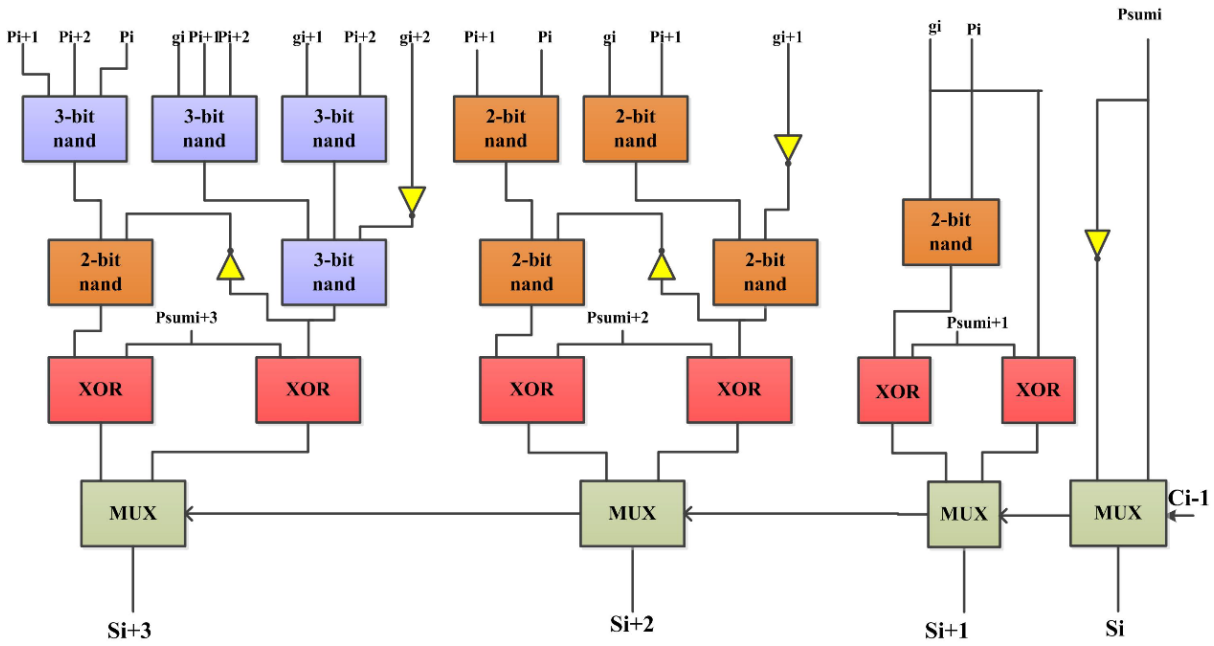


Figure 8: 4-bit conditional sum generator

The block diagram of the self-checking modulo 2^n+1 multiplier based on residue arithmetic codes consist of modulo 2^n+1 multiplier, modulo generators and dual rail checker. Modulo generator calculates residue of input operands with the check bases as 2^c+1 or 2^c-1. The dual rail checker compares the outputs of without self-checking modulo multipliers and self-checking modulo multiplier (compute residues for input operands).

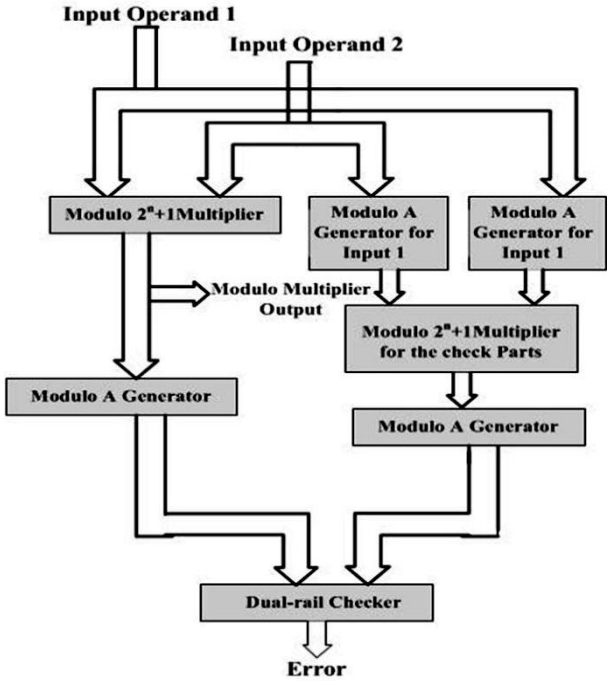


Figure 9: Self-checking modulo 2^n+1 multiplier

3.1 Modulo a Generator

Modulo A Generator are widely used in self-checking digital circuits to detect errors immediately as soon as they occur. In modulo a generator based on residue codes, the input operands are divided into n-bit vectors which then added by a sparse tree inverted end around carry adder. Let us consider the input operand n is a 16-bit binary number, and check base as'4', now the input operand is divided into 4 bytes based on

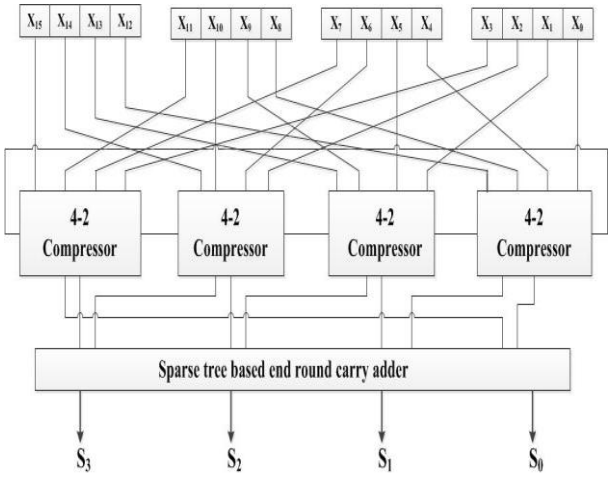


Figure 10: Modulo generator with check base 2^4-1

the check base and compressed into sum and carry vector by using 4:2 compressors and finally added using sparse tree inverted end around carry adder. For various values of input operands, different form of check bases (2^c+1or 2^c-1) are selected.

3.2 Dual Rail Checker

Dual rail checker in self-checking circuits plays a major role, since it detects all the faults occurred in the circuit. Dual rail checker is used to check dual blocks and duplicate blocks by inverting one of the outputs. It consists of four coded input vector of various length and two outputs. One output shows correct operation, and another shows the presence of error.

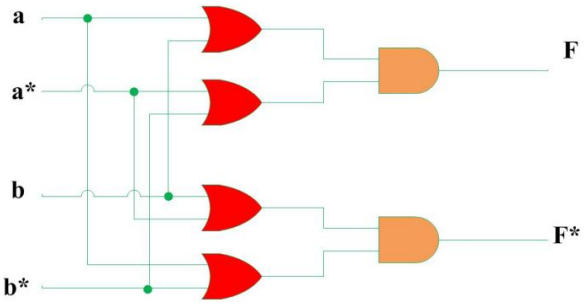


Figure 11: Dual rail checker

4. EXPERIMENTAL RESULTS

The simulation results of an area and power efficient self-checking modulo 2^n+1 multiplier are carried out using Xilinx ISE and synthesized using cadence RTL encounter tool .The comparison of area and power among 3:2, 4:2 and 5:2 compressors are shown in Table 1:

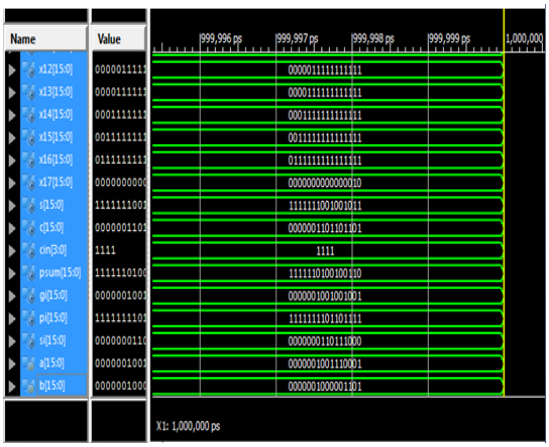


Figure 12: Simulation result of modulo 2^{16}+1 multiplier

From Table 1, we observe that the 3:2 compressor occupies less area and consumes less power compared to 4:2 and 5:2 compressors.

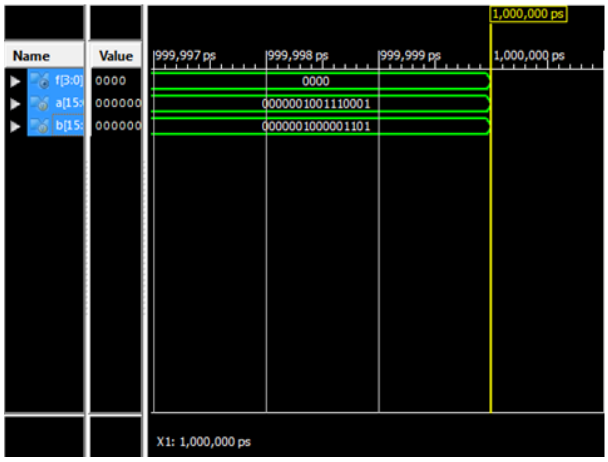


Figure 13: Simulation result of self-checking modulo 2^{16}+1 multiplier

Table 1: Area and Power comparisons among 3:2, 4:2 and 5:2 compressors

Parameters	3:2	4:2	5:2
Number of cells	4	14	17
Area(nm)	128.04	360.85	442.34
Leakage power (nW)	1.59	5.12	7.60

5. CONCLUSION

In this paper, a new design of an area and power efficient self-checking modulo 2^n+1 multiplier based on residue codes are proposed to detect errors online. To reduce area and power, MUX-based compressors are used instead of full adders to implement modulo multipliers and modulo A generators. Sparse-tree based inverted end-around-carry adder is used instead of Kogge-stone adder, to reduce the critical path of circuit. The proposed self checking modulo 2^n+1 multiplier based on residue codes are simulated using Xilinx-ISE and synthesized using cadence RTL encounter tool.

6. REFERENCES

[1] Wonhak Hong, Rajashekhar Modugu, and Minsu Choi, "Efficient Online Self-Checking Modulo2^n +1 Multiplier Design", IEEE Transactions On Computers, Vol. 60, No. 9, September 2011

[2] Modugu, R., Kim, Y.B., and Choi, M., "A fast low-power modulo 2^n+1 multiplier", Proc. 2009 IEEE Int'l Instrumentation and Measurement Technology Conf., pp. 951-956, May 2009.

[3] Curiger, A., Bonnenberg, H., and Kaeslin, H., "Regular VLSI Architectures for Multiplication Modulo (2^n+ 1)", IEEE Journal of Solid-State Circuits, Vol. 26, No. 7, July 1991.

[4] Zimmerman, R., "Efficient VLSI implementation of modulo (2n ± 1) addition and multiplication" IEEE trans. Comput., Vol. 51, pp. 1389-1399, 2002.

[5] 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.

[6] 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.

[7] Yi-Jung Chen, Dyi-Rong Duh and Yunghsiang Sam Han, Improved Modulo (2^n + 1) Multiplier For IDEA.

[8] Mathew, S., Anders, M., Krishnamurthy, R.K. and Borkar, S., "A 4-GHz 130-nm address generation unit with 32-bit sparse-tree adder core" In IEEE Journal of Solid-State Circuits, Vol.38, No. 5, pp. 689-695, May 2003.

[9] 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.

[10] 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.

[11] B. Parhami, computer arithmetic: algorithms and hardware designs oxford univ. press, 2000

[12] Samir Palnitkar, verilog hdl a guide to digital design and synthesis, SunSoft Press 1996.